UNIVERSITY OF KWA-ZULU NATAL

VARIABLE SENSOR SYSTEM FOR GUIDANCE AND NAVIGATION OF AGVs

STUDENT:
Ardhisha Pancham – 202513794
BSc Eng
School of Mechanical Engineering
Durban
202513794@ukzn.ac.za

SUPERVISOR:
Prof. Glen Bright
brightg@ukzn.ac.za

July 2008

Submitted in fulfilment of the academic requirements for the degree of Master of Science in Engineering at the School of Mechanical Engineering, University of KwaZulu Natal.

Preface

The author hereby states that this entire dissertation, unless specifically stated otherwise, is her own work, and has not been submitted in part or whole to any other university. This dissertation records the work completed by the author at the School of Mechanical Engineering, University of KwaZulu Natal from January 2006 to July 2008.

A. Pancham

Abstract

Automated Guided Vehicles (AGVs) depend on sensor systems to guide and navigate in manufacturing environments. The sensor system must ensure the correct path is followed and monitor the vehicle's progress as it proceeds through the environment to complete assigned tasks. An AGV therefore needs an efficient guidance and navigation system, based on sensor technology that allows it to navigate safely and accurately.

A variable sensor system will provide an AGV with the ability to interchange guidance and navigation techniques as the parameters in the environment change in order to complete tasks. This will maintain material handling operations and guarantee production rates. The standardization of sensor based guidance and navigation systems for AGVs and other autonomous robots will optimize performance.

The project involved research, design, construction, assembly and testing of a variable sensor system for guidance and navigation of AGVs. Sensor technology facilitated AGV operation in changing environments and avoidance of other robots and automated machines in an Agile manufacturing environment. Routing algorithms and procedures enabled safe movement and AGV task completion. The project required mechanical, electronic and software integration for AGV navigation and guidance.

Acknowledgements

I would like to thank:

1. The Supreme Intelligence that governs the Universe.

2. Prof Bright for giving me this opportunity and his encouragement and dedicated supervision.

3. My family for their patience, support and motivation.

4. The National Research Foundation (NRF) and the University of KwaZulu Natal for the financial support received.

5. Mr. Greg Loubser for the assistance with the electronic hardware.

6. Mr A Walker for the assistance with setting up the comm. port and programming advice.

7. Mr S Davraj for advice and support throughout the project.

8. Mr. N Rabikrisson and J. Padayachee for repainting the floor and assistance with some of the drawings.

9. Mr K. Smith, Mr. D. Naransamy and the staff of the mechanical engineering workshop for assistance in the construction and assembly of the mechanical components.

List of Acronyms

AGV	Automated guided vehicles
AGVS	Automated Guided Vehicle System
AMS	Agile Manufacturing System
API	Application Programming Interface
AS/RS	Automated Storage and Retrieval System
BT	Blue Tooth
CCD	Charge-Coupled Device
CdS	Cadmium Sulfide
CIM	Computer-Integrated Manufacturing
CNC	Computer Numerical Control
DC	Direct Current
DGPS	Differential Global Positioning System
EKF	Extended Kalman Filter
E-stop	Emergency stop
FMS	Flexible Manufacturing System
GPS	Global Positioning System
I2C	Inter-Intergrated circuits
INS	Inertial Navigation System
IR	Infrared
LDR	Light dependent resistor
LED	Light Emitting Diode
O/S	Operating system
PC	Personal computer
PCB	Printed circuit board
PCI	Peripheral Component Interconnect
PCI-E	Peripheral Component Interconnect Express
PID	Proportional Integral Derivative
PNP	Type of Bipolar junction transistor
pose	position and orientation
PWM	Pulse Width Modulation
PXI	PCI eXtensions for instrumentation
R-B	Rao-Blackwellised

RM	Reconfigurable Machine
SA	Selective Availability
SCL	Serial CLock line
SDA	Serial DAta line
SLA	Sealed Lead Acid
SLAM	Simultaneous Localization and Mapping
TCP / IP	Transmission Control Protocol (TCP) and the Internet Protocol (IP),
TTL	Transistor–Transistor Logic
USB	Universal serial bus
VB	Visual Basic
VSLAM	Visual Simultaneous Localization and Mapping

Contents

Preface		**ii**
Abstract		**iii**
Acknowledgements		**iv**
List of Acronyms		**v**
Contents		**vii**
List of Figures		**xvi**
List of Tables		**xxiii**
1	**Introduction**	**1**
1.1	Objectives	2
1.2	Mechatronics	3
1.3	Automated Guided Vehicles (AGVs)	3
1.4	AGV guidance and routing criteria	4
1.4.1	Guidance and routing	4
1.4.2	Traffic control and safety	6
1.4.3	System management	7
1.5	Variable sensor system criteria	9
1.6	Research Publications	10
1.7	Dissertation Outline	11
1.8	Summary	12
2	**Automated Guided Vehicles**	**13**
2.1	AGV Applications	13
2.2	AGV Types	13
	a. Unit Load	14
	b. Tuggers	14
	c. Fork	15
	d. Pallet trucks	15
2.3	Materials handling platform	15
	a. Standard deck	15
	b. Lift deck	16

	c. Conveyor deck	17
	d. Combination Lift / Conveyor deck	17
2.4	Safety Mechanisms	18
2.4.1	Types of Bumpers	18
	a. Front bumper	18
	b. Rear bumper	18
	c. Fork tip bumper	19
	d. Soft bumper	19
	e. Laser scanners	19
2.4.2	Side protections	20
2.4.3	Other safety mechanisms	20
2.5	Power Requirements	20
	a. Automatic / Opportunity charging	20
	b. Full – Cycle charging	20
2.6	Summary	21
3	**Mechanical design**	**22**
3.1	AGV requirements	22
3.2	AGV meccanum wheels	23
3.3	Materials handling platform	25
3.4	Docking Stations	26
3.5	Charging station	28
3.5	Attachment of a bumper	30
3.7	Summary	32
4	**Guidance and Navigation Techniques**	**33**
4.1	Guidance and navigation techniques for AGVs and mobile robots	33
4.1.1	Dead reckoning	33
4.1.2	Inertial navigation Systems (INS)	34
4.1.3	Active beacons	34
	a. Trilateration	35
	b. Triangulation	35
4.1.4	Types of Beacons	35
	a. Radio beacons	35

b. Ultrasonic Beacons 37
c. Optical Beacons 37
4.1.5 Landmark Navigation 37
4.1.6 Natural Landmarks 37
4.1.7 Artificial Landmarks 38
4.1.8 Line following 38
4.1.9 Light following 38
4.1.10 Global Positioning System 39
4.1.11 Simultaneous Localization and Mapping (SLAM) 40
4.1.11.1 Probabilistic SLAM 41
4.1.11.2 Solutions to SLAM problem 43
a. EKF-SLAM 43
b. FastSLAM 45
4.2 Techniques used specifically for AGVs 48
4.2.1 Optical/chemical/metallic guidance 48
4.2.2 Wire guidance 49
4.2.3 Magnet-gyro guidance 51
4.2.4 Laser guidance 51
4.3 Conclusion of guidance and navigation techniques 52
4.4 Sensors 55
4.4.1 Line following and light following sensors 55
a. Photodiodes 55
b. Phototransistors 55
c. Light Dependent Resistors 56
4.4.2 Concept Selection 57
4.4.3 Conclusion of line following and light following sensors 58
4.5 Obstacle avoidance sensors 58
4.5.1 Infrared proximity sensors 58
a. Reflected IR strength 58
b. Modulated IR signal 58
c. Triangulation 59
4.5.2 Ultrasonic sensors 60
4.5.3 Limit switches 61
4.5.4 Conclusion of obstacle avoidance sensors 62

4.6	Summary	62
5	**Electronic Hardware**	**63**
5.1	Control hardware	63
5.1.1	Peripheral Component Interconnect (PCI)	63
5.1.2	PCI eXtensions for instrumentation (PXI)	64
5.1.3	USB MicroDAQ and Controller cards	65
5.1.4	Motion Control Module	66
5.1.5	Telemetry System	67
5.1.6	Conclusion of Brainstem controller	70
5.1.7	Conclusion of custom made Telemetry System	71
5.2	Telemetry System	71
5.2.1	Motor Boards	73
5.3	Modular boxes for electronic hardware	74
5.4	Summary	75
6	**Preliminary Software**	**76**
6.1	Setup	76
6.2	User interface for preliminary program	77
6.3	Line Following Technique	80
6.3.1	Hardware	80
6.3.2	Program	83
6.3.3	Motion of the AGV	88
	a. Motion from conveyor to lathe	88
	b. Motion from conveyor to RM	89
	c. Motion from lathe to conveyor	89
	d. Motion from lathe to RM	89
	e. Motion from RM to lathe	90
	f. Motion from RM to conveyor	91
6.4	Light Following Technique	92
6.4.1	Hardware	92
6.4.2	Program	93
6.4.3	Motion of the AGV	99
	a. Motion from conveyor to RM	99

b. Motion from RM to conveyor 100
c. Motion from lathe to RM 101
d. Motion from RM to lathe 101
6.5 Ultrasonic sensor integration 102
6.6 Limit Switch Integration 104
6.7 Summary 108

7 Preliminary Testing 110
7.1 Line Testing 110
7.1.1 Linear Path: Lathe to Conveyor 110
7.1.2 Non - Linear Path: Lathe to RM 111
7.1.3 Conclusion 115
7.2 Light Testing 115
7.2.1 Linear paths 115
a. Path 1: Lathe to Conveyor 115
b. Path 2: RM and conveyor 116
7.2.2 Non - Linear Path: Lathe to RM 117
7.2.3 Conclusion 121
7.3 Light Intensity Test 122
7.3.1 Light sensors 122
7.3.2 Analysis 123
7.3.3 Line Sensors 123
7.3.4 Analysis 124
7.3.5 Conclusion 124
7.4 High Noise Level Test 125
7.4.1 Conclusion 125
7.5 Summary 128

8 Software 129
8.1 User Interface program 129
8.2 Program progress and optimization 133
8.2.1 First Program: LineLight.c 133
8.2.2 Second Program: InterLineLight.c 135
8.2.3 Third Program: LineLightSim.c 136

8.2.4 Conclusion 138
8.3 Final Line Variation Program 138
8.4 Light variation program 139
8.5 New routes 142
a. First method 144
b. Second method 144
c. Third method 145
8.6 Line variation program routes 145
a. Lathe to conveyor 145
b. Conveyor to lathe 147
c. Conveyor to RM 147
d. RM to Conveyor 148
e. Lathe to RM 149
f. RM to lathe 150
8.7 Light variation program routes 151
a. Lathe to conveyor 152
b. Conveyor to lathe 153
c. Conveyor to RM 153
d. RM to Conveyor 154
e. Lathe to RM 154
f. RM to lathe 156
8.8 Summary 157

9 AGV Testing 158
9.1 High noise level environment 158
9.2 Unsatisfactory floor environment 159
9.2.1 Line Variation Technique Testing 160
a. Motion of the AGV RM to lathe utilising line following 161
b. Motion of the AGV RM to lathe utilising light following 161
9.2.2 Environment Setup 162
9.2.3 First test scenario 164
9.2.4 Second test scenario 166
9.2.5 Third test scenario 170
9.2.6 Fourth test scenario 173

9.3	Light intensity environment	177
9.3.1	Light variation technique testing	177
	a. High to low light intensity testing	177
	b. Zero Watt Testing	178
9.3.2	First test scenario	178
9.3.3	Second test scenario	183
9.3.4	Third test scenario	187
9.3.5	Fourth test scenario	192
9.4	Summary	195
10	**Robot Testing**	**196**
10.1	Line following technique	197
10.1.1	Mechanical	197
10.1.2	Electronic	197
10.1.3	Software	197
10.1.4	Testing	199
10.2	Light Following Technique	200
10.2.1	Mechanical	200
10.2.2	Electronic	201
10.2.3	Software	201
10.2.4	Testing	201
10.3	Line variation technique	203
10.3.1	First Scenario	203
10.3.2	Second Scenario	205
10.4	Light variation technique	206
10.4.1	First scenario	206
10.4.2	Second scenario	208
10.5	Discussion	209
10.6	Summary	210
11	**Quantitative Analysis of AGV systems**	**211**
11.1	Calculations	213
11.2	Conclusion	214
11.3	Summary	214

12 Discussion 216
12.1 Guidance and Routing 216
12.2 Testing 218
12.2.1 Constant environment 218
12.2.2 High noise level environment 218
12.2.3 Unsatisfactory floor environment 219
12.2.4 Light intensity environment 219
12.3 Mechanical Design 219
12.4 Electronic hardware and Software 220
12.5 Summary 221

13 Conclusion 223

14 References 225

Appendices

A. Mechanical drawings 230

B. Tables 252

C. Figures 261

D. Electronic hardware features 268

E. Programming (Please refer to CD) **273**

1. AGV Programming 273
1.1 Access routines 273
1.2 Interfaces 275
1.3 Line following Technique 290
1.4 Light following Technique 395
1.5 Line variation Technique 453

1.6	Light variation Technique	652
2.	Robot Programming	879
2.1	Interfaces	879
2.2	Line following Technique	889
2.3	Light following Technique	894
2.4	Line variation Technique	899
2.5	Light variation Technique	913
3.	Program progress and optimization	931
3.1	First Program: LineLight.c	931
3.2	Second Program: InterLineLight.c	939
3.3	Third Program: LineLightSim.c	943

List of Figures

Figure 1-1 Graphical representation of Mechatronics [60] 3

Figure 2-1 Unit load AGVs [46] 13
Figure 2-2 Tugger [43] 13
Figure 2-3 Compact Fork AGV [43] 13
Figure 2-4 Standard deck AGV [43] 16
Figure 2-5 Lift deck AGV [43] 16
Figure 2-6 Conveyor deck AGV [43] 16
Figure 2-7 Combination Lift / Conveyor deck AGV [43] 16
Figure 2-8 Loading and unloading operations for a unit load lift deck AGV. [1] 17
Figure 2-9 Forklift AGV with bumpers and E-stop button. [41] 19

Figure 3-1 AGV before modifications 22
Figure 3-2 AGV frame, motors and old meccanum wheels 23
Figure 3-3 AGV wheels with resultant vectors and directions [1] 24
Figure 3-4 New materials handling platform 25
Figure 3-5 Motor held by bracket and O-ring with pulley 26
Figure 3-6 AGV docked at lathe docking station. 27
Figure 3-7 Aerial view of AGV docked at docking station. 28
Figure 3-8 Charging station 29
Figure 3-9 AGV with contacts 29
Figure 3-10 Wiring on charging station and AGV 30
Figure 3-11 AGV with bumper attached 31
Figure 3-12 AGV with bumper attachment mechanism 31

Figure 4-1 Representation of triangulation [3] 36
Figure 4-2 Possible robot positions at the intersection of circles. [4] 36
Figure 4-3 The SLAM problem. [8] 41
Figure 4-4 Spring network analogy. [8] 43
Figure 4-5 A single realization of robot trajectory using the FastSLAM algorithm. [8] 46
Figure 4-6 Optical guidance setup. Photo cells detect reflected light of guide 48

path. [2]
Figure 4-7 Channel with three wires embedded. [45] 50
Figure 4-8 Antenna with the two coils that detect the electro-magnetic field around the embedded wire. [45] 50
Figure 4-9 An AGV using a rotating laser for guidance. [26] 52
Figure 4-10 Scales of Navigation [4] 52
Figure 4-11 Angles obtained with different distances [53] 59
Figure 4-12 (a) Polaroid sonar transducers (b) Hitechnic sonar sensors. [48] 60
Figure 4-13 Beam pattern [48] 61
Figure 4-14 (a) Lever-operated (b) Roller-operated (c) Cam operated limit switches [11] 62

Figure 5-1 PXI system [52] 64
Figure 5-2 Schematic of DAQ interfacing 65
Figure 5-3 Schematic of Module interfacing 67
Figure 5-4 Schematic of Telemetry System 68
Figure 5-5 RF telemetry system components 71
Figure 5-6 Motor controller board 72
Figure 5-7 RF telemetry system components 73
Figure 5-8 Schematic of wiring for connectors in box 75

Figure 6-1 Preliminary program layout 77
Figure 6-2 User interface 78
Figure 6-3 Sensor feedback 79
Figure 6-4 Interface after task completion 79
Figure 6-5 Directions of movement of the AGV. 80
Figure 6-6 Line sensor board 80
Figure 6-7 LDRs, f2, f 0, and f1 in sensor bracket tubes. 81
Figure 6-8 LDR bracket 81
Figure 6-9 Position of LDR and light brackets on AGV. 82
Figure 6-10 Light bracket with 20W light. 82
Figure 6-11 Light bracket with 20W light on and line sensor bracket showing area of light exposure. 83
Figure 6-12 Line sensor feedback in HyperTerminal 83

Figure 6-13 Position of sensors. 84
Figure 6-14 Line routes indicated by black line. 87
Figure 6-15 Motion of the AGV from the conveyor to lathe is indicated by green arrows. Motion of the AGV from the conveyor to RM is indicated by red arrows. The green and red arrow represents a common path. 88
Figure 6-16 Motion of the AGV from the lathe to conveyor is indicated by grey arrows. Motion of the AGV from the lathe to RM is indicated by red arrows. The grey and red arrow represents a common path. 90
Figure 6-17 Motion of the AGV from RM to lathe is indicated by green arrows. Motion of the AGV from RM to conveyor is indicated by grey arrows. The green and grey arrows represent a common path. 91
Figure 6-18 One of the wedges in which the light sensors were inserted. 92
Figure 6–19 AGV with wedges in which light sensors were inserted. 93
Figure 6-20 Light on docking station. 93
Figure 6-21 Light sensor feedback in HyperTerminal 94
Figure 6-22 Position of sensors used. The sensors are inserted in the wedges. 95
Figure 6-23 Light routes indicated by orange dotted lines. 99
Figure 6-24 Motion of the AGV from the conveyor to lathe is indicated by the green arrow. Motion of the AGV from the lathe to conveyor is indicated by the grey arrow. 100
Figure 6-25 Motion of the AGV from the RM to conveyor to is indicated by the grey arrow. Motion of the AGV from the conveyor to RM is indicated by the red arrow. 101
Figure 6-26 Motion of the AGV from RM to lathe is indicated by the green arrow. Motion of the AGV from lathe to RM is indicated by the red arrow. 102
Figure 6-27 Ultrasonic sensor board 103
Figure 6-28 Position of ultrasonic sensors 103
Figure 6-29 Ultrasonic sensor feedback in HyperTerminal 103
Figure 6-30 Limit switch board 105
Figure 6-31 HyperTerminal feedback 105
Figure 6-32 Switch 0 located behind the docking mechanism of the AGV 106
Figure 6-33 Switches 2-5 on materials handling platform 106
Figure 6-34 Switch 6 behind bumper 107

Figure 7-1 AGV docked at conveyor, pallet being loaded, docking station under conveyor. 111
Figure 7-2 Test 1: Actual path versus theoretical path 113
Figure 7-3 Test 2: Actual path versus theoretical path 113
Figure 7-4 Test 3: Actual path versus theoretical path 114
Figure 7-5 Average path versus theoretical path indicate on graph 114
Figure 7-6 AGV with resultant motion vector and light exposure to sensors 117
Figure 7-7 AGV in lateral direction at RM docking station 118
Figure 7-8 AGV in longitudinal direction at RM docking station 118
Figure 7-9 AGV following arc in lateral direction to RM 120
Figure 7-10 AGV in longitudinal direction to RM 120
Figure 7-11 Graphs traced of light paths from lathe to RM. 121
Figure 7-12 Average light path from lathe to RM. 122
Figure 7-13 Graph illustrating distance achieved from noise test and deviation of the experimental distance from the actual distance for individual machines. 126
Figure 7-14 Graph illustrating distance achieved from noise test and deviation of the experimental distance from the actual distance for combined machines. 127

Figure 8-1 Program layout 129
Figure 8-2 User interface 131
Figure 8-3 Sensor feedback 132
Figure 8-4 Interface after task completion 132
Figure 8-5 Flow chart illustrating limit switch and ultrasonic sensor integration 134
Figure 8-6 Flow chart illustrating logic flow in the first program 135
Figure 8-7 Flow chart illustrating logic flow in the second program 137
Figure 8-8 Flow chart illustrating logic flow in the third program for alternating from line following to light following 139
Figure 8-9 Flow chart illustrating logic flow in the third program for alternating from light following to line following 140
Figure 8-10 Flow chart illustrating logic flow for light following variation program 143
Figure 8-11 Fusion of line and light routes. 144

Figure 8-12 Variable sensor routes 146
Figure 8-13 Program layout for routine from the lathe to conveyor. 146
Figure 8-14 Program layout for routine from the conveyor to lathe. 147
Figure 8-15 Program layout for routine from the conveyor to RM. 148
Figure 8-16 Program layout for routine from the RM to conveyor. 148
Figure 8-17 Program layout for routine from the lathe to RM. 150
Figure 8-18 Program layout for routine from the RM to lathe. 151
Figure 8-19 Program layout for routine from the lathe to conveyor. 152
Figure 8-20 Program layout for routine from the conveyor to lathe. 153
Figure 8-21 Program layout for routine from the conveyor to RM. 153
Figure 8-22 Program layout for routine from the RM to conveyor. 154
Figure 8-23 Program layout for routine from the lathe to RM. 155
Figure 8-24 Program layout for routine from the RM to lathe. 156

Figure 9-1 Path from RM to lathe indicated by arrows. 160
Figure 9-2 AGV at start position 162
Figure 9-3 AGV at junction 163
Figure 9-4 AGV at junction 163
Figure 9-5 First test scenario. 164
Figure 9-6 Graph of Average Motion of the AGV versus Path. 167
Figure 9-7 Second test scenario. 167
Figure 9-8 Graph of Average Motion of the AGV versus Path. 170
Figure 9-9 Third test scenario. 171
Figure 9-10 Graph of Average Motion of the AGV versus Path. 173
Figure 9-11 Fourth test scenario. 174
Figure 9-12 Graph of Average Motion of the AGV versus Path. 177
Figure 9-13 Test scenario (constant line) 179
Figure 9-14 First test scenario 179
Figure 9-15 Graph of Average Motion of the AGV versus Path. 183
Figure 9-16 Second test scenario. 184
Figure 9-17 AGV docking in lateral direction at lathe. 185
Figure 9-18 AGV docking in longitudinal direction at lathe. 185
Figure 9-19 Graph of Average Motion of the AGV versus Path. 187
Figure 9-20 Third test scenario. 188

Figure 9-21 AGV in longitudinal position after turning 90 degrees. 189
Figure 9-22 AGV leaving line in reverse direction at uncertain point and steering toward light in side-left direction. AGV does not follow line closely. 191
Figure 9-23 Graph of Average Motion of the AGV versus Path. 192
Figure 9-24 Fourth test scenario. 193
Figure 9-25 Graph of Average Motion of the AGV versus Path. 194

Figure 10-1 Robot before modifications 196
Figure 10-2 Robot after modifications 198
Figure 10-3 Slot for light and tubes for line sensors 198
Figure 10-4 Robot program layout 199
Figure 10-5 Robot in light route from lathe to RM. Robot turns from forward direction to face light as left sensor receives more light. 202
Figure 10-6 Robot at start position for first scenario 203
Figure 10-7 Average motion of the robot on the path 204
Figure 10-8 Second scenario 205
Figure 10-9 Average motion of the robot on the path 206
Figure 10-10 First scenario 207
Figure 10-11 Average motion of the robot on the path 207
Figure 10-12 Second scenario 208
Figure 10-13 Average motion of the robot on the path 209

Figure C-1 Transformer and battery charger module 261
Figure C-2 The Cybot hobby robot uses light following [49] 261
Figure C-3 Sensor board box with cover and female connectors 261
Figure C-4 Motor board box 262
Figure C-5 Sensor board box 262
Figure C-6 Light sensor and limit switch with male connectors 262
Figure C-7 Motor with male connector. 263
Figure C-8 Line laid along average light path between RM and lathe 263
Figure C-9 Diagonal line path 264
Figure C-10 Interface illustrating transition from high to low light following. b21, b22, b31, b32 refer to the reverse light sensors. us1, us2, us3 refer to the reverse ultrasonic sensor reading. s6, s7, s8, s9 refer to the limit switch status 265

i.e. open (O) or closed (C). Sensor feedback continued until the relevant limit switch was compressed on docking or in the event of a collision.
Figure C-11 User prompted again to reposition docking station. 267

List of Tables

Table 4-1 Scales of navigation 53

Table 4-2 Comparison of guidance and navigation techniques 54

Table 4-3 Selection criteria table for sensors 57

Table 5-1 Selection criteria table for control hardware 69

Table 5-2 Concept scoring 70

Table 6-1 Readings in HyperTerminal and C 84

Table 6-2 Position of the twelve sensors 84

Table 6-3 Translation of readings and action for line following in the forward direction. 85

Table 6-4 Translation of readings and action for line following in the reverse direction. 85

Table 6-5 Translation of readings and action for line following in the side-right direction. 86

Table 6-6 Translation of readings and action for line following in the side-left direction. 86

Table 6-7 Directions taken in order to follow a particular path. 87

Table 6-8 Readings in HyperTerminal and C 94

Table 6-9 Direction of sensors 94

Table 6-10 Translation of readings and action for light following in the forward direction. 95

Table 6-11 Translation of readings and action for light following in the reverse direction. 96

Table 6-12 Translation of readings and action for light following in the side-right direction. 96

Table 6-13 Translation of readings and action for light following in the side-left direction. 97

Table 6-14 Translation of readings and action for light following from the RM to lathe. 97

Table 6-15 Translation of readings and action for light following from the 98

lathe to RM.
Table 6-16 Ultrasonic board location and address. 104
Table 6-17 Distance of object and Action taken 104
Table 6-18 Limit switch allocation for line following. 106
Table 6-19 Limit switch allocation light following 108

Table 7-1 Time to complete the path between the lathe and conveyor. 111
Table 7-2 Time to complete the path between the conveyor and RM. 115
Table 7-3 Time to complete the path between the lathe and conveyor. 116
Table 7-4 Time to complete the path between the RM and conveyor. 117
Table 7-5 Time to complete the path between the lathe and RM. 119
Table 7-6 Stopping distance from the light. 121
Table 7-7 Light intensity test results for light sensors 123
Table 7-8 Light intensity test results for line sensors 123
Table 7-9 Average value in metres, at each distance, at the different noise level for individual machines. 126
Table 7-10 Average value in metres, at each distance, at the different noise level for combined machines. 127

Table 8-1 Techniques suitable for environment 130
Table 8-2 Advantages and disadvantages of the three programs. 138
Table 8-3 Light intensity sensor readings for steering and program allocation. 141
Table 8-4 Light intensity sensor readings for changing direction and program allocation. 142

Table 9-1 AGV's new reaction to an object at a particular distance 159
Table 9-2 Technique used under parameter status for first scenario 166
Table 9-3 Technique used under parameter status for second scenario 169
Table 9-4 Offshoot results and average 169
Table 9-5 Technique used under parameter status for third scenario 172
Table 9-6 Offshoot of not detecting line, results and average 173
Table 9-7 Technique used under parameter status for first scenario 175
Table 9-8 Average stopping distance and undershoot from light in reverse direction with reference to front of AGV. 176

Table 9-9 Average stopping distance and overshoot from light in reverse direction. 176
Table 9-10 Technique used under parameter status for first scenario 181
Table 9-11 Average stopping distance and undershoot from light in reverse direction. 181
Table 9-12 Average stopping distance and overshoot from light in reverse direction. 182
Table 9-13 Technique used under parameter status for second scenario 186
Table 9-14 Technique used under parameter status for third scenario 190
Table 9-15 Technique used under parameter status for first scenario 194

Table 10-1 Time for robot to complete linear path 200
Table 10-2 Time for robot to complete non-linear path 200
Table 10-3 Time for robot to reach the light at conveyor. 201
Table 10-4 Time for robot to reach the light at RM. 202
Table 10-5 Technique used under parameter status for first scenario 204
Table 10-6 Offshoot results and average 204
Table 10-7 Technique used under parameter status for second scenario 206
Table 10-8 Technique used under parameter status for first scenario 208
Table 10-9 Technique used under parameter status for second scenario 209

Table B-1 X and y values of the line path (theoretical) and the actual paths followed between the conveyor and RM. 252
Table B-2 X and y values of the actual paths at intersections and parallel sections. 256
Table B-3 X and y values of the light paths followed from the lathe to RM. 258
Table B-4 Average x and y values of the light paths followed from the lathe to RM. (The average was calculated from table B-2. 259
Table B-5 Comparison of light sensor characteristics [10] 260

1 Introduction

Agile manufacturing systems are environments characterised by unpredictable change. Automated Guided Vehicles (AGVs) are required in agile manufacturing environments to complete material handling tasks. The accuracy with which these tasks are executed contributes to the production rate.

It is necessary for an AGV to have an efficient guidance and navigation system that will allow the AGV to complete its tasks irrespective of the changing conditions. The guidance and navigation system must ensure the correct path is followed and monitor the AGV's progress, as it proceeds through the environment, to complete the assigned tasks. The system must allow for safe interaction and cooperation with other AGVs, robots, machines and humans. A reliable guidance and navigation system guarantees task completion and maintains the production rate in an agile manufacturing environment.

Guidance and navigation systems depend on constant parameters in the environment for efficient operation. The variation of these parameters may deter task completion and hence the production rate if the system cannot decipher the new parameters, and act accordingly.

A line following system depends on consistent readings from the line sensors. These readings depend on the condition of the line the AGV is following and the floor on which the line is laid. If the line following environment were to change such that the line sensors could not gather dependable readings, for example, a worn out line or dirty floor, the AGV would not be able to follow the line and execute its tasks. This environment would be referred to as an unsatisfactory floor environment.

If the AGV had an alternate guidance and navigation technique that did not depend on the line following parameters, such as light following, the AGV could utilise this technique to complete its material handling tasks. Alternatively, if the AGV was assigned a task in a light following environment, and the light intensity proved inadequate for accurate light sensor readings, the AGV could utilise its line following technique. Such an environment would be referred to as a light intensity environment.

If the AGV was operating in an environment with high noise levels, it would require ultrasonic sensors that overcame the noise levels, thereby allowing the AGV to continue to avoid obstacles.

This environment would be referred to as a high noise level environment. By utilising an alternative or more robust guidance and navigation technique the AGV would be able to function safely and efficiently in the changing environments.

AGV navigation in uncertain and dynamic environments demands perception and adaptation capabilities. [7] A variable sensor system would implement multiple guidance and navigation techniques, which enable the AGV to alternate to a more appropriate technique when faced with a changing environment. The AGV would be able to 'adapt' from one situation to another to complete material handling tasks. This would eliminate drawbacks in manufacturing systems, accelerate material handling operations and increase production rates.

A generic variable sensor system would facilitate integration of the system onto any mobile robot. The standardization of guidance and navigation systems for material handling robots and AGVs will optimize performance. Routing algorithms and procedures will ensure safe movement and task completion.

1.1 Objectives

The objectives of this research project were to:

1. Use Mechatronic engineering principles to research, design, construct, assemble and test a variable sensor system for guidance and navigation of AGVs.

2. Research, design and implement routing algorithms and procedures for AGV movement and cooperation.

3. Design and implement software for the system to operate in changing environments.

4. Test the reliability and robustness of the variable sensor system in three different environments:

    a. High noise level environment
    b. Light intensity environment
    c. Unsatisfactory floor environment

5. Test the performance of the variable sensor system on another mobile robot in the same environment to demonstrate the sensor system's flexibility, modularity, and exchangeability.

1.2 Mechatronics

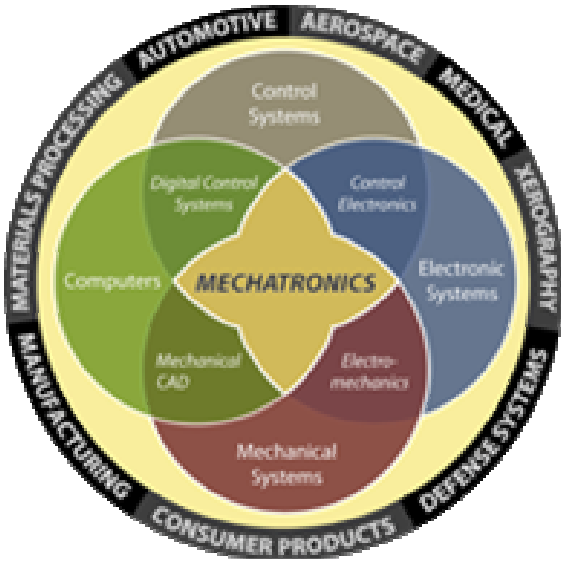


Figure 1-1 Graphical representation of Mechatronics [60]

Mechatronics is the fusion of mechanical, electronic, control and software engineering to further develop and enhance manufacturing operations and product design. [28] Figure 1-1 illustrates the components of Mechatronics. The project involved integrating these components in order to produce a variable sensor system for AGVs.

1.3 Automated Guided Vehicles

AGVs came into existence in the 1950's. AGVs are driverless vehicles that are designed to complete materials handling tasks in a number of different environments. They follow a guide path or are able to move around freely under some type of sensor system. They are controlled by computers or have on board microprocessors.

In Flexible Manufacturing System (FMS) environments AGVs are used to convey material between production areas and for the transport of assemblies. In hospitals AGVs transport beds, food and surgical supplies. In offices AGVs deliver mail and office supplies. In Japan AGVs are used to

serve food in restaurants. AGVs reduce labour costs and product and equipment damage. They improve safety and increase efficiency and productivity. [1, 43]

1.4 AGV guidance and routing criteria

There following criteria must be satisfied to operate an Automated Guided Vehicle System (AGVS) successfully:

1. Guidance and routing
2. Traffic control and safety
3. System management

1.4.1 Guidance and routing

Guidance system describes the method by which the AGVS guide paths are defined and the control systems that follow the guide paths. There are two main methods of defining guide paths along the floor: embedded guide wires and paint strips.

In wire guidance a wire is embedded in a channel cut in the floor. A relatively low current flowing in the wire forms a magnetic field. Two sensors (coils) are mounted on the AGV on either side of the guide wire to monitor the difference in the magnetic field. The difference is used to correct the direction of the AGV such that the field measured in both coils are equal. Wire guidance is common in warehouse and factory applications.

Paint strips are detected by an optical sensor system on board the AGV. The strips are taped, sprayed or painted on the floor. The difference in reflected light is used to direct the AGV such that it follows the path. Paint guidance is used in environments where electrical noise hinders the reliability of wire guidance or when guide wire installation in the floor surface is not suitable. [12, 41, 57]

A disadvantage of this method is that the paint strip must be kept in good condition, i.e. clean an unscratched. If the paint strip is neglected sensor readings of the floor and strip may become difficult to differentiate between. Without the difference in sensor readings the AGV would not be

able to follow the path leading to the required work station. If parts of the track are absent the AGV will not be able to continue with its tasks and this would hinder material handling operations.

This is where a variable sensor system would be beneficial. When valid sensor readings to follow the paint strip are not available or the strip itself is not present, the variable sensor system will allow the AGV to alternate to a more appropriate guidance technique. This will enable the AGV to continue with its assigned tasks and maintain material handling operations.

A safety precaution implemented in guidance systems is 'automatic stopping' should the AGV stray more than a few centimetres from the path, usually 5.1 cm to 15.2 cm. Automatic stopping prevents the AGV from running free in the environment. The AGV would then have to be positioned manually on the guide path. If the AGV is off the guide path, it is able to detect the path if moved within the same number of centimetres of it. This distance is referred to as the 'acquisition distance'.

If the AGV strayed from the line following guide path the variable sensor system would change from line following to light following. While light following the system would check for the presence of the line. If the line was acquired within the acquisition distance the variable sensor system would return to line following. This would be advantageous as the AGV would be able to continue with its material handling tasks unhindered, thereby maintaining production rates.

Routing is involved with the AGV choosing the correct path, among alternate paths, to reach its destination. A typical path layout, that utilises the capabilities of AGVs includes multiple loops, branches, side tracks and spurs with the required loading and unloading stations. [12, 41, 57]

When the AGV reaches a branching point, where the path divides into two or more directions, a decision is required as to which path must be followed. This is referred to as a 'decision point' for the AGV. Two methods are employed in commercial AGVs to allow the AGV to determine which path to follow:

1. Frequency select method
2. Path switch select method

The frequency select method implements different frequencies in the guide wires leading into separate paths at branches. On entering the decision point, the AGV reads an identification code on the floor to determine its location. The AGV selects the path with the frequency corresponding to its programmed destination. A separate frequency generator is required for each frequency used in the path layout. Two or three generators are usually needed in the system. Additional channels need to be cut in the floor for bypass channels, where only the main channel is powered for tracking.

The path switch select method implements the same frequency throughout the guide path layout. To control path selection at the decision point, the power is switched off in all branches except the one which the AGV has to use. The guide path layout is divided into blocks that can be operated by means of controls mounted on the floor close to the respective blocks. The control units are turned off and on by the AGVs as they move in the blocks. On entering a decision point, the AGV activates a floor mounted switching device attached to the control unit for the concerned block. The control unit activates the required path and deactivates the alternative branches. [12, 41, 57]

1.4.2 Traffic control and safety

Traffic control in an AGVS prevents collisions between vehicles travelling on the same path. A blocking system, whereby a vehicle travelling on a path is prevented from colliding with a vehicle ahead of it, is usually implemented. The following methods are employed in commercial AGV systems:

1. On-board vehicle sensing
2. Zone blocking

These methods are often combined to form a comprehensive blocking system. On-board sensing (also referred to as forward sensing) utilises a sensor system to detect the presence of vehicles and carts ahead on the same path. Commercial AGVs implement optical sensors and ultrasonic sensors. If the sensor detects an obstacle (e.g. another vehicle) in front of it, the AGV stops. If the obstacle is removed, the AGV continues on the path. In this way, traffic is controlled and collisions prevented. The effectiveness of this sensing method is restricted by the capability of the sensor system to detect obstacles ahead of it on the path. The sensors are most effective in detecting obstacles directly in front of the vehicle and are therefore more suitable for layouts with long straight path ways. The

sensors are less effective at turns and convergence points where obstacles may not be directly ahead of the sensor.

In zone control the AGVS layout is divided into separate zones. The operating rule is that no vehicle is allowed to enter a zone already occupied by another vehicle. The length of the zone accommodates one vehicle and an allowance for safety and other considerations. These considerations include the number of vehicles in the system, the size and complexity of the layout, and the aim of reducing the number of separate zone controls. When a given zone is occupied by one vehicle, a trailing vehicle is not allowed to enter that zone. The leading vehicle must leave the zone before the trailing vehicle can occupy it. By controlling the movement of vehicles in zones, collisions are avoided and traffic is controlled.

The safety of humans along the guide path must also be maintained. Obstacle-detection sensors in the form of ultrasonic, infrared and optical sensors are used to avoid collisions with humans. The same sensor used to detect vehicles in the blocking system is often used. The sensor is not restricted to one type of obstacle and can detect humans as well. Vehicles are programmed to stop or slow down if an obstacle is detected. Slowing down is used when the obstacle is on the side of the path or located ahead beyond a turn in the path. In this way, the AGV is able to proceed slowly along the path until it has passed the obstacle or turn.

A safety feature included on AGVs is the emergency bumper. The bumper surrounds the front of the AGV and protrudes in front of it 0.3m or more. When the bumper makes contact with an obstacle, the software instructs the AGV to brake immediately. The braking distance varies depending on the speed of the vehicle, the load and other factors. Manual restarting is usually required when the bumper makes contact with an obstacle. Different types of bumpers and other safety mechanisms are discussed in section 2.4.1. [12, 41, 57]

1.4.3 System management

System management is concerned with problem of dispatching vehicles to delivery and pick-up points in a timely and efficient way. Management is dependent on reliable guidance, routing and traffic control operations. Commercial AGV systems usually employ a combination of methods for dispatching vehicles responsively and effectively. The following methods are applied:

1. On-board control panel
2. Remote call stations
3. Central computer control

On-board control panels provide manual vehicle control, vehicle programming, and other functions. Most vehicles can be dispatched through the control panel to a station in the AGVS layout. This is the lowest level of sophistication among the possible dispatching methods. The advantage of on-board control panels is that they provide the AGVS with flexibility and responsiveness to varying demands in materials handling operations. The disadvantage is that manual attention is required to control the AGV.

Remote control call stations also allow the AGVS to respond to changing demands in the handling system. The simplest type of call station is a press button mounted close to the transfer station. A signal is transmitted to any passing vehicle to stop at the station to perform a load transfer. The on board control panel may be used to dispatch the vehicle to the new destination.

More advanced call stations comprise of control panels mounted close to the various stations along the layout. This method allows a vehicle to be stopped at a certain station and its next destination to be assigned from the remote call panel. This is a more automated approach to the dispatching function. It is applied where AGV systems can load and unload automatically. Both the call stations mentioned require a human interface at the transfer station.

Large factory and warehouse systems involve a high level of automation. The AGVS is required to be highly automated to accomplish efficient operation of the entire production-storage-handling system. Central computer control is used to achieve automatic dispatching of vehicles pertaining to a schedule of pickups and deliveries or in response to calls from the transfer stations in the layout. The central computer gives instructions to the vehicles with regard to their destinations and operations to execute. The central computer gathers real-time information of the vehicles in the system so it can determine which vehicles to send to which locations. The vehicles therefore continually communicate their whereabouts to the central computer. [12, 41, 57]

The operation of central computer dispatching differs. One difference is the distribution of the decision-making responsibilities between the central computer and the vehicles. In one distribution method the central computer is responsible for all decisions about vehicle routing and other

functions. The central computer assigns the routes for the vehicles and controls the guide path zone operation and other functions. In the other method each individual vehicle has decision making capabilities to make its own routing decisions and control its operations. The central computer is still required to control scheduling and to assign vehicles to tasks. The vehicles choose the route to be taken and control their transfer operations. They are referred to as 'smart' vehicles. [12, 41, 57]

1.5 Variable sensor system criteria

In order to satisfy the routing and guidance criteria of the AGV, the variable sensor system had to provide for the following:

Guidance

The sensor system must allow the AGV:

- To execute tasks in a changing environment, i.e. if the path is damaged, absent or lost.
- Automatic stopping should it stray from the path.
- Relocate the path within the acquisition distance if lost.

Routing

The sensor system must provide the AGV with ability to choose the correct path:

- At a complete decision point.
- And at an incomplete decision point.

Traffic control and safety

The sensor system must facilitate:

- On-board vehicle sensing for obstacle avoidance.
- Work in conjunction with mechanical safety features such as a bumper.

System management

The sensor system for the AGV must be operated by:

- Central computer control

Other criteria

The sensor system must be:

- Low-cost
- Robust
- Modular
- Flexible

By satisfying the above criteria the variable sensor system will optimise AGV performance thereby maintaining production rates in an Agile manufacturing system.

1.6 Research Publications

1. Proceedings of the International Conference on Competitive Manufacturing 2007, "Modular Mechatronic Navigation and Guidance Systems for Cooperation of Mobile Robots," G. Bright and A. Pancham.

2. Proceedings of the 23rd ISPE International Conference on CAD/CAM Robotics and Factories of the Future 2007, "Generic Navigation and Guidance System for Autonomous Mobile Robots," G. Bright and A Pancham.

3. Proceedings of the 2nd International Conference on Sensing Technology 2007*,* "Variable Sensor System for Guidance and Navigation of AGVs," A. Pancham, G. Bright and J. Potgeiter.

4. Proceedings of the 1st Robotics and Mechatronics Symposium 2007, "Generic Navigation and Guidance System for AGVs" (Abstract), Prof Glen Bright and A Pancham.

1.7 Dissertation Outline

Chapter 1, Introduction: Introduces the concept of a variable sensor system for guidance and navigation of AGVs. Presents the objectives and criteria to be achieved.

Chapter 2, Automated Guided Vehicles: Lists AGV applications and explains AGV types. Investigates materials handling platforms, safety mechanisms and charging methods.

Chapter 3, Mechanical Design: Describes design and implementation of the new materials handling platform, charging station, docking stations, bumper attachment and maintenance of meccanum wheels to enable AGV operation.

Chapter 4, Guidance and Navigation Techniques: Explains guidance and navigation techniques utilised for AGVs and mobile robots. Techniques are compared and a conclusion is made as to which techniques and sensors should be implemented for the AGV's variable sensor system.

Chapter 5, Electronic Hardware: Discusses and compares different control hardware. Draws a conclusion as to which hardware should be used for the variable sensor system. Describes the telemetry system and its components.

Chapter 6, Preliminary Software: Describes the line following and light following techniques applied. The implementation of ultrasonic sensors and limit switches is explained.

Chapter 7, Preliminary Testing: Assesses the reliability of the line following and light following techniques in the constant environment. Results are derived for light intensity and high noise level testing.

Chapter 8, Software: Explains the program structure. The line variation and light variation techniques are described. The AGV's routes are explained.

Chapter 9, AGV Testing: Describes testing of the AGV's variable sensor system in the high noise level, unsatisfactory floor, and light intensity environments to assess its reliability and robustness.

Chapter 10, Robot Testing: Describes testing of the variable sensor system on another robot, in the constant environment and the three changing environments, to demonstrate the system's flexibility and modularity.

Chapter 11, Quantitative Analysis of AGV systems: utilises equations to describe the operation of an AGV at constant velocity. The number of AGVs required to serve the Mechatronics CIM cell is calculated.

Chapter 12, Discussion: Explains how the variable sensor system criteria were satisfied. Problems encountered and solutions are discussed. Future work that can be done is considered.

Chapter 13, Conclusion: Concludes that the project objectives were achieved.

1.8 Summary

This chapter introduced the concept of a variable sensor system for guidance and navigation of AGVs. The objectives of the project were presented. The definition of Mechatronics pertaining to the project was explained. A brief introduction to AGVs was provided. The criteria of the variable sensor system were derived from the AGV guidance and routing, traffic control and safety, and system management criteria discussed.

2 Automated Guided Vehicles

2.1 AGV Applications

AGVs are utilised in the following industries: aerospace, apparel, automotive, beauty products, books and library systems, communications, dairy, electronic, food and beverage, healthcare, household items, mail order fulfillment, office and computer equipment, pharmaceuticals and health care, refrigerator and freezer applications, retail semiconductors, sporting goods, textiles, toys, warehouse and storage facilities. [57]

2.2 AGV Types

Figure 2-1 Unit load AGVs [46]

Figure 2-2 Tugger [43]

Figure 2-3 Compact Fork AGV [43]

a. Unit Load

Unit load AGVs are illustrated in figure 2-1. Unit load AGVs are referred to as "top carriers" because the load rests over most of the vehicle. They are designed to transport individual loads and are rated according to load capacity. Loads vary from pallets, drums, carts, racks, rolls, to custom containers. They have more complex drive/steer wheel combinations making them highly maneuverable and flexible for confined situations. They follow guide paths and are capable of random movement.

Lift and conveyor deck unit load AGVs interface with stands and conveyors for load transfer. When load transfer occurs between an AGV and conveyor "handshake" sensors verify the transfer of a load. Non-powered unit deck AGVs have the load transferred manually or by automatic equipment such as cranes, forklift trucks or other AGVs.

Unit load AGVs are suitable for medium rate, material flow situations with multiple stops. They are used in food and beverage, pharmaceutical, automobile industries hospitals and offices. In FMS and assembly environments the deck serves as a work platform for work-in-progress parts and assemblies. They can work in conjunction with CNC machines to deliver or pick up tooling fixtures, or parts. [43, 1]

b. Tuggers

A tugger is illustrated in figure 2-2. Tuggers are suitable for the transport of heavy material over moderate to long distances with a few transfer stops. They are the most productive AGV for tugging, as they transport more loads per trip than other AGV types.

They are used to transport material from shipping docks to warehouses to manufacturing environments. They are capable of pulling a series of trailers which can be loaded and unloaded with material automatically or manually. Forked AGVs can automatically load trailers.

There are two types of tuggers. Closed tow system tuggers consist of a tractor trailer unit. Trailers remain attached to the tugger. Open tow system tuggers can attach or detach trailers automatically or manually. Some tuggers do not operate in reverse. They therefore follow a loop or have turnaround loops at end points. [1, 43]

c. Fork

A fork AGV is illustrated in figure 2-3. Fork AGVs offer versatility and flexibility due to their ability to load and unload both at floor levels and at different heights. They are capable of stacking loads in racks. They transport pallets, racks, trays, rolls, and carts. They interface with stands, conveyors and racking.

Forks are powered by hydraulic or electric actuators to increase speed and accuracy. Forks can raise, lower, reach, tilt, and adjust width. They are well suited for applications where changes to the facility are expected. They can be easily adapted to accommodate new tasks. [43]

d. Pallet trucks

Pallet trucks are suitable for the transport of low material loads, over moderate to long distances with many transfer stops. They usually transport pallets and skids. They travel slowly due to the load carry method. Loads must have open access under the load platform for fork entrance. Pallet trucks are able to leave the guide path for load transfer and return. They can transport loads to and from floor level unlike unit load AGVs. This eliminates the need for stands. Both pallet trucks and tuggers require operators to monitor the operation to reduce delays. [1]

2.3 Materials handling platform

Unit load AGVs use various platforms for materials handling. These include standard, lift, conveyor, and combination lift/conveyor decks.

a. Standard deck

A standard deck AGV is illustrated in figure 2-4. Standard decks have no actuators to load or unload the load. They must be loaded and unloaded by external devices. Shuttle mechanisms are used for load transfer. [43]

Figure 2-4 Standard deck AGV [43]

Figure 2-5 Lift deck AGV [43]

Figure 2-6 Conveyor deck AGV [43]

Figure 2-7 Combination Lift / Conveyor deck AGV [43]

b. Lift deck

A lift deck AGV is illustrated in figure 2-5. Lift decks either lift or deposit a load to a station. A passive stand with no power requirements maybe used for load transfers. Figure 2-8 describes the operation of a lift deck AGV. The AGV moves under a stand, lifts the load off and moves to another stand to deposit the load. Lift Deck AGVs are well suited for handling of pallets and racks. [1, 43]

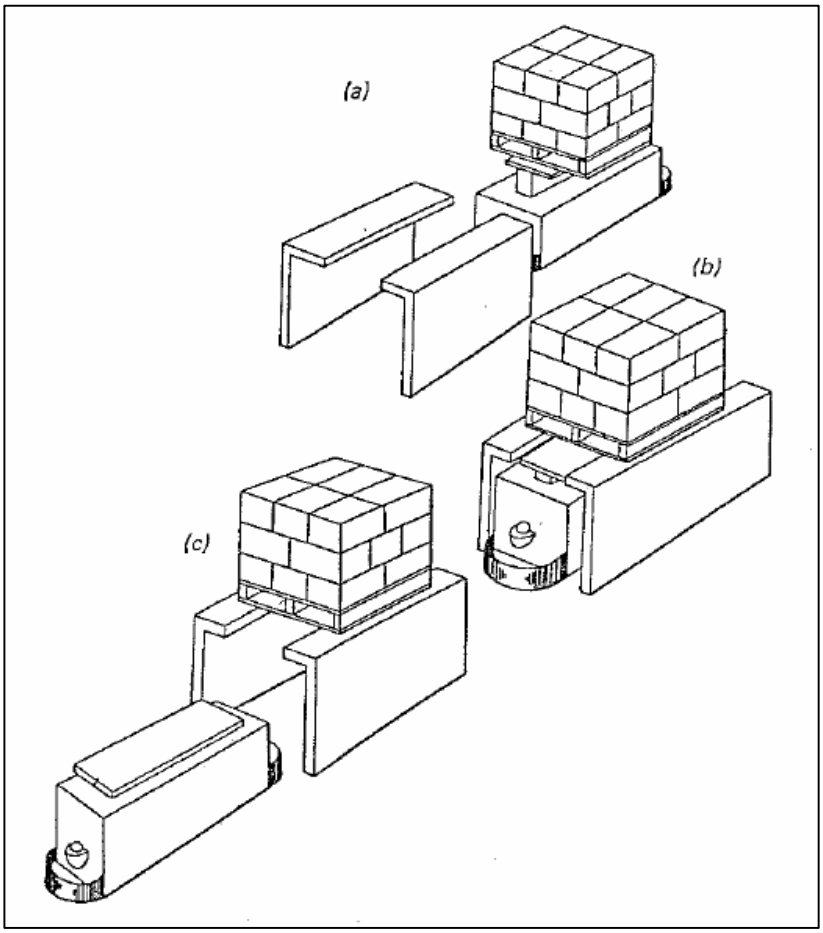


Figure 2-8 Loading and unloading operations for a unit load lift deck AGV. [1]

c. Conveyor deck

A conveyor deck AGV is illustrated in figure 2-6. Conveyor deck AGVs are well suited for handling of pallets and racks to and from conveyors. Conveyor decks consist of powered rollers to allow for smooth load transfer. AGVs align with powered conveyors for load transfers (known as slave drive stands). Load transfers may also occur between stationary conveyors. The AGV connects the conveyors without the physical requirements of a conveyance system. [43, 44]

d. Combination Lift / Conveyor deck

A combination lift / conveyor deck AGV is illustrated in figure 2-7. Combination Lift / conveyor decks are used when AGVs interface with conveyors of different heights.

2.4 Safety Mechanisms

AGVs are active vehicles that may encounter obstacles in their work paths. In order to avoid collisions and react correctly AGVs are equipped with mechanical safety devices and sensors.

2.4.1 Types of Bumpers

A bumper serves as a protective device for the AGV should it bump into a wall, machine, or person. It prevents damage to the AGV and the object or person it is obstructed by. When a collision occurs, sensors on or near the bumper, send signals to the controller to stop the AGV. The stroke of the bumper is designed to stop the vehicle before the AGV frame or any rigid part of the AGV makes contact with the object. Bumpers are usually mounted on the front and back of the AGV. This is a disadvantage to lateral moving AGVs as the sides are unprotected. [1, 41] Figure 2-9 illustrates a forklift AGV with different bumpers and an Emergency stop (E-stop) button. The different types of bumpers are described:

a. Front bumper

The front bumper acts as a safety device for the AGV in the main direction of travel. It provides protection for the AGV from the front and sides. The standard bumpers are usually made of polycarbonate plastic. Photocells maybe mounted on the AGV bumper to produce light beams across protected areas. A disadvantage is that the protection area is limited to the photocell beam height. Limit switches behind the bumper may also be used as sensors.

b. Rear bumper

The rear bumpers are located in the rear end of each support leg, and offer protection when the AGV is reversing. Optical sensors can be used to provide side protection and protection of the load area of an AGV.

c. Fork tip bumper

Forklift AGVs are equipped with fork bumpers in each fork tip to avoid colliding with loads or stationary objects if loads or lift heights are misaligned. This is a very useful safety device in automatic material handling.

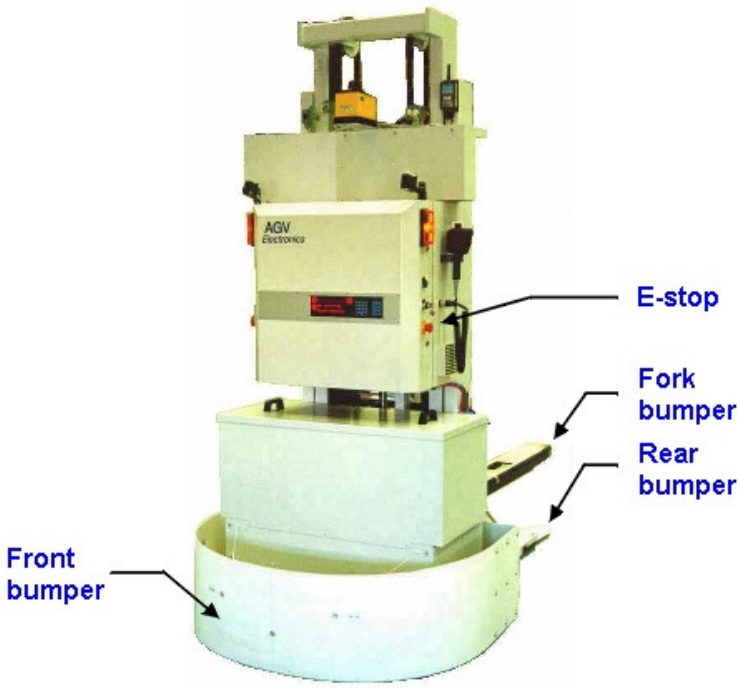


Figure 2-9 Forklift AGV with different bumpers and E-stop button. [41]

d. Soft bumper

Coated foam bumpers with sensors cover the whole width where they are mounted. A disadvantage is that the bumpers are easily damaged in industrial environments and are expensive to maintain and repair.

e. Laser scanners

Laser scanners may be used as stand-alone safety devices, or in conjunction with mechanical bumpers. The scanner can be programmed to shape protection zones for the AGV application. The laser scanner can be used to slow down the AGV's speed when objects are detected further away.

Disadvantages are that the protection area is limited to the laser beam height and that the scanner is expensive. Figure 2-2 illustrates an AGV with a laser bumper on the front. [41, 40]

2.4.2 Side protections

Longer vehicles require side protections as there is a large area of the AGV that sweeps in curves and a side protection device can protect this area. A side protection can consist of an optical beam that is broken by objects and will stop the AGV. Safety edges can be used to protect the AGV sides. They usually consist of a profile of rubber material with a sensor inside. The sensor maybe contact strip material or optical fiber. [41]

2.4.3 Other safety mechanisms

Proximity sensors are used to detect and avoid obstacles. Warning lights and hooters are used to warn humans. Emergency stop (E-stop) buttons are also available. [12]

2.5 Power Requirements

AGVs utilise batteries that range from 24 to 48V DC with amp – hour ratings determined by the system requirements. To facilitate the efficient operation of an AGV a method of charging its batteries is necessary. The following methods are available:

a. Automatic / Opportunity charging

Batteries are charged while the AGV is in operation or idle. No manual intervention is required. If opportunity charging is being used the AGV receives a charge whenever the opportunity arises. If batteries reach a predetermined level, the AGV completes its assigned task before it proceeds to a charging station. Collector brushes make contact with a floor charger bus plate and charging occurs.

b. Full – Cycle charging

The AGV is removed from service in order for batteries to charge. When batteries are nearly discharged, the AGV proceeds to a charging station. Batteries are charged automatically or

manually. In automatic charging, a docking mechanism such as a probe, plugs into a docking station connection.

In manual charging an operator removes the discharged battery from the AGV and replaces it with a fully charged battery. Batteries may also be swapped automatically by an automatic battery swap machine. The machine replaces the AGV's discharged batteries with charged batteries and no operator is required. [1, 39]

2.6 Summary

This chapter described AGVs. The industries in which AGVs perform material handling operations were listed. The various types of AGVs, i.e. unit load, tuggers, fork lift, pallet trucks and their applications were described. The different types of material handling platforms, i.e. standard deck, lift deck, conveyor deck and combination lift/conveyor deck were explained. Bumpers, side protections, laser scanners, and sensors are safety mechanisms employed to prevent damage to AGVs and humans. Automatic charging and full - cycle charging are methods used to charge batteries and maintain AGV power requirements.

3 Mechanical Design

3.1 AGV requirements

An AGV that was the result of previous projects served as the platform for the variable sensor system. The AGV is illustrated in figure 3-1. The AGV's function was to transport pallets between the conveyor, lathe and RM in the CIM cell environment. The AGV's frame was made of 38 x 38 x 1.6 mm square steel tubing (refer to figure 3-2). The drive system consisted of four meccanum wheels that allowed for omni - directional movement and zero-radius rotation. The wheels were driven independently by four 12V DC, windscreen wiper motors. The AGV was not operational. The AGV required:

a. A guidance and navigation system
b. Re-tubing of the meccanum wheel rollers
c. A new materials handling platform
d. Docking stations
e. A charging station
f. The attachment of an existing bumper

The guidance and navigation system is explained under section 5.2.

Figure 3-1 AGV before modifications. Old materials handling platform with threaded rods.

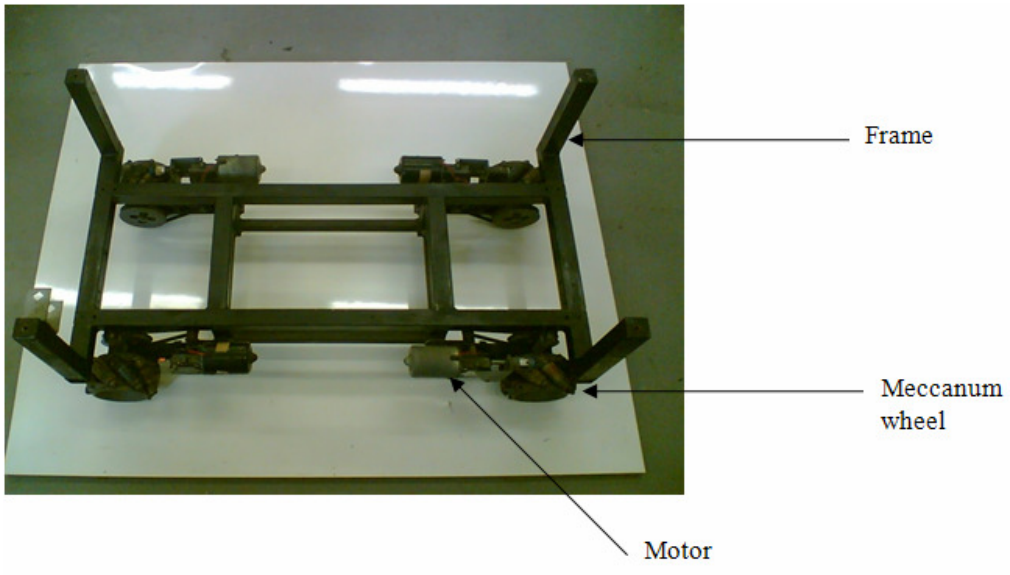


Figure 3-2 AGV frame, motors and old meccanum wheels.

3.2 AGV meccanum wheels

The AGV is unlike conventional vehicles that are limited to forward and reverse movement. It has the unique ability to move laterally. This is advantageous at decision points in manufacturing environments where branching routes are employed for routing purposes.

Lateral movement was made possible by the AGV's meccanum wheels. Each wheel consisted of a steel rim with eight wooden rollers. The rollers were positioned at 45 degrees to the rotation direction and rotated freely about their own longitudinal axis.

A small area of contact was made between a single roller and the floor when the wheel was stationary. As the wheel rotated, during forward or reverse motion, a particular roller left the floor and the contact area was picked up by the next roller and so on.

When the AGV moved laterally or rotated about its own axis, the forces from counter wheel rotation caused the contact area to rotate the particular roller. When the roller left the floor the area was picked up by the next roller and it started to rotate.

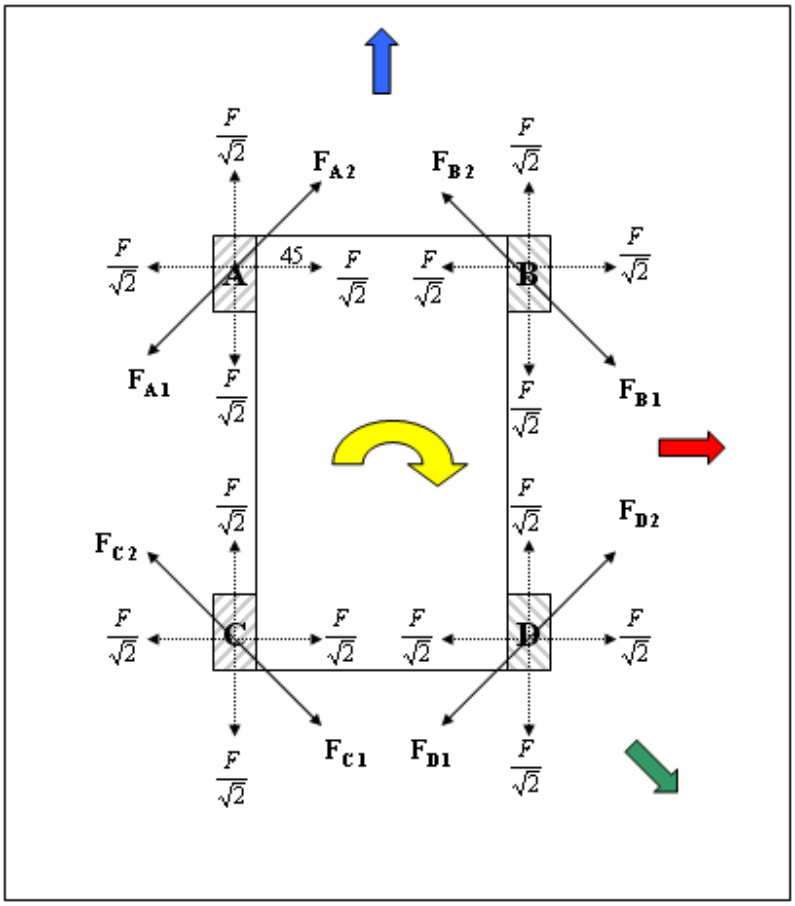


Figure 3-3 AGV wheels with resultant vectors and directions [1]

Direction control of the AGV was attributed to the resultant forces produced when moving. This is illustrated in figure 3-3. When wheel A rotated forward (indicated by blue arrow), two forces, Fsin45 and Fcos45 occurred at right angles to each other. Fsin45 and Fcos45 each yielded $\frac{F}{\sqrt{2}}$. The forces were parallel to the floor and the resultant was force $F_{A2.}$ Similarly, if wheel A rotated in the reverse direction, force F_{A1} resulted.

Movement of the AGV in a specific direction was achieved as follows:

a. Forward motion: All wheels rotated in the direction indicated by the blue arrow. The vector sum of F_{A2}, F_{B2}, F_{C2} and F_{D2} resulted in the forward motion.

b. Sideways motion: The wheels on the left rotated away from each other and wheels on the right rotated towards each other. The vector sum of F_{A2}, F_{B1}, F_{C1} and F_{D2} resulted in the sideways motion, indicated by the red arrow. To move sideways at an angle, the speed of wheels B and C were 'ratioed' with that of wheels A and D. Motion in the direction of the green arrow resulted if wheels B and C rotated faster than wheels A and D.

c. Rotation about axis: Wheels A and D rotated in the forward direction (blue arrow), and wheels B and C rotated in the reverse direction. The vector sum of F_{A2}, F_{B1}, F_{C2} and F_{D1} resulted in the turning motion, indicated by the yellow arrow.

The rollers on all four wheels were fitted with new bicycle tubing to increase the friction when contact was made. This prevented wheel slippage when moving. [1, 13]

3.3 Materials handling platform

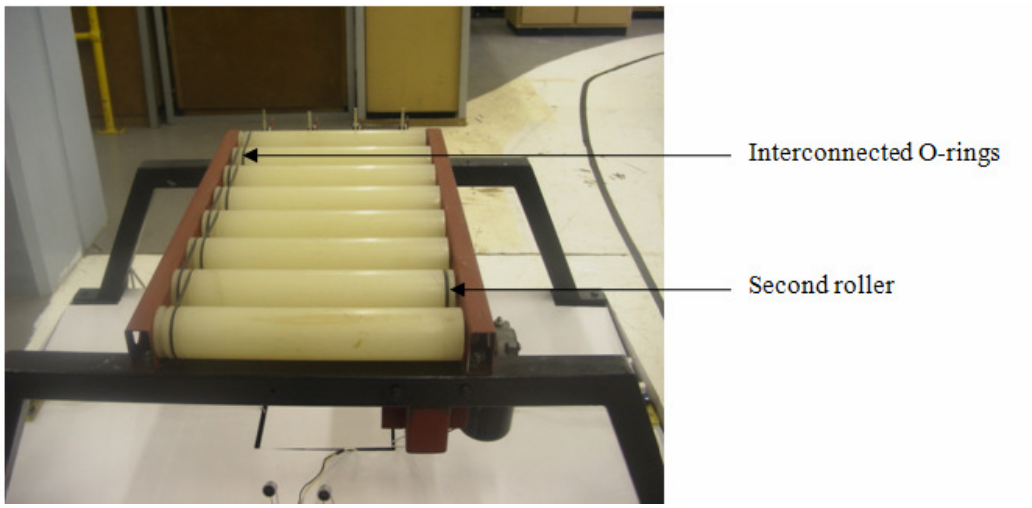


Figure 3-4 New materials handling platform

The old materials handling platform is illustrated in figure 3-1. It consisted of a lifting table supported by four threaded rods. The lifting table was driven by a 12V DC motor. It did not move up and down smoothly and was rather unstable. A new platform was needed for the purpose of materials handling.

In order to allow interfacing between the conveyor and AGV for automatic pallet transfer, a conveyor deck materials handling platform was designed for the AGV (refer to section 2.3.c). The

new platform is illustrated in figure 3-4. It consisted of two 50 x 25 x 1.6mm steel rectangular tubes that supported eight 50mm rollers.

Each roller was made of polypropylene and had a 5mm groove on one end. A steel rod was press fitted through the center of the roller. Bearings were located on both ends of the rod inside the roller for secure positioning. The threaded ends of the rods permitted the rollers to be fastened to the rectangular tubes by nuts.

The rollers were interconnected by rubber O-rings that fitted in the grooves. The second roller had grooves on both ends (refer to figure 3-4). This allowed one O-ring to be connected to a pulley. The pulley was grub screwed to the shaft of a 12V DC motor that drove the entire platform. The motor was held by a bracket that was mounted to the AGV frame (refer to figure 3-5). The simultaneous motion of the rollers allowed smooth part transfer between the conveyor and AGV.

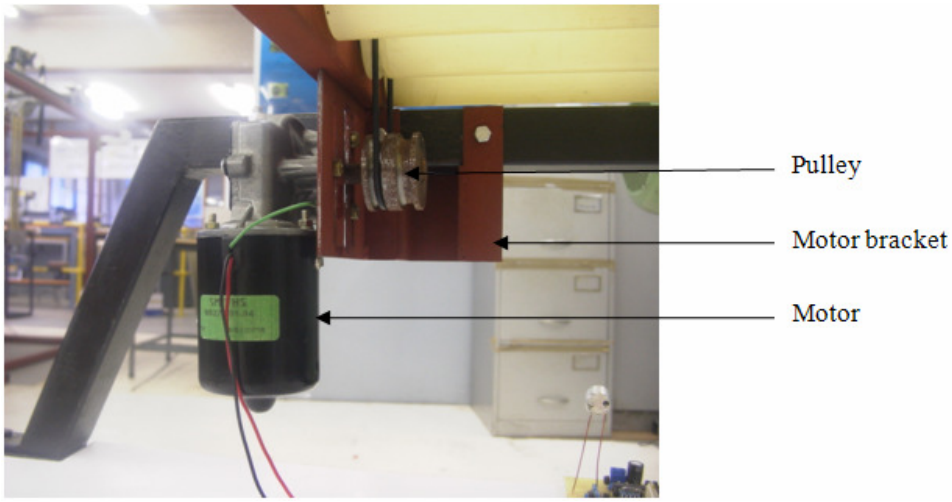


Figure 3-5 Motor held by bracket and O-ring with pulley

3.4 Docking Stations

Docking stations were designed for the AGV to allow interfacing with the conveyor, RM, and lathe. The docking stations allowed the AGV to load or unload pallets manually at the lathe and RM and automatically at the conveyor. Figure 3-6 illustrates the AGV docked at the lathe docking station. Figure 3-7 is an aerial view of the AGV docked at the docking station illustrating the docking mechanisms when springs were compressed.

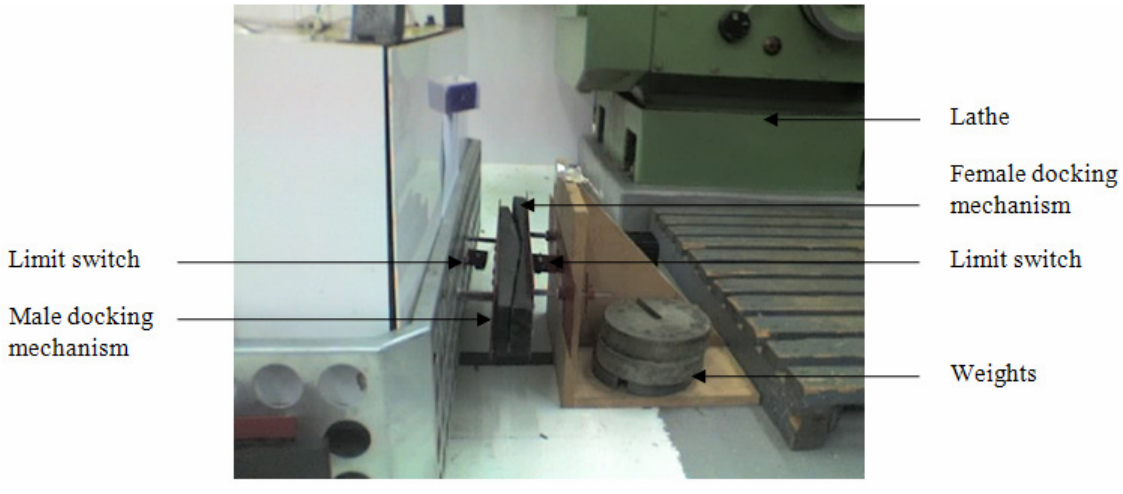


Figure 3-6 AGV docked at lathe docking station

The docking mechanisms consisted of male and female parts. Male docking mechanisms were located on all four sides of the AGV to allow it to dock in four directions. Female docking mechanisms were located on the docking stations.

The male parts were convex wooden shaped structures and the female parts were concave. Each male docking mechanism was screwed to a steel plate that had three rods welded on the back. The rods slotted into holes drilled in the AGV bumper, and were secured with circular bushes which were held by grub screws. Springs around the rods absorbed the impact when the AGV docked with the docking station.

Each docking station consisted of a wooden angle-bracket frame. A steel plate was screwed to the frame. The plate had two holes in it and two circular bushes, to guide the rods of the female docking mechanism. The female docking mechanism was screwed to a steel plate that had two rods welded on the back. The rods were inserted through the holes in the frame and secured by bushes.
When the male and female docking mechanisms mated, limit switches behind the docking mechanisms were compressed. The limit switch behind the female docking mechanism, on the docking station, notified the work station that the AGV had docked. The limit switch behind the male docking mechanism, on the AGV, signaled the AGV to stop movement and start the materials handling platform, to assist pallet loading at the conveyor.

The pallet was pushed onto the materials handling platform by a piston mechanism. The platform rotated backwards and drew the pallet in. The pallet compressed the limit switches at the end of the

platform. This indicated that the pallet was loaded and the platform stopped rotating. The AGV continued with its tasks. Manual transfer occurred at the lathe and RM.

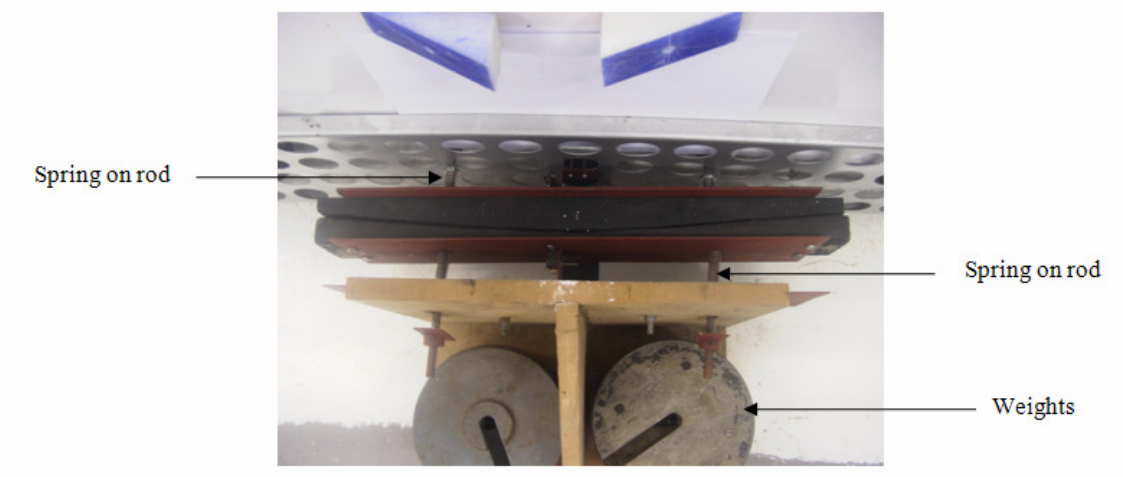


Figure 3-7 Aerial view of AGV docked at docking station with springs compressed.

3.5 Charging station

A charging station that correlated with the AGV's dimensions was constructed of wood. The charging station is illustrated in figure 3-8. Two brass contacts (one for positive and one for negative) were located on the charging station. The spacing between the contacts and the fact that the AGV was line following allowed for correct charging of the AGV.

The AGV had two hinged brass contacts attached to the bottom of the bumper (refer to figure 3-9). A limit switch was located directly behind each contact. When the limit switches were compressed, a signal was sent to stop the AGV for charging. As the electronic modules did not consume a large amount of current it was not necessary to disconnect them electronically with a relay while charging.

Careful attention was paid so that the brass contacts were not in contact with the steel bumper as this would cause a surge in current. Each contact was attached to a Perspex rectangular piece and grub screwed to the bumper. This prevented any metal contact between the brass contacts and the bumper.

A 220-16V AC transformer, connected to a 1A battery charger module, was located on the charging station, as illustrated in figure C-1, Appendix C. When contact was made the two parallel connected 12V batteries charged. The batteries took approximately 4-5 hours to charge to from 10V to 13V. Figure 3-10 illustrates the wiring on the charging station and AGV.

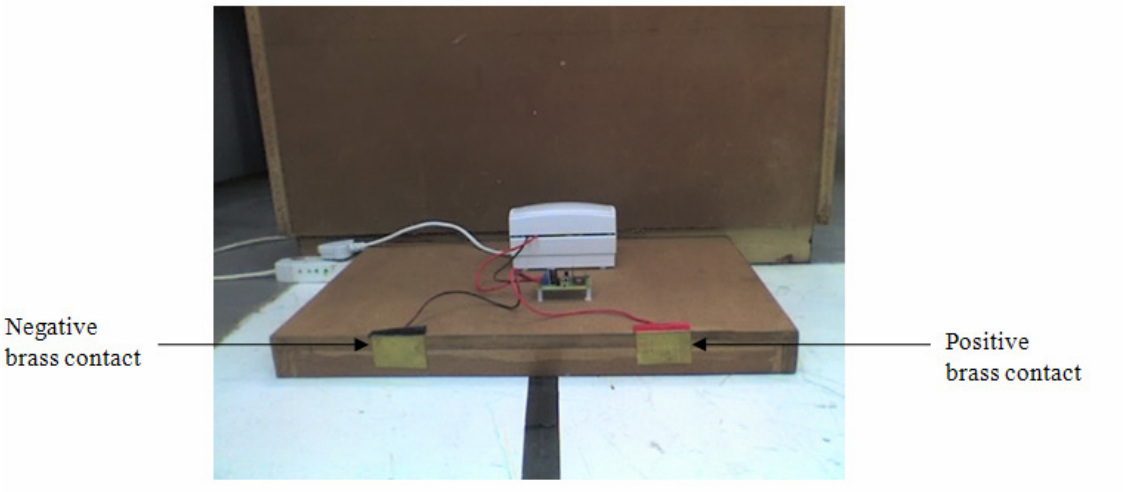


Figure 3-8 Charging station

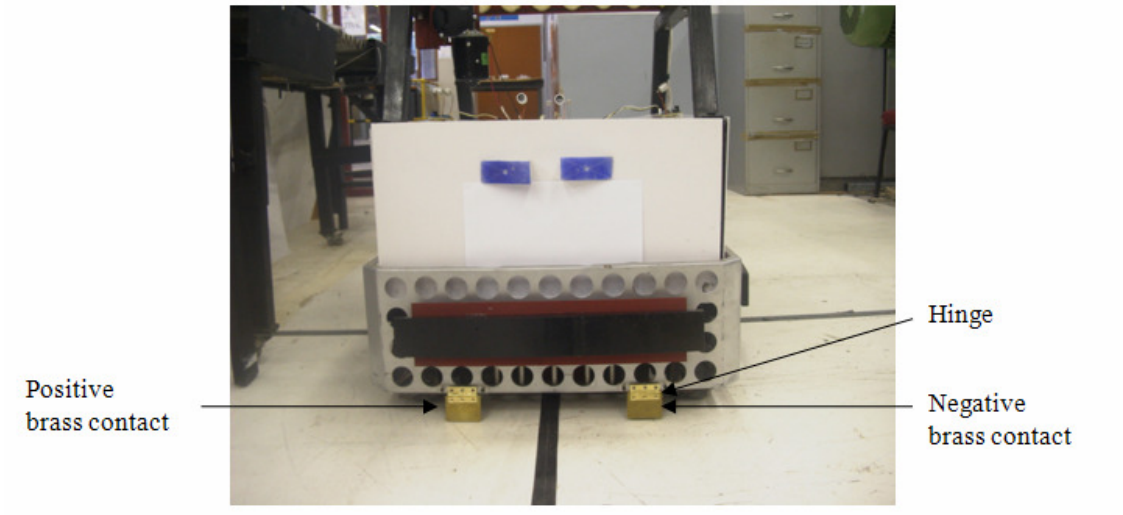


Figure 3-9 AGV with contacts

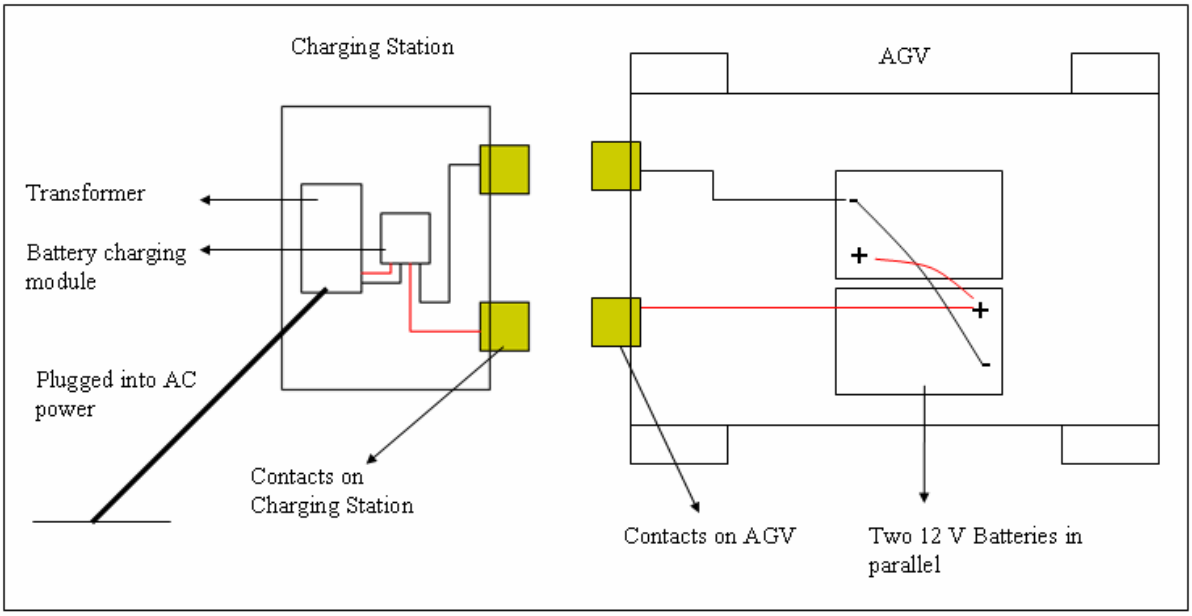


Figure 3-10 Wiring on charging station and AGV. Red indicates positive. Black indicates negative. The contacts have been exaggerated in this diagram.

Future work on the AGV charging station could include a power management module. Charging was only possible when line following as light following in a curve was unpredictable. There was a risk of the positive contact making contact with the negative one or vice versa when light following in a curve. Incorrect contact can be prevented by using magnets to guide the AGV to the correct charging position.

3.6 Attachment of a bumper

A bumper needed to be attached to the AGV. The bumper served as a safety mechanism in the event of a collision. The bumper was made of 1.6 mm thick steel and had circular cutouts that made it light weight (refer to figure 3-11). The bumper was an all round bumper that offered a stroke depending on the method in which it was attached, either longitudinally or laterally. The bumper required a means of attachment to the AGV frame.

Figure 3-12 illustrates one of the four mechanisms, designed to attach the bumper to the AGV's frame, in the longitudinal direction. Each mechanism consisted of a 5mm plate with a 10mm rod welded to it. The plate was attached to the inside of the bumper by bolts and nuts. The rod was slotted in the AGV frame and held by a circular bush. Springs allowed for the bumper to be pushed

back approximately 15mm. When the bumper was pushed back, limit switches were activated and a signal was sent to the control program to stop the AGV. Appendix A contains drawings of all mechanical designs.

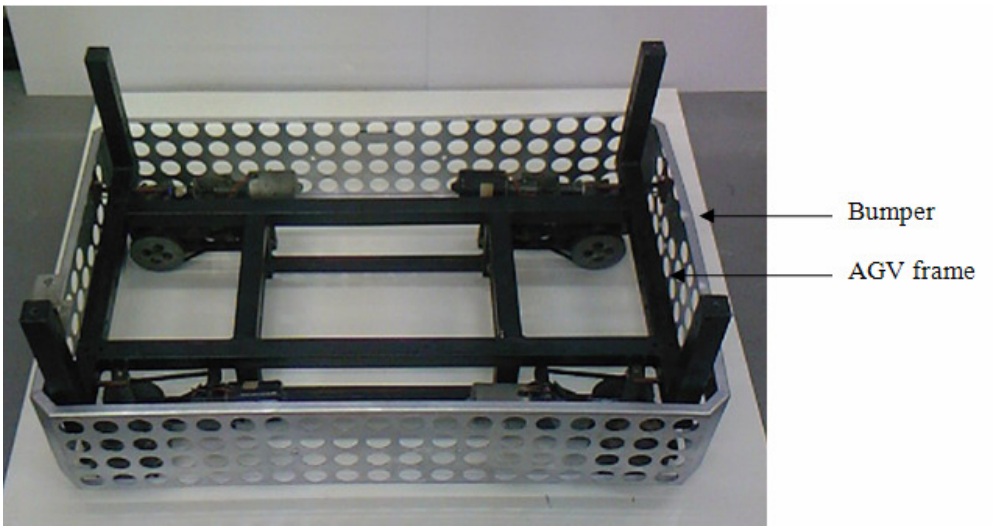


Figure 3-11 AGV with bumper attached

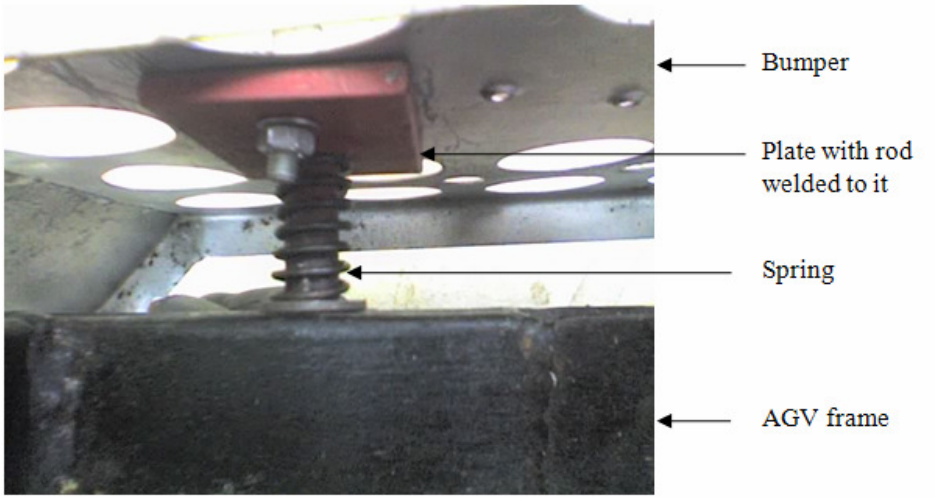


Figure 3-12 AGV with bumper attachment mechanism

3.7 Summary

This chapter described the mechanical design of the project. The requirements of the AGV for operation were determined. The function and maintenance of the meccanum wheels were explained. The design and operation of the new materials handling platform, docking stations, charging station and bumper attachment were discussed. The new materials handling platform comprised of a conveyor deck driven by a 12V DC motor. The docking station consisted of a wooden angle bracket with a female docking mechanism. The AGV's male docking mechanism mated with the female docking mechanism and activated the relevant limit switches for docking at the various work stations. The charging station had two brass contacts that the AGV's brass contacts made contact with and enabled the AGV's batteries to charge. The bumper was attached to the AGV with springs and allowed a stroke of 15mm in the event of a collision.

4 Guidance and Navigation Techniques

Guidance is defined as the technique whereby a vehicle is given directions and motion instructions, fulfilling a set of conditions, such that it reaches a specified location. Navigation can be described as the determination of the state of motion of the vehicle i.e. position and velocity. Navigation can be classified into global, local and personal navigation. [3]

Global navigation is the ability to determine position in absolute or map-referenced terms, and to advance to a destination point. Local navigation is the ability to determine position relative to objects in the environment and to cooperate with them correctly. Personal navigation is being aware of the positioning of the different parts that one consists of, in relation to each other and in handling objects.

The variable sensor system would facilitate guidance and navigation for mobile robots and AGVs that are working in changing environments. Mobile robot and AGV guidance and navigation techniques were investigated. The main difference between AGVs and mobile robots is that AGVs are usually restricted to a fixed path whereas mobile robots can move freely in the environment.

There are several techniques that allow an AGV and mobile robot to move from one location to another in a working environment. These techniques aid the vehicle to complete its task without collision. Criteria such as cost, reliability, accuracy, control, flexibility and robustness should be considered when implementing a guidance and navigation technique. [1, 4]

4.1 Guidance and navigation techniques for AGVs and mobile robots

4.1.1 Dead reckoning

Odometric sensors are used to monitor the rotation of each wheel and thus determine the relative position of the vehicle with reference to its starting point. The integration of incremental motion data over time leads to unbounded error accumulation. Orientation errors result in large lateral position errors and increase proportionally with the distance traveled by the robot. Accumulative errors can be eliminated by combining dead reckoning with inertial navigation systems (refer to 4.1.2).

Odometric sensors include optical encoders, magnetic encoders and potentiometers. They provide good short term accuracy and are inexpensive. The system is flexible as there are no guide paths to be modified. Dead reckoning is used in free ranging vehicles. It is used in conjunction with laser guidance techniques. [2, 3, 5]

4.1.2 Inertial navigation Systems (INS)

Inertial navigation employs gyroscopes and accelerometers. Gyroscopes measure rate of rotation and accelerometers measure acceleration. The angular rate data from gyroscopes is integrated once to determine orientation. The linear velocity rate data from accelerometers is integrated twice to determine linear position.

Accelerometers have a very poor signal – to – noise ratio at lower accelerations. They experience extensive drift and are sensitive to uneven grounds resulting in unacceptable errors. Gyroscopes can be used to compensate for odometry errors, by detecting and correcting orientation errors. Fiber-optic or laser gyros are inexpensive for robot navigation and provide the appropriate accuracy.

Inertial navigation provides quick, low-latency dynamic measurements. The sensors don't require external references. The system is flexible as no guide paths have to be altered. [5, 4, 2]

4.1.3 Active beacons

Active beacons utilise laser, radio, infrared or sonar media. Most methods depend on line-of-sight. Range is limited to the number of beacons. Accurate mounting of beacons determines accurate positioning. The technique offers good reliability but has high costs in installation and maintenance. Beacon systems have the following transmission schemes:

a) "Scanning detectors with fixed active transmitting beacons.
b) Rotating emitters with fixed receiving beacons.
c) Scanning emitter/detectors with passive reflective beacons.
d) Scanning emitter/detectors with active receiver/transmitter beacons." [4]

Two methods are used to determine the vehicle's position: trilateration and triangulation. Trilateration uses distance measurements between beacons and the vehicle. Triangulation uses bearing measurements between beacons and the vehicle's heading. [5, 4]

a. Trilateration

Trilateration can be used to calculate a robot's position based on distance measurements to known beacon sources. The system may use three or more transmitters mounted at known locations in the environment and one receiver on board the robot. Alternatively, one transmitter may be used on board the robot and the receivers maybe mounted on the walls. Time-of-flight data is used to determine the distance between the stationary transmitters and onboard receiver. GPS uses trilateration. [5]

b. Triangulation

Triangulation can be used to calculate a robot's position based on angular measurements to known beacon sources. Three beacons are positioned at known locations and there is an unknown position, P (refer to figure 4-1). The location of P can be calculated by angular measurements ($\alpha_1, \alpha_{2,} \alpha_3$) between the beacons and P. It is necessary for at least three beacons to be visible for triangulation. This may cause a disadvantage should one of the beacons be obstructed by a machine, mobile robot or human. To solve this problem multi-beacon arrangements are utilised. A combination of triangulation algorithms may be applied for better results. [3, 5]

4.1.4 Types of Beacons

a. Radio beacons

Localized beacon systems apply transmission scheme (a) in 4.1.3, for autonomous mobile robot control. The positional information is calculated at the mobile end. The system allows for an unlimited number of robots with a finite number of beacons. All transmitters are synchronized to transmit a continuous wave in phase but the receiver is not synchronized to do so. The robot can therefore, only measure differences in the time taken for signals to arrive from the transmitters located in the environment. The robot determines its position by locating the intersection of the

hyperbolic line-of-positions (refer to figure 4-2). The intersection is formed by the difference in phase of signals received from two pairs of continuously broadcasting transmitters [4, 6].

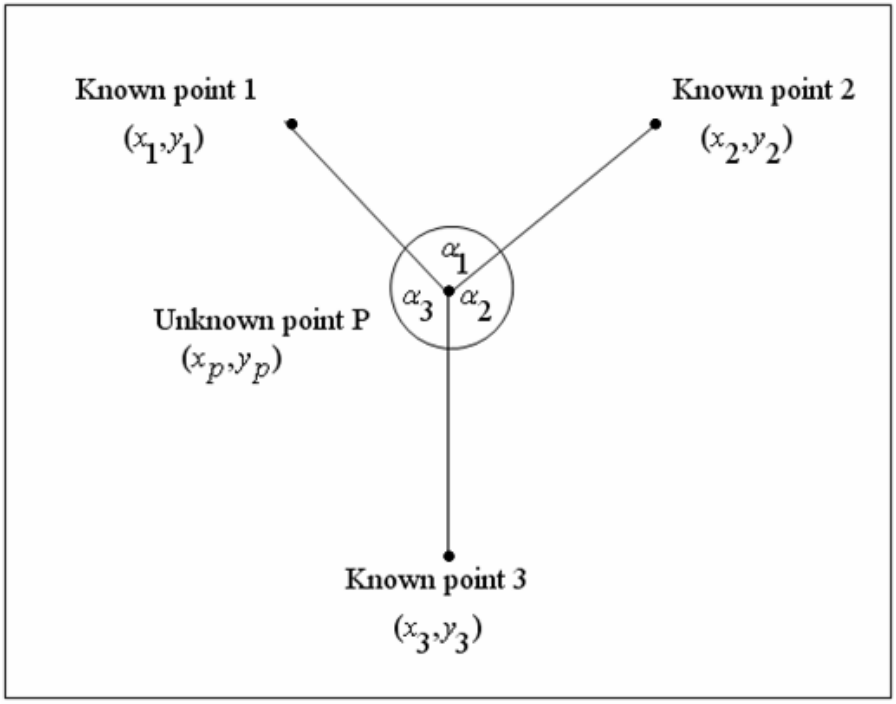


Figure 4-1 Representation of triangulation [3]

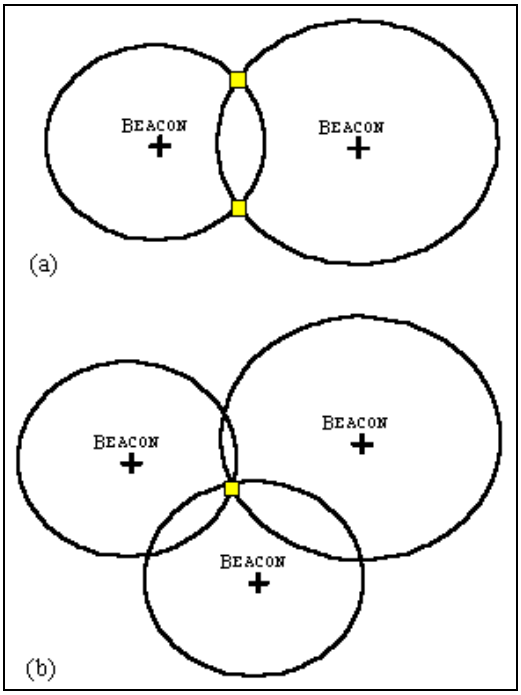


Figure 4-2 Possible robot positions at the intersection of circles when range to (a) two, and (b) three, transmitters is known [4]

b. Ultrasonic Beacons

Ultrasonic beacons use either scheme (a) or (b). Ultrasonic sensors are also used for proximity detection. To facilitate both localization and obstacle avoidance, passive sonic beacons with unique reflection properties can be implemented. By using trilateration the robot can perform an absolute position fix, and find its position relative to any objects in the vicinity. [4]

c. Optical Beacons

Optical and laser systems usually use scheme (c), with passive retro-reflective beacons as they are low-cost. Beacon systems are flexible as the target beacons can be easily moved. However, a high level of processing power is needed to interpret the data. [2]

4.1.5 Landmark Navigation

The robot uses its sensors to identify distinct features. These features may be natural or artificial landmarks. The landmarks have to be easy for the robot to identify and well contrasted with the environment. Landmarks are saved in the robots memory and then used to calculate the robot's position.

It is usually assumed that the current pose (position and orientation) of the robot is known. Therefore the robot only has to search for landmarks in a limited area. Accurate odometry is necessary for landmark identification. [5]

4.1.6 Natural Landmarks

Natural landmarks are sometimes difficult to detect and match from sensory inputs. Computer vision natural landmarks include long vertical edges, such as doors, wall junctions, and ceiling lights. Range sensor natural landmarks include corners or edges, or long straight walls. It is important to select features that are easily distinguishable from the environment as this will increase positioning accuracy. No changes have to be made to the environment making this a flexible system. However, the high demands of processing power needed to analyze the natural landmark algorithms render some on board computers insufficient to allow real time motion. [5]

4.1.7 Artificial Landmarks

Artificial landmarks are easier to detect and well contrasted with the environment. Computer vision artificial landmarks include black rectangles with white dots in the corners, or spheres with horizontal and vertical calibration circles. The accuracy achieved depends on the accuracy with which the geometric parameters of the landmark images are extracted from the image plane. This in turn depends on the relative position and angle between the robot and the landmark. Artificial landmarks are inexpensive. The environment has to be adapted to accommodate the landmarks. Ambient lighting, welding arcs and electronic drift may affect the reliability of both landmark systems. [5, 2]

4.1.8 Line following

Line following is another form of landmark navigation. It is applied widely in industry for most AGVSs. The line can be considered as a continuous landmark. Sensors are located very close to the line, limiting the vehicle range to the immediate vicinity of the line. Line following does not permit free movement of the vehicle – this is the main feature that sets AGVs apart from mobile robots. Line types include:

a. Inductive or wire guidance
b. Reflecting tape guidance or optical tape guidance
c. Ferrite painted guidance
d. Thermal marker guidance [4]

These methods are explained in section 4.2.

4.1.9 Light following

Light following is another form of landmark navigation. Light following involves using photo-receptive sensors to detect the presence of light. The direction of the vehicle is controlled through software depending on readings received.

Fluorescent or halogen lights can be used. The light/s can be setup in several arrangements. The lights can be suspended from the ceiling, for example in the work done by G. Bright [2], or

positioned close to the destination point, on the floor, or on the machine itself. There can be a single light positioned at the destination point or multiple lights leading to the destination point.

The lights are turned on at the destination point or on the path which the vehicle has to follow to reach the destination point. If there is a single light, the vehicle can move freely, correcting itself as it proceeds to the destination. If there is a path of lights to the destination, the vehicle follows a 'line of light' or rigid path and is limited to the path as in line following.

The technique can be easily implemented in any environment. However, sensor readings can be affected by ambient lighting conditions. Routing must be simple as junctions are not easily managed. If the lights are not permanently fixed they can be moved around allowing for a flexible path.

Reliability depends on the intensity of the light/s used. The lights have to be replaced when intensity decreases or is depleted. Accuracy depends on the type of sensing devices being used. This technique is utilised in entertainment, hobby and toy robots as illustrated in figure C-2, Appendix C.

4.1.10 Global Positioning System

Global Positioning System (GPS) is a satellite-based navigation system developed by the US Department of Defense. It consists of twenty four satellites that orbit the Earth about 12,000 miles above and transmit RF signals to GPS receivers. The receiver's location is determined through advanced trilateration methods. Selective Availability (SA) errors introduced for defense reasons, only allow an accuracy of 100 m for non-military users. By using differential techniques, the accuracy of GPS is suitable for global reference navigation.

Differential GPS (DGPS) has reference stations that receive the satellite broadcast GPS signal at a known site. The reference stations then transmit a correction according to the error in the received signal to mobile GPS users. DGPS has an accuracy of about 4-6m. Performance increases as the distance between the user and the beacon decreases. [35, 4]

GPS has the following restrictions with regards to mobile robot navigation:

(a) "Periodic signal blockage due to foliage and hilly terrain,

(b) Multi-path interference, and
(c) Insufficient position accuracy for primary (stand-alone) navigation systems." [5]

GPS was rendered unsuitable for the required robot application as it can not be used indoors.

4.1.11 Simultaneous Localization and Mapping (SLAM)

SLAM is a process by which a mobile robot builds a map of its environment and at the same time uses the map to determine its location. Figure 4-3 illustrates a mobile robot moving through an environment taking observations of unknown landmarks with a sensor. At a time instant k, the following quantities are derived:

" x_k : The state vector describing the location and orientation of the vehicle.

u_k : The control vector, applied at time $k-1$ to drive the vehicle to a state x_k at time k.

m_i : A vector describing the location of the i $_{\text{th}}$ landmark whose true location is assumed time invariant.

z_{ik} : An observation taken from the vehicle of the location of the i $^{\text{th}}$ landmark at time k.

The following sets are defined:

$X_{0:k} = \{x_0 , x_1 \Lambda , x_k\} = \{X_{0:k-1} , x_k\}$: The history of vehicle locations.

$U_{0:k} = \{u_1 , u_2 \Lambda , u_k\} = \{U_{0:k-1} , u_k\}$: The history of control inputs.

$m = \{m_1 , m_2 \Lambda , m_n\}$: The set of all landmarks.

$Z_{0:k} = \{z_1 , z_2 \Lambda , z_k\} = \{Z_{0:k-1} , z_k\}$: The set of all landmark observations." [8]

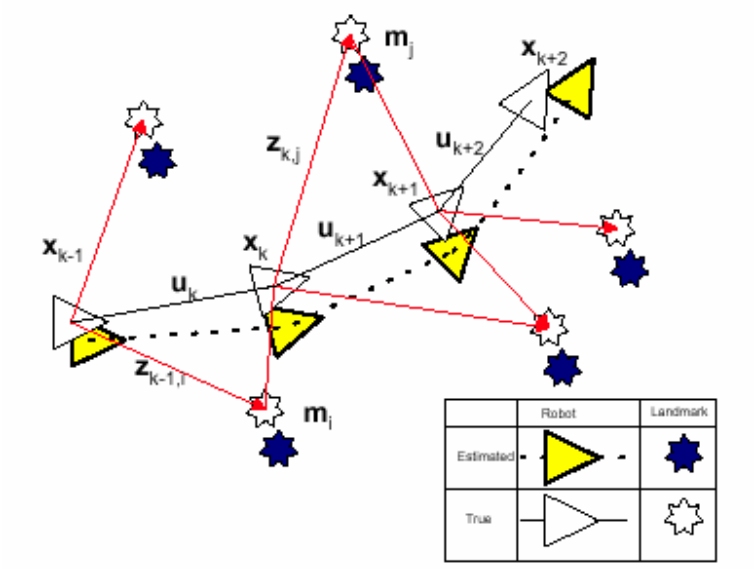


Figure 4-3 "The SLAM problem." A simultaneous estimate of robot and landmark location is needed. The true locations are not known or measured directly. Observations are derived between true robot and landmark locations. [8]

4.1.11.1 Probabilistic SLAM

$$P\left(x_k, m \mid Z_{0:k}, U_{0:k}, x_0\right) \qquad (1)$$

SLAM requires the probability distribution, equation (1), to be determined for all times k. The probability distribution represents the joint posterior density of the landmark locations, m, and vehicle state at time k, x_k, given the recorded observations, $Z_{0:k}$, and control inputs up to and including time k, $U_{0:k}$, with the initial state of the vehicle, x_0. A recursive solution to the SLAM problem is needed. [8]

An estimate for the distribution $P\left(x_{k-1}, m \mid Z_{0:k-1}, U_{0:k-1}, x_0\right)$ at time $k-1$ is used to calculate the joint posterior, following a control, u_k, and observation, z_k, with Bayes Theorem. A state transition model and an observation model are required to describe the effect of the control input and observation respectively.

The observation model depicts the probability of making an observation when the vehicle location and landmark locations are known. It is assumed once the vehicle location and map are derived, observations are independent given the map and the current vehicle state.

$$P\left(z_k \mid x_k, m\right) \qquad (2)$$

The motion model for the vehicle is expressed as a probability distribution in terms of state transitions by a Markov process. The next state, x_k, depends on the immediately proceeding state, x_{k-1}, and the applied control, u_k. It is independent of observations and the map.

$$P\left(x_k \mid x_{k-1}, u_k\right) \qquad (3)$$

The SLAM algorithm is implemented in a time-update form, equation (4), and a measurement-update form, equation (5). Equations (4) and (5) provide a recursive procedure for calculating the joint posterior, equation (1). This is based on all observations, $Z_{0:k}$, and all control inputs, $U_{0:k}$, up to and including time, k. [8]

$$\begin{aligned} &P\left(x_k, m \mid Z_{0:k-1}, U_{0:k}, x_0\right) \\ &= \int P\left(x_k \mid x_{k-1}, u_k\right) \times P\left(x_{k-1}, m \mid Z_{0:k-1}, U_{0:k-1,} x_0\right) dx_{k-1} \end{aligned} \qquad (4)$$

$$\begin{aligned} &P\left(x_k, m \mid Z_{0:k}, U_{0:k}, x_0\right) \\ &= \frac{P\left(z_k \mid x_k, m\right) P\left(x_k, m \mid Z_{0:k-1}, U_{0:k}, x_0\right)}{P\left(z_k \mid Z_{0:k-1}, U_{0:k}\right)} \end{aligned} \qquad (5)$$

The most important insight in SLAM is that the correlations between landmark estimates increase monotonically as more observations occur. Landmarks form a network linked by relative locations or correlations. Precision increases whenever an observation is made. This can be perceived as a network of springs connecting all the landmarks together, as illustrated in figure 4-4. As more observations of landmarks are made by the robot, the springs become increasingly (and monotonically) stiffer. A map of landmarks is formed. The robot's location accuracy measured relative to the map depends on the map quality and the relative measurement sensor.

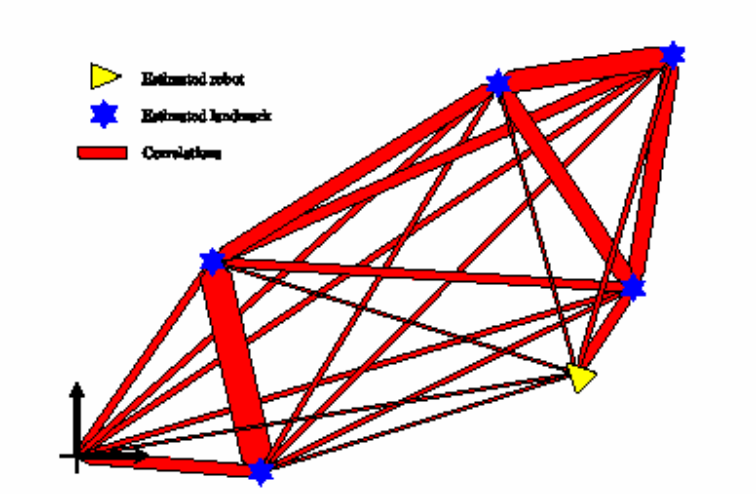


Figure 4-4 "Spring network analogy." The landmarks are connected by springs representing correlations between landmarks. Spring stiffness or correlations increase as the vehicle moves back and forth through the environment. This is indicated by the thicker red links. Changes are propagated through the spring network as landmarks are observed and estimated locations are corrected. The robot itself is correlated to the map. [8]

4.1.11.2 Solutions to SLAM problem

Extended Kalman Filter – SLAM (EKF-SLAM) and FastSLAM are the two main solution methods.

a. EKF-SLAM

The EKF-SLAM method describes the vehicle motion as in equation (6). Where $f(x_{k-1}, u_k)$ represents vehicle kinematics and where w_k are additive, zero mean uncorrelated Gaussian motion disturbances with covariance, Q_k. [8]

$$P(x_k \mid x_{k-1}, u_k) \Leftrightarrow x_k = f(x_{k-1}, u_k) + w_k \qquad (6)$$

The observation model is represented as in equation (7). Where $h(x_k, m)$ depicts the geometry of the observation and where v_k are additive, zero mean uncorrelated Gaussian observation errors with covariance R_k.

$$P(z_k \mid x_k, m) \Leftrightarrow z_k = h(x_k, m) + v_k \tag{7}$$

These definitions are used to calculate the mean

$$\begin{bmatrix} \hat{x}_{k|k} \\ \hat{m}_k \end{bmatrix} = E\begin{bmatrix} x_k \\ m \end{bmatrix} \mid Z_{0:k} \tag{8}$$

And covariance

$$\begin{aligned} P_{k|k} &= \begin{bmatrix} P_{xx} & P_{xm} \\ P_{xm}^T & P_{mm} \end{bmatrix}_{k|k} \\ &= E\left[\begin{pmatrix} x_k - \hat{x}_k \\ m - \hat{m}_k \end{pmatrix} \begin{pmatrix} x - \hat{x}_k \\ m - \hat{m}_k \end{pmatrix}^T \mid Z_{0:k} \right] \end{aligned} \tag{9}$$

of the joint posterior distribution $P(x_k, m \mid Z_{0:k}, U_{0:k}, x_0)$ from:

Time-update

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{10}$$

$$p_{xx,k|k-1} = \nabla f \, P_{xx,k-1|k-1} \nabla f^T + Q_k \tag{11}$$

Where ∇f represents the Jacobian of f evaluated at the estimate $\hat{x}_{k-1|k-1}$. [8]

Observation-update

$$\begin{bmatrix} \hat{x}_k \\ \hat{m} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{m}_{k-1} \end{bmatrix} + W_k \left[z(k) - h(\hat{x}_{k|k-1} \ , \ \hat{m}_{k-1}) \right] \tag{12}$$

$$P_{k|k} = P_{k|k-1} - W_k S_k W_k^T \tag{13}$$

Where

$$S_k = \nabla h \, P_{k|k-1} \, \nabla h^T + R_k \tag{14}$$

$$W_k = P_{k|k-1} \nabla h^T S_k^{-1} \tag{15}$$

Where ∇h represents the Jacabion of h evaluated at $\hat{x}_{k|k-1}$ and $\hat{m}_{k-1}$. [8]

This EKF-SLAM solution has many of the same advantages and disadvantages of the standard EKF solutions to navigation or tracking problems.

b. FastSLAM

The joint SLAM state has a vehicle component and a conditional map component.

$$\begin{aligned} & P\left(X_{0:k} \ , m \mid Z_{0:k} \ , U_{0:k} \ , x_0\right) \\ & = P\left(m \mid X_{0:k} \ , Z_{0:k}\right) P\left(X_{0:k} \mid Z_{0:k} \ , U_{0:k} \ , x_0\right) \end{aligned} \tag{12}$$

The probability distribution is on the trajectory $x_{0:k}$ instead of the single pose x_k. When the map landmarks are conditioned on the trajectory they become independent. This is the reason for FastSLAM's speed. The map is denoted as a set of independent Gaussians with linear complexity.

The main structure of FastSLAM is a Rao-Blackwellised (R-B) state. The trajectory is denoted by weighted samples and the map is computed analytically. [8]

The joint distribution, at time, k is represented by the set

$$\left\{ w_k^{(i)} \ , X_{0:k,}^{(i)} \ P\left(m \mid X_{0:k}^{(i)} \ , Z_{0:k}^{(i)} \ \right) \right\}_i^N$$

The map for each particle is composed of independent Gaussian distributions.

$$P\left(m \mid X_{0:k}^{(i)}, Z_{0:k}\right) = \prod_{j}^{M} P\left(m_j \mid X_{0:k}^{(i)}, Z_{0:k}\right)$$

Recursive estimation is used for the pose states, and the EKF for the map states.

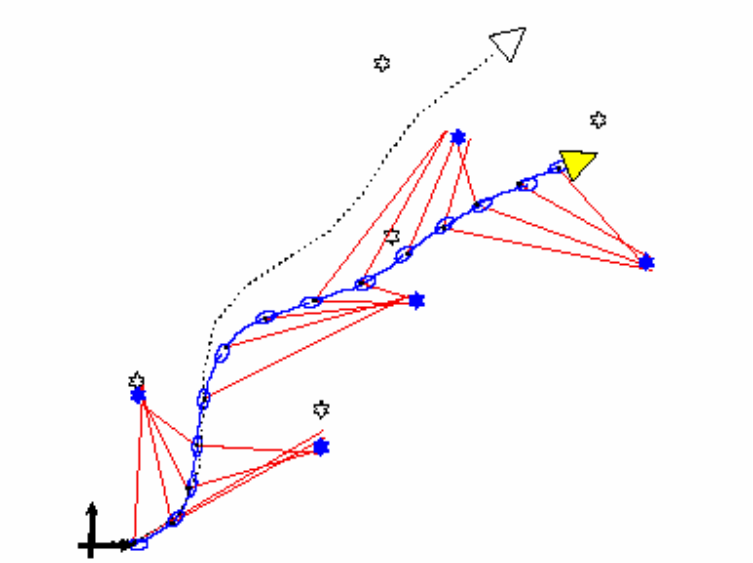


Figure 4-5 A single realization of robot trajectory using the FastSLAM algorithm. The ellipsoids indicate the proposal distribution for each update stage, from which the robot pose is derived. Assuming the pose is perfect, the observed landmarks are updated accordingly. The map for a single particle is determined by the accuracy of the trajectory. Many trajectories offer a probabilistic model of robot location.

Updating the map, for a trajectory particle $X_{0:k}^{(i)}$ is simple. Observed landmarks are analyzed individually as an EKF measurement update from a known pose (refer to figure 4-5). Propagating the pose particles is more complex. [8]

The form of a R-B particle filter is as follows. At time k-1, the joint state is represented by

$$\left\{ w_{k-1}^{(i)}, X_{0:k-1}^{(i)}, P\left(m \mid X_{0:k-1}^{(i)}, Z_{0:k-1}^{(i)}\right) \right\}_{i}^{N}$$

1. A proposal distribution is computed for each particle, conditioned on the specific particle history. A sample is drawn.

$$x_k^{(i)} \approx \pi\left(x_k \mid X_{0:k-1}^{(i)}, Z_{0:k}, u_k\right) \tag{13}$$

The new sample is connected to the particle history.

$$X_{0:k}^{(i)} \stackrel{\Delta}{=} \left\{ X_{0:k-1}^{(i)}, x_k^{(i)} \right\}$$

2. Weight samples are determined using the importance function

$$w_k^{(i)} = w_{k-1}^{(i)} \quad \frac{P\left(z_k \mid X_{0:k}^{(i)}, Z_{0:k-1}\right) P\left(x_k^{(i)} \mid x_{k-1}^{(i)}, u_k\right)}{\pi\left(x_k^{(i)} \mid X_{0:k-1}^{(i)}, Z_{0:k}, u_k\right)} \tag{14}$$

The numerator terms of equation (14) are the observation model and the motion model, respectively. The observation model differs from equation (2) because R-B needs dependency on the map to be marginalized away.

$$\begin{aligned} & P\left(z_k \mid X_{0:k}, Z_{0:k-1}\right) \\ & = \int P\left(z_k \mid x_k, m\right) P\left(m \mid X_{0:k-1}, Z_{0:k-1}\right) dm \end{aligned} \tag{15}$$

3. If necessary resampling is performed, by selecting particles, with replacement, from the set $\left\{ X_{0:k}^{(i)} \right\}_i^N$. Their associated maps, with probability of selection proportional to w_k are included. Selected particles attain uniform weight $w_k^{(i)} = \frac{1}{N}$.

4. An EKF update is performed for each particle, on the observed landmarks, as a simple mapping operation with known vehicle pose. [8]

FastSLAM 1.0 and FastSLAM 2.0 differ in terms of their proposal distribution (step 1) and their importance weight (step 2). FastSLAM 2.0 is more efficient. FastSLAM 2.0 applied in outdoor environments shows that the algorithm is capable of generating an accurate map.

The implementation of SLAM would provide the AGV with the ability to build a map of its environment and determine its location by correlating landmarks as it moves through the environment. Sensors such as laser range finders, sonar sensors and visual sensors are used for SLAM. Sensor data requires high amounts of processing power and equipment can be expensive. Landmarks can be moved allowing for a flexible system. SLAM would enable the AGV to move around freely instead of being confined to a fixed path or line. The AGV would be able to determine the shortest and fastest route to its destination allowing for a more efficient system. Accuracy of the technique is dependent on the sensors, data processing rate and the algorithms applied.

4.2 Techniques used specifically for AGVs

The following techniques are applied mainly for AGVs:

4.2.1 Optical/chemical/metallic guidance

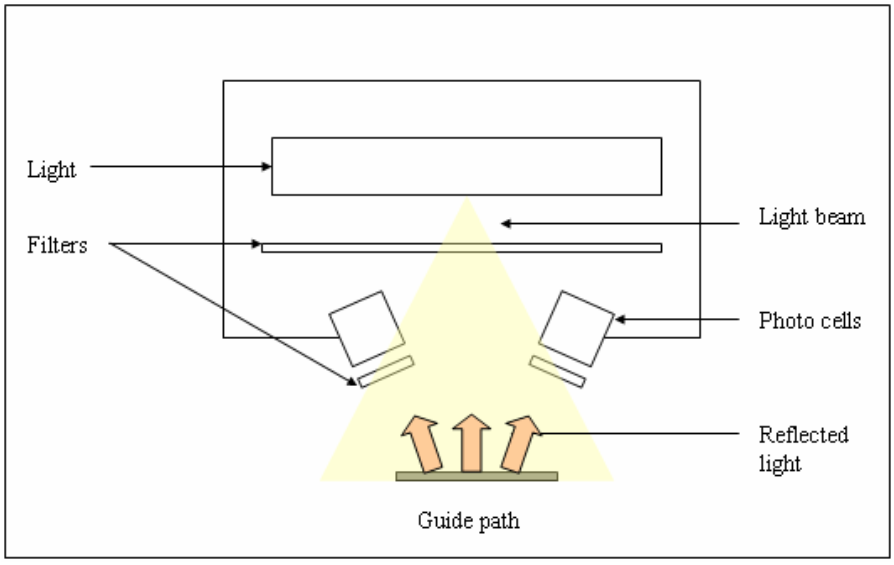


Figure 4-6 Optical guidance setup. Photo cells detect reflected light of guide path. [2]

As illustrated in figure 4-6, a passive strip serves as a path that is detectable by optical sensors. Optical guidance involves a tape laid on the floor. Chemical guidance involves an illuminating fluorescent dye painted on the floor. Ultra violet light on board the vehicle is used to stimulate particles in the chemical path. Metallic guidance involves a metal strip laid on the floor. The paths

reflect light for the optical sensors to follow. A variation in reflected light indicates that the vehicle has strayed from the path. Adjustments need to be made through software to direct the vehicle back to the path.

The cost of the optical/chemical/metallic guidance system is proportional to the length of the guide path. The path can be easily implemented on any surface. Routing must be simple as junctions are not easily managed as in wire guidance.

The guide paths are not embedded in the floor and can be easily moved. This promotes flexibility as new guide paths can be painted or bonded onto the floor with no alteration to the floor and environmental disruption.

Reliability depends on the maintenance of the exposed path. Wear, dirt, sheer resistance and chemical damage deteriorate the condition of the path. Protective covers, good adhesive wear resistant paint and intelligent sensors can be implemented to increase reliability.

Accuracy depends on the type of sensing devices being used. Optical/chemical/metallic systems provide good accuracy due to the precise focusing ability of optical waves. High precision sensors increase the cost of the system.

This technique is utilised in clean industries such as offices and electronic installations. The path tends to become obscured in dirty industries. Regular maintenance is needed to prevent the vehicle from straying from the guide path. [2, 12]

4.2.2 Wire guidance

Wire guidance is also known as inductive guidance or electromagnetic guidance. As illustrated in figure 4-7, a wire about 1.5 mm^2 is embedded in a channel cut into the floor surface. The channel is about 20mm deep and the width is depends on the number of wires in the channel. The normal width is 4mm wide and 8 mm wide if more than three wires are used in the channel. A low voltage (less than 40 V), low alternating current (less than 400 mA) flowing through the wire forms an electromagnetic field around the wire. The field is stronger closer to the wire and weaker further away.

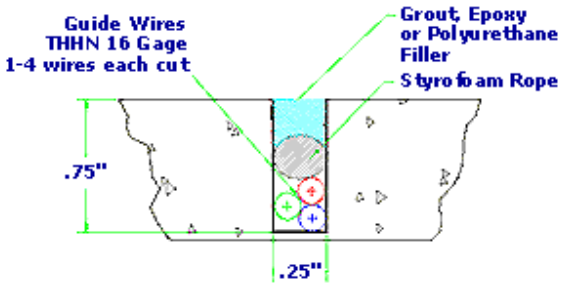


Figure 4-7 Channel with three wires embedded. [45]

The AGV is able to follow the wire using a guide antenna that has two sensing coils positioned on each side of the wire as illustrated in figure 4-8. The electromagnetic field induces a voltage in the coils that is proportional to the strength of the field. The signals generated from the coils are amplified, filtered and compared.

If the signals are equal, the voltage in the coils is the same, and the AGV is centered over the wire. If the AGV deviates to one side of the wire the voltage will be stronger in the coil on that side. The difference in voltage is used to direct the AGV in the correct direction.

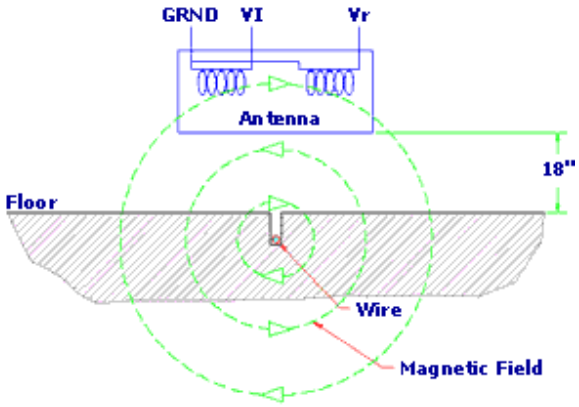


Figure 4-8 Antenna with the two coils that detect the electro-magnetic field around the embedded wire. [45]

Accuracy is limited by the distortion of the magnetic fields. Distortion may result due to poor slotting of the floor or metal objects near the wire. The cost of the electronics is reasonable as compared to free range systems. The embedded wire is protected from damage, making it a reliable guide path. Paths can not be easily changed as this would require cutting the floor and reinstalling

the wires. Flexibility is therefore low. Some environments make it difficult and expensive to install the wires. Wire guidance falls second to laser guidance. [2, 41, 45]

4.2.3 Magnet-gyro guidance

This is a form of inertial guidance. Magnetic sensors are used to detect small magnets embedded in the floor. A gyroscope monitors the AGVs heading. Dead reckoning provides odometer and angle reading of the wheels. Magnet sensors provide information about position deviations and update the odometer distance counting as the magnets are passed.

Pairs of magnets are separated 5 to 10m apart. Installation costs are much lower than wire guidance. However, the on board navigation required costs more than wire guidance equipment. Paths can not be changed easily offering little flexibility. Magnets degrade over ten to fifteen years. Accuracy is less than that offered by wire or laser guidance. [41, 45]

4.2.4 Laser guidance

Laser guidance is the most popular technique applied. Two methods are applied. In the first, a rotating laser is located on top of the AGV, as illustrated in figure 4-9. Targets made of reflective tape or barcodes are positioned throughout the environment. A minimum of three targets must be detected in order to triangulate the AGV's position. Scanned data is compared with stored data in an on-board computer. Dead reckoning provides odometer and angle reading of the wheel which increases position accuracy.

In the second method, laser beams scan over a floor area. Photo receivers located on the floor detect the beams. They send signals to an omni-directional transmitter that transmits a synchronizing signal. Two photo detectors on the AGV detect the signals. By comparing time relationships the AGVs pose can be determined.

Accuracy of both methods is related to range. Line of sight problems may occur. Targets can be easily moved providing flexibility. Laser scanners that provide X and Y coordinates and heading are available. Laser scanners are expensive and the data processing power required is high. [43, 2]

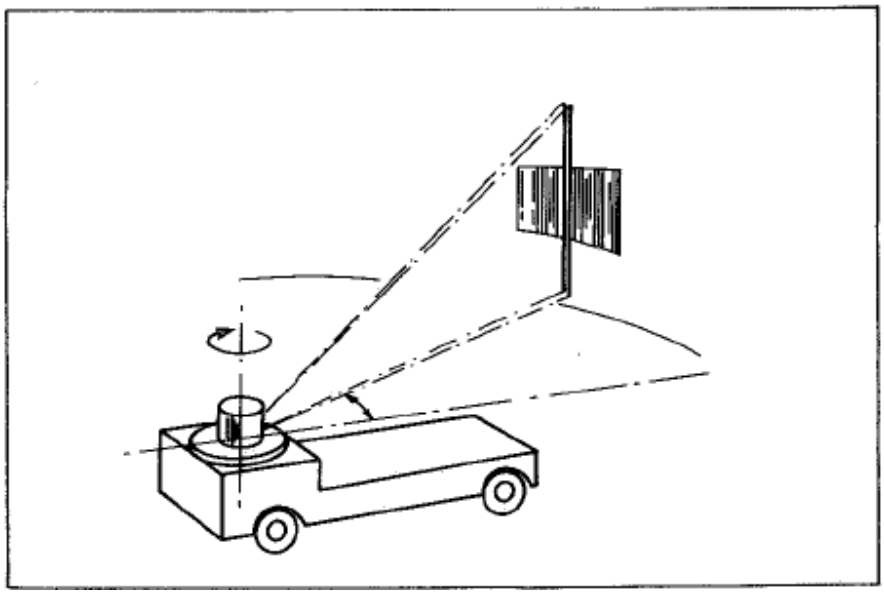


Figure 4-9 An AGV using a rotating laser for guidance. [26]

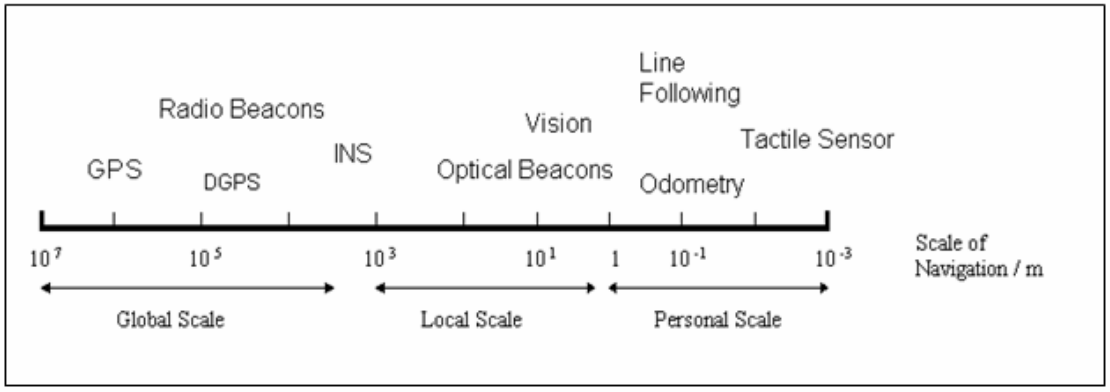


Figure 4-10 Scales of Navigation [4]

4.3 Conclusion of guidance and navigation techniques

Figure 4-10 illustrates the distance achieved by each guidance and navigation system. Odometry, tactile sensors and line following are used on a personal navigation scale. Vision and optical beacons are used on a local navigation scale. INS, DGPS, GPS and radio beacons are used on a global navigation scale. This can be summarized as in table 4-1:

Table 4-1 Scales of navigation.

Distance	**Techniques**
0.001 – 1 m (personal scale)	Odometry
	Tactile sensors
	Line following
1 m – 1 km (local navigation scale)	Vision
	Optical beacons
1 km – 10 000 km (global navigation scale)	Ins
	DGPS
	GPS
	Radio beacons

Laser scanners combined with ultrasonic sensors and odometry sensors prove to be the optimal solution for AGV and mobile robot guidance and navigation. A combination of at least two of the above sensors is applied in robots such as MINERVA [27], Pioneer and AGVs [59].

Laser scanners provide mapping capabilities that would allow the AGV to map its environment and then determine its current pose. However, lasers require large amounts of processing power due to the high data rate and are very expensive. One of the criteria of the project was to provide a low-cost solution as indicated in section 1.5. A commercially available AGV with laser guidance, compared to a wire guidance AGV, has a 15% to 30% increase in vehicle price. It may provide less guiding accuracy and additional programming may be required on site at the system installation. [41]

SLAM could not be implemented as it required a laser scanner. SLAM can be implemented with a camera as in the work performed by Evolution Robotics [61] and Kai Arras [62]. However, high computing power is required to process the data and this leads to further expenses.

The comparisons in table 4-2 were based on a low cost guidance and navigation system for an AGV in a manufacturing environment. Induction guidance is inflexible and installation would require modifications to the laboratory floor and environment disruptions. Active beacons depend on line of sight and the laboratory machines would have caused an obstruction. The beacons available were expensive. GPS could not be applied indoors. Laser mapping was very expensive. Dead reckoning could be applied to indicate the distance traveled and increase accuracy.

It was concluded that optical tape guidance and light following be implemented. Optical tape guidance (or line following) is implemented in industries, warehouses and hospitals. The technique

is reliable and feasible. The AGV would be programmed to follow a path made of black tape against the white laboratory floor. The taped paths allow for flexibility as they can be easily removed.

Table 4-2 Comparison of guidance and navigation techniques

Criteria	**Technique**						
	Wire Guidance	**Optical Tape Guidance (Line)**	**Light Following**	**Active Beacons (IR/RF)**	**GPS**	**Laser Mapping**	**Dead Reckoning**
Flexibility	Rigid	Rigid	Moderate	Moderate	Moderate	Good	Good
Ease of Installation	Difficult	Easy	Easy	Moderate	Easy	Moderate	Easy
Range	Limited to track	Limited to line	Depends on light intensity of environment	Limited to number of beacons	Outdoors only	Unlimited	Unlimited
Accuracy	Good	Good	Moderate	Good	Good	Excellent	Moderate
Cost	Expensive	Cheap	Cheap	Moderate	Expensive	Expensive	Cheap
Reliability	Excellent	Good	Moderate	Good	Good	Excellent	Moderate
Environment feasibility (Manufacturing)	Excellent	Moderate	Moderate	Good	Inapplicable	Excellent	Good

The light following technique allows for path flexibility and is low-cost. The AGV would be programmed to follow a single light positioned at the destination. Light following would be used an alternative technique that the AGV can rely on when conditions for line following are not suitable, such as if the line is damaged. Similarly, if the light intensity is inadequate for light following, line following can be used to allow the AGV to continue with tasks.

Line following depends on a constant and well-maintained line and light following depends on the light intensity of the light being followed. Should either of these parameters vary the AGV would not be able to complete its tasks.

By implementing a variable sensor system the two techniques can easily compliment each other in the changing environment. The system would alternate techniques to compensate for the varying parameters and thereby allow the AGV to complete its tasks. The techniques would be integrated with sensors for obstacle avoidance, emergency and docking purposes.

4.4 Sensors

Sensors that could be utilised for line following, light following and obstacle avoidance, emergency and docking purposes were investigated.

4.4.1 Line following and light following sensors

a. Photodiodes

Silicon photodiodes are semiconductor devices that are sensitive to high energy particles and photons. They can detect the presence or absence of minute quantities of light. They can be calibrated for extremely accurate measurements from intensities below 1 pW/cm^2 to intensities above 100 mW/cm^2.

They have many advantages that make them practical for a wide range of applications:

- They can measure from picowatts to milliwatts of optical power.
- They come in standard packages or the package can be customized to suit your application.
- "They can detect wavelengths from 190 to > 2000 nm.
- They are small and light weight." [10]
- They have reproducible sensitivity.
- They are cheap.
- They have response times as fast as 10 picoseconds.

Silicon photodiodes are utilised in applications such as spectroscopy, photography, analytical instrumentation, optical position sensors, beam alignment, surface characterization, laser range finders, optical communications, and medical imaging instruments. [10, 63]

b. Phototransistors

Phototransistors are transistors that capture light. The most-common type is a NPN bipolar transistor which has an exposed base region. Light strikes the base instead of voltage normally being applied to the base. The phototransistor amplifies variations in the light striking it.

Unlike photodiodes, phototransistors have built in gain. The light absorbed in the phototransistor's base region creates a current. The current gain ranges from 100 to several thousands. The gain enables the phototransistor to be coupled with a load resistor to provide for TTL level voltages for a wide range of light levels.

Phototransistors are low cost and have TTL compatible signal levels. They are utilised in applications where there is more than a few hundred nanowatts of optical power available. As compared to photodiodes, the frequency bandwidth and linearity of phototransistors are relatively limited. Spectral response is restricted between 350 and 1100 nm. There are also very large variations in sensitivity between individual devices and few standard package options. [10, 63]

c. Light Dependent Resistors

Light Dependent Resistors (LDRs) are also known as photoresistors or photoconductors. The sensor consists of a thick film semiconductor material. Its electrical resistance decreases with increasing incident light. Rugged assemblies can endure hundreds of volts and are usually smaller than a 0.25 inch diameter. They react to many frequencies including infrared, visible light, and ultraviolet.

Cadmium sulfide (CdS) based LDRs display sensitivity curves that closely correlate with the sensitivity of the human eye. They are useful in human light perception applications such as headlight dimmers and intensity adjustments on information displays. They can be designed for measuring microwatts to milliwatts of optical power and are inexpensive in large quantities. These characteristics make CdS LDRs suitable in applications such as street light control and in the toy industry where economy is a major consideration.

LDRs are limited in more sophisticated applications requiring sensitivities over a wide spectral range, small variations between individual parts, or no history-dependent response. The sensor resistance depends on the thick-film microstructure, allowing for a wide resistance specification tolerance - a max/min ratio of 3 is common. The resistance also has long term memory which depends on the amount of incident light actually on the sensor and the sensor light history for the past several days. [10, 47]

4.4.2 Concept Selection

The sensor characteristics were compared in table B-5 (Appendix B), and the selection criteria were compared in table 4-3.

Table 4-3 Selection criteria table for sensors

Selection criteria	**Light Sensor**		
	Photo-diodes (Reference)	**Photo-transistors**	**Light dependent resistors**
Available Wavelengths (µm)	0	-	-
Performance-to-cost ratio	0	+	+
Sensitivity	0	0	0
Linearity	0	-	-
Ambient Noise Performance	0	0	0
Dynamic Range	0	-	-
Stability	0	-	-
Reproducibility	0	-	-
Cost	0	+	+
Ruggedness	0	0	0
Physical Size	0	0	0
Ease of Customization	0	-	-
Cost of Customization	0	-	0
Lead time for Customization (weeks)	0	-	0
Sum			
+'s	0	2	2
-'s	0	7	6
0's	14	5	6
Net Score	0	-6	-4
Rank	1	3	2
Continue?	Yes	No	Yes

4.4.3 Conclusion of line following and light following sensors

Photodiodes have a fast signal response. High speed is unnecessary for the application involved. The signal needs to be amplified. The sensor is a high impedance source and is susceptible to noise interference.

The phototransistor has the advantage of built in gain. It can be interfaced in the same way as the LDR i.e. with a pull-up or pull-down resistor. However, the phototransistor is more sensitive to longer wavelengths of the spectrum (red and infrared) and may ignore shorter wavelengths (blue).

The LDR is less selective about the spectrum received. The LDR has a slow signal response but provides a large signal which is less susceptible to noise interference. The LDR is therefore more suitable for the application involved. LDRs are readily available, cost-effective and reliable.

LDRs were implemented for the line and light following techniques. The same type of light sensor was used for line and light following therefore another board did not have to be designed. Identical boards were used for the two techniques but addressed differently.

4.5 Obstacle avoidance sensors

4.5.1 Infrared proximity sensors

Infrared proximity sensors transmit a beam of IR light, and then compute the distance to nearby objects from characteristics of the received (reflected) signal. The distance of the object can be calculated in several ways, each with its own advantages and disadvantages:

a. Reflected IR strength

A simple IR proximity sensor that consists of an IR LED and IR photodiode can be applied. However, the IR receiver responds to naturally present IR as well as reflected IR. [56]

b. Modulated IR signal

To prevent interference from natural IR, the transmitted IR signal can be modulated, i.e. a rapidly-

varying IR signal. The receiver circuitry would only respond to the level of the received, matching, modulated IR signal. The DC component of the received signal would be ignored and only the AC component triggered. However, characteristics such as IR reflectance of the obstacle being sensed affects readings. [56]

c. Triangulation

Triangulation is most reliable method to implement. Many new IR sensors use triangulation and a small linear CCD array to compute the distance and/or presence of objects in the field of view. Operation of the Sharp GP2DXX IR sensor is described. The emitter emits a pulse of IR light. The light travels out in the field of view and either hits an object or continues to travel. If there is no object, the light is never reflected and the reading indicates that no object is present. If the light is reflected off an object, it returns to the detector and forms a triangle between the point of reflection, the emitter, and the detector.

The angles in the triangle depend on the distance to the object (refer to figure 4-11). The receiver is a precision lens that transmits the reflected light onto various portions of the enclosed linear CCD array based on the angle of the triangle formed. The CCD array determines what angle the reflected light was returned at and can calculate the distance to the object.

This new method of ranging is almost immune to interference from ambient light and is indifferent to the color of the object being detected. Detecting a black wall in full sunlight is also possible. [53]

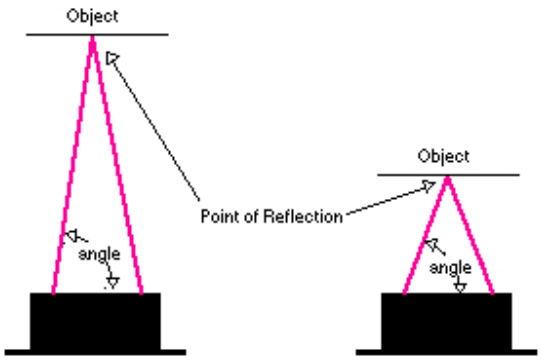


Figure 4-11 Angles obtained with different distances [53]

4.5.2 Ultrasonic sensors

Ultrasonic sensors are used for obstacle avoidance, navigation and map building. There are two types of ultrasonic sensors, Polaroid sonar transducers and Hitechnic sonar sensors. The Hitechnic sensor has separate transmitter and receiver components while the Polaroid transducer combines both in a single piezoelectric transceiver. Figure 4-12 illustrates how waves are emitted and reflected back to the two different sensors.

The transmitter emits a short burst of ultrasonic waves (usually 40 kHz) and the sound wave reflected off an object is detected by the receiver. By measuring the time taken for the echo to return one can determine how far the object is.

Time-of-flight information returned by an ultrasonic sensor depends on the orientation of the object relative to the sensor. A large, flat object, like a wall, parallel to the sensor face produces reasonable data. However, reflection off a non-ideal object may lead to errors. Errors are also caused by the signal (similar to an acoustic wave of energy) that spreads, as it propagates further from the sensor. Most of the energy of the leading edge is confined to a 30 degree cone. If the surface of the object is at an angle with respect to the sensor face then the time of flight information will record the distance to the nearest point within the 30-degree cone. [48, 22]

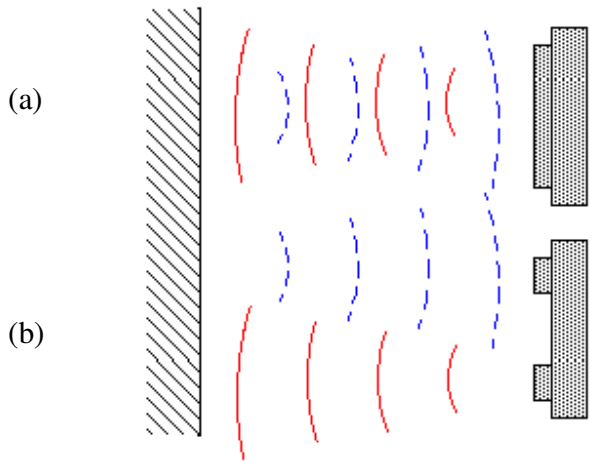


Figure 4-12 (a) Polaroid sonar transducers (b) Hitechnic sonar sensors. [48]

Time-of-flight information returned by an ultrasonic sensor depends on the orientation of the object relative to the sensor. A large, flat object, like a wall, parallel to the sensor face produces reasonable data. However, reflection off a non-ideal object may lead to errors. Errors are also caused by the signal (similar to an acoustic wave of energy) that spreads, as it propagates further from the sensor. Most of the energy of the leading edge is confined to a 30 degree cone. If the surface of the object is at an angle with respect to the sensor face then the time of flight information will record the distance to the nearest point within the 30-degree cone.

The beam is also not entirely confined to a narrow cone as illustrated in figure 4-13. There are so-called 'side lobes' which if reflected first can lead to errors. The equi-potentials of the sound energy level are represented by the dark curve. [48, 22]

4.5.3 Limit switches

Limit switches are electromechanical devices that alert the user when the limit of motion is reached. As an object makes contact with the switch's activator, it moves the activator to the 'limit' where the contacts change state. The limit switch signals to the computer that a limit has been reached and motion is stopped. [11] Figure 4-14 illustrates three different limit switches.

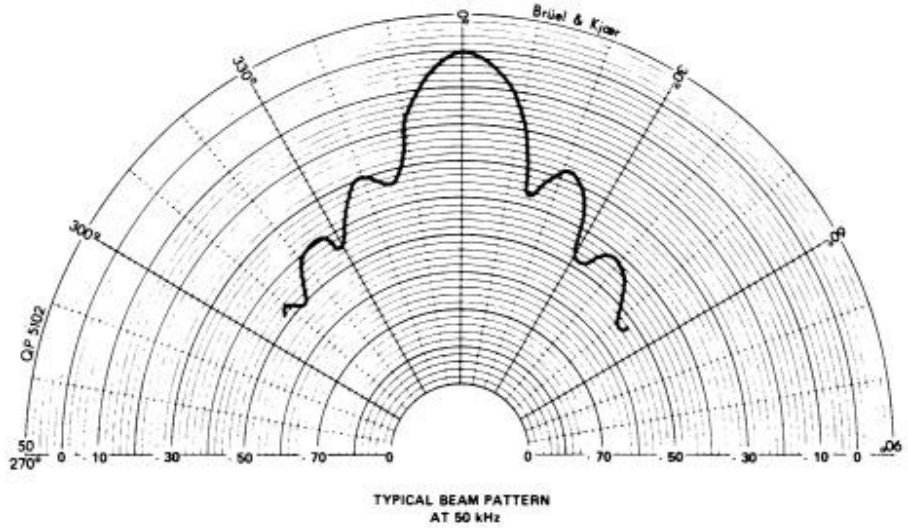


Figure 4-13 Beam pattern [48]

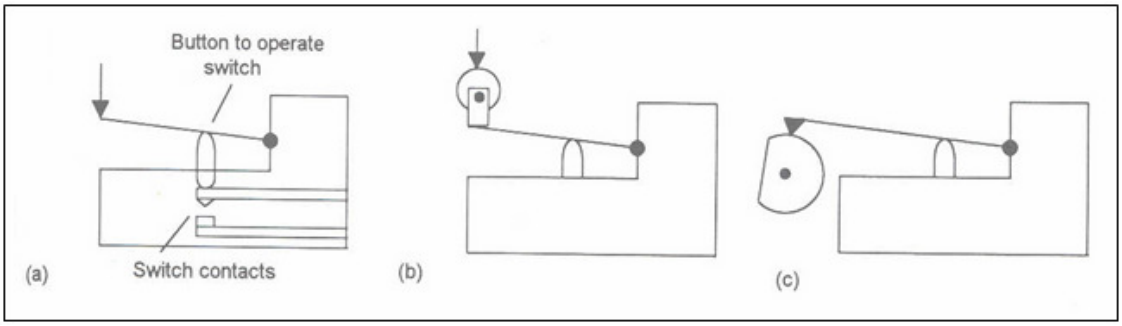


Figure 4-14 (a) Lever-operated (b) Roller-operated (c) Cam operated limit switches [11]

4.5.4 Conclusion of obstacle avoidance sensors

The high speed of the IR signal makes it difficult to measure. The electronic circuitry required to obtain a distance measurement from IR sensors is complicated to implement. Ultrasonic sensors were used because the electronic circuitry required to obtain a distance measurement were less complicated to implement. Distance measurements were obtained by time-of-flight for the ultrasonic sensors. Limit switches were used for emergency and docking purposes as they were readily available and low-cost.

4.6 Summary

This chapter described guidance and navigation techniques used for AGVs and mobile robots. The concept of SLAM was explained. The techniques were compared against relevant criteria in a table. Line following and light following were implemented based on a low-cost guidance and navigation solution. Sensors for the two techniques were investigated and compared in characteristic and selection criteria tables. Light dependent resistors proved to satisfy the necessary sensor requirements. Sensors for obstacle avoidance were discussed. Ultrasonic sensors were implemented to detect obstacles. Limit switches were implemented for docking and emergency purposes.

5 Electronic Hardware

5.1 Control hardware

The AGV was not operational. It needed hardware to enable it to function. It required a sensor system for guidance and five motor boards for navigation. Many hardware interfaces were considered. The main ones are indicated below with a suitable example. Hardware features can be found in Appendix D.

5.1.1 Peripheral Component Interconnect (PCI)

PCI is a high performance expansion bus architecture developed by Intel. Its successor, PCI Express (PCI-E) offers a much higher bandwidth. PCI is still used for applications that do not require the higher bandwidth of PCI-E. [50] The PCI-8158 card from Adlink was considered.

PCI solution from Adlink

PCI-8158**:** Advanced 8-axis Stepping & Servo Motion Control Card

Advantages

1. The PCI interface provides a plug-and play feature that allows easy maintenance.
2. PCI-8158 delivers high-frequency pulse rates of up to 6.55 MHz.
3. It provides smooth motion control for a wide-range of manufacturing applications.
4. The PCI design is modularized. [51]

Disadvantages

1. The card does not support DC motors therefore the AGV's DC motors will have to be replaced. This will be costly and certain mechanical changes may need to be made to the present drive system if the motors are not the same size as the current motors.
2. Some of the features offered by the card are not required at present.
3. The card does not have wireless capability. The AGV will have to be connected by a linking cable. This may prove cumbersome if the cable gets tangled in the wheels.
4. The card is expensive.

5.1.2 PCI eXtensions for instrumentation (PXI)

PXI is a rugged PC based platform for measurement and automation systems. PXI is based on CompactPCI. As illustrated in figure 5-1, PXI systems consist of three basic components: the chassis, the system controller, and peripheral modules. PXI offers high performance. [52]

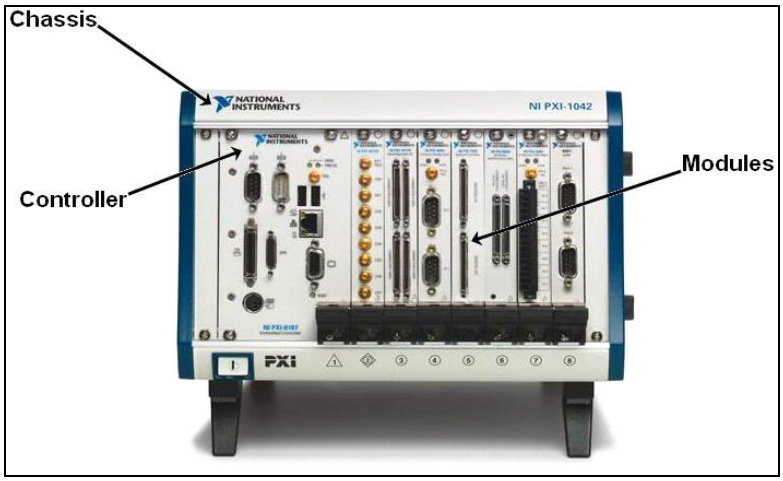


Figure 5-1 PXI system [52]

PXI solution from National Instruments

NI PXI-7356

Advantages

1. PXI-7356 can be applied for simple and complex motion applications.
2. It has eight 16-bit analog-to-digital converters for onboard data acquisition.
3. It is equipped with advanced motion trajectory and triggering features.
4. It offers the most advanced features, including sinusoidal commutation for brushless motors.
5. Low-cost copper links or fibre-optic links are available for extended distances and electrical isolation.
6. Chassis can be multi-linked for multiple control. [52, 54]

Disadvantages

1. The controller does not support DC motors.
2. PXI is a rigid system.
3. Some of the features offered are not required at present.
4. No wireless capability is offered.
5. The controller is expensive.

5.1.3 USB MicroDAQ and Controller cards

The USB MicroDAQ is a device that acts as an interface between the computer and controller card, as illustrated in figure 5-2. The DAQ has a TTL logic line and can be programmed as ports in banks of eight. It has the ability to transmit and receive signals.

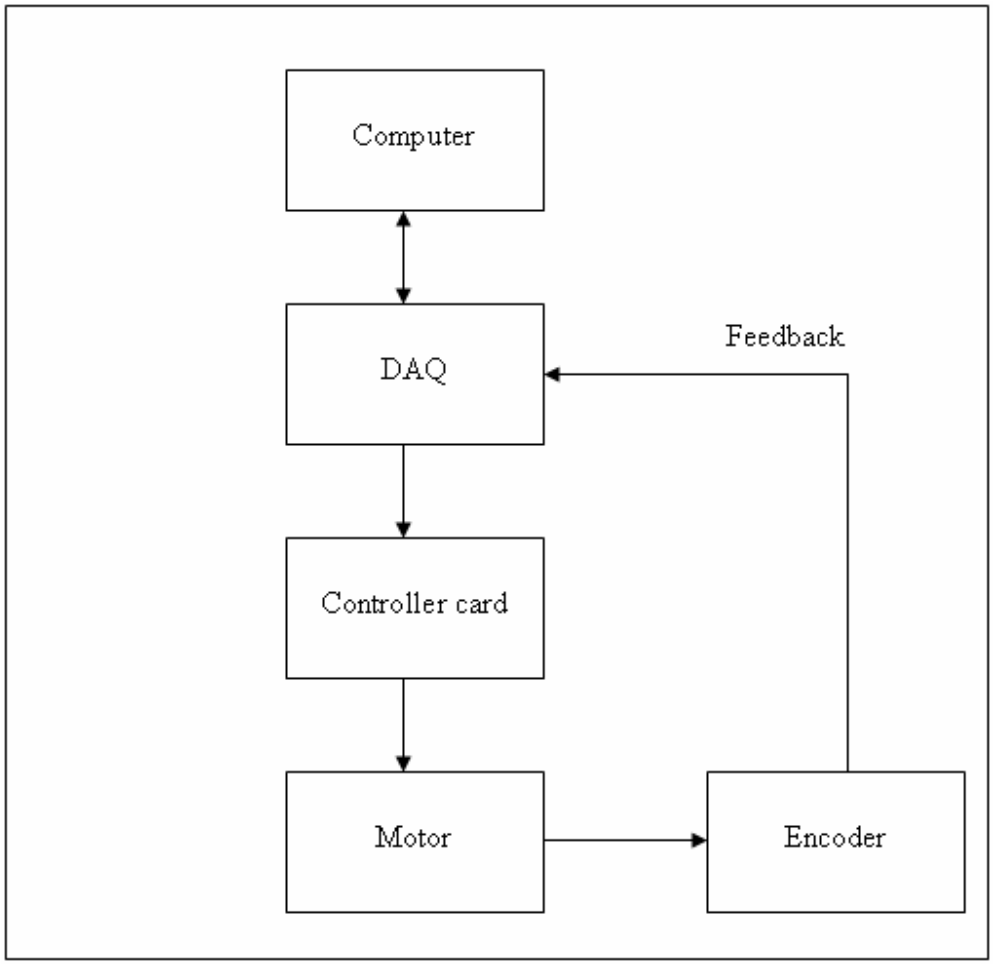


Figure 5-2 Schematic of DAQ interfacing

DAQ solution from Eagle Technology

MicroDAQ BT-120A

Advantages

1. The DAQ system is flexible.
2. Software support is available.
3. Many controller cards can be controlled from one device.
4. The DAQ comes with a USB wireless dongle.
5. The DAQ achieves up to 10m of wireless communication.

Disadvantages

1. The wireless DAQ will require a 9V portable charging system on the AGV. Therefore a linking cable will still be needed.
2. The DAQ has a limited no of I/O lines.
3. DAQs cannot be multi-linked to increase I/O lines.
4. The DAQ is expensive.

5.1.4 Motion Control Module

As illustrated in figure 5-3, a motion control module can be connected via an interface connector to a computer to provide control. A motor driver is connected to the module to control the motor. The module is regarded as a master device, which receives commands from the computer and sends them to the slave device, the motor driver. Modules are capable of various control modes such as PID or PWM and provide for encoder feedback.

Module solution from Acroname

Brainstem Moto 1.0

Advantages

1. The Brainstem is flexible.
2. Modules can be stacked to provide multiple control.

3. Software support is available.
4. The Brainstem offers wireless control with the aid of a wireless device.

Disadvantages

1. The module is expensive.

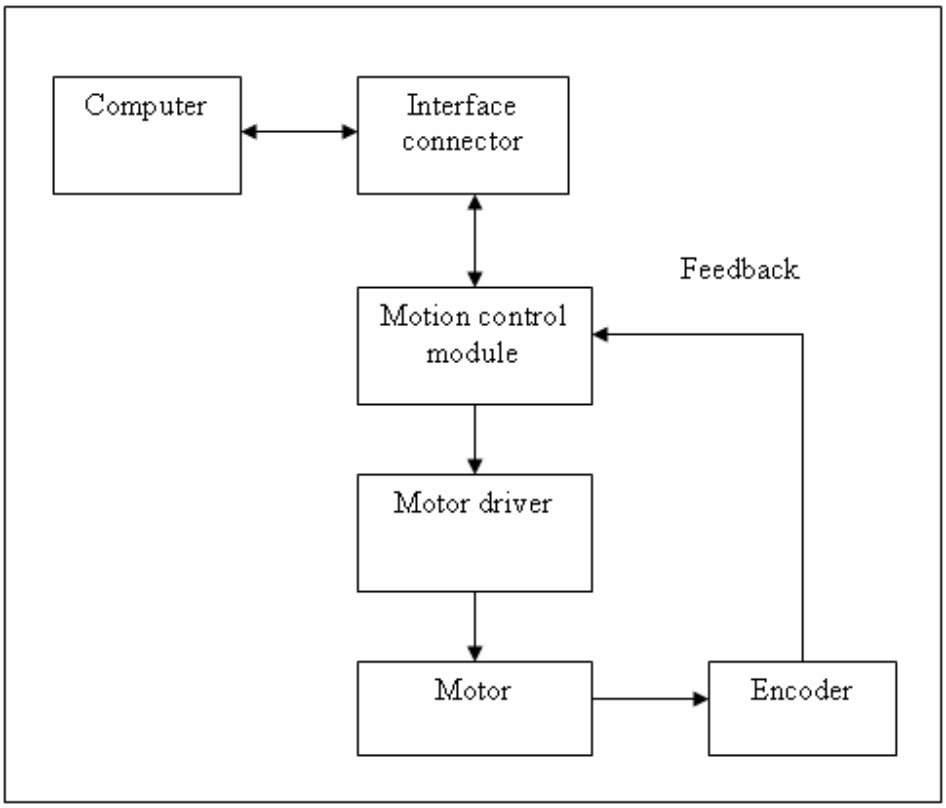


Figure 5-3 Schematic of Module interfacing

5.1.5 Telemetry System

A custom made system was considered. The system would be made according to specifications of the guidance and navigation requirements. As illustrated in figure 5-4, the telemetry system consists of two main components: A USB-Transceiver unit and a robot CPU unit that communicate to each other via RF. The robot CPU unit transfers data to the respective slave modules via the I2C bus. The robot CPU unit acts as a data acquisition device that one can read and write from. The higher level programming to control the AGV had to be done by the user.

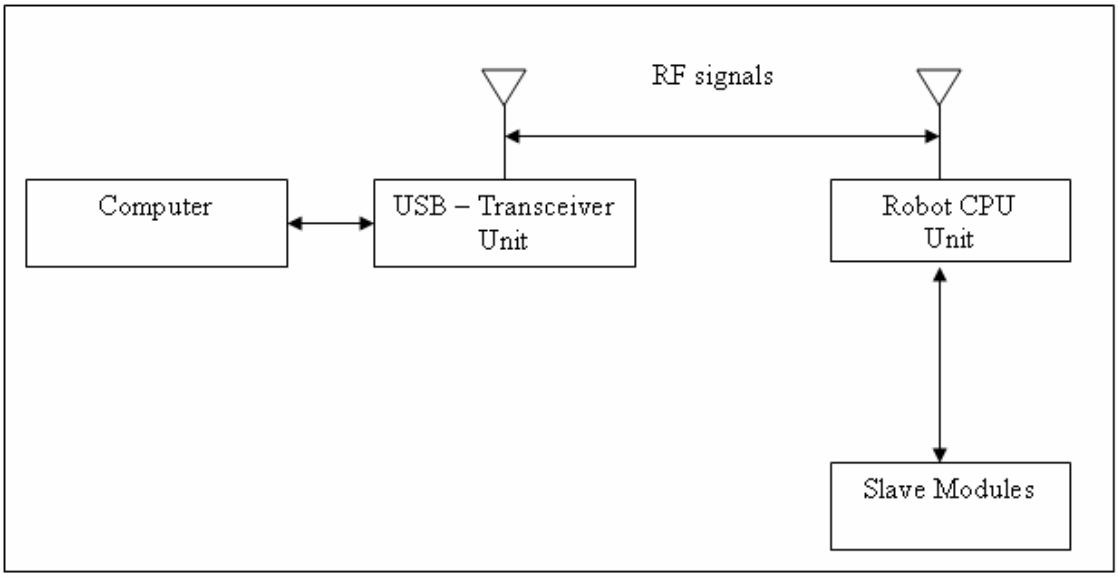


Figure 5-4 Schematic of Telemetry System

Advantages

1. The system is flexible and modular.
2. Many slave modules can be controlled from the master module.
3. Slave modules can be daisy chained.
4. Future slave modules can be integrated to the system.
5. Wireless communication is possible.
6. The system is not limited to a particular program for higher level programming.

Disadvantages

1. The system is expensive.
2. No software support is available.

The control hardware examples were compared in selection criteria table 5-1.

Table 5-1 Selection criteria table for control hardware

Selection Criteria	**Control Hardware**				
	PXI	**PCI**	**BT 120A DAQ & Motion Controller**	**Brainstem Motion Control Module with H-Bridge Motor Drivers & BT device**	**Custom Made Telemetry System**
Capability	+	0	+	+	+
Accuracy	+	0	+	+	+
Flexibility	-	0	+	+	+
Reliability	+	0	-	-	+
Repeatability	+	0	+	+	+
Ease of implementation	0	0	+	-	+
Ease of use	+	0	-	-	-
Programming Requirements	0	0	+	+	-
Wireless capability	-	-	+	+	+
I/O Lines	-	-	-	+	+
Cost	-	0	-	-	-
Sum					
+'s	5	0	7	7	8
-'s	2	0	4	4	3
0's	2	11	0	0	0
Net Score	3	0	3	3	5
Rank	5	4	3	2	1
Continue?	No	No	No	Yes	Yes

It was decided to further assess the competency of the Brainstem motion control module and the custom made telemetry system in order to determine the optimum hardware solution. The need, or weight, for the selection criteria were compared as in table 5-2.

Table 5-2 Concept scoring

Selection Criteria	**Weight (%)**	**Concepts**			
		Custom Made Telemetry System		**Brainstem Moto 1.0**	
		Rating	**Weighted Score**	**Rating**	**Weighted Score**
Accuracy	15	4	0.6	4	0.6
Ease of Implementation	10	5	0.5	4	0.4
Programming	15	3	0.45	3	0.45
Flexibility	20	5	1	3	0.6
Cost	10	3	0.3	3	0.3
Capacity	10	5	0.5	4	0.4
Support facilities	5	4	0.2	4	0.2
Reliability	15	4	0.6	3	0.45
Total Score		4.15		3.4	
Rank		1		2	
Continue		Yes		No	

5.1.6 Conclusion of Brainstem controller

After consultation with previous owners of Brainstem it was concluded that the Brainstem would be unsuitable for the intended application for the following reasons:

Although the Brainstem module is reliable for the control of two motors, operation of two or more Brainstem modules in ENCPOS (encoder position) mode proves inadequate. This is because I2C bus overflows and hence communication between modules is deterred. The microprocessor has a limited capacity. It frequently dies when the module is restarted after values are set to EEPROM. [17]

5.1.7 Conclusion of custom made Telemetry System

The telemetry system satisfied the requirements of the control hardware. The telemetry system was modular. Each slave module had a unique address. The robot CPU processor addressed a specific slave module when required. The system was configurable. The addressing scheme could be changed to accommodate for any number of slave modules. A new address could be added for a new slave module. This allowed for future slave module additions and a more flexible system. The maximum number of addressable slave modules at present is thirty two. A detailed description of the RF telemetry system can be found in section 5.2.

5.2 Telemetry System

An RF telemetry system was custom designed. Figure 5-5 illustrates the system.

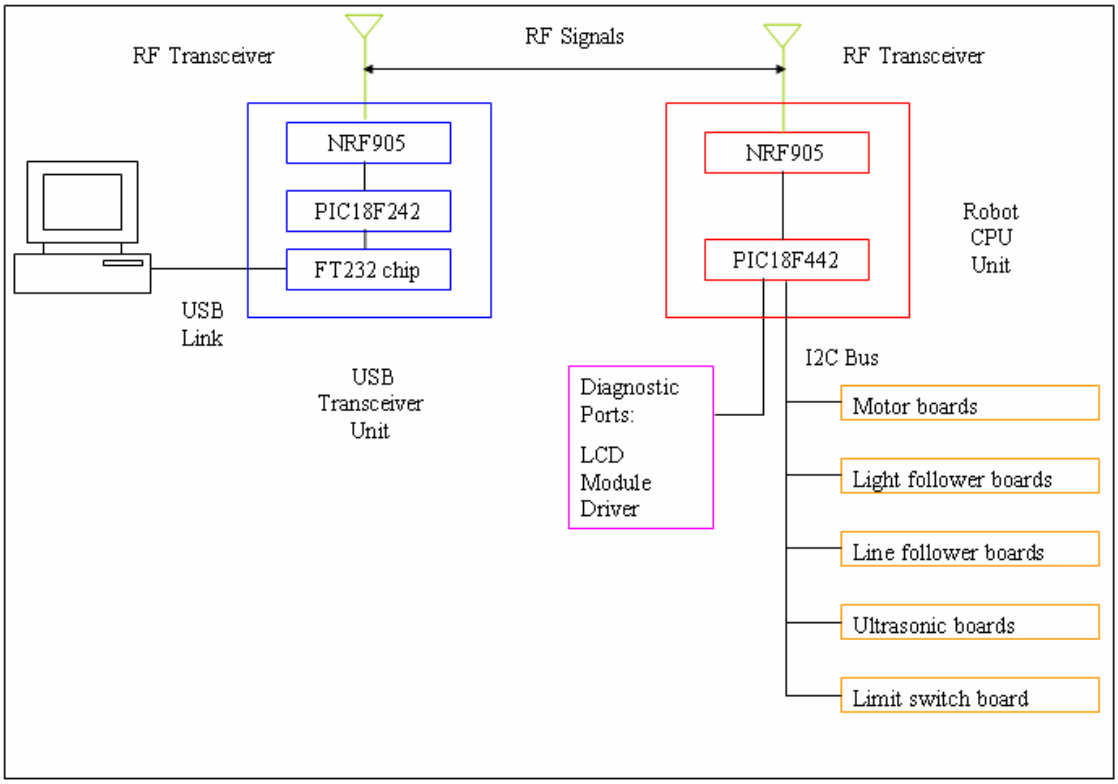


Figure 5-5 RF telemetry system components

The telemetry system consisted of a USB-Transceiver unit and a Robot CPU unit. USB communication was established with a FT232 USB chip. The USB chip allowed communication

between the computer and the PIC18F242 microcontroller. Data was sent from the computer to the PIC at 16 bytes/packet. The PIC controlled the transfer of signals between the computer and NRF905 transceiver chip. Data was transferred via RF signals to the robot transceiver chip (NRF905). The robot microcontroller was a PIC18F442. It served as the master device for the robot control bus. The PIC accepted data and distributed it down the I2C (Inter-Integrated Circuits) bus to the appropriate slave device.

The I2C bus is a bi-directional serial bus that allows communication between integrated circuits. It has two active wires and a ground connection. The active wires are SDA (Serial DAta line) and SCL (Serial CLock line). [36, 37]

As illustrated in figure 5-7, the slave devices were the:

1. Motors boards (five)
2. Light sensor boards (two)
3. Line sensor boards (two)
4. Ultrasonic sensor boards (four)
5. Limit switch board (one)

An LCD module acted as a diagnostic device and verified signal transfer to the robot. Digital signals were sent back to the main computer via RF. The inputs were interpreted and used accordingly in the main program.

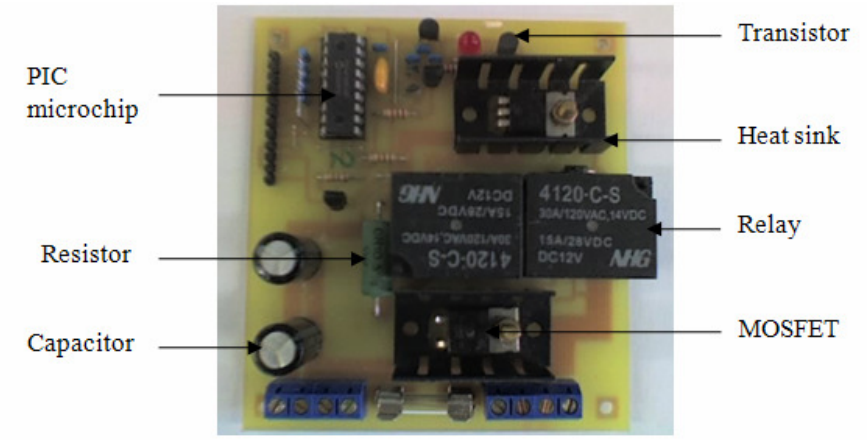


Figure 5-6 Motor controller board

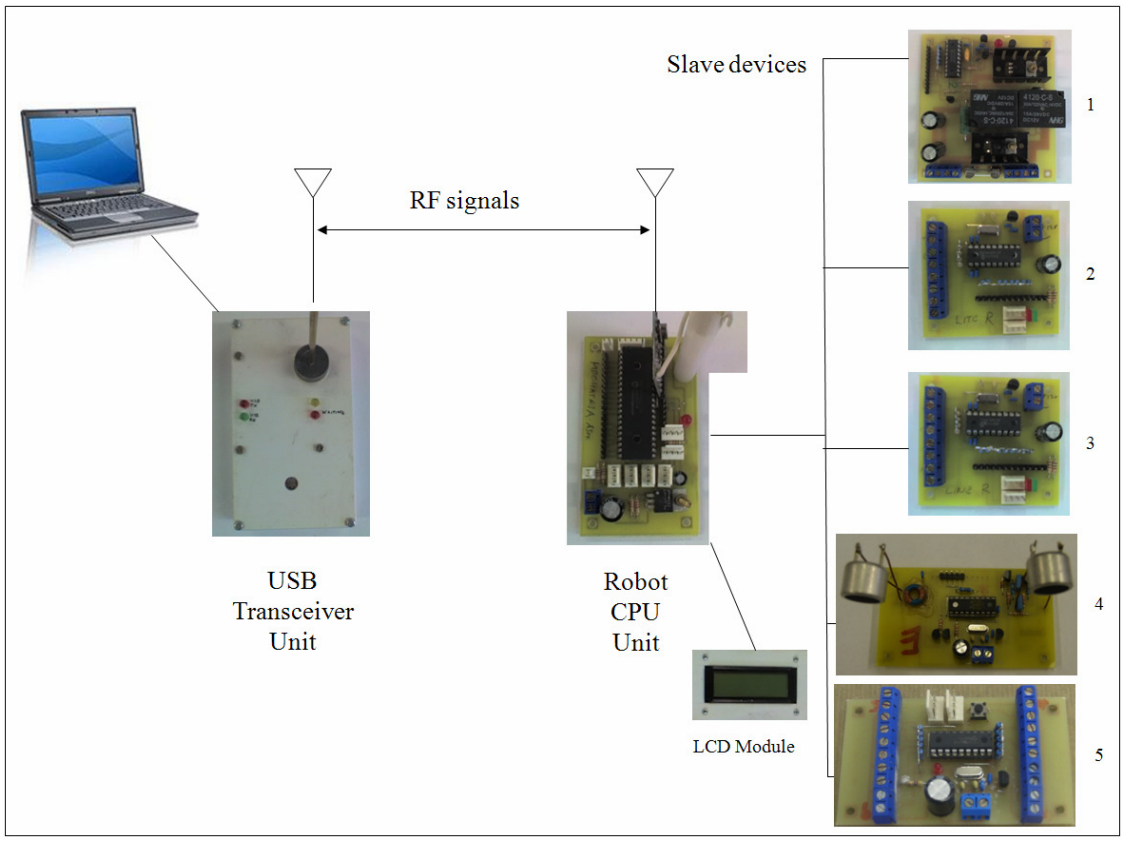


Figure 5-7 RF telemetry system components

5.2.1 Motor Boards

The motor controller board is illustrated in figure 5-6. A maximum of eight motor boards were addressable by the robot CPU. The AGV had five 12V DC motors. Four motors drove the four wheels, one motor per wheel, and one motor drove the materials handling platform. Five motor controller boards were used. Each board was individually addressed and had a PIC16F819 microcontroller.

The command "#%+MOT1SPF1000 *\" was sent to activate motor 1. Where

MOT1 addresses motor 1,
SPF means go forward, SPR means reverse,
1000 is the speed, which ranges between 0000 and 1023.

Motors 2, 3, 4, and 5 were addressed by the respective numbers. The user was able to send two outputs:

1. Pulse Width Modulation (PWM) for speed control
2. Direction control

The following inputs were received:

1. 1 by 10 bit analog to digital conversion (general purpose)
2. 1 switch for open or close
3. 1 motor current indication

PWM is a technique where the power is varied to achieve different speeds. Relays were used to achieve direction control of the motors as they were robust. Each motor required independent speed and direction control to enable omni - directional motion. By varying the speed of each meccanum wheel the direction and motion of the AGV was controlled. [3]

Power was supplied to the motors using two parallel connected 12 V, 7 AMP sealed lead acid batteries. These batteries required less maintenance than a standard 12 V car battery. Power for the sensors was reduced to 3.6V by using a voltage regulator on each PCB.

The line sensor boards, light sensor boards, ultrasonic sensor boards, and limit switch board are explained under sections 6.3.1, 6.4.1, 6.5 and 6.6 respectively.

5.3 Modular boxes for electronic hardware

The electronic hardware was housed in plastic boxes, as illustrated in figures C-3 to C-5, Appendix C. The motor board box housed the four motor boards for driving the AGV. The sensor board box housed the sensor boards, the fifth motor board for the AGV's materials handling platform, and a spare motor board.

The boxes had slots where sensors and motors were easily plugged into their respective boards and activated through user software. Figure 5-8 illustrates the wiring for the connectors in the box. Female connectors were fixed in the slots in the boxes. Wires connected the back of the female

connectors to the boards. Male connectors were connected to the sensor or motor wire ends, as illustrated in figures C-6 and C-7. The male connector plugged into the female connectors to make a connection. The boxes demonstrated modularity and flexibility of the system. They also allowed for an organized system.

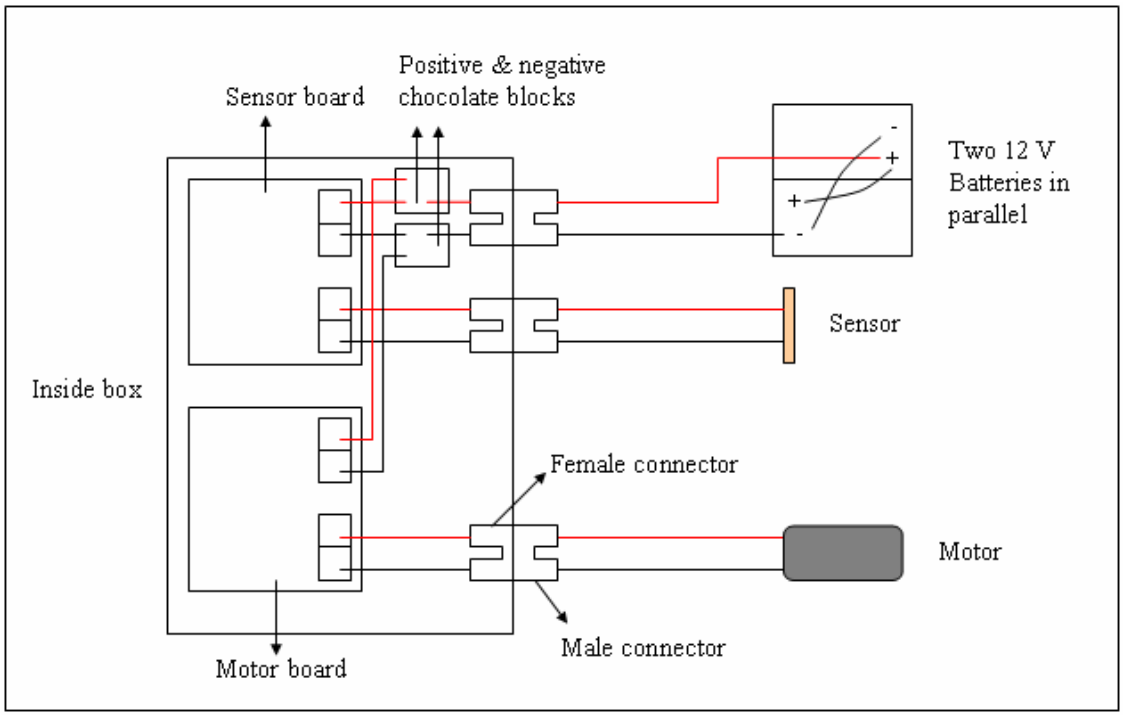


Figure 5-8 Schematic of wiring for connectors in box

5.4 Summary

This chapter described the different types of control hardware investigated: PCI, PXI, USB DAQ, motion control module and telemetry system. The advantages and disadvantages of each type of hardware were discussed. The different types of hardware were compared in a selection criteria table. The Brainstem module and telemetry system provided suitable criteria scores. They were further investigated in a concept scoring table by comparing the need of various criteria (i.e. the weight). The telemetry system proved to be the optimal solution. The operation and components of the telemetry system were described. The features of the motor boards were explained. The modular boxes containing the electronic hardware were presented.

6 Preliminary Software

A preliminary program was written to assess the behaviour of the AGV when line following and light following under constant conditions on the different routes. The data derived from testing would ascertain the accuracy and reliability of the individual techniques and how the two techniques could be combined to compensate for a changing environment.

6.1 Setup

(The setup described was also used for the main software for the variable sensor system, Chapter 8.)

A program for the variable sensor system was written in C using the Dev-C++ compiler, to control the AGV. C is an intermediate-level programming language. C allowed for the quick transmission of signals without delay unlike other programs such as Visual Basic (VB). VB interprets signals slower than compiled languages such as C.

The telemetry system's USB-Transceiver unit communicated via RF to the robot CPU unit. The USB-transceiver unit was connected to the laptop's USB port. Comm. port 4 was setup at a baud rate of 19200, using the Brainstem C API. This API is a very good frame work for general purpose C coding on a number of different OS Platforms. [58]

The aIO shared library was used. This is the lowest level library of the Brainstem C API. It offers cross-platform interaction with files, streams, serial ports, TCP/IP ports, timing, and directories. [58]

The Brainstem files were available from the Brainstem Reference section at [58]. The Brainstem files served as a frame work for the intended files. C header files and c files were compiled to initialise, build, clean up or destroy, flush the stream and write and read from the stream. The write command was sent to activate the line board as follows:

Write (ssp, 17, "#%+LINEL *\\")

Where 'ssp' was the pointer to the serial file, '17' the number of bits and '"#%+LINEL *\\' the command to activate the board. The read command was sent to obtain the feed back as follows:

err = Read (ssp, 50)

Where an error, ‘err’, occurred if the data was not returned correctly and ‘50’ referred to the number of bits read. A delay was needed in between write and read commands and, before the next motor or sensor board was activated. The stream was flushed regularly to prevent incorrect data from being read.

Frequently needed files were called by their header files. The AGV ‘Motor’ files were called when motion commands needed to be executed, based on the data received from sensors. The routines where called when the user specified which route was required. The line following and light following files were called based on the sensor data obtained when using the line variation and light variation techniques (sections 8.3 and 8.4 respectively).

6.2 User interface for preliminary program

The preliminary program demonstrated the AGV’s line following and light following techniques in a constant environment. Ultrasonic sensors were integrated to allow obstacle avoidance and limit switches were integrated for docking and emergency stopping, in both techniques. The AGV’s tasks included transporting pallets between the conveyor, lathe and RM.

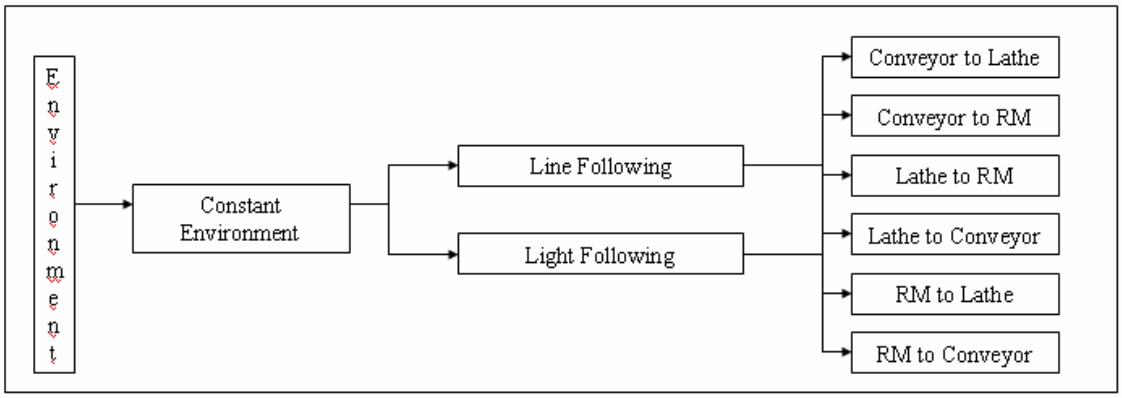


Figure 6-1 Preliminary program layout

The preliminary program layout is illustrated in figure 6-1. The user was prompted to choose either line following or light following for a constant environment. The user was then prompted to choose a route. The AGV utilised the particular technique to complete the assigned task on the route. If an

unknown technique or route was selected, the user was kindly prompted to select again. The main user interface with selections made is illustrated in figure 6-2.

The user has selected the line following technique, 1, and the conveyor to lathe route, 1. The sequence commenced and the AGV proceeded to transport a pallet from the conveyor to the lathe utilising line following.

Sensor feedback was obtained as illustrated in figure 6-3. Sensor feedback continued until the relevant limit switch was compressed on task completion or if a collision occurred. The user was informed through the interface of the AGV status.

When the AGV docked at the lathe, limit switch 1 was compressed on the AGV (s1 = C). The interface appeared as illustrated in figure 6-4. The program returned to the main user interface after task completion.

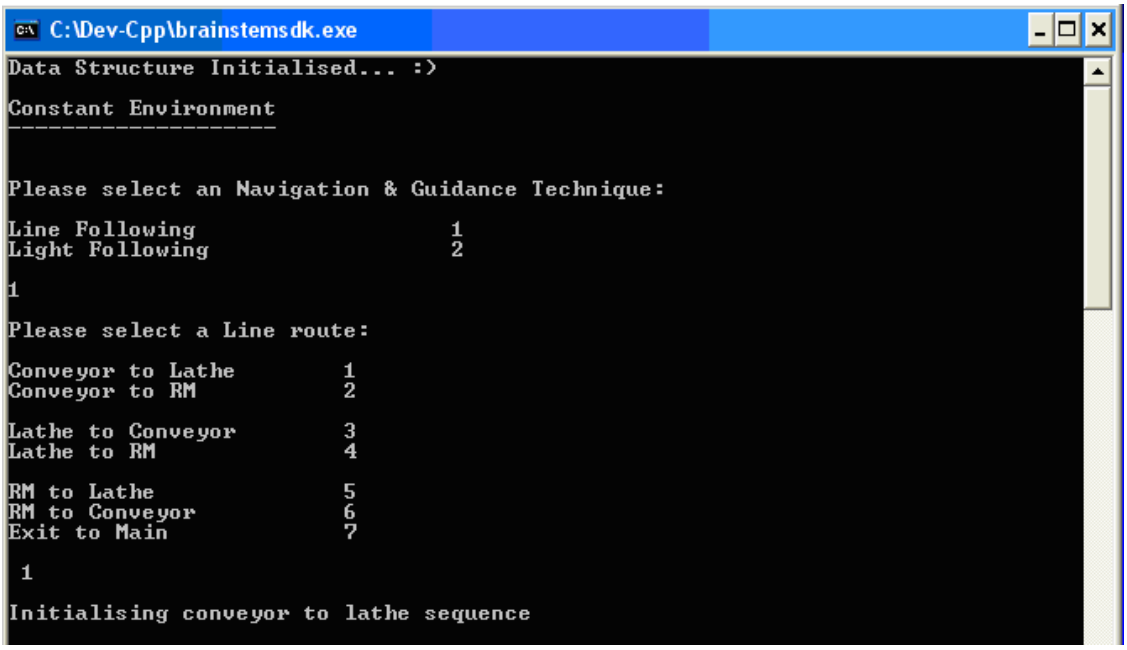


Figure 6-2 Main user interface

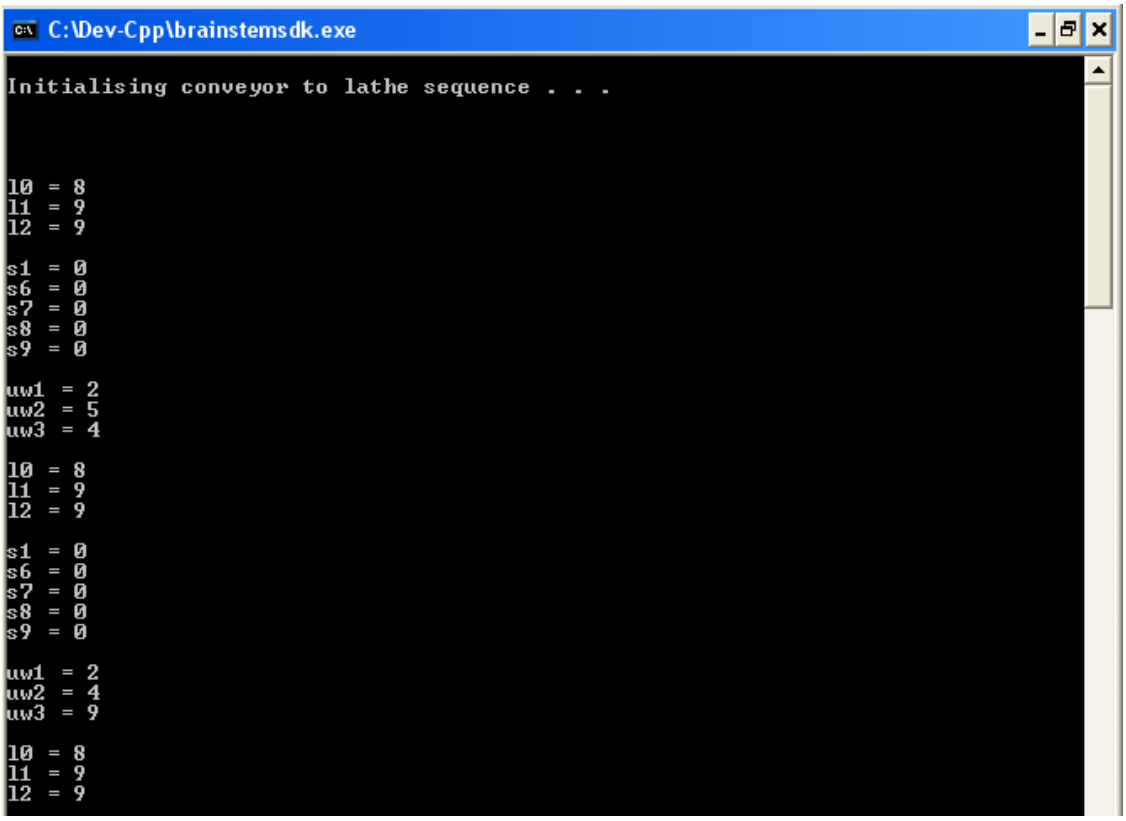


Figure 6-3 Sensor feedback. l0-l2 indicate line sensors, s1-s9 indicate limit switches, uw1-uw2 indicate the ultrasonic sensor reading of the distance an object.

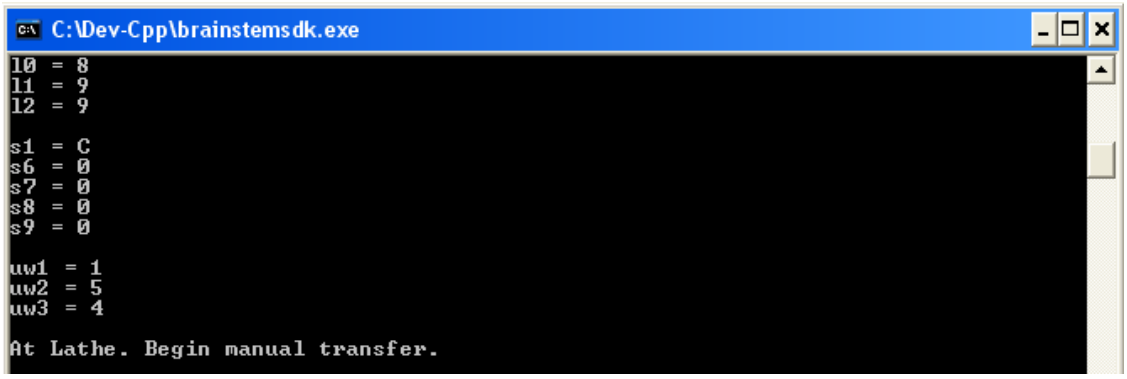


Figure 6-4 Interface after task completion

6.3 Line Following Technique

6.3.1 Hardware

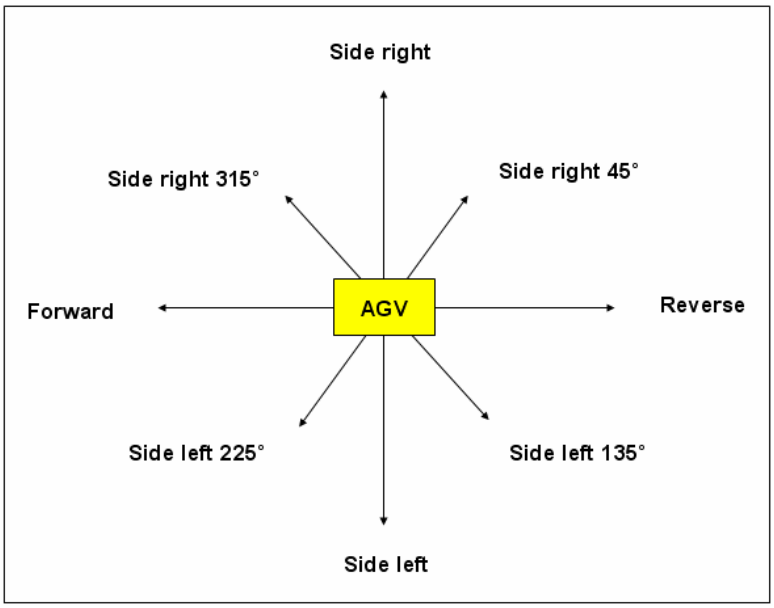


Figure 6-5 Directions of movement of the AGV.

The AGV was programmed for omni-directional movement. Figure 6-5 illustrates the directions of movement of the AGV. Black tape was laid out on a white floor to provide a track for the AGV. Figure 6-6 illustrates the line sensor board. Each line sensor board had a PIC18F819 microcontroller and allowed for four Light Dependent Resistor (LDR) inputs. The boards were addressed as LINEL and LINER.

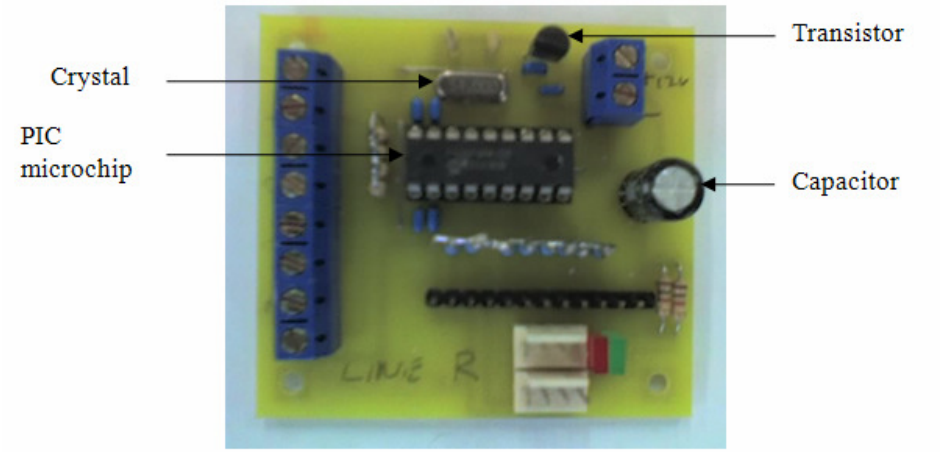


Figure 6-6 Line sensor board

A set of three ORP12 LDRs were positioned in LDR brackets on the four sides of the AGV (refer to figure 6-9), twelve LDRs altogether. As illustrated in figure 6-8, each LDR bracket was suspended from the AGV's frame approximately 30 mm from the floor and consisted of 180 mm long tubes. The LDRs were lowered to a depth of 150 mm in the tubes as illustrated in figure 6-7. This allowed for light collimation of 30 mm which provided suitable LDR readings.

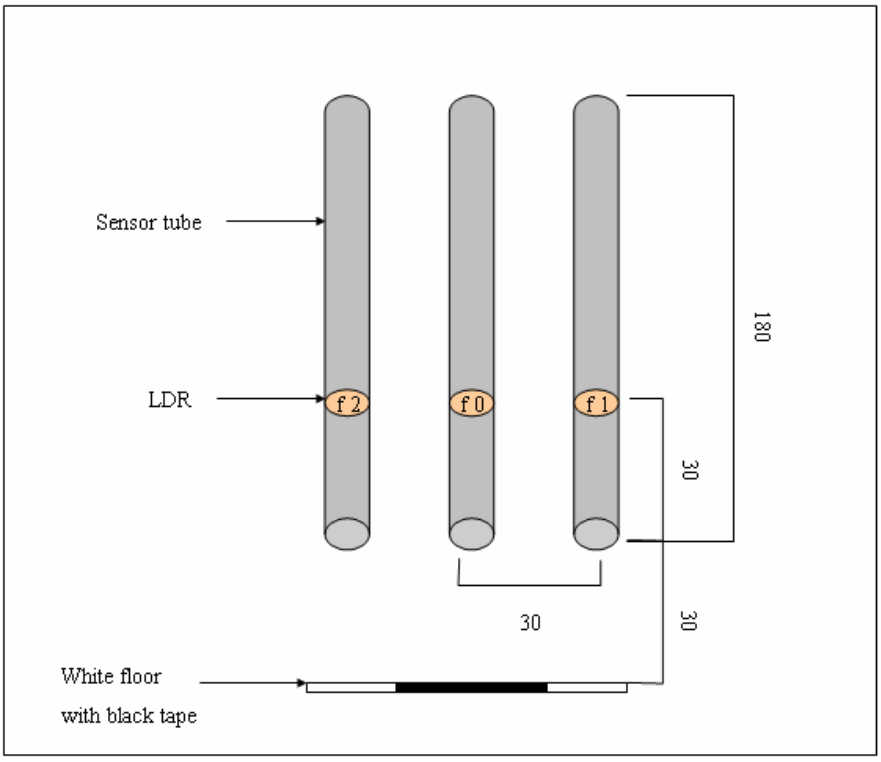


Figure 6-7 LDRs, f2, f 0, and f1 in sensor bracket tubes. All dimensions are in mm.

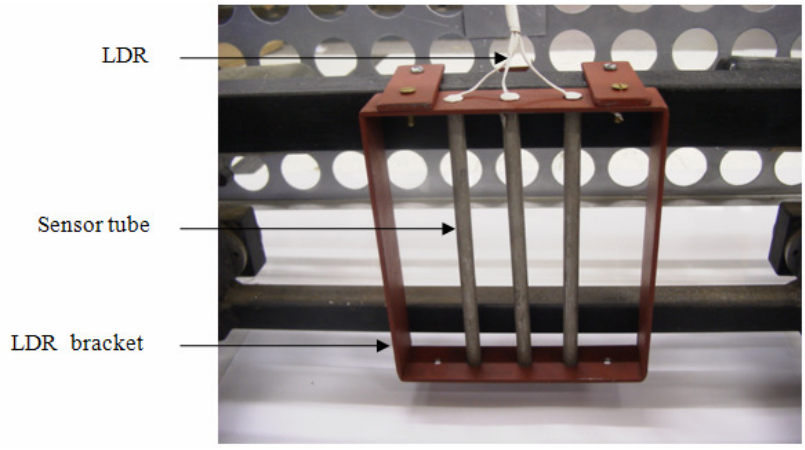


Figure 6-8 LDR bracket

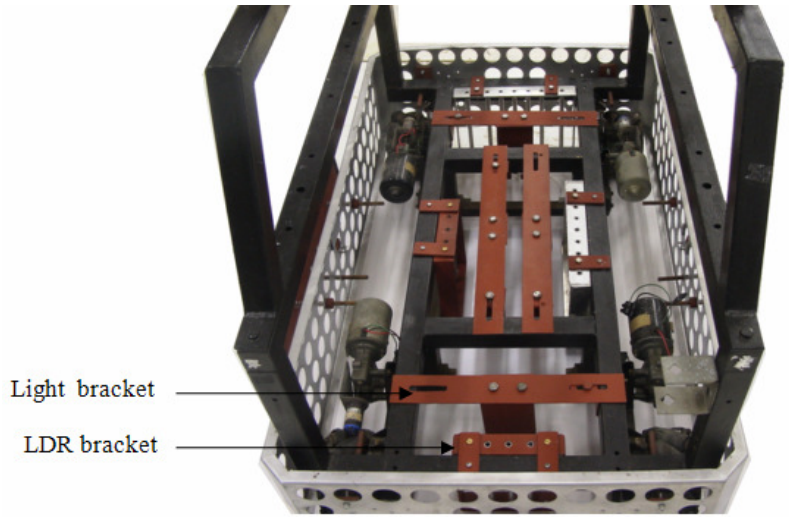


Figure 6-9 Position of LDR and light brackets on AGV.

A shadow created by the AGV's frame made it difficult to differentiate between LDR readings. One could not determine if the LDR was on the black line or white floor as the readings were similar. To eliminate the shadow, 20W bi pin halogen lamps were positioned in light brackets, under the AGV (refer to figures 6-9 to 6-11). The lamps provided sufficient light for the LDRs and allowed differentiable readings between black and white.

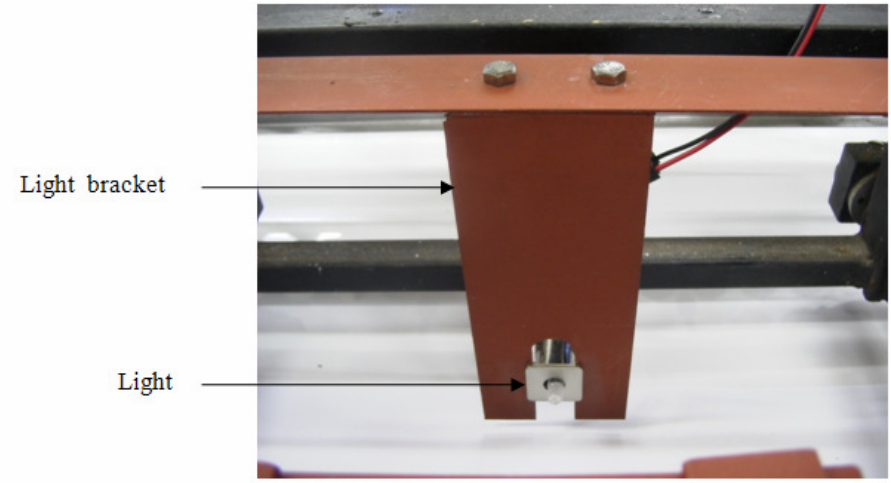


Figure 6-10 Light bracket with 20W light.

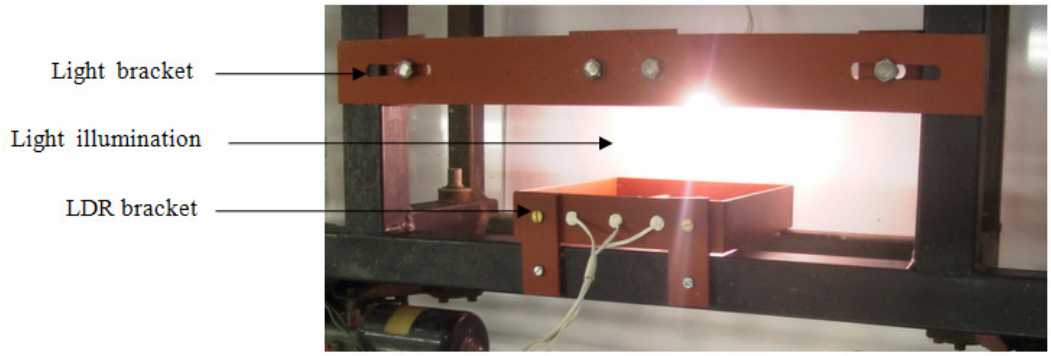


Figure 6-11 Light bracket with 20W light on and line sensor bracket showing area of light illumination.

ORP12 LDRs have a maximum resistance, of 1MΩ, in darkness and a minimum resistance of 100Ω in very bright light. [67] The resistance increased over the black tape as light was absorbed, and decreased over the white floor as light was reflected back to the LDRs. The difference in resistance resulted in a difference in the readings obtained in HyperTerminal. The readings were used to program the AGV to follow the black line.

6.3.2 Program

To activate the line sensor board the command "#%+LINEL *\" was sent.

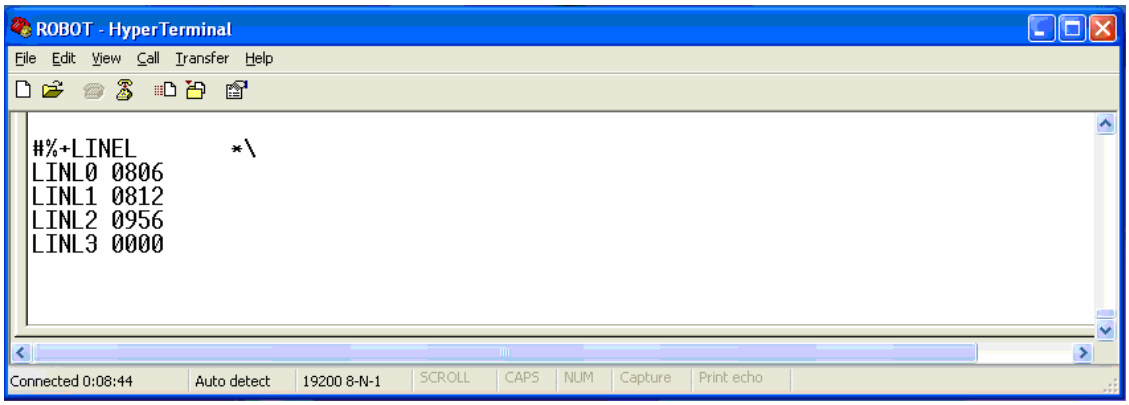


Figure 6-12 Line sensor feedback in HyperTerminal

Readings obtained in HyperTerminal are indicated in table 6-1. A reading below 900 was obtained if the LDR was on the black line (on line). A reading above 900 was obtained if the LDR was on the white floor (off line). In figure 6-12, LINL0, LINL1, LINL2 and LINL3 refer to LDRs 0, 1, 2 and 3

respectively. LINL0 and LINL1 are over the black line and LINL2 is over the white floor. LINL3 is not connected.

Data was returned in the C program in the form of an array. Specific elements were accessed in the array and analyzed to determine if the AGV was on line. The value '9' was used as a threshold in the programming. If an element value of '9' was returned it meant the AGV was off line and if a value below '9' such as '8' or '7' was returned it meant the AGV was on line. Table 6-1 indicates the readings used in C. The logic values are used in tables 6-3 to 6-6. Table 6-2 indicates the position of the twelve sensors used. Figure 6-13 illustrates the sensors used (represented by white circles) and their position.

Table 6-1 Readings in HyperTerminal and C

Position	HyperTerminal	C	Logic Values
Off line	>900	>9	0
On line	<900	<=9	1

Table 6-2 Position of the twelve sensors

Direction	LDR		
	Left	**Center**	**Right**
Forward	f2	f0	f1
Reverse	b2	b0	b1
Side-left	l2	l0	l1
Side-right	r2	r0	r1

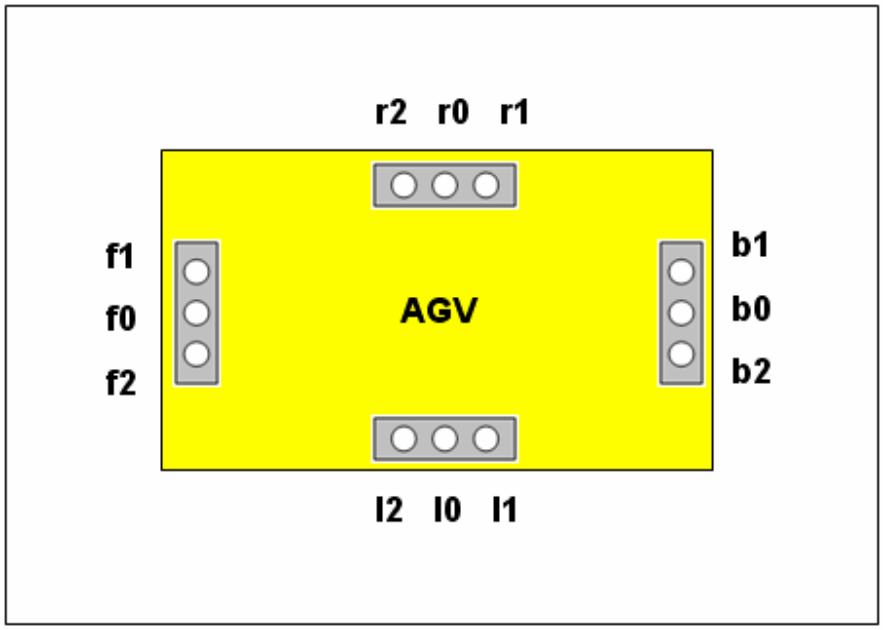


Figure 6-13 Position of sensors. Sensors are indicated by white circles. r2, r0, r1, etc refer to how the sensors were addressed in programming.

Tables 6-3 to 6-6 indicate readings that occurred, the translation, and the action taken in the forward, reverse and lateral directions.

Table 6-3 Translation of readings and action for line following in the forward direction.

Forward Direction				
LDR			**Translation**	**Action**
f 2	**f 0**	**f 1**		
0	0	0	Off line	Stop. User intervention required.
0	0	1	L1 on line	Go right
0	1	0	L0 on line	Go forward
0	1	1	L0 and L1 on line	Go right
1	0	0	L2 on line	Go left
1	0	1	Confused	Stop. User intervention required.
1	1	0	L0 and L2 on line	Go left
1	1	1	L0, L1 and L2 on line	Stop. User intervention required.

Table 6-4 Translation of readings and action for line following in the reverse direction. (Reverse directions were defined as if the user was facing the reverse direction.)

Reverse Direction				
LDR			**Translation**	**Action**
b2	**b0**	**b1**		
0	0	0	Off line	Stop. User intervention required.
0	0	1	B1 on line	Go reverse left
0	1	0	B0 on line	Go reverse
0	1	1	B0 and B1 on line	Go reverse left
1	0	0	B2 on line	Go reverse right
1	0	1	Confused	Stop. User intervention required.
1	1	0	B0 and B2 on line	Go reverse right
1	1	1	Confused	Stop. User intervention required.

Table 6-5 Translation of readings and action for line following in the side-right direction.

Side-right Direction				
LDR			**Translation**	**Action**
r2	**r0**	**r1**		
0	0	0	Off line	Stop. User intervention required.
0	0	1	R1 on line	Go side-right 45
0	1	0	R0 on line	Go side-right
0	1	1	R0 and R1 on line	Go side-right 45
1	0	0	R2 on line	Go side-right 315
1	0	1	Confused	Stop. User intervention required.
1	1	0	R0 and R2 on line	Go side-right 315
1	1	1	Confused	Stop. User intervention required.

Table 6-6 Translation of readings and action for line following in the side-left direction.

Side-left Direction				
LDR			**Translation**	**Action**
l 2	**l 0**	**l 1**		
0	0	0	Off line	Stop. User intervention required.
0	0	1	R1 on line	Go side-left 135
0	1	0	R0 on line	Go side-left
0	1	1	R0 and R1 on line	Go side-left 135
1	0	0	R2 on line	Go side-left 225
1	0	1	Confused	Stop. User intervention required.
1	1	0	R0 and R2 on line	Go side-left 225
1	1	1	Confused	Stop. User intervention required.

As the AGV proceeded along the line, the control program continually corrected the direction of the AGV such that the center LDR remained above the black line and the left and right LDRs remained on either side of the line. If the AGV strayed away from the line the program automatically stopped the AGV and prompted for user intervention. Limit switches were integrated as in table 6-18, but later changed to the allocation in table 6-19 to maintain consistency in the guidance techniques for the different environments and routes. Ultrasonic sensors were initially integrated as in table 6-17,

but changed after conducting high noise level testing to the allocation in table 9-1, to accommodate for the high noise levels. Programming can be found in Appendix E.1.3.

The AGV transferred pallets between the conveyor, lathe and RM by following the black line as illustrated in figure 6-14. The control program ensured that when a “cross line” was reached the AGV followed the correct path. Table 6-7 indicates the directions taken in order to follow a particular path.

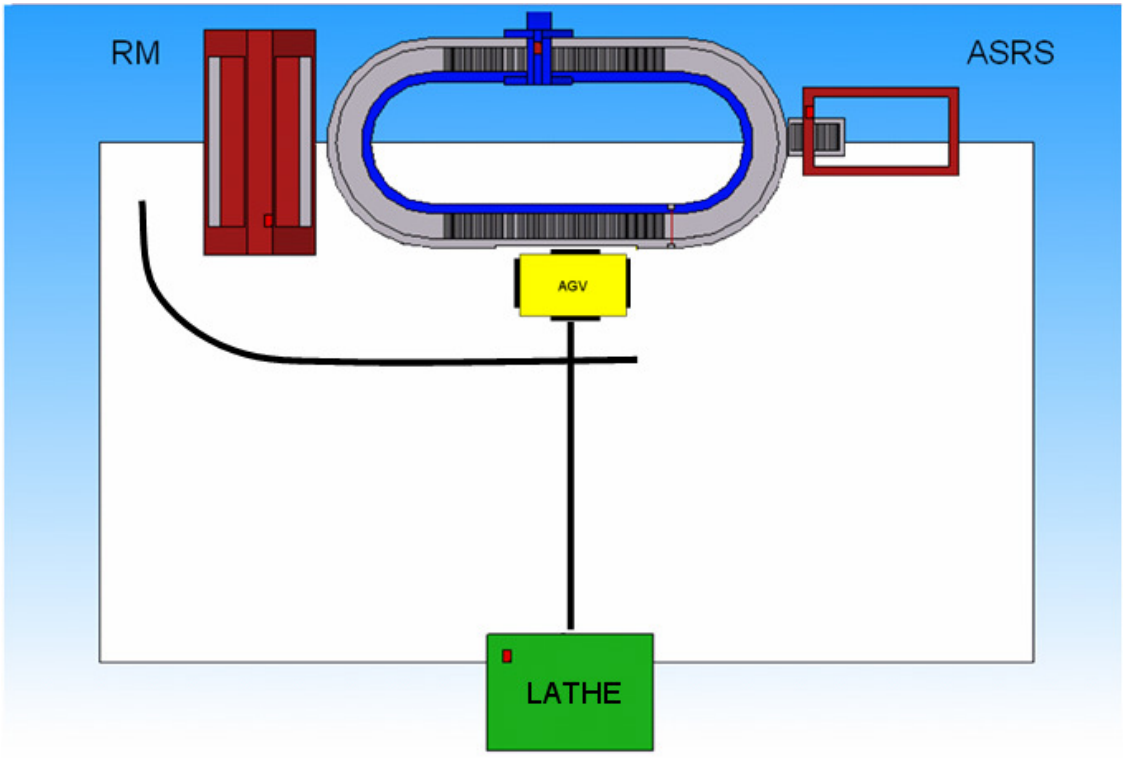


Figure 6-14 Line routes indicated by black lines.

Table 6-7 Directions taken in order to follow a particular path.

Start	**End**	**Directions**
Conveyor	Lathe	Side-left
	RM	Side-left and Forward
Lathe	Conveyor	Side-right
	RM	Side-right and Forward
RM	Lathe	Reverse and Side-left
	Conveyor	Reverse and Side-right

6.3.3 Motion of the AGV

The motion of AGV on the six paths is described below:

a. Motion from conveyor to lathe

The motion of the AGV is illustrated in figure 6-15 by green arrows.

1. The AGV was docked at the docking station at the conveyor. The docking station was located under the conveyor with the docking mechanism in adequate distance for docking. (The docking station is illustrated in figure 7-1.) Docking stations are indicated by brown rectangles with black protrusions in figures. Charging stations are indicated by brown rectangles.
2. The AGV followed the line in a side-left direction toward the lathe.
3. The AGV docked at the lathe on compressing limit switch 1.

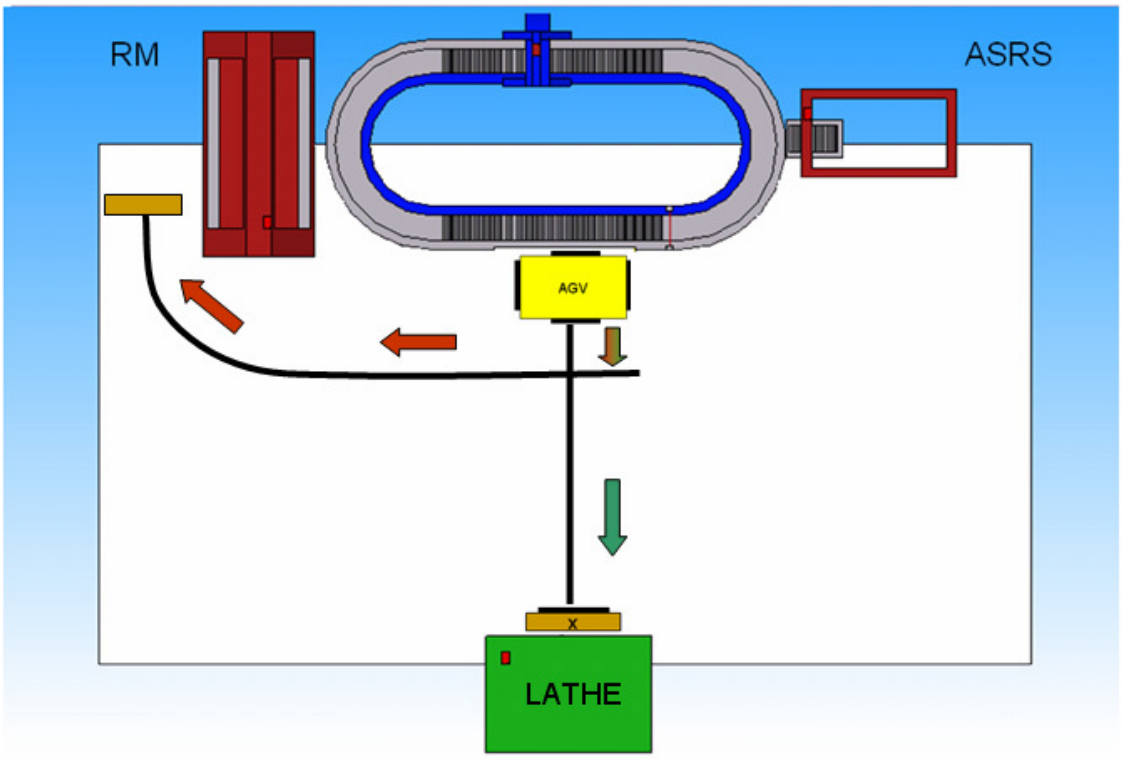


Figure 6-15 Motion of the AGV from the conveyor to lathe is indicated by green arrows. Motion of the AGV from the conveyor to RM is indicated by red arrows. The green and red arrow represents a common path.

b. Motion from conveyor to RM

The motion of the AGV is illustrated in figure 6-15 by red arrows.

1. The AGV was docked at the docking station at the conveyor.
2. The AGV followed the line in a side-left direction until the cross line was detected.
3. On detecting the cross line, the AGV changed direction and moved in a forward direction toward the RM.
4. The AGV charged at the charging station at the RM by compressing limit switches 8 and 9.

c. Motion from lathe to conveyor

The motion of the AGV is illustrated in figure 6-16 by grey arrows.

1. The AGV was docked at the docking station at the lathe.
2. The AGV followed the line in a side-right direction toward the conveyor.
3. The AGV docked at the conveyor on compressing limit switch 0.

d. Motion from lathe to RM

The motion of the AGV is illustrated in figure 6-16 by red arrows.

1. The AGV was docked at the docking station at the lathe.
2. The AGV followed the line in a side-right direction until the cross line was detected.
3. On detecting the cross line, the AGV changed direction and moved in a forward direction toward the RM.
4. The AGV charged at the charging station at the RM by compressing limit switches 8 and 9.

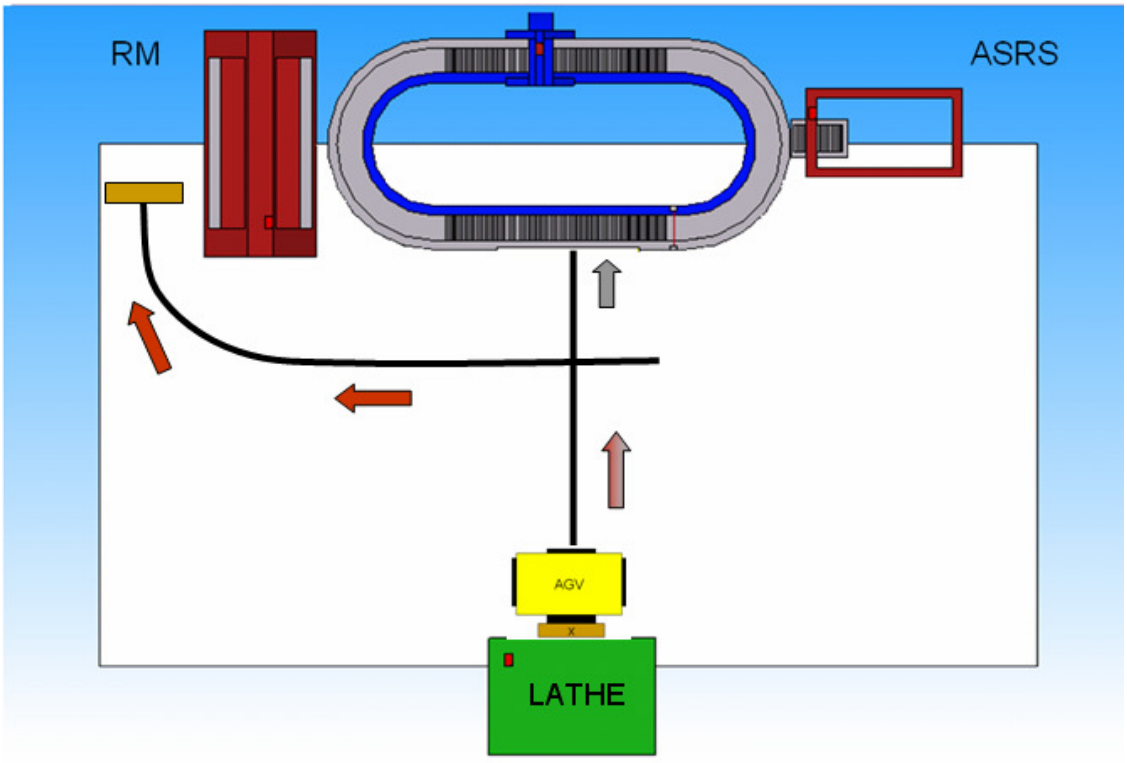


Figure 6-16 Motion of the AGV from the lathe to conveyor is indicated by grey arrows. Motion of the AGV from the lathe to RM is indicated by red arrows. The grey and red arrow represents a common path.

e. Motion from RM to lathe

The motion of the AGV is illustrated in figure 6-17 by green arrows.

1. The AGV was docked at the charging station at the RM.
2. The AGV followed the line in a reverse direction until the cross line was detected.
3. On detecting the cross line, the AGV changed direction and moved in a side-left direction toward the lathe.
4. The AGV docked at the lathe on compressing limit switch 1.

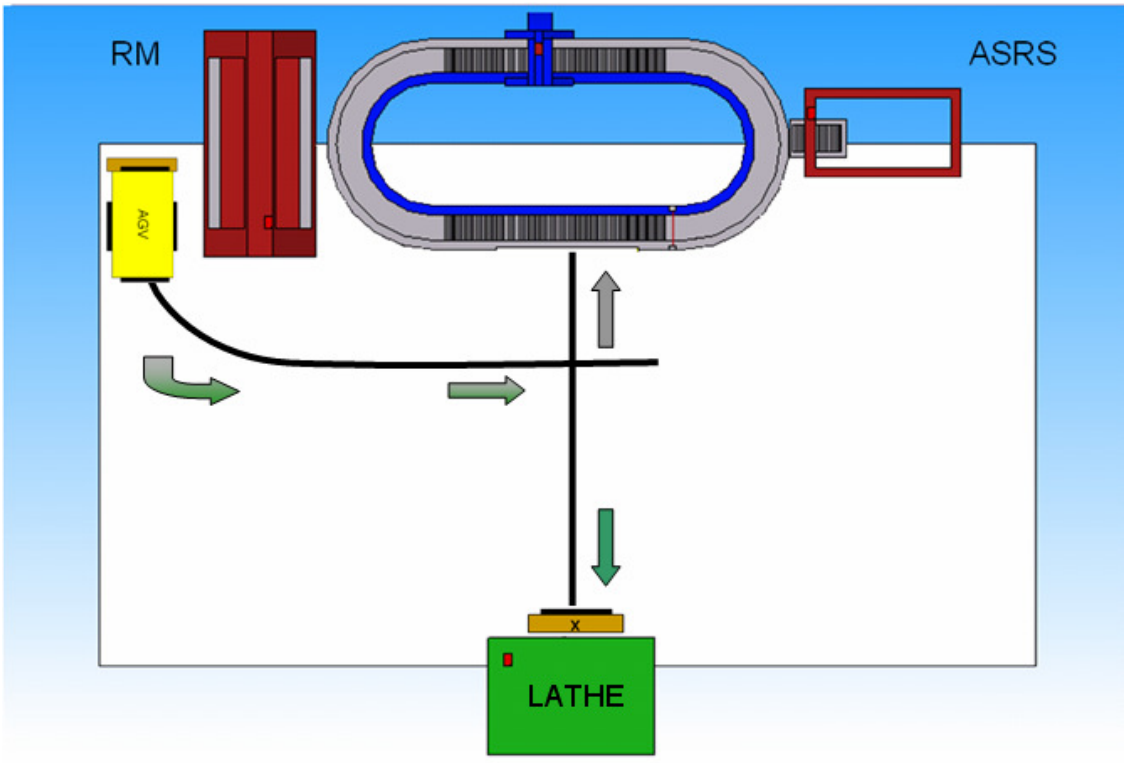


Figure 6-17 Motion of the AGV from RM to lathe is indicated by green arrows. Motion of the AGV from RM to conveyor is indicated by grey arrows. The green and grey arrows represent a common path.

f. Motion from RM to conveyor

The motion of the AGV is illustrated in figure 6-17 by grey arrows.

1. The AGV was docked at the charging station at the RM.
2. The AGV followed the line in a reverse direction until the cross line was detected.
3. On detecting the cross line, the AGV changed direction and moved in a side-right direction toward the conveyor.
4. The AGV docked at the conveyor on compressing limit switch 0.

The AGV's materials handling platform allowed it to interface with the conveyor and execute automatic pallet transfer. However, at the lathe and RM manual transfer occurred, i.e. the AGV was programmed to stop at the lathe and RM and wait for a human operator to remove or replace the pallet. This occurred as there was no conveyor table to interface with at these work stations but only

a docking station. The charging station was located at the RM. This allowed the AGV to charge its batteries while manual pallet transfer occurred at the RM or when the AGV was not being used.

6.4 Light Following Technique

6.4.1 Hardware

Light following can be utilised when floor conditions are unsatisfactory for line following such as:

1. If the tape had worn out over time and had not been replaced due to poor maintenance.
2. If the floor had become dirty making it difficult for the line sensors to provide distinguishable readings between black and white.
3. If there was an oil spill on the floor.

The light sensor boards were identical to the line sensor boards (section 6.3.1) but addressed as LITEL and LITER instead of LINEL and LINER. Two ORP12 LDRs were positioned in wedges on all four sides of the AGV, eight altogether (refer to figures 6-18 and 6-19). By positioning the sensor in the wedge, light could be detected at an angle. The LDR resistance decreased when exposed to light therefore the closer the light was, the lower the resistance. The difference in resistance resulted in a difference in the readings obtained in HyperTerminal. By comparing the readings one could determine how far or near the AGV was to a light source (refer to table 6-8).

A 20W halogen light was used for light following in a constant environment, as it provided suitable readings for programming (refer to table 7-7, 20W column). Light readings were detected within a 5m range. The readings were used to program the AGV to follow the strongest source of light. The light was turned on at the docking station when required for a specific route (refer to figure 6-20).

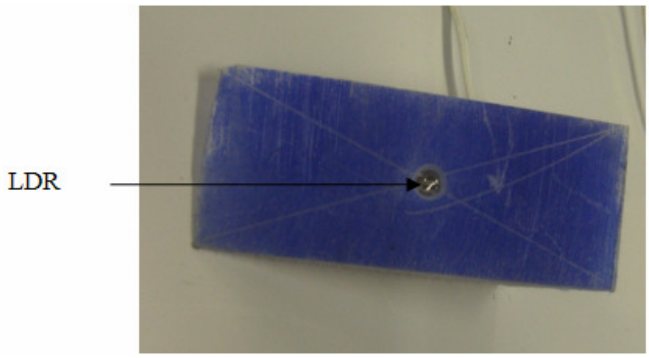


Figure 6-18 One of the wedges in which the light sensors were inserted.

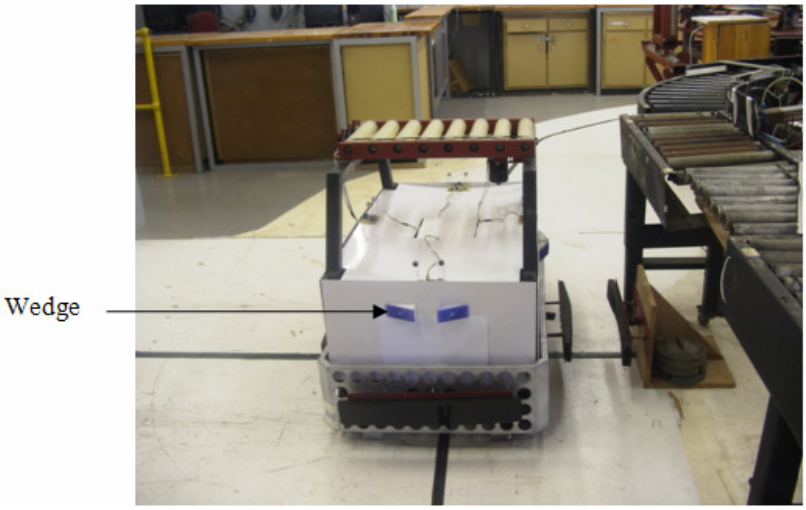


Figure 6-19 AGV with wedges in which light sensors were inserted.

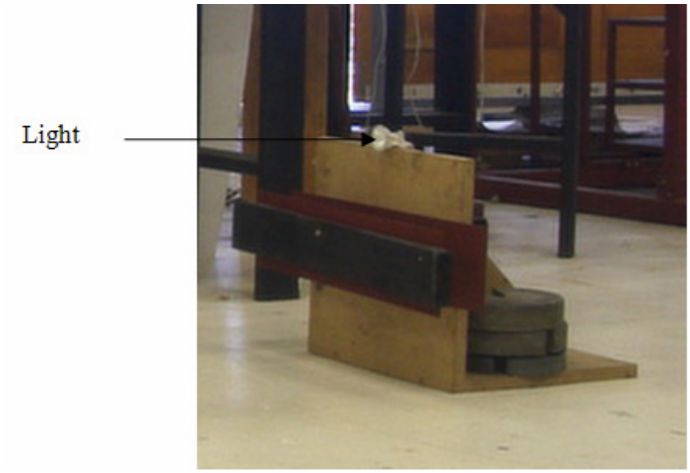


Figure 6-20 Light on docking station.

6.4.2 Program

To activate the light sensor board the command "#%+LITEL *\" was sent. Readings obtained in HyperTerminal are indicated in table 6-8. A reading below 900 indicated that the light was greater than 2m away. A reading above 900 indicated that the light was less than 2m away. A reading of 1000 indicated that the light was less than 0.5m away.

In figure 6-21, LITR0, LITR1, LITR2 and LITR3 refer to LDRs 0, 1, 2 and 3 respectively. LITR1 and LITR2 are greater than 2m away from the light therefore the reading is below 900. LITR0 and LITR3 are not connected.

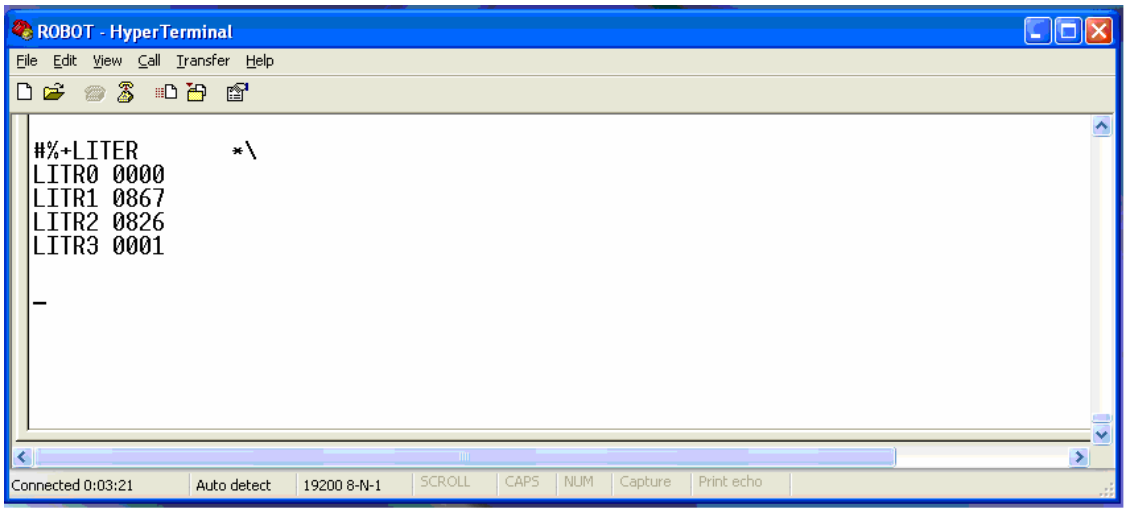


Figure 6-21 Light sensor feedback in HyperTerminal

Table 6-8 Readings in HyperTerminal and C

Light Position	HyperTerminal	C	Logic
> 2m	< 900	< 9	1
<= 2m	>= 900	>= 9	0
<= 0.5 m	>= 1000	>= 10	2

Table 6-9 Direction of sensors

Direction	LDRs	
	1	**2**
Forward	f2	f3
Reverse	b3	b2
Side-left	l1	l0
Side-right	r0	r1

Data was returned in the C program in the form of an array. Specific elements were accessed in the array and analyzed to direct the AGV towards the light and hence the work station. If for example, array element 22 had a value of '9', it meant the AGV was 2m away from the light and the AGV proceeded in the direction of the light. If array element 21 had a value of '1' and array element 22 had a value '0', it meant the AGV was approximately 0.5m away from the light and the AGV stopped for routes between the RM and lathe. Table 6-8 indicates the readings used in C. The logic values are used in tables 6-10 to 6-15. Table 6-9 indicates the direction of the eight sensors used. Figure 6-22 indicates the position of the eight sensors used.

Tables 6-10 to 6-15 indicate readings that occurred, the translation, and the action taken in the forward, reverse and lateral directions.

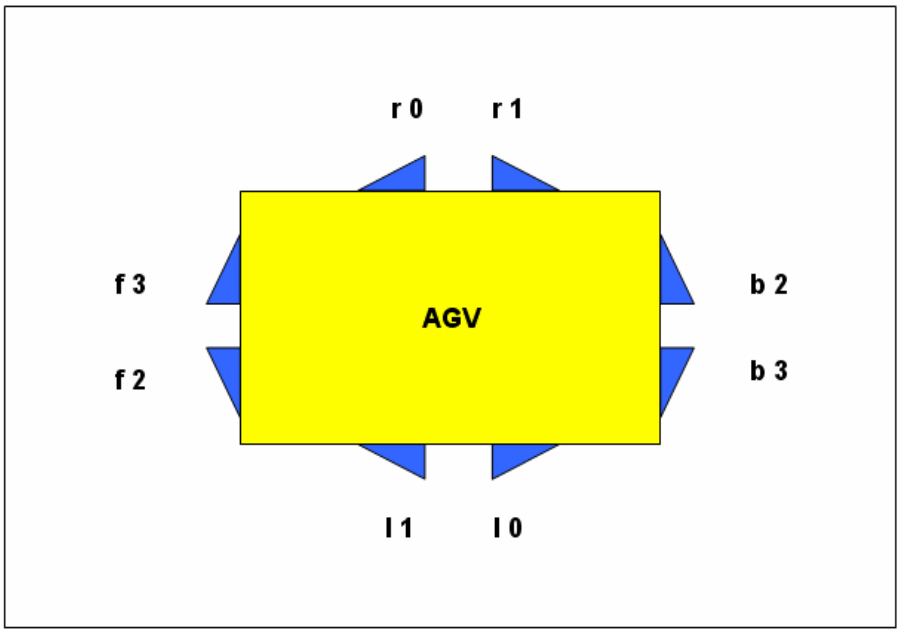


Figure 6-22 Position of sensors used. The sensors are inserted in the blue wedges.

Table 6-10 Translation of readings and action for light following in the forward direction.

Forward direction			
LDR		**Translation**	**Action**
F2	**F3**		
1	0	More light from left direction	Left
0	1	More light from right direction	Right
1	1	Equal light	Forward
2	0	More light from left direction	Left
0	2	More light from right direction	Right
2	2	Equal light	Forward
0	0	No light. Search for light	Rotate until light detected
Note: Stops when limit switch 8 compressed i.e. when AGV docks with RM docking station			

Table 6-11 Translation of readings and action for light following in the reverse direction.

Reverse direction			
LDR		**Translation**	**Action**
B2	**B3**		
1	0	More light from reverse left direction	Reverse left
0	1	More light from reverse right direction	Reverse right
1	1	Equal light	Reverse
2	0	More light from reverse left direction	Reverse left
0	2	More light from reverse right direction	Reverse right
2	2	Equal light	Reverse
0	0	No light. Search for light	Rotate until light detected
Note: Stops when limit switch 9 compressed i.e. when AGV docks with conveyor docking station			

Table 6-12 Translation of readings and action for light following in the side-right direction.

Side-right direction			
LDR		**Translation**	**Action**
R0	**R1**		
1	0	More light from side-right 315 direction	Side-right 315
0	1	More light from side-right 45 direction	Side-right 45
1	1	Equal light	Side-right
2	0	More light from side-right 315 direction	Side-right 315
0	2	More light from side-right 45 direction	Side-right 45
2	2	Equal light	Side-right
0	0	No light. Search for light	Rotate until light detected
Note: Stops when limit switch compressed i.e. when AGV docks with conveyor docking station			

Table 6-13 Translation of readings and action for light following in the side-left direction.

Side-left direction			
LDR		**Translation**	**Action**
R0	**R1**		
1	0	More light from side-left 135 direction	Side-left 135
0	1	More light from side-left 225 direction	Side-left 225
1	1	Equal light	Side-left
2	0	More light from side-left 135 direction	Side-left 135
0	2	More light from side-left 225 direction	Side-left 225
2	2	Equal light	Side-left
0	0	No light. Search for light	Rotate until light detected
Note: Stops when limit switch compressed i.e. when AGV docks with lathe docking station			

Table 6-14 Translation of readings and action for light following from the RM to lathe.

RM to Lathe					
LDR				**Translation**	**Action**
L0	**L1**	**B2**	**B3**		
1	0	0	0	More light from side-left 135 direction	Side-left 135
0	1	0	0	More light from side-left 225 direction	Side-left 225
1	1	0	0	Equal light	Side-left
0	0	1	0	More light from reverse left direction	Reverse left
0	0	0	1	More light from reverse right direction	Reverse right
0	0	1	1	Equal light	Reverse
1	0	0	1	More light from side-left 135 direction	Side-left 135
1	1	0	1		
1	1	1	1		
1	0	1	1	More light from reverse right direction	Reverse right
2	0	0	0	0.5 m away from the light	Stop
0	2	0	0		
0	0	2	0		
0	0	0	2		

Table 6-15 Translation of readings and action for light following from the lathe to RM.

Lathe to RM					
LDR				**Translation**	**Action**
R0	**R1**	**F2**	**F3**		
1	0	0	0	More light from side-right 315 direction	Side-right 315
0	1	0	0	More light from side-right 45 direction	Side-right 45
1	1	0	0	Equal light from side-right direction	Side-right
0	0	1	0	More light from left direction	Left
0	0	0	1	More light from right direction	Right
0	0	1	1	Equal light forward direction	Forward
1	0	0	1	More light from side-right 315 direction	Side-right 315
1	1	0	1		
1	1	1	1		
1	0	1	1	More light from right direction	Right
2	0	0	0	0.5 m away from the light	Stop
0	2	0	0		
0	0	2	0		
0	0	0	2		

The control program continually corrected the direction of the AGV such that the AGV proceeded toward the light. Limit switches were integrated as in table 6-19. Ultrasonic sensors were initially integrated as in table 6-17, and changed after conducting high noise level testing to the allocation in table 9-1, to accommodate for the high noise levels. Programming can be found in Appendix E.1.4.

The AGV transferred pallets between the conveyor, lathe and RM by following the light as illustrated in figure 6-23. The AGV proceeded in a linear direction between the conveyor and lathe, and conveyor and RM, and in a curve between the lathe and RM. The AGV was programmed to dock in the lateral direction when it moved between the conveyor and lathe, as illustrated in figure 9-17. Docking between the RM and conveyor occurred in the longitudinal direction, as illustrated in figure 9-18. When the AGV moved between the lathe and the RM, the AGV was programmed to stop at a distance of 0.5m from the light, at the docking station. (Docking between the lathe and the RM is explained further under preliminary testing, section 7.2.2.) No charging occurred at the RM as limit switches 8 and 9 were assigned for docking in the longitudinal direction (refer to section 6.6).

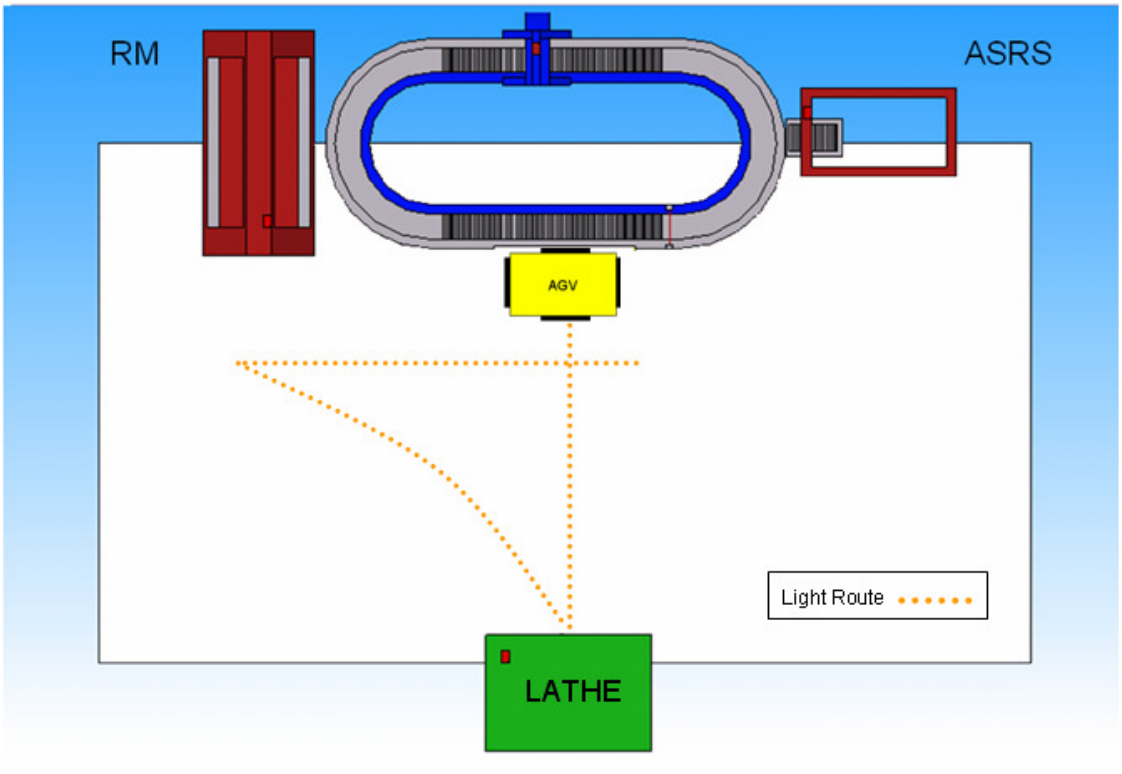


Figure 6-23 Light routes indicated by orange dotted lines.

6.4.3 Motion of the AGV

Motion of the AGV between the conveyor and lathe is the same as described in the line following section 6.3.3 (a) and (c), except the AGV is following the light instead of the line. The light is turned on at the lathe docking station, for motion from the conveyor to lathe, and at the conveyor docking station, for motion from the lathe to conveyor. The light is indicated by an 'x' on the docking station in figure 6-24.

a. Motion from conveyor to RM

The motion of the AGV is illustrated in figure 6-25 by the red arrow.

1. The AGV was docked at the docking station at the conveyor.
2. The AGV proceeded in a forward direction toward the light at the docking station at the RM.
3. The AGV docked at the docking station at the RM on compressing limit switch 8.
4. Manual pallet transfer occurred.

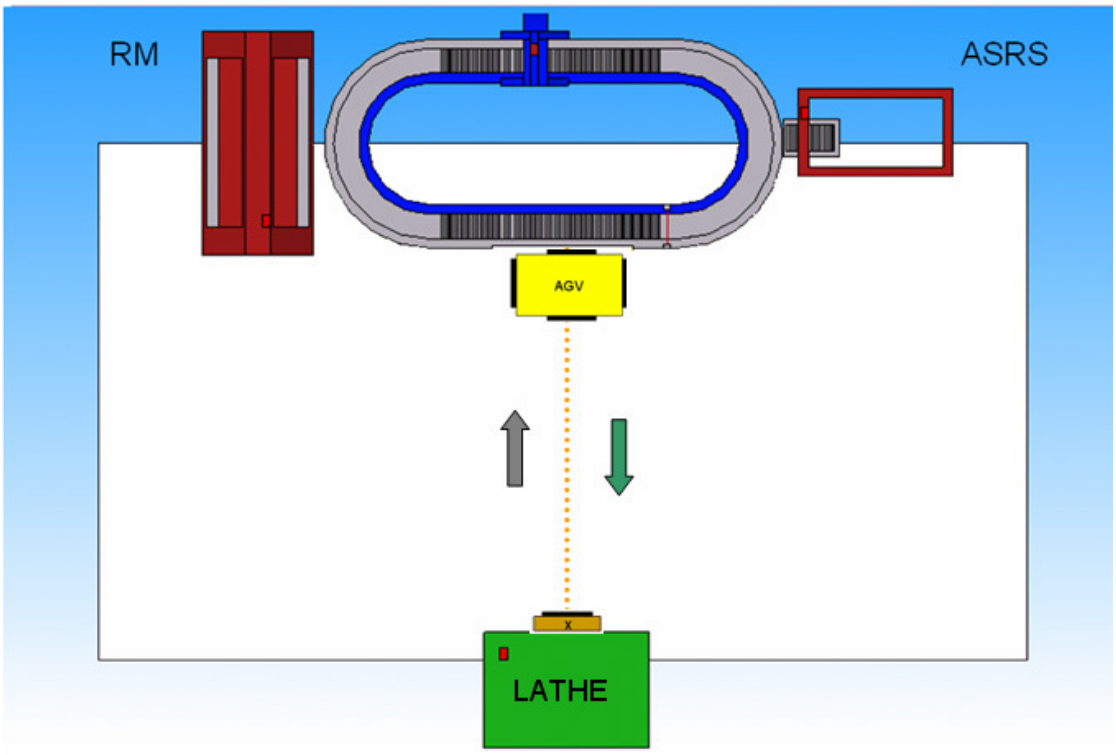


Figure 6-24 Motion of the AGV from the conveyor to lathe is indicated by the green arrow. Motion of the AGV from the lathe to conveyor is indicated by the grey arrow.

b. Motion from RM to conveyor

The motion of the AGV is illustrated in figure 6-25 by the grey arrow.

1. The AGV was docked at the docking station at the RM.
2. The AGV proceeded in a reverse direction toward the light at the docking station at the conveyor.
3. The AGV docked at the docking station at the conveyor on compressing limit switch 9.
4. Manual pallet transfer occurred.

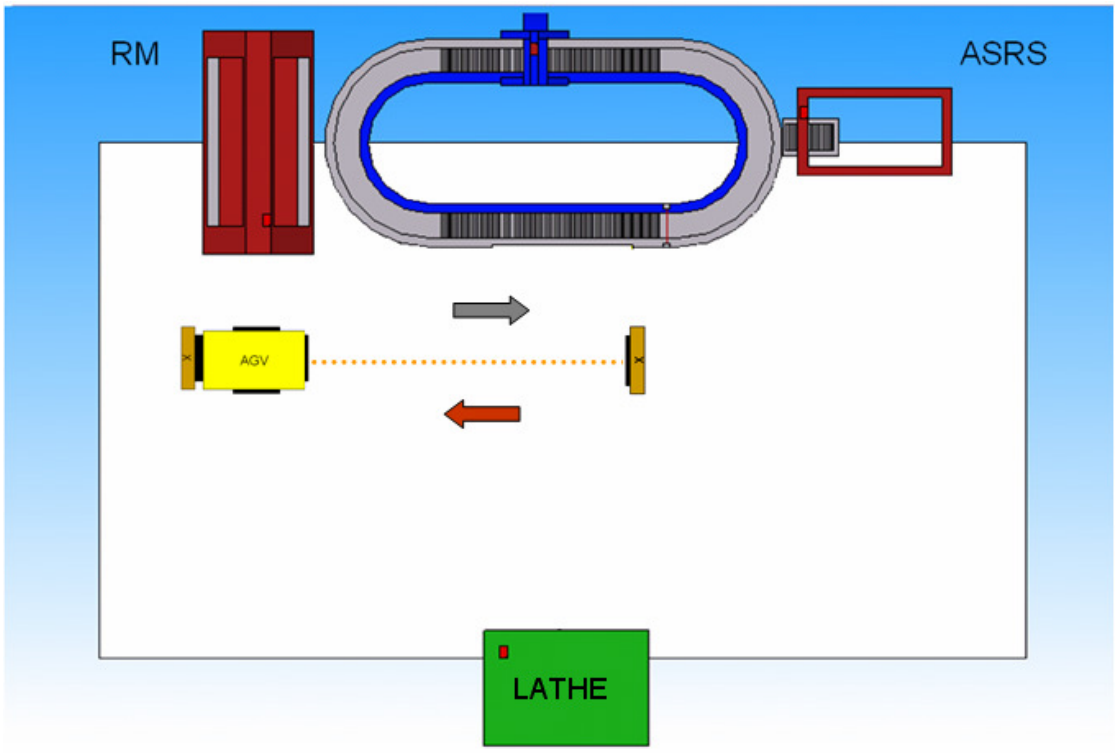


Figure 6-25 Motion of the AGV from the RM to conveyor is indicated by the grey arrow. Motion of the AGV from the conveyor to RM is indicated by the red arrow.

c. Motion from lathe to RM

The docking station was positioned at a 45 degree angle to increase light exposure for the light sensors. The motion of the AGV is illustrated in figure 6-26 by the red arrow.

1. The AGV was docked at the docking station at the lathe.
2. The AGV proceeded in a resultant side-right-315˚ direction toward the light at the docking station at the RM. The AGV corrected itself such that it approached the light.
3. The AGV stopped 0.5m in front of the light as programmed.

d. Motion from RM to lathe

The motion of the AGV is illustrated in figure 6-26 by the green arrow.

1. The AGV was docked at the charging station at the RM.

2. The AGV proceeded in a resultant side-left-135˚ direction toward the light at the docking station at the lathe. The AGV corrected itself such that it approached the light.
3. The AGV stopped 0.5m in front of the light as programmed.

Motion for (c) and (d) is elaborated on in section 7.2.2

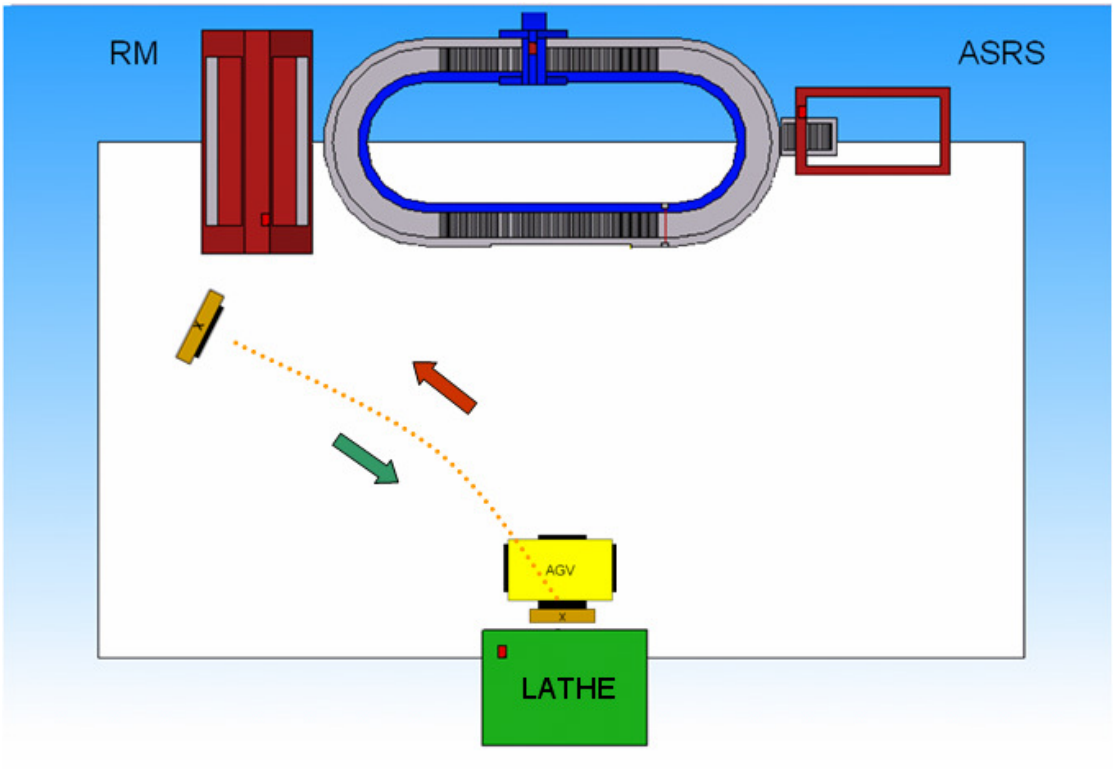


Figure 6-26 Motion of the AGV from RM to lathe is indicated by the green arrow. Motion of the AGV from lathe to RM is indicated by the red arrow.

6.5 Ultrasonic sensor integration

Ultrasonic sensors were applied to avoid obstacles and hence collisions. This protected the AGV as well as the machines, other mobile robots and humans in the laboratory. Four ultrasonic boards were used. Figure 6-27 illustrates the ultrasonic sensor board. Each board had a PIC16F88-I/P microcontroller and was designed for an ultrasonic transmitter and receiver. A T40-16 848S ultrasonic transmitter and R40-16 931S receiver were used. Objects were detected within a range of 0-3m. A transmitter and receiver were positioned on all four sides of the AGV as illustrated in figure 6-28.

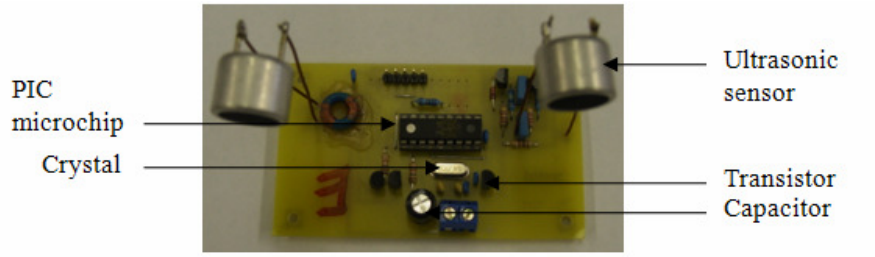


Figure 6-27 Ultrasonic sensor board

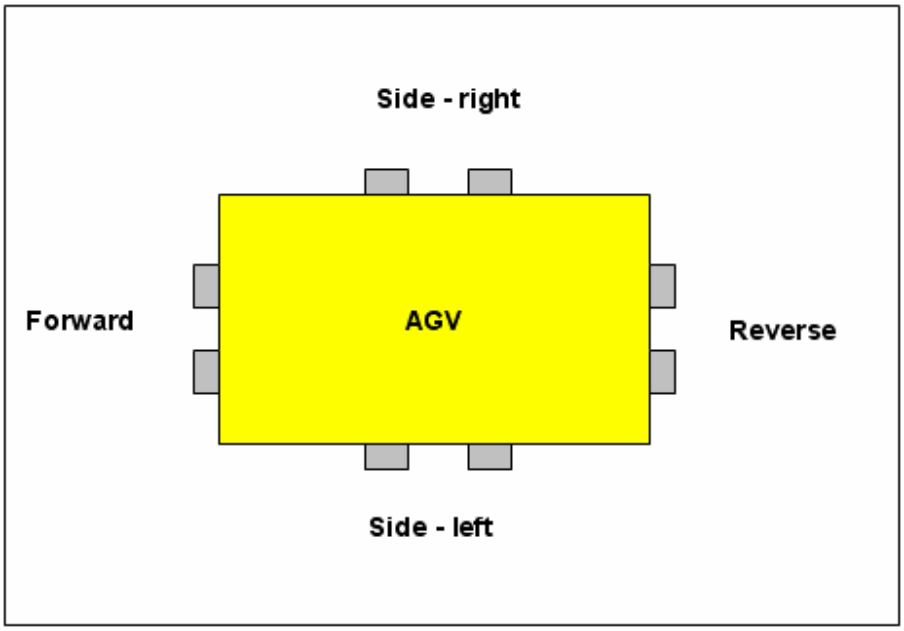


Figure 6-28 Position of ultrasonic sensors. The sensors are indicated by the grey rectangles.

To activate the forward ultrasonic sensor board the command "#%+ULTSN *\" was sent.

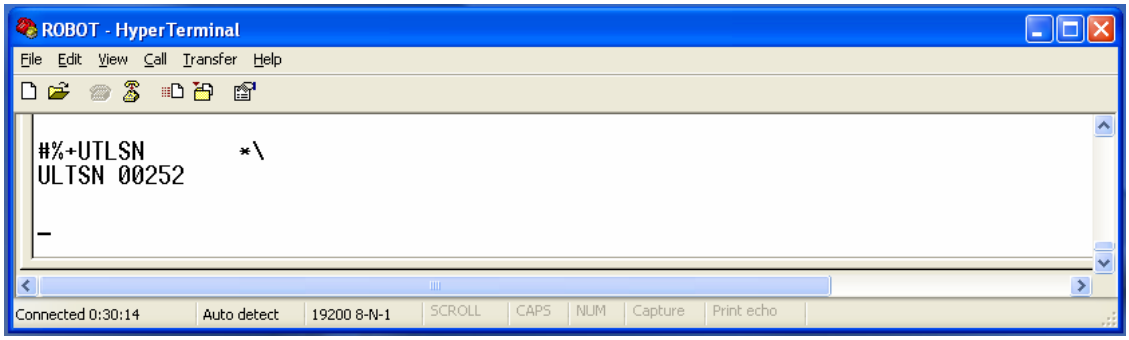


Figure 6-29 Ultrasonic sensor feedback in HyperTerminal

Feedback was received as illustrated in figure 6-29. The distance of the object was returned in centimeters. The object is 252cm or 2.52m meters away. If an object was out of the 0-3m range an error was received. The other three boards were addressed and allocated as in table 6-16.

The ultrasonic sensors were integrated with both the line following and light following techniques so that the AGV could complete its tasks without collision. The table 6-17 indicates how the AGV reacted to an object at a particular distance.

Table 6-16 Ultrasonic board location and address.

Direction	**Board address**
Forward	ULTSN
Reverse	ULTSS
Side - left	ULTSW
Side - right	ULTSE

Table 6-17 Distance of object and Action taken

Distance of object	**Action**
0m	Assume crashed! Stop.
< 1m	Stop. Proceed when clear.
1-2m	Rotate until clear space found.
> 2m	Proceed forward

If the object was 0m away the program assumed that the AGV had crashed and stopped the AGV. If the object was less than 1m away the AGV stopped. The program checked if the path was clear and the AGV proceeded forward. If the object was within a distance of 1-2m, the AGV rotated until a clear space was found and proceeded in that direction. If there were no objects within a distance of 2m or greater, i.e. an object free zone, the AGV continued forward.

6.6 Limit Switch Integration

Limit switches were compressed in the event that the AGV crashed into an object, for charging and for docking purposes. Figure 6-30 illustrates the limit switch board. The board had a PIC16F88-I/P microcontroller and allowed for ten limit switch inputs.

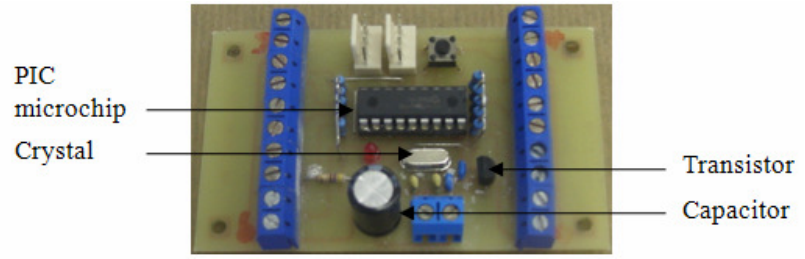


Figure 6-30 Limit switch board

To activate the limit switch board the command "#%+SWTCH *\ " was sent.

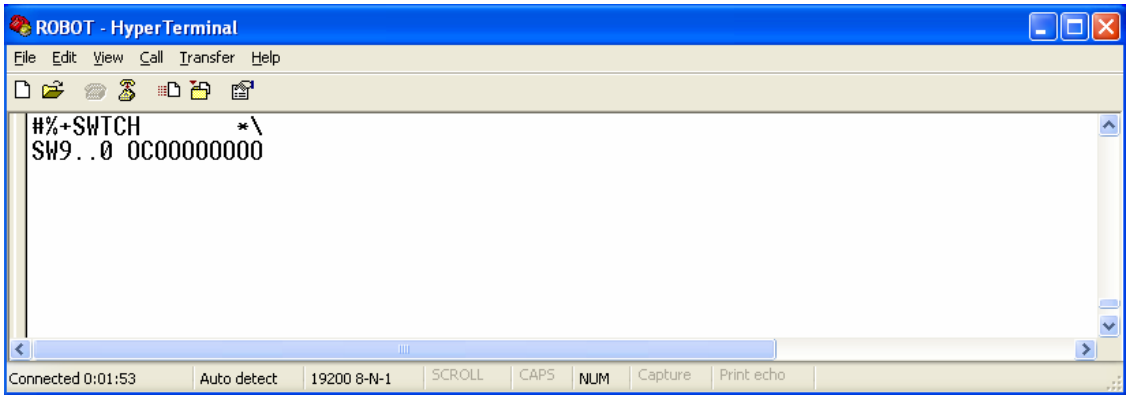


Figure 6-31 HyperTerminal feedback

Feedback was received as indicated in figure 6-31. A 'C' indicated that the switch was closed and a 'O' indicated that the switch was open. Limit switches were allocated as indicated in table 6-18.

Switch 0 was located behind the docking mechanism of the AGV that docked with the conveyor, as illustrated in figure 6-32. If the user instructed the AGV to transfer a pallet from the lathe to the conveyor using line following, the AGV moved toward the conveyor in a side-right direction. The control program continually corrected the AGV to ensure that it remained on the line. When the AGV's male docking mechanism made contact with the female docking mechanism on the conveyor docking station, switch 0 was compressed. The user was informed that the conveyor docking sequence had been activated. The AGV stopped and started the materials handling platform.

When the pallet reached the end of the materials handling platform, switches 2-5 were compressed (refer to figure 6-33). The platform stopped rotating and the control program sent a message to the user to indicate the pallet had been loaded.

Table 6-18 Limit switch allocation for line following.

Limit switch	Allocation
0	Docking at conveyor
1	Docking at lathe
2	Materials handling platform
3	
4	
5	
6	Crash
7	
8	Charging
9	

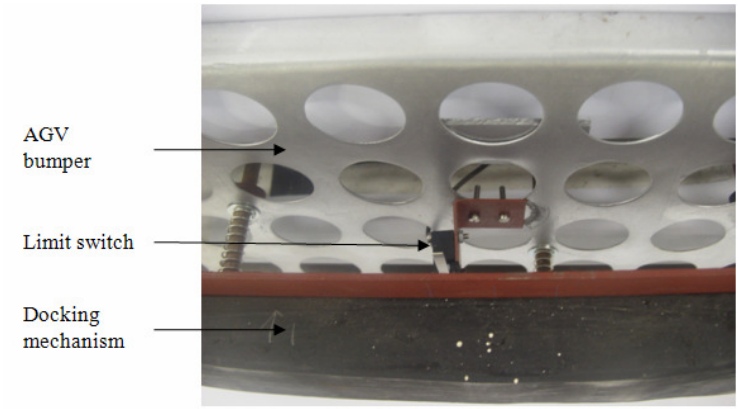


Figure 6-32 Switch 0 located behind the docking mechanism of the AGV

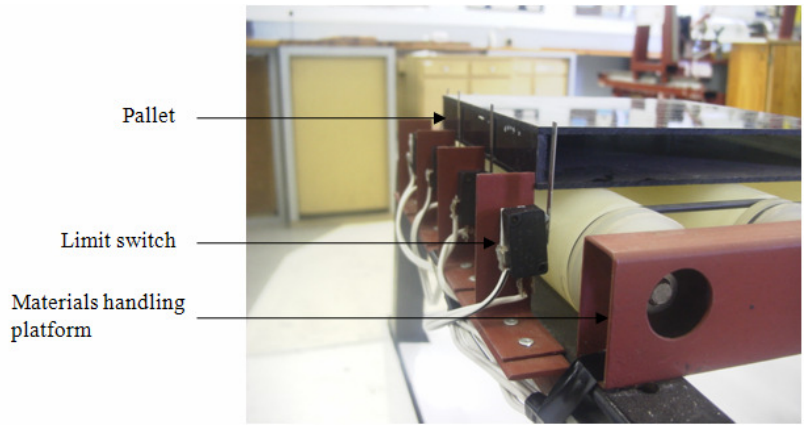


Figure 6-33 Switches 2-5 on materials handling platform

The same sequence occurred if light following was used instead of line following. The control program corrected the AGV such that it moved toward the light positioned on the conveyor docking station until switch 0 was compressed. The piston mechanism was activated and pushed the pallet on the materials handling platform.

Switch 1 was located behind the docking mechanism of the AGV that docked with the lathe (similar positioning to figure 6-32 but opposite side of AGV). If the user instructed the AGV to transfer a pallet from the conveyor to lathe using line following or light following, the AGV moved toward the lathe in a side-left direction.

When the AGV's docking mechanism made contact with the lathe docking station, switch 1 was compressed. The user was informed that the lathe docking sequence had been activated. The AGV stopped and awaited manual transfer, i.e. a human operator removed or reloaded the pallet as there was no conveyor table to interface with.

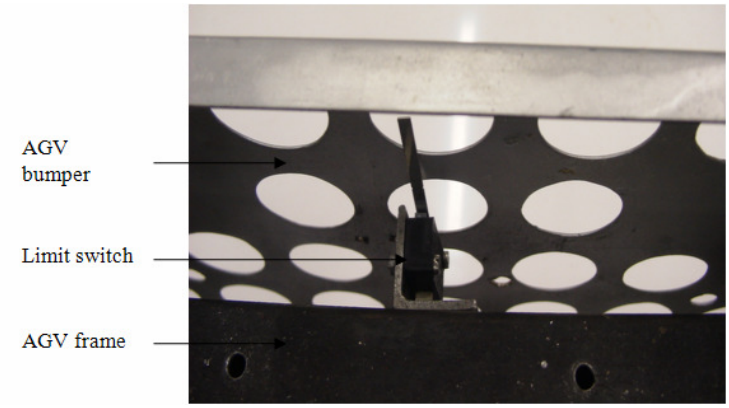


Figure 6-34 Switch 6 located behind bumper

Switches 6 and 7 were positioned behind the bumper in the forward and reverse directions (refer to figure 6-34). The bumper could only be compressed in the forward and reverse directions therefore it was unnecessary to locate switches in the side-right and side-left directions. When the bumper was compressed from the forward direction switch 6 was compressed. The AGV made an emergency stop. The user was informed that the AGV had crashed into an object. The same occurred if switch 7 was compressed by the bumper in the reverse direction.

Switches 8 and 9 were positioned behind the charging contacts. When the AGV contacts made contact with the charging station contacts the switches 8 and 9 were compressed and the AGV's batteries charged. Charging occurred only in the line following technique. When light following

there was a chance that incorrect contact might occur i.e. contact between positive and negative contacts.

Switches 8 and 9 were later allocated on the bumper, behind the longitudinal docking mechanisms, as illustrated in figure 6-32. This allowed docking between the conveyor and the RM when light following. The limit switch board was limited to only ten inputs. Future work could include a board that accommodated more inputs.

The allocation in table 6-19 was used in the new line following, light following, line variation and light variation programs (section 8.3 and 8.4).

Table 6-19 Limit switch allocation for light following

Limit switch	**Allocation**
0	Docking at conveyor
1	Docking at lathe
2	Materials handling platform
3	
4	
5	
6	Crash
7	
8	Docking at conveyor
9	Docking at RM

6.7 Summary

This chapter described the preliminary program for the AGV when line following and light following under constant conditions. The setup of the comm. ports and the read and write functions using the Brainstem C API was explained. The user interface was described. The electronic hardware and sensor positions for line following and light following techniques were explained.

The programs for the two techniques were described. The response of the AGV to a specific set of sensor readings was presented in tables for the directions/routes for each technique. The movement of the AGV on the six routes in the constant environment was explained for each technique.

The limit switch board, ultrasonic boards and respective sensors were described. The allocation of limit switches for a specific task for each technique was explained. The integration of the ultrasonic sensors was discussed. The action of the AGV at a particular distance of an obstacle was explained.

7 Preliminary Testing

Preliminary testing of the line following and light following techniques was conducted in a constant environment. The line was always present for line following and the light had the same light intensity for light following throughout the AGV's travel. Testing determined the accuracy and reliability of the individual techniques and how the two techniques could be combined to compensate for an unpredictable environment.

Testing was also conducted under high noise levels to determine the robustness of the ultrasonic sensors in a high noise level environment and, under different light intensities to determine the light sensor range in the light intensity environment. The data derived from preliminary testing was used for the main software, Chapter 8.

7.1 Line Testing

In order to test the line following ability of the software and hardware, the AGV was tested on a linear path and a non – linear path. Figure 6-14 illustrates the paths.

7.1.1 Linear Path: Lathe to Conveyor

The linear path ran between the conveyor and the lathe and was 2.6 m in length. The ability of the AGV to follow a line laterally and dock at the conveyor was tested. The AGV was instructed to pick-up a pallet from the conveyor by following the lathe to conveyor route.

The AGV was positioned at the lathe. The AGV proceeded to the conveyor in a side-right direction as it followed the line. The AGV docked with the docking station at the conveyor on compressing limit switch 0 (refer to figure 7-1). The control program activated the materials handling platform which rotated backwards (i.e. away from the conveyor) in order to load the pallet. The piston mechanism pushed the pallet onto the materials handling platform. The pallet was drawn towards limit switches 2 to 5 as the platform rotated. When the pallet was loaded (i.e. the limit switches were compressed) the platform stopped rotating. The program informed the user that the pallet had been loaded and awaited further instructions. Table 7-1 indicates timing results to complete the lathe to conveyor route in the lateral direction. Three readings were recorded and the average calculated.

Table 7-1 Time to complete the path between the lathe and conveyor.

Test	Time (min)
1	3.3
2	3.6
3	3.4
Average	3.4

The AGV followed the line with negligible deviation in the lateral directions i.e. side-right and side-left. The AGV was tested ten times on the lathe to conveyor route and strayed off the line two times. The AGV was 80% reliable in following the straight line in the lateral direction and had a 20% chance of straying off the line. The AGV was tested ten times on the same path in the forward direction and strayed off the line one time. The AGV was 90% reliable in following the straight line in the forward direction and had a 10% chance of straying off the line. The software automatically stopped the AGV when it strayed off the line i.e. when the line was not detected.

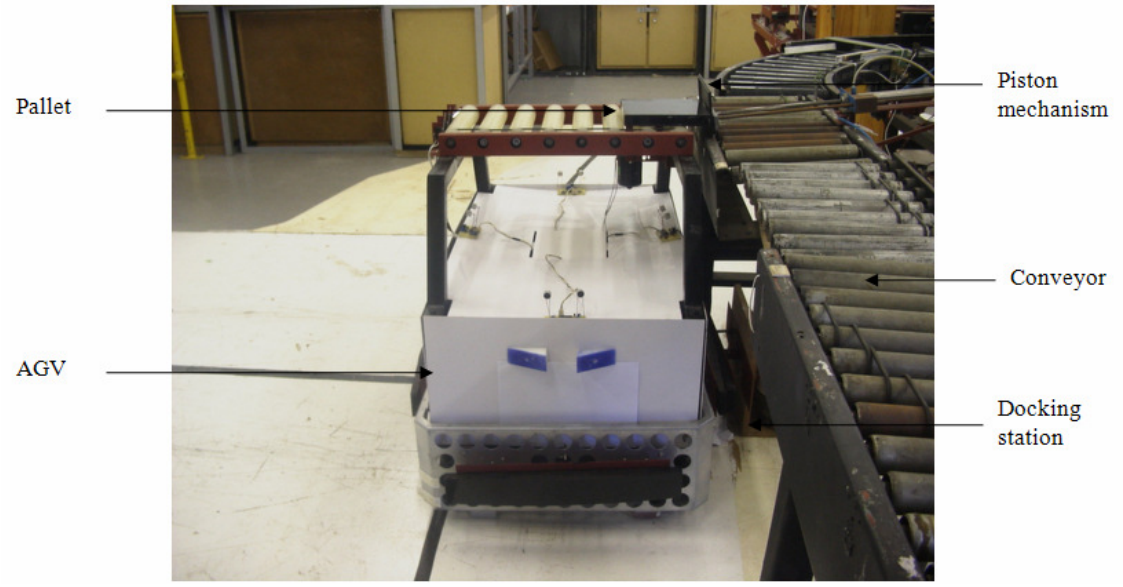


Figure 7-1 AGV docked at conveyor, pallet being loaded, docking station under conveyor.

7.1.2 Non - Linear Path: Lathe to RM

The non – linear path started at the linear path and ended at the charging station positioned at the RM. The ability of the AGV to follow the curved line in the forward direction and dock at the charging station was tested. The ability of the AGV to change from the side-right direction to

forward at the decision point or junction was also tested. The AGV was instructed to transport a pallet from the lathe to the RM by following the lathe to RM route.

The AGV moved in the side-right direction toward the conveyor. When the AGV came to the junction the forward sensors detected the line in the forward direction. The AGV stopped moving in the side-right direction and changed to the forward direction. The AGV continued along the curved path, correcting itself as it proceeded, until it docked with the charging station by compressing limit switches 8 and 9, where upon the AGV stopped for charging.

If the AGV did not detect the forward line to the RM, the AGV proceeded toward the conveyor and docked when limit switch 0 was compressed. The AGV had to be re-instructed to complete the task. The AGV was tested ten times on the lathe to RM route and missed the forward line two times. Therefore the AGV was 80% reliable in detecting the forward or change of direction line, and had a 20% chance of missing the forward line. The AGV strayed off the non-linear line two times. It was 80% reliable in following the non-linear path and had a 20% chance of straying off the line. The AGV was stopped by the program if the line was not detected.

A felt pen was positioned in the centre of the AGV in line with the centre sensor in the side-right and forward direction. The deviation from the centre was traced along the path each time the AGV was instructed to follow the path. A grid was drawn on the floor. The x and y values of the line path (theoretical) and the actual paths followed were measured off the floor. The values were taken at definable points that shaped the path and tabulated in table B-1 (Appendix B). Three graphs (figures 7-2 to 7-4) were plotted of the deviation of the centre sensor from the forward path. Minimal deviation occurred in the side-right direction and was regarded as negligible. Therefore that part of the graph is not illustrated.

The average of the three graphs was calculated and plotted in the graph illustrated in figure 7-5. The x and y values were measured at points where the three graphs intersected and at parallel sections and tabulated in Table B-2 (Appendix B).

A maximum average deviation of 6.3cm occurred at 369.5cm (indicated in figure 7-5 by a blue x). The AGV followed the path with an accuracy of 84% with an error of 16%. The AGV took an average of 6.5 minutes to complete the curved path, as indicated in table 7-2.

% Accuracy = y actual / y theoretical

= 32.8 / 39.1 x 100

= 84 %

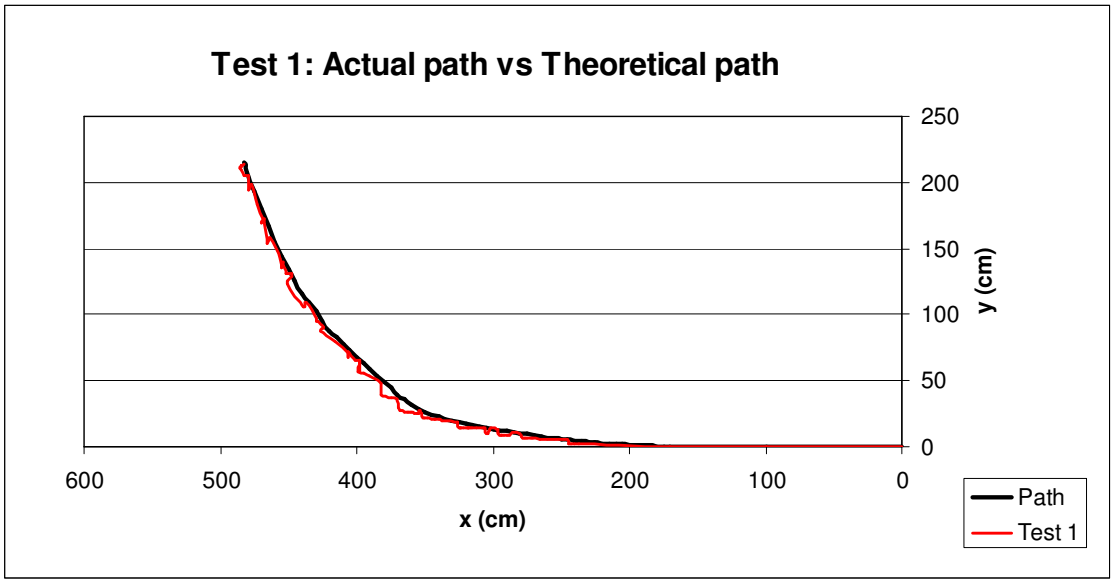


Figure 7-2 Test 1: Actual path versus theoretical path

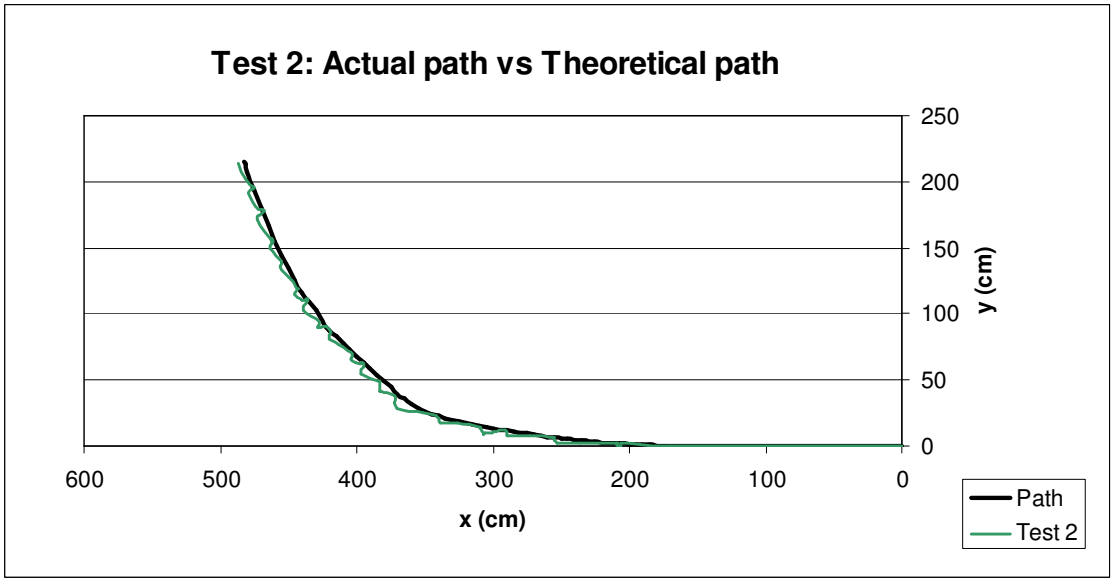


Figure 7-3 Test 2: Actual path versus theoretical path

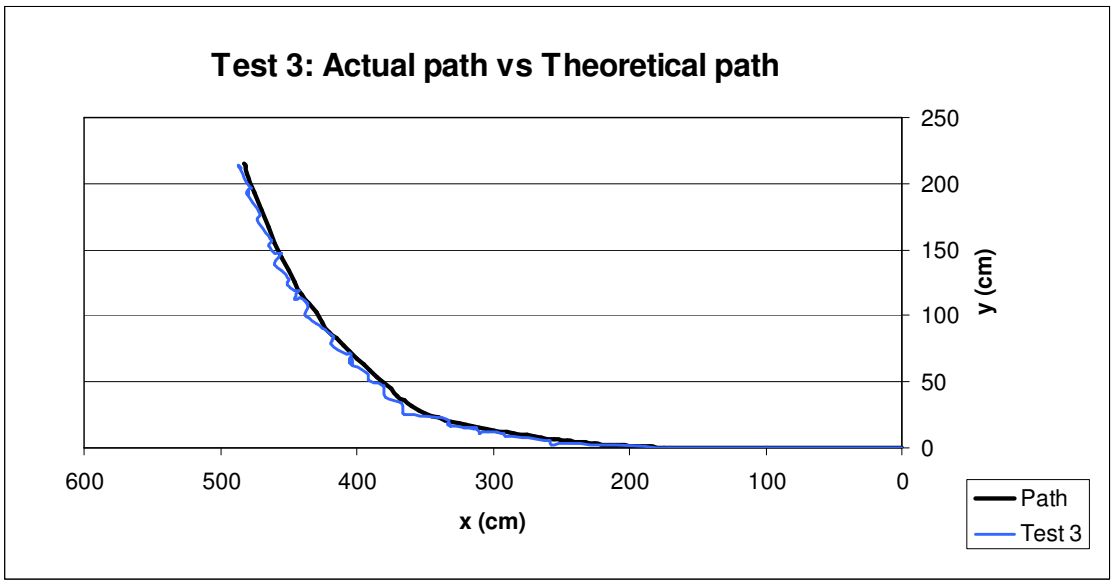


Figure 7-4 Test 3: Actual path versus theoretical path

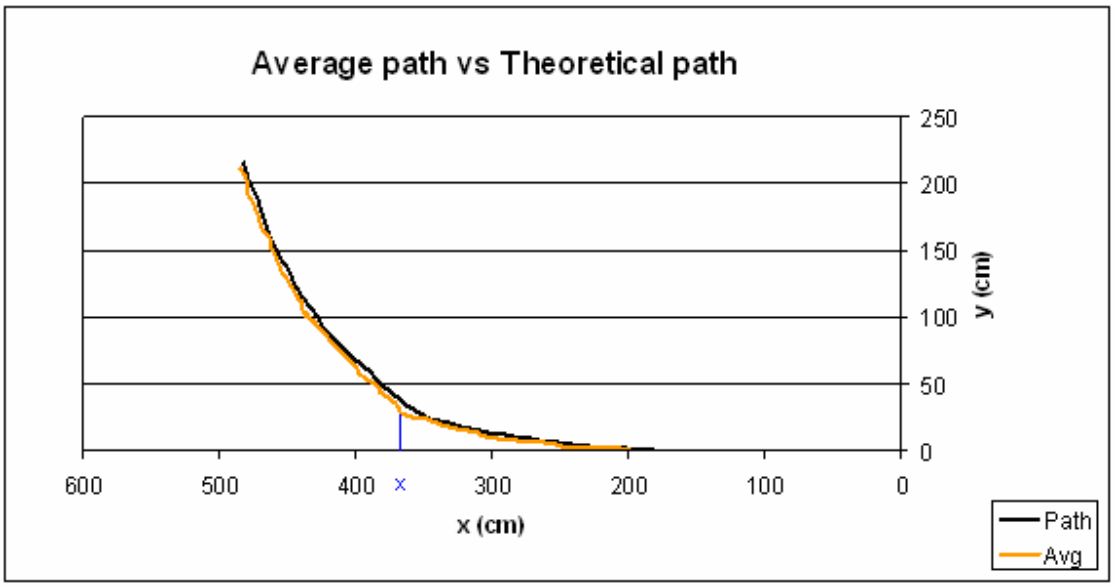


Figure 7-5 Average path versus theoretical path. 'x' indicates maximum deviation of 6.3cm at x = 369.5cm.

Table 7-2 Time to complete the path between the conveyor and the RM.

Test	Time (min)
1	6.6
2	6.2
3	6.7
Average	6.5

7.1.3 Conclusion

The use of three LDRs instead of one LDR reduced the risks of the AGV going off the line. When the centre sensor was off the line, one of the other two sensors was on the line. The control program was able to correct the AGV's motion so that the AGV did not lose the line. The line following technique proved reliable on a non – linear path. Slight movements of the felt pen may have resulted in errors in measurements of the centre sensor from the centre of the line.

The lateral movement of the AGV was not tested on the non-linear path as the AGV's material handling tasks did not require lateral motion when moving to the RM. The AGV's platform did not interface directly with the RM. Manual transfer occurred as there was no conveyor-type platform to interface with (mentioned in section 6.3.3).

7.2 Light Testing

In order to test the light following ability of the software and hardware, the AGV was tested on two linear paths and a non – linear path. Figure 6-23 illustrates the paths. The fluorescent lighting in the lab was switched off to prevent interference from ambient light.

7.2.1 Linear paths

a. Path 1: Lathe to Conveyor

The first linear path was described in section 7.1.1. The ability of the AGV to follow the light in the lateral direction was tested. The AGV was instructed to transport a pallet from the lathe and carry it to the conveyor by following the lathe to conveyor path.

The light was turned on at the docking station at the conveyor. The AGV moved toward the light with minimal deviation in the side-right direction. The AGV docked and automatic pallet transfer occurred as described in section 7.1.1. The AGV took an average of 3.6 minutes to move between the lathe and conveyor using light following, as indicated in table 7-3.

Table 7-3 Time to complete the path between the lathe and conveyor.

Test	Time (min)
1	3.8
2	3.6
3	3.5
Average	3.6

b. Path 2: RM and conveyor

The second linear path ran between the RM and the conveyor. The ability of the AGV to follow light in the reverse direction was tested. The AGV was instructed to transport a pallet from the RM and carry it to the conveyor using the RM to conveyor path.

When the AGV moved between the RM and the conveyor, it was not possible for docking to occur as it did between the lathe and conveyor in the lateral direction. This is because the AGV was moving in the reverse direction. The docking station was also located under the conveyor. Therefore the lateral docking mechanism on the AGV could not make contact with the docking mechanism on the conveyor docking station.

A docking station was located at the end of the light path as indicated in figure 6-25. The AGV used the docking mechanisms on its front and rear to dock between the RM and the conveyor. Manual pallet transfer occurred at the conveyor when light following in the reverse direction. The AGV took an average of 2.7 minutes to move between the RM and the conveyor using light following, as indicated in table 7-4.

Table 7-4 Time to complete the path between the RM and conveyor.

Test	Time (min)
1	2.8
2	2.6
3	2.8
Average	2.7

7.2.2 Non - Linear Path: Lathe to RM

The non – linear path ran between the lathe and RM. The ability of the AGV to move in a combination of directions was tested. The resultant motion vector of the AGV was assumed to be in the side – right 315˚ direction as the light sensors received more light from this direction (refer to figure 7-6). All other directions were used for corrective purposes.

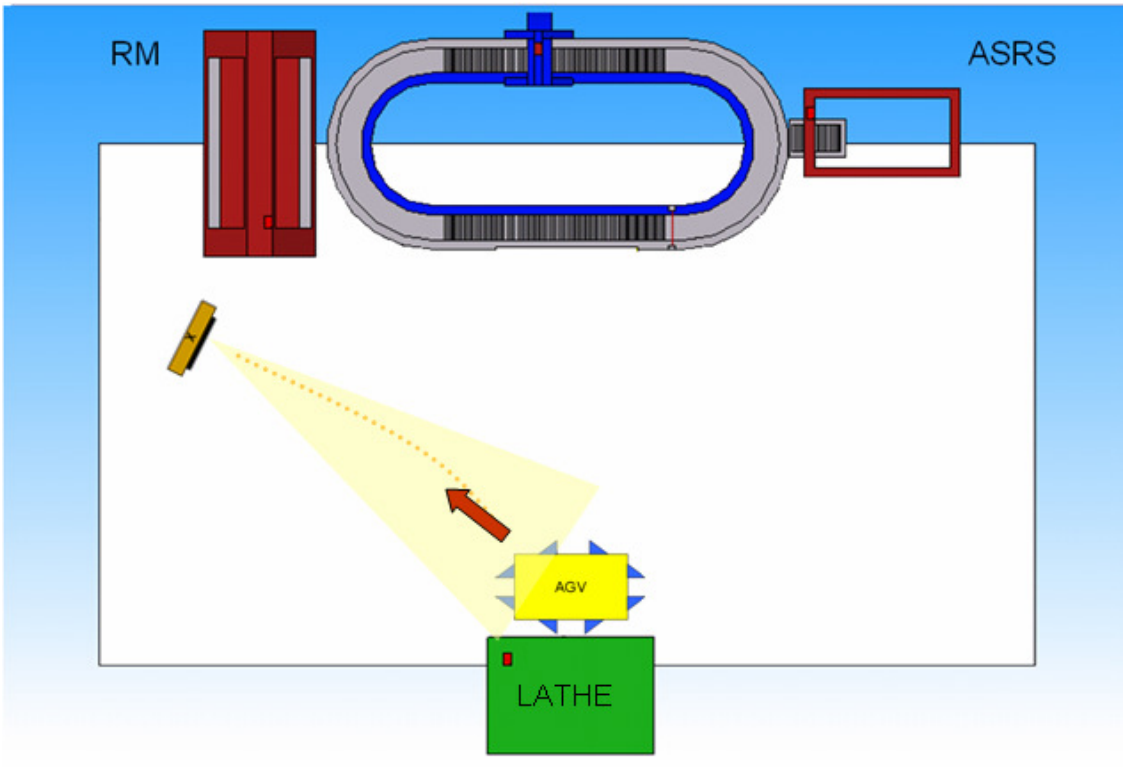


Figure 7-6 AGV with resultant motion vector and light exposure to sensors (indicated by blue triangles).

The AGV was positioned at the lathe and the light on the docking station at the RM. The docking station was positioned at a 45 degree angle to allow maximum light exposure to the light sensors, as illustrated in figure 7-6.

The AGV exhibited different motion patterns when navigating from the lathe to RM depending on the amount and direction of light received. As a result, its orientation when it reached the RM was uncertain. It reached the RM either in the lateral or longitudinal direction as indicated in figures 7-7 and 7-8.

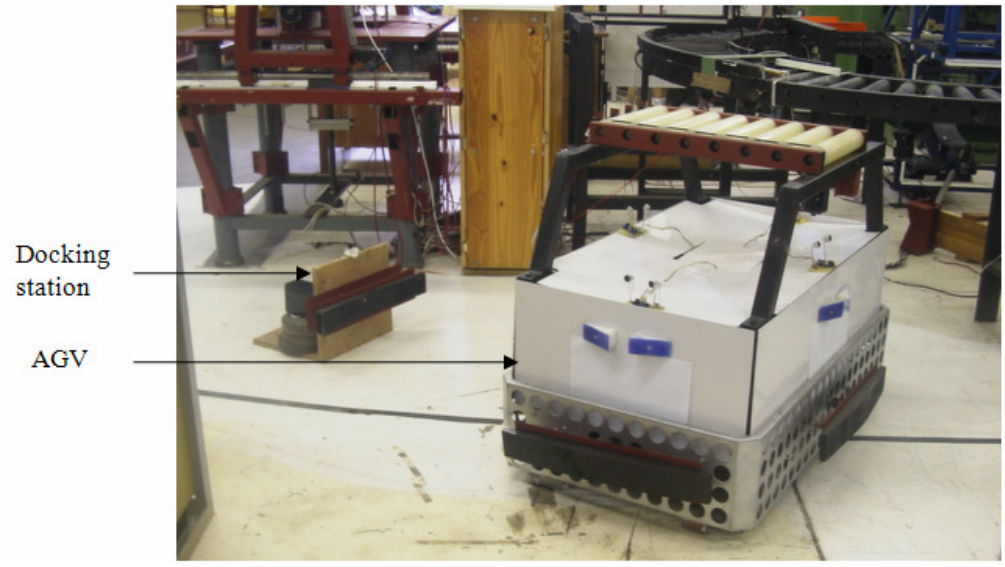


Figure 7-7 AGV in lateral direction at RM docking station

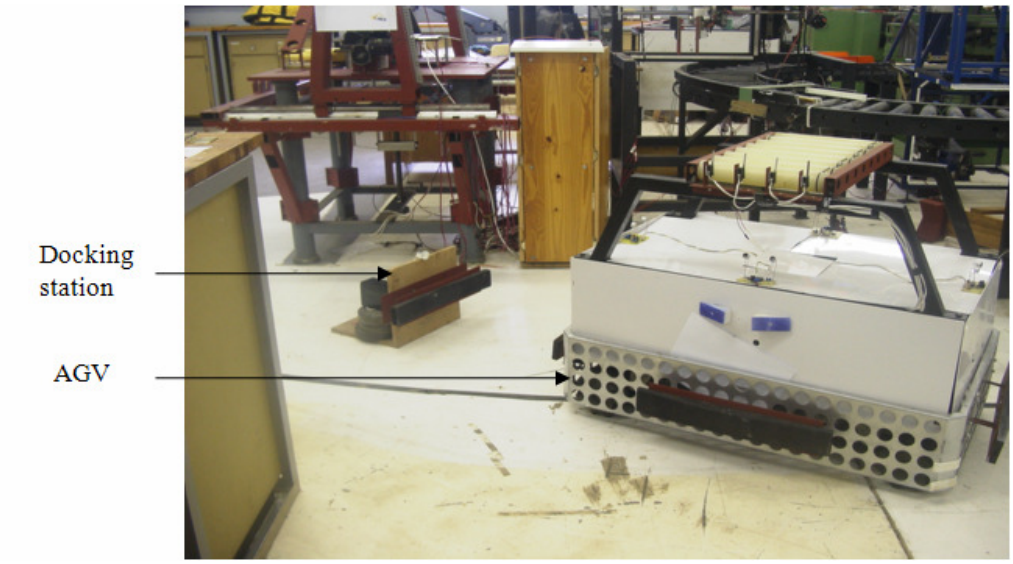


Figure 7-8 AGV in longitudinal direction at RM docking station

The two motion patterns followed by the AGV are indicated in figures 7-9 and 7-10. The AGV was docked at the lathe in the longitudinal position in both cases. If more light was detected by the lateral sensors, the AGV moved laterally in an arc, toward the light at the RM (refer to figure 7-9). If more light was detected by the longitudinal (or forward) sensors, the AGV turned 45 degrees toward the light, and then followed the light longitudinally in an arc (refer to figure 7-10).

Due to the uncertainty of the AGV's orientation at the destination, docking between the RM and lathe did not occur. Instead the AGV was programmed to stop 0.5 m away from the light located on the docking station. The RM also did not have a conveyor interface like the conveyor therefore the AGV's material handling platform could not interface directly with the RM. Manual pallet transfer occurred at the RM.

The ability of the AGV to follow the light in an arc was tested from the lathe to RM. Results were obtained from the longitudinal pattern as this occurred more frequently than the lateral pattern. The AGV was instructed to follow the light three times. As the resultant motion vector of the AGV was in the side-right 315˚ direction, a felt pen was positioned on the side-right 315˚ side of the AGV bumper. The path followed by the AGV to reach the RM was traced each time the AGV was instructed to follow the light from the lathe to RM. Three graphs were derived from the three paths as illustrated in figure 7-11. The average of the three graphs was calculated and plotted as illustrated in figure 7-12. X and y values and averages are indicated in table B-3 and B-4, in Appendix B.

The AGV took an average of 5.4 minutes to move between the lathe and RM using light following, as indicated in table 7-5. The AGV stopped at an average of 0.5m from the light, as indicated in table 7-6.

Table 7-5 Time to complete the path between the lathe and RM.

Test	**Time (min)**
1	5.3
2	5.4
3	5.6
Average	5.4

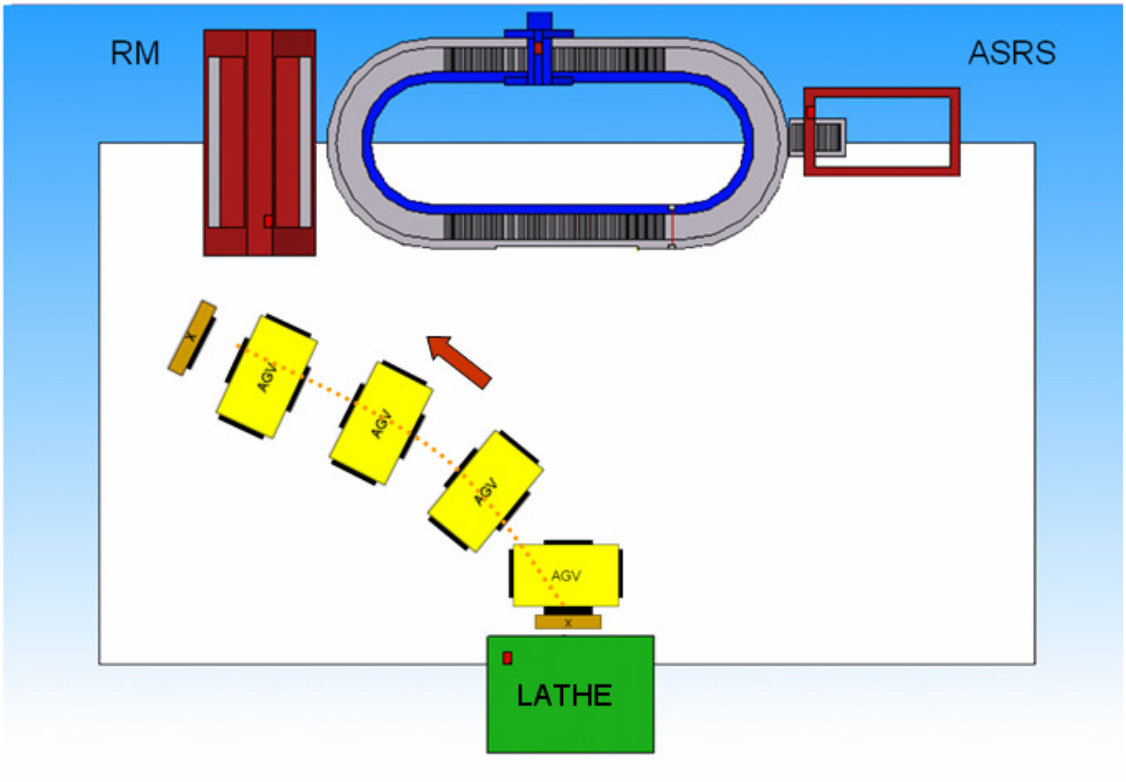


Figure 7-9 AGV following arc in lateral direction to RM

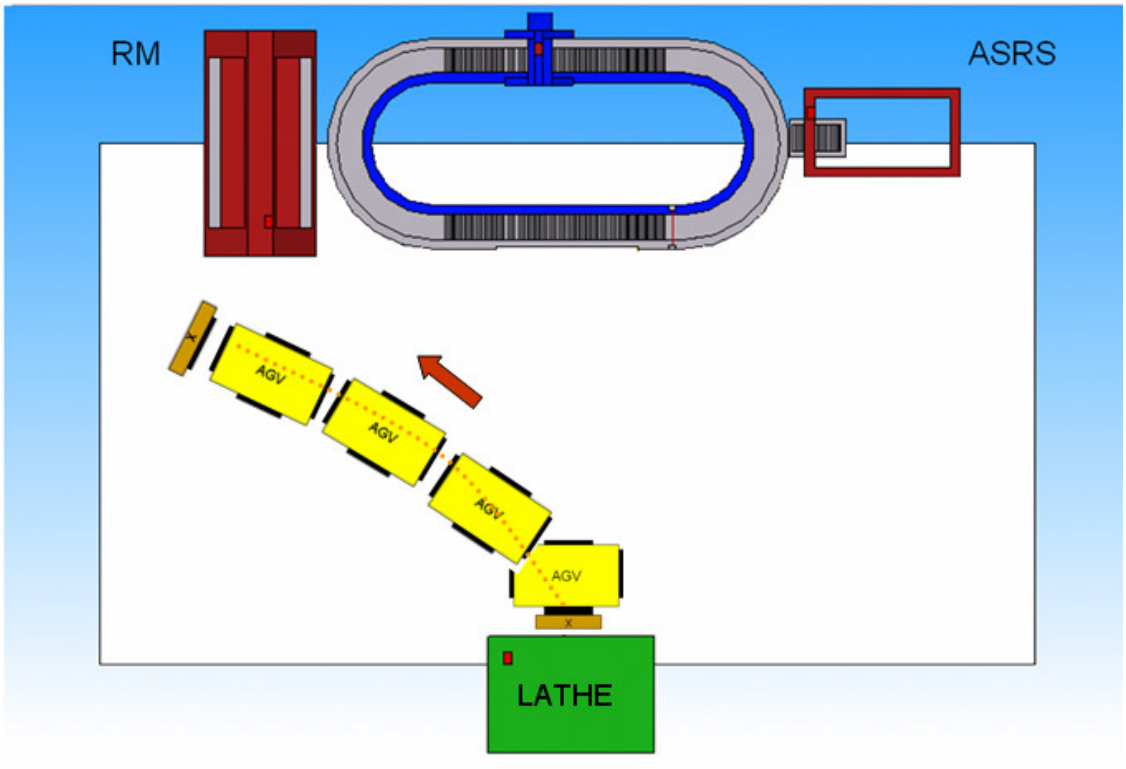


Figure 7-10 AGV in longitudinal direction to RM

Table 7-6 Stopping distance from the light.

Attempt	Stopping distance (m)
1	0.48
2	0.52
3	0.5
4	0.49
5	0.51
Average	0.5

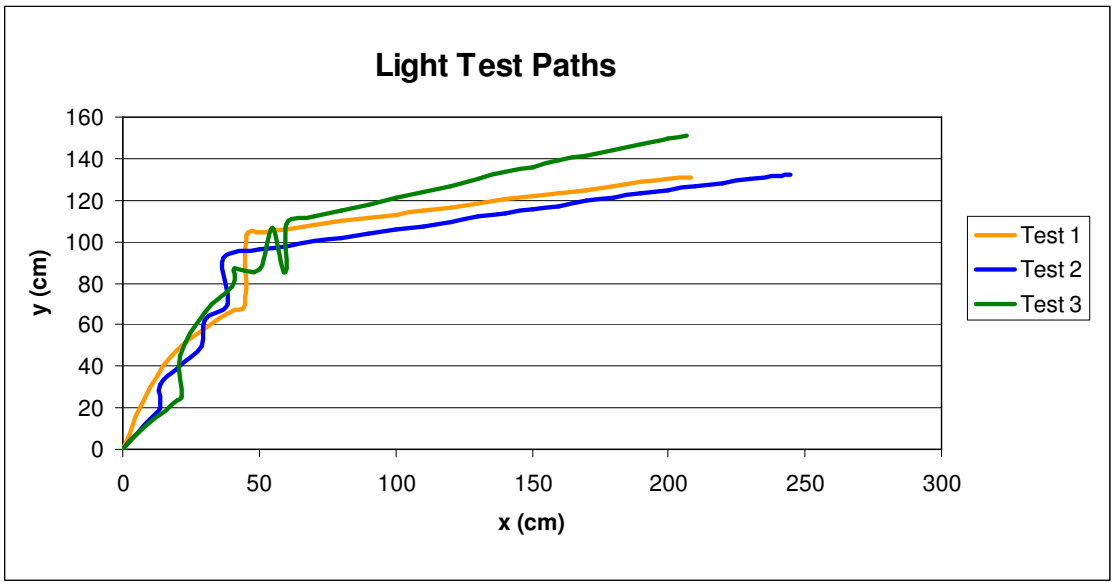


Figure 7-11 Graphs traced of light paths from lathe to RM.

7.2.3 Conclusion

The AGV followed the light in all paths. Docking was possible in the first two paths. The AGV exhibited two different motion patterns when travelling between the RM and lathe, as a result its orientation on reaching its destination point was uncertain.

In order for the AGV to alternate between line following and light following the AGV needed to follow a constant route. Its orientation on reaching its destination point needed to be consistent for docking purposes.

Due to the uncertainty of the curved route it was decided that the path followed between the RM and lathe for the variable sensor program be a combination of straight paths that are clearly defined. This path is explained in section 8.5.c.

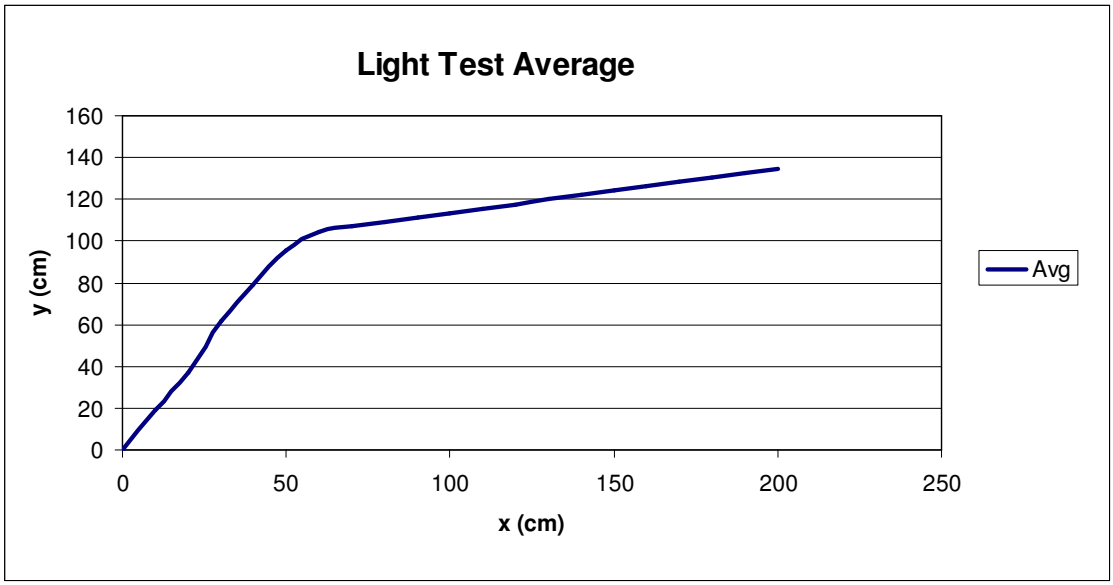


Figure 7-12 Average light path from lathe to RM.

7.3 Light Intensity Test

To determine the light intensity range of the light and line sensors, the sensors were exposed to 5W, 10W, 20W, 35W and 50W quartz halogen light bulbs. The bulbs were powered by a 12V sealed lead acid battery. The fluorescent lighting in the lab was switched off to avoid interference from ambient light.

7.3.1 Light sensors

The AGV was positioned at 0m. The light bulb was fixed on the docking station and placed at distances between 0.3m to 5m away from the AGV's light sensors. Five readings of the two light sensors were taken at every metre with HyperTerminal. The lowest and highest sensor values were taken into account at every metre to determine the sensor reading range. Table 7-7 indicates the results.

7.3.2 Analysis

With reference to table 7-7, the 5W, 10W and 20W bulbs produced readings that were spread out from 595 to 1015 in the 0.3m to 5m range, and suitably defined for programming. The 35W and 50W bulbs produced readings in the 930 to 1020 range for the 0.3m to 5m range, owing to the high light intensity emitted by the bulbs. Readings tended to overlap in the 0.3 to 1m range for the 35W and 50W bulbs. The 5W, 10W and 20W bulbs provided better readings and proved more ideal for practical purposes as they consumed less current.

Table 7-7 Light intensity test results for light sensors

Readings from HyperTerminal = x					
Distance (m)	**5 W**	**10 W**	**20 W**	**35 W**	**50 W**
0.3	855 < x < 905	995 < x < 1010	1010 < x < 1015	1015 < x < 1020	1000 < x <1010
0.5	900 < x < 965	990 < x < 1000	1000 < x < 1005	1010 < x < 1020	1000 < x < 1015
1	910 < x < 945	960 < x < 970	960 < x < 970	1005 < x < 1010	1000 < x <1010
2	830 < x < 870	890 < x < 915	900 < x < 905	985 < x < 995	980 < x < 990
3	740 < x < 790	825 < x < 865	845 < x < 860	970< x < 980	965 < x < 975
4	660 < x < 720	765 < x < 825	790 < x < 820	950 < x < 965	940 < x < 955
5	595 < x < 680	710 < x < 790	740 < x < 775	930 < x < 945	940 < x < 950

7.3.3 Line Sensors

The line sensors were exposed to the light intensity emitted by 5W, 10W, 20W, 35W and 50W bulbs over the black tape and on the white floor. Table 7-8 illustrates the average value obtained from the five readings in HyperTerminal.

Table 7-8 Light intensity test results for line sensors

Readings from HyperTerminal = x					
	5W	**10W**	**20W**	**35 W**	**50W**
Black	340 < x < 430	450 < x < 600	710 < x < 740	700 < x < 840	730 < x < 800
White	700 < x < 850	900 < x < 950	940 < x < 950	970 < x < 990	960 < x < 970

7.3.4 Analysis

All bulbs produced distinguishable readings on the white floor and black tape. The 35W and 50W bulbs drew more current from the 12V battery than the 5W, 10W and 20W bulbs. The battery that powered the light for line following was on board the AGV therefore power consumption had to be minimised. The 5W, 10W and 20W bulbs proved more ideal for line following.

7.3.5 Conclusion

Programming was optimized so that the user could specify what light intensity, high or low, was to be implemented for the light variation technique (section 8.4) by using the light sensor readings in table 7-7. By taking the highest and lowest readings in the 0.5 to 2m range, the 5W and 10W bulb fell in the low light intensity range, 830-1000, and the 20W, 35W and 50W bulbs fell in the high light intensity range, 900-1020. These results were used for steering the AGV when light following under different light intensities for the light variation technique.

The AGV was programmed to stop at specific distances in front of the light to change direction from reverse to side-left when travelling between the RM and lathe (figure 9-3). Readings in the 0.3 to 0.5m range overlapped for the high light intensity bulbs therefore the value of '10' (1000-1015) was used to stop the AGV at 0.5m in front of the light for high intensity light following. The 5W bulb would never receive the '10' value, therefore the value of '9' (910-970) was used to stop the AGV 1m in front of the light for low intensity light following. The 10W bulb did receive the '10' reading at 0.5m but this overlapped with the high intensity bulb readings. Therefore when light following at the lower light intensity the AGV was programmed to stop 1m in front of the light to change direction. This prevented confusion between high and low light intensities and ensured correct routing at junctions (sections 8.7.e and 8.7.f).

The line sensor readings were implemented in the line variation technique for line following with a 20W bulb as in the line following technique (section 6.3.1). The results in table 7-8 could be included in future work for line following under different light intensities to demonstrate flexibility of the variable sensor system.

7.4 High Noise Level Test

To determine the AGV's ability to detect obstacles under high noise levels, the AGV's ultrasonic sensors were exposed to the sound emitted by a mill, lathe, belt grinder, wheel grinder, wood saw and fans in a workshop environment. The sound was measured using a sound meter. The sensors were also exposed to the combined sound emitted by machines in the same work room. (The grinders and fans were in one room and the mill and lathe in another room.)

The AGV was positioned at 0m and an obstacle was moved from 0.5 to 3 m. Three readings were taken at every meter and the average calculated. Tables 7-9 and 7-10 indicate the average value in metres, at each distance under the different noise levels.

The ultrasonic sensors worked from 0.5m to 2m under the 74.4 db to 93.5 db range (refer to table 7-9). At 3m the sensors produced an error reading as they could not detect the obstacle under the 74.4 db to 93.5 db range. At 70.3 db, obstacle detection at 3m was possible, due to the lower sound of the wheel grinder. The graph in figure 7-13 illustrates the deviation of the experimental distance from the actual distance. The sensors yielded minimal deviation from the actual distance.

The ultrasonic sensors worked from 0.5m to 2m under the average sound emitted by the combined machines for the 76.3 db to 84.1 db range (refer to table 7-10). At 3m the sensors produced an error reading as they could not detect the obstacle. The graph in figure 7-14 illustrates the deviation of the experimental distance from the actual distance. Minimal deviation from the actual distance was noted.

7.4.1 Conclusion

The tests concluded that the ultrasonic sensors were robust in the 0-2m range. The results were used for programming the AGV to operate under the high noise levels by setting a threshold value of 1m for steering and stopping the AGV (section 9.1).

Table 7-9 Average value in metres, at each distance, at the different noise level for individual machines.

	Individual Machine – Noise level					
Distance (m)	**Wheel Grinder 70.3 db**	**Mill 74.4 db**	**Workshop Fans 82.2 db**	**Lathe 84.7 db**	**Belt Grinder 86 db**	**Wood Saw 93.5 db**
0.5	0.37	0.37	0.38	0.35	0.38	0.37
1	0.92	0.90	0.92	0.88	0.90	0.89
2	1.95	1.92	1.94	1.92	1.95	1.93
3	2.80	Error	Error	Error	Error	Error

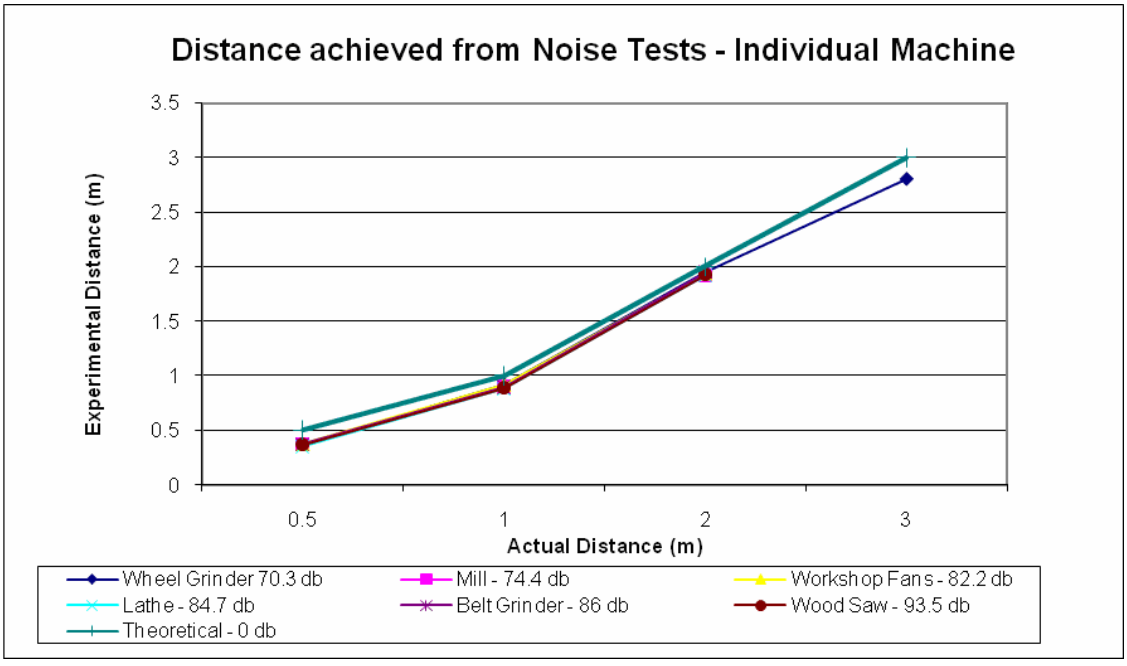


Figure 7-13 Graph illustrating distance achieved from noise test and deviation of the experimental distance from the actual distance for individual machines.

Table 7-10 Average value in metres, at each distance, at the different noise level for combined machines.

	Combination of Machines – Average Noise level			
Distance (m)	**Wheel Grinder & Fans 76.3 db**	**Belt Grinder & Fans 84.1 db**	**Both Grinders & Fans 79.5 db**	**Mill & Lathe 79.6 db**
0.5	0.36	0.33	0.33	0.35
1	0.89	0.91	0.89	0.89
2	1.96	1.95	1.95	1.92
3	Error	Error	Error	Error

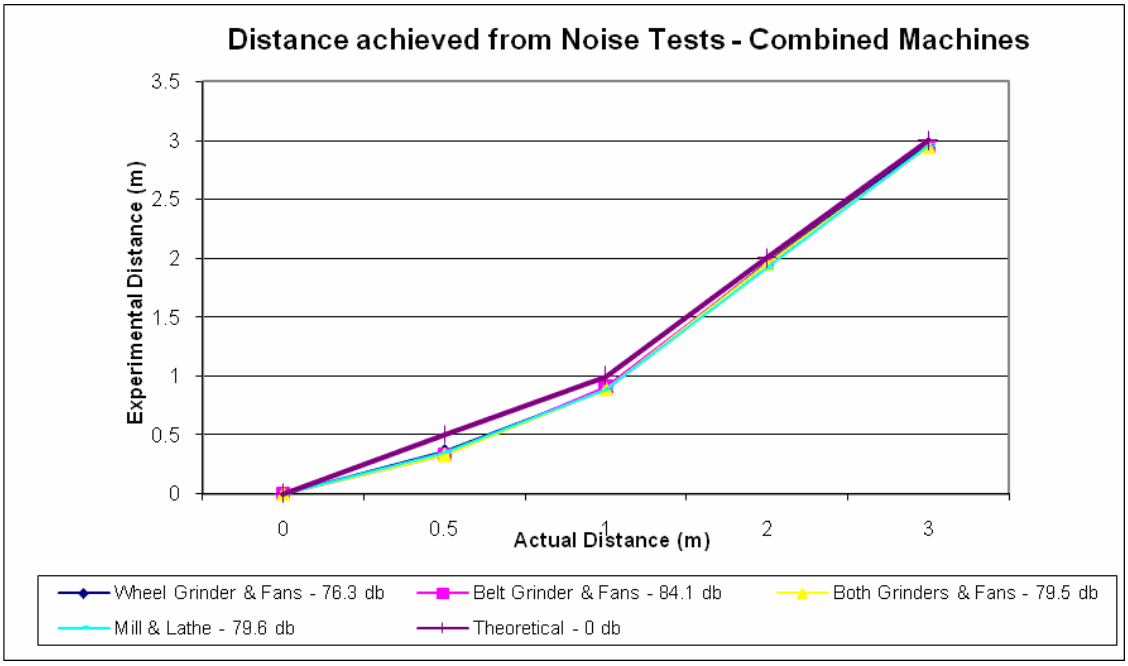


Figure 7-14 Graph illustrating distance achieved from noise test and deviation of the experimental distance from the actual distance for combined machines.

7.5 Summary

This chapter described preliminary testing of the line following and light following techniques in the constant environment. Preliminary testing was conducted to determine the accuracy and reliability of the two techniques and how they could be combined for the changing environments. The AGV was tested to follow the line on a linear and non-linear path. The AGV was 80% reliable in the lateral direction and 90% reliable in the forward direction on the linear path. The AGV was 84% accurate on the non-linear path. Graphs were derived of the movement of the AGV on the non-linear path and the average motion and accuracy calculated.

The AGV was tested to light follow on two linear paths and one non-linear path. The AGV followed the light and docked in the reverse and lateral directions on the respective linear paths. The AGV displayed longitudinal and lateral movement patterns on the non-linear path. The AGV's end orientation was inconsistent on the non-linear path. Graphs were derived of the motion of the AGV on the non-linear path for the longitudinal pattern and the average movement calculated. Light following in a non-linear path was uncertain and unreliable. A combination of straight paths was therefore used for the unpredictable environments.

Light intensity testing was conducted for 5W to 50W light bulbs at distances from 0.3m to 5m. LDR readings were used in the light variation technique for high and low intensity light following. High noise level testing was conducted to determine the durability of the ultrasonic sensors under different noise levels. Graphs were derived of the sensor readings at the distances tested under the different noise levels. The sensors were found reliable within a 0m to 2m range. Results were used to tailor the programming such that the AGV would be able to complete its tasks regardless of the high noise levels.

8 Software

8.1 User Interface program

The program for the variable sensor system integrated the different sensors and allowed the AGV to complete its material handling tasks. These tasks included transporting pallets between the conveyor, lathe and RM. The basic program was optimized according to results obtained from preliminary testing, Chapter 7.

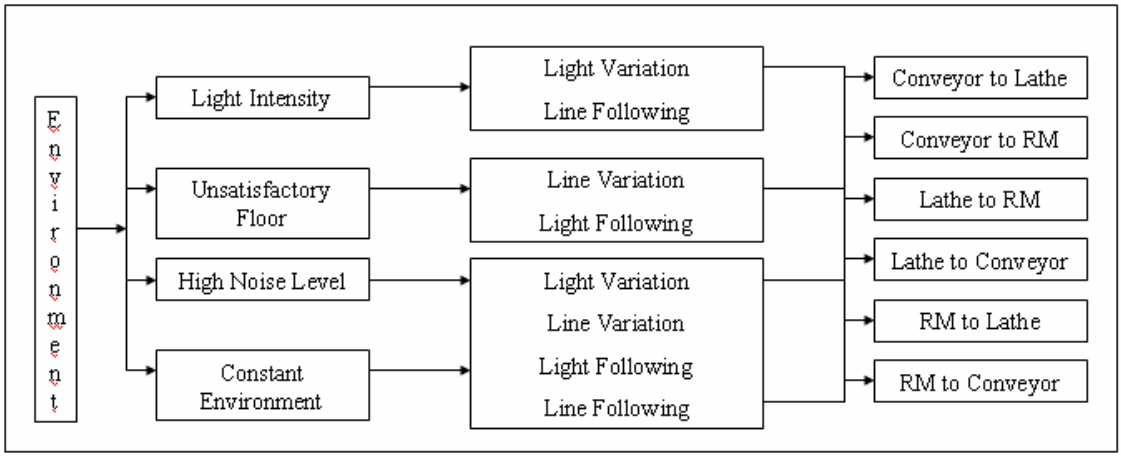


Figure 8-1 Program layout

The program layout is illustrated in figure 8-1. The user was prompted to choose an environment that the AGV would be operating in: light intensity, unsatisfactory floor, high noise level, or constant environment. If the light intensity environment was chosen, the user was prompted to indicate the light intensity, high or low, dependent on the bulb watt being used. Once an environment was chosen, the variable sensor techniques suitable for guidance and navigation in that environment were displayed. All techniques incorporated the use of limit switches for docking and emergency stopping, and ultrasonic sensors for obstacle avoidance. The user was prompted to choose an appropriate technique and a desired route. The AGV utilised the particular technique to complete the assigned task on route. If an unknown environment, technique, or route was selected, the user was kindly prompted to select again.

Table 8-1 Techniques suitable for environment

Environment	Techniques
High noise level	Line variation
	Light variation
	Line following
	Light following
Light intensity	Light variation
	Line following
Unsatisfactory floor	Line variation
	Light following
Constant	Line following
	Light following
	Line variation
	Light variation

Table 8-1 lists the techniques that the user could use in a specific environment. In the high noise level environment, any one of the four techniques could be applied as the programming accommodated for the high noise level by setting threshold values of 1m (section 9.1). In the light intensity environment, light intensity was varied, therefore line following could be used as it was not dependent on constant light readings. The light variation technique was designed especially for the light intensity environment. The technique allowed the AGV to operate under the varying light intensity by alternating from high to low light intensity, or line following when there was no light (section 8.4).

In the unsatisfactory floor environment, floor conditions were not suitable for line following, i.e. there was no line in certain parts of the track. Light following could be used as it was not dependent on constant line readings. The line variation technique was designed especially for the unsatisfactory floor environment. The technique allowed the AGV to alternate from line following to light following in the absence of the line (section 8.3). In the constant environment any one of the four techniques could be applied. The line variation program would constantly line follow due to the constant line. The light variation program would constantly light follow due to the constant light. The environments are explained in detail in Chapter 9.

A typical interface with selections made is illustrated in figure 8-2. The user has selected the light intensity environment, 2, the line variation technique, 1, a high light intensity, 1, and the lathe to conveyor route, 3. The sequence was initialized and the AGV proceeded to transport a pallet from the lathe to the conveyor. Sensor feedback was obtained as illustrated in figure 8-3.

When limit switch 0 was compressed (s0=C in figure 8-4) on docking at the conveyor, automatic pallet transfer occurred. The pallet compressed limit switches 2 to 5 (s2=C and s3=C) at the end of the platform. The user was informed that the pallet was loaded. The program returned to the main interface in figure 8-2 and the AGV awaited further instructions.

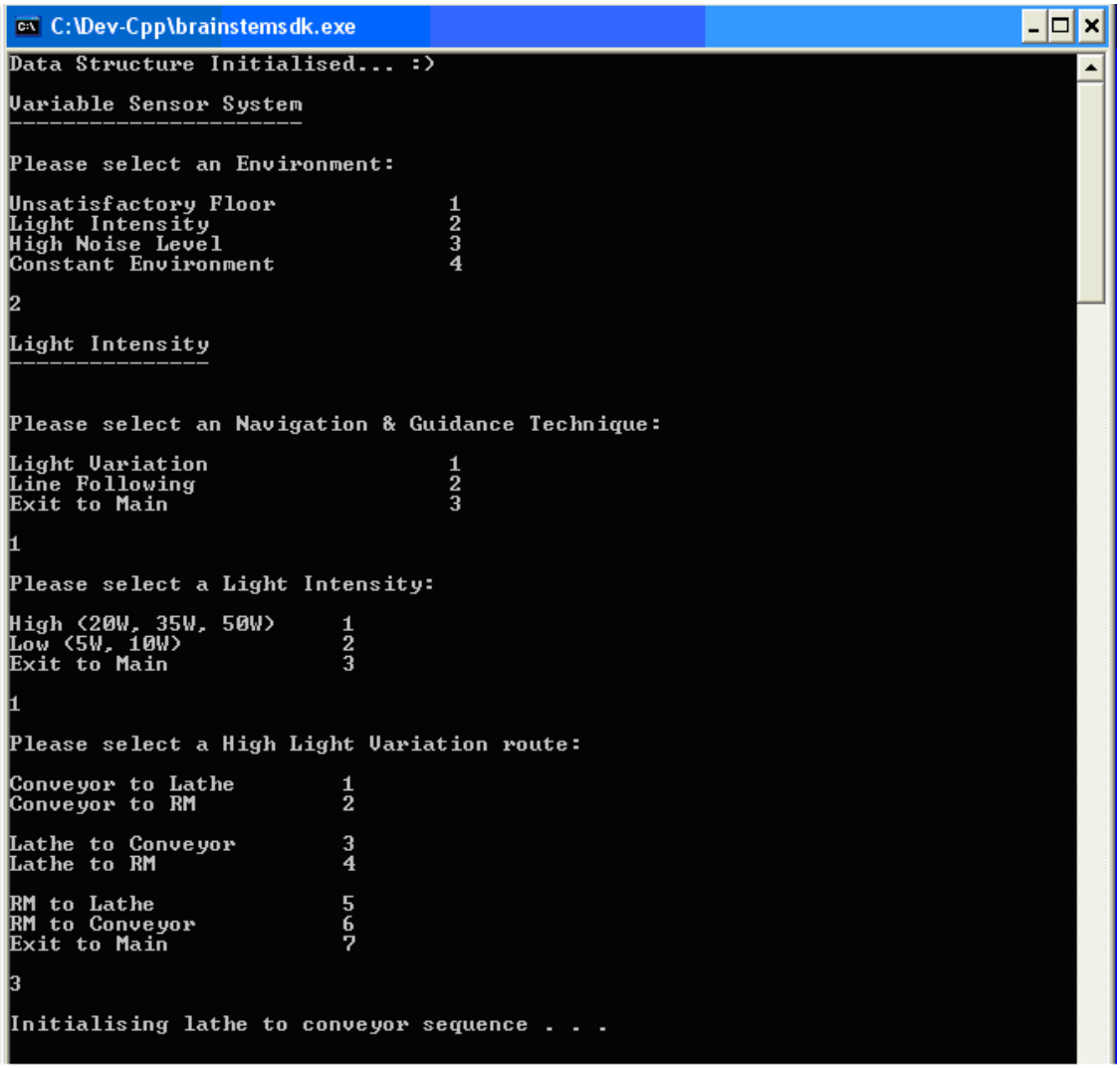


Figure 8-2 User interface

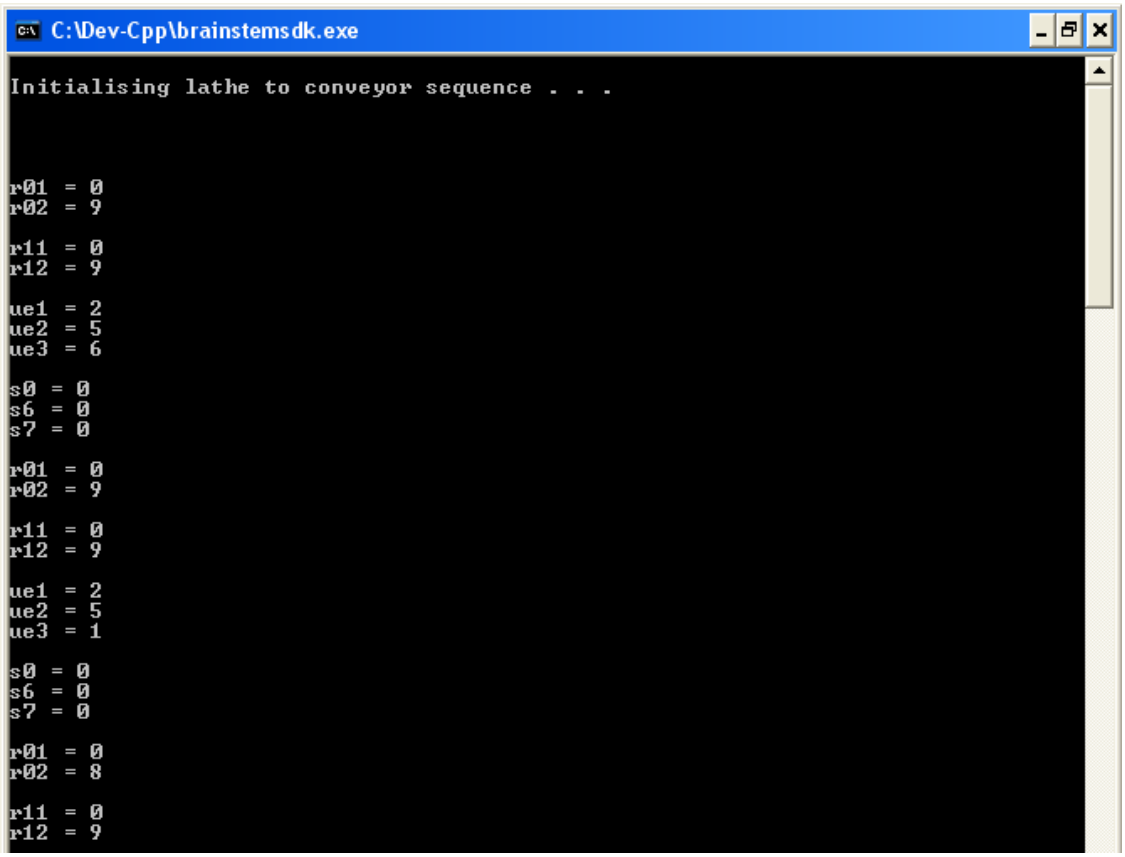


Figure 8-3 Sensor feedback

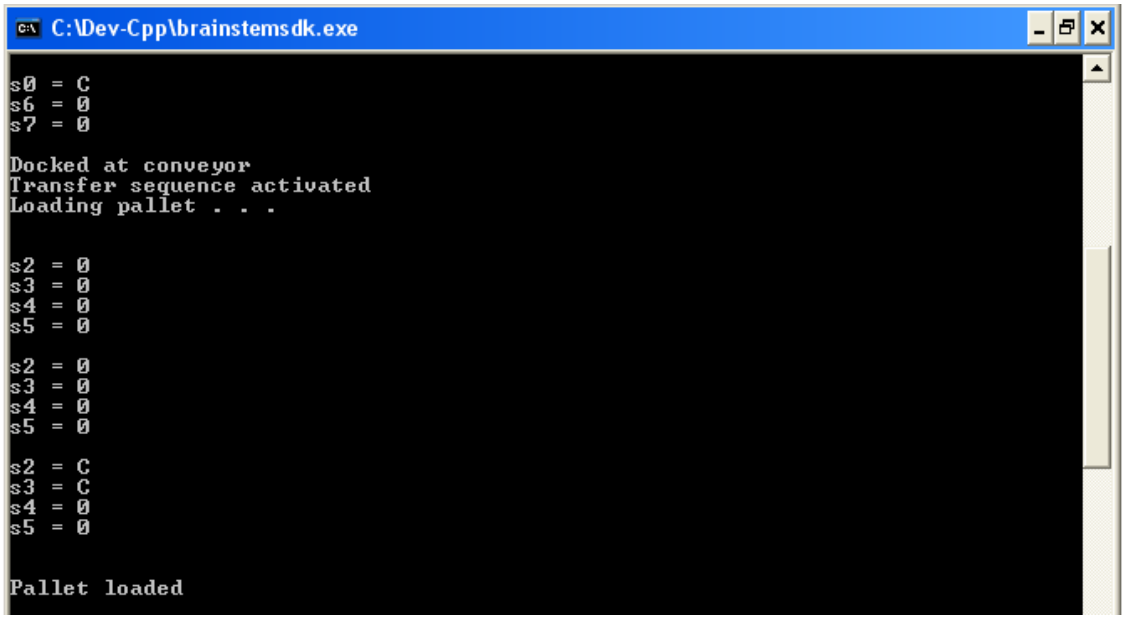


Figure 8-4 Interface after task completion

8.2 Program progress and optimization

The main objective of the software was to provide the AGV with the ability to alternate guidance and navigation techniques, as the parameters in the environment changed, in order to complete material handling tasks. This would maintain material handling operations and guarantee production rates in Agile manufacturing environments.

The variable sensor system incorporated two new techniques: a line variation and light variation technique. The final programs for these techniques were an improvement of two earlier test programs (sections 8.2.1 to 8.2.3). As a temporary measure the test programs were stopped by the user through software when required, as they did not include the relevant limit switches and ultrasonic sensors for stopping. These sensors were integrated in the final programs. The test programs were tested and modified to determine the most appropriate method in which the program could alternate between line and light following techniques. Programming for the three test programs is included in Appendix E.3. The program name is indicated next to the respective heading. Optimization with respect to the line variation program is discussed in sections 8.2.1 to 8.2.3.

Ultrasonic sensors and limit switches were integrated in the final four guidance techniques as applied in tables 9.1 and 6-19 respectively. The logic flow for the ultrasonic sensors and limit switches is illustrated in figure 8-5. When the respective limit switch was compressed, for either docking or emergency stopping, in the event of a collision, the user was informed through the program interface. The user determined whether a new task should be allocated. When the ultrasonic sensors detected an obstacle the AGV paused until the path was clear for travel. On sensing a clear path the AGV continued steering, utilising the respective technique, towards the destination. In the flow charts illustrated in figures 8-6 to 8-10, the ‘end’ is not indicated as this was internal to each guidance technique. Guidance ended when the relevant limit switch was compressed while using a particular technique. The user could then assign another task.

8.2.1 First Program: LineLight.c

The line variation program allowed the AGV to change from line following to light following if the line was absent in a certain section of the path. The flow chart in figure 8-6 illustrates the logic flow in the first program, LineLight.c. (Appendix E.3.1)

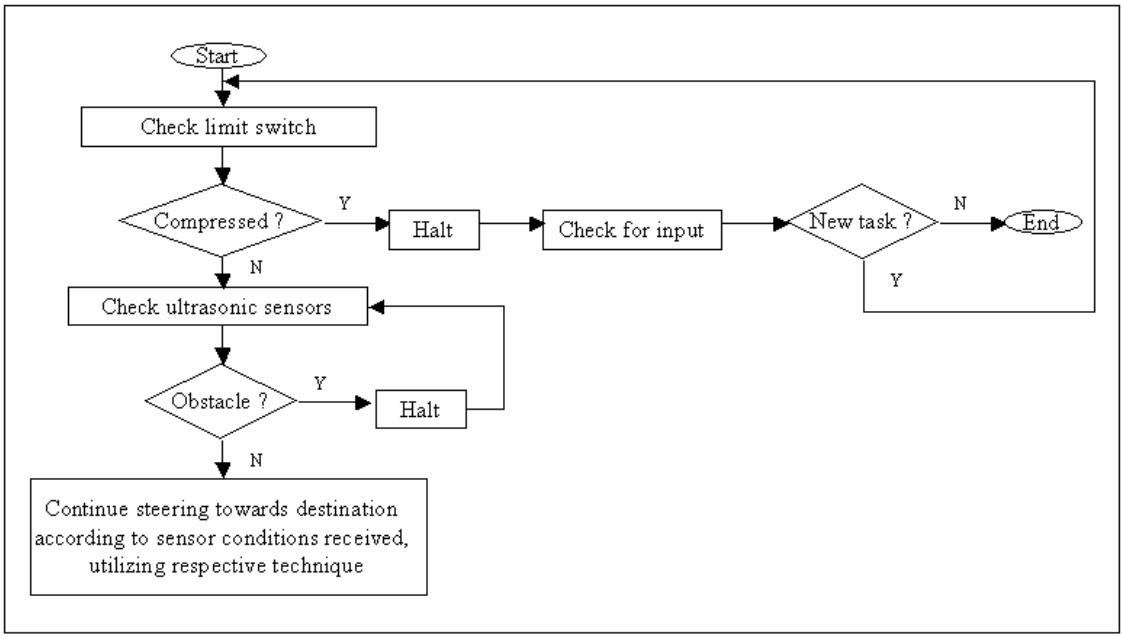


Figure 8-5 Flow chart illustrating limit switch and ultrasonic sensor logic.

The AGV was instructed to complete a task using line following. The light was located on the docking station at the end of the line. If at any time during travel the line sensors did not detect the line, the program automatically changed to light following. If the light sensors did not detect the light initially, the AGV moved in a clockwise direction to search for the light. Once the light was detected the AGV proceeded toward the light. It was noted that as the AGV followed the light at the end of the line, it moved more or less along the line or tended towards the line.

The disadvantage of this program was that it only changed once from line following to light following. The program could not change back to line following should the line sensors detect the line again. The advantage of this program was that it provided the AGV with the ability to search for the light when it changed from line following to light following.

The interchange program from light following to line following (LightLine.c) worked in a similar manner. When the light sensors did not detect light the program changed from light following to line following. The AGV moved in a clockwise direction to search for the line if the line could not be detected initially. A disadvantage noted was that the AGV would sometimes detect the line when it had turned 180 degrees and follow the line in the opposite direction.

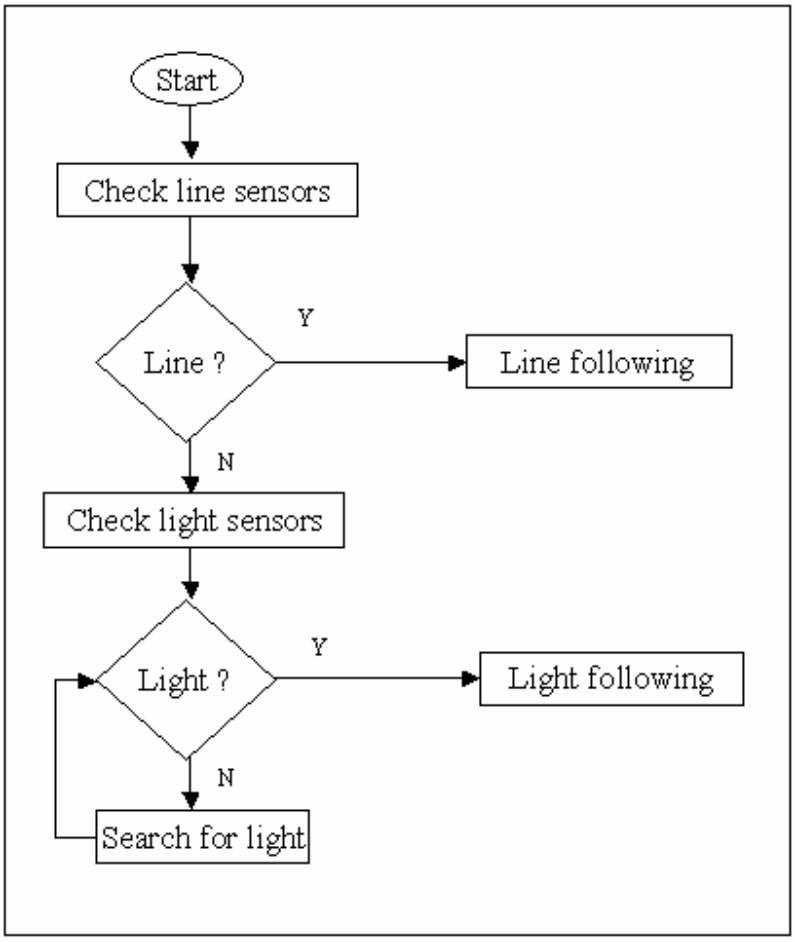


Figure 8-6 Flow chart illustrating logic flow in the first program

8.2.2 Second Program: InterLineLight.c

The second program eliminated the disadvantage of the first program whereby the program could only change once from line following to light following. The program provided the AGV with the ability to change from line following to light following and alternate between the two techniques, a multiple number of times, as the parameters in the environment changed. The program consisted of two loops that looped within each other. The loops would end when the relevant limit switch was compressed on docking or if the AGV crashed. The flow chart in figure 8-7 illustrates the logic flow for the second program, InterLineLight.c. (Appendix E.3.2)

The AGV was instructed to complete a task using line following. If the line sensors did not detect the line, the program automatically changed to light following. While light following if the light

sensors did not detect the light, the AGV started line following. The program interchanged between line following and light following as the parameters changed in order to reach the destination.

The logic in the program was such that if the line could not be detected, the program changed to light following. If the light could not be detected, the program changed to line following. The disadvantage of this program was that the AGV could not search for the line or light as this would prevent alternating between the two loops. Another disadvantage was that the light or line might not always be present when the program needed to alternate techniques. For example, if the AGV could not detect the light and changed to line following, the line might be broken in that part of the track. The program would keep interchanging techniques but if it did not detect the light or line again the AGV would remain in the same position. The advantage of this program was that it alternated between techniques as many times as the parameters changed until the destination was reached.

The interchange program from light following to line following (InterLightLine.c) worked in a similar manner. The AGV started off light following and changed to line following if the light sensors did not detect the light. If the line sensors did not detect the line the program changed back to light following. The program alternated between the two techniques as the parameters varied in order to reach the destination.

8.2.3 Third Program: LineLightSim.c

The third program was based on the observation that as the AGV followed the light at the end of the path, it moved closely to the line following path (established in section 8.2.1). This meant that the AGV could return to line following if the line was detected again.

The third program incorporated the ability of the second program to interchange between techniques a multitude of times. The program also provided the AGV with the ability to search for the line or light eliminating the disadvantage in the second program. The logic flow from line following to light following is illustrated in the flow chart in figure 8-8 for the third program, LineLightSim.c. (Appendix E.3.3)

The AGV was instructed to complete a task using line following. If the line sensors did not detect the line, the program automatically changed to light following. If the light sensors did not detect the

light initially, the AGV moved in a clockwise direction to search for the light. When the light was detected the AGV proceeded toward the light.

While light following, the line sensors checked for the presence of the line. If the line was detected the program changed back to line following. The program alternated between the two techniques as the parameters changed, as many times as needed, in order to reach the destination.

The logic flow from light following to line following (LightLineSim.c) is illustrated in the flow chart in figure 8-9. The AGV was instructed to complete a task using light following. If the light sensors did not detect the light, the program automatically changed to line following. If the line sensors did not detect the line initially, the AGV moved in a clockwise direction until the line was detected. When the line was detected the AGV proceeded along the line. While line following, the light sensors checked for the presence of light. If the light was detected the program changed back to light following. The program interchanged between the two techniques as the parameters varied, as many times as required, in order to reach the destination.

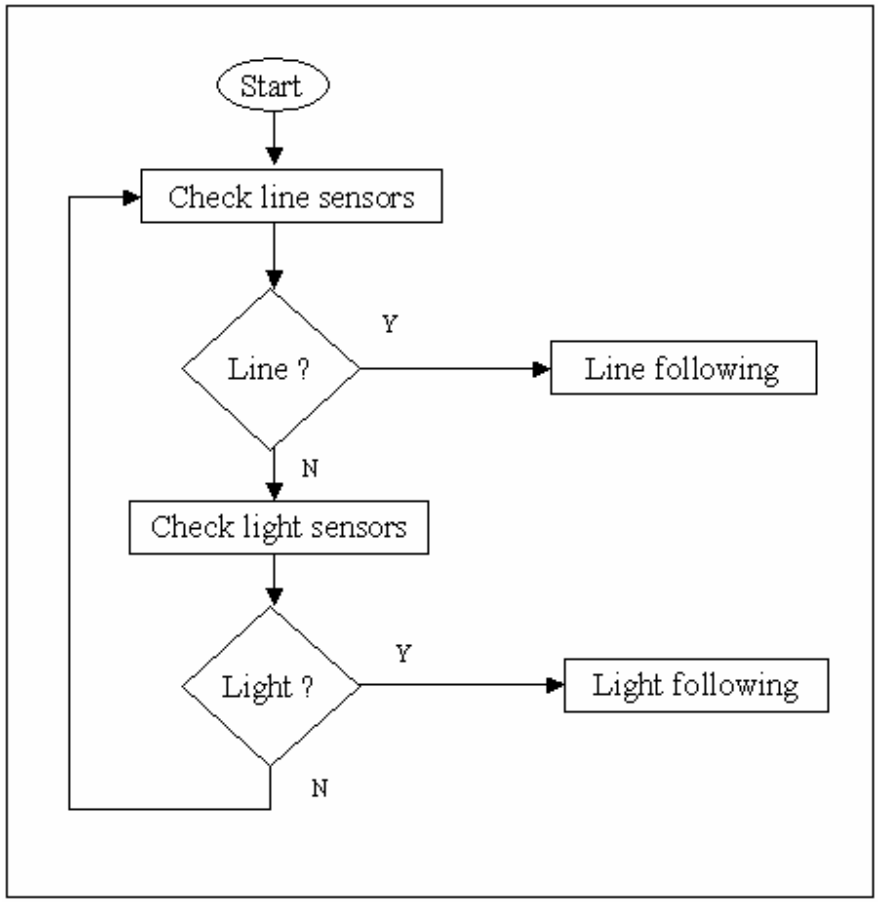


Figure 8-7 Flow chart illustrating logic flow in the second program

The advantage of this program was that it alternated between the two techniques as many times as required, as the parameters changed. The program also provided the AGV with the ability to search for the line or light by rotating in a clockwise direction until the line or light was detected. A disadvantage of the program was that while searching for the line there was a possibility that the AGV might miss the line, turn a full 180 degrees, and detect the line in the opposite direction. In which case, the AGV would have to be reinstructed to complete the task. The advantages and disadvantages of the three programs are compared in table 8-2.

8.2.4 Conclusion

The third program eliminated most of the disadvantages of the first two programs. The logic of the third program was used for the final line and light variation programs with the integration of ultrasonic sensors and limit switches as explained in section 8.2.

Table 8-2 Advantages and disadvantages of the three programs.

Program	Advantages	Disadvantages
First Program	1. Searched for line or light.	1. Could only interchange once from line following to light following or vice-versa. 2. When searching for line, sometimes detected line in opposite direction.
Second Program	1. Could successively interchange from line following to light following as needed until destination was reached or vice-versa.	1. Could not search for the line or light. 2. Light or line may not always be present when program interchanging needs to take place.
Third Program	1. Could successively interchange from line following to light following as needed until destination was reached or vice-versa 2. Searched for line or light	1. When searching for line, sometimes detected line in opposite direction.

8.3 Final Line Variation Program

The final line variation program provided the AGV with the ability to line follow under unpredictable conditions in the environment, such as if the line was absent, damaged or worn out in certain parts of the path. If conditions were not suitable for line following the program changed to light following. The program checked if acceptable conditions prevailed for line following later during travel, whereupon it returned to line following. The programming for all line variation routes is in Appendix E.1.5 with the prefix LineLight. The line variation routes are explained in section 8.6.

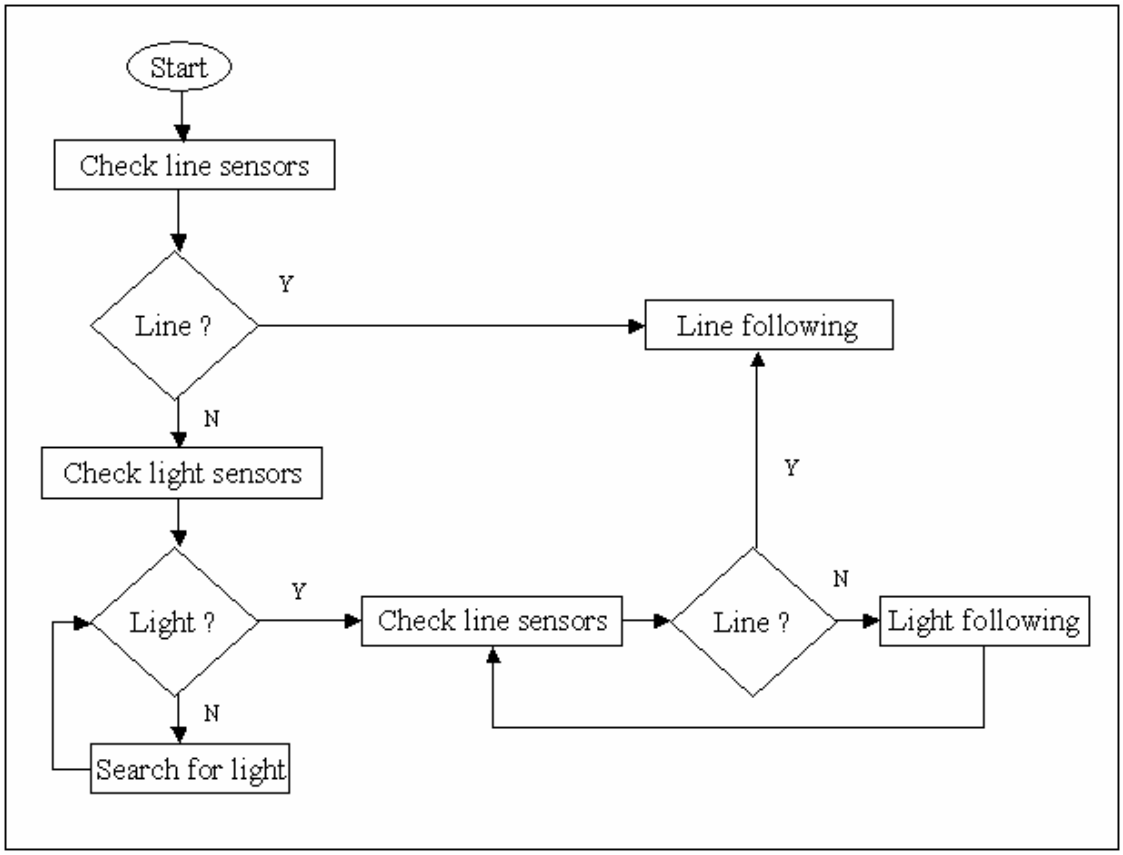


Figure 8-8 Flow chart illustrating logic flow in the third program for alternating from line following to light following

8.4 Light variation program

The light variation program provided the AGV with the ability to operate under different light intensities. The light sensors required light readings within a specific range, for a given light intensity, to steer the AGV in the correct direction. If the light intensity changed, from high to low, or no light during task execution the sensor readings received also changed and affected the AGV's ability to follow the light correctly. The light variation program allowed the AGV to operate under the different light intensities produced by 0W, 5W, 10W, 20W, 35W and 50W light bulbs, in order to complete its tasks by alternating from high to low light following, and line following in the absence of light.

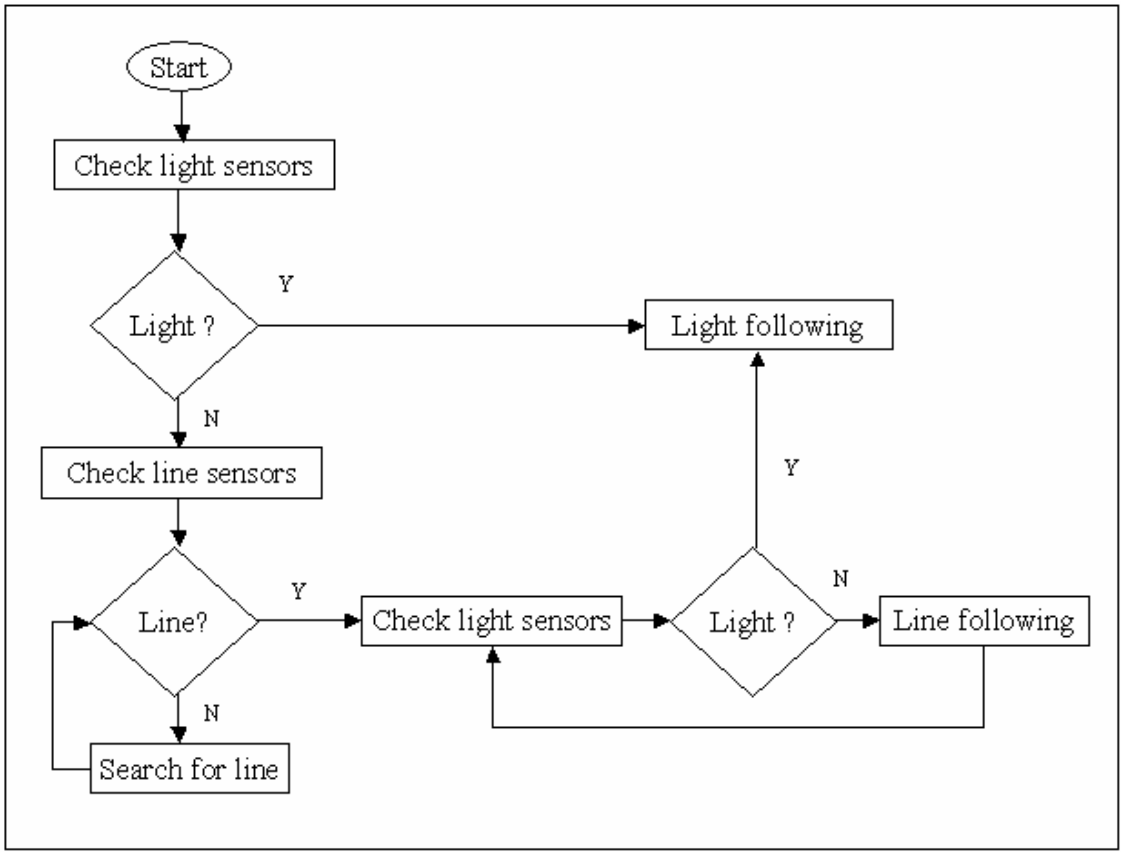


Figure 8-9 Flow chart illustrating logic flow in the third program for alternating from light following to line following

The ability of the AGV to continue to light follow regardless of the variation in light intensity is required when a high intensity light bulb's readings decrease in value due to a lower light intensity emitted by the bulb. In such a situation the program will allow the AGV to change from high intensity light following to low intensity light following. When the light intensity changes to no light intensity in the case of a power outage or when a bulb is being changed, the program will alternate from high or low intensity light following to line following to enable the AGV to continue operation.

The logic flow is illustrated in the flow chart in figure 8-10. The user started light following in the high light intensity range. When a decrease in light intensity was detected the program loaded the low light following routine. This allowed the AGV to continue with its task. Both light sensors in the direction of motion had to have the same sensor reading of '8' (Refer to table 8-3) in order for the program to change to the low light intensity. The 20W, 35W, 50W bulbs operated within the high light intensity range (high light following) and the 10W and 5W bulbs operated within the low

light intensity range (low light following). The program loaded the low light intensity file when the high light intensity bulbs started to decrease in light intensity or when the user specified that the low light intensity was to be used.

When there was no light, 0W, the program alternated to line following and had the ability to return to the high light following program, if light was detected again. High light following would change to low light following if the light intensity decreased, and to line following if there was no light.

Low light following did not return to high light following because the value '9' which would be used to detect high light intensity was used to stop the AGV for changing directions at junctions. It was also unnecessary for low light following to return to high light following because if a bulb was decreasing in light intensity it would have to be switched off, replaced, and then switched on. While the light was switched off the program would change to line following. When the light was switched on again, the program would change to high light following. If a low light intensity was detected the program would change from high light following to low light following. Tables 8-3 and 8-4 indicate sensor readings and values used for programming.

In order to accomplish routing at junctions for light following the AGV was programmed to stop a pre-determined distance in front of the light it was following in the first travel direction, and then change to the second direction on sensing the light in the second direction. For example, if the AGV was travelling from the RM to lathe as illustrated in figure 9-3, the AGV was programmed to stop 0.5m in front of the light, in the reverse direction, and on sensing the light in the side-left direction, change direction to side-left.

Table 8-3 Light intensity sensor readings for steering and program allocation.

Light intensity	**Light intensity range at 2m in HyperTerminal**	**Programming value for both sensors**	**Program loaded**
0W	$x = 0$	$x < 7$	Line following
5W	$830 < x < 870$	8	Low light following
10W	$890 < x < 915$		
20W	$900 < x < 905$	9	High light following
35W	$985 < x < 995$		
50W	$980 < x < 990$		

Table 8-4 Light intensity sensor readings for changing direction and program allocation.

Light intensity	**Light intensity range at 0.5m in HyperTerminal**	**Light intensity range at 1m in HyperTerminal**	**Programming value for both light sensors to change travel direction**	**Program loaded**
0W	x = 0	x = 0	N/A uses line to change direction	Line following
5W	900 < x < 965	910 < x < 945	9	Low light following
10W	990 < x < 1000	960 < x < 970		
20W	1000 < x < 1005	960 < x < 970	10	High light following
35W	1010 < x < 1020	1005 < x < 1010		
50W	1000 < x < 1015	1000 < x <1010		

By stopping a fixed average distance in front of the light, the light following motion path of the AGV was maintained such that it overlapped with the line following motion path as illustrated in figure 8-12. The overlapping of paths ensured that the light path coincided with the line path so that when light following could not be used, due an absence of light, line following could be used.

Relocation of the docking stations was required when high light following and low light following because the AGV was programmed to stop at the junction point by using different light readings. When high light following a value of '10' was used to stop the AGV at an average distance of 0.5m from the light. When low light following a value of '9' was used to stop the AGV an average distance of 1m from the light. (Refer to table 8-4) The reasons for which are explained in section 7.3.5. The user was informed to confirm the docking station location when choosing either high or low light intensity or when the AGV had changed from high to low light following.

The docking stations had to be relocated to ensure that the AGV stopped at the correct position at the junction when changing directions. The relocation distances of docking stations are explained under the relative routes in sections 8.6.d, 8.7.e and 8.7.f. The programming for all light variation routes is in Appendix E.1.6 with the prefix LightLine. The light variation routes are explained in section 8.7.

8.5 New routes

The routes followed by the AGV were from the conveyor to the lathe, conveyor to the RM, lathe to the RM, lathe to the conveyor, RM to the lathe and RM to the conveyor. The line and light routes

were fused in order to accommodate for line following and light following, as illustrated in figure 8-11.

Three methods to fuse the path between the RM and the lathe were considered. The path had to ensure that the line and light paths overlapped for technique alternation. The path had to allow for docking and test the AGV's ability to change direction at the decision point or junction.

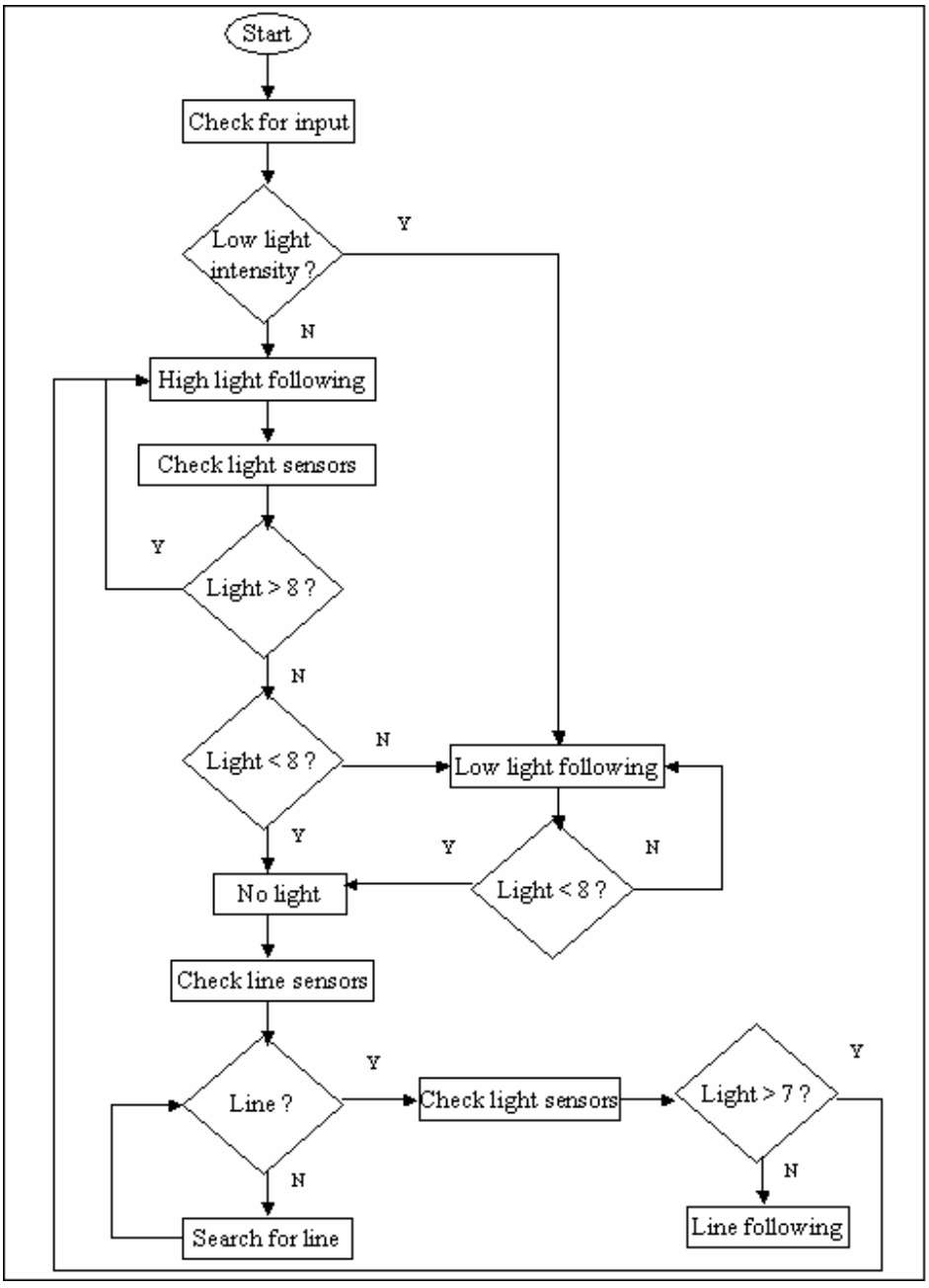


Figure 8-10 Flow chart illustrating logic flow for light variation program

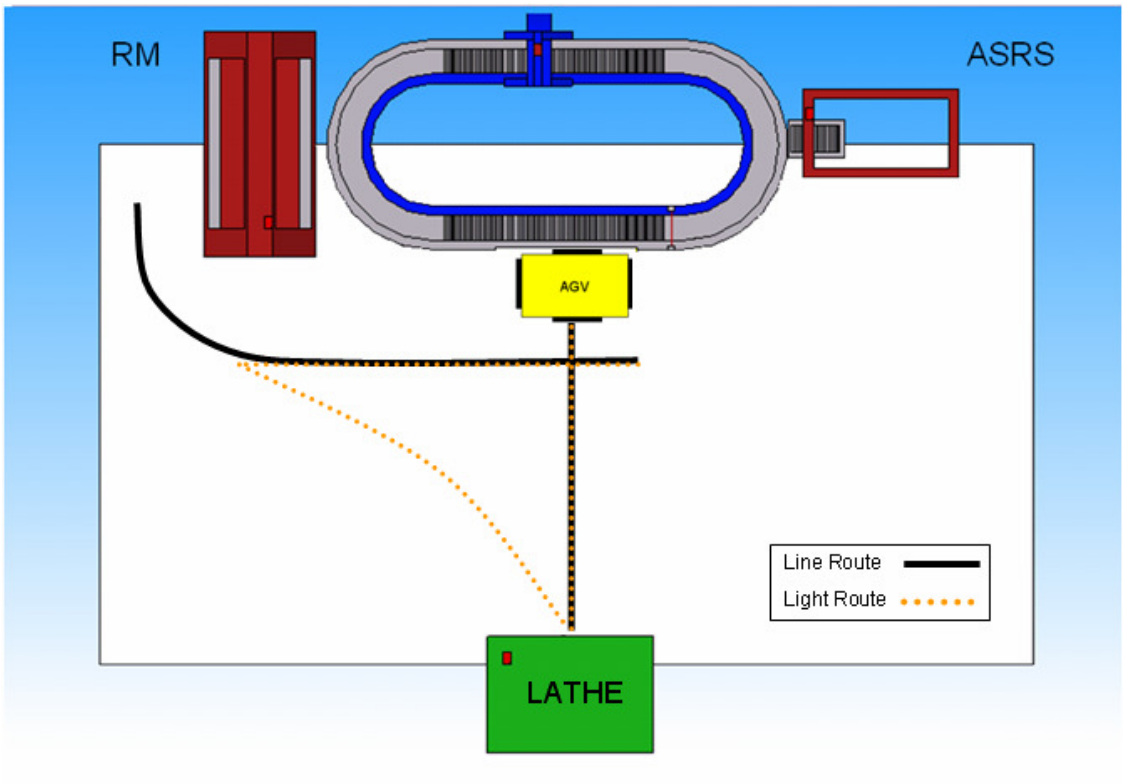


Figure 8-11 Fusion of line and light routes.

a. First method

The average light path followed by the AGV from the lathe to RM was derived from the light testing results in section 7.2.2 (figure 7-12). A line path could have been laid along the average light path, as illustrated in figure C-8, Appendix C. This would allow the AGV to move closely to the line should it need to change from light following to line following. However, the AGV's motion was uncertain when following the light between the RM and lathe in a curve. Its orientation at the end of the path was also uncertain. Docking was not possible. The ability of the AGV to change direction at the junction could not be tested.

b. Second method

A diagonal line track could have been laid between the RM and the lathe, as illustrated in figure C-9. The docking stations at the RM and the lathe could be tilted at angles to allow the light to illuminate the sensors. The AGV would also have to be tilted at an angle to allow for motion on the diagonal line. Docking would be possible. The ability of the AGV to change direction at the junction could not be tested.

c. Third method

The AGV could follow a combination of straight paths in order to navigate between the RM and the lathe. The AGV's motion on straight paths has proved reliable for both line following and light following as determined in sections 7.1.1 and 7.2.1. Docking stations could be positioned at the RM, lathe and conveyor in pre-determined orientations to ensure docking. The ability of the AGV to change direction at the junction could be tested.

The third method was applied. As illustrated in figure 8-12, the fused path between the conveyor and the lathe remained the same. The path between the RM and the conveyor was shortened by omitting the curve. It was found from testing (section 7.2.2) that the AGV did not light follow reliably along a curve.

The docking station at the end of the horizontal line path next to the conveyor had an original position of 3m from the AGV's front. The docking station under the conveyor had an original position of 0m. (It is not seen in figure 8-12 as it is located under the conveyor.) These two docking stations had to be relocated for certain routes to ensure correct routing at the junction, as explained in section 8.4. Relocation positions of the docking stations are explained under the relevant routes in sections 8.6.d, 8.7.e and 8.7.f. The docking stations at the RM and the lathe remained in the same positions.

8.6 Line variation program routes

The light was constant at all times. The line was varied.

a. Lathe to conveyor

Figure 8-13 illustrates the program layout for the routine from the lathe to the conveyor. The light was turned on at the docking station under the conveyor. The AGV started motion from the lathe in the side-right direction using line following if the line was present. If the line was not present the program changed to light following. The program checked for the line while light following and resumed line following if the line was detected. The program alternated between the two techniques as the parameters changed during travel. When the AGV docked at the conveyor, limit switch 0 was

compressed, the materials handling platform was activated, and the pallet was loaded by the piston mechanism at the conveyor.

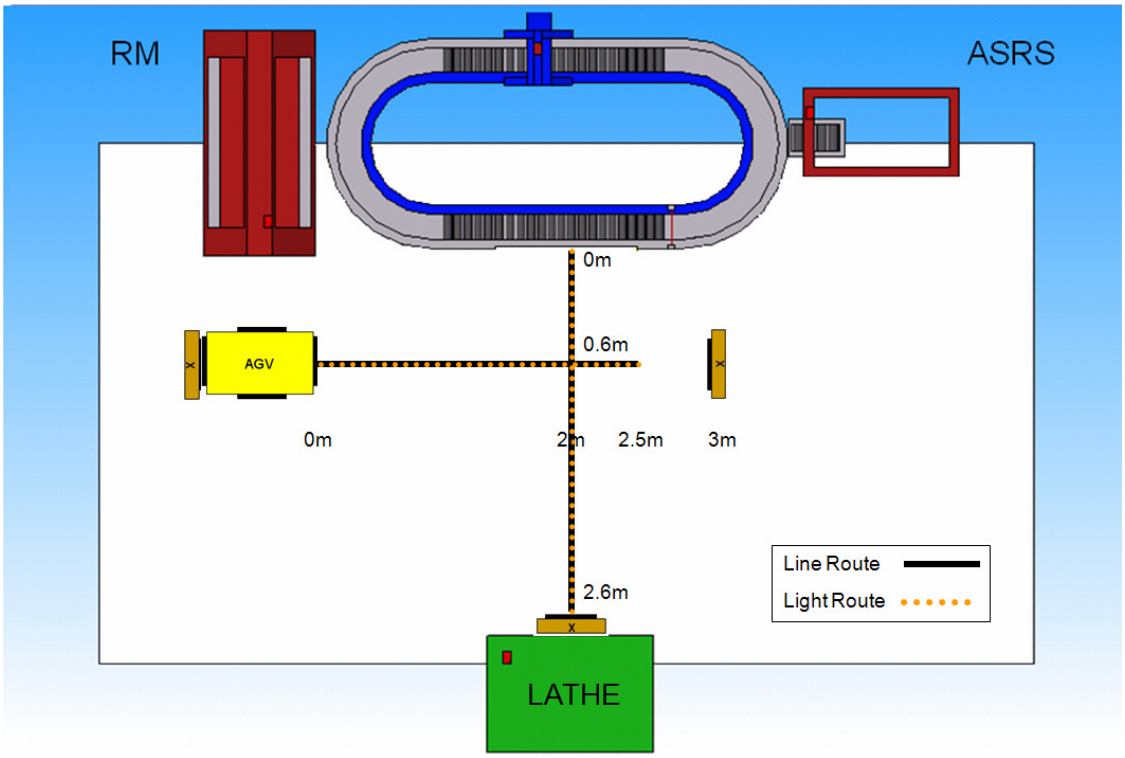


Figure 8-12 Variable sensor routes

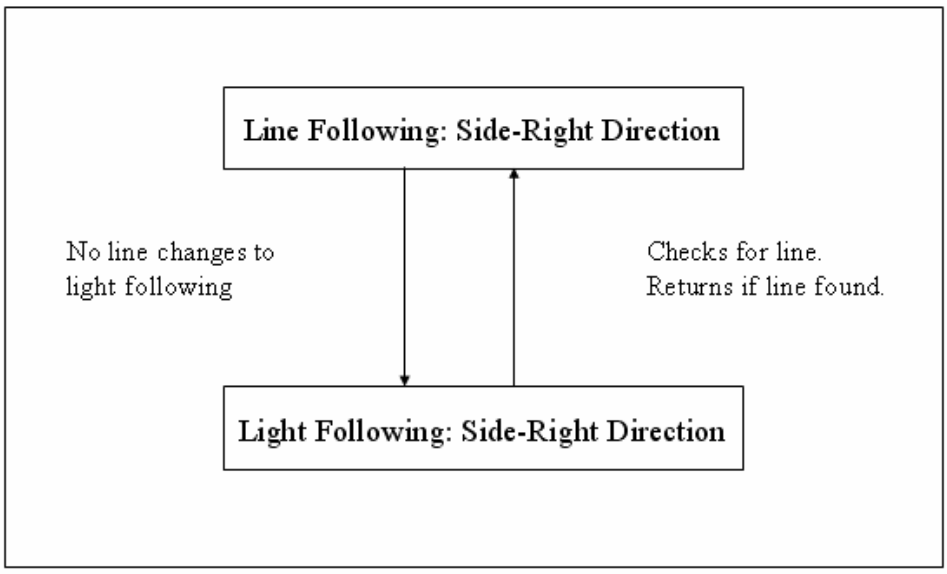


Figure 8-13 Program layout for routine from the lathe to conveyor.

b. Conveyor to lathe

Figure 8-14 illustrates the program layout for movement from the conveyor to the lathe. The light was turned on at the lathe docking station. The program was similar to the lathe to conveyor description in (a), except the AGV was moving in the side-left direction. The AGV docked on compressing limit switch 1 and manual pallet transfer occurred at the lathe.

c. Conveyor to RM

Figure 8-15 illustrates the program layout for movement from the conveyor to the RM. The light was turned on at the RM docking station. The AGV started motion in the forward direction using line following if the line was present. If the line was not present the program changed to light following. The program checked for the line while light following and resumed line following if the line was detected. The program alternated between the two techniques as the parameters changed during travel. The AGV docked at the RM docking station on compressing limit switch 8. Manual pallet transfer occurred at the RM.

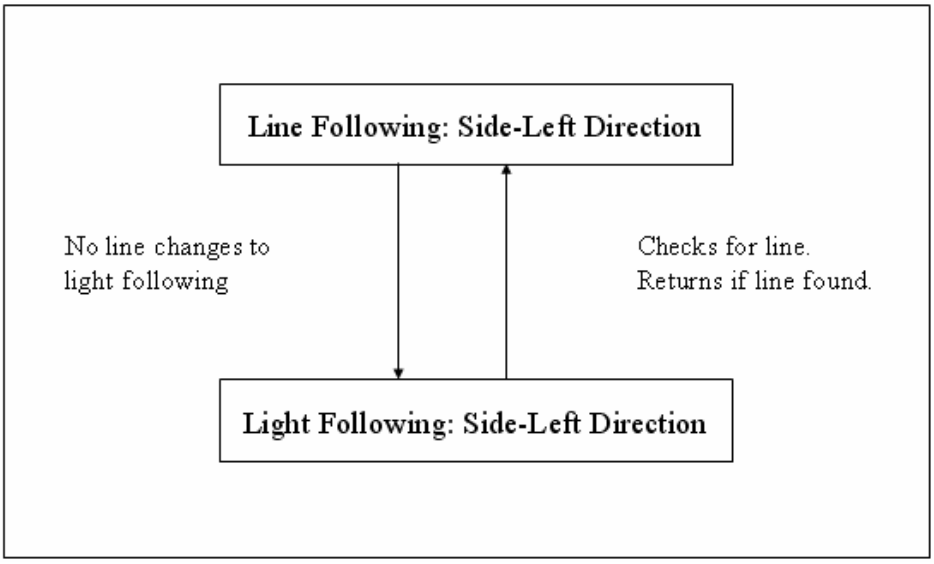


Figure 8-14 Program layout for routine from the conveyor to lathe.

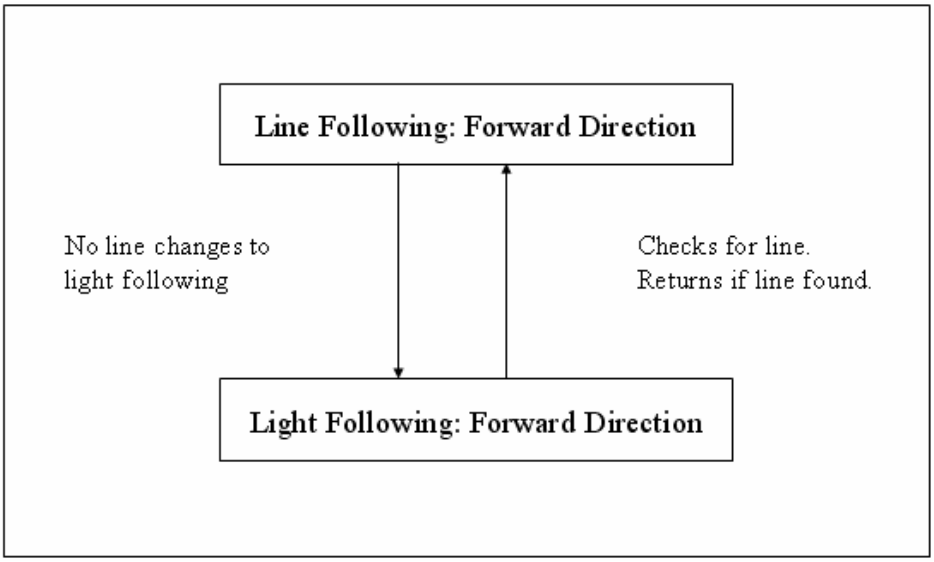


Figure 8-15 Program layout for routine from the conveyor to RM.

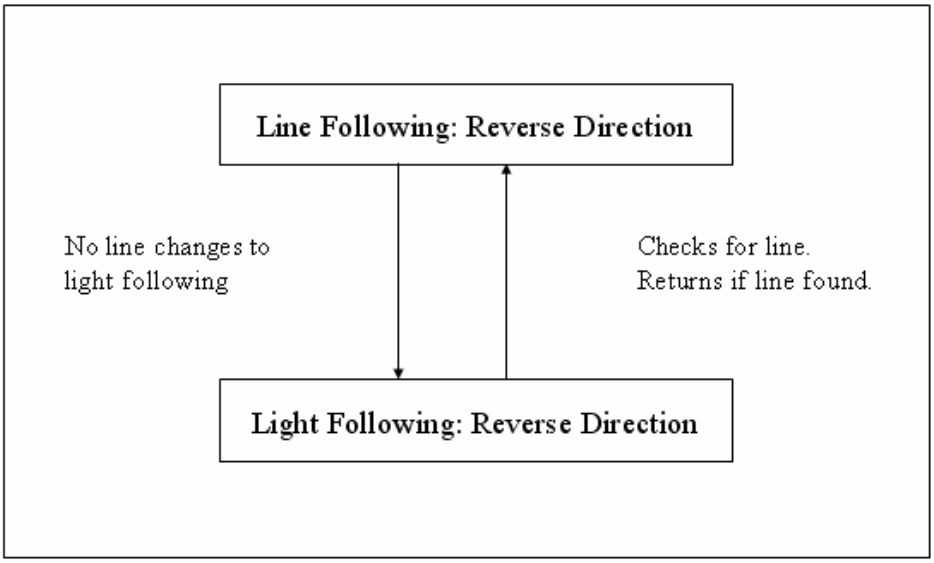


Figure 8-16 Program layout for routine from the RM to conveyor.

d. RM to Conveyor

Figure 8-16 illustrates the program layout for movement from the RM to the conveyor. The light was turned on at the docking station near the conveyor. The program was similar to the conveyor to the RM description in (c), except the AGV moved in the reverse direction. The user was prompted to confirm that the docking station was at 2.5m at the end of the line path. The AGV docked at the

docking station on compressing limit switch 8. Manual pallet transfer occurred at the conveyor for this instance as explained in section 7.2.1.b.

If the docking station was not at 2.5m, it would be at 3m in its original position (figure 8-12). At the end of the line path, the line sensors would not detect the line. The program would change to light following. The light sensors would detect the light on the docking station 0.5m away and the AGV would stop with its front at approximately 2.5m, 0.5m in front of the docking station near the conveyor. The line path ended at 2.5m for routing purposes, to ensure the AGV changed direction if it missed the side-left line in the RM to lathe route, as explained in sections 9.2.4, 9.2.5, 9.3.3, 9.3.4.

e. Lathe to RM

Figure 8-17 illustrates the program layout for movement from the lathe to the RM. The light was turned on at the docking station under the conveyor to allow side-right motion and at the RM for forward motion. The AGV started motion in the side-right direction using line following if the line was present. If the line was not present the program would change to light following. The program checked for the line while light following and resumed line following if the line was detected. The program alternated between the two techniques as the parameters changed during travel.

The AGV came to the junction, where it changed direction to forward on sensing the line in the forward direction if line following. If the AGV was light following, it would stop 0.5m in front of the light (at the docking station under the conveyor). The docking station was moved 0.1m forward for this routine. The AGV would then follow light in the forward direction.

The AGV would continue in the forward direction following the line if the line was present. If the line was not present it would change to light following in the forward direction. The program alternated between the two techniques as the parameters changed until it reached the docking station at the RM. Limit switch 8 was compressed when docking mechanisms mated, the AGV stopped and awaited manual pallet transfer.

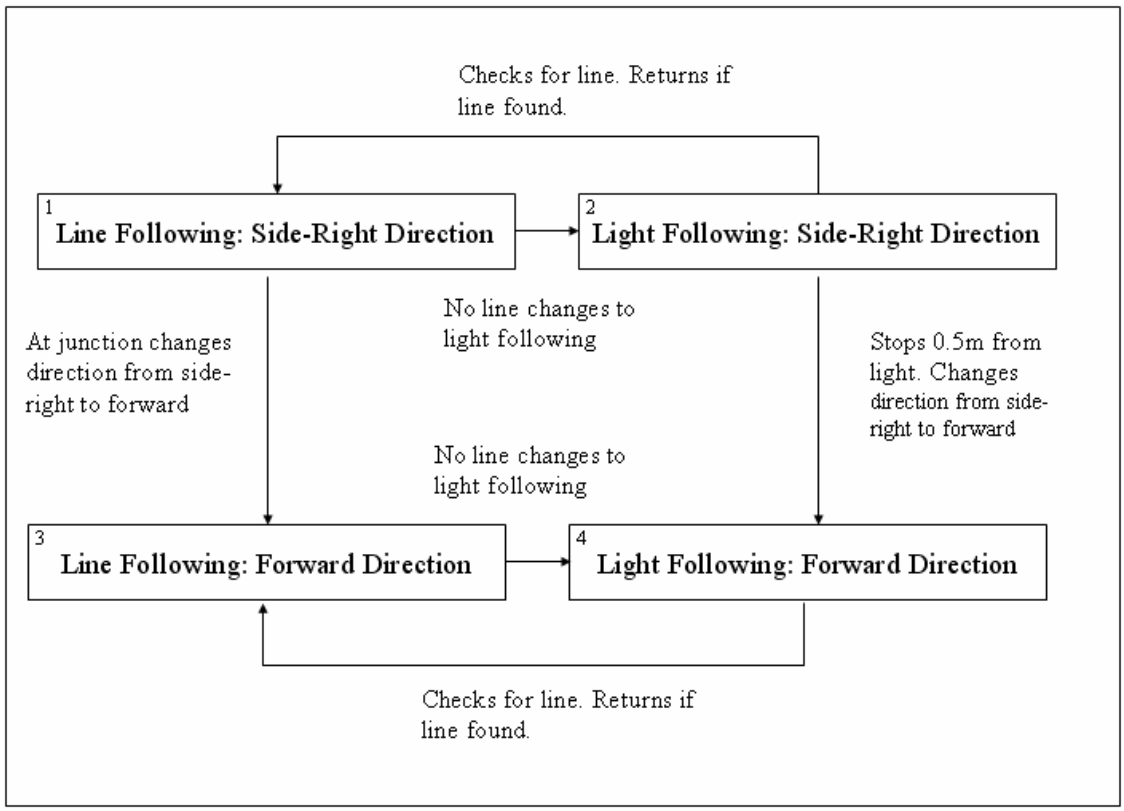


Figure 8-17 Program layout for routine from the lathe to RM.

f. RM to lathe

Figure 8-18 illustrates the program layout for movement from the RM to the lathe. The light was turned on at the docking station near the conveyor to allow the reverse motion and at the lathe for side-left motion. The AGV started motion in the reverse direction using line following if the line was present. If the line was not present the program would change to light following. The program checked for the line while light following and resumed line following if the line was detected. The program alternated between the two techniques as the parameters changed during travel.

The AGV came to the junction, where it changed direction on sensing the line in the side-left direction if line following. If the AGV was light following, at the junction, it would stop 0.5m in front of the light (at the docking station near the conveyor) and change direction to side-left.

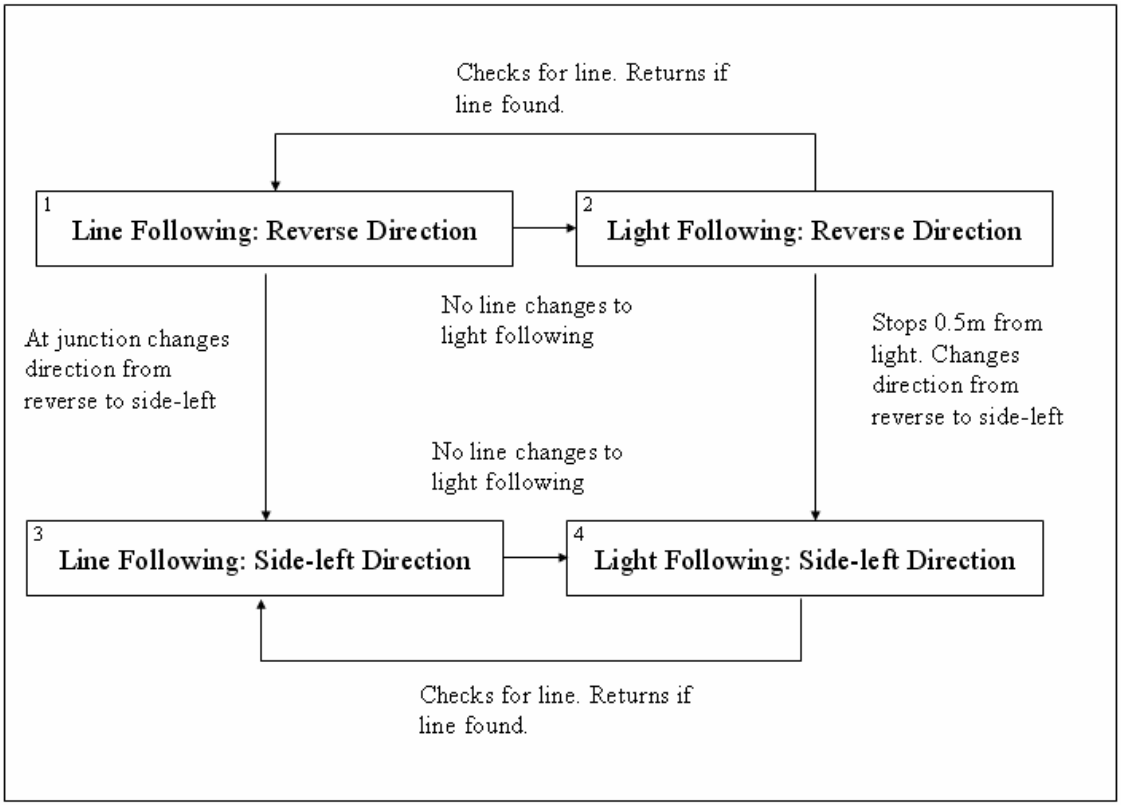


Figure 8-18 Program layout for routine from the RM to lathe.

The AGV would continue in the side-left direction following the line if the line was present. If the line was not present it would change to light following in the side-left direction. The program alternated between the two techniques as the parameters changed until it reached the docking station at the lathe. Limit switch 1 was compressed when docking mechanisms mated. The AGV stopped and awaited manual pallet transfer.

8.7 Light variation program routes

The line was constant at all times. The light intensity was varied. It is assumed that the user has chosen a high light intensity on being prompted as in figure 8-2.

a. Lathe to conveyor

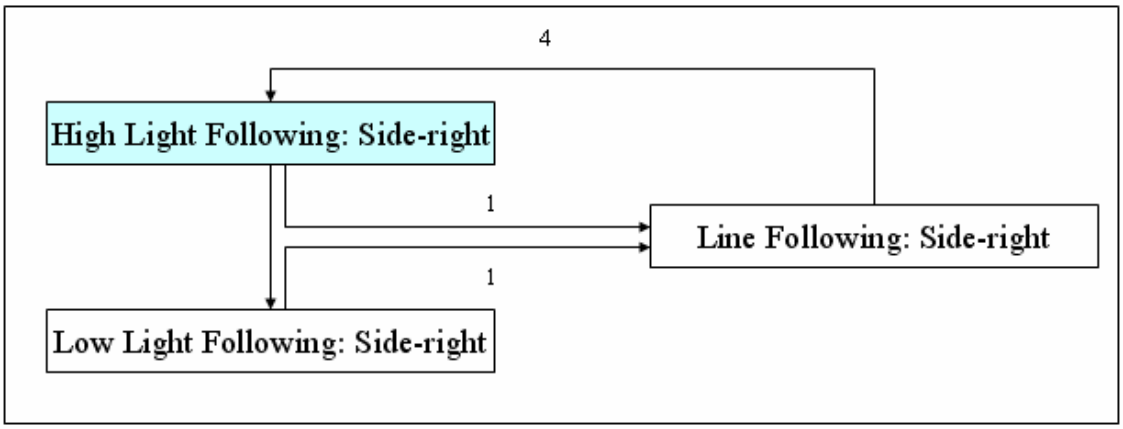


Figure 8-19 Program layout for routine from the lathe to conveyor.

Key	
	Start
1	No light, program changes to line following.
2	AGV stops 0.5m from light, and changes direction from reverse to side-left at junction.
3	Decrease in light intensity, program changes from high to low light following.
4	Program checks for light and returns to high light following if light is detected.
5	AGV stops 0.5m from light and changes direction from side-right to forward at junction.

The key explains the meaning of the numbers in figures 8-19 to 8-24. Figure 8-19 illustrates the program layout for the routine from the lathe to the conveyor. The light was turned on at the docking station under the conveyor. The AGV started motion from the lathe in the side-right direction using light following for the high light intensity (20W, 35W, 50W), if the light readings were received in the high range (8-10). If the light intensity decreased to the low range (8-9), the program changed to the low light intensity (5W, 10W).

If the light was not present (0W) the program changed to line following. The program checked for the light while line following, and resumed high light following if the light was detected again. The high light following program changed to low light following if a lower light intensity was present. The program alternated between high and low light following and line following as the parameters changed during travel. On docking at the conveyor, limit switch 0 was compressed and the AGV

stopped. The materials handling platform was activated and the pallet was loaded by the piston mechanism at the conveyor

b. Conveyor to lathe

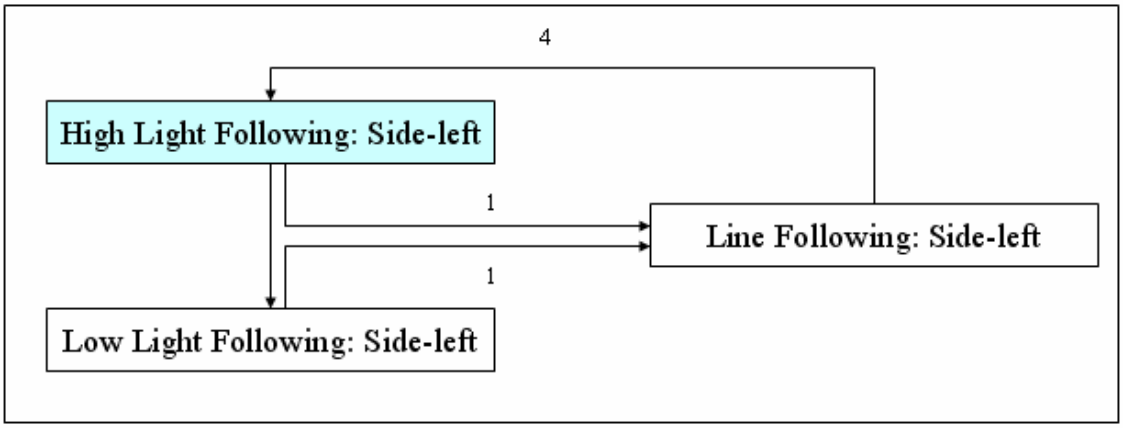


Figure 8-20 Program layout for routine from the conveyor to lathe.

Figure 8-20 illustrates the program layout for movement from the conveyor to the lathe. The light was turned on at the docking station at the lathe. The program was similar to the lathe to conveyor description (a), except the AGV was moving in the side-left direction. The AGV docked on compressing limit switch 1 and manual pallet transfer occurred at the lathe.

c. Conveyor to RM

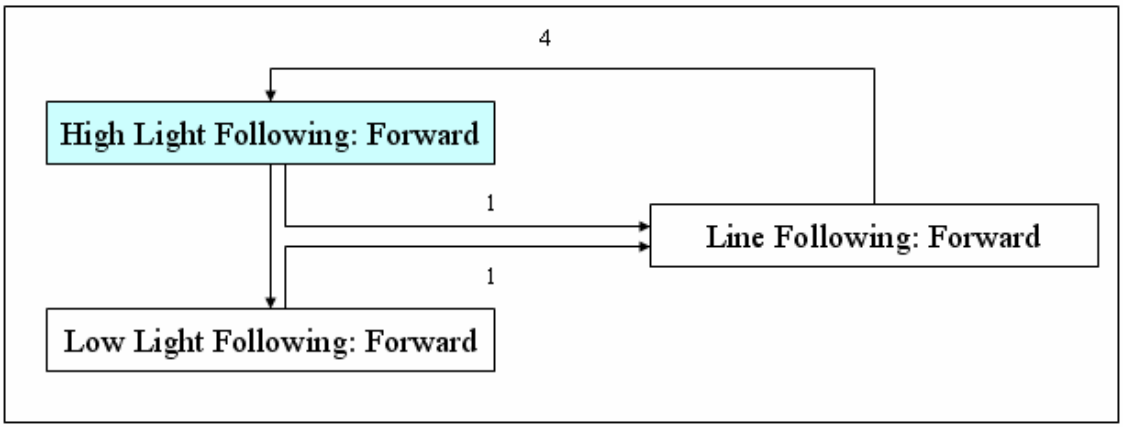


Figure 8-21 Program layout for routine from the conveyor to RM.

Figure 8-21 illustrates the program layout for movement from the conveyor to the RM. The light was turned on at the docking station at the RM. The program was similar to the lathe to conveyor description (a), except the AGV was moving in the forward direction. The AGV docked at the docking station at the RM on compressing limit switch 8. Manual pallet transfer occurred at the RM.

d. RM to Conveyor

Figure 8-22 illustrates the program layout for movement from the RM to the conveyor. The light was turned on at the docking station near the conveyor. The program was similar to the lathe to conveyor description (a), except the AGV moved in the reverse direction. The AGV docked at the docking station near the conveyor on compressing limit switch 8. Manual pallet transfer occurred at the conveyor for this instance.

e. Lathe to RM

Figure 8-23 illustrates the program layout for movement from the lathe to the RM. The conveyor docking station was moved 0.1m forward, from its original position at 0m under the conveyor, to allow stopping 0.5m in front of the light for high light following (figure 8-12). The docking station needed to be moved 0.4m backward, to allow stopping 1m in front of the light for low light following if required. Relocation of the docking station was required as explained in section 8.4.

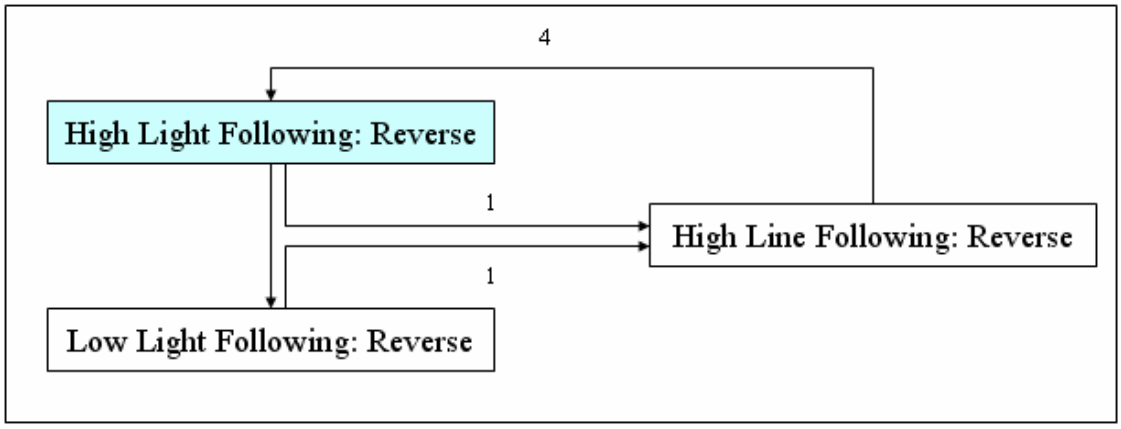


Figure 8-22 Program layout for routine from the RM to conveyor.

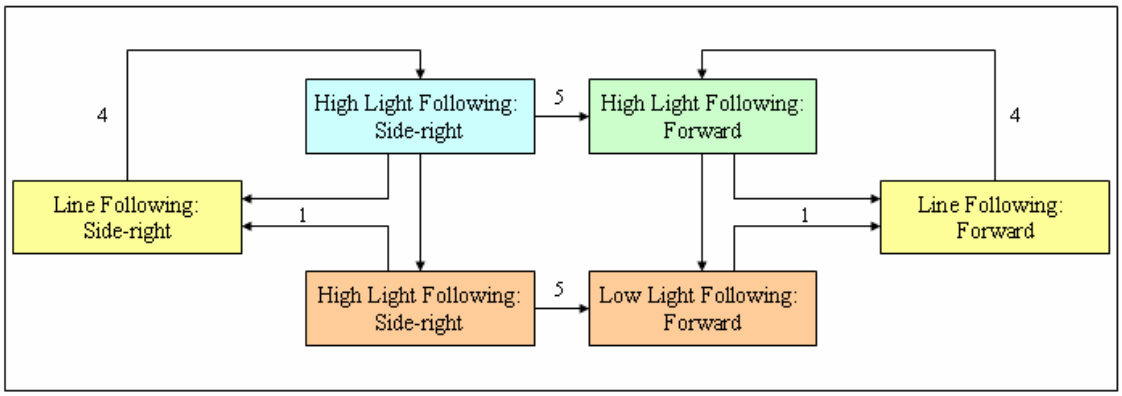


Figure 8-23 Program layout for routine from the lathe to RM.

The light was turned on at the docking station under the conveyor to allow side-right motion and at the RM for forward motion. The AGV started motion from the lathe in the side-right direction using light following for the high light intensity (20W, 35W, 50W) if the light readings were received in the high range (8-10). If the light intensity decreased to the low range (8-9) the program changed to the low light intensity (5W, 10W).

If the light was not present, 0W, the program changed to line following. The program checked for the light while line following, and resumed high light following if the light was detected again. The high light following program changed to low light following, if a lower light intensity was present. The program alternated between high and low light following, and line following techniques as the parameters changed during travel.

The AGV came to the virtual junction, where it changed direction from side-right to forward. The AGV stopped 0.5m in front of the light at the docking station under the conveyor for high light following. The AGV stopped 1m in front of the light at the docking station under the conveyor for low light following. The AGV changed direction to forward and followed the light in the forward direction.

If there was no light at the virtual junction the program would change to line following. On sensing the line in the forward direction the AGV would change direction from side-right to forward.

The AGV would continue in the forward direction using high light following, low light following or line following, depending on the light intensity present, until the docking station at the RM was

reached. Limit switch 8 was compressed when docking mechanisms mated and the AGV stopped and awaited manual transfer.

f. RM to lathe

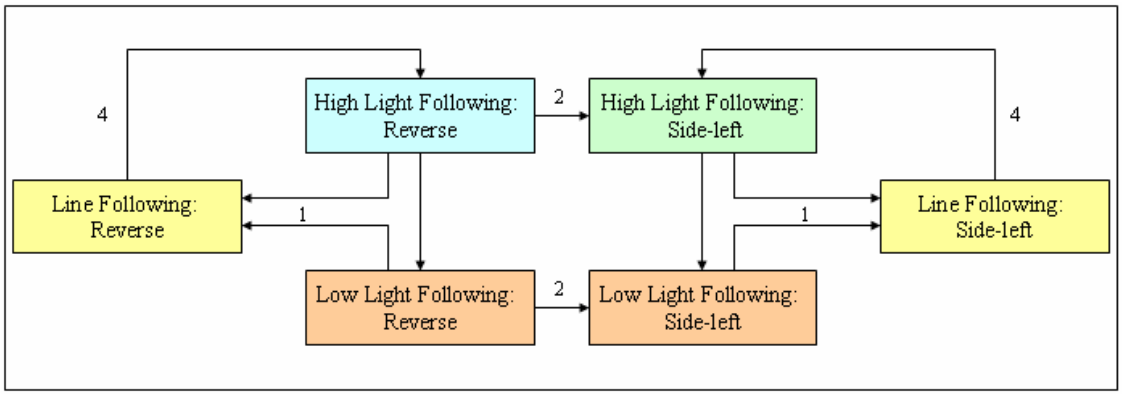


Figure 8-24 Program layout for routine from the RM to lathe.

Figure 8-24 illustrates the program layout for movement from the RM to the lathe. The docking station near the conveyor, maintained its original position at 3m, to allow stopping 0.5m in front of the light for high light following. The docking station needed to be moved to 3.5m, to allow stopping 1m in front of the light for low light following, for reasons explained in section 8.4.

The light was turned on at the docking station near the conveyor to allow the reverse motion and at the lathe for side-left motion. The AGV started motion from the RM in the reverse direction, using light following for the high light intensity (20W, 35W, 50W), if the light readings were received in the high range (8-10). If the light intensity decreased to the low range (8-9), the program changed to the low light intensity (5W, 10W).

If the light was not present (0W), the program changed to line following. The program checked for the light while line following, and resumed high light following if the light was detected. The high light following program changed to low light following if a lower light intensity was present. The program alternated between high and low light following and line following techniques as the parameters changed during travel.

The AGV came to the virtual junction, where it changed direction from reverse to side-left. The AGV stopped 0.5m in front of the light at the docking station near the conveyor for high light

following. The AGV stopped 1m in front of the light at the docking station near the conveyor for low light following. The AGV changed direction to side-left and followed the light in the side-left direction.

If there was no light at the virtual junction the program would change to line following. On sensing the line in the side-left direction the AGV would change direction from reverse to side-left.

The AGV would continue in the side-left direction using high light following, low light following or line following, depending on the light intensity present, until the docking station at the lathe was reached. Limit switch 1 was compressed when docking mechanisms mated and the AGV stopped and awaited manual transfer.

8.8 Summary

This chapter described the software for the variable sensor system that was implemented on the AGV. The user interface was presented. Program optimisation was explained. Flow charts illustrating the program logic flow were provided. Two new techniques, i.e. the line variation technique and light variation technique were introduced. These techniques are a combination of line following and light following. They are implemented under inconsistent environment conditions to enable the AGV to complete its materials handling tasks. New routes were defined. Program routes for both techniques were described for the six routes.

9 AGV Testing

In order to test the robustness and reliability of the variable sensor system, testing was conducted in three different environments:

1. High noise level environment
2. Unsatisfactory floor environment
3. Light intensity environment

9.1 High noise level environment

Industries experience high levels of noise during manufacturing. Workers have to wear ear pieces to block out the noise as a safety measure. The high noise level can hinder the use of certain sensors such as ultrasonic sensors. If the sensors can not obtain accurate readings to make decisions, it may result in collisions that will reduce production rates. In order to maintain safety in industry, sensors need to accommodate for the high noise levels.

The ultrasonic sensors were tested under high noise levels in a workshop environment. From results obtained in section 7.4, it was determined that the ultrasonic sensors functioned accurately within a 2m range, when exposed to noise from 70db to 93.5db for an individual machine, and 76.3db to 79.6db for a combination of machines. Programming accommodated for the noise exposure in the environment by setting conditions around a 1m threshold. A distance of 1m was used instead of 2m as it reduced the number of conditions required for steering and stopping of the AGV. (Appendix E)

The new ultrasonic sensor data was integrated in the line variation, light variation, line following, and light following techniques. The robot avoided obstacles accurately in the high noise level environment using any of the four guidance techniques. Table 9-1 indicates the AGV's new reaction to an object at a particular distance.

Table 9-1 AGV's new reaction to an object at a particular distance

Distance of object and Action taken	
Distance of object	**Action**
0m	Assume crashed! Stop.
< 1m	Stop. Proceed when clear.
> 1m	Proceed forward or Rotate to search for line/light.

If the object was 0m away, the program assumed that the AGV had crashed and stopped the AGV. The program prompted for user intervention to assess the status of the AGV, i.e. if the AGV could continue to operate and if any damage had been caused. If the object was less than 1m away, the AGV stopped. It checked if the path was clear and then proceeded forward. If the object was greater than 1m away, the AGV continued with line or light following in the forward direction, in the constant environment and changing environments. If the light could not be detected while using the line variation technique, the AGV rotated to find the light. If the line could not be detected while using the light variation technique, the AGV rotated to find the line.

9.2 Unsatisfactory floor environment

The line sensors depend on differentiable readings between black tape and white floor in order for the AGV to maintain accurate line following. An unsatisfactory floor environment is an environment where the tape is worn out, damaged, absent in certain parts of the track (broken track) or there is a variable, for example, an oil spill. Such conditions are not conducive for line following as sensor readings between black and white become similar (or overlap) and one cannot decipher between black and white.

In order for the AGV to execute tasks, it required an alternative technique such as the line variation technique. The line variation technique allowed the AGV to continue with its tasks in an unsatisfactory floor environment, despite the inconsistent line path, by alternating to light following.

9.2.1 Line Variation Technique Testing

To test the ability of the program to alternate between line following and light following the program was tested on the RM to lathe path. The RM to conveyor path was not tested as previous testing, in sections 7.1.1 and 7.2.1, established that successful docking occurred at the conveyor.

As illustrated in figure 9-1, the path from the RM to lathe was in the reverse and side-left direction. The AGV was not concerned with the side-right line as it was not needed for motion from the RM to lathe. Hence the cross junction became a T-junction (refer to figure 9-5) to the AGV and is referred to as such. To illustrate the T junction better, the side-right line is omitted in the testing figures that follow.

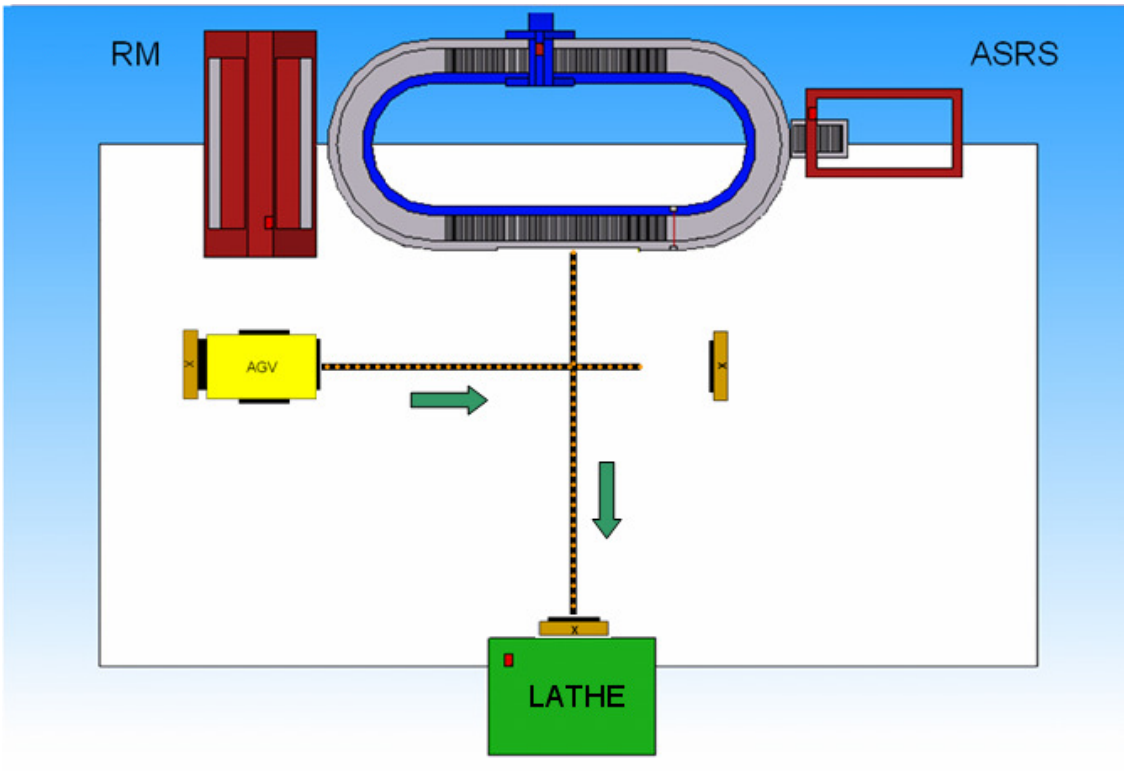


Figure 9-1 Path from the RM to the lathe indicated by arrows.

The motion of the AGV is described on the RM to lathe path utilising individual techniques in a constant environment.

a. Motion of the AGV RM to lathe utilising line following

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using line following is described below:

1. The AGV was docked at the RM docking station.
2. The AGV line followed in the reverse direction i.e. away from the RM.
3. The AGV came to the junction, where the line sensors detected the side-left line, leading towards the lathe.
4. On sensing the path to the lathe, the AGV stopped line following in the reverse direction, and started line following in the side-left direction.
5. The AGV docked at the lathe and awaited manual transfer.

b. Motion of the AGV RM to lathe utilising light following

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using light following is described below:

1. The AGV was docked at the RM docking station.
2. The AGV light followed in the reverse direction i.e. away from the RM, towards the light.
3. The AGV stopped approximately 0.5 m in front of the light as programmed.
4. The lateral light sensors detected the light in the side-left direction.
5. The AGV light followed in the side-left direction i.e. towards the lathe.
6. The AGV docked at the lathe and awaited manual transfer.

Line following and light following were combined in the line variation and light variation techniques. In the line variation technique, line following is used as a primary technique and light following as a secondary technique. In the light variation technique, light following is used as a primary technique and line following as a secondary technique.

The combined motion of the AGV utilising line following and light following for the line variation and light variation techniques is described in the testing scenarios in sections 9.2.3 to 9.2.6 and 9.3.2 to 9.3.5, respectively.

9.2.2 Environment Setup

If there was no tape or an oil spill the AGV could not line follow and would have to use light following. In order to simulate the unsatisfactory floor environment, a broken track was formed. In absent parts of the track the AGV relied on light following.

Light conditions were kept constant. A 20W halogen light was positioned at docking stations at the end of the reverse track, at 3m, and side-left track, at 2m. The light is indicated in figures by an 'x' on the docking stations. The light was on at all times during testing, as illustrated in figure 9-5. The AGV's start position was at the RM, as illustrated in figure 9-2. The light program allowed for the AGV to stop 0.5m in front of the light, which was positioned at 3m. This means the front of the AGV was at 2.5m and the centre of the AGV at 2m. This is illustrated in figures 9-3 and 9-4.

Four main scenarios were tested by varying the presence of the line in parts of the track, and at junctions required for changing directions, to reach the required work station. In all scenarios there was a 20% error of not detecting the side-left line, and 80% reliability that the side-left line would be detected as determined in section 7.1.2.

Figure 9-2 AGV at start position

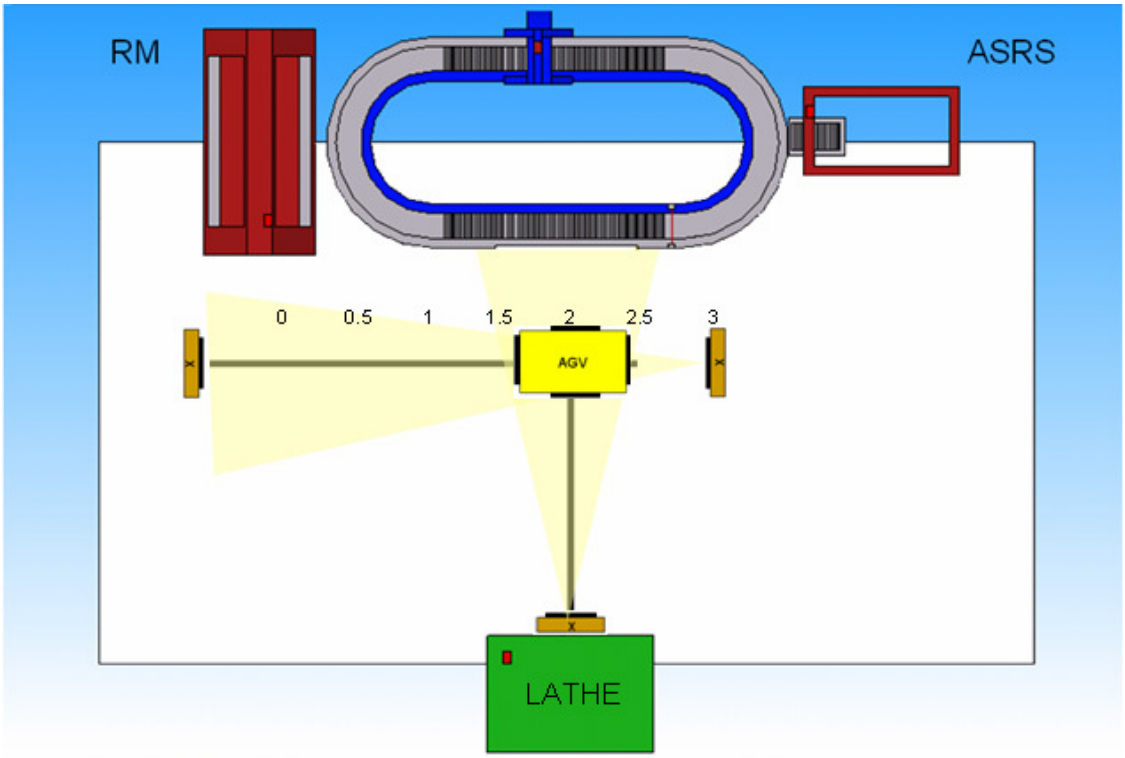


Figure 9-3 AGV at junction

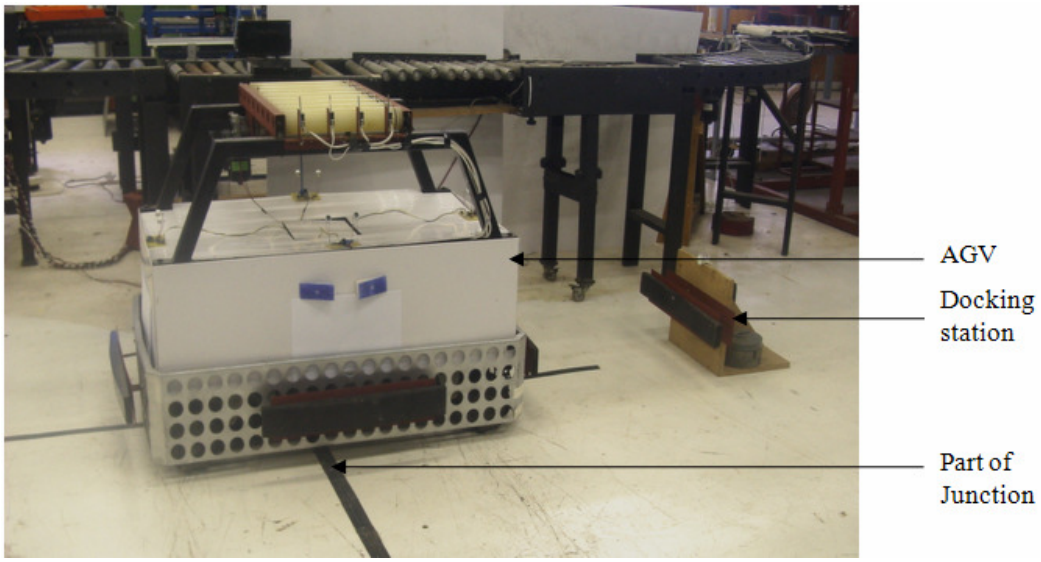


Figure 9-4 AGV at junction

9.2.3 First test scenario

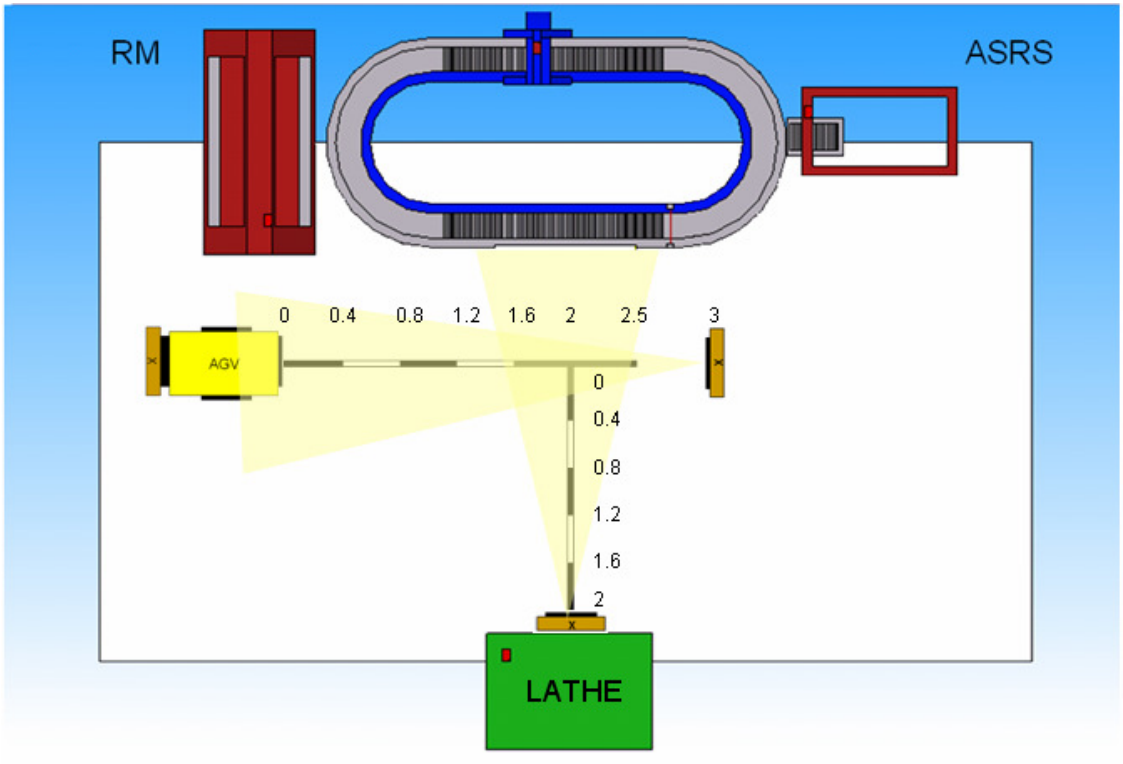


Figure 9-5 First test scenario.

The track for the first test scenario is illustrated in figure 9-5. The path consisted of 0.4m lengths of tape separated by 0.4m gaps. A T junction that was 0.9m in the reverse direction and 0.4m in the side-left direction was present. The test demonstrated the ability of the AGV to successively alternate between line and light following as the parameters varied. The AGV's ability to change direction from reverse to side-left at the complete T junction and dock at the lathe was also tested.

Motion of the AGV utilising line variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the line variation technique is described.

1. The AGV was docked at the RM docking station.
2. The AGV line followed in the reverse direction i.e. away from the RM.

3. When the reverse line sensors did not detect the line at 0.4m, the AGV started light following in the reverse direction.
4. At 0.8m the line sensors detected the line and the AGV resumed line following.
5. At 1.2m the line was not detected and the AGV light followed.
6. The line was detected at 1.6m and line following resumed.
7. At 2m the AGV continued to line follow until the centre of the AGV was above the 2m mark. The front of the AGV was at 2.5m.
8. The AGV was at the T junction (refer to figure 9-3) where the side-left line sensors detected the line leading toward the lathe, in the side-left direction.
9. On sensing the path to the lathe, the AGV stopped line following in the reverse direction, and started line following in the side-left direction i.e. towards the lathe.
10. At 0.4m in the side-left direction, the line sensors did not detect the line therefore the AGV started light following.
11. At 0.8m the line sensors detected the line and the AGV resumed line following.
12. At 1.2m the line was not detected and the AGV light followed.
13. The line was detected at 1.6m and line following resumed.
14. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-2 indicates the technique used under the different parameters for the first scenario. The distance indicated is with reference to the position of the line sensors of the AGV.

A felt pen was attached to the reverse and side-left sides of the AGV, in line with the centre sensors, and the path traced five times. Deviations of the centre sensors from the path were so minute that they were regarded as negligible. They were therefore not indicated on the graphs for the four scenarios. Offshoots from the path were measured and indicated on graphs where relevant as in scenarios 2 to 4 (sections 9.2.4 to 9.2.6).

Figure 9-6 illustrates the graph, of the average motion of the AGV, versus the path for the first scenario. The path in black indicates where the line path is present and orange indicates the average motion of the AGV. In the black sections the AGV line followed and in between the black sections the AGV light followed.

The test concluded that the AGV could alternate from line following to light following four times, in order to complete its task. The program could return to line following, from light following,

when the line was redetected four times. These are not the maximum number of times the program can alternate between line following and light following. Alternation is looped until the relevant limit switch is compressed or the ultrasonic sensors detect that the AGV has crashed.

Table 9-2 Technique used under parameter status for first scenario

Distance (m)	**Parameter status**	**Technique used:**	
	Variable: Line Constant: Light	**Line following**	**Light following**
Reverse			
0	Line	✓	
0.4	No line		✓
0.8	Line	✓	
1.2	No line		✓
1.6	Line	✓	
2	Line	✓	
T junction Centre of AGV above junction. AGV changes direction.			
Side-left			
0	Line	✓	
0.4	No line		✓
0.8	Line	✓	
1.2	No line		✓
1.6	Line	✓	
2	Docks	✓	

9.2.4 Second test scenario

Figure 9-7 illustrates the second test scenario. The second track consisted of 0.5m lengths of tape, separated by 0.5m gaps. The horizontal part of the T junction was present. The vertical part of the T junction was absent. The test demonstrated the ability of the AGV to change direction from reverse to side-left, with only the horizontal part of the T junction present.

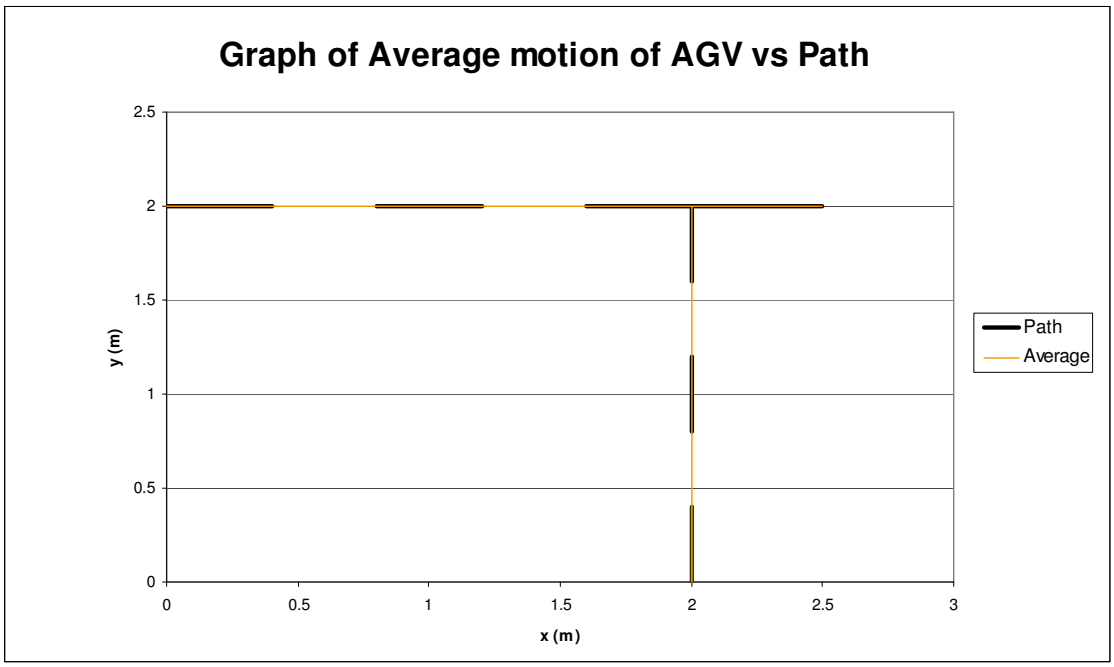


Figure 9-6 Graph of Average Motion of the AGV versus Path.

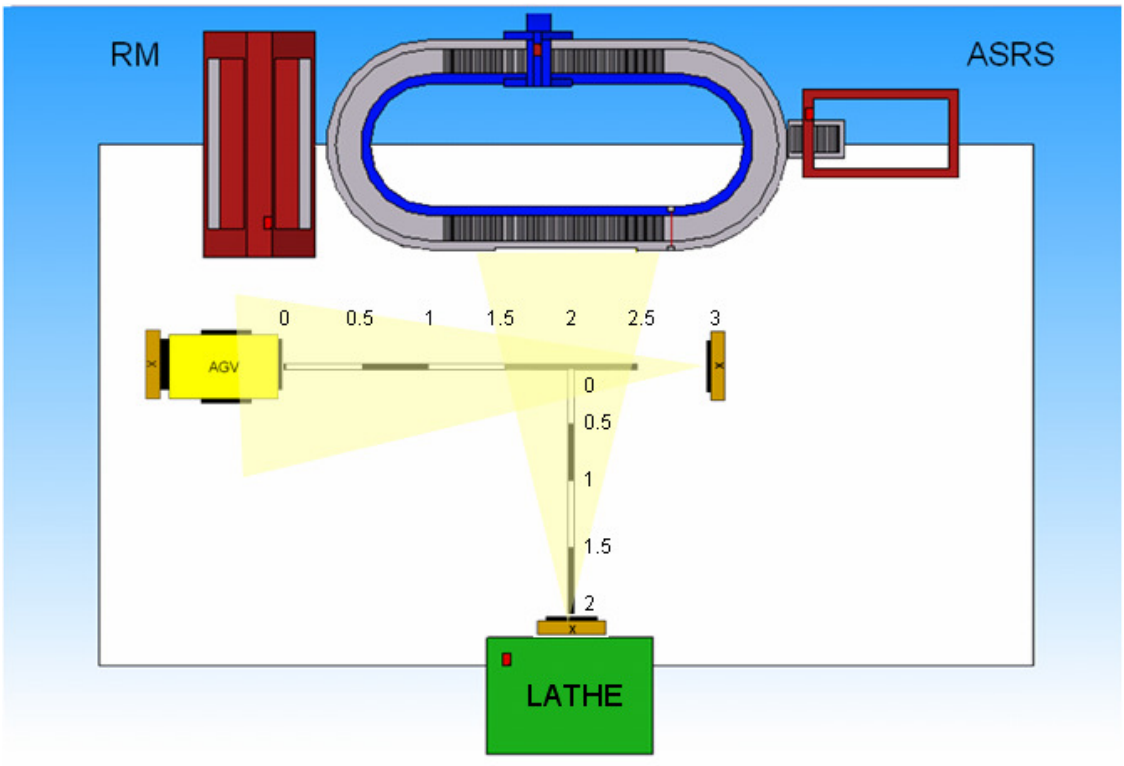


Figure 9-7 Second test scenario.

Motion of the AGV utilising line variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the line variation technique is described.

1. The AGV was docked at the RM docking station.
2. As there was no line present at 0m, the AGV automatically switched to light following.
3. The reverse line sensors detected the line at 0.5m, hence the AGV started line following.
4. At 1m the line was not detected and the AGV started light following.
5. At 1.5m the line sensors detected the line and the AGV resumed line following.
6. At 2m the AGV continued to line follow. The centre of the AGV reached the 2m mark. The front of the AGV was at 2.5m.
7. The AGV was at the T junction where it changed direction from reverse to side-left. However, as there was no line to indicate a change of direction to side-left, the AGV continued to line follow in the reverse direction.
8. At 2.5m the AGV proceeded over the line.
9. When the line was not detected, the AGV started light following. On detecting the light approximately 0.5m away, the AGV stopped as programmed. The average offshoot of the centre reverse sensor from 2.5m is indicated in table 9-4. This was obtained by attaching a felt pen to the front of the AGV, in line with the centre sensor, and measuring the offshoot traced five times.
10. The light sensors detected the light in the side-left direction, i.e. towards the lathe. The AGV proceeded side-left and corrected itself as it moved such that the light was in the centre of the AGV. Hence the AGV tended towards the line.
11. At 0.5m the side-left line sensors detected the line and the AGV started line following in the side-left direction.
12. At 1m the line was not detected and the AGV started light following.
13. At 1.5m the line sensors detected the line and the AGV resumed line following.
14. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-3 indicates the technique used under the different parameters for the second scenario.

Table 9-3 Technique used under parameter status for second scenario

Distance (m)	**Parameter status** **Variable: Line Constant: Light**	**Technique used:**	
		Line following	**Light following**
Reverse			
0	No line		✓
0.5	Line	✓	
1	No line		✓
1.5	Line	✓	
2	Line	✓	
2.5	No line		✓
T junction Centre of AGV above junction. No line in side-left direction. AGV changes direction using light following.			
Side-left			
0	No line		✓
0.5	Line	✓	
1	No line		✓
1.5	Line	✓	
2	Docks	✓	

Table 9-4 Offshoot results and average

Test	**Distance (m)**	**Offshoot (m)**
1	2.520	0.020
2	2.518	0.018
3	2.517	0.017
4	2.516	0.016
5	2.518	0.018
Average	**2.518**	**0.018**

The AGV was tested as in scenario 1, with a felt pen five times. Figure 9-8 illustrates the graph of the average motion of the AGV, versus the path. The test concluded that the AGV changed direction from reverse to side-left, with only the horizontal part of the T junction present, with an offshoot of 0.018m, indicated in green on the graph. This was assisted by the light following

technique as explained in point (10). The AGV was able to transport the pallet from the RM to lathe.

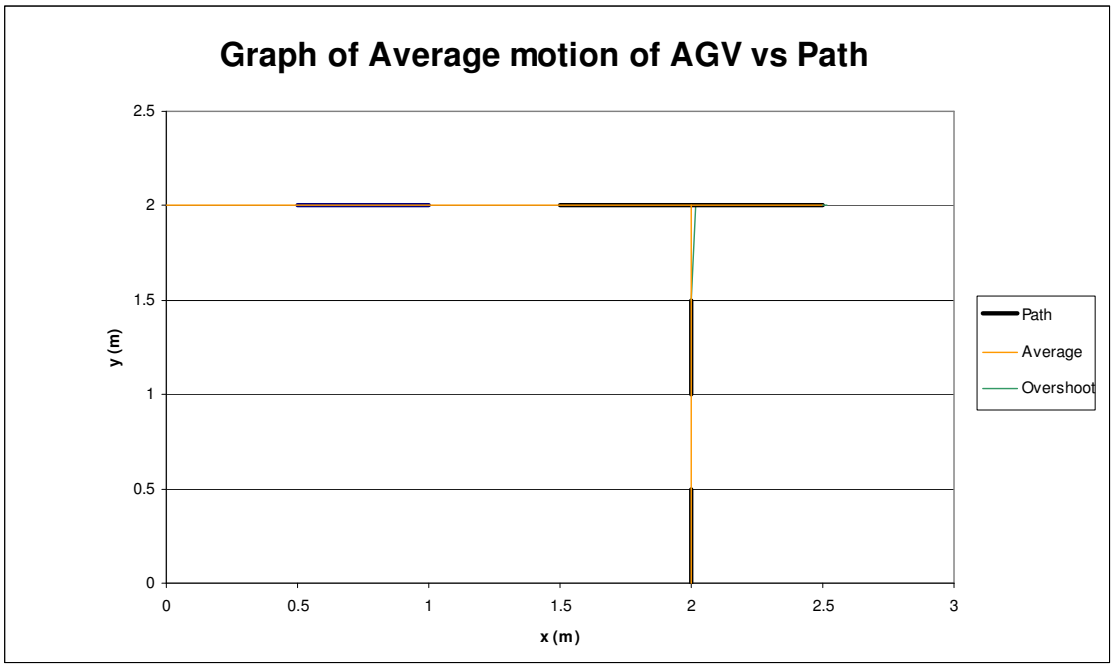


Figure 9-8 Graph of Average Motion of the AGV versus Path.

9.2.5 Third test scenario

Figure 9-9 illustrates the third test scenario. The third track consisted of 0.5m lengths of tape separated by 0.5m gaps. The vertical part of the T junction was present. The horizontal part of the T junction was absent. The test demonstrated the ability of the AGV to change direction from reverse to side-left, with only the vertical part of the T junction present.

Motion of the AGV utilising line variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the line variation technique is described.

1. The AGV was docked at the RM docking station.

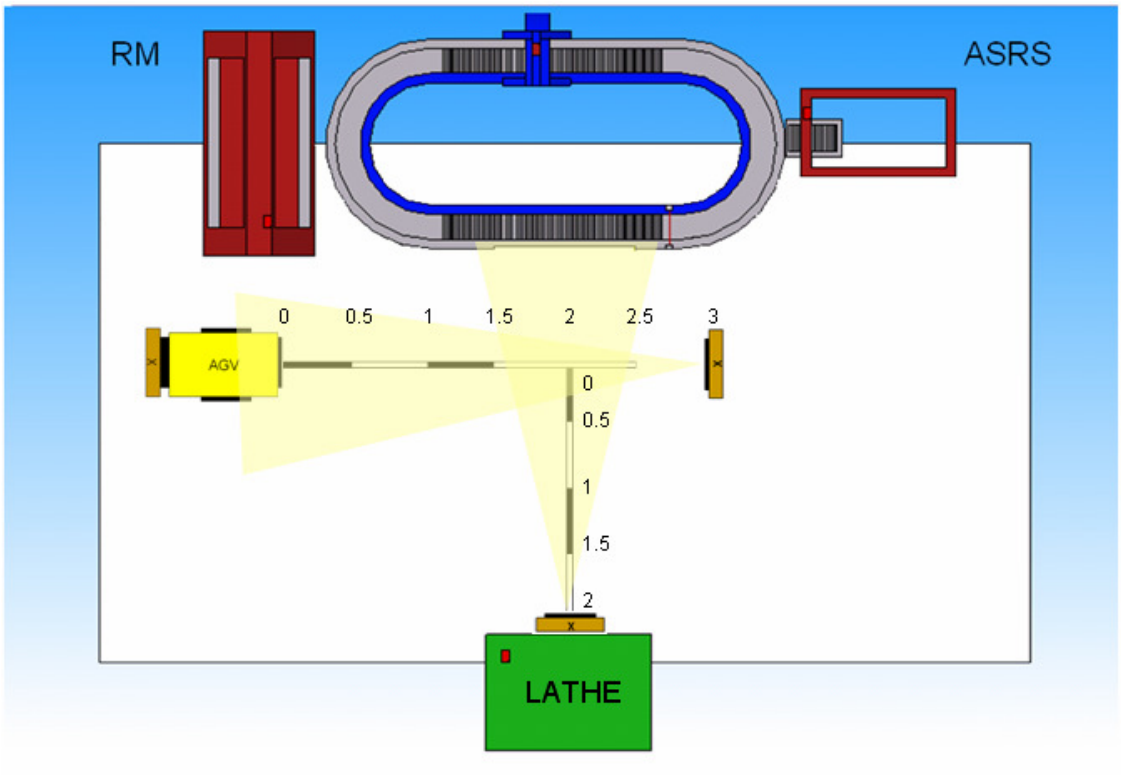


Figure 9-9 Third test scenario.

2. The AGV line followed in the reverse direction i.e. away from the RM.
3. When the reverse line sensors did not detect the line at 0.5m, the AGV started light following.
4. At 1m the line sensors detected the line and the AGV resumed line following.
5. At 1.5m the line was not detected and the AGV light followed.
6. At 2m the AGV continued to light follow. The centre of the AGV reached the 2m mark. The front of the AGV was at 2.5m.
7. The AGV was at the T junction where the AGV changed direction from reverse to side-left. The program accommodated for two occurrences:

a. The line sensors detected the line in the side-left direction and started line following in the side-left direction.

b. The AGV passed the side-left line while light following. On detecting light 0.5m away in the reverse direction, the AGV stopped at approximately 2.5m. On sensing light in the side-left direction, the AGV started light following in the side-left direction. When the line sensors detected the line, the AGV started line following in the side-left direction.

8. At 0.5m the line sensors did not detect the line in the side-left direction so the AGV started light following.
9. At 1m the line sensors detected the line and the AGV resumed line following.
10. At 1.5m the line was not detected and the AGV light followed.
11. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-5 indicates the technique used under the different parameters for the third scenario. The test concluded that the AGV changed direction from reverse to side-left, with only the vertical part of the T junction present. This was assisted by the light following technique, if the line was not detected, as in situation (7.b). There was a 20% chance that the line would not be detected, as determined in section 7.2.1. The average offshoot of not detecting the side-left line to change direction was 0.016m, and is indicated in table 9-6. Figure 9-10 illustrates the graph of the average motion of the AGV, versus the path for situations (7.a) and (7.b). (7.a) is indicated in orange and (7.b) in green.

Table 9-5 Technique used under parameter status for third scenario

Distance (m)	**Parameter status**	**Technique used:**	
	Variable: Line Constant: Light	**Line following**	**Light following**
Reverse			
0	Line	✓	
0.5	No line		✓
1	Line	✓	
1.5	No line		✓
2	No line		✓
2.5	No line		✓
T junction Centre of AGV above junction. Line in side-left direction. Refer to point (7) above.			
Side-left			
0	Line	✓ (7.a)	✓ (7.b)
0.5	No line		✓
1	Line	✓	
1.5	No line		✓
2	Docks		✓

Table 9-6 Offshoot of not detecting line, results and average

Attempt	Distance (m)	Offshoot (m)
1	2.517	0.017
2	2.518	0.018
3	2.517	0.017
4	2.520	0.020
5	2.516	0.016
Average	**2.516**	**0.016**

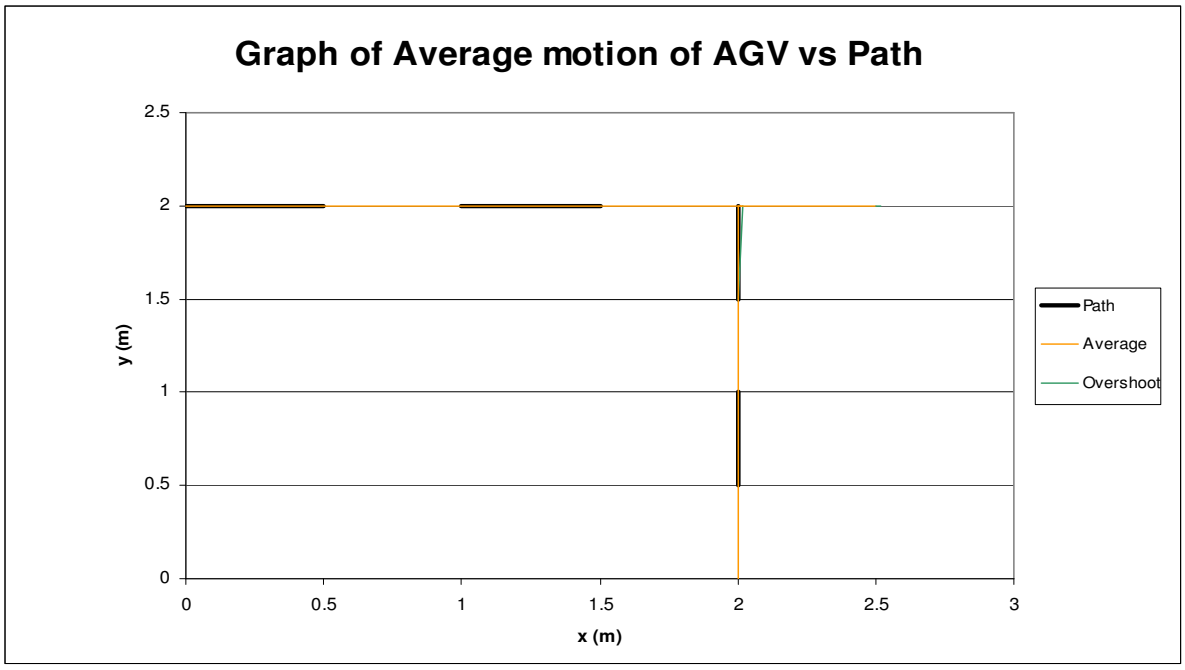


Figure 9-10 Graph of Average Motion of the AGV versus Path.

9.2.6 Fourth test scenario

Figure 9-11 illustrates the fourth test scenario. The fourth track consisted of 0.5m lengths of tape separated by 0.5m gaps. The T junction was absent. The test demonstrated the ability of the AGV to change direction from reverse to side-left, with no T junction.

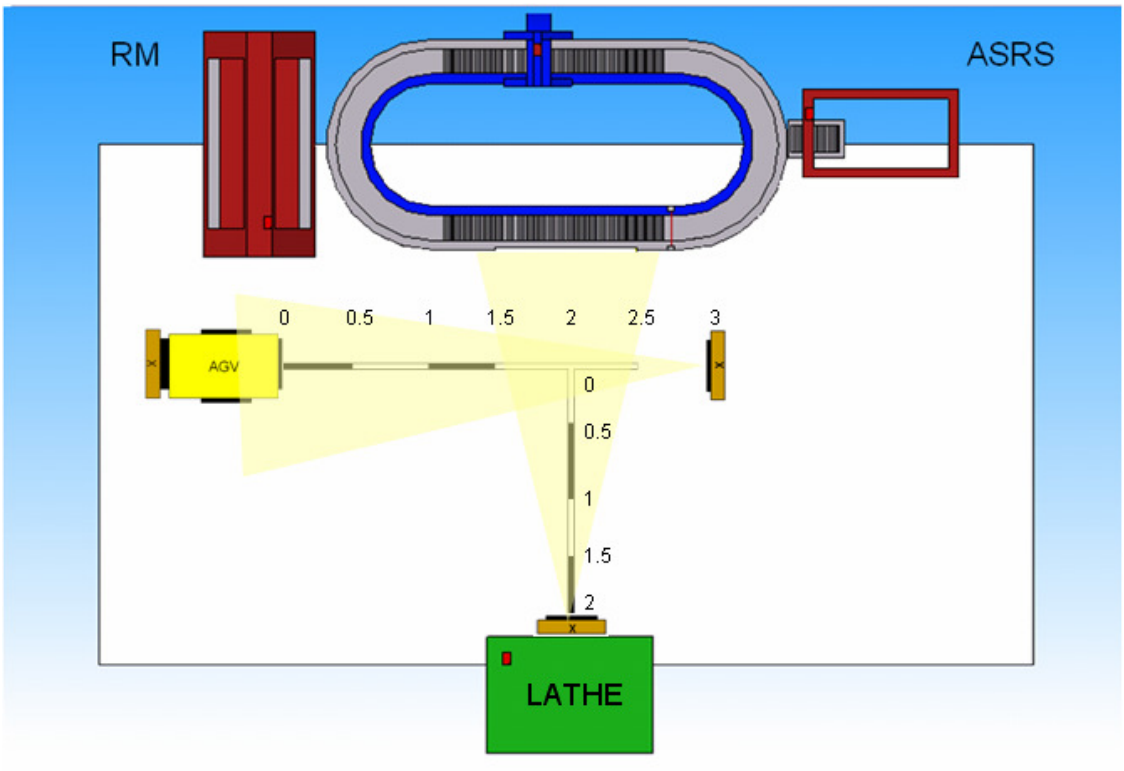


Figure 9-11 Fourth test scenario.

Motion of the AGV utilising line variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the line variation technique is described.

1. The AGV was docked at the RM docking station.
2. The AGV line followed in the reverse direction i.e. away from the RM.
3. When the reverse line sensors did not detect the line at 0.5m, the AGV started light following.
4. At 1m the line sensors detected the line, and the AGV resumed line following.
5. At 1.5m the line was not detected and the AGV light followed.
6. At 2m the AGV continued to light follow. The centre of the AGV reached the 2m mark. The front of the AGV was at 2.5m.
7. The AGV was at the virtual T junction, where the AGV changed direction from reverse to side-left, but there were no lines present to indicate a direction change.

8. The AGV stopped at an average of 2.5m, 0.5 m in front of the light as programmed, using light following. The average stopping distance was calculated from the results obtained in tables 9-8 and 9-9.
9. The side-left light sensors detected the light in the side-left direction.
10. The AGV light followed in the side-left direction. While light following, the AGV tended towards the light and hence towards the line.
11. At 0.5m the line sensors detected the line and the AGV started line following in the side-left direction.
12. At 1m the line was not detected and the AGV started light following.
13. At 1.5m the line sensors detected the line and the AGV resumed line following.
14. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-7 indicates the technique used under the different parameters for the fourth scenario.

Table 9-7 Technique used under parameter status for fourth scenario

Distance (m)	**Parameter status** **Variable: Line Constant: Light**	**Technique used:**	
		Line following	**Light following**
Reverse			
0	Line	✓	
0.5	No line		✓
1	Line	✓	
1.5	No line		✓
2	No line		✓
2.5	No line		✓
Virtual T junction AGV stops 0.5m in front of light in revere direction and changes to side-left on detecting light in the side-left direction.			
Side-left			
0	No line		✓
0.5	Line	✓	
1	No line		✓
1.5	Line	✓	
2	Docks	✓	

Table 9-8 Average stopping distance and undershoot from light in reverse direction with reference to front of AGV.

Attempt	**Stopping distance (m)**	**Undershoot from light (m)**
1	2.46	0.04
2	2.5	0
3	2.45	0.05
4	2.5	0
5	2.47	0.03
Average	**2.48**	**0.024**

Table 9-9 Average stopping distance and overshoot from light in reverse direction.

Attempt	**Stopping distance (m)**	**Overshoot from light (m)**
1	2.54	0.04
2	2.52	0.02
3	2.54	0.04
4	2.5	0
5	2.5	0
Average	**2.52**	**0.02**

Tables 9-8 and 9-9 indicate the stopping distance, undershoot and overshoot from the light. The values in the tables were used to calculate the average stopping distance and average deviation from the light:

Average stopping distance = (2.48 + 2.52) / 2 = 2.5m

Average deviation = (0.024 + 0.02) / 2 = 0.022 m

Figure 9-12 illustrates the graph of the average motion of the AGV, versus the path. The overshoot of 0.02 m is indicated in green. The undershoot of 0.024m is indicated in yellow. The test concluded that the AGV changed direction from reverse to side-left when the T junction was not present. This was assisted by the light following technique as explained from points (8) to (10).

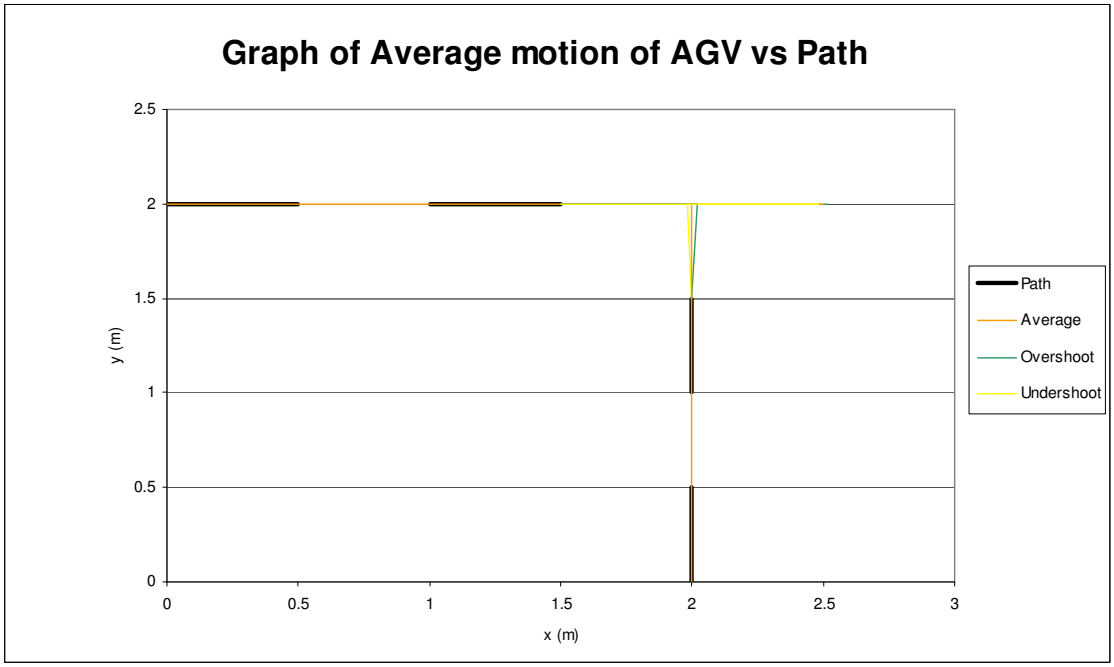


Figure 9-12 Graph of Average Motion of the AGV versus Path.

9.3 Light intensity environment

This is an environment where the light intensity is varied from a high (20W), to low (5W), to no light (0W), intensity. The light variation technique was tested in the light intensity environment.

9.3.1 Light variation technique testing

Testing determined the AGV's flexibility to light follow under the different light intensities and to alternate to line following, if there was no light, in order to continue with task completion.

a. High to low light intensity testing

To test the ability of the program to change from high to low intensity light following, the AGV was instructed to light follow from the RM to the lathe. (This path is illustrated in figure 9-1.) When the AGV was light following using the high light following program with a 20W bulb, the 5W bulb was switched on, and the 20W bulb switched off. The AGV's control program changed from the high light following program to the low light following program, allowing the AGV to continue

light following. This was indicated by sensor feedback and a user message as in figure C-10, in Appendix C.

User messages appeared to indicate that the docking station positions (sections 8.7.e and 8.7.f) needed to be confirmed for high and low light following. The user was prompted until the docking station was in the correct position, as illustrated in figure C-11. The AGV only continued with motion if the correct docking station position was confirmed. Incorrect positioning of the docking stations would prevent the AGV from reaching the correct destination for the RM-lathe routes.

The 20W and 5W bulbs were changed for testing purposes as they produced a different light intensity. In industry changing from high to low light following can be used when bulbs decrease in intensity, or when a high intensity bulb is replaced by a low intensity bulb. The technique accommodates for the decrease in light intensity and allows the AGV to continue to operate.

b. Zero Watt Testing

The light variation technique allowed the AGV to operate if there was no light. The program changed from high or low light following to line following in the absence of light. The program returned to high light following if light was detected again, and then to low light following if a low light intensity was present (refer to figure 8-10). The program was tested in the light intensity environment by switching off the light in certain parts of the track. A constant line path was always present. The test scenario is illustrated in figure 9-13. Four main scenarios were tested.

9.3.2 First test scenario

The 20W light was switched off in 0.4m intervals. Figure 9-14 illustrates where the light was switched off (black line) and switched on (orange dotted line). The test demonstrated the ability of the AGV to successively alternate between light and line following as the parameters varied. It also tested the AGV's ability to change direction from reverse to side-left at the virtual T junction and dock at the lathe.

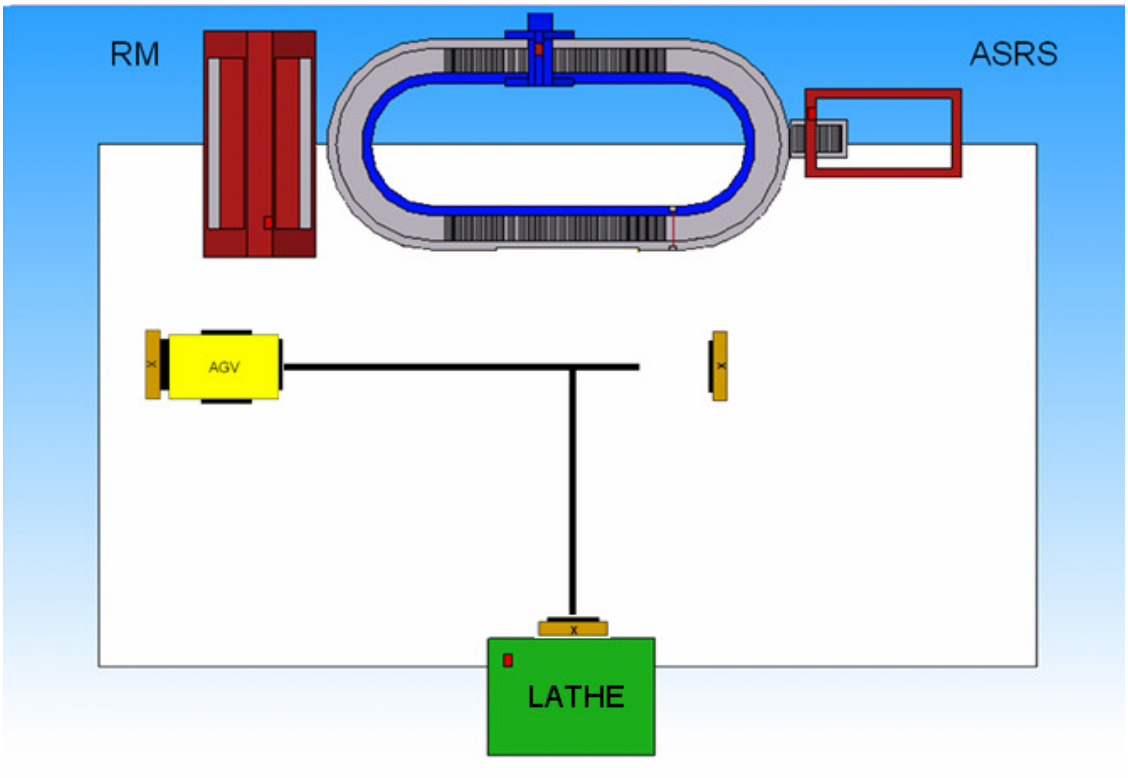


Figure 9-13 Test scenario (constant line)

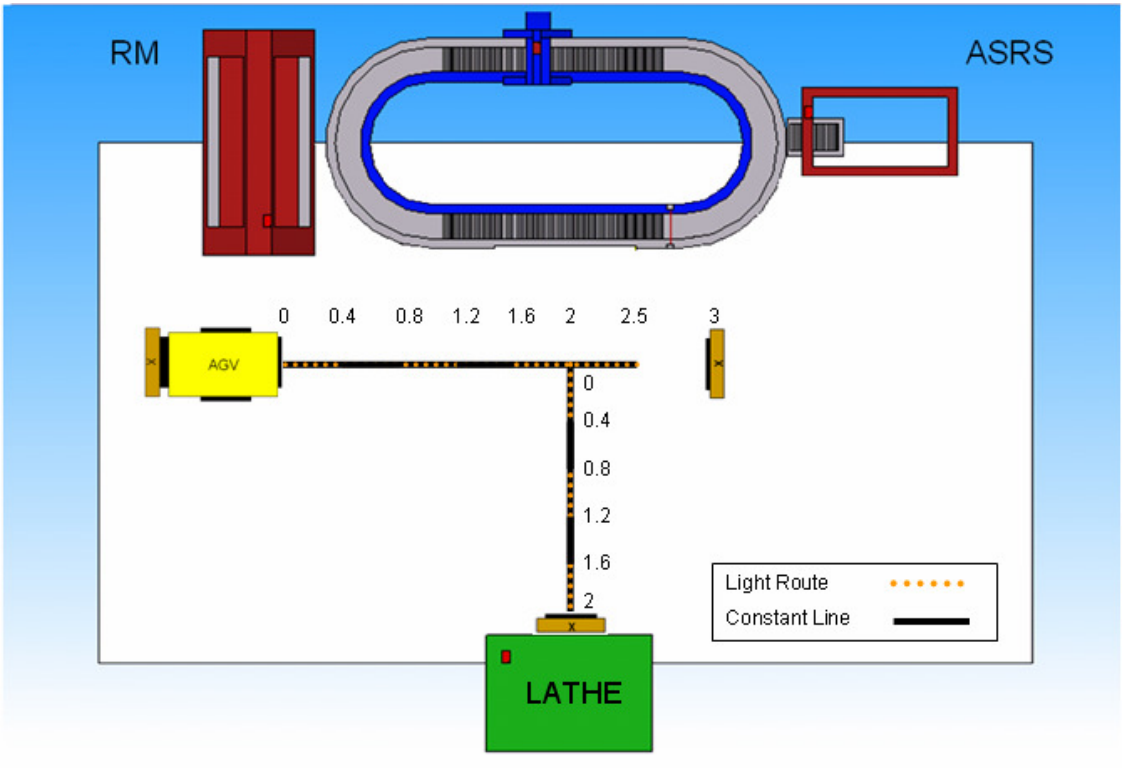


Figure 9-14 First test scenario

Motion of the AGV utilising light variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the light variation technique is described.

1. The AGV was docked at the RM docking station.
2. The AGV light followed in the reverse direction i.e. away from the RM.
3. When the light sensors did not detect the light at 0.4m, the AGV started line following.
4. At 0.8m the light sensors detected the light and the AGV resumed light following.
5. At 1.2m the light was not detected and the AGV line followed.
6. The light was detected at 1.6m and the AGV resumed light following.
7. At 2m the AGV continued to light follow. The centre of the AGV reached the 2m mark. The front of the AGV was at 2.5m.
8. The AGV was at the virtual T junction where the AGV changed direction from reverse to side-left.
9. The reverse light sensors detected the light positioned at 3m. The AGV stopped at an average of 2.5m, 0.5m in front of the light as programmed. The average stopping distance was calculated from the results obtained in tables 9-11 and 9-12.
10. The side-left light sensors detected the light in the side-left direction.
11. The AGV light followed in the side-left direction i.e. towards the lathe where another light was positioned. While light following, the AGV tended towards the light and hence towards the line.
12. At 0.4m in the side-left direction, the light was not detected by the light sensors so the AGV started line following.
13. At 0.8m the light sensors detected the light and the AGV resumed light following.
14. At 1.2m the light was not detected and the AGV line followed.
15. The light was detected at 1.6m and light following resumed.
16. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-10 indicates the technique used under the different parameters for the first scenario.

Table 9-10 Technique used under parameter status for first scenario

Distance (m)	Parameter status Variable: Light Constant: Line	Technique used:	
		Light following	Line following
Reverse			
0	Light	✓	
0.4	No light		✓
0.8	Light	✓	
1.2	No light		✓
1.6	Light	✓	
2	Light	✓	
2.5	Light	✓	
Virtual T junction AGV stops 0.5m in front of light in revere direction and changes to side-left on detecting light in the side-left direction.			
Side-left			
0	Light	✓	
0.4	No light		✓
0.8	Light	✓	
1.2	No light		✓
1.6	Light	✓	
2	Docks	✓	

Table 9-11 Average stopping distance and undershoot from light in reverse direction.

Attempt	Stopping distance (m)	Undershoot from light at 3m (m)
1	2.45	0.05
2	2.5	0
3	2.5	0
4	2.48	0.02
5	2.47	0.03
Average	**2.48**	**0.02**

Table 9-12 Average stopping distance and overshoot from light in reverse direction.

Attempt	**Stopping distance (m)**	**Overshoot from light at 2.5m (m)**
1	2.5	0
2	2.52	0.02
3	2.54	0.04
4	2.5	0
5	2.55	0.05
Average	**2.52**	**0.022**

Tables 9-11 and 9-12 indicate the stopping distance, undershoot and overshoot from the light. The values in the tables were used to calculate the average stopping distance and average deviation from the light:

Average stopping distance = (2.48 + 2.52) / 2 = 2.5 m
Average deviation = (0.02 + 0.022) / 2 = 0.021 m

The average deviation of 0.021m obtained in test scenario 1, light variation testing, correlated to the average deviation of 0.022m, obtained in test scenario 4, line variation testing (section 9.2.6). A felt pen was attached to the reverse and side-left sides of the AGV, in line with the centre line sensors, and the path traced five times. This position corresponded to the reverse and side-left centres, and the centre of the light sensors. Deviations of the centre from the path were so minute that they were regarded as negligible. They were therefore not indicated on the graphs for the four scenarios. Offshoots from the path were measured and indicated where relevant as in scenarios 2 to 4 (sections 9.3.4 to 9.3.6).

Figure 9-15 illustrates the graph of the average motion of the AGV, versus the path. The path in orange indicates where the light is present, and blue indicates the average motion of the AGV. In the orange sections the AGV light followed, and in between the orange sections the AGV line followed. Green indicates the overshoot of 0.022m. Yellow indicates the undershoot of 0.02m.

The test concluded that the AGV could alternate from light following to line following four times, in order to complete its task. The program could return to light following from line following when the light was redetected four times. As mentioned in section 9.2.3 these are not the maximum

number of times the program can alternate between light following and line following. The AGV was able to change direction from reverse to side-left at the virtual T junction and dock at the lathe.

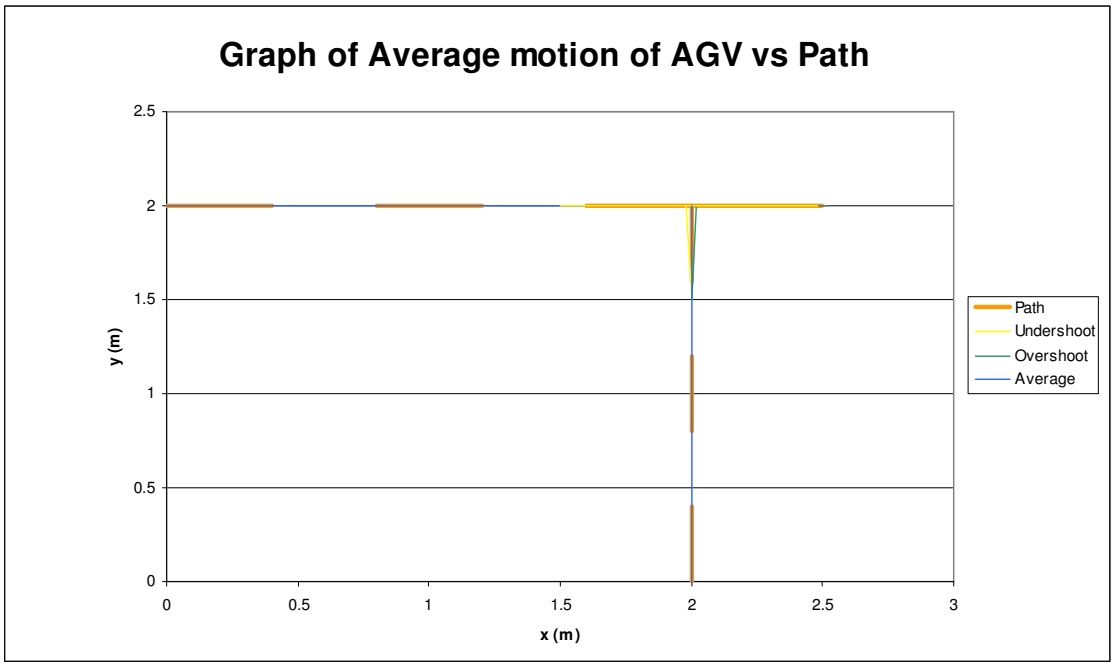


Figure 9-15 Graph of Average Motion of the AGV versus Path.

9.3.3 Second test scenario

Figure 9-16 illustrates the second test scenario. The 20W light was switched off from 0m, in 0.5m intervals. The test demonstrated the AGV's ability to change direction from reverse to side-left at the virtual T junction with no light in the side-left direction.

Motion of the AGV utilising light variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the light variation technique is described.

1. The AGV was docked at the RM docking station.
2. There was no light detected at 0m, therefore the AGV automatically switched to line following.
3. The light sensors detected the light at 0.5m hence the AGV started light following.
4. At 1m the light was not detected and the AGV started line following.

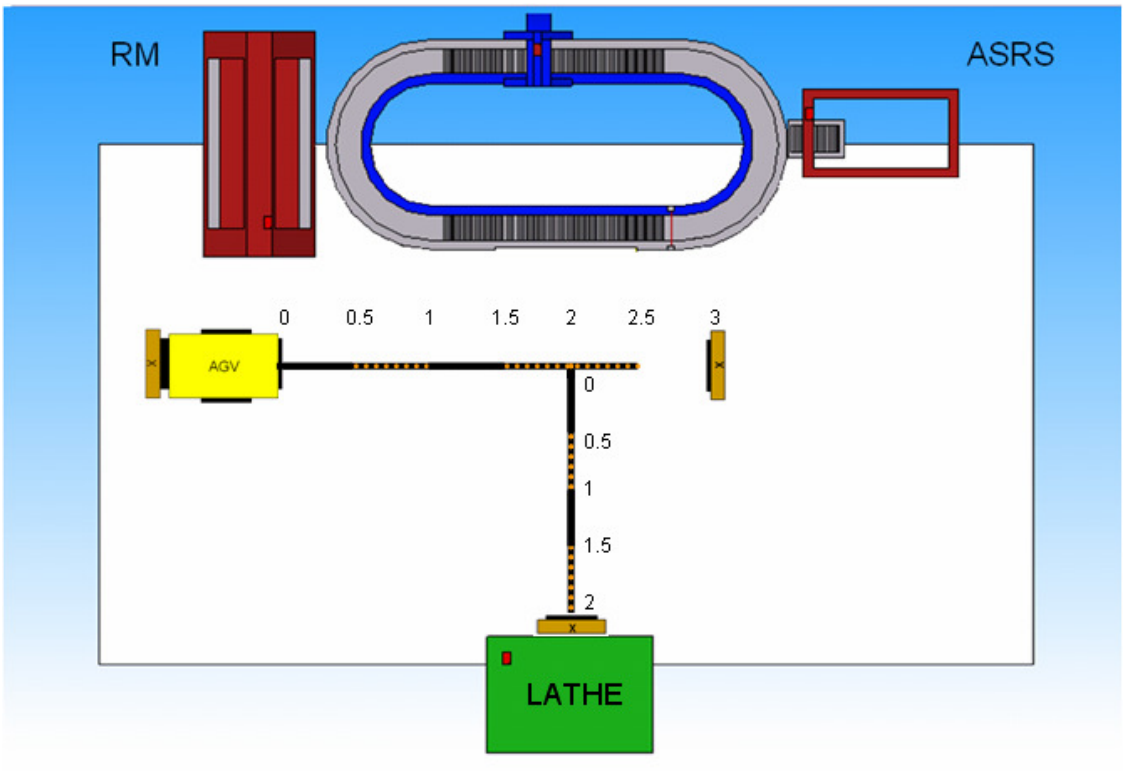


Figure 9-16 Second test scenario.

5. At 1.5m the light sensors detected the light and the AGV resumed light following.
6. At 2m the AGV continued to light follow. The centre of the AGV reached the 2m mark. The front of the AGV was at 2.5m.
7. The AGV was at the virtual T junction where it changed direction from reverse to side-left.
8. The reverse light sensors detected the light positioned at 3m. The AGV stopped at an average of 2.5m, 0.5 m in front of the light as programmed.
9. The side-left light sensors did not detect the light in the side-left direction as the light was switched off. Therefore the AGV changed to line following. The program accommodated for two occurrences:

a. The AGV detected the line in the side-left direction and started line following towards the lathe in a lateral direction. The AGV docked in the lateral direction, as illustrated in figure 9-17.

b. The AGV did not detect the side-left line and started to search for the line by moving in a clockwise direction. After turning 90 degrees, the AGV detected the line and proceeded towards

the lathe. The AGV was now in a longitudinal direction but docking was still possible, as illustrated in figure 9-18.

10. At 0.5m the light sensors detected the light and the AGV started light following in the side-left direction.
11. At 1m the light was not detected and the AGV started line following.
12. At 1.5m the light sensors detected the light and the AGV resumed light following.
13. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-13 indicates the technique used under the different parameters for the second scenario.

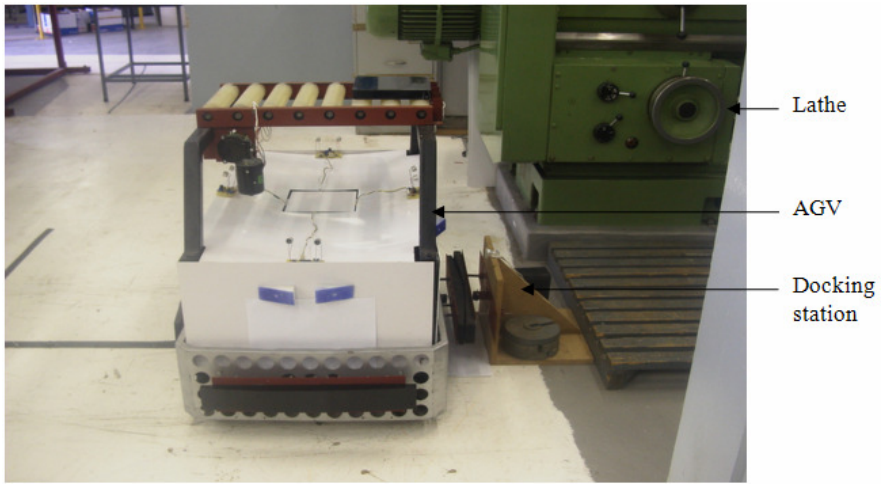


Figure 9-17 AGV docking in lateral direction at lathe.

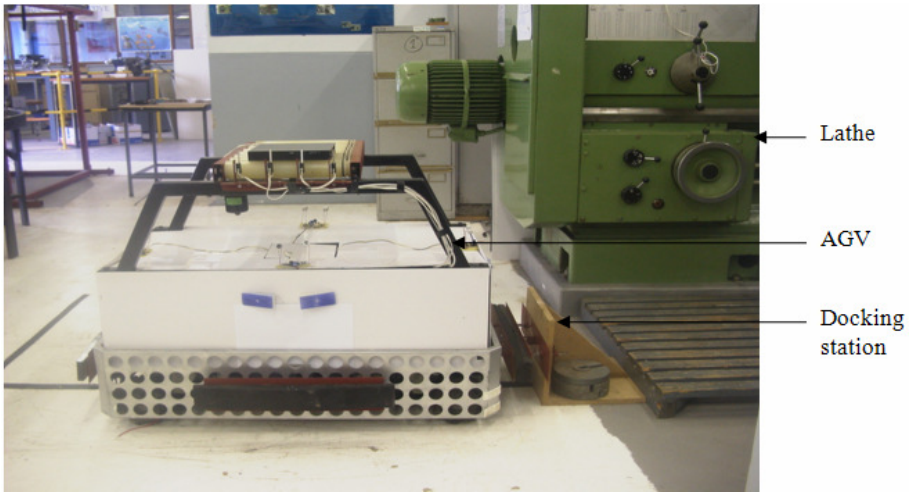


Figure 9-18 AGV docking in longitudinal direction at lathe.

Table 9-13 Technique used under parameter status for second scenario

Distance (m)	**Parameter status** **Variable: Light Constant: Line**	**Technique used:**	
		Light following	**Line following**
Reverse			
0	No light		✓
0.5	Light	✓	
1	No light		✓
1.5	Light	✓	
2	Light	✓	
2.5	Light	✓	
Virtual T junction AGV stops 0.5m in front of light in reverse direction. No light side-left direction. Changes to line following.			
Side-left			
0	No light		✓
0.5	Light	✓	
1	No light		✓
1.5	Light	✓	
2	Docks	✓	

(9.b) had a 20% chance of occurring as determined in section 7.1.2. The light was positioned such that the AGV would stop within the width of the side-left line, 5.4cm. A maximum offshoot and undershoot of 2.7cm was accommodated for from the centre of the line. The average offshoot of the centre line sensor from the centre of the line was calculated to be 2.1 cm. The three sensors covered a distance of 7cm. In this way at least one of the three sensors would be over the line. The side-left line could be detected by at least one of the line sensors thus allowing the AGV to continue motion.

There was a possibility of the AGV missing the line completely, when searching for it by rotating, and detecting it in the opposite direction i.e. a full 180 degree turn. The AGV would have to be re-instructed to complete the task in such a case.

Figure 9-19 illustrates the graph of the average motion of the AGV, versus the path. The test concluded that the AGV changed direction from reverse to side-left at the virtual T junction with no

light in the side-left direction. This was assisted by the line following technique when the light was not detected in the side-left direction as explained in point (9).

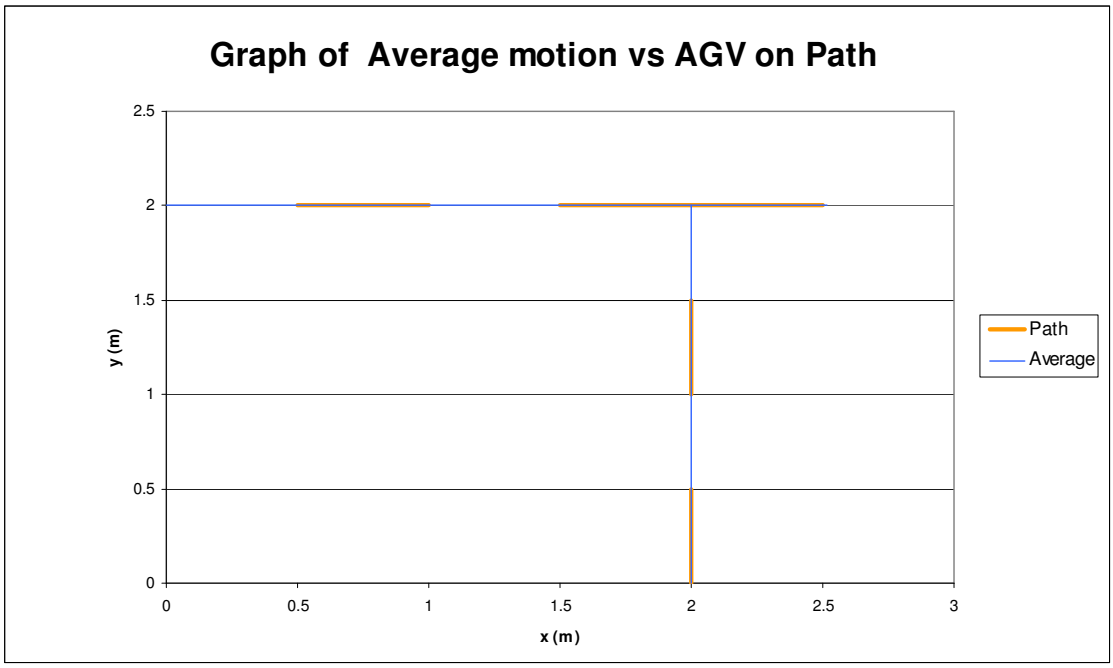


Figure 9-19 Graph of Average Motion of the AGV versus Path.

9.3.4 Third test scenario

Figure 9-20 illustrates the third test scenario. The 20W light was switched on at 0m and 1m. The light was switched off at 0.5m and 1.5m. The test demonstrated the AGV's ability to change direction from reverse to side-left at the virtual T junction with no light in the reverse direction.

Motion of the AGV utilising light variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the light variation technique is described.

1. The AGV was docked at the RM docking station.
2. The AGV light followed in the reverse direction i.e. away from the RM.
3. When the reverse light sensors did not detect the light at 0.5m the AGV started line following.
4. At 1m the light sensors detected the light and the AGV resumed light following.

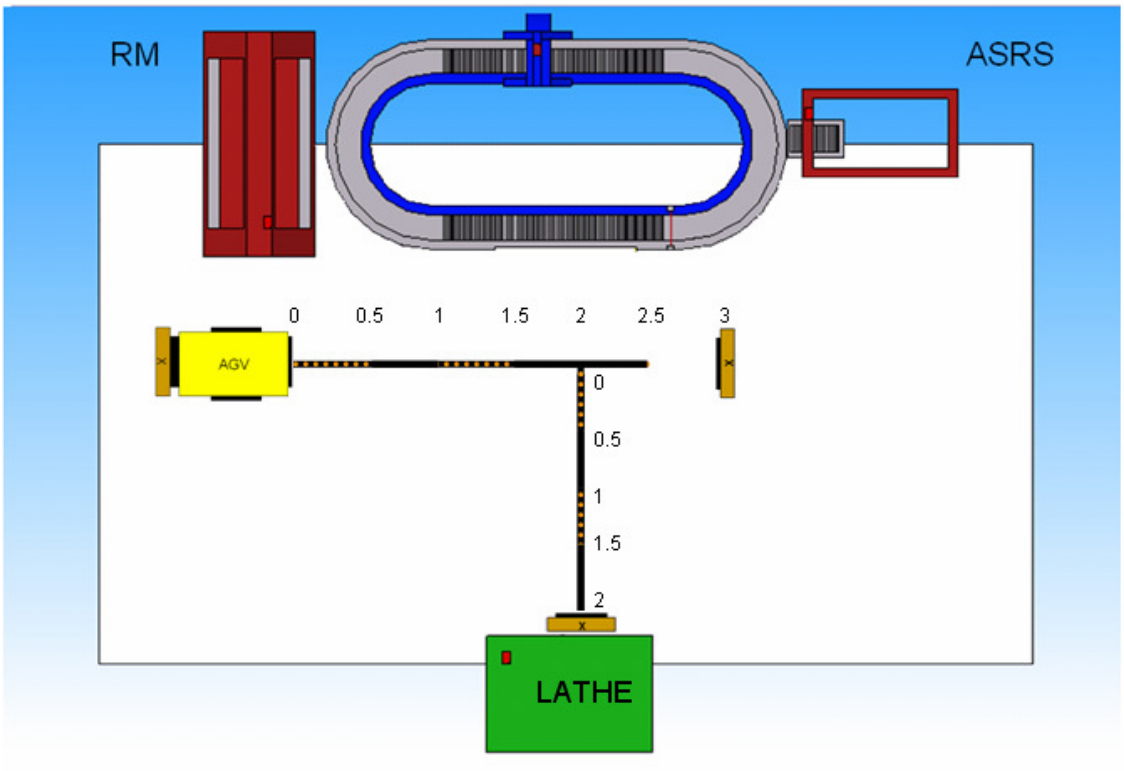


Figure 9-20 Third test scenario.

5. At 1.5m the light was not detected and the AGV line followed.
6. At 2m the AGV continued to line follow. The centre of the AGV reached the 2m mark. The front of the AGV was at 2.5m.
7. The AGV was at the virtual T junction where the AGV changed direction from reverse to side-left. However there was no light to stop the AGV 0.5m in front of the light. The program accommodated for two occurrences:

a. While the AGV was line following, the side-left line sensors detected the line in the side-left direction and the AGV changed direction (80% reliable as determined in section 7.1.2). The side-left light sensors then detected light in the side-left direction and the AGV started to light follow.

b. If the side-left line was not detected the AGV continued to line follow in the reverse direction. When the reverse line sensors did not detect the line after 2.5m, the AGV started to search for the line by moving in a clockwise direction. After turning 90 degrees the reverse line or light sensors detected the line or light respectively in the side-left direction and the AGV proceeded

towards the lathe in the longitudinal orientation. This is illustrated in figure 9-21. If both the line and light were detected the AGV would light follow, as the light variation technique was being utilised, and light takes preference. In the line variation technique, the line takes preference. Situations did occur where the line was missed when the AGV was rotating but the light was detected. Detecting the light or line within the first 0.5m is dependent on the angle at which the AGV turns i.e. if the sensors are within range of the light or line within the first 0.5m. It is irrelevant as to whether the light or line is detected first as either way the programming will allow the AGV to tend towards the path and hence the destination.

8. At 0.5m the light sensors did not detect the light so the AGV started line following in the side-left direction.
9. At 1m the light sensors detected the light and the AGV resumed light following.
10. At 1.5m the light was not detected and the AGV line followed.
11. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-14 indicates the technique used under the different parameters for the third scenario.

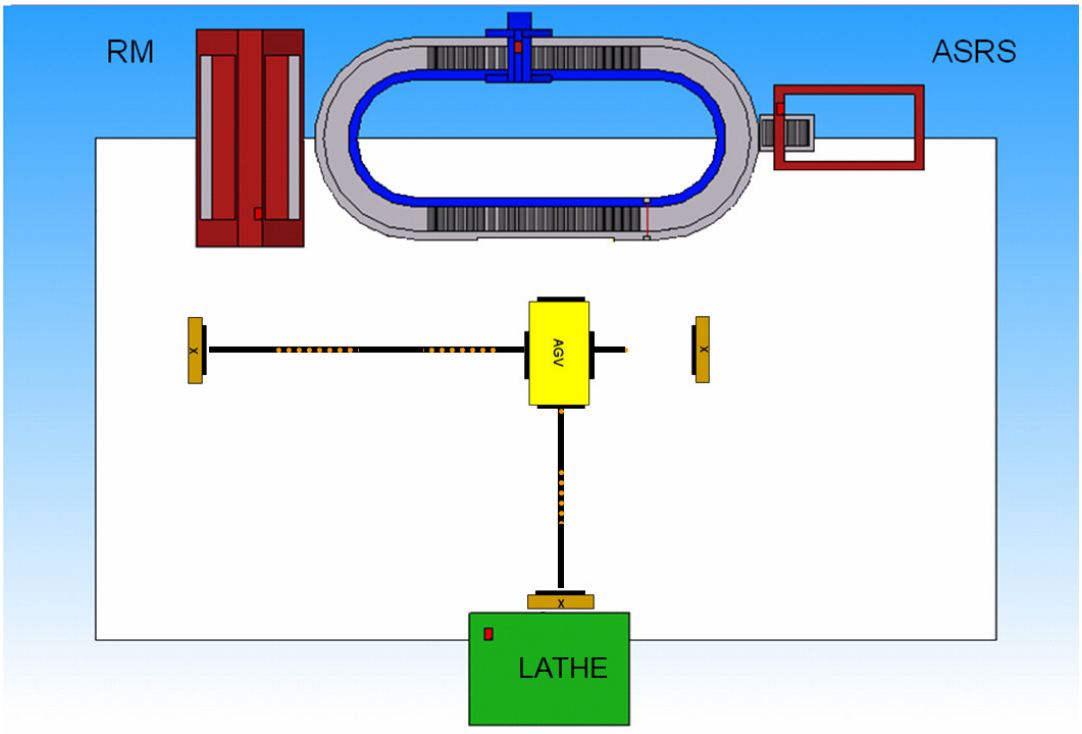


Figure 9-21 AGV in longitudinal position after turning 90 degrees.

Table 9-14 Technique used under parameter status for third scenario

Distance (m)	**Parameter status**	**Technique used:**	
	Variable: Light Constant: Line	**Light following**	**Line following**
Reverse			
0	Light	✓	
0.5	No light		✓
1	Light	✓	
1.5	No light		✓
2	No light		✓
2.5	No light		✓
Virtual T junction No light in reverse direction. AGV detects side-left line and changes direction. Refer to 7.			
Side-left			
0	Light	✓	
0.5	No light		✓
1	Light	✓	
1.5	No light		✓
2	Docks		✓

With reference to point (7), the programming did not accommodate for the side-left light sensors sensing the light in the side-left direction until the AGV stopped at the junction. The AGV was programmed to stop 0.5m in front of the light in the reverse direction, detect light in side-left direction, and then proceed side-left to ensure correct routing. This also ensured that the AGV light followed close to the line should it need to change to line following.

If the side-left sensors were programmed to detect the light throughout travel while the AGV was travelling in the reverse direction, the AGV could have left the reverse line at any point (within a given area of light detection) on detecting the light in the side-left direction, and proceeded in the side-left direction. The AGV would not light follow close to the line but would get to it when steering towards the light. This is illustrated in figure 9-22.

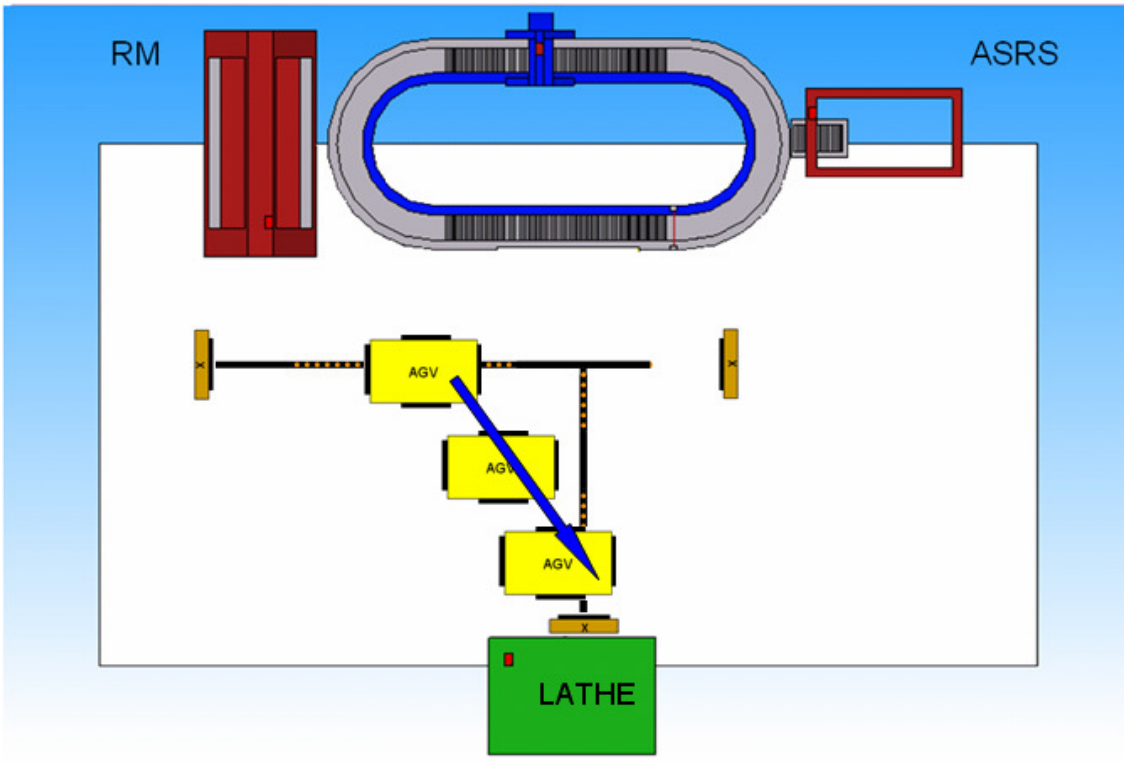


Figure 9-22 AGV leaving line in reverse direction at uncertain point and steering toward light in side-left direction. AGV does not follow the line closely.

As the line would not always be present in the uncertain travel direction, it would hinder alternation of light and line following when required. The AGV was therefore programmed to stop 0.5m in front of the light in the reverse direction and then proceed side-left. The AGV's behaviour was more controlled and predictable and a constant line was present in the travel direction for technique alternation. There was a 20% chance of (7.b) occurring as determined in section 7.1.2.

Figure 9-23 illustrates the graph of the average motion of the AGV versus the path. The test concluded that the AGV changed direction from reverse to side-left with no light in the reverse direction. This was assisted by the line following technique as explained in point (7).

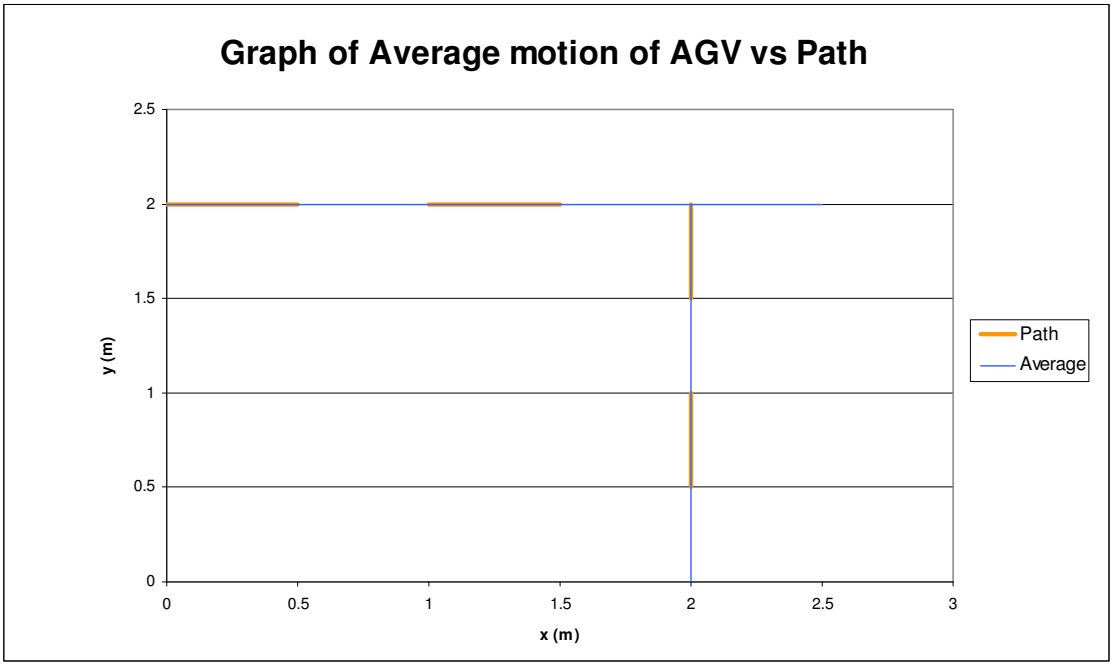


Figure 9-23 Graph of Average Motion of the AGV versus Path.

9.3.5 Fourth test scenario

Figure 9-24 illustrates the fourth test scenario. In the reverse direction the 20W light was switched off at 0.5m and 1.5m, and switched on at 0m and 1m. In the side-left direction the light was switched off at 0m and 1m, and switched on at 0.5m and 1.5m. The test demonstrated the AGV's ability to change direction from reverse to side-left at the virtual T junction with no light in the reverse and side-left directions.

Motion of the AGV utilising light variation technique

The motion of the AGV when instructed to transfer a pallet, from the RM to the lathe, using the light variation technique is described.

1. The AGV was docked at the RM docking station.
2. The AGV light followed in the reverse direction i.e. away from the RM.
3. When the light sensors did not detect the light at 0.5m, the AGV started line following.
4. At 1m the light sensors detected the light and the AGV resumed light following.
5. At 1.5m the light was not detected and the AGV line followed.

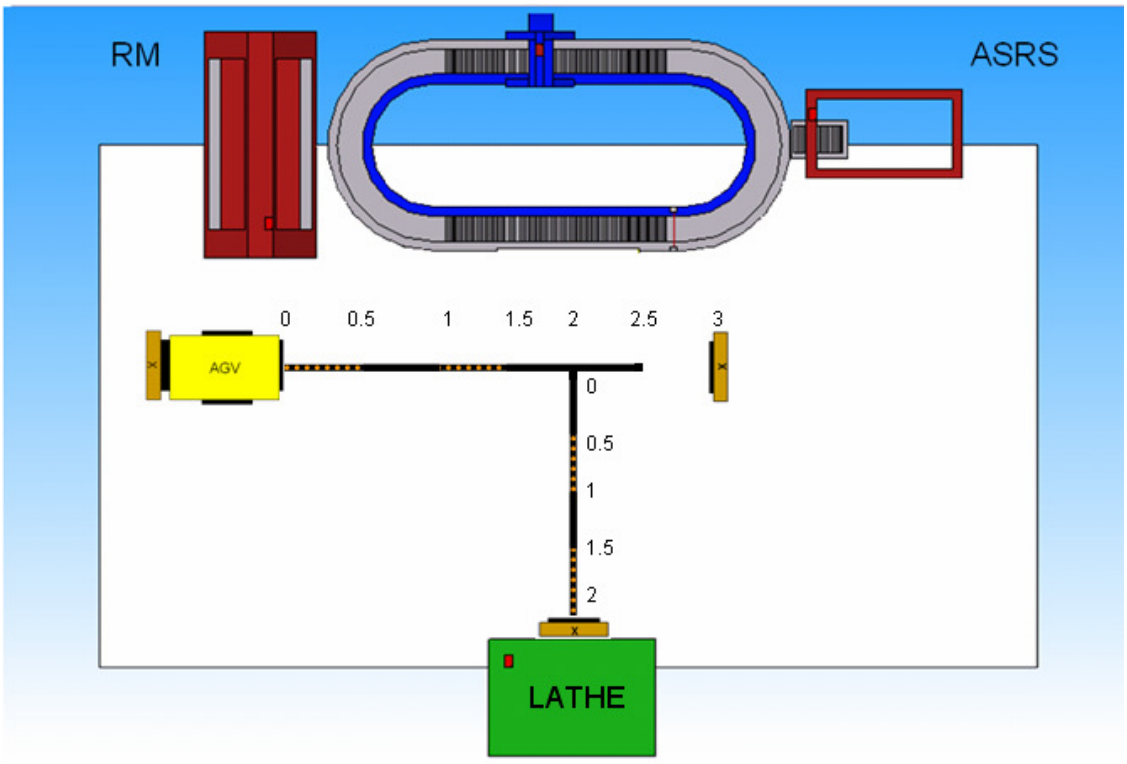


Figure 9-24 Fourth test scenario.

6. At 2m the AGV continued to line follow. The AGV's centre reached the T junction where the line sensors detected the line leading toward the lathe in the side-left direction. The AGV's front was at 2.5m.
7. On sensing the path to the lathe the AGV stopped line following in the reverse direction and started line following in the side-left direction.
8. At 0.5m the light was detected by the light sensors hence the AGV started light following.
9. At 1m the light was not detected and the AGV started line following.
10. At 1.5m the light sensors detected the light and the AGV resumed light following.
11. At 2m the AGV docked at the lathe and awaited manual transfer.

Table 9-15 indicates the technique used under the different parameters for the fourth scenario. Figure 9-25 illustrates the graph of the average motion of the AGV, versus the path. The test concluded that the AGV changed direction from reverse to side-left with no light in both directions. This was assisted by the line following technique when the line sensors detected the line in the side-left direction, as explained in points (6) and (7).

Table 9-15 Technique used under parameter status for fourth scenario

Distance (M)	Parameter Status Variable: Light Constant: Line	Technique used:	
		Light Following	Line Following
Reverse			
0	Light	✓	
0.5	No light		✓
1	Light	✓	
1.5	No light		✓
2	No light		✓
2.5	No light		✓
T junction Centre of AGV above junction. AGV changes direction.			
Side-left			
0	No light		✓
0.5	Light	✓	
1	No light		✓
1.5	Light	✓	
2	Docks	✓	

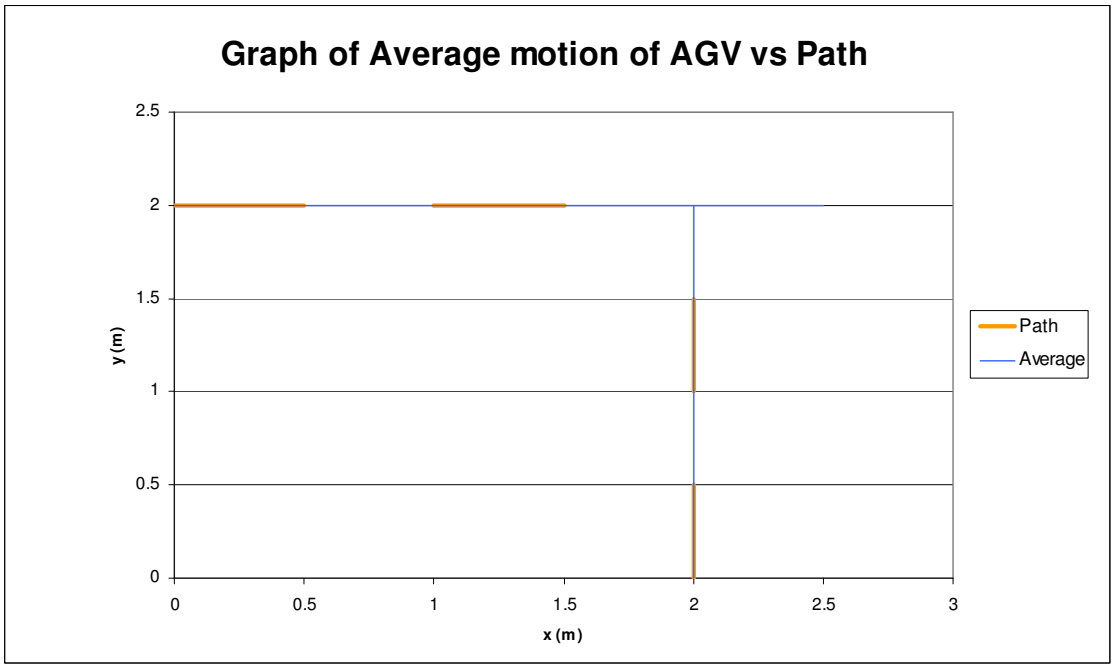


Figure 9-25 Graph of Average Motion of the AGV versus Path.

9.4 Summary

This chapter described testing of the variable sensor system on the AGV in three different environments in order to determine the robustness and reliability of the system. The line variation technique was tested in the unsatisfactory floor environment. The light variation technique was tested in the light intensity environment. Programming allowed both techniques to be implemented in the high noise level environment. The route from the RM to the lathe was tested under different scenarios. The track was varied to test the alternation of line following and light following in the line variation and light variation techniques. The AGV's ability to change direction at the junction and complete its task regardless of the changing parameters was tested. In the unsatisfactory floor environment parts of the line were absent and the light was kept constant. In the light intensity environment the light intensity was varied from high, to low, to no light, and the line was kept constant.

A description of the AGV's movement and technique alternation was provided for each scenario tested. Graphs illustrating the movement of the AGV under the different scenarios were derived. Insignificant offshoots were regarded as negligible. Significant offshoots from the path were indicated on the graphs. Testing concluded that the AGV could alternate between line following and light following as the parameters in the environment changed to complete its task. The AGV had an 80% chance of changing direction correctly at the junction point, as determined in section 7.1.2. Programming accommodated for the 20% error of missing the change of direction line, thereby allowing the AGV to reach its destination, as in sections 9.2.5, 9.3.3 and 9.3.4. The variable sensor system provided the AGV with reliable guidance and navigation to complete its tasks.

10 Robot Testing

To demonstrate the flexibility, modularity, and exchangeability of the variable sensor system, the system was implemented and tested on another robot. The robot is illustrated in figure 10-1. The robot was a result of a fourth year project. Its mechanical structure consisted of a circular aluminium base and a Perspex top. The top and base were 0.4m in diameter and held together by four bolts. The robot's drive system consisted of two 7.2V DC gear motors, directly connected to wheels, on either side of the robot. Two castor wheels supported the front and back of the robot. The robot was intended as a surveillance robot that would monitor manufacturing operations in the Mechatronics lab.

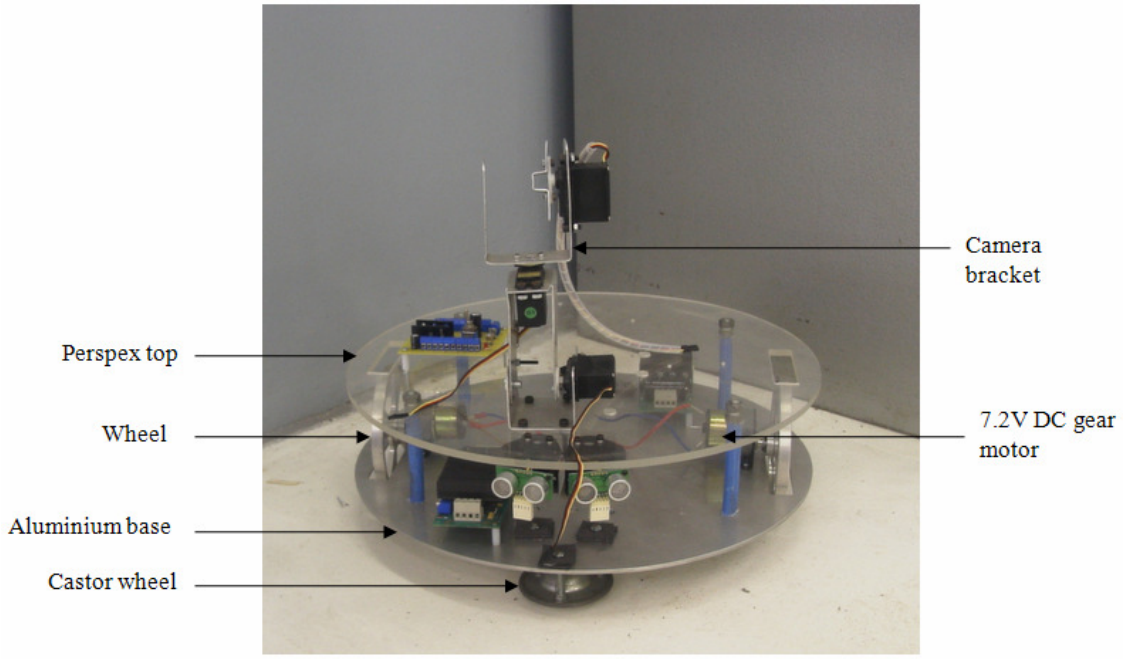


Figure 10-1 Robot before modifications

The robot was not omni-directional like the AGV. As a result, line and light following were limited to the forward/reverse directions. The guidance techniques, with the use of ultrasonic sensors were tested on the robot in the forward direction. Limit switches were not utilised as docking was not necessary for the robot. The ultrasonic sensors provided sufficient obstacle avoidance to prevent collisions.

10.1 Line following technique

10.1.1 Mechanical

To enable the robot to line follow, certain mechanical modifications were made. A mechanism to hold the LDRs in place, and allow for light collimation was needed. Three holes were drilled in the base and top of the robot. 10mm Aluminium tubes, 95 mm in length, were inserted through each pair of holes (refer to figure 10-3). A groove was made at the top of each tube and a washer was fitted around the groove. The washers allowed the tubes to be easily removed and inserted when needed. (The tubes were a temporary addition and could be removed in the future if the robot was needed by another student.) The three sensors were inserted 60mm into the tubes, allowing for 30mm light collimation. A slot was milled in the aluminium base to hold a 20W bi-pin halogen lamp. The lamp eliminated the shadow formed by the robot base to allow for distinguishable sensor readings between the black tape and white floor.

10.1.2 Electronic

The modular guidance and navigation system box was used (refer to figure 10-2). The motor boards, line sensor and ultrasonic sensor board were utilised for guidance and navigation. The motor boards for the AGV (described in section 5.2.1) were designed for 12V DC motors and had 12V relays and MOSFETS. To accommodate for the 7.2V DC motors, the motors were operated below 500 i.e. half of the maximum speed, with the 12V motor boards. This prevented over heating of the 7.2 V motors. The ultrasonic sensors were positioned in front of the robot.

10.1.3 Software

The AGV's program was adapted to suit the two-wheeled robot. The robot was more agile and faster than the AGV, due to its lighter and smaller body. The robot had a completely different drive system from the AGV and was limited to forward, reverse, left and right directions. (The robot's drive system was described in the first paragraph of chapter 10 and the AGV's drive system was described in the first paragraph of section 3.1.)

The robot was operated at very low speeds (95-125) to prevent it from straying over the line. A "RobotMotor" file was created to accommodate for the lower speeds and the control of two motors

instead of four. The "RobotMotor" file was accessed internally instead of the AGV's "Motor" file when a direction change was necessary in all robot guidance programs.

The robot program layout is illustrated in figure 10-4. The robot's program layout was similar to the AGV's program layout (section 8.1) but the routes were not included as only the forward/reverse directions were possible. The robot was positioned manually at the start point for testing. Only the forward direction was tested. The robot line following file was accessed by the user through the robot program. Programming is located in Appendix E.2.2.

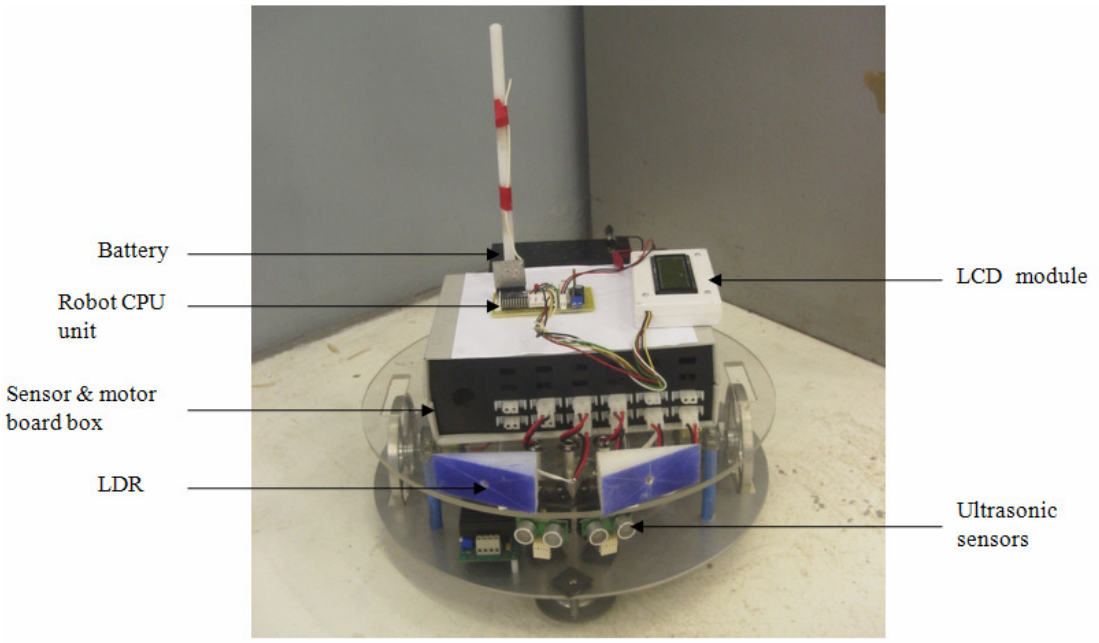


Figure 10-2 Robot after modifications

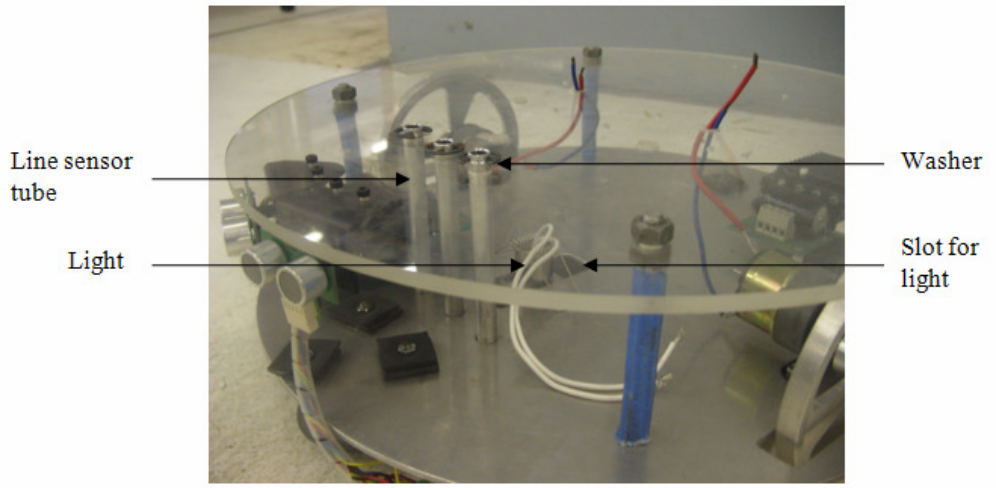


Figure 10-3 Slot for light and tubes for line sensors

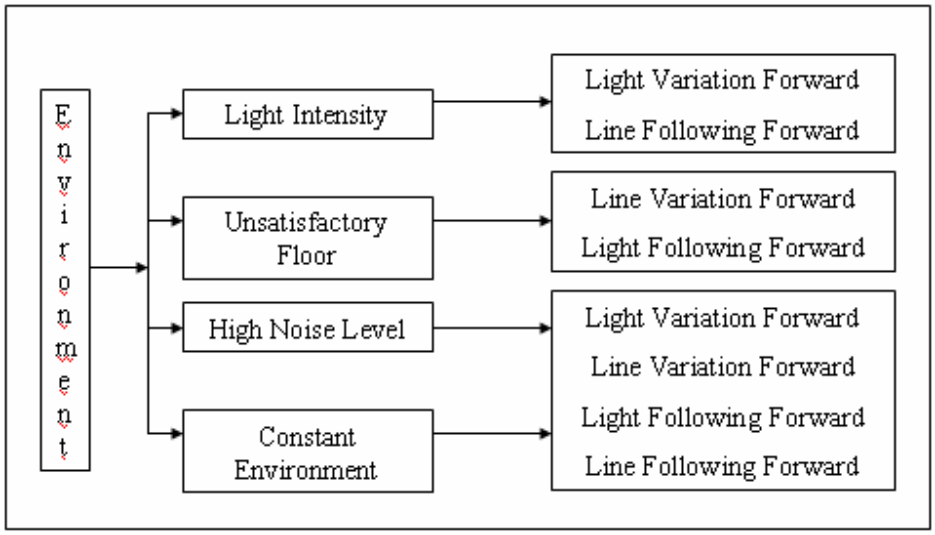


Figure 10-4 Robot program layout

10.1.4 Testing

Line following testing was conducted in the constant environment. The robot was tested to follow the non-linear path and the linear path, illustrated in figure 6-14, in the forward direction. The sensors continually corrected the robot's motion such that the centre sensor stayed above the line and the remaining two sensors on either side of the line.

The robot was tested ten times on the linear and non-linear paths in the forward direction. The robot strayed off the linear path two times. It was 80% reliable in following the linear path and had a 20% chance of straying off the line. The robot strayed off the non-linear path three times. It was 70% reliable in following the non-linear path had a 30% chance of straying off the line. The software automatically stopped the robot when the line was not detected.

The AGV was 90% reliable in following the linear path in the forward direction and had a 10% chance of straying off the line, as determined in section 7.1.1. The AGV was 80% reliable in following the non-linear path in the forward direction and had a 20% chance of straying off the line, as determined in section 7.1.2.

The robot's straying error of 20% on the linear path and 30% on the non- linear path can be attributed to the dynamics of its drive system, mass and faster speed. The control program operated at a constant speed. The AGV moved at 0.125m/s and experienced slow deviation from the line therefore the program could control steering with 90% reliability on the linear path and 80%

reliability on the non-linear path. The robot moved faster than the AGV, therefore it did not react to the readings with the same speed the AGV did, and as a result it strayed off the line more than the AGV.

The robot took a shorter time than the AGV to complete the paths. The robot took an average of 0.55 minutes to complete the linear path and 3.41 minutes to complete the non-linear path, as indicated in tables 10-1 and 10-2. The AGV took an average of 6.5 minutes to complete the non-linear path as indicated in table 7-2. (The AGV was timed in the lateral direction on the linear path, table 7-1, and therefore cannot be compared with the robot timing results for the forward direction.)

Table 10-1 Time for robot to complete linear path

Time to complete linear path	
Attempt	**Time (Minutes)**
1	0.52
2	0.57
3	0.55
Average	0.55

Table 10-2 Time for robot to complete non-linear path

Time to complete non-linear path	
Attempt	**Time (Minutes)**
1	3.37
2	3.42
3	3.45
Average	3.41

10.2 Light Following Technique

10.2.1 Mechanical

Two light sensors were inserted into holes, in Perspex wedges. The wedges were positioned at the front of the robot, as illustrated in figure 10-2. The holes in the wedges collimated the light, allowing for acceptable readings between light and dark.

10.2.2 Electronic

The motor boards, light sensor and ultrasonic sensor board were utilised for navigation and guidance.

10.2.3 Software

The AGV's light following program was adapted for the robot. The "RobotMotor" file was accessed when a direction change was necessary in the light following program. The robot was programmed to turn left, if more light was detected in the left direction, right, if more light was detected in the right direction and proceed forward if equal light was detected by both sensors. The robot light following file was accessed by the user through the robot program. Programming is located in Appendix E.2.3.

10.2.4 Testing

Light following testing was conducted in the constant environment. The robot was tested to follow the light on the linear path, from the lathe to the conveyor, illustrated in figure 6-24. Light following in the forward direction was tested. The light was turned on at the docking station under the conveyor. The light sensors corrected the direction of the robot as it moved toward the light. The robot took an average of 0.57 minutes to reach the conveyor, as indicated in table 10-3.

Table 10-3 Time for robot to reach the light at the conveyor.

Attempt	**Time (Minutes)**
1	0.54
2	0.56
3	0.6
Average	0.57

The robot was also tested to follow the light on the non-linear path, from the lathe to the RM, illustrated in figure 6-26. The robot was positioned at the lathe, with its front facing the conveyor. The light was turned on at the docking station at the RM. The left light sensor detected more light than the right sensor. The robot rotated left, toward the light, and proceeded forward when equal light was detected (refer to figure 10-5). The robot corrected its motion until it reached the light at

the RM docking station. The path followed was non-linear and uncertain. The robot took an average of 1.22 minutes to reach the RM, as indicated in table 10-4. The robot stopped approximately 0.5 m from the light, as programmed.

Table 10-4 Time for robot to reach the light at the RM.

Attempt	**Time (Minutes)**
1	1.18
2	1.22
3	1.25
Average	1.22

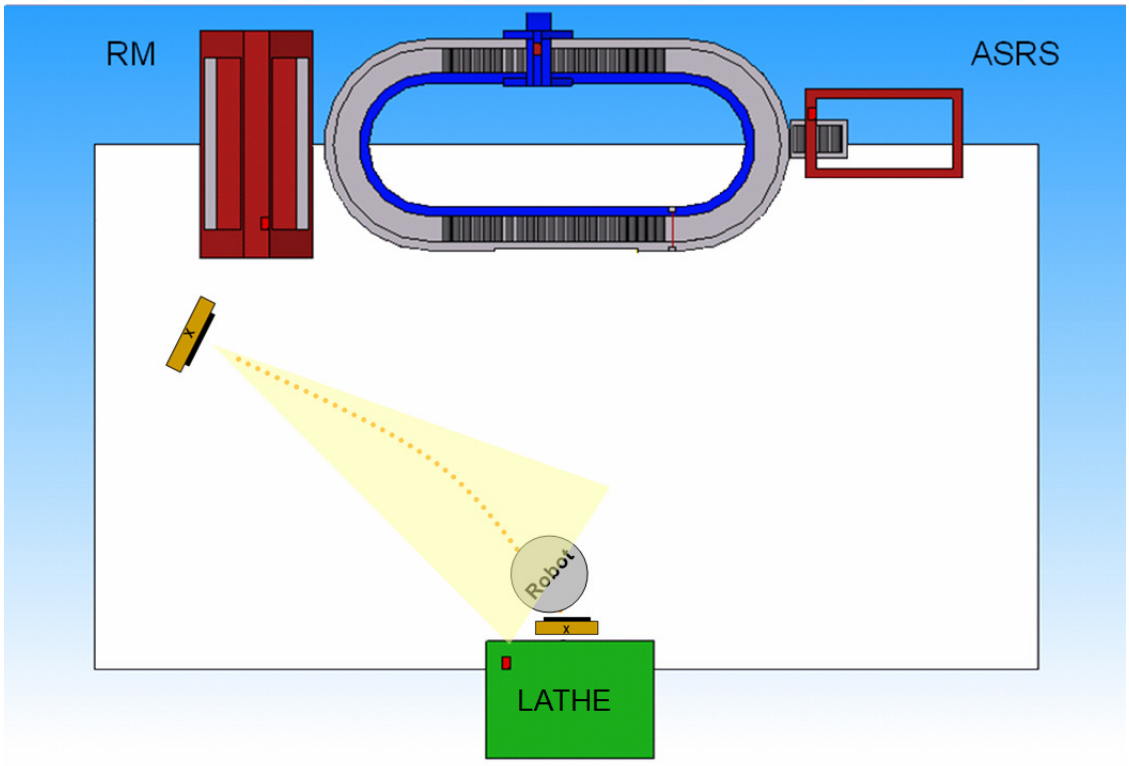


Figure 10-5 Robot in light route from the lathe to the RM. Robot turns from forward direction to face light as left sensor receives more light.

10.3 Line variation technique

The robot was tested between the RM and the conveyor, in the unsatisfactory floor environment under two scenarios. The robot was positioned such that the line sensors where at the AGV sensor start position, at 0m. Light was kept constant and positioned at 3m. (Programming is located in Appendix E.2.4.)

10.3.1 First Scenario

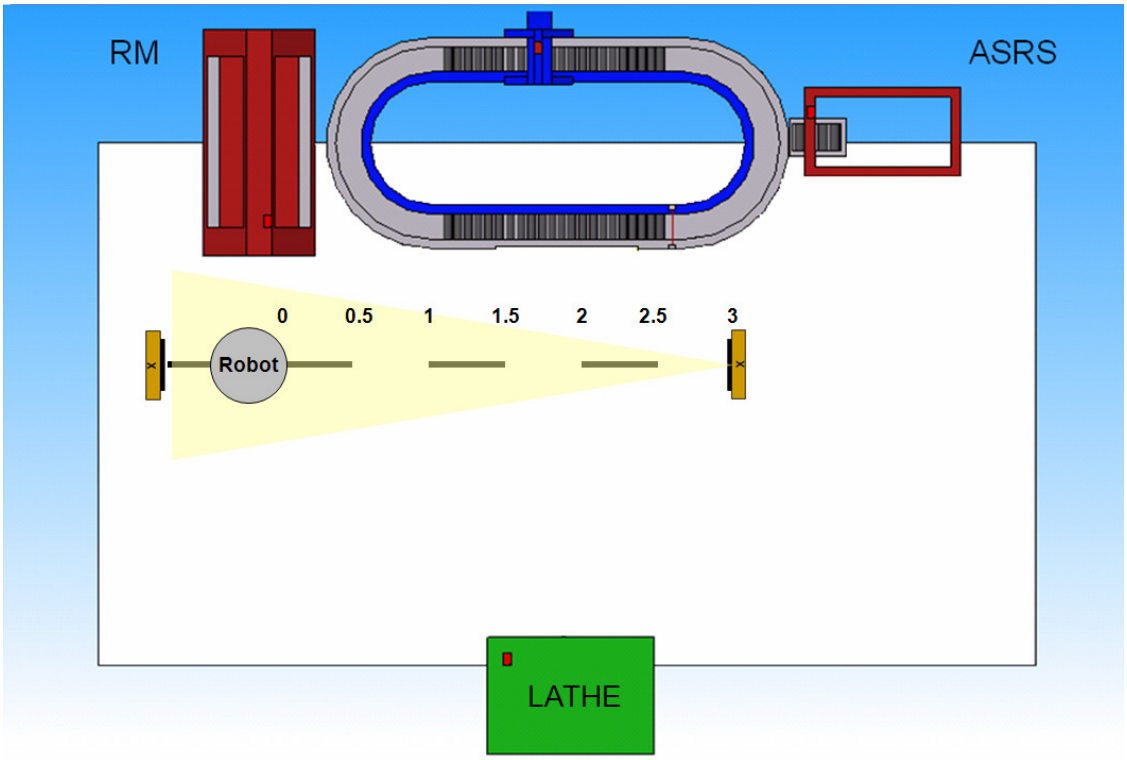


Figure 10-6 Robot at start position for first scenario

The first scenario is indicated in figure 10-6. The line was present at 0m and absent in 0.5m intervals. The robot started line following as the line was present at 0m. The program allowed the robot to alternate from line following to light following when the line sensors did not detect the line at 0.5m, 1.5m and 2.5m. The robot resumed line following when the line sensors redetected the line at 1m and 2m. At 2.5m, the end of the track, no line was detected and the robot changed to light following. The light sensors detected the light at 3m and signalled the robot to stop. The robot stopped approximately 0.47m in front of the light at 2.53m (3 - 2.53 = 0.47m). An average offshoot

of 0.03m is calculated in table 10-6. The AGV was assisted by light following to stop when it did not detect the line at the end of the track. Table 10-5 indicates the technique used under the different parameters. The graph in figure 10-7 indicates the average motion of the robot on the path.

Table 10-5 Technique used under parameter status for first scenario

Distance (m)	**Parameter status** **Variable: Line Constant: Light**	**Technique used:**	
		Line following	**Light following**
0	Line	✓	
0.5	No line		✓
1	Line	✓	
1.5	No line		✓
2	Line	✓	
2.5	No line – Stops		✓

Table 10-6 Offshoot results and average

Test	**Distance (m)**	**Offshoot (m)**
1	2.53	0.03
2	2.55	0.05
3	2.54	0.04
4	2.52	0.02
5	2.53	0.03
Average	**2.534**	**0.034**

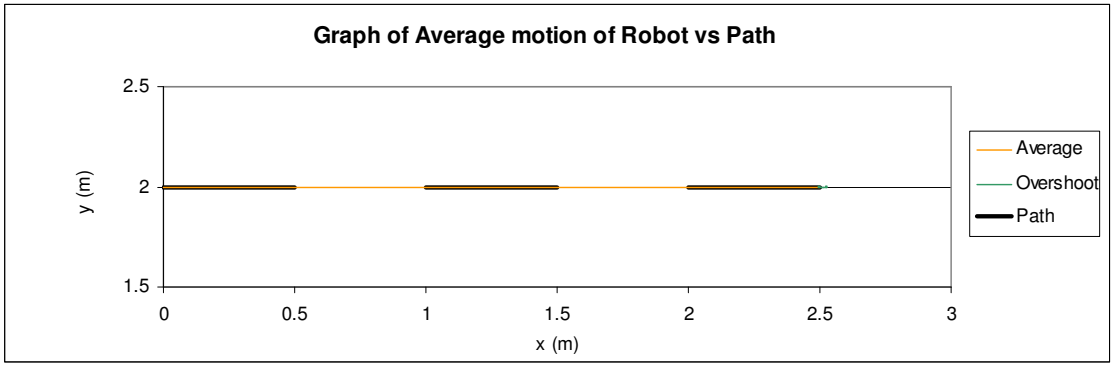


Figure 10-7 Average motion of the robot on the path

10.3.2 Second Scenario

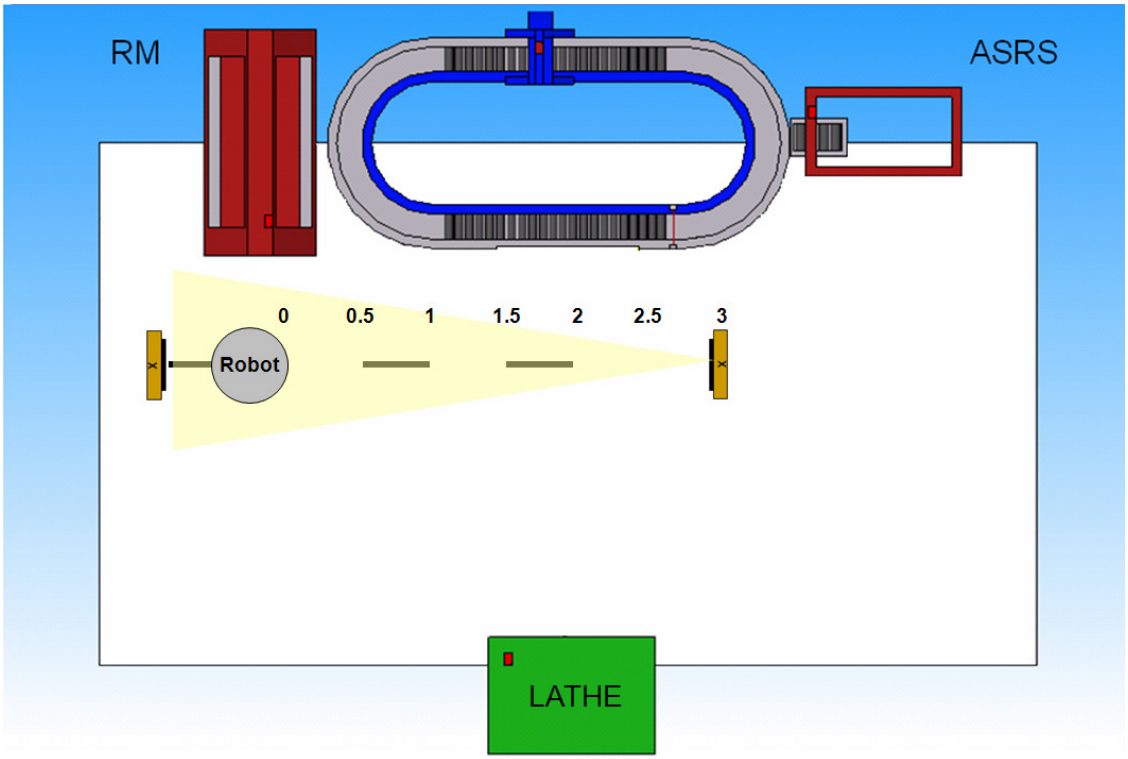


Figure 10-8 Second scenario

The second scenario is indicated in figure 10-8. The line was absent at 0m and present in 0.5m intervals. The robot started off light following as the line was absent at 0m. The program allowed the robot to resume line following when the line sensors redetected the line at 0.5m and 1.5m. The program allowed the robot to alternate from line following to light following when the line sensors did not detect the line at 0m, 1m and 2m. When the light sensors detected the light at 3m, the robot was signalled to stop. The robot stopped approximately 0.5m in front of the light at 2.5m. Table 10-7 indicates the technique used under the different parameters. The graph in figure 10-9 indicates the average motion of the robot on the path.

Table 10-7 Technique used under parameter status for second scenario

Distance (m)	**Parameter status:** **Variable: Line Constant: Light**	**Technique used:**	
		Line following	**Light following**
0	No line		✓
0.5	Line	✓	
1	No line		✓
1.5	Line	✓	
2	No line		✓
2.5	No line – Stops		✓

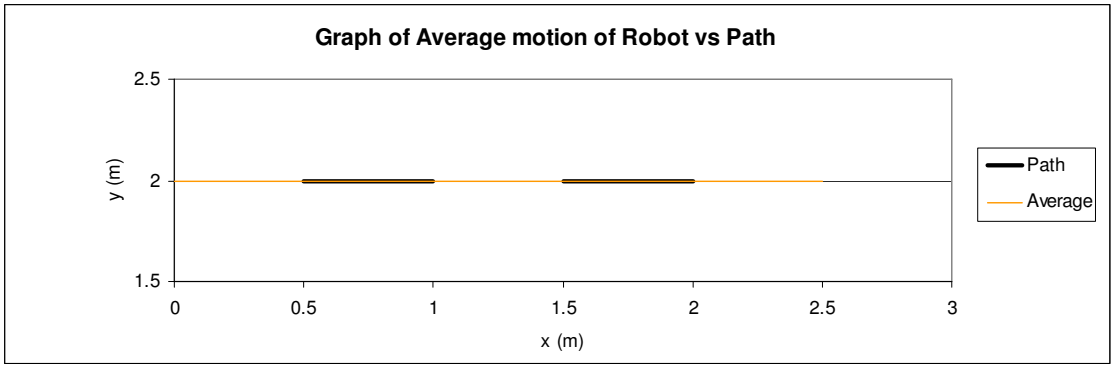


Figure 10-9 Average motion of the robot on the path

10.4 Light variation technique

The robot was tested between the RM and conveyor, in the light intensity environment under two scenarios. The robot was positioned such that the light sensors where at the AGV sensor start position, at 0m. The line was kept constant. (Programming is located in Appendix E.2.5.)

10.4.1 First scenario

The first scenario is indicated in figure 10-10. The light was present at 0m and switched off and on in 0.5m intervals.

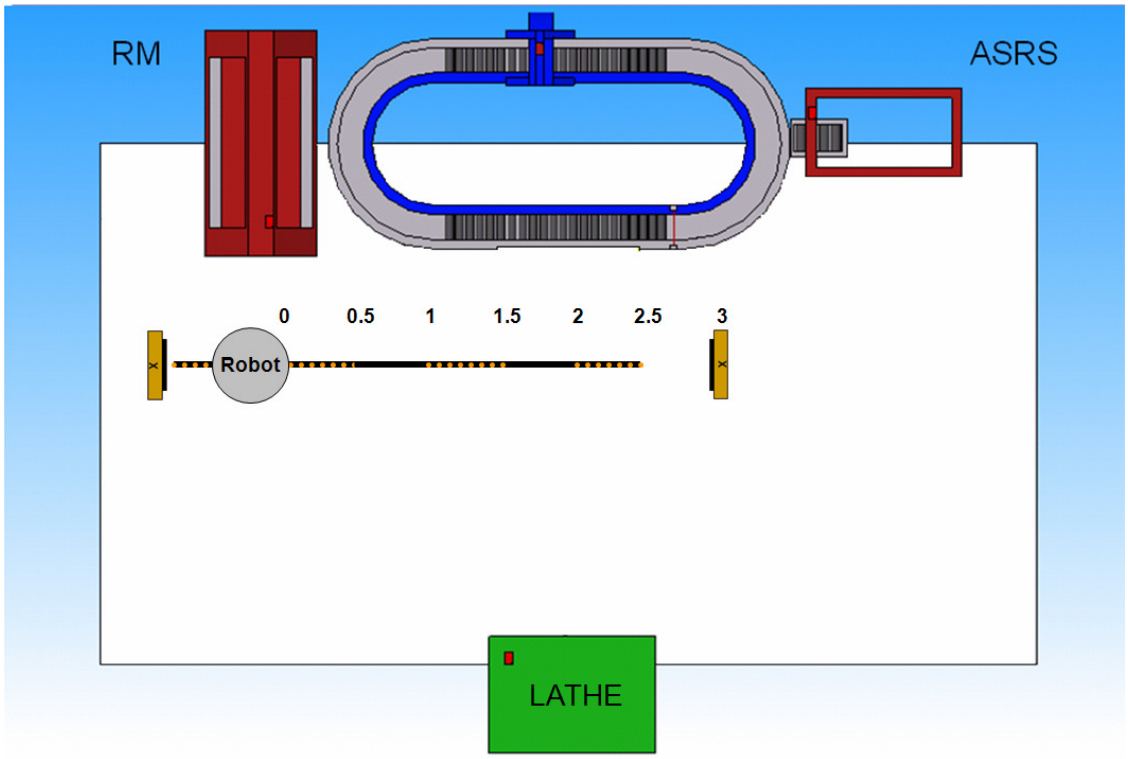


Figure 10-10 First scenario

The robot started off light following as the light was present at 0m. The program allowed the robot to alternate from light following to line following when the light sensors did not detect light at 0.5m and 1.5m. The robot resumed light following when the light sensors redetected light at 1m and 2m. The robot stopped approximately 0.5m in front of the light at 2.5m. Table 10-8 indicates the technique used under the different parameters. The graph in figure 10-11 indicates the average motion of the robot on the path.

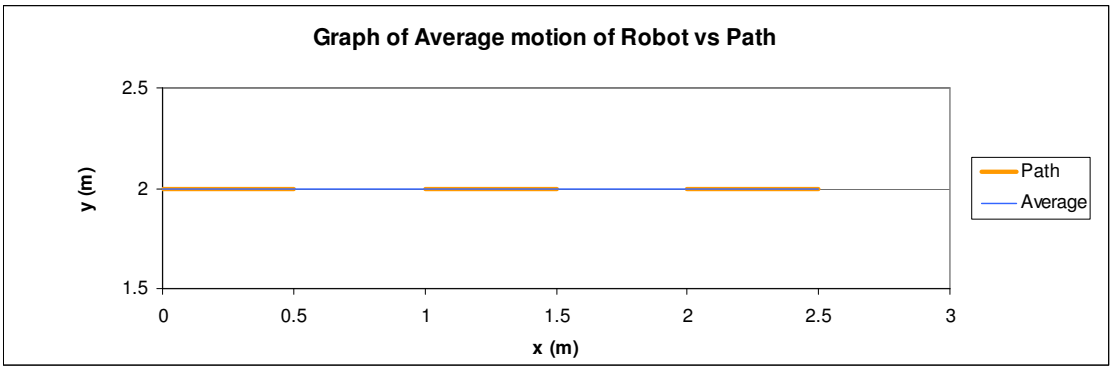


Figure 10-11 Average motion of the robot on the path

Table 10-8 Technique used under parameter status for first scenario

Distance (m)	Parameter status Variable: Light Constant: Line	Technique used:	
		Light following	Line following
0	Light	✓	
0.5	No light		✓
1	Light	✓	
1.5	No light		✓
2	Light	✓	
2.5	Light – Stops	✓	

10.4.2 Second scenario

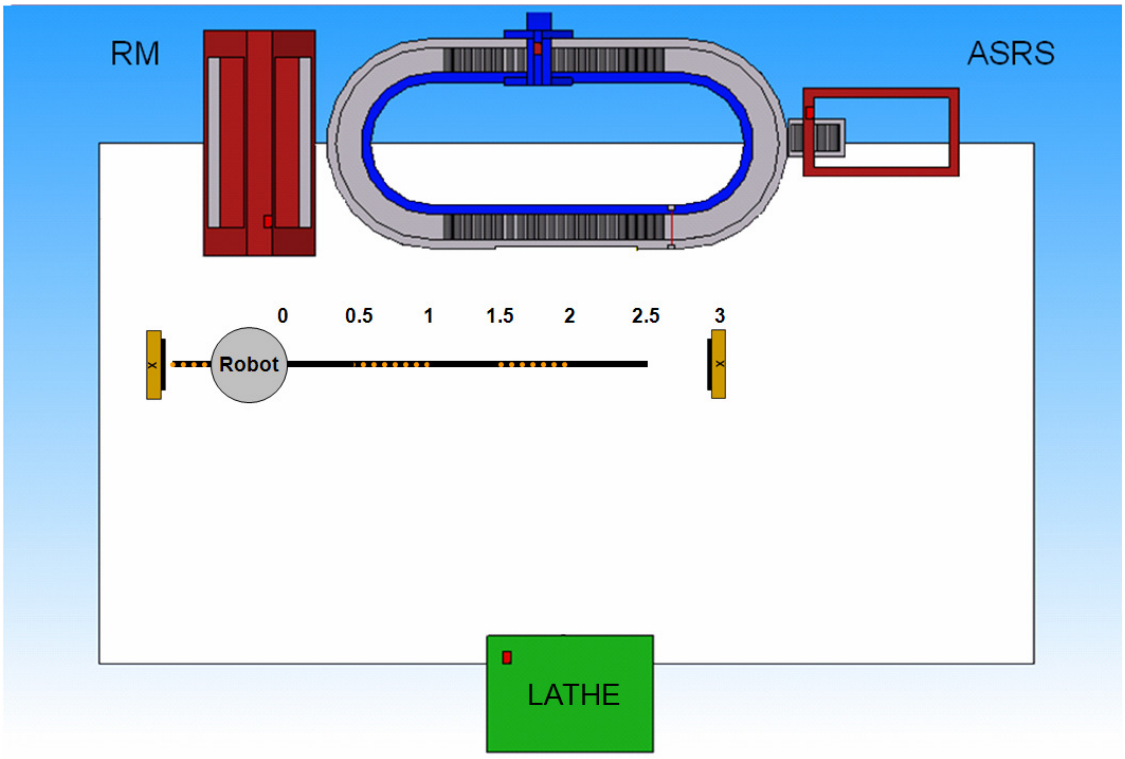


Figure 10-12 Second scenario

The second scenario is indicated in figure 10-12. The light was absent at 0m and switched on and off in 0.5m intervals. The robot started off line following as the light was absent at 0m. The program allowed the robot to resume light following when the light sensors redetected light at 0.5m and 1.5m. The program allowed the robot to alternate from light following to line following when the light sensors did not detect light at 0m, 1m and 2m. The robot was stopped through the software

at 2.5m. This is because the robot could not dock like the AGV and stop motion. The AGV stopped on compressing the limit switches when it docked. (Providing the robot with docking ability was not one of the project requirements. Implementation of the variable sensor system on the robot needed to be tested to prove modularity and flexibility.) Table 10-9 indicates the technique used under the different parameters. The graph in figure 10-13 indicates the average motion of the robot on the path.

Table 10-9 Technique used under parameter status for second scenario

Distance (m)	**Parameter status**	**Technique used:**	
	Variable: Light Constant: Line	**Light following**	**Line following**
0	No light		✓
0.5	Light	✓	
1	No light		✓
1.5	Light	✓	
2	No light		✓
2.5	No light	Stopped through software	

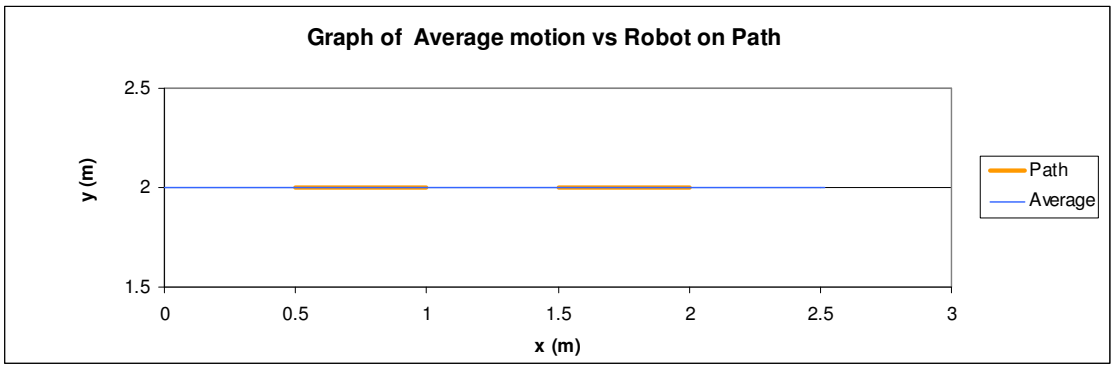


Figure 10-13 Average motion of the robot on the path

10.5 Discussion

The robot was not omni-directional like the AGV and therefore could not be tested to change direction at the decision point as the AGV was tested. Changing directions at the decision point could be achieved by the addition of line sensors on the sides of the robot. This would allow the robot to sense if a line is present in either of the lateral directions.

The robot could be programmed to rotate, on sensing the side line, for a timed period such that the robot completed a 90 degree turn. At the end of the 90 degree turn, the robot would be in the forward position in the new direction. In this way the robot could change direction and continue on its path for surveillance purposes.

Robot testing concluded that the variable sensor system was modular, flexible and exchangeable. Implementation of standard guidance and navigation systems on robots working in the same environment would optimise robot performance.

10.6 Summary

This chapter described testing of the variable sensor system on another robot. Mechanical and software modifications were discussed. The line following and light following techniques were tested on linear and non-linear paths in the constant environment. The line variation technique was tested on the RM to conveyor route in the unsatisfactory floor environment. The light variation technique was tested on the RM to conveyor route in the light intensity environment. Graphs illustrating the movement of the robot under the different scenarios were derived. Testing concluded that the variable sensor system was adaptable, and compatible with the robot. The system provided the robot with the necessary guidance and navigation to follow the assigned routes. The system can be utilised to enable the robot to perform surveillance operations.

11 Quantitative Analysis of AGV systems

The operation of an AGV system can be defined by equations. It is assumed that the AGV operates at constant velocity, V_c , throughout its operation. The effects of acceleration, deceleration and other speed differences that result from load carrying are ignored. The total time for delivery is defined by the following equation:

$$T_v = \frac{L_d}{V_c} + T_h + \frac{L_e}{V_c} \qquad (1)$$

Where

T_v represents the total time for delivery.

$\frac{L_d}{V_c}$ represents the travel time to the drop off station.

T_h load handling time which consists of the loading operation at the pick up station and the unloading operation at the drop off station.

$\frac{L_e}{V_c}$ represents the empty travel time of the vehicle between deliveries.

The reciprocal of T_v yields the number of deliveries per hour, per vehicle if there are no traffic congestion losses. Traffic losses have a significant effect on the performance of an AGVS and are accommodated for by the traffic factor, F_t . The traffic factor estimates the effect of losses on system performance. Losses include blocking of vehicles, waiting at intersections, vehicles waiting in line, poor scheduling, inefficient routing of vehicles and poor layout of the guide path.

Blocking, waiting at intersections, and vehicles waiting in line, are determined by the number of vehicles in the system relative to the layout size. If there is only one vehicle in the system, blocking will not occur and the traffic factor will be close to 1. However, if there are many vehicles in the system, blocking and congestion increases and the traffic factor takes a lower value. Traffic congestion is determined by scheduling, routing, and the AGVS layout. Central computer control, programmed with optimum scheduling algorithms, is more efficient for scheduling and routing as compared to on-board or remote call dispatching. Traffic factors usually fall between 0.85 and 1. [12]

The losses can be combined into a measure of efficiency of a material handling system by:

$$E_h = \frac{\frac{L_d}{V_c}}{\frac{L_d}{V_c} + T_h + \frac{L_e}{V_c}} F_t \qquad (2)$$

Where E_h is the efficiency of the handling system. The number of deliveries that can be made per hour by a single vehicle is defined by:

$$\text{Number of deliveries/hour/vehicle} = \frac{60F_t}{T_v} \qquad (3)$$

Where T_v is given in minutes and F_t is a decimal value. Using handling system efficiency the equation becomes:

$$\text{Number of deliveries/hour/vehicle} = \frac{60Eh}{\frac{L_d}{V_c}} \qquad (4)$$

Where $\frac{L_d}{V_c}$ represents the direct travel time per vehicle for a delivery without delays or lost time.

The handling system efficiency excludes the losses that result from an AGVS with long or superfluous pathways that result in longer vehicle travel to reach the destination.

The total number of AGVs is given by:

$$\text{Number of AGVs} = \frac{\text{number of deliveries required/hour}}{\text{number of deliveries/hour/vehicle}} \qquad (5)$$

The maximum demand rate should be used in equation (5). [12]

11.1 Calculations

Consider the following scenario: (Refer to figure 8-12.) The AGV is docked at the lathe and is required to pick-up a part from the conveyor (2.6m), transport it back to the lathe (2.6m), and after the part has been worked on, deliver it to the RM (2m + 3m). The AGV is 1m in length. Twenty deliveries/h are required by the AGVS.

Given that:

Vehicle velocity in lateral direction = 0.0625m/s
Vehicle velocity in forward direction = 0.125m/s
Average velocity = (0.0625 + 0.125) / 2 = 0.09375m/s
Average distance travelled per delivery = 10.2m (2.6 + 2.6 + 2 + 3)
Pick-up time at conveyor = 0.3 s
Drop-off time at RM = 0.5s
Average distance travelling empty = 2.6m
Traffic factor = 0.9

The number of AGVs required to satisfy the delivery demands can be calculated as follows:

Substituting in equation (1) in section 11

$$T_v = \frac{10.2}{0.09375} + 0.3 + 0.5 + \frac{2.6}{0.09375}$$

= 137.33 s

= 2.29 min

Substituting in equation (3)

$$\text{Number of deliveries/hour/vehicle} = \frac{60(0.9)}{2.29}$$

= 23.58 deliveries/hour/vehicle

Substituting in equation (5)

Number of AGVs = 20 / 23.58

= 0.85 vehicles

This is rounded up to 1 vehicle.

Substituting in equation (2) the handling efficiency is

$$E_h = \frac{(10.2)(0.9)}{(0.09375)(2.29)}$$

$$= 42.76$$

The number of vehicles per hour can be verified using E_h by equation (4)

$$\text{Number of deliveries/hour/vehicle} = \frac{(60)(42.76)(0.09375)}{10.2}$$

$$= 23.58 \text{ deliveries/hour/vehicle}$$

11.2 Conclusion

One AGV is sufficient to serve the CIM cell in the Mechatronics laboratory as there are only three machines that require the service of the AGV. The number of deliveries/hour/vehicle required is not as high as in an industrial environment, which is made up of many CIM cells, containing more than three machines. The number of AGVs required to serve such an environment would increase proportionally to the number of deliveries required per hour.

11.3 Summary

This chapter presented equations that describe the operation of an AGV at constant velocity. The total delivery time is defined as the sum of the travel time to the drop off station, load handling time and empty travel time. Handling efficiency is defined as the product of the travel time to the drop off station and traffic factor, divided by the total delivery time. The number of deliveries that can be made per hour by an AGV is defined as the product of 60 and the traffic factor, divided by the total delivery time. The number of deliveries can also be defined as the product of 60 and the handling efficiency, divided by the travel time to the drop off station. The number of AGVs is calculated by dividing the number of deliveries required per hour, by the number of deliveries per hour per

vehicle. The equations were used to calculate the number of AGVs required to serve the laboratory machines based on travel distances of the AGV. One AGV was needed to perform material handling operations for the three machines.

12 Discussion

12.1 Guidance and routing

Routing was achieved by activating only the relevant sensor boards in the programming, when a specific path was chosen. In this way, all other branches at the junction, except the one needed, were ignored by the AGV while line following and light following. This eliminated confusion when changing direction at the junction.

The AGV changed direction at the junction with 80% reliability when line following. The AGV had a 20% chance of missing the change of direction line at the junction, as determined in section 7.1.2. (E.g. If the AGV was transporting a pallet from the RM to the lathe, the change of direction line would be the side-left line.) Missing the change of direction line was accommodated for in the programming for the line variation and light variation techniques.

In the line variation technique, when the AGV did not detect the direction change line, the program alternated to light following at the end of the line track. The light sensors detected the light in the new direction the AGV had to take. (E.g. The light sensors detected the light in the side-left direction when travelling between the RM and the lathe as described in section 9.2.4.)

In the light variation technique, if the light in the new direction had been switched off, or there was no light to stop the AGV to change direction, the AGV changed to line following. If the direction change line was not detected, the AGV rotated in a clockwise direction to search for the line. When the line was located the AGV continued along its path, as described in sections 9.3.3 and 9.3.4. Missing the direction change line can be reduced by applying more line sensors or using a thicker line.

When line following in the constant environment, if the AGV strayed from the line path, it relocated the line within an acquisition distance of 6.3 cm, with respect to the centre line sensor, as determined in section 7.1.2. In the unsatisfactory floor environment, if the AGV strayed from the line or could not detect the line, the line variation program alternated from line following to light following. While light following the AGV checked for the line. The AGV resumed line following if the line was redetected.

Similarly, in the light intensity environment, if the AGV could not detect the light, the light variation program changed from light following to line following. While line following the AGV checked for the presence of light. The AGV resumed light following if the light was redetected.

In the constant environment, the line following technique allowed automatic stopping when the AGV strayed off the line. The user was notified by an onscreen message that the AGV was off the line and required user assistance. The user was also notified if the AGV had crashed into an obstacle, for all four techniques, in all environments. The AGV followed the linear line path with 90% reliability as determined in section 7.1.1. The AGV followed the non-linear path with 84% accuracy and 80% reliability, as calculated in section 7.1.2.

The AGV followed the light on the linear paths with negligible deviation, in the constant environment. The AGV was programmed to rotate in a clockwise direction to search for the light when it could not be detected. The user was notified when the AGV searched for light by an onscreen message. Light searching was tested by turning the light off and on, and repositioning the light. Only the light required for the relevant path was switched on to prevent the incorrect path from being followed for routing. The AGV followed the non-linear path, between the lathe and the RM, with considerable uncertainty, resulting in an inconsistent docking orientation. In order to avoid the uncertainty only linear paths were used for the three changing environments.

To prevent the inaccuracy of following the light in a curve, instead of a solitary light at the destination, a series of lights can be placed along the path for the AGV to follow. The AGV would tend to the path of lights in order to reach its destination and have a consistent docking orientation.

The light variation technique accommodated for varying light intensities. Docking stations were relocated to predetermined locations at the junction to ensure correct routing for high and low intensity light following (section 8.4). If the docking stations were not relocated the AGV would not complete its task and have to be re-instructed (section 9.3.1.a).

User intervention to change the position of docking stations can be prevented by implementing a linear motion system. The docking station can be mounted on a linear stage that is driven by a servo motor, or on linear bearings located on guide rails. One guide rail system can be used at the centre of the docking station or two guides on either side of the docking station for more stability. A DC motor rack and pinion drive system can be implemented for the guide rail system. The system can

be embedded in the floor at sub-level or the docking mechanism can be lowered so that it lines up with the AGV's docking mechanism. The linear motion system can be programmed to move the docking station to the correct position, when the control program changes from high to low light following.

Automatic stopping and light searching features were not available in the line variation and light variation techniques, which were utilised in the unsatisfactory floor and light intensity environments, respectively. The program compensated for the changing environment by interchanging between line and light following to enable the AGV to complete its tasks.

12.2 Testing

Testing results concluded that the variable sensor system enabled the AGV to function reliably in the constant environment and the three unpredictable environments by utilising the appropriate technique.

12.2.1 Constant environment

In the constant environment, the conditions were ideal for either line following or light following, i.e. the line was always present for line following and the light was always present for light following. Preliminary testing provided results from which the accuracy and reliability of the basic line following and light following techniques were assessed. The results were discussed in sections 7 and 12.1.

12.2.2 High noise level environment

In the high noise level environment, the ultrasonic sensors were exposed to different noise levels from workshop machines. The results obtained indicated that the sensors functioned accurately within a 2m range when exposed to noise levels between 70.3 db and 93.5 db. The noise parameters were accommodated for in the programming, for all four techniques, by setting the ultrasonic sensor conditions around a 1m threshold. This allowed the AGV to avoid collisions with obstacles and safely complete its tasks.

12.2.3 Unsatisfactory floor environment

In the unsatisfactory floor environment the line was absent in certain parts of the track, and at the junction required for changing direction, to reach the intended work station. The line variation program alternated from line following to light following in the absence of the line to allow the AGV to continue with task procedures. The program worked 80% reliably in scenario 1 (section 9.2.3) with the complete T junction, i.e. the line was present in the reverse and side-left directions. An average offshoot of 18mm was produced in scenario 2 (section 9.2.4), where only the horizontal part of the T junction was present. An average offshoot of 16mm was produced in scenario 3 (section 9.2.5), where only the vertical part of the T junction was present. The AGV depended completely on light following to change direction when the entire T junction was absent in the fourth scenario (section 9.2.6). An average deviation of 22mm was produced.

12.2.4 Light intensity environment

In the light intensity environment the light was varied from high, to low, to no light intensity. The light variation program facilitated changing from high to low light intensity, and changing from light to line following, when no light was present. An average deviation of 21mm was produced in scenario 1 (section 9.3.2) with the complete virtual T junction, i.e. light was present in the reverse and side-left directions. The AGV relied on line following to change direction in scenario 2 (section 9.3.3), where no light was present in the side-left direction and scenario 3 (section 9.3.4), where no light was present in the reverse direction. The AGV depended completely on line following to change direction when the virtual junction was absent i.e. light was absent in the reverse and side-left directions, in the fourth scenario (section 9.3.5).

12.3 Mechanical Design

The materials handling platform, docking stations, docking mechanisms, and charging station worked reliably. The charging station and docking station could be combined for future use. This could be done by attaching the positive and negative contacts to the docking mechanisms. The docking station frame would have to be enlarged to accommodate the weights and charging equipment. Alternatively, the charging equipment could be kept nearby and connected by longer wires. Combining the charging and docking stations would allow for a more efficient system as the AGV would be able to charge while docking.

The bumper attached to the AGV had a stroke of 15mm. A bumper with a larger stroke may have allowed for more flexibility. The bumper served only as a precautionary measure should the ultrasonic sensors not detect an obstacle. When compressed the limit switches behind the bumper signaled the control program to stop the AGV.

12.4 Electronic hardware and Software

Computer control of the system was implemented through user instructions. The computer made decisions about routing and other functions through the software. These functions included emergency stopping, docking, and interchanging guidance techniques when the environment was inappropriate for the present technique.

A problem experienced with the project was a lag in reaction time of the initial program, compiled in Visual Basic 6, which was used to control the AGV. The lag was due to VB's slower processing rate of data. Data from the sensors was only processed by the program a few seconds later than they were assimilated. Due to the lag in processing, the AGV could not follow the line accurately and sometimes moved clear off the path.

A new control program was written in C, an intermediate level software package. C allowed for faster processing of signals. The new program proved efficient and the AGV was able to complete its tasks accurately and reliably.

Delays were required after each write command (e.g. a motor board command), and in between each pair of write and read commands (e.g. a sensor board command). The delays did not deter the AGV's ability to perform tasks but if eliminated the AGV would be able to perform the tasks faster.

An alternate operating system such as Linux would prevent the use of delays as compared to a Windows operating system. Due to the capacity and structure of the present software it was advised not to change to Linux as the only advantage would be an increase in speed. The AGV's present speed of 0.125m/s was sufficient for task completion.

Converting the software from Windows to Linux would also lead to many compiler errors that would have to be solved and there was a limited time period available. Linux was therefore not implemented.

Another problem experienced, was that the commands to sensor boards were sent in series to each other. This slowed down the system, as delays were required in between the commands to different boards. If commands were sent in parallel to each other, the system might have worked faster. For example, the ultrasonic sensors could be called in the back ground to check for an obstacle, parallel to the line sensors, instead of being called in series. However, the telemetry unit did not provide for parallel processes and it is in need of improvement in this area.

Although Brainstem does offer parallel process ability, Brainstem modules only accommodate for two motors and have to be stacked for the use of more motors. As the AGV required five motor channels, this would require stacking three Brainstems. The modules have proved unreliable when stacked as they tend to malfunction.

Future work could entail including the entire control program on the robot CPU microcontroller, i.e. an embedded microchip. The computer can be used merely as an instruction interface for the user. This would enhance the efficiency of the system in terms of speed. Speed was not a major requirement of the project but it can be improved on in the future. The present speed did not deter the AGV's ability to execute tasks accurately and reliably. The use of an embedded microchip would reduce the flexibility to alter the programming.

The variable sensor system hardware was cost-effective as compared to commercial hardware available. The telemetry system hardware proved reliable and satisfied the requirements of the variable sensor system unlike the commercial hardware. The sensors used were low-cost. The system was robust and flexible as it functioned in unpredictable environments regardless of the changing parameters. The system was modular and adaptable as it could be implemented on another robot. The modular boxes for the electronic hardware, described in section 5.3, allowed for implementation on the other robot and an organised system.

12.5 Summary

This chapter described how the variable sensor system criteria, in section 1.5, were satisfied. Guidance and routing, testing, mechanical design, electronic hardware, and software aspects of the project were discussed. Recommendations were made to eliminate problems experienced. The unreliability of light following in a curve could be reduced by implementing a series of lights for the AGV to follow. A linear motion system could be implemented to prevent the relocation of

docking stations, when light following under different light intensities. This would ensure continuous motion of the AGV and task completion. Testing results were discussed for the high noise level, unsatisfactory floor and light intensity environments. Future mechanical design could include combining the docking station and charging station into one unit. Software lags occurred with VB therefore C was used instead. Delays were implemented between the write and read commands. The problem of sending data in series to the sensor boards was discussed. The implementation of an embedded microchip was suggested but presented programming inflexibility. The variable sensor system proved to be low-cost, robust, modular and flexible.

13 Conclusion

The objectives of this research project were achieved:

1. Mechatronic engineering principles were used to research, design, construct, assemble and test a variable sensor system for guidance and navigation of AGVs.

2. Routing algorithms and procedures for AGV movement and cooperation were researched and implemented.

3. Software was designed and implemented for the system to operate in changing environments.

4. The reliability and robustness of the variable sensor system was tested in three different environments:

   a. High noise level environment
   b. Light intensity environment
   c. Unsatisfactory floor environment

5. The performance of the variable sensor system was tested on another mobile robot in the same environment.

By applying the mechatronic approach of integrating mechanical, electronic, and software engineering principles, a variable sensor system was realized.

Mechanical modifications were made to the AGV to enable it to complete its tasks. These included a conveyor-type, materials handling platform, to interface with the conveyor, the attachment of an existing bumper for the protection of the AGV, and the design of docking stations, and a charging station.

Electronic hardware was implemented in the form of a telemetry system which consisted of motor boards, light sensor boards, line sensor boards, ultrasonic sensor boards and a limit switch board together with the motors and various sensors applied.

Software integration of the mechanical and electronic components enabled the AGV to function as a whole. The AGV was able to guide and navigate through the environment safely and reliably using routing procedures. The AGV was able to avoid obstacles and cooperate with other machines such as the conveyor. The software provided the AGV with the ability to alternate guidance and navigation techniques in order to complete its tasks in the changing environments.

The variable sensor system was tested in three different environments. The high noise level environment was accommodated for in the software. In the unsatisfactory floor environment, the system allowed the AGV to alternate from line following to light following in absent parts of the line track. In the light intensity environment, the system allowed the AGV to change from a high to low light intensity with the decrease of light. The system alternated from light following to line following in the absence of light. Testing of the system on another robot demonstrated that the system was flexible and modular.

In an Agile manufacturing system, production rates are determined by the reliable completion of tasks. Should parameters in the environment change, the AGV must be able to continue with its materials handling tasks in order to maintain production rates.

The variable sensor system facilitated the alternation of guidance and navigation techniques to compensate for the changing environment, thereby allowing the AGV to complete its materials handling tasks reliably. Materials handling operations, and hence the production rate in a manufacturing environment, would therefore be maintained through the utilization of a variable sensor system.

14 References

1. MSc Thesis: “Automated guided Vehicle System for manufacturing operations,” G. Bright, University of KwaZulu Natal, 1989.

2. PhD Thesis: “Guidance techniques, data transfer processing, path planning and computer aided control for automated guided vehicles,” G. Bright, University of KwaZulu Natal, 1993.

3. MSc Thesis, “Communication, mapping and navigational aspects for a free-ranging automated guided vehicle,” J. Asbury, University of KwaZulu Natal, 1992.

4. “Mobile Robot Navigation,” J. Dixon, O. Henlich, Imperial College, London, 1997.

5. “Mobile Robot Positioning and Sensors and Techniques,” J. Borenstein, , H.R. Everett, L. Feng, D. Wehe, Journal of Robotic Systems, Special Issue on Mobile Robots, 14/4, 231 – 249, 1996.

6. “Sensors for Mobile Robots: Theory and Application,” H. R. Everett, A K Peters Ltd., Wellesley, MA, 1995.

7. “Intelligent Mobile Robot Navigation,” Cuesta, Federico, Ollero, Aníbal, Series: Springer tracts in advanced robotics, 1 edition, Vol. 16, Springer, 2005.

8. “Simultaneous Localization and Mapping (SLAM): Part I The Essential Algorithms,” H. Durrant-Whyte, T Bailey, IEEE R&A Magazine, 2006.

9. “Simultaneous Localization and Mapping (SLAM): Part 2, State of the Art,” H. Durrant-Whyte, T Bailey, IEEE R&A Magazine, 2006.

10. “Choosing the Detector for your Unique Light Sensing Application,” L. Godfrey, PerkinElmer, Inc, 1998-2003. Available at - http://www.engr.udayton.edu/faculty/jloomis/ece445/topics/egginc/tp4.html.

11. “Mechatronics: Electronic Control Systems in Mechanical and Electrical Engineering”, by W. Bolton; third edition, 2003. Pearson Education Limited, England.

12. Automation, production systems, and computer integrated manufacturing, Mikell P. Groover, Prentice Hall International Editions, Inc, 1987, USA.

13. MSc Thesis: Johannes Petrus Jacobs, Intelligent integration of an industrial robot and an automated guided vehicle, RAU, 2001.

14. MSc Thesis: "A Mechatronic approach to develop the concept of materials handling system for a reconfigurable manufacting environment," P Naidu, Department of Mechanical Eng, University of KwaZulu Natal, South Africa, 2007.

15. Research Project: Laser navigation system for an omnidiectional robotic platform, Ensign Matthew J Tinnelly, Massey University, New Zealand, 2003.

16. Research Project: "Cooperation between mobile robots," R. Fisk, Massey University, New Zealand, 2004.

17. PhD Thesis: "Modular mechatronic robotic plug 'n' play controller," Jonathan Richard Zyzalo, Massey University, New Zealand, 2003.

18. PhD Thesis: "Predictive Autonomous Robot Navigation," A Foka, Department of Computer Science, University of Crete, Crete, 2005.

19. "Autonomous Mobile Robots – Vehicles with cognitive control," A Meystel, World Scientific Series in Automation-Vol1, World Scientific Publishing Co. Pte. Ltd, Singapore, 1991
Didn't use

20. "Introduction to robotics 1," P. Lopez and J. Foulic, Glentop Publishers Ltd, Great Britain, 1986.

21. "Sensor Technology Handbook," J. Wilson, First edition, Newnes/Elsevier, 2004.

22. "Choosing An Ultrasonic Sensor For Proximity Or Distance Measurement; Part 1: Acoustic Considerations; Part 2: Optimizing Sensor Selection" D. P. Massa, Sensors, February and March 1999. http://www.sensorsmag.com/sensors/article/articleDetail.jsp?id=321383

23. “C - How to program,” H.M. Deitel, P.J. Deitel, Fifth edition, Prentice Hall, USA, 2005.

24. “Visual Basic 6 - How to program,” H.M. Deitel, P.J. Deitel, Third edition, Prentice Hall, USA, 2005.

25. “A review of agile manufacturing systems”, L. M. Sanchezy, R Nagiy, Int. J. Prod. Res., 2001, vol. 39, no. 16, pages 3561-3600.

26. “FMS at work,” J. Hartley, IFS (Publications) Ltd., UK, North-Holland, a division of Elsevier Science Publishers B.V, 1984.

27. “MINERVA: A Second-Generation Museum Tour-Guide Robot,” Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B. Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles Rosenberg, Nicholas Roy, Jamieson Schulte, Dirk Schul, Proceedings of IEEE International Conference on Robotics and Automation, 1999, Vol. 3, pages 1999-2005.

28. “Mechatronics – An established discipline or a concept in need of direction,” D. Bradley, Elsevier Science Ltd, 2000.

29. “Manufacturing Flexibility: a literature review,” A. de Toni, S. Tonchia. International Journal of Production Research, 1998, vol. 36, no. 6, pages 1587-617.

30. “Computer control of manufacturing systems,” Y. Korem, McGraw Hill, Inc. 1983.

31. “Logistics and Supply Chain Management: Creating Value-Adding Networks,” M. Christopher, Financial Times Prentice Hall, 2005.

33. “Principles of Computer-Integrated Manufacturing,” J. B. Waldner, John Wiley & Sons, 1992.

34. “Agile competitors and virtual organizations – Strategies for enriching the customer,” L. Goldman, R.L. Nagel and K. Preiss, Van Nostrand Reinhold, 1995.

35. http://www.garmin.com/aboutGPS/

36. http://searchsmb.techtarget.com/sDefinition/0,,sid44_gci537791,00.html

37. http://www.esacademy.com/faq/i2c/general/i2cproto.htm

38. http://www.ranchbots.com/line_follower/line_follower.htm

39. http://www.egeminusa.com/pages/agvs/agvs_battery_charging.html

40. http://www.sick.com/us/news/archive/articles/1/en.html

41. http://www.agve.se/

42. http://www.egeminusa.com/pages/agvs/agvs_ulv.html

43. http://www.fmcsgvs.com/index2.htm

44. www.hksystems.com

45. http://www.agvp.com

46. http://www.amerden.com/

47. http://www.technologystudent.com/elec1/ldr1.htm

48. http://www.cs.brown.edu/~tld/courses/cs148/02/sonar.html

49. http://www.mstracey.btinternet.co.uk/technical/Theory/theorysensors.htm

50. http://www.mug.jhmi.edu/mirrors/infoalley/0995/18/pci.html

51. http://www.adlinktech.com/PD/big5/PD_detail.php?pid=630

52. http://zone.ni.com/devzone/cda/tut/p/id/4811

53. http://www.acroname.com/robotics/info/articles/sharp/sharp.html#e2

54. http://sine.ni.com/nips/cds/view/p/lang/en/nid/13087

55. http://www.eagledaq.com/display_product_2359.htm

56. http://www.solarbotics.net/library/circuits/sensors_prox.html#ir

57. http://www.bastiansolutions.com/products/automated-guided-vehicles/default.asp

58. http://www.acroname.com/brainstem/ref/ref.html

59. http://www.mobilerobots.com/

60. http://mechatronics.rpi.edu/

61. http://www.evolution.com/er1/

62. http://www.nada.kth.se/~kai-a/

63. http://www.udt.com

64. http://encyclobeamia.solarbotics.net/articles/phototransistor.html

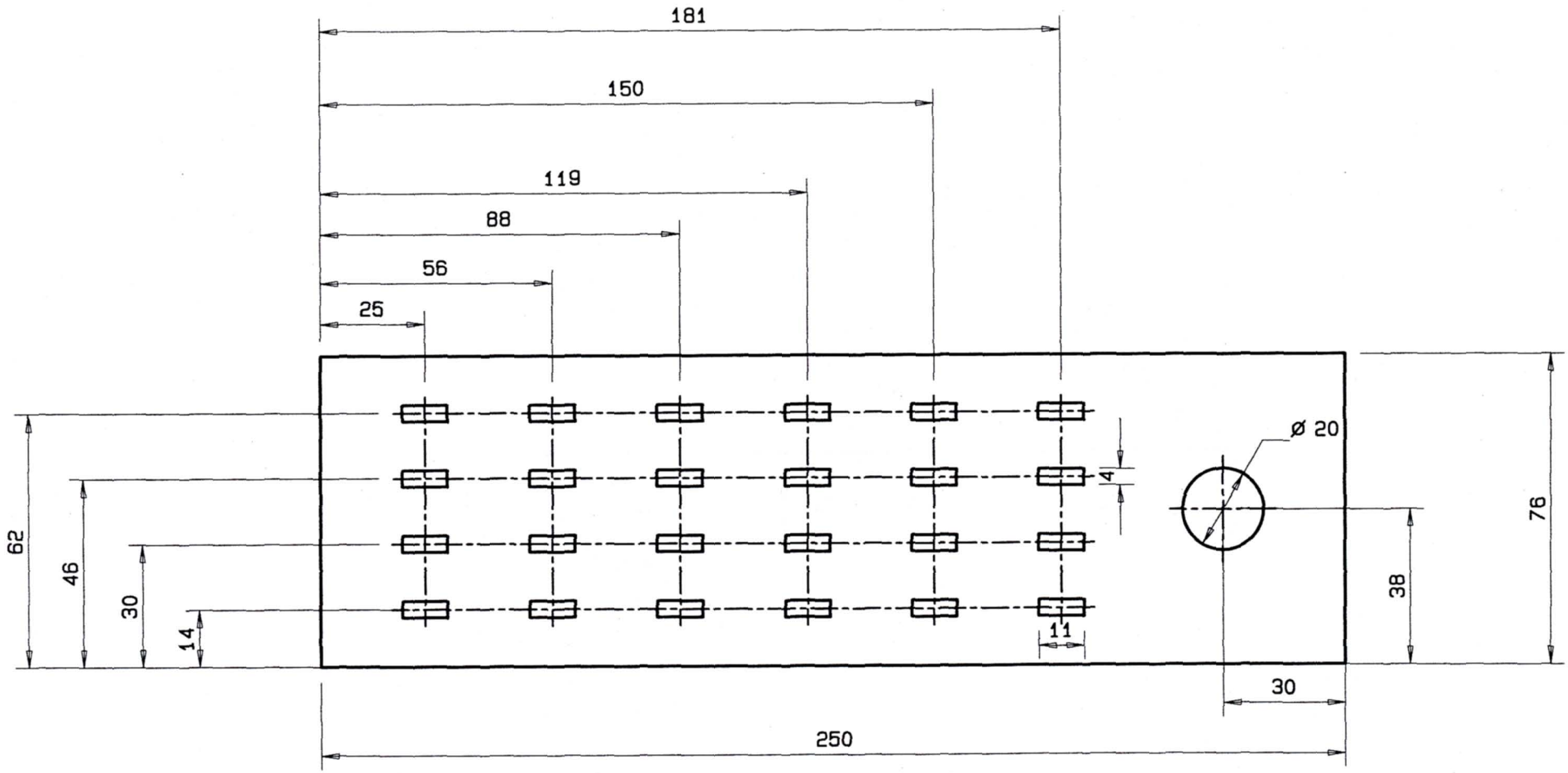
65. http://www.acroname.com/robotics/info/concepts/pwm.html

66. http://www.michaelhorowitz.com/Linux.vs.Windows.html

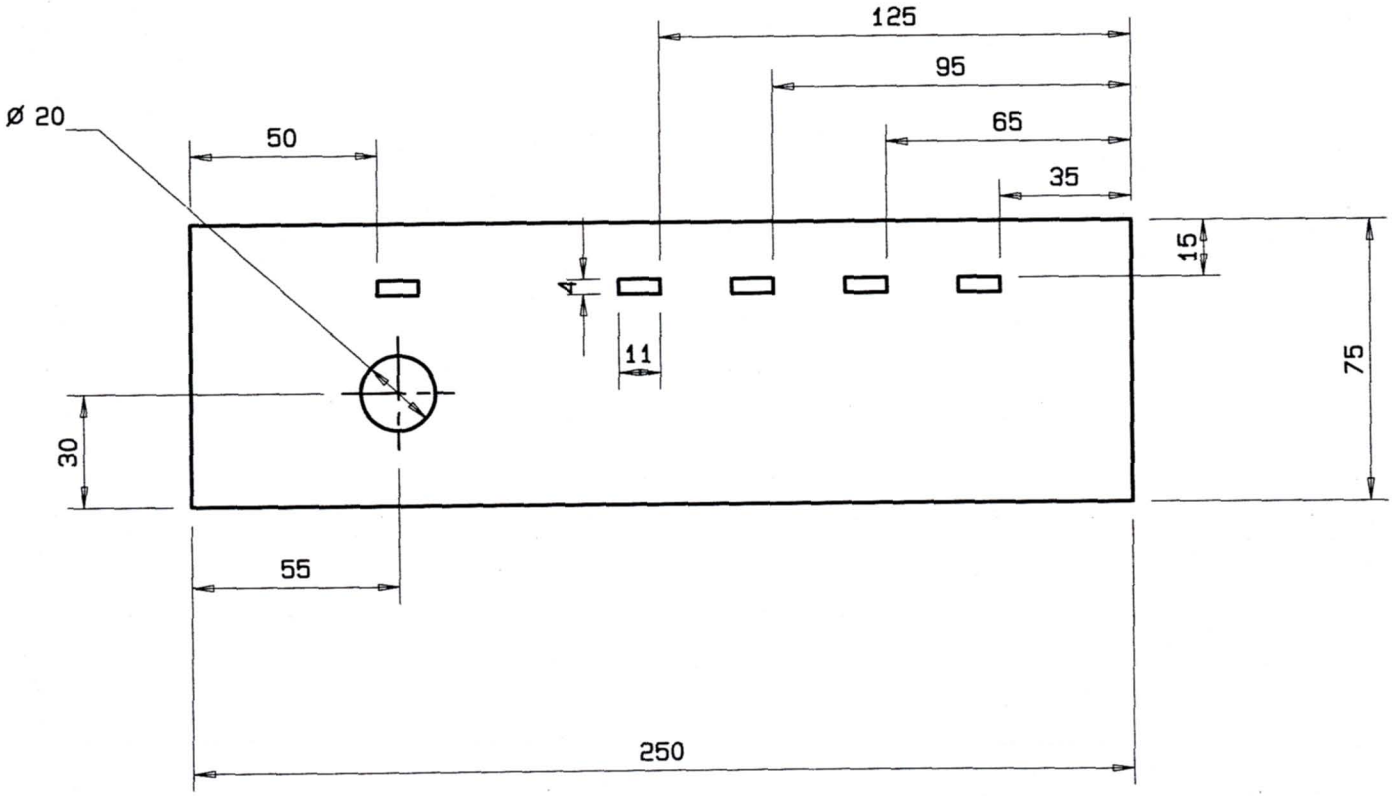
67. http://www.kpsec.freeuk.com/components/other.htm

Appendices

A. Mechanical Drawings

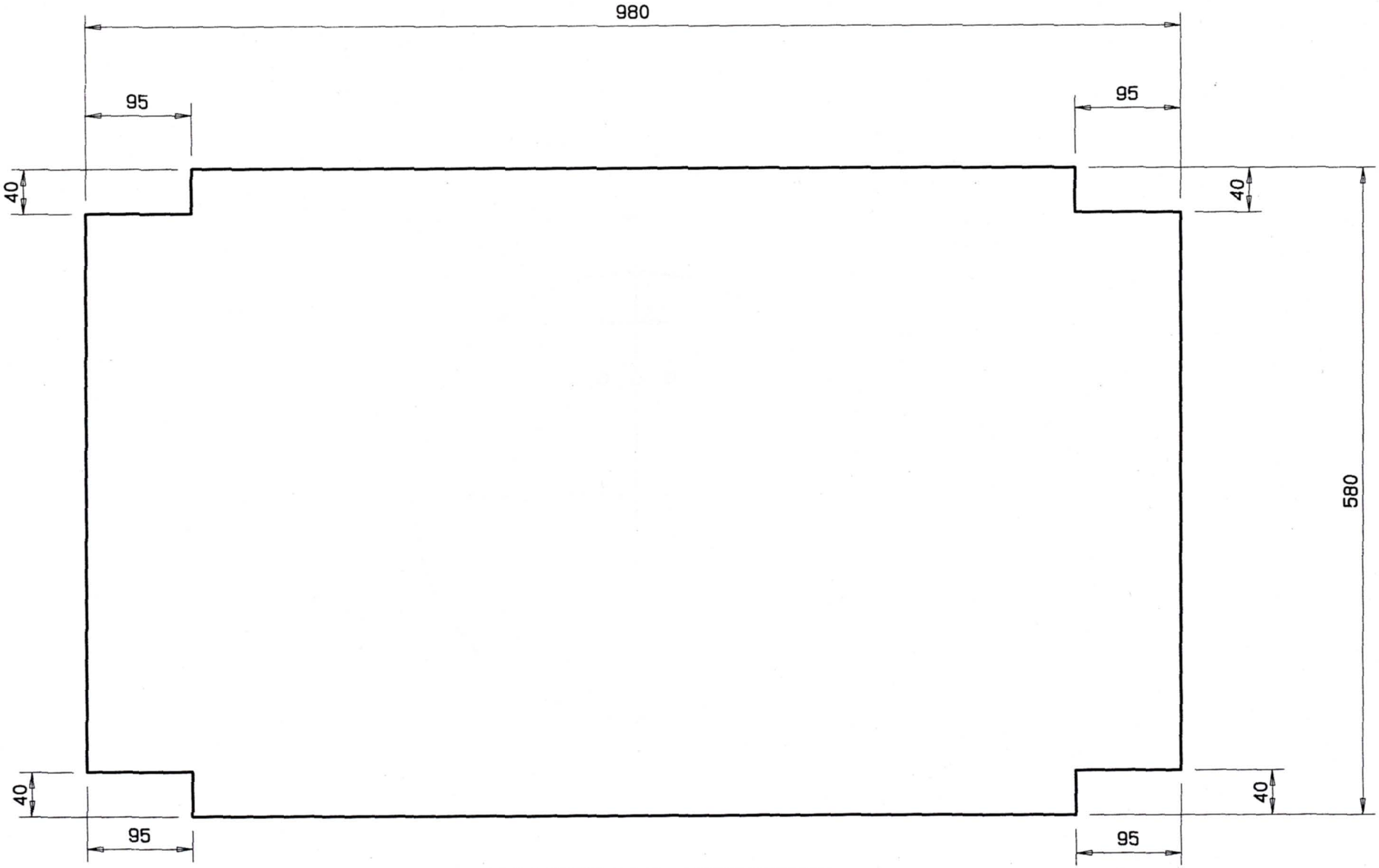


School of Mechanical Engineering, University of KwaZulu-Natal		DATE	CHECKED	SCALE 1 : 1.5 UNITS : mm	PROJECT: AGV
	Project Supervisor			STUDENT NAME	No.
	W'Shop Technician			E-MAIL	TITLE: COVER
	Technical Manager			TEL NO.	



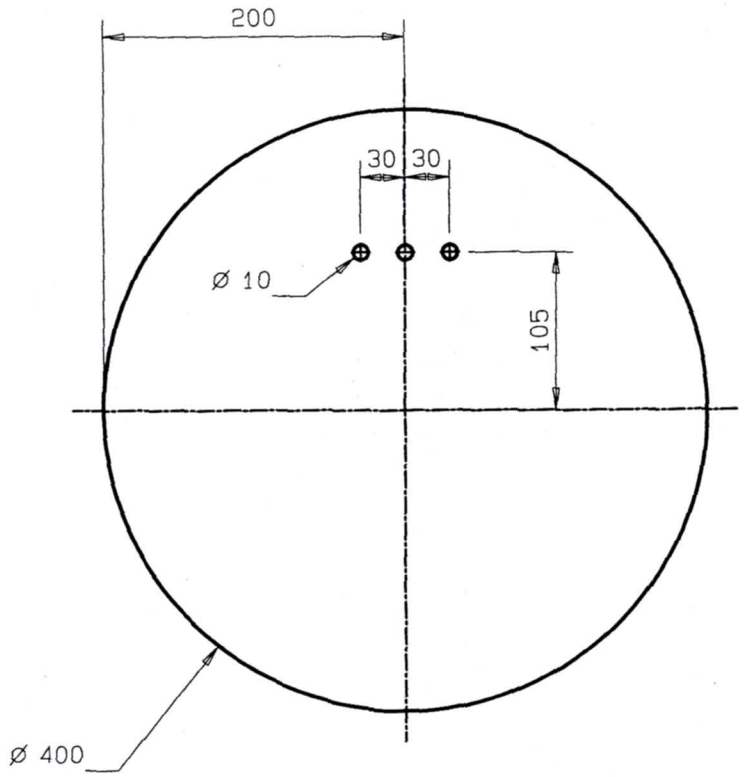
School of Mechanical Engineering		DATE	CHECKED	SCALE 1 : 1	UNITS : mm	PROJECT
UNIVERSITY OF KWAZULU-NATAL	Project Supervisor			STUDENT NAME		TITLE COVER
	W'Shop Technician			E-MAIL		
	Technical Manager			TEL NO.		

No.



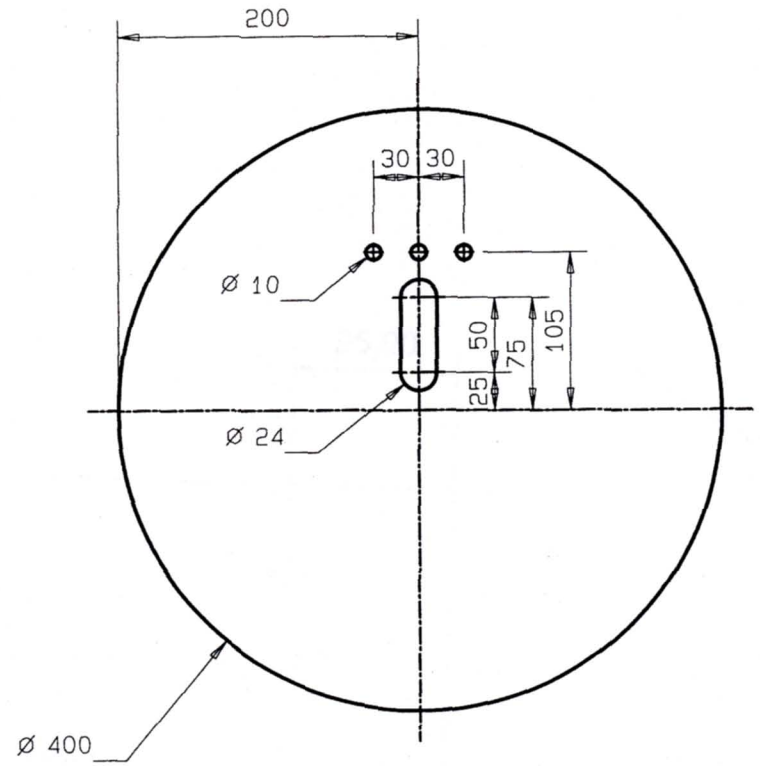
MATERIAL: PERSPEX

School of Mechanical Engineering, University of KwaZulu-Natal		DATE	CHECKED	SCALE 1 : 5 UNITS : mm	PROJECT AGV / No.
	Project Supervisor			STUDENT NAME ARDISHA	
	W'Shop Technician			E-MAIL 202513794	TITLE COVER
	Technical Manager			TEL NO. EXT 260 1227	



MATERIAL: PERSPEX
THICKNESS: 4 mm

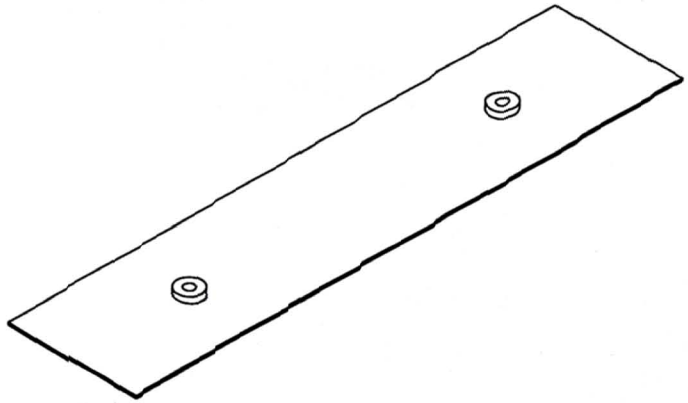
School of Mechanical Engineering		DATE	CHECKED	SCALE 1 : 5 UNITS : mm	PROJECT AGV / No. 1
UNIVERSITY OF KWAZULU-NATAL	Project Supervisor			STUDENT NAME ARDISHA	
	W'Shop Technician			E-MAIL 202513794	TITLE BASE
	Technical Manager			TEL NO. EXT 260 1227	



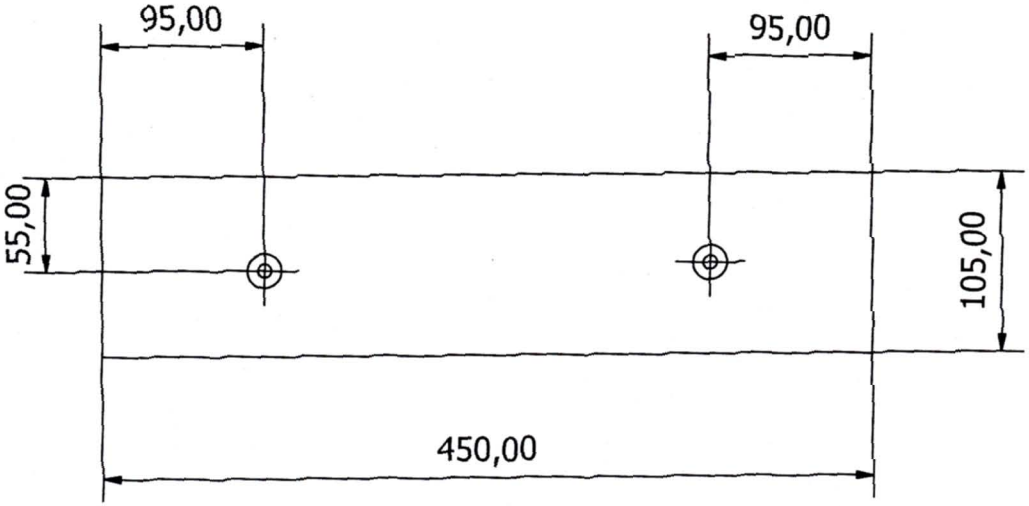
MATERIAL: ALUMINIUM
THICKNESS: 4 mm

School of Mechanical Engineering		DATE	CHECKED	SCALE 1 : 5	UNITS : mm	PROJECT AGV / No. 1
	Project Supervisor			STUDENT NAME ARDISHA		
	W'Shop Technician			E-MAIL 202513794		TITLE BASE
	Technical Manager			TEL NO. EXT 260 1227		

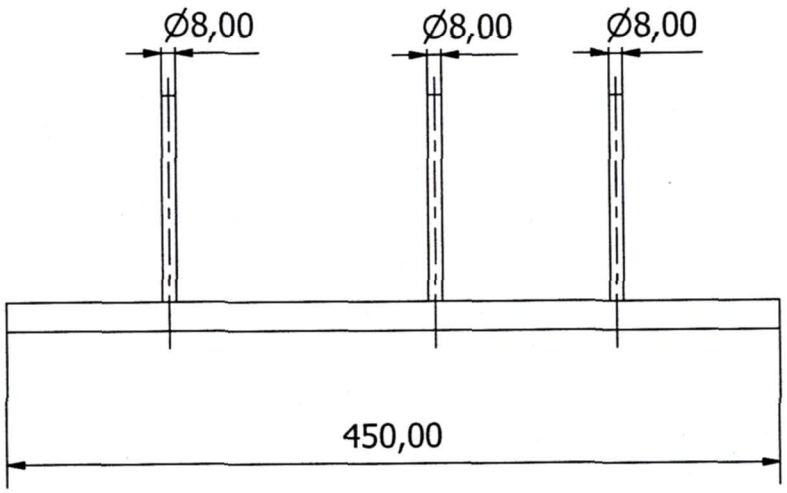
UNIVERSITY OF KWAZULU-NATAL

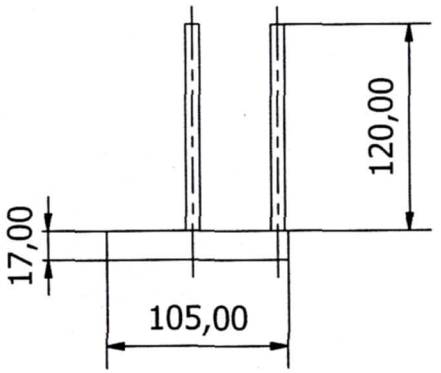


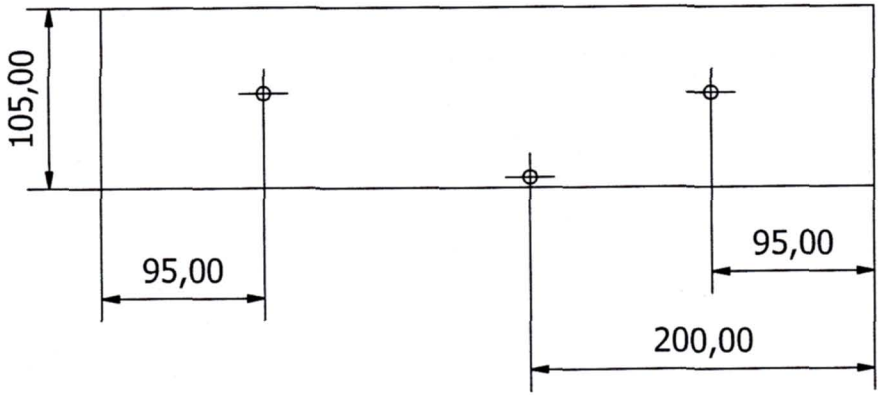
MATERIAL: MILD STEEL

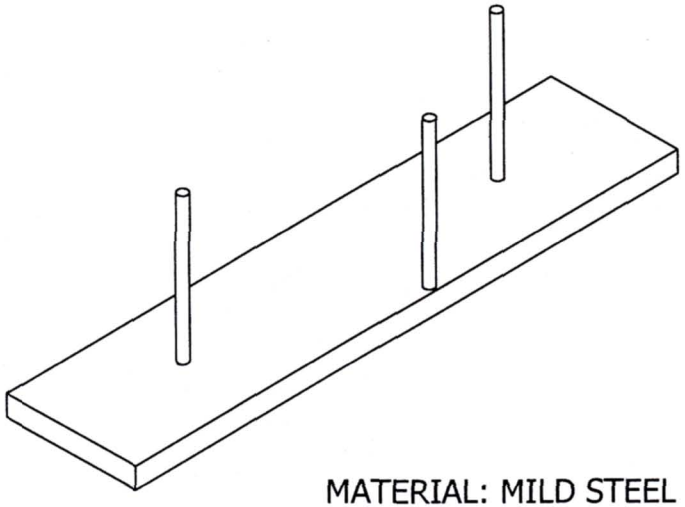


UNIVERSITY OF KWA-ZULU NATAL School of Mechanical Engineering		DATE	CHECKED	SCALE 1:1 UNITS:mm	PROJECT: AGV / No.: 1
	Project Supervisor			STUDENT NAME	
	W'Shop Technician			E-MAIL	TITLE: DOCKING STATION PLATE
	Technical Manager			TEL NO.	



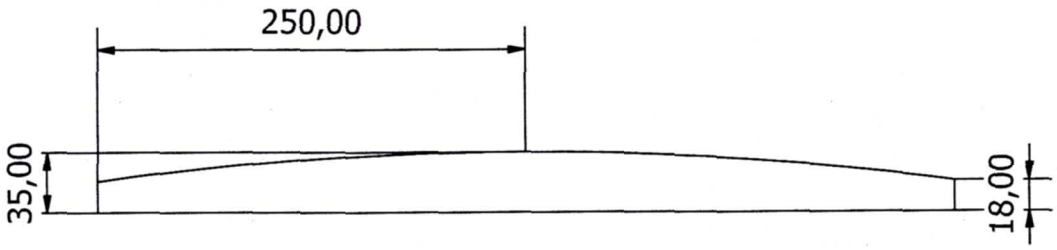


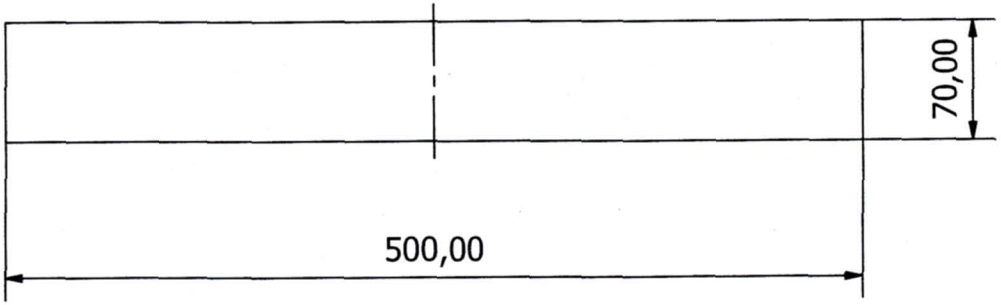


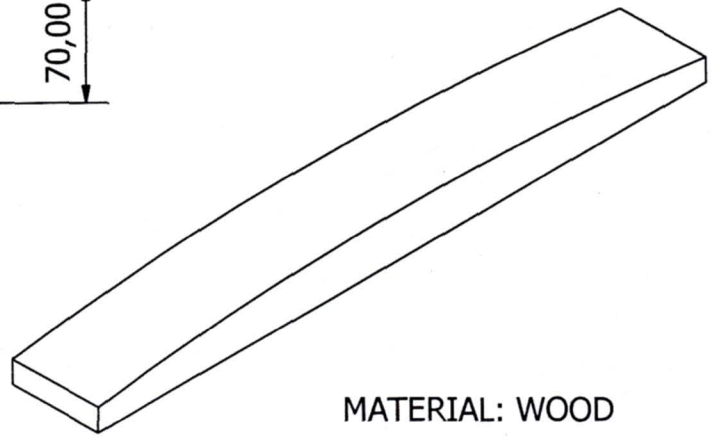


MATERIAL: MILD STEEL

UNIVERSITY OF KWA-ZULU NATAL School of Mechanical Engineering		DATE	CHECKED	SCALE 1:1 UNITS:mm	PROJECT: AGV / No. 1
	Project Supervisor			STUDENT NAME	
	W'Shop Technician			E-MAIL	TITLE: MALE DOCKING MECHANISM PLATE
	Technical Manager			TEL NO.	



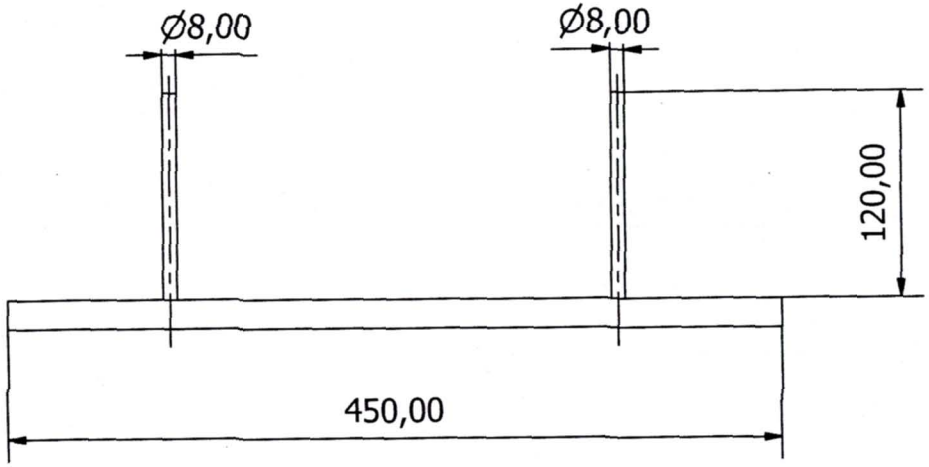


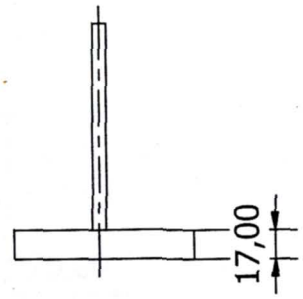


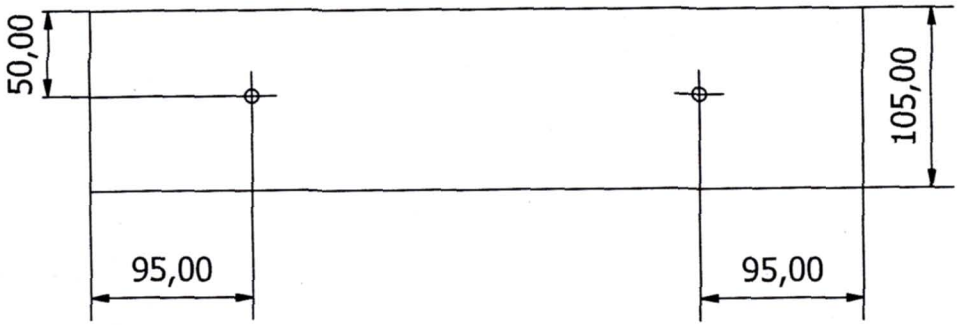
MATERIAL: WOOD

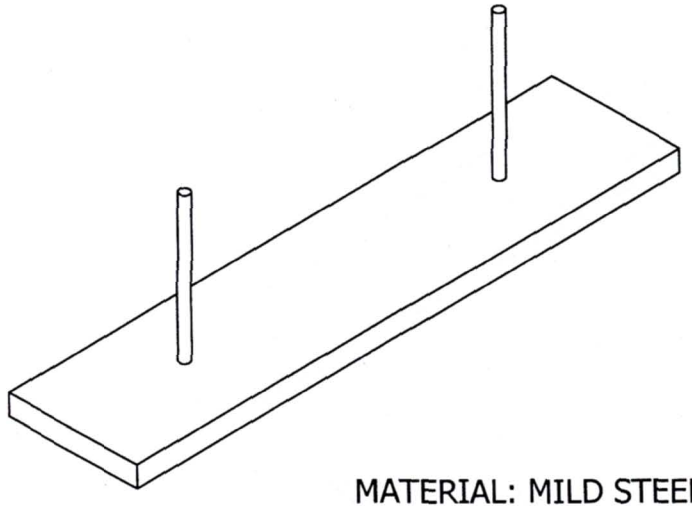
UNIVERSITY OF KWA-ZULU NATAL School of Mechanical Engineering		DATE	CHECKED	SCALE 1:1 UNITS:mm	PROJECT MALE DOCKING MECHANISM
	Project Supervisor			STUDENT NAME	
	W'Shop Technician			E-MAIL	TITLE
	Technical Manager			TEL NO.	

No. 1





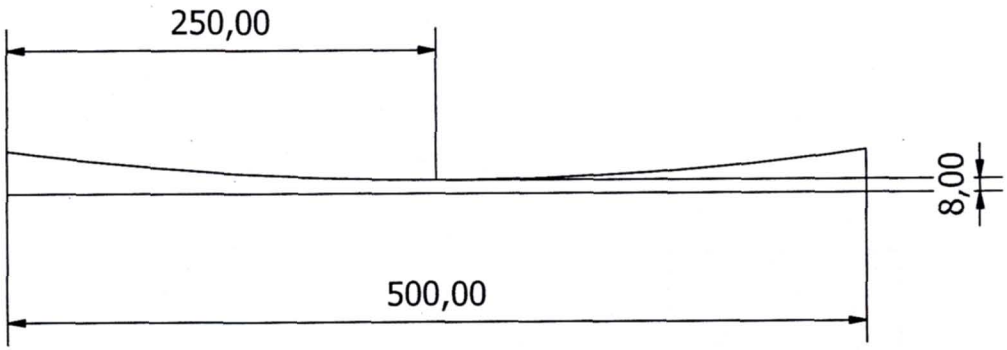


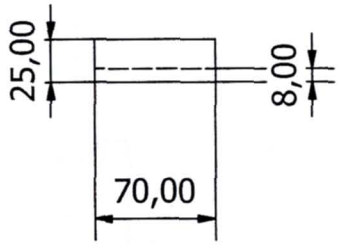


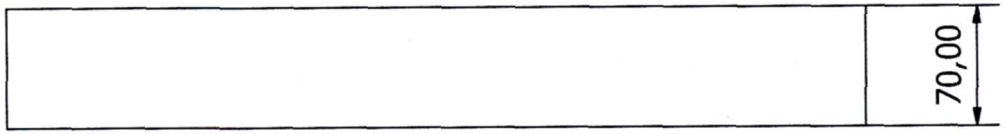
MATERIAL: MILD STEEL

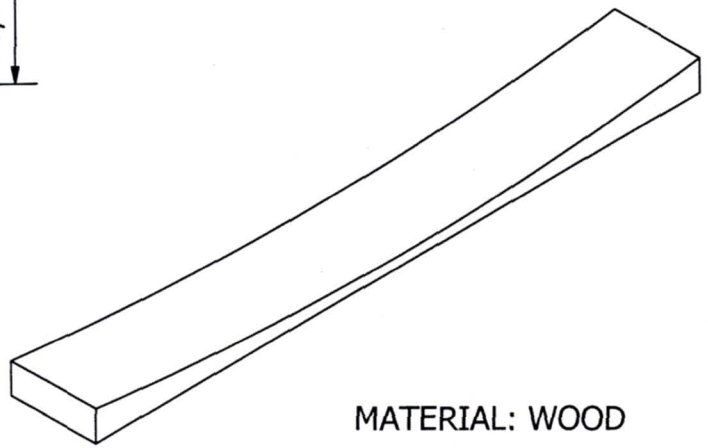
UNIVERSITY OF KWA-ZULU NATAL School of Mechanical Engineering		DATE	CHECKED	SCALE 1:1	UNITS:mm	PROJECT: FEMALE DOCKING MECHANISM PLATE
	Project Supervisor			STUDENT NAME		TITLE
	W'Shop Technician			E-MAIL		
	Technical Manager			TEL NO.		

No. 1



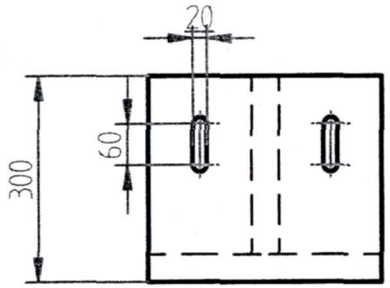


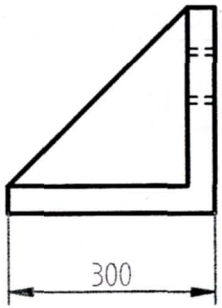


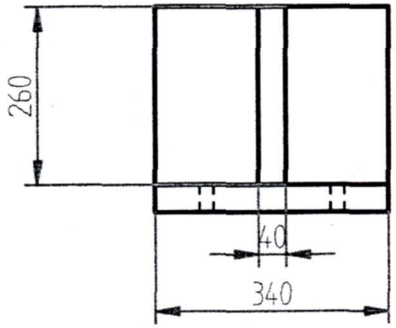


MATERIAL: WOOD

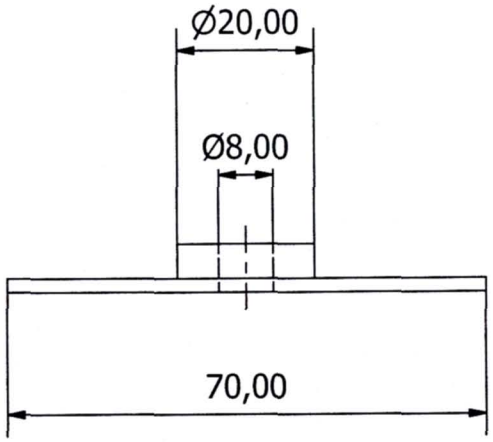
UNIVERSITY OF KWA-ZULU NATAL School of Mechanical Engineering		DATE	CHECKED	SCALE 1:1	UNITS:mm	PROJECT AGV	No. 1
	Project Supervisor			STUDENT NAME			
	W'Shop Technician			E-MAIL		TITLE FEMALE DOCKING MECHANISM	
	Technical Manager			TEL NO.			

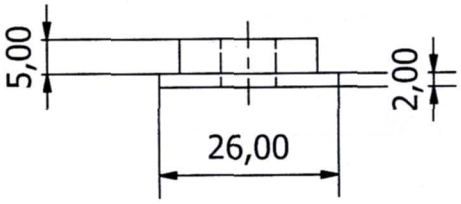


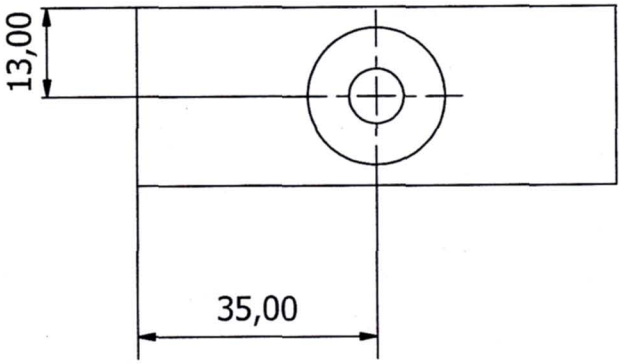


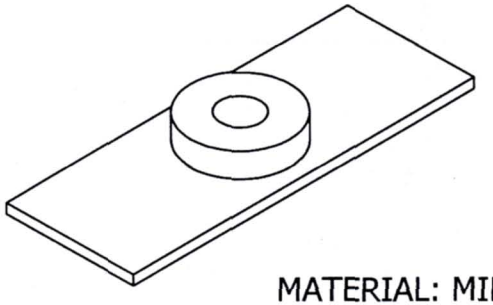


UNIVERSITY OF KWA-ZULU NATAL School of Mechanical Engineering		DATE	CHECKED	SCALE 1 : 1	UNITS : mm	TITLE: DOCKING STATION
	W'Shop Technician			MATERIAL: MILD STEEL	NOTES:	No. 3
	Technical Manager			STUDENT NAME : A PANCHAM		TEL No. : 083 639 531
	Project Supervisor			Project : AGV		E-MAIL : 202513794@ukzn.ac.za





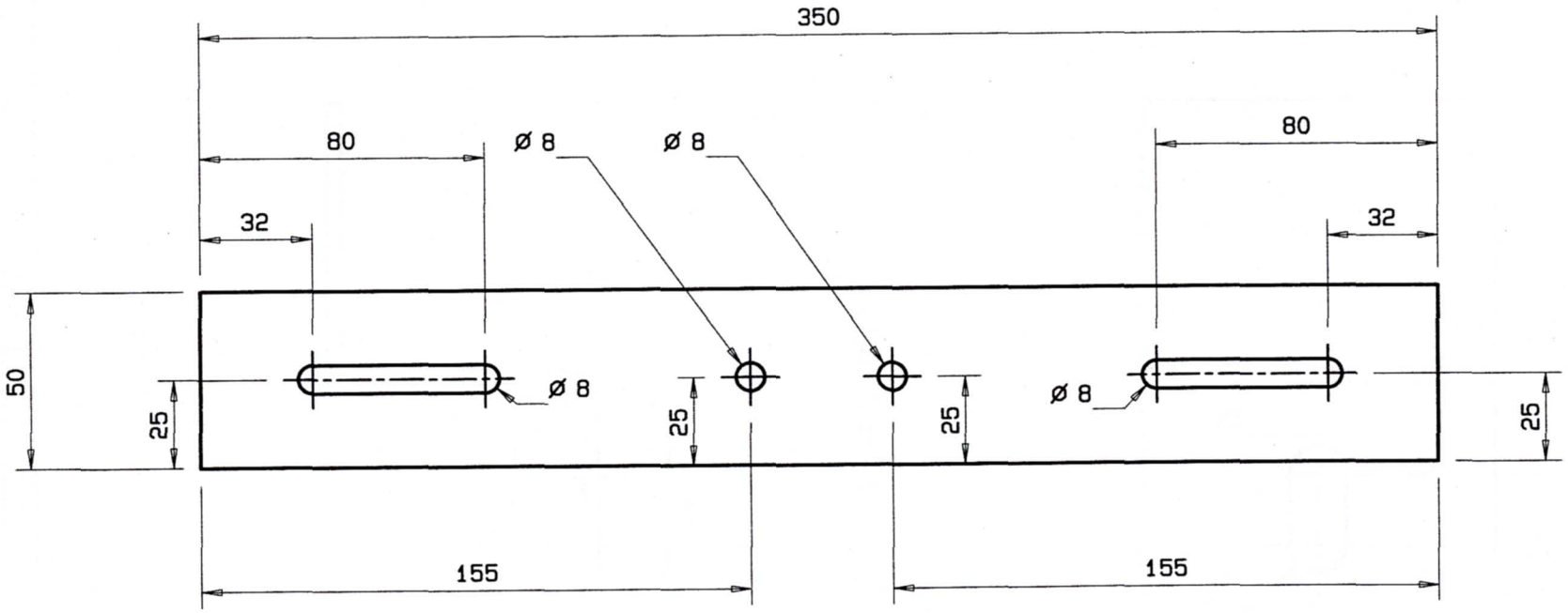




MATERIAL: MILD STEEL

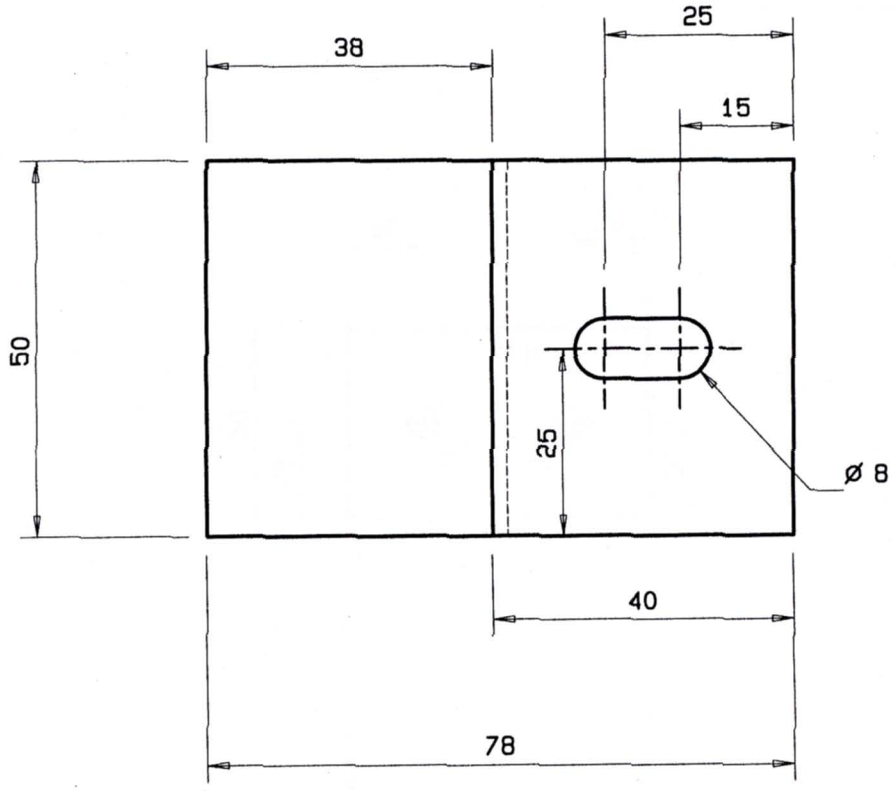
UNIVERSITY OF KWA-ZULU NATAL School of Mechanical Engineering		DATE	CHECKED	SCALE 1:1	UNITS:mm	PROJECT AGV	No. 1
	Project Supervisor			STUDENT NAME			
	W'Shop Technician			E-MAIL		TITLE BUSH PLATE	
	Technical Manager			TEL NO.			

PLATE THICKNESS 2mm

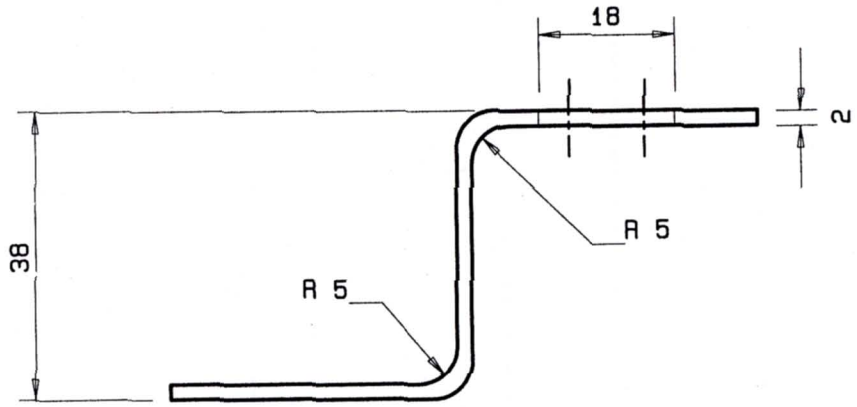


School of Mechanical Engineering UNIVERSITY OF KWAZULU-NATAL		DATE	CHECKED	SCALE 1 : 2	UNITS : mm	PROJECT
	Project Supervisor			STUDENT NAME		
	W'Shop Technician			E-MAIL		TITLE TOP PLATE
	Technical Manager			TEL NO.		

No. 1/3



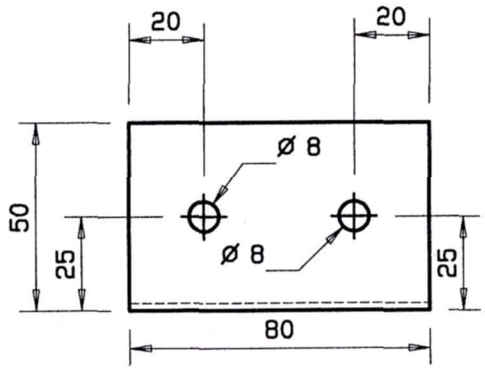
TOP VIEW



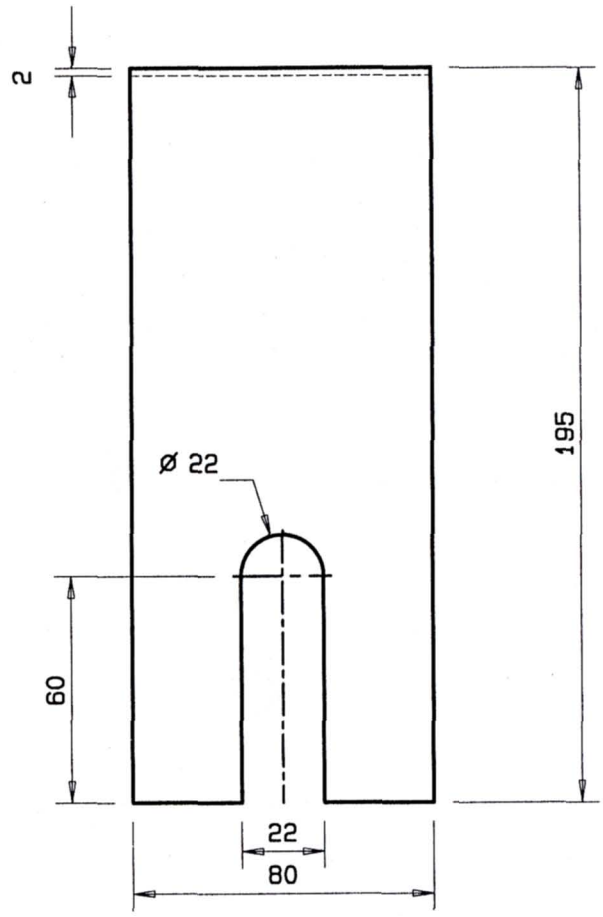
FRONT VIEW

QUANTITY : 2

School of Mechanical Engineering UNIVERSITY OF KWAZULU-NATAL	Orthographic Projection	SCALE 1 : 1		TITLE *SIDES*
		UNITS : mm		
		Date	Checked	PROJECT
	Project Supervisor			
	W'Shop Technician			STUDENT NAME
	Technical Manager			TEL NO. E-MAIL



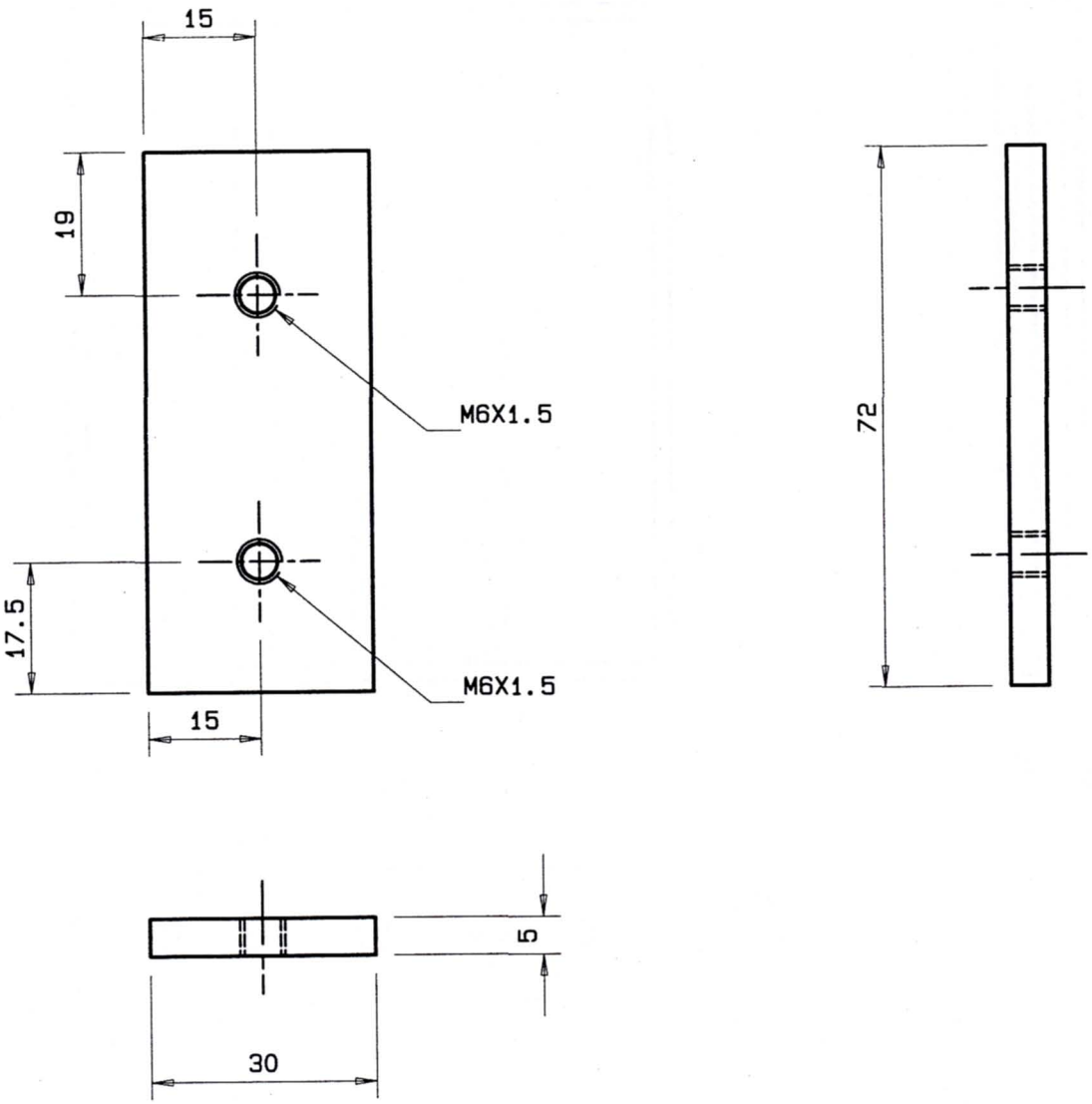
TOP VIEW



FRONT VIEW

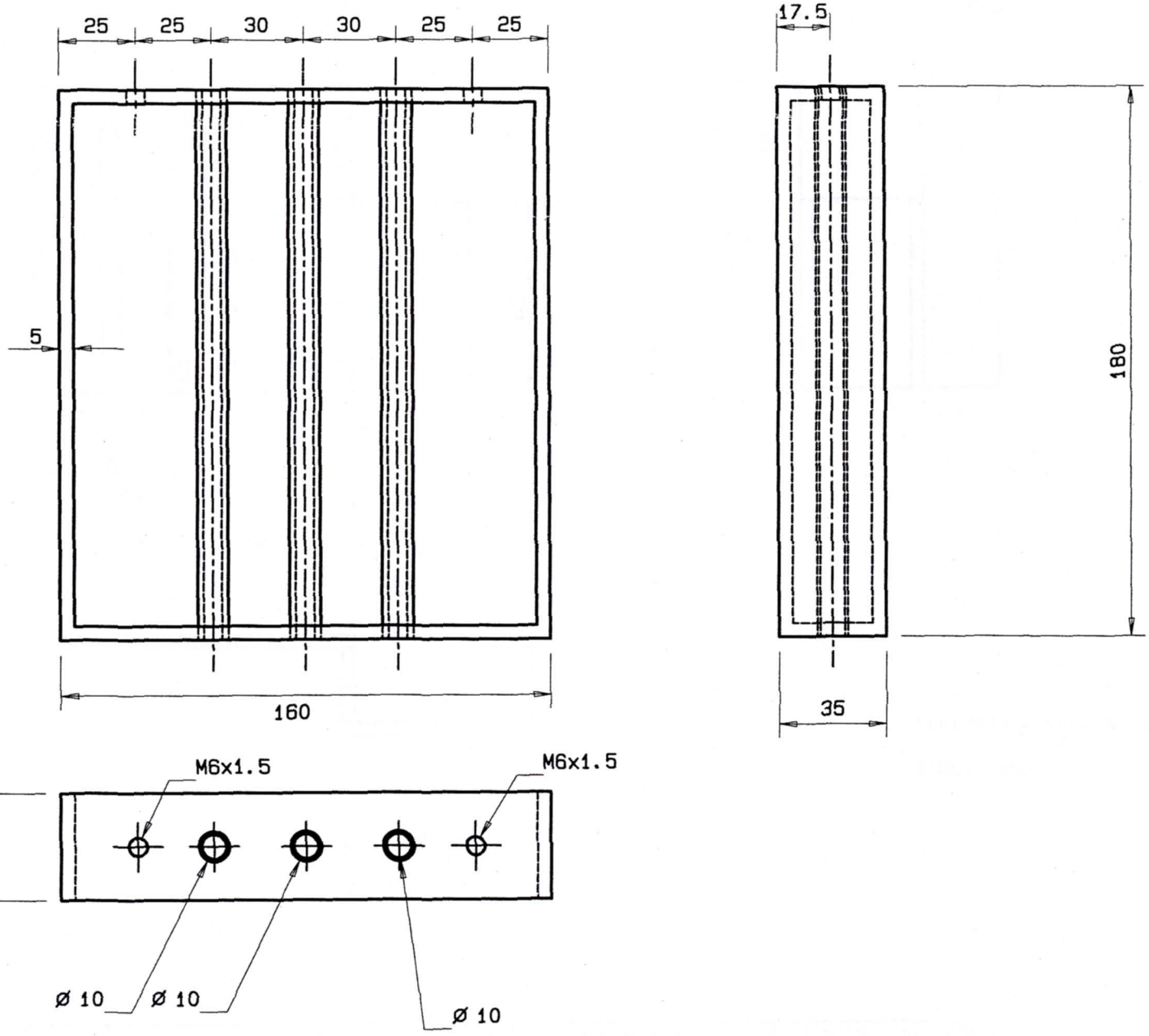
MATERIAL: MILD STEEL SHEET. THICKNESS 2mm

School of Mechanical Engineering UNIVERSITY OF KWAZULU-NATAL	Orthographic Projection	SCALE 1 : 2		TITLE LIGHT BRACKET
		UNITS : mm		
		Date	Checked	PROJECT
	Project Supervisor			
	W'Shop Technician			STUDENT NAME
	Technical Manager			TEL NO. E-MAIL

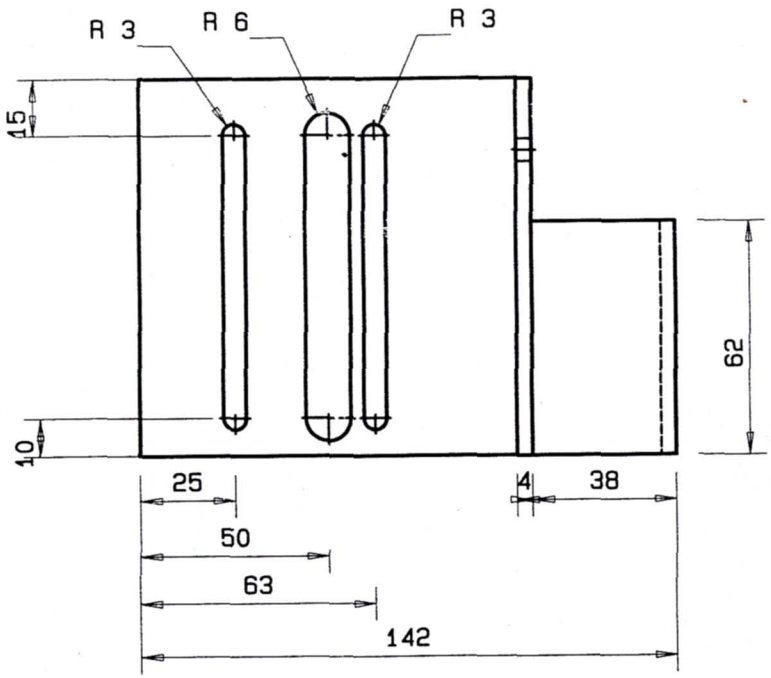


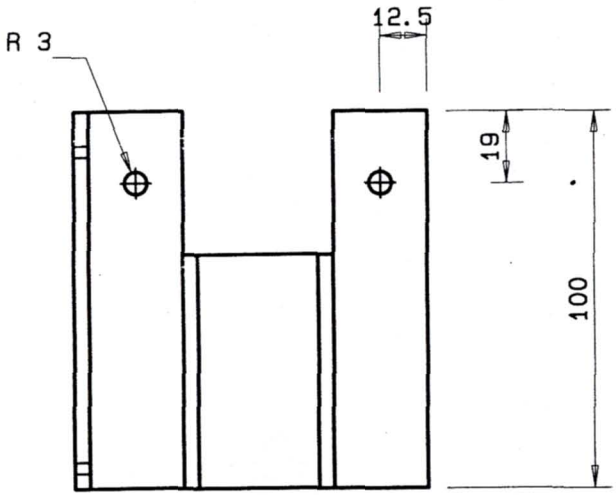
QUANTITY REQUIRED: 4

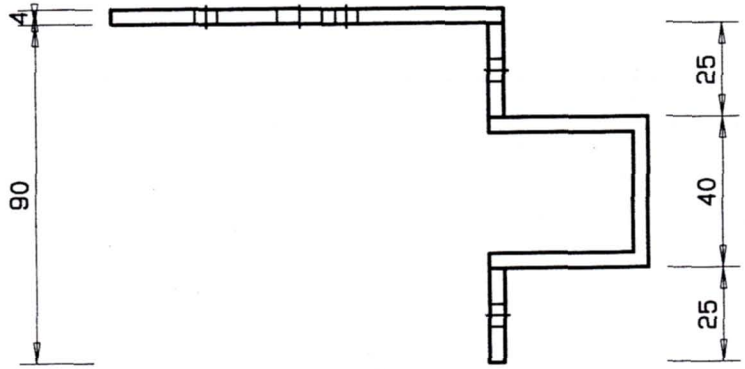
School of Mechanical Engineering		DATE	CHECKED	SCALE 1 : 1 UNITS : mm	PROJECT AGV / No.
UNIVERSITY OF KWAZULU-NATAL	Project Supervisor			STUDENT NAME ARDISHA	
	W'Shop Technician			E-MAIL 202513794	TITLE PLATE
	Technical Manager			TEL NO. EXT 260 1227	



School of Mechanical Engineering UNIVERSITY OF KWAZULU-NATAL		DATE	CHECKED	SCALE 1 : 2 UNITS : mm	PROJECT AGV
	Project Supervisor			STUDENT NAME ARDISHA	No. 2
	W'Shop Technician			E-MAIL 202513794	TITLE SENSOR HOLDER
	Technical Manager			TEL NO. EXT 206 1227	







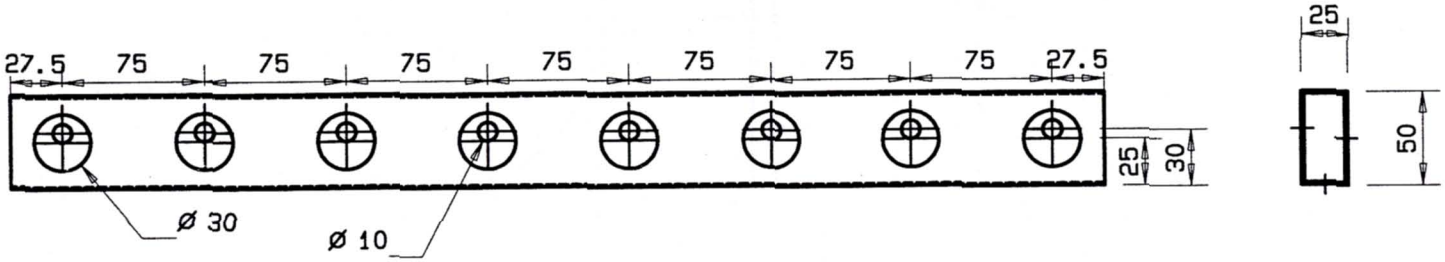
MATERIAL : MILD STEEL SHEET METAL

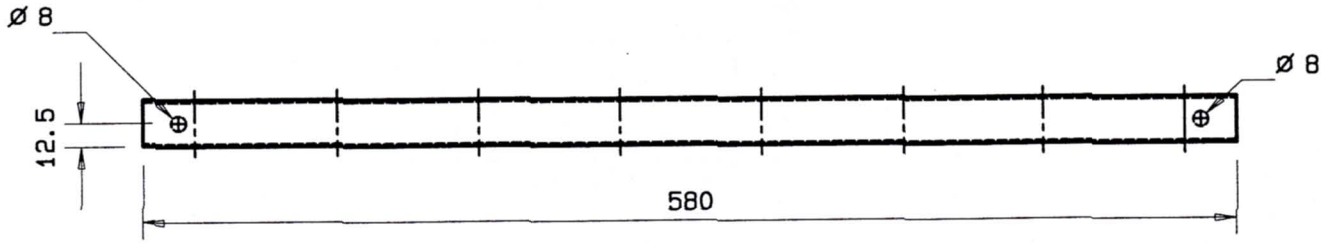
THICKNESS: 4mm

University of KwaZulu-Natal **School of Mechanical Engineering**		DATE	CHECKED	SCALE 1 : 2 UNITS : mm	PROJECT MECHATRONICS : AGV
	Project Supervisor			STUDENT NAME A PANCHAM	
	W'Shop Technician			E-MAIL 202513794@ukzn.ac.za	TITLE MOTOR BRACKET
	Chief Technician			TEL NO. 083 639 9531 / 260 1227	

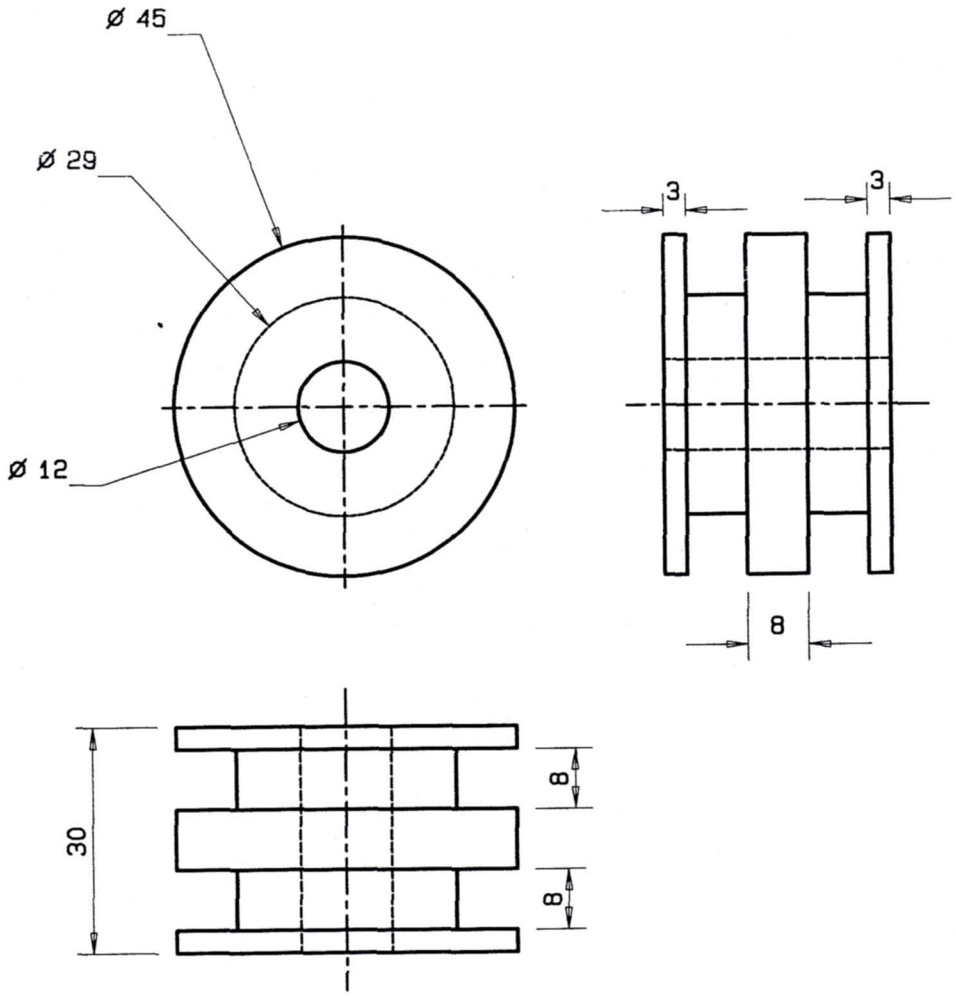
No. 1

MATERIAL:
RECT STEEL TUBE
50 X 25 X 1.6 mm



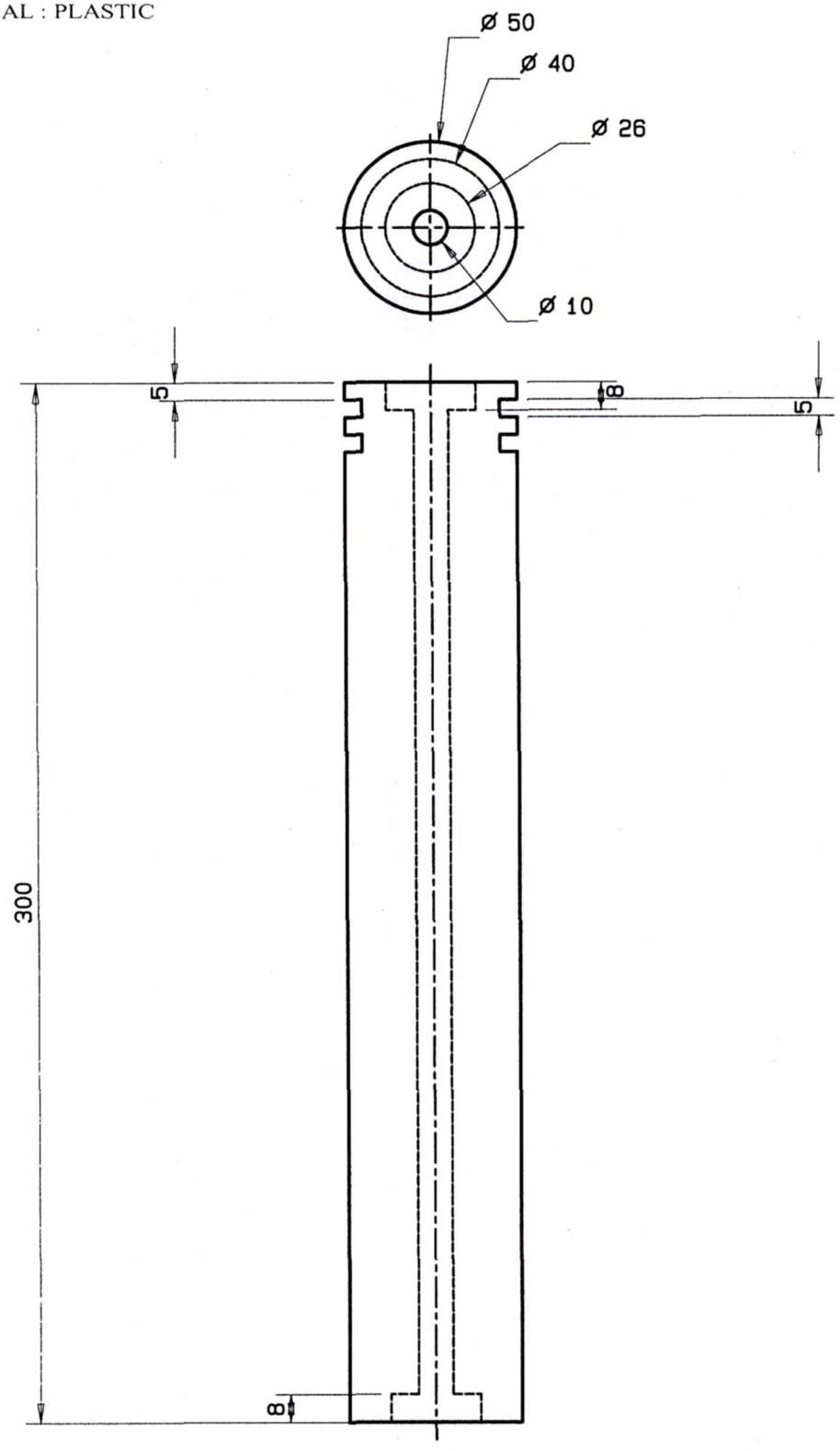


		DATE	CHECKED	SCALE 1 : 4	UNITS : mm	PROJECT	No.
School of Mechanical Engineering	Project Supervisor			STUDENT NAME	A PANCHAM	MECHATRONICS : AGV	2
UNIVERSITY OF KWAZULU-NATAL	W'Shop Technician			E-MAIL	202513794@ukzn.ac.za	TITLE	
	Technical Manager			TEL NO.	083 639 9531 / 260 1227	CONVEYOR PLATFORM FRAME	



University of KwaZulu-Natal School of Mechanical Engineering		DATE	CHECKED	SCALE 1 : 1	UNITS : mm	PROJECT	No.
	Project Supervisor			STUDENT NAME A PANCHAM		MECHATRONICS : AGV	1
	W'Shop Technician			E-MAIL 202513794@ukzn.ac.za		TITLE	
	Chief Technician			TEL NO. 083 639 9531 / 260 1227		PULLEY	

MATERIAL : PLASTIC



University of KwaZulu-Natal
School of Mechanical Engineering

	Date	Checked
Project Supervisor		
W'Shop Technician		
Chief Technician		

TITLE: MECHATRONICS : AGV

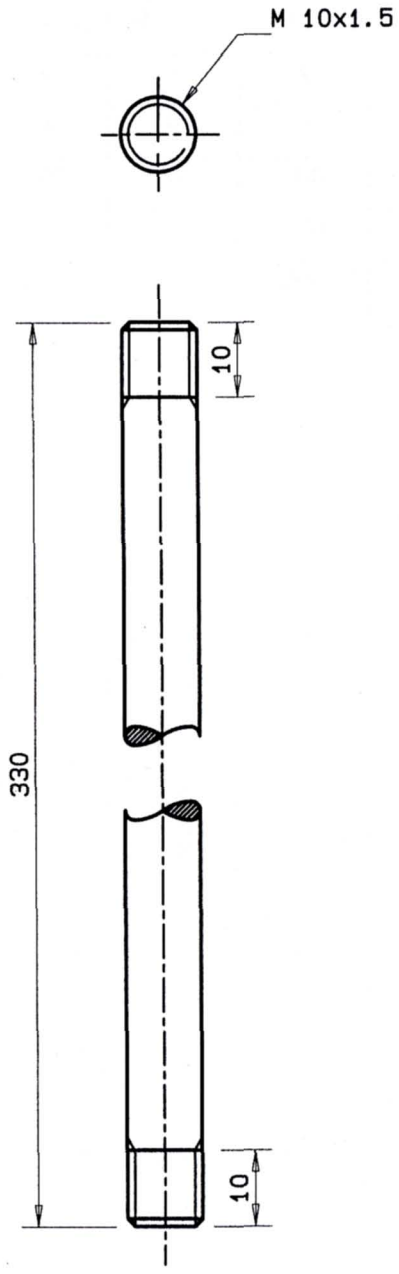
No. 8

SCALE 1 : 2

UNITS : mm

PROJECT: CONVEYOR ROLLER

STUDENT NAME A PANCHAM
TEL NO. 260 1227 E-MAIL 202513794



University of KwaZulu-Natal
School of Mechanical Engineering

	Date	Checked
Project Supervisor		
W'Shop Technician		
Chief Technician		

TITLE: ROD

No. 8

SCALE 1 : 1

UNITS : mm

PROJECT: MECHATRONICS : AGV

STUDENT NAME: A PANCHAM
TEL NO: 260 1227 E-MAIL: 202513794

B. Tables

Table B-1 X and y values of the line path (theoretical) and the actual paths followed between the conveyor and RM.

Line path		**Test 1: Red**		**Test 2: Green**		**Test 3: Blue**	
x	**y**	**x**	**y**	**x**	**y**	**x**	**Y**
0	0	0	0	0	0	0	0
100	0	100	0	100	0	100	0
170	0	170	0	170	0	170	0
175	0.2	237	1.9	206	1.8	242	3
180	0.4	240	2.3	206.5	0	257.5	2.5
185	0.8	242.5	2.3	208.8	0	258.5	5
190	1	243.5	1.7	210	1.9	259.5	6
195	1.2	244.5	2.7	208	2.2	278.5	7.8
200	1.3	246	5.3	236.2	2.4	291.2	8.7
205	1.8	266	6	253	1.7	292	10
210	2	271	6.4	254.5	4.5	292.5	11.1
215	2.3	272	6.2	255.5	6.4	303.5	11.6
220	2.65	275	6.2	273	7.2	310	10.7
225	3.1	276	6.8	288.5	7.7	310.5	10.7
230	3.55	277	6.8	289.5	9	312	13.9
235	3.95	278.5	6.4	290.5	11.7	321.5	15.8
240	4.5	279.5	8.1	299.7	11.3	312.5	15
245	5.1	280	10	301	9.5	331.5	16.2
250	5.7	286	11	306	9.5	333	18.5
255	6.4	288	9	307	9.1	331	16.3
260	7	288.5	8.4	309	13.3	334	21.5
265	7.8	290	8.6	310	14.6	349	23.6
270	8.55	292	8.7	326	17.2	365	25.4
275	9.3	294	8.6	338.9	17.8	365.5	25.5
280	10	294.5	8.3	340	20.5	366	25.8
285	10.9	295.2	9	341.5	22.7	367	34.2
290	11.7	296	10.8	351.4	24.8	379	38

295	12.55	296.5	12.8	370	28	380.2	46.2
300	13.45	301.5	13.7	371	37.5	380	47
305	14.2	302.7	12	372	38	381.5	47.5
310	15.4	304	10	382	41.6	392	51.7
315	16.3	304.5	9.5	383	41.6	392.5	56.6
320	17.5	306	12.6	383.5	42	405	64.5
325	18.7	306.5	13.7	383.5	48.7	404.5	69.2
330	20.1	312	13.8	384	49	403	63.8
335	21.2	315	14.1	396.5	54.4	404.2	71.4
340	22.4	316	14.3	395	60.1	405.5	72.5
345	23.9	317	14.5	394.5	62.5	407	71.5
350	26.2	318.1	14	396.5	63.5	419	79
355	28.8	318.5	14	397	62.6	418.5	80.4
360	31.8	319	14.2	404	65	418	81.6
365	35.5	320	14.4	403.5	68.2	417	83.8
370	39.5	322	14.6	403.5	69.5	425	90.4
375	44.5	323.5	14.7	411	75.3	430	94.3
380	49	324	14.4	420.5	81.3	435	98
385	53.4	324.5	14.3	420	83.7	438	100
390	58.8	325	14.5	419.5	86	436	107.7
395	63.5	326	17	422	90.2	438.5	111.5
400	67.7	326	18.5	424.5	90.6	436.5	109.5
405	72.8	338	20	428.5	89	437.5	110.7
410	77	340	20.4	427	93.5	441	113
415	82.5	342.5	20.8	439	102.5	445	112
420	87.5	343	21	437	108.3	443	117.5
425	95	345	21.1	436	110.5	442	119.5
430	102.5	348	21.5	436.5	111	446	118.6
435	109.3	350	21.7	438	110.5	451	123
440	116	351	21.5	440	110.4	449.5	127.5
445	125.5	352.5	25.3	445	114.8	455	133.8
450	134.8	353	27.5	443	119	460	139.1
455	143.8	354	26.1	443	119.5	457.5	144.5

460	154.5	355	25.3	448	125.3	455.5	147.7
465	167.5	355.5	25	450	127.5	459	147.2
470	180.1	356	25.2	456.5	135	460	147
475	193.5	358	25.5	455	139.5	464.5	152.3
480	207.3	360	25.8	463.5	150	462.5	156.3
482	213.4	361	26	461	154.5	462	156.8
482.6	215.4	362	26	462	157	464.5	160
		363	26.2	461.5	153.8	473.5	172.6
		365	26.6	473	171.3	471	177
		367	27	470	176.1	476.5	186
		368.5	27	468	177.7	480	192.2
		369	29	469.5	179.3	480.5	193
		370	35.6	471.5	179.2	479	195
		375	37	479	190.7	479	191
		381.5	39.2	477	194	480	193.4
		382	48.2	475.5	195	478	196.5
		390	53	481	202.5	477.5	197
		395	55.5	486.5	214	482	202.5
		398.5	57			485.5	212
		397.5	63.3			486.5	214
		397.5	65.4			485	211.5
		398	56.5				
		398	66				
		400	65				
		401	65				
		406	72.5				
		406.5	67.7				
		406.5	71				
		416	79.5				
		426	87				
		424.5	91				
		429	95				
		429.5	95.3				

		429.5	97.5				
		437.5	109.5				
		439	105.5				
		451	123				
		448	128.5				
		448	130.6				
		451	130.7				
		451.5	131				
		454	139.5				
		454.5	138.7				
		455.5	135.3				
		456	142				
		463.5	158.5				
		466	154.2				
		467.5	173				
		470	169				
		468.5	172				
		473.5	183.7				
		477	198.5				
		478	196.7				
		479.5	194.5				
		479.5	205.3				
		481	205.3				
		482.6	205.7				
		484.5	209.2				
		486	210.3				
		485	211.6				
		484.4	213				
		483	214				

Table B-2 X and y values of the actual paths at intersections and parallel sections.

Test 1: Red		Test 2: Green		Test 3: Blue		Average	
x	**y**	**x**	**y**	**x**	**y**	**x**	**Y**
200	1.9	200	2.1	200	-0.4	200	1.2
215.5	2	215.5	2	215.5	2	215.5	2
242	2.2	242	2.2	242	3	242	2.5
248	5.5	248	2.2	248	3	248	3.6
258	6	258	6.6	258	4	258	5.5
279	7.4	279	7.4	279	8	279	7.6
289	8.5	289	8.7	289	8.8	289	8.7
304.5	9.5	304.5	9.5	304.5	11.3	304.5	10.1
305	11.1	305	9.5	305	11.1	305	10.6
308	13.7	308	11	308	11	308	11.9
312	13.8	312	15	312	13.8	312	14.2
325	15.8	325	17	325	15.8	325	16.2
332.5	19.5	332.5	17.6	332.5	17.6	332.5	18.2
351.5	24	351.5	25	351.5	24	351.5	24.3
354.5	25.4	354.5	25.4	354.5	24.5	354.5	25.1
355	25.2	355	25.5	355	24.6	355	25.1
360	25.8	360	26.4	360	25	360	25.7
365	26.6	365	27.4	365	25.4	365	26.5
366	26.8	366	27.5	366	26.8	366	27
366.5	26.8	366.5	27.6	366.5	27.6	366.5	27.3
370	35.8	370	30	370	35.3	370	33.7
369.5	35.2	369.5	28	369.5	35.2	369.5	32.8
371	36	371	35.7	371	35.5	371	35.7
380	38.8	380	41	380	47	380	42.3
382	41.6	382	41.6	382	47.6	382	43.6
382.2	47.8	382.2	41.7	382.2	47.8	382.2	45.8
390	52.5	390	51.3	390	51.3	390	51.7
392	53.4	392	52.9	392	53.2	392	53.2
394	53.9	394	53.9	394	57.5	394	55.1
395	54.4	395	54.2	395	58.2	395	55.6

396.5	55.3	396.5	55.3	396.5	58.9	396.5	56.5
400	65	400	63.5	400	62	400	63.5
404.5	67.3	404.5	70.6	404.5	67.3	404.5	68.4
420	82.7	420	82.7	420	85.8	420	83.7
425	90.3	425	90.3	425	90.3	425	90.3
429	94.7	429	94.7	429	93.2	429	94.2
429.5	95.3	429.5	95.3	429.5	93.8	429.5	94.8
435	102.2	435	96	435	98	435	98.7
437.5	104	437.5	101.5	437.5	104	437.5	103.2
430	97.8	430	96	430	94	430	95.9
438	104.3	438	101.6	438	101.6	438	102.5
438.5	110.4	438.5	110.4	438.5	112	438.5	110.9
440	112	440	110.7	440	113	440	111.9
440.5	113	440.5	111.3	440.5	113	440.5	112.4
442	114.2	442	112.5	442	112.5	442	113.1
443	116	443	113.7	443	116	443	115.2
443.5	116.5	443.5	114.2	443.5	114.2	443.5	115
444	117	444	117	444	118.8	444	117.6
450	127.2	450	127.2	450	128.2	450	127.5
449.5	127.7	449.5	126.8	449.5	126.8	449.5	127.1
450	122.3	450	127.7	450	122.4	450	124.1
455	135.2	455	133.5	455	134.7	455	134.5
455.5	143.3	455.5	135.5	455.5	135.5	455.5	138.1
460	148.4	460	146.5	460	147.2	460	147.4
462.5	151	462.5	151	462.5	151	462.5	151
463	158.5	463	158.5	463	158.5	463	158.5
464.5	161.3	464.5	160.5	464.5	160.5	464.5	160.8
465	162.2	465	161.3	465	161	465	161.5
468	167.3	468	166	468	166	468	166.4
470	169	470	168	470	168	470	168.3
475	186.2	475	184	475	184	475	184.7
479	193.8	479	190.7	479	190.7	479	191.7
481.5	205.5	481.5	206	481.5	205.5	481.5	205.7

485	211.5	485	211.7	485	211.4	485	211.5
485.5	211	485.5	212	485.5	212	485.5	211.7

Table B-3 X and y values of the light paths followed from the lathe to RM.

Test 1: Orange		**Test 2: Blue**		**Test 3: Green**	
x	**y**	**x**	**y**	**x**	**y**
0	0	0	0	0	0
10	30	10	14.5	10	13.5
20	48	13.7	20	20	23.5
30	58.5	13.5	31	21.5	25.7
40	66.5	20	39	21	45.2
45	70	29	50	30	65.5
45.2	103	30	62	40	78.5
50	104.5	38.5	70	41	84
60	106	37	92	41	87.5
70	108	50	96	50	86.5
80	110	60	98	55	106.5
90	111.5	70	100.5	59	85
100	113	80	102	60	108.5
110	115	90	104	70	112
120	116.5	100	106	80	115
130	118.5	110	107.5	90	118
140	120.5	120	109.5	100	121
150	122	130	112	110	124
160	123.5	140	113.5	120	127
170	125	150	115.5	130	130.5
180	126.5	160	117	140	133.5
190	128.5	170	119.5	150	136
200	130	180	121	160	139
208	131	190	123	170	141
		200	124.5	180	144
		210	126.5	190	147
		220	128	200	149.5

		230	130	206.5	151
		240	131.5		
		244.5	132		

Table B-4 Average x and y values of the light paths followed from the lathe to RM. (The average was calculated from table B-2.

Average	
x	**y**
0	0
10	19.3
20	36.8
30	62
40	79
50	95.7
60	104.2
70	106.8
80	109
90	111.2
100	113.3
110	115.5
120	117.7
130	120.3
140	122.5
150	124.5
160	126.5
170	128.5
180	130.5
190	132.8
200	134.7

Table B-5 Comparison of light sensor characteristics [10]

Electrical Characteristics	**Light Sensor**		
	Photo-diodes	**Photo-transistors**	**Light dependent resistors**
Available Wavelengths (µm)	0.2-2.0	0.4-1.1	0.4-0.7
Performance-to-cost ratio	Good	Excellent	Excellent
Sensitivity	Very Good	Very Good	Very Good
Linearity	Excellent	Good	Good
Ambient Noise Performance	Very Good	Very Good	Very Good
Dynamic Range	Excellent	Very Good	Good
Stability	Very Good	Good	Poor
Reproducibility	Excellent	Fair	Poor
Cost	Low	Very Low	Very Low
Ruggedness	Excellent	Excellent	Excellent
Physical Size	Small	Small	Small
Ease of Customization	Easy	Fair	Fair
Cost of Customization	Low	Medium	Low
Lead time for Customization (weeks)	12	14	12

C. Figures

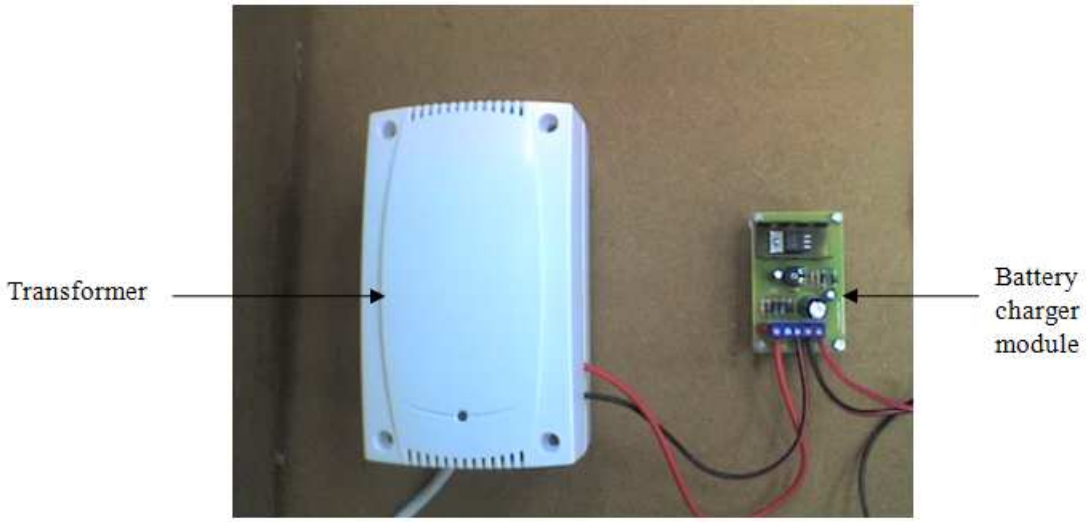


Figure C-1 Transformer and battery charger module.

Figure C-2 The Cybot hobby robot uses light following [49]

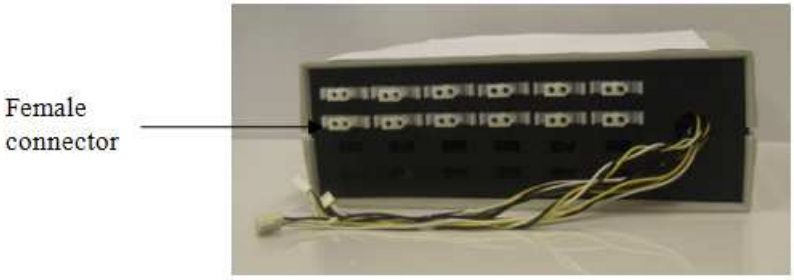


Figure C-3 Sensor board box with cover and female connectors.

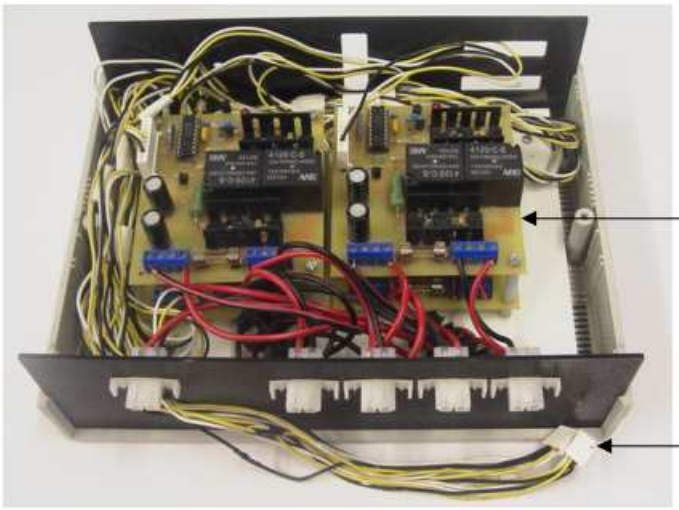


Figure C-4 Motor board box

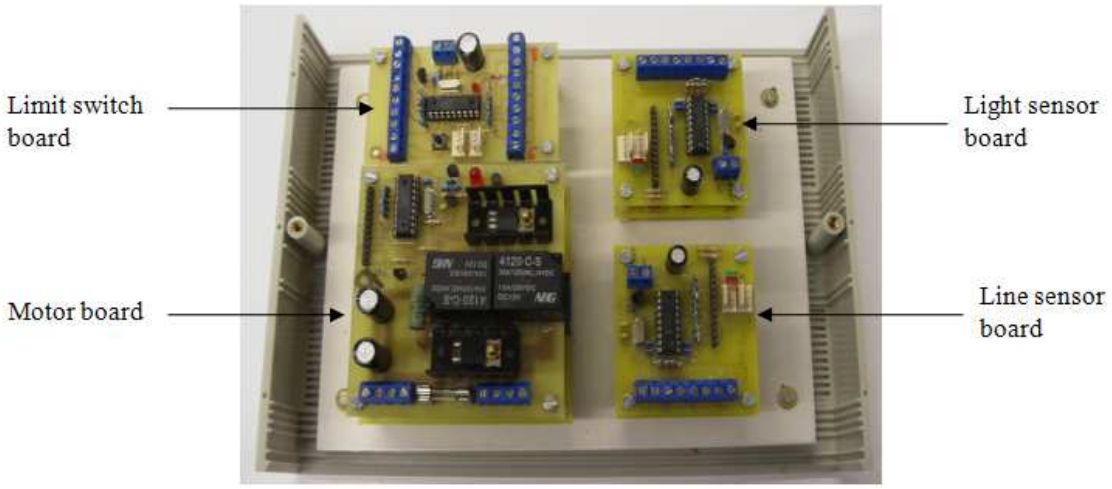


Figure C-5 Sensor board box. Clockwise, from top left to bottom left, limit switch board, light sensor boards, line light sensor boards and motor boards.

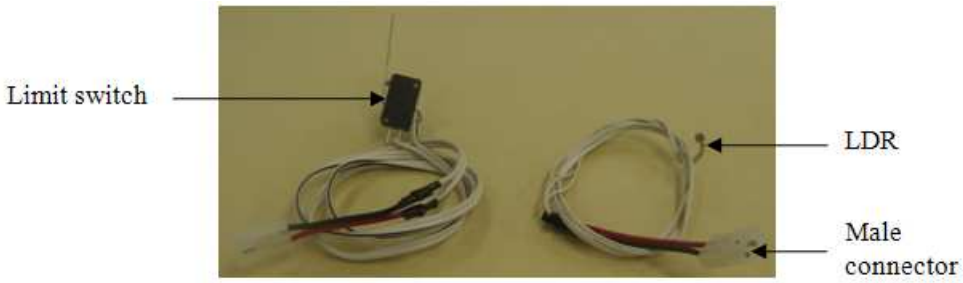


Figure C-6 Light sensor and limit switch with male connectors.

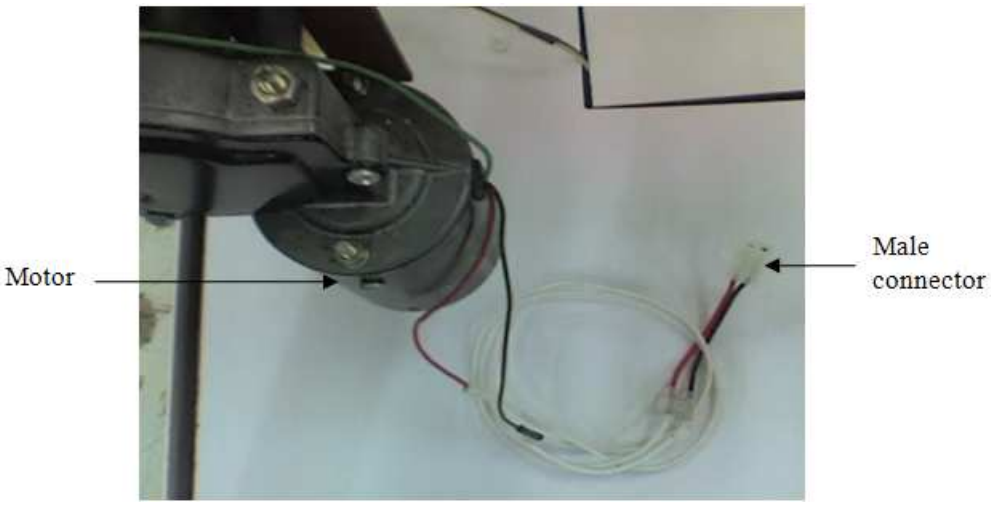


Figure C-7 Motor with male connector.

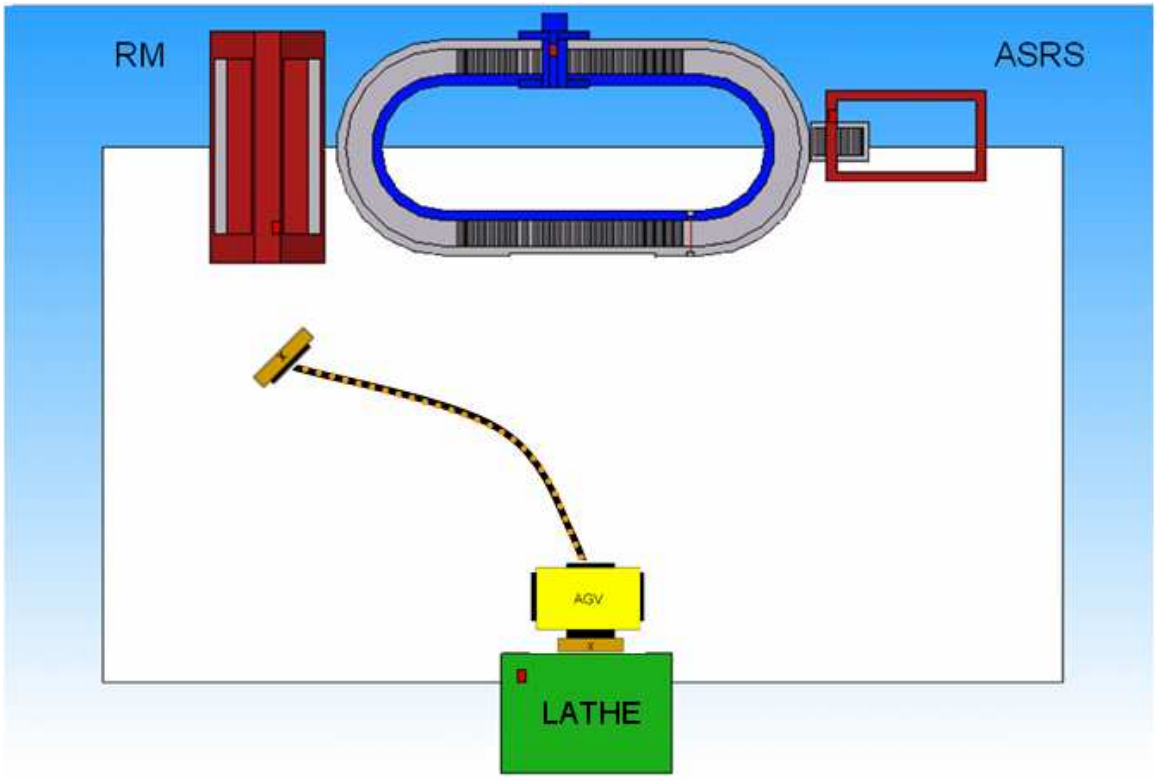


Figure C-8 Line laid along average light path between RM and lathe

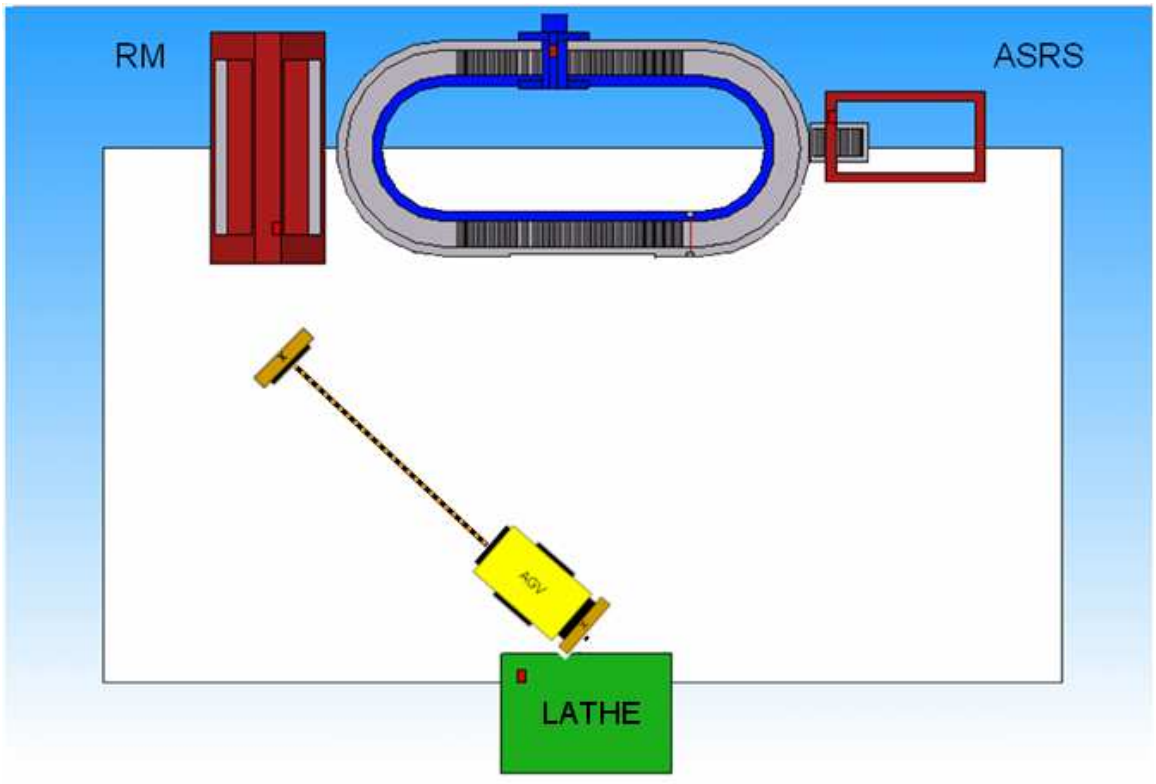
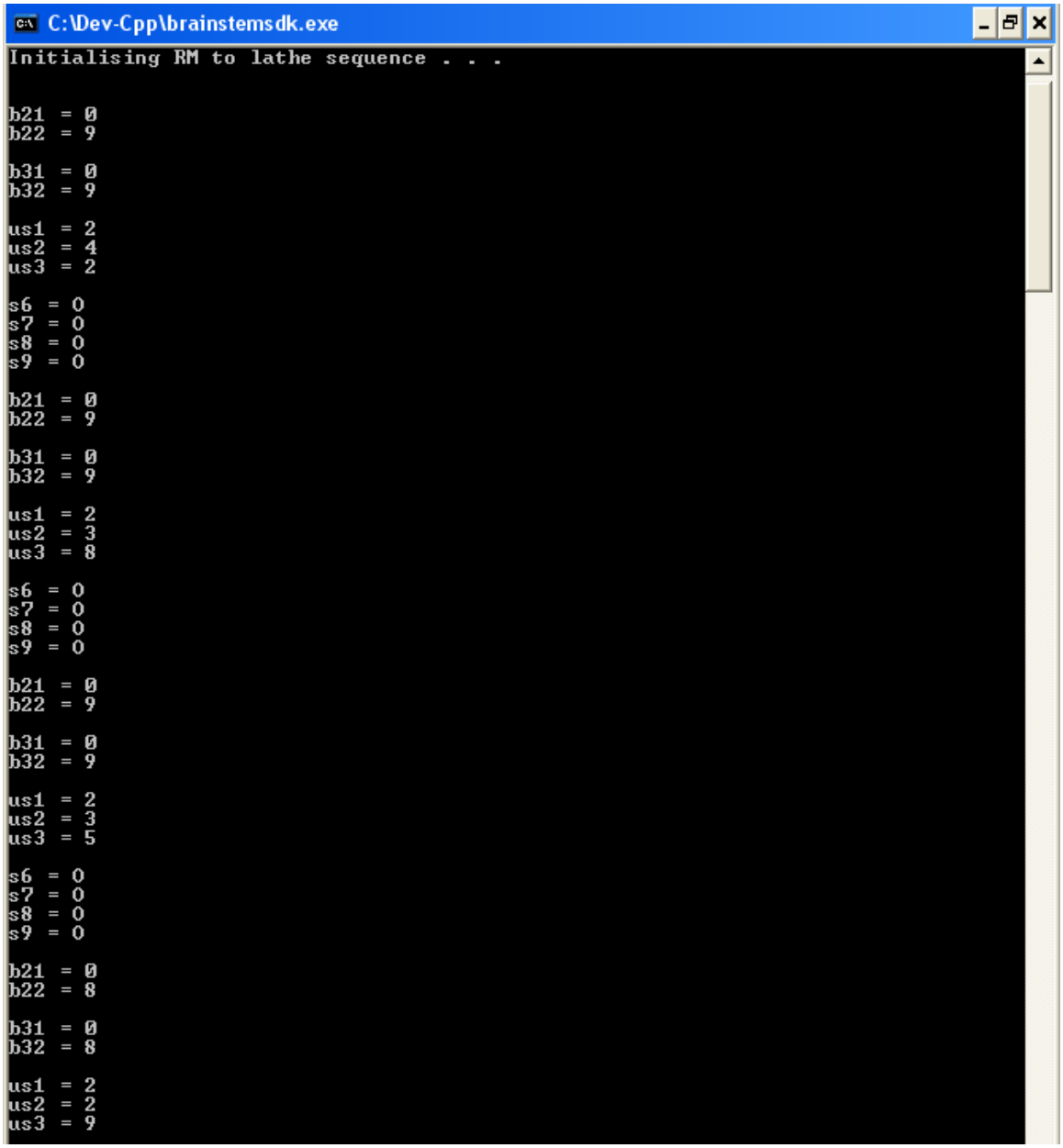


Figure C-9 Diagonal line path


C:\Dev-Cpp\brainstemsdk.exe
Initialising RM to lathe sequence . . .
b21 = 0
b22 = 9
b31 = 0
b32 = 9
us1 = 2
us2 = 4
us3 = 2
s6 = 0
s7 = 0
s8 = 0
s9 = 0
b21 = 0
b22 = 9
b31 = 0
b32 = 9
us1 = 2
us2 = 3
us3 = 8
s6 = 0
s7 = 0
s8 = 0
s9 = 0
b21 = 0
b22 = 9
b31 = 0
b32 = 9
us1 = 2
us2 = 3
us3 = 5
s6 = 0
s7 = 0
s8 = 0
s9 = 0
b21 = 0
b22 = 8
b31 = 0
b32 = 8
us1 = 2
us2 = 2
us3 = 9

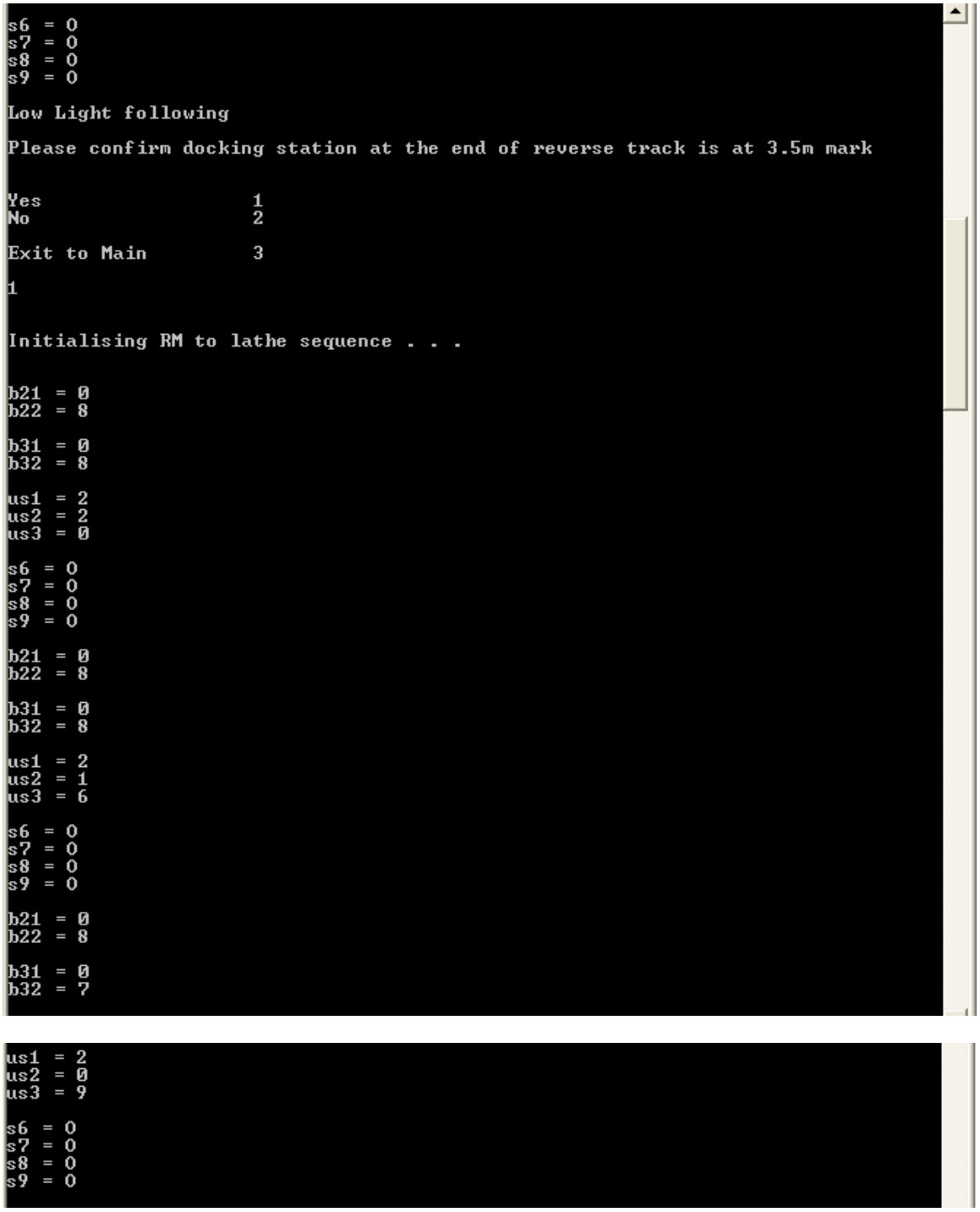


Figure C-10 Interface illustrating transition from high to low light following. b21, b22, b31, b32 refer to the reverse light sensors. us1, us2, us3 refer to the reverse ultrasonic sensor reading. s6, s7, s8, s9 refer to the limit switch status i.e. open (O) or closed (C). Sensor feedback continued until the relevant limit switch was compressed on docking or in the event of a collision.

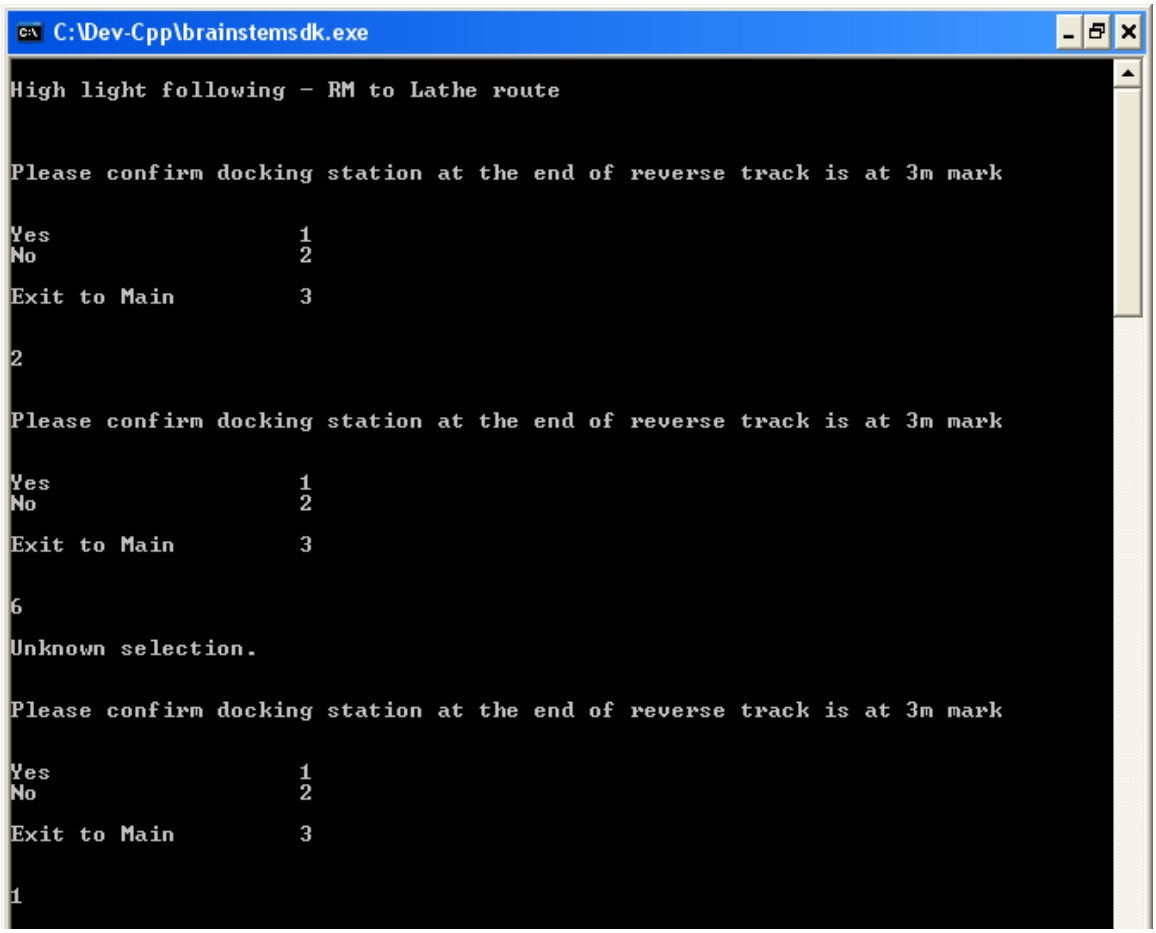


Figure C-11 User prompted again to reposition docking station.

D. Electronic Hardware Features

PCI solution from Adlink

PCI-8158**:** Advanced 8-axis Stepping & Servo Motion Control Card

Figure D-1 PCI-8158 control card

Features

1. “Pulse output options: OUT/DIR, CW/CCW
2. Pulse output rate up to 6.55Mpps
3. Linear/circular interpolation with continuous contouring
4. Hardware emergency input
5. Security protection for users' program
6. Position/speed change on-the-fly
7. 13 home return modes and auto home search
8. Support manual pulse generator.” [51]

PXI solution from National Instruments

NI PXI-7356

Figure D-2 NI PXI-7356 controller. [54]

Features

1. “6 axes of stepper or servo motor control
2. High-speed 4 MHz position triggering
3. 64 lines of digital I/O
4. Built-in data acquisition with eight 16-bit analog I/O lines”

[52, 54]

DAQ solution from Eagle Technology

MicroDAQ BT-120A

Figure D-3 BT-120A wireless DAQ [55]

Features

1. “Connects via Wireless connection
2. 120 DIO lines (model dependent)
3. TTL level I/O
4. I/O Connector: 1x DB25 (-24A); 3x DB25(-72A); 5x DB25(-120A) Male
5. LED indication for power connection
6. Ideal for portable/laptop applications
7. Housing: Plastic ABS with rubber feet
8. Operating Temp: 0° to 70°C
9. Power: 100mA (BT-24A); 260mA (BT-72A); 360mA (BT-120A) max
10. O/S Support for Windows 98/ME/XP/2000 & WinCE (PocketPC2002 & PocketPC2003)
11. Includes EDRE SDK, EDRE-Labview, EDRE-Testpoint and WaveView for Windows”

[55]

Brainstem Moto 1.0

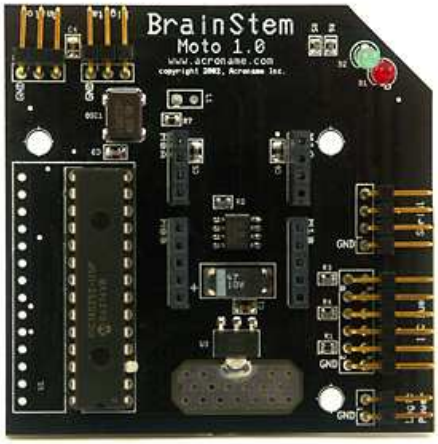


Figure D-4 Brainstem Moto 1.0

Features

1. “40 MHz RISC processor
2. 2 motion control channels with PWM frequencies from 2.5khz - 5mhz
3. 1 dedicated 10-bit A/D
4. 1 dedicated digital I/O lines
5. 1 MBit IIC port
6. IIC routing
7. Status LED
8. 11 1k TEA file slots and 1 16k TEA file slot
9. 368 bytes of RAM available to user
10. RS-232 serial port
11. Reflex architecture
12. Ability to run 3 concurrent TEA processes
13. Hardware EEPROM lock option
14. Small size (2.5" square, 0.5" high)
15. Supported on Windows, WinCE, MacOS X, Linux (access libraries)” [58]

Telemetry System

Features

1. 32 addressable modules.
2. Wireless communication up to 100m line of sight.
3. USB-CPU unit and a robot CPU (data acquisition) unit.
4. Slave devices: 8 motor boards, 2 line sensor boards, 2 light sensor boards, 1 limit switch board and 4 ultrasonic sensor boards.
5. “LED indication for power connection on unit.” [55]
6. LED indication for data transfer.
7. Data distribution to modules via I2C bus.
8. LCD diagnostic display for data transfer.
9. Data transfer at 16 bytes/second.
10. “Ideal for portable/laptop applications.” [55]

E.1 AGV Programming
1.1 Access routines.c

```
#define aWIN
#include "Access.h"

int initialise(serial_struct* ssp, const char* port, int baudrate)
{
                    aErr err = aErrNone;
                    int i;

                    aIO_GetLibRef(&ssp->ioaccess, &err);

                    if (err != aErrNone)
                    puts("Error loading IOlibrary");

                    aStreamBuffer_Create(ssp->ioaccess, 1024, &ssp->bufferstream, &err  );

                    ssp->serial_port = port;

                    ssp->baud = baudrate;


  for (i = 0; i < sizeof(ssp->inputbuffer); i++)
  {
     ssp->inputbuffer[i] = 0;

  }
  ssp->pinputbuff = (char*)malloc(sizeof(ssp->inputbuffer));
  if(err == aErrNone)
  puts("Data Structure Initialised... :)");

  return err;

}

int buildstream(serial_struct* ssp)
{
                    aErr err = aErrNone;

                    aStream_CreateSerial(ssp->ioaccess, ssp->serial_port,
                    ssp->baud, &ssp->serial_stream, &err);

                    if(err)
                    {
                     puts("initialisation of serial stream failed");
                    }

                    return err;
}
```

```
int Read(serial_struct* ssp, const unsigned long numberofbytes)
{
		aErr err = aErrNone;

		aStream_Read(ssp->ioaccess, ssp->serial_stream,
		ssp->pinputbuff, numberofbytes, &err);

		int i;
		for (i = 0; i < numberofbytes; i++);
		{
		   ssp->inputbuffer[i] = ssp->pinputbuff[i];
		}

		if(err)
		{
		      puts("Read operation failed");
		}
		return err;
}

int Write(serial_struct* ssp, const unsigned long numberofbytes, const char* data)
{
		 aErr err = aErrNone;

		 ssp->outputbuffer = data;

		 aStream_Write(ssp->ioaccess, ssp->serial_stream,
		 ssp->outputbuffer, numberofbytes, &err);

		// aStreamBuffer_Create(ssp->ioaccess, 1024,
		//&ssp->serial_stream, &err);

		 if(err)
		 {
		       puts("Write operation failed");
		 }
		return err;
}

int cleanupserial(serial_struct* ssp)
{
			aErr err = aErrNone;
			// kill the stream
			aStream_Destroy(ssp->ioaccess, ssp->serial_stream,
			 &err);
			// release the library
			aIO_ReleaseLibRef(ssp->ioaccess, &err);
			if(err)
			puts("IOLibrary not released!  Memory Leak!!!");
			free((void*)ssp->pinputbuff);
			return err;
```

```
}
int Flush(serial_struct* ssp)
{
        aErr err = aErrNone;
        aStream_Flush(ssp->ioaccess, ssp->serial_stream, ssp->bufferstream, &err);
        return err;
}
```

E 1.2 Interfaces

AGV Program.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define aWIN

#include "Access.h"
#include "Motor.h"
#include "Directions.h"

#include "Ultrasonics.h"
#include "LimitSwitch.h"

#include "Line.h"
#include "LineSide.h"
#include "LineFusion.h"
#include "LineInterface.h"
#include "LineRoutes.h"
#include "LineDock.h"
#include "LineSide.h"

#include "Light.h"
#include "LightFusion.h"
#include "LineInterface.h"
#include "LightRoutes.h"
#include "LightDock.h"
#include "LightTest.h"

#include "RobotMotor.h"
#include "RobotLightTest.h"
#include "RobotLineTest.h"

#include "LineOwnLight.h"
#include "LightOwnLine.h"

#include "LineOwnLightSim.h"
#include "LightOwnLineSim.h"

#include "InterLightLine.h"
#include "InterLineLight.h"
```

```
#include "LightLineSimRMLatheReverseFusion.h"
#include "LightLineSimRMLatheSideleftFusion.h"
#include "LightOwnLineSimRMLatheSideleftFusion.h"

int Stop(serial_struct* ssp);

int sw = 1;

int main(int argc, char *argv[])

{

   aErr err = aErrNone;

   serial_struct Serial;
   initialise(&Serial, "COM4", 19200);

   buildstream(&Serial);

    MainInterface(&Serial);

 //MainInterfaceConstant(&Serial);
 //RobotMainInterface(&Serial);

 cleanupserial(&Serial);

 system("PAUSE");

 return 0;
}
```

Main Interface.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"

#include "MainInterface.h"
#include "MainInterfaceEnvironment.h"

int MainInterface(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone;
   int e ;
```

```
    printf("\nVariable Sensor System "  ) ;
    printf("\n----------------------\n") ;

    printf("\nPlease select an Environment: \n\n"  ) ;

    printf("Unsatisfactory Floor \t\t 1  \nLight Intensity \t\t 2 \nHigh Noise Level \t\t 3 \nConstant
Environment \t\t 4 \n\n") ;
    scanf("%d" , &e) ;

    printf("\n") ;

    if (e == 1)
    UnsatisfactoryFloor(ssp) ;

    else if (e == 2)
    LightIntensity(ssp) ;

    else if (e == 3)
    HighNoiseLevelEnvironment(ssp) ;

    else if (e == 4)
    ConstantEnvironment(ssp) ;

    else printf("\nUnknown Environment. Please enter new Environment\n") ;

    return err;

}
```

Main Interface Environment.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "MainInterfaceEnvironment.h"
#include "LineVariationInterface.h"
#include "LightVariationInterface.h"
#include "LightInterface.h"
#include "LineInterface.h"

int UnsatisfactoryFloor(serial_struct* ssp)

{
    //only load line var routes here
    aErr err = aErrNone;
    int t;

    printf("Unsatisfactory Floor");
    printf("\n--------------------\n") ;
```

```
   printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

   printf("Line Variation \t\t\t 1 \nLight Following \t\t 2 \nExit to Main \t\t\t 3 \n\n") ;
   scanf("%d", &t);

   if (t == 1)
   LineVariationInterface(ssp) ;

   else if (t == 2)
   LightInterface(ssp) ;

   else if (t == 3)
   MainInterface(ssp) ;

   else printf("\nUnknown technique. Please enter new technique\n") ;

  return err;

}

int LightIntensity(serial_struct* ssp)

{
   aErr err = aErrNone;
   int t;

    printf("Light Intensity");
   printf("\n---------------\n") ;

   printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

   printf("Light Variation \t\t 1 \nLine Following \t\t\t 2 \nExit to Main \t\t\t 3 \n\n") ;
   scanf("%d", &t);

   if (t == 1)
   LightVariationInterface(ssp) ;

   else if (t == 2)
   LineInterface(ssp) ;

   else if (t == 3)
   MainInterface(ssp) ;

   else printf("\nUnknown technique. Please enter new technique\n") ;

   return err;
}

int HighNoiseLevelEnvironment(serial_struct* ssp)
```

```
{
   aErr err = aErrNone;
   int t;

   printf("\nHigh Noise Level Environment") ;
   printf("\n---------------------------\n") ;

   printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

   printf("Line Variation \t\t\t 1  \nLight Variation \t\t 2 \nLine Following \t\t\t 3  \nLight Following
\t\t 4 \nExit to Main \t\t\t 5 \n\n") ;
   scanf("%d", &t);

   if (t == 1)
   LineVariationInterface(ssp) ;

   if (t == 2)
   LightVariationInterface(ssp) ;

   else if (t == 3)
   LineInterface(ssp) ;

   else if (t == 4)
   LightInterface(ssp) ;

   else if (t == 5)
   MainInterface(ssp) ;

   else printf("\nUnknown technique. Please enter new technique\n") ;

   return err;
}

int ConstantEnvironment(serial_struct* ssp)

{
   aErr err = aErrNone;

   int t;

   printf("\nConstant Environment"  ) ;
   printf("\n--------------------\n") ;

   printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

   printf("Line Variation \t\t\t 1  \nLight Variation \t\t 2 \nLine Following \t\t\t 3  \nLight Following
\t\t 4 \nExit to Main \t\t\t 5 \n\n") ;
   scanf("%d", &t);
   if (t == 1)
   LineVariationInterface(ssp) ;
```

```
   if (t == 2)
   LightVariationInterface(ssp) ;

   else if (t == 3)
   LineInterface(ssp) ;

   else if (t == 4)
   LightInterface(ssp) ;

   else if (t == 5)
   MainInterface(ssp) ;

   else printf("\nUnknown technique. Please enter new technique\n") ;

   return err;

}
```

Main Interface Constant.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"

#include "MainInterfaceConstant.h"
#include "LightInterface.h"
#include "LineInterface.h"

int MainInterfaceConstant(serial_struct* ssp)


{
   aErr err = aErrNone;
   int t;

   printf("\nConstant Environment"  ) ;
   printf("\n--------------------\n") ;

   printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

   printf("Line Following \t\t\t 1  \nLight Following \t\t 2 \n\n") ;
   scanf("%d", t);

   if (t == 1)
   LineInterface(ssp) ;

   else if (t == 2)
   LightInterface(ssp) ;
```

```
    else printf("\nUnknown technique. Please enter new technique.\n") ;

    return err;

}
```

Motor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"

int Chill(serial_struct *ssp)

{
    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT1SPF0000 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT2SPF0000 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT3SPF0000 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPF0000 *\\");

    sleep(400);

    return err;

}

int Forward(serial_struct* ssp)

{


    Chill(ssp) ;

     /*
    1 2       front      order motors come on
```

```
    3 4       back
    */

    aErr err = aErrNone;
    Write(ssp, 17, "#%+MOT1SPF0500 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT2SPF0500 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT3SPF0500 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPF0500 *\\");

    sleep(400);

    return err;

}

int Reverse(serial_struct* ssp)

{

    /*
    3 4
    1 2
    */

    Chill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT3SPR0500 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPR0500 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT1SPR0500 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT2SPR0500 *\\");
```

```
    sleep(400);

    return err;

}

int Left(serial_struct* ssp)

{

    /*
    2 1
    3 4

    1 & 4 reverse 2 & 3 forward
    */

    Chill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT2SPF0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT1SPR0400 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT3SPF0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPR0400 *\\");

    sleep(400);

    return err;

}

 int Right(serial_struct* ssp)

{

      /*
    1 2
    4 3

    2 & 3 reverse 1 & 4 forward
    */
```

```
    Chill(ssp) ;
    aErr err = aErrNone;
    Write(ssp, 17, "#%+MOT1SPF0600 *\\");
    sleep(400);
    Write(ssp, 17, "#%+MOT2SPR0400 *\\");
    sleep(400);
    Write(ssp, 17, "#%+MOT4SPF0600 *\\");
    sleep(400);
    Write(ssp, 17, "#%+MOT3SPR0400 *\\");
    sleep(400);
    return err;
}
int Sideleft(serial_struct* ssp)
{
      /*
     1 4
     2 3
     1 & 4 in 2 & 3 out
     */
    Chill(ssp) ;
    aErr err = aErrNone;
    Write(ssp, 17, "#%+MOT1SPR0500 *\\");
    sleep(400);
    Write(ssp, 17, "#%+MOT4SPF0500 *\\");
    sleep(400);
    Write(ssp, 17, "#%+MOT2SPF0500 *\\");
```

```
    sleep(400);

    Write(ssp, 17, "#%+MOT3SPR0500 *\\");

    sleep(400);

    return err;

}

int Sideleft225(serial_struct* ssp)

{

      /*
     1 4
     2 3

     1 & 4 in 2 & 3 out
     */

    Chill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT1SPR0400 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPF0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT2SPF0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT3SPR0400 *\\");

    sleep(400);

    return err;

}

int Sideleft135(serial_struct* ssp)

{

      /*
```

```
    1 4
    2 3

    1 & 4 in 2 & 3 out
    */

   Chill(ssp) ;

   aErr err = aErrNone;

   Write(ssp, 17, "#%+MOT1SPR0600 *\\");

   sleep(400);

   Write(ssp, 17, "#%+MOT4SPF0400 *\\");

   sleep(400);

   Write(ssp, 17, "#%+MOT2SPF0400 *\\");

   sleep(400);

   Write(ssp, 17, "#%+MOT3SPR0600 *\\");

   sleep(400);

   return err;

}

int Sideright(serial_struct* ssp)

{
     /*
    2 3
    1 4

    1 & 4 out 2 & 3 in
    */

    Chill(ssp) ;

   aErr err = aErrNone;

   Write(ssp, 17, "#%+MOT2SPR0500 *\\");

   sleep(400);

   Write(ssp, 17, "#%+MOT3SPF0500 *\\");

   sleep(400);
```

```
    Write(ssp, 17, "#%+MOT1SPF0500 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPR0500 *\\");

    sleep(400);

    return err;

}

int Sideright45(serial_struct* ssp)

{
      /*
     2 3
     1 4

     1 & 4 out 2 & 3 in
     */

    Chill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT2SPR0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT3SPF0400 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT1SPF0400 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPR0600 *\\");

    sleep(400);

    return err;

}

int Sideright315(serial_struct* ssp)

{
      /*
```

```
    2 3
    1 4

    1 & 4 out 2 & 3 in
    */

    Chill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT2SPR0400 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT3SPF0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT1SPF0400 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPR0600 *\\");

    sleep(400);

    return err;

}

int Reverseleft(serial_struct* ssp)

{

    /*
    4 3
    1 2

    1 & 4 reverse 2 & 3 forward
    */

    Chill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT4SPR0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT3SPF0400 *\\");
```

```
    sleep(400);

    Write(ssp, 17, "#%+MOT1SPR0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT2SPF0400 *\\");

    sleep(400);

    return err;

}

int Reverseright(serial_struct* ssp)

{

    /*
    3 4
    2 1

    2 & 3 reverse 1 & 4 forward
    */

    Chill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT3SPR0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT4SPF0400 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT2SPR0600 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT1SPF0400 *\\");

    sleep(400);

    return err;

}

//changed 2-5 3-6 reverse & sideleft
```

```
/*Reverse functions

2       3       u r here facing reverse      left hand       reverse left
1       4                                    right hand      reverse right

*/
```

1.3 Line Following Technique

Line Interface.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"

#include "LineInterface.h"
#include "LineRoutes.h"
#include "LineFusion.h"
#include "MainInterface.h"

//#include "Directions.h"

int LineInterface(serial_struct* ssp)

{
   Flush(ssp) ;

   aErr err = aErrNone;
   extern int sw ;
   int route ;

   printf("\nPlease select a Line route: \n\n"  ) ;

   printf("Conveyor to Lathe \t 1 \nConveyor to RM \t\t 2 \n\nLathe to Conveyor \t 3 \nLathe to RM
\t\t 4 \n \nRM to Lathe \t\t 5 \nRM to Conveyor \t\t 6 \nExit to Main \t\t 7 \n\n ") ;
   scanf("%d", &route) ;

   if (route == 1)
   Conveyor_Lathe(ssp) ;

   else if (route == 2)
   Conveyor_RM(ssp) ;

   else if (route == 3)
   Lathe_Conveyor(ssp) ;

   else if (route == 4)
   Lathe_RM(ssp) ;
```

```
   else if (route == 5)
   RM_Lathe(ssp) ;

   else if (route == 6)
   RM_Conveyor(ssp) ;

   else if (route == 7)
   MainInterface(ssp) ;

   else printf("\nUnknown route. Please enter new route\n") ;

return err;

}
```

Line Routes.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineRoutes.h"
#include "LineFusion.h"

//Conveyor

int Conveyor_Lathe(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("\nInitialising conveyor to lathe sequence \n\n\n") ;

   while(sw == 1)
   SideLeftLine(ssp) ;

   return err;
}


int Conveyor_RM(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("Starting Conveyor_RM sequence \n\n\n") ;
```

```
   while(sw == 1)
   SideLeftForwardLine(ssp) ;

   return err;
}


//Lathe

int Lathe_Conveyor(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("Starting Lathe_Conveyor sequence \n\n\n") ;

   while(sw == 1)
   SideRightLine(ssp) ;

   return err;
}

int Lathe_RM(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("Starting Lathe_RM sequence \n\n\n") ;

   while(sw == 1)
   SideRightForwardLine(ssp) ;

   return err;
}

//RM

int RM_Lathe(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("Starting RM_Lathe sequence \n\n\n") ;

   while(sw == 1)
   ReverseSideLeftLine(ssp) ;
```

```
   return err;
}

int RM_Conveyor(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("Starting RM_Conveyor sequence \n\n\n") ;

   while(sw == 1)
   ReverseSideRightLine(ssp) ;
   return err;
}
```

Line Dock.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "RobotMotor.h"
#include "LineDock.h"
#include "LineTestInterface.h"

//return to main route

int LatheChill(serial_struct* ssp)

/*
Docked at Lathe
Prompts user for route by Interface
*/

{
   aErr err = aErrNone;
   Chill(ssp);
   printf("At Lathe. Begin manual transfer. \n") ;

   MainInterface(ssp) ;

   return err;
}

int RMChill(serial_struct* ssp)

/*
Docked at RM
Prompts user for route by Interface
```

```
*/

{
   aErr err = aErrNone;
   Chill(ssp);

   printf("At RM. Begin manual transfer \n") ;

   MainInterface(ssp) ;

   return err;
}


int EmergencyStop(serial_struct* ssp)

{
   aErr err = aErrNone;

   printf("\n") ;
   printf("Crashed!! User intervention required. \n") ;

   Chill(ssp);

   MainInterface(ssp) ;

   return err;
}

int Charging(serial_struct* ssp)

{
   aErr err = aErrNone;

   Chill(ssp);

   printf("\nCharging \n") ;

   MainInterface(ssp) ;

   return err;
}

int TableSequence(serial_struct* ssp)

{
  aErr err = aErrNone;

   Chill(ssp) ;

   int w = 1 ;
```

```
    printf("\n") ;
    printf("Transfer sequence activated \n") ;
    printf("Loading pallet . . . \n") ;

    while(w==1)
{

    Flush(ssp) ;
    Write(ssp, 17, "#%+MOT5SPF0000 *\\");
    sleep(400) ;
    Write(ssp, 17, "#%+MOT5SPF0500 *\\");
    sleep(400) ;

 //Check if s2 3 4 5 compressed

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    int n ;
    for (n = 0; n < 19 ; n++)
    {
    printf("%d \t %c\n", n , ssp->pinputbuff[n]);
    }

    char s2 = ssp->pinputbuff[11] ;
    char s3 = ssp->pinputbuff[12] ;
    char s4 = ssp->pinputbuff[13] ;
    char s5 = ssp->pinputbuff[14] ;

    printf("s2 = %c \n" , s2);
    printf("s3 = %c \n" , s3);
    printf("s4 = %c \n" , s4);
    printf("s5 = %c \n" , s5);

    printf("\n");

//Stop Table

if (s2 == 'C' || s3 == 'C' || s4 == 'C' || s5 == 'C')
{
    Flush(ssp) ;
    Write(ssp, 17, "#%+MOT5SPF0000 *\\");
    sleep(400) ;

    printf("\n") ;
    printf("Pallet loaded \n") ;
```

```
   MainInterface(ssp) ;
   w = 0 ;
}
}
   return err;

}

int RobotEmergencyStop(serial_struct* ssp)

{
   aErr err = aErrNone;
   printf("\n") ;
   printf("Crashed!! User intervention required. \n") ;
    RobotChill(ssp);

   MainInterface(ssp) ;

   return err;

}

int ConveyorChill(serial_struct* ssp)

/*
Docked at Conveyor
Prompts user for route by Interface
*/

{
   aErr err = aErrNone;
   Chill(ssp);

   printf("At Conveyor. Begin manual transfer. \n") ;

   MainInterface(ssp) ;

   return err;
}
```

Line Fusion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "Directions.h"
#include "DirLmtUlt.h"
#include "LineDock.h"
#include "LineFusion.h"
```

```
//check ultra crash

/* SideRightLine */

int SideRightLine(serial_struct* ssp)

/*                                  E
                                 r2 r0 r1  */
{

    Flush(ssp) ;
    aErr err = aErrNone;
    extern int sw ;

    Write(ssp, 17, "#%+LINEL       *\\");
    sleep(400) ;

    err = Read(ssp, 50);

    char r0 = ssp->pinputbuff[9] ;
    char r1 = ssp->pinputbuff[21] ;
    char r2 = ssp->pinputbuff[33] ;

    printf("r0 = %c \n" , r0);
    printf("r1 = %c \n" , r1);
    printf("r2 = %c \n" , r2);

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

      char s0 = ssp->pinputbuff[9] ;
    char s1 = ssp->pinputbuff[10] ;
    char s2 = ssp->pinputbuff[11] ;
    char s3 = ssp->pinputbuff[12] ;
    char s4 = ssp->pinputbuff[13] ;
    char s5 = ssp->pinputbuff[14] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s0 = %c \n" , s0);
    printf("s1 = %c \n" , s1);
    printf("s2 = %c \n" , s2);
```

```
    printf("s3 = %c \n" , s3);
    printf("s4 = %c \n" , s4);
    printf("s5 = %c \n" , s5);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    //Ultrasonic Process Right

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\\"");          //east
    sleep(400) ;
    err = Read(ssp, 17);

    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("ue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

//Line Analysis
//r0 on black r1 r2 on white && u >= 1 go Sideright              m2  r2 r0 r1

if ( (r0 < '9') && (r1 == '9') && (r2 == '9') && ue1 >= '1' && ue2 >= '0' && ue3 >= '0')

Sideright(ssp);

//r1 on black r0 r2 on white && u >= 1 go Sideright45

else if ( (r1 < '9') && (r0 == '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright45(ssp) ;

//r2 on black r0 r1 on white && u >= 1 go Sideright315

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright315(ssp) ;

//r0/r1 on black r2 on white && u >= 1 go Sideright315

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//r0/r2 on black r1 on white && u >= 1 go Sideright315
```

```
else if ( ( r0 < '9' && r1 == '9'&& r2 < '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp);

//Main Chills

//l2 l0 l1 on black

else if ( r1 < '9' && r0 < '9' && r2 < '9')

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white
else if ( r1 == '9' && r0 == '9' && r2 == '9')

{
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' )
)

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//l2 l0 l1 on white
else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1
== 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//Offline and obstacle crash

//l2 l0 l1 on black
```

```
else if ( (r1 < '9' && r0 < '9' && r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white
else if ( (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//On line obstacle detected waiting for clear

//l0 on black l1 l2 on
else if ( (r0 < '9' && r1 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9')&& ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9'&& r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Lmt integration s0

//l0 on black l1 l2 on white go fwd

if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
    printf(" Exit \n");
}

//l1 on black l0 l2 on white go Right

else if ( (s0 == 'C') && (r1 < '9' && r0 == '9'&& r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
```

```
    printf(" Exit \n");
}

//l2 on black l0 l1 on white go Left

else if ( (s0 == 'C') && (r2 < '9' && r0 == '9' && r1 == '9')&& ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
{
    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
    printf(" Exit \n");
}

//l0/l1 on black l2 on white go right

else if ( (s0 == 'C')&& (r0 < '9' && r1 < '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
     //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 < '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
}

//More Chills
//s6 s7

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}
```

```
//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
}

return err;
}

//test

/* SideRightForwardLine */
```

```
int SideRightForwardLine(serial_struct* ssp)

//going sideright sees black line starts going foward

{
/*                                  E
                                  r2 r0 r1

                              f1
                        N     f0
                              f2


 */

  //1st board going right                LINEL

  Flush(ssp) ;
  aErr err = aErrNone;
  extern int sw ;

  Write(ssp, 17, "#%+LINEL       *\\");
  sleep(400) ;

  err = Read(ssp, 50);

  /*int j ;
  for (j = 0; j < 50 ; j++)

  {
  printf("%d \t %c\n", j , ssp->pinputbuff[j]);
  }
 */

  char r0 = ssp->pinputbuff[9] ;
  char r1 = ssp->pinputbuff[21] ;
  char r2 = ssp->pinputbuff[33] ;


  printf("\nr0 = %c \n" , r0);
  printf("r1 = %c \n" , r1);
  printf("r2 = %c \n" , r2);

   sleep(500) ;

   //2nd board foward                          LINER

   Flush(ssp) ;

   Write(ssp, 17, "#%+LINER       *\\");             //change to LITEL
   sleep(400) ;
```

```
 err = Read(ssp, 50);

char f0 = ssp->pinputbuff[9] ;
char f1 = ssp->pinputbuff[21] ;
char f2 = ssp->pinputbuff[33] ;

printf("\nf0 = %c \n" , f0);
printf("f1 = %c \n" , f1);
printf("f2 = %c \n" , f2);

sleep(400) ;

  //Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s0 = ssp->pinputbuff[9] ;
char s1 = ssp->pinputbuff[10] ;
char s2 = ssp->pinputbuff[11] ;
char s3 = ssp->pinputbuff[12] ;
char s4 = ssp->pinputbuff[13] ;
char s5 = ssp->pinputbuff[14] ;
char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("\ns0 = %c \n" , s0);
printf("s1 = %c \n" , s1);
printf("s2 = %c \n" , s2);
printf("s3 = %c \n" , s3);
printf("s4 = %c \n" , s4);
printf("s5 = %c \n" , s5);
printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process SideRight

Flush(ssp) ;
```

```
    Write(ssp, 17, "#%+ULTSW       *\\");          //east
    sleep(400) ;
    err = Read(ssp, 17);

    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("\nue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSS       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

//Sideright Analysis                        m2  r2 r0 r1

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright(ssp);

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp) ;
```

```
//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright45(ssp) ;

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright315(ssp);

// Foward Analysis                              front

/*                                          f1
                                            f0
                                            f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0')  )

Forward(ssp);

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))
 Forward(ssp);

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//normal right
//f1 on black f0 f2 on white go Right
```

```
else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Right(ssp) ;

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp) ;

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Right(ssp) ;

//f0-f1 r0 on black f2 r1 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9'  && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;
```

```
//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r1 < '9'  && r0 == '9' && r2 == '9'&& f2 == '9' && f0 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Left(ssp) ;

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f2 r0-r1 r0-r2 combinations
//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);
```

```
//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Left(ssp);

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

Forward(ssp);
//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

Forward(ssp) ;
```

```
//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//f0 r0-r2 on black go Forward

else if ( (f0 < '9'  && r2 < '9' && r0 < '9' && r1 == '9' && f2 == '9' && f1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp) ;

//Chill time - obstacle < 1m or crashed - obstacle

//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9')) /*|| (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 == '0') ||
```

```
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') )*/

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9')) /*|| (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') )*/

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//f2 f0 f1 on black r1 r2 r0 on white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9')) /* || (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') )*/

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//f2 f0 f1 on white r1 r2 r0 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9')) /* || (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') )*/

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}
```

```
//Lmt switch

//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7
== 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//l2 l0 l1 on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 ==
'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//f0-2 on black r0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9')&& (s6 == 'C' ||
s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//f0-2 on white r0-2 black
else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' ||
s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//Offline and obstacle crash

//All on black
```

```
else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//f0-2 on black r0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
```

```
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//On line obstacle detected waiting for clear

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315
```

```
else if ( (r0 < '9') && (r1 == '9')&& (r2 < '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// Foward Analysis                                front

/*                                                 f1
                                                  f0
                                                  f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward
```

```
else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' && r1 < '9'  && r0 == '9' && r2 == '9' && f2 == '9' && f0 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0 on black f0 f1 on white go Forward
else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward
```

```
else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 f1 [r0 r1 r2] on black go Forward
```

```
else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9' && r2 < '9' && r0 < '9' && r1 == '9'&& f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More Chills
//s6 s7

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right
```

```
else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
}

//More Chills
//s8 s9      Charging

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Charging(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' )&& ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Charging(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s8 == 'C' || s9 == 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Charging(ssp);
```

```
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Charging(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s8 == 'C' || s9 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Charging(ssp) ;
    sw = 0;
}
return err;
}

/* SideLeftLine */

int SideLeftLine(serial_struct* ssp)
/*                                          l2 l0 l1
                                             W       */
{
   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LINEL       *\\");
   sleep(400) ;

   err = Read(ssp, 50);

   char l0 = ssp->pinputbuff[9] ;
   char l1 = ssp->pinputbuff[21] ;
   char l2 = ssp->pinputbuff[33] ;

   printf("\n");
   printf("l0 = %c \n" , l0);
   printf("l1 = %c \n" , l1);
   printf("l2 = %c \n" , l2);
```

```
// sleep(400) ;

 //Limit Switch Analysis

 Flush(ssp) ;

 Write(ssp, 17, "#%+SWTCH       *\\");
 sleep(400) ;

 err = Read(ssp, 19);

 char s0 = ssp->pinputbuff[9] ;
 char s1 = ssp->pinputbuff[10] ;
 char s2 = ssp->pinputbuff[11] ;
 char s3 = ssp->pinputbuff[12] ;
 char s4 = ssp->pinputbuff[13] ;
 char s5 = ssp->pinputbuff[14] ;
 char s6 = ssp->pinputbuff[15] ;
 char s7 = ssp->pinputbuff[16] ;
 char s8 = ssp->pinputbuff[17] ;
 char s9 = ssp->pinputbuff[18] ;

 printf("\n");
 printf("s0 = %c \n" , s0);
 printf("s1 = %c \n" , s1);
 printf("s2 = %c \n" , s2);
 printf("s3 = %c \n" , s3);
 printf("s4 = %c \n" , s4);
 printf("s5 = %c \n" , s5);
 printf("s6 = %c \n" , s6);
 printf("s7 = %c \n" , s7);
 printf("s8 = %c \n" , s8);
 printf("s9 = %c \n" , s9);
 printf("\n");

 //sleep(400);

 //Ultrasonic Process

 Flush(ssp) ;


 Write(ssp, 17, "#%+ULTSW       *\\");
 sleep(400) ;
 err = Read(ssp, 17);

 char uw1 = ssp->pinputbuff[10] ;
 char uw2 = ssp->pinputbuff[11] ;
 char uw3 = ssp->pinputbuff[12] ;

 printf("\n");
```

```
    printf("uw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

    //sleep(400);

//Line Analysis
//on line no obstacle proceed

//l0 on black l1 l2 on white & no obstacle go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')  )

Sideleft(ssp);

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft135(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft225(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9')&& (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft135(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft225(ssp);

//Main Chills
//l2 l0 l1 on black

else if ( l1 < '9' && l0 < '9' && l2 < '9')

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white
```

```
else if ( l1 == '9' && l0 == '9' && l2 == '9')
{
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//Offline and switch closed
//l2 l0 l1 on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9') && (s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' )
)

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//l2 l0 l1 on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1
== 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//Offline and obstacle detected

//l2 l0 l1 on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white
else if ( (l1 == '9' && l0 == '9' && l2 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp) ;
```

```
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//On line obstacle detected waiting for clear

//l0 on black l1 l2 on
else if ( (l0 < '9' && l1 == '9' && l2 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9')&& ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More Chills on line but switch closed
//s6 s7

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 >
'0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp) ;
    sw = 0;
```

```
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >=
'0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp) ;
    sw = 0;
}

//More Chills
//s1

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s1 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9')&& (s1 == 'C')&& ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right
```

```
else if ( (l0 < '9' && l1 < '9' && l2 == '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    sw = 0;
}

return err;

}

/* SideLeftForwardLine */

int SideLeftForwardLine(serial_struct* ssp)
                        /*

                                        f1
                                N       f0
                                        f2
                                          l2 l0 l1
                                             W */

{
   //1st left board

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LINEL       *\\\"");       //side left
   sleep(400) ;

   err = Read(ssp, 50);
   char l0 = ssp->pinputbuff[9] ;
   char l1 = ssp->pinputbuff[21] ;
   char l2 = ssp->pinputbuff[33] ;
```

```
printf("\nl0 = %c \n" , l0);
printf("l1 = %c \n" , l1);
printf("l2 = %c \n" , l2);

sleep(400) ;

//2nd foward board

Flush(ssp) ;

Write(ssp, 17, "#%+LINER       *\\");              //foward  change to LITEL
sleep(400) ;

err = Read(ssp, 50);

char f0 = ssp->pinputbuff[9] ;
char f1 = ssp->pinputbuff[21] ;
char f2 = ssp->pinputbuff[33] ;

printf("\nf0 = %c \n" , f0);
printf("f1 = %c \n" , f1);
printf("f2 = %c \n" , f2);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s0 = ssp->pinputbuff[9] ;
char s1 = ssp->pinputbuff[10] ;
char s2 = ssp->pinputbuff[11] ;
char s3 = ssp->pinputbuff[12] ;
char s4 = ssp->pinputbuff[13] ;
char s5 = ssp->pinputbuff[14] ;
char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("\ns0 = %c \n" , s0);
printf("s1 = %c \n" , s1);
printf("s2 = %c \n" , s2);
printf("s3 = %c \n" , s3);
printf("s4 = %c \n" , s4);
printf("s5 = %c \n" , s5);
```

```
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    sleep(400) ;

    //Ultrasonic Process SideLeft

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char uw1 = ssp->pinputbuff[10] ;
    char uw2 = ssp->pinputbuff[11] ;
    char uw3 = ssp->pinputbuff[12] ;

    printf("\nuw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

    sleep(400) ;

    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSN       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

//Analysis with sideleft

//l0 on black l1 l2 [f0 f1 f2] on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0')  )

Sideleft(ssp);
```

```
//l1 on black l0 l2 [f0 f1 f2] on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft135(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft225(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft135(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9' && l2 < '9' && f0 == '9' && f2 == '9' && f1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft225(ssp);

//Analysis with foward

//normal forward

//f0 on black f1 f2 l1 l2 l0 on white stop sideleft continue fwd - check this

if ( (f0 < '9' && f1 == '9' && f2 == '9' && l0 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0 & l0 on black f1 f2 l1 l2 on white stop sideleft go fwd

else if ( (f0 < '9' && l0 < '9' && f1 == '9' && f2 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0 l1 on black go forward

else if ( (f0 < '9' && l0 == '9' && l1 < '9' && f1 == '9' && f2 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )
```

```
Forward(ssp);

//f0 l2 on black go forward

else if ( (f0 < '9' && l2 < '9' && l1 == '9' && f1 == '9' && f2 == '9' && l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp);

//f0 l0-l1 l0-l2 combinations

//f0 l0-l1 on black go forward

else if ( (f0 < '9' && l0 < '9' && l1 < '9' && f1 == '9' && f2 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp);

//f0 l0-l2 on black go forward

else if ( (f0 < '9' && l0 < '9' && l2 < '9' && f1 == '9' && f2 == '9' && l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f1 l0 on black f2 l1 l2 on white stop sideleft go foward

else if (( f0 < '9' && l0 < '9' && f1 < '9' && f2 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f2 l0 on black r1 l1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && l0 < '9' && f2 < '9' && f1 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f1 l0-l1 l0-l2 combinations

//f0-f1 l0-l1 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && f1 < '9' && l0 < '9' && l1 < '9' && f2 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp);

//f0-f1 l0-l2 on black r1 l2 on white stop sideleft go foward
```

```
else if ( (f0 < '9' && f1 < '9' && l0 < '9' && l2 < '9' && f2 == '9' && l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f2 l0-l1 l0-l2 combinations

//f0-f2 l0-l1 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && f2 < '9' && l0 < '9' && l1 < '9' && f1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f2 l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && f2 < '9' && l0 < '9' && l2 < '9' && f1 == '9' && l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0 l1-l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f1 == '9' && f2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f1 f0-f2 l1-l0-l2 combinations

//f0-f1 l1-l0-l2 combinations

else if ( (f0 < '9' && f1 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f0-f2 l1-l0-l2 combinations

else if ( (f0 < '9' && f2 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp);

//f1 l1-l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f1 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f0 == '9' && f2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);
```

```
//f2 l1-l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f2 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f0 == '9' && f1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp);

//f1 l0 on black r0 r2 l1 l2 on white stop sideleft go foward

else if ((f1 < '9' && l0 < '9' && f0 == '9' && f2 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f1 l0-l1 on black r0 r2 l1 l2 on white stop sideleft go foward

else if ((f1 < '9' && l0 < '9' && l1 < '9' &&  f0 == '9' && f2 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f1 l0-l2 on black r0 r2 l1 l2 on white stop sideleft go foward

else if ((f1 < '9' && l0 < '9' && l2 < '9' &&  f0 == '9' && f2 == '9' && l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f2 l0 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9' && l0 < '9' &&  f0 == '9' && f1 == '9'  && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f2 l0-l1 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9' && l0 < '9' && l1 < '9' && f0 == '9' && f1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f2 l0-l2 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9' && l0 < '9' && l2 < '9' &&  f0 == '9' && f1 == '9' &&  l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))
 Forward(ssp);

//f2 l1 on black r0 r1 l1 l2 on white stop sideleft go foward
```

```
else if ( (f2 < '9' && l1 < '9' && l0 == '9' && l2 == '9' &&  f0 == '9' && f1 == '9')&& (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f2 l2 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9' && l2 < '9' && l0 == '9' && l1 == '9' && f0 == '9' && f1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f1 l1 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f1 < '9' && l1 < '9' && l0 == '9' && l2 == '9' &&  f0 == '9' && f2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Forward(ssp);

//f1 l2 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f1 < '9' && l2 < '9' && l0 == '9' && l1 == '9' && f0 == '9' && f2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Forward(ssp);

//normal right
//f1 on black f0 f2 [ l1 l2 l0 ] on white go Right on white

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && l0 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Right(ssp) ;

//normal left
//f2 on black f0 f1 [ l1 l2 l0 ] on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && l0 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Left(ssp) ;

//normal right
//f0/f1 on black f2 [ l1 l2 l0 ] on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && l0 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
Right(ssp) ;

//normal left
//f0/f2 on black f1 [ l1 l2 l0 ] on white go Left
```

```
else if ( (f0 < '9' && f1 == '9'&& f2 < '9' && l0 == '9' && l1 == '9' && l2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

Left(ssp);

//Chill Time

//Main Chills

//All on black

else if (f1 < '9' && f0 < '9' && f2 < '9' && l0 < '9' && l1 < '9' && l2 < '9')

{
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//All on white

else if (f1 == '9' && f0 == '9' && f2 == '9' && l0 == '9' && l1 == '9' && l2 == '9')

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//f0-2 on black l0-2 white

else if (f1 < '9' && f0 < '9' && f2 < '9' && l0 == '9' && l1 == '9' && l2 == '9')

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//f0-2 on white l0-2 black

else if (f1 == '9' && f0 == '9' && f2 == '9' && l0 < '9' && l1 < '9' && l2 < '9')
{
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//Offline and switch closed
```

```
//All on black

else if ( (f1 < '9' && f0 < '9' && f2 < '9' && l0 < '9' && l1 < '9' && l2 < '9') && (s6 == 'C' || s7 ==
'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9' && l0 == '9' && l1 == '9' && l2 == '9') && (s6 == 'C'
|| s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//f0-2 on black l0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9' && l0 == '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7
== 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//f0-2 on white l0-2 black

else if ( (f1 == '9' && f0 == '9' && f2 == '9' && l0 < '9' && l1 < '9' && l2 < '9') && (s6 == 'C' || s7
== 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//Offline and obstacle detected

//All on black
else if ( (f1 < '9' && f0 < '9' && f2 < '9' && l0 < '9' && l1 < '9' && l2 < '9') && ( (uw1 == '0' &&
uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0'
&& uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (un1 == '0' && un2 == '0' && un3
```

```
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9' && l0 == '9' && l1 == '9' && l2 == '9')  && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (un1 == '0' && un2 == '0'
&& un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//f0-2 on white l0-2 black

else if ( (f1 == '9' && f0 == '9' && f2 == '9' && l0 < '9' && l1 < '9' && l2 < '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//f0-2 on black l0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9' && l0 == '9' && l1 == '9' && l2 == '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}
```

```
//Analysis with sideleft

//Online + obstacle

//l0 on black l1 l2 [f0 f1 f2] on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 on black l0 l2 [f0 f1 f2] on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && ( (uw1 ==
'0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2
> '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && ( (uw1 ==
'0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2
> '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9' && f0 == '9' && f2 == '9' && f1 == '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l2 on black l1 on white go Left
```

```
else if ( (l0 < '9' && l1 == '9' && l2 < '9' && f0 == '9' && f2 == '9' && f1 == '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Analysis with foward

//normal forward

//f0 on black f1 f2 l1 l2 l0 on white stop sideleft continue fwd - check this

if ( (f0 < '9' && f1 == '9' && f2 == '9' && l0 == '9' && l1 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 & l0 on black f1 f2 l1 l2 on white stop sideleft go fwd

else if ( (f0 < '9' && l0 < '9' && f1 == '9' && f2 == '9' && l1 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 l1 on black go forward

else if ( (f0 < '9' && l0 == '9' && l1 < '9' && f1 == '9' && f2 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 l2 on black go forward

else if ( (f0 < '9' && l2 < '9' && l1 == '9' && f1 == '9' && f2 == '9' && l1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 l0-l1 l0-l2 combinations

//f0 l0-l1 on black go forward

else if ( (f0 < '9' && l0 < '9' && l1 < '9' && f1 == '9' && f2 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 l0-l2 on black go forward

else if ( (f0 < '9' && l0 < '9' && l2 < '9' && f1 == '9' && f2 == '9' && l1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 l0 on black f2 l1 l2 on white stop sideleft go foward

else if (( f0 < '9' && l0 < '9' && f1 < '9' && f2 == '9' && l1 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//f0-f2 l0 on black r1 l1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && l0 < '9' && f2 < '9' && f1 == '9' && l1 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//f0-f1 l0-l1 l0-l2 combinations

//f0-f1 l0-l1 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && f1 < '9' && l0 < '9' && l1 < '9' && f2 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && f1 < '9' && l0 < '9' && l2 < '9' && f2 == '9' && l1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 l0-l1 l0-l2 combinations

//f0-f2 l0-l1 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && f2 < '9' && l0 < '9' && l1 < '9' && f1 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f0 < '9' && f2 < '9' && l0 < '9' && l2 < '9' && f1 == '9' && l1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//f0 l1-l0-l2 on black r1 l2 on white stop sideleft go foward
```

```
else if ( (f0 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f1 == '9' && f2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 f0-f2 l1-l0-l2 combinations

//f0-f1 l1-l0-l2 combinations

else if ( (f0 < '9' && f1 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f2 == '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 l1-l0-l2 combinations

else if ( (f0 < '9' && f2 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f1 == '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 l1-l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f1 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f0 == '9' && f2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 l1-l0-l2 on black r1 l2 on white stop sideleft go foward

else if ( (f2 < '9' && l0 < '9' && l1 < '9' && l2 < '9' && f0 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 l0 on black r0 r2 l1 l2 on white stop sideleft go foward

else if ((f1 < '9' && l0 < '9' && f0 == '9' && f2 == '9' && l1 == '9' && l2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 l0-l1 on black r0 r2 l1 l2 on white stop sideleft go foward

else if ((f1 < '9') && (l0 < '9') && (l1 < '9') &&  (f0 == '9')&& (f2 == '9')&& (l2 == '9') && ( (un1
== '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2
> '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 l0-l2 on black r0 r2 l1 l2 on white stop sideleft go foward

else if ((f1 < '9') && (l0 < '9') && (l2 < '9') &&  (f0 == '9')&& (f2 == '9')&& (l1 == '9') && ( (un1
== '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2
> '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 l0 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9') && (l0 < '9') &&  (f0 == '9')&& (f1 == '9')  && (l1 == '9') && (l2 == '9') && (
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 l0-l1 on black r0 r1 l1 l2 on white stop sideleft go foward
```

```
else if ( (f2 < '9') && (l0 < '9') && (l1 < '9') && (f0 == '9')&& (f1 == '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 l0-l2 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9' && l0 < '9'  && l2 < '9' &&  f0 == '9'&& f1 == '9'  && l1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 l1 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9') && (l1 < '9')&& (l0 == '9')  && (l2 == '9') &&  (f0 == '9')&& (f1 == '9')&& ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 l2 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f2 < '9') && (l2 < '9') && (l0 == '9')  && (l1 == '9') && (f0 == '9')&& (f1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 l1 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f1 < '9') && (l1 < '9')&& (l0 == '9')  && (l2 == '9') &&  (f0 == '9')&& (f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//f1 l2 on black r0 r1 l1 l2 on white stop sideleft go foward

else if ( (f1 < '9') && (l2 < '9') && (l0 == '9')  && (l1 == '9') && (f0 == '9')&& (f2 == '9') && (
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f1 on black f0 f2 [ l1 l2 l0 ] on white go Right on white

else if ( (f1 < '9') && (f0 == '9') && (f2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f2 on black f0 f1 [ l1 l2 l0 ] on white go Left

else if ( (f2 < '9') && (f0 == '9') && (f1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f0/f1 on black f2 [ l1 l2 l0 ] on white go right

else if ( (f0 < '9') && (f1 < '9') && (f2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//normal left
//f0/f2 on black f1 [ l1 l2 l0 ] on white go Left
```

```
else if ( (f0 < '9') && (f1 == '9')&& (f2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More Chills on line but switch closed
//s6 s7

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (uw1 == '1' && uw2 >=
'0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 == '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 == '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (uw1 == '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
```

```
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)
{
    EmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9'&& f2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 == '1' && uw2 >=
'0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp) ;
    sw = 0;
}

//More Chills
//s8 s9      Charging

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C'  ) && ( (uw1 == '1' && uw2 >=
'0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    Charging(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' ) && ( (uw1 == '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    Charging(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s8 == 'C' || s9 == 'C' ) && ( (uw1 == '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)
```

```
{
    Charging(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right
else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' )&& ( (uw1 == '1' && uw2 >=
'0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    Charging(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9'&& f2 < '9') && (s8 == 'C' || s9 == 'C' ) && ( (uw1 == '1' && uw2 >=
'0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    Charging(ssp) ;
    sw = 0;
}
return err ;

}

/* ReverseSideLeftLine */

int ReverseSideLeftLine(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe


                                    b1
                                    b0      S
                                    b2
                            l2 l0 l1
                               W
*/

{

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;
```

```
Write(ssp, 17, "#%+LINER       *\\");          //change to LITER
sleep(400) ;

err = Read(ssp, 50);

char b0 = ssp->pinputbuff[9] ;
char b1 = ssp->pinputbuff[21] ;
char b2 = ssp->pinputbuff[33] ;

printf("b0 = %c \n" , b0);
printf("b1 = %c \n" , b1);
printf("b2 = %c \n" , b2);

sleep(400) ;

//Sideleft

Flush(ssp) ;

Write(ssp, 17, "#%+LINEL       *\\");
sleep(400) ;

err = Read(ssp, 50);

char l0 = ssp->pinputbuff[9] ;
char l1 = ssp->pinputbuff[21] ;
char l2 = ssp->pinputbuff[33] ;

printf("l0 = %c \n" , l0);
printf("l1 = %c \n" , l1);
printf("l2 = %c \n" , l2);

sleep(400) ;

 //Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s0 = ssp->pinputbuff[9] ;
char s1 = ssp->pinputbuff[10] ;
char s2 = ssp->pinputbuff[11] ;
char s3 = ssp->pinputbuff[12] ;
char s4 = ssp->pinputbuff[13] ;
char s5 = ssp->pinputbuff[14] ;
char s6 = ssp->pinputbuff[15] ;
```

```
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("s0 = %c \n" , s0);
printf("s1 = %c \n" , s1);
printf("s2 = %c \n" , s2);
printf("s3 = %c \n" , s3);
printf("s4 = %c \n" , s4);
printf("s5 = %c \n" , s5);
printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");
sleep(400) ;
err = Read(ssp, 17);

char us1 = ssp->pinputbuff[10] ;
char us2 = ssp->pinputbuff[11] ;
char us3 = ssp->pinputbuff[12] ;

printf("us1 = %c \n" , us1);
printf("us2 = %c \n" , us2);
printf("us3 = %c \n" , us3);

sleep(400) ;

//Ultrasonic Process SideLeft

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");
sleep(400) ;
err = Read(ssp, 17);

char uw1 = ssp->pinputbuff[10] ;
char uw2 = ssp->pinputbuff[11] ;
char uw3 = ssp->pinputbuff[12] ;

printf("uw1 = %c \n" , uw1);
printf("uw2 = %c \n" , uw2);
printf("uw3 = %c \n" , uw3);
```

```
//Reverse Analysis

//b0 on black b1 b2 on white go fwd

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverse(ssp);

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b2 on black b0 b1 on white go Left

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp) ;

//b0/b1 on black b2 on white go right

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b0/b2 on black b1 on white go Left

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp);

//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))
Sideleft(ssp);

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))
```

```
Sideleft(ssp);

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);
```

```
//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b1 on black l1 l2 on white go fwd
```

```
else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )
 Sideleft(ssp);

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft135(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft225(ssp) ;

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft135(ssp) ;

//Normal Sideleft225
//l0/l2 on black l1 on white go Left
```

```
else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft225(ssp);

//Lmt Ultra integration

//Main Chills

//All on black

else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//All on white

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9')

{
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9')

{
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//Offline and switch closed
//All on black
```

```
else if ((l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}
//All on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//Offline and obstacle detected

//All on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//All on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0'
&& us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') ||
(us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3
== '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 ==
'0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3
== '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 ==
'0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//On line obstacle detected waiting for clear
```

```
//Reverse Analysis

//b0 on black b1 b2 on white go Reverse

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1
== '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 >
'0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b1 on black b0 b2 on white go Reverseleft

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b2 on black b0 b1 on white go Reverseright

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//b0/b1 on black b2 on white go Reverseleft

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b2 on black b1 on white go Reverseright

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go Sideleft
else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right
```

```
else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft225
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


/*More Chills on line but switch closed
s6 s7
Reverse
*/
```

```
if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right
else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
```

```
}

/*More Chills
s1
Sideleft
*/

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s1 == 'C' ) && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9')&& (s1 == 'C')&& ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left
```

```
else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    sw = 0;
}
return err;
}

/* ReverseSideRightLine */

int ReverseSideRightLine(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideright
RM to Conveyor
                                    E
                                r2 r0 r1
                                    b1
                                    b0     S
                                    b2
 */

{

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LINER       *\\\"");          //change to LITER
   sleep(400) ;

   err = Read(ssp, 50);

   char b0 = ssp->pinputbuff[9] ;
   char b1 = ssp->pinputbuff[21] ;
   char b2 = ssp->pinputbuff[33] ;

   printf("\nb0 = %c \n" , b0);
   printf("b1 = %c \n" , b1);
   printf("b2 = %c \n" , b2);

   sleep(400) ;

   //Sideright

   Flush(ssp) ;

   Write(ssp, 17, "#%+LINEL       *\\\");                //change to LINER
   sleep(400) ;
```

```
err = Read(ssp, 50);

char r0 = ssp->pinputbuff[9] ;
char r1 = ssp->pinputbuff[21] ;
char r2 = ssp->pinputbuff[33] ;

printf("\nr0 = %c \n" , r0);
printf("r1 = %c \n" , r1);
printf("r2 = %c \n" , r2);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s0 = ssp->pinputbuff[9] ;
char s1 = ssp->pinputbuff[10] ;
char s2 = ssp->pinputbuff[11] ;
char s3 = ssp->pinputbuff[12] ;
char s4 = ssp->pinputbuff[13] ;
char s5 = ssp->pinputbuff[14] ;
char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("\ns0 = %c \n" , s0);
printf("s1 = %c \n" , s1);
printf("s2 = %c \n" , s2);
printf("s3 = %c \n" , s3);
printf("s4 = %c \n" , s4);
printf("s5 = %c \n" , s5);
printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process Reverse

Flush(ssp) ;
```

```
    Write(ssp, 17, "#%+ULTSS       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Ultrasonic Process SideRight

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSE       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("\nue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

//Reverse Analysis

//b0 on black b1 b2 on white go fwd

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') && (us1
>= '1' && us2 >= '0' && us3 >= '0'))

Reverse(ssp);

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b2 on black b0 b1 on white go Left

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9')&& (r0 == '9') && (r1 == '9') && (r2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))
```

```
Reverseright(ssp) ;

//b0/b1 on black b2 on white go right

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b0/b2 on black b1 on white go Left

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp);

//Sideright analysis

//Normal Sideright
//r0 on black r1 r2 on white go fwd

if ( (r0 < '9') && (r1 == '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (ue1
>= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0 b0 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b0 < '9') && (r1 == '9') && (r2 == '9') && (b1 == '9') && (b2 == '9') && (ue1
>= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0 b1 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b1 < '9') && (r1 == '9') && (r2 == '9') && (b0 == '9') && (b2 == '9') && (ue1
>= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0 b2 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b2 < '9') && (r1 == '9') && (r2 == '9') && (b0 == '9') && (b1 == '9') && (ue1
>= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0-r1 b0 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b0 < '9')  && (r2 == '9') && (b1 == '9') && (b2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))
```

```
Sideright(ssp);

//r0-r2 b0 on black r1 on white go fwd

if ( (r0 < '9') && (r2 < '9') && (b0 < '9')  && (r1 == '9') && (b1 == '9') && (b2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0 b0-b1 on black r1 r2 on white go fwd
if ( (r0 < '9') && (b1 < '9') && (b0 < '9')  && (r2 == '9') && (r1 == '9') && (b2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0 b0-b2 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b2 < '9') && (b0 < '9')  && (r2 == '9') && (r1 == '9') && (b1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0-r1 b0-b1 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b1 < '9') && (b0 < '9')  && (r2 == '9') && (b2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0-r1 b0-b2 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b2 < '9') && (b0 < '9')  && (r2 == '9') && (b1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0-r2 b0-b1 on black r1 on white go fwd

if ( (r0 < '9') && (r2 < '9') && (b1 < '9') && (b0 < '9')  && (r1 == '9') && (b2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);

//r0-r2 b0-b2 on black r1 on white go fwd

if ( (r0 < '9') && (r2 < '9') && (b2 < '9') && (b0 < '9') && (r1 == '9') && (b1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0'))

Sideright(ssp);
```

```
//r0 b0-b1-b2 on black r1 r2 on white go fwd

if ( (r0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (r1 == '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r0-r1 b0-b1-b2 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )
Sideright(ssp);

//r0-r2 b0-b1-b2 on black r1 on white go fwd

if ( (r0 < '9') && (r2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (r1 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r1 b0-b1-b2 on black r0 r2 on white go fwd

if ( (r1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (r0 == '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r2 b0-b1-b2 on black r1 r0 on white go fwd

if ( (r2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (r1 == '9') && (r0 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r1 b0 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b0 < '9') && (r0 == '9') && (b1 == '9')&& (b2 == '9')&& (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r1 b0-b1 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b0 < '9') && (b1 < '9')&& (r0 == '9') && (b2 == '9')&& (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r1 b0-b2 on black r1 r2 on white go fwd
```

```
if ( (r1 < '9') && (b0 < '9') && (b2 < '9')&& (r0 == '9') && (b1 == '9')&& (r2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r2 b0 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b0 < '9') && (r0 == '9') && (b1 == '9')&& (b2 == '9')&& (r1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r2 b0 b1 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b0 < '9') && (b1 < '9')&& (r0 == '9') && (b2 == '9')&& (r1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r2 b0 b2 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b0 < '9') && (b2 < '9')&& (r0 == '9') && (b1 == '9')&& (r1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r2 b1 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b1 < '9') && (b2 == '9')&& (r0 == '9') && (b0 == '9')&& (r1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r2 b2 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b2 < '9') && (b1 == '9')&& (r0 == '9') &&  (b0 == '9')&& (r1 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r1 b1 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b1 < '9') && (b2 == '9')&& (r0 == '9') && (b0 == '9')&& (r2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp);

//r1 b2 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b2 < '9') && (b1 == '9')&& (r0 == '9') && (b0 == '9')&& (r2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )
```

```
Sideright(ssp);

//Normal Sideright45
//r1 on black r0 r2 on white go Right

else if ( (r1 < '9') && (r0 == '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright45(ssp) ;

//Normal Sideright315
//r2 on black r0 r1 on white go Left

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright315(ssp) ;

//Normal Sideright45
//r0/r1 on black r2 on white go right

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright45(ssp) ;

//Normal Sideright315
//r0/r2 on black r1 on white go Left

else if ( (r0 < '9') && (r1 == '9')&& (r2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

Sideright315(ssp);

//Main Chills

//All on black

else if ( (r1 < '9') && (r0 < '9') && (r2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9'))

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//All on white

else if ( (r1 == '9') && (r0 == '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') )
{
```

```
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//r2 r0 r1 on white b0 b1 b2 on black

else if ( (r1 == '9') && (r0 == '9') && (r2 == '9') && (b1 < '9') && (b0 < '9') && (b2 < '9') )

{
    Chill(ssp);
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//r2 r0 r1 on black b0 b1 b2 on white

else if ( (r1 < '9') && (r0 < '9') && (r2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9'))

{
    Chill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//Offline and switch closed
//All on black

else if ( ( (r1 < '9') && (r0 < '9') && (r2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')) && (s6 ==
'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//All on white

else if ( ( (r1 == '9') && (r0 == '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') )
&& (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//r2 r0 r1 on white b0 b1 b2 on black
```

```
else if ( ( (r1 == '9') && (r0 == '9') && (r2 == '9') && (b1 < '9') && (b0 < '9') && (b2 < '9') ) &&
(s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    sw = 0;
}

//r2 r0 r1 on black b0 b1 b2 on white

else if ( ( (r1 < '9') && (r0 < '9') && (r2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9')) &&
(s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )
{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    sw = 0;
}

//Offline and obstacle crash

//All on black

else if ( ( (r1 < '9') && (r0 < '9') && (r2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9'))  && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (us1 == '0' && us2 == '0' && us3 ==
'0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0'
&& us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//All on white

else if ( ( (r1 == '9') && (r0 == '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') )
&& ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 ==
'0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (us1 == '0' && us2 ==
'0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0')
|| (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//Offline and obstacle crash
```

```
//r2 r0 r1 on white b0 b1 b2 on black

else if ( ( (r1 == '9') && (r0 == '9') && (r2 == '9') && (b1 < '9') && (b0 < '9') && (b2 < '9') )  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}
//r2 r0 r1 on black b0 b1 b2 on white

else if ( ( (r1 < '9') && (r0 < '9') && (r2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9')) && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    sw = 0;
}

//On line obstacle detected waiting for clear

//Reverse Analysis

//b0 on black b1 b2 on white go fwd

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//b2 on black b0 b1 on white go Left

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9')&& (r0 == '9') && (r1 == '9') && (r2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//b0/b1 on black b2 on white go right

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//b0/b2 on black b1 on white go Left

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (r0 == '9') && (r1 == '9') && (r2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Sideright analysis

//Normal Sideright
//r0 on black r1 r2 on white go fwd

if ( (r0 < '9') && (r1 == '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0 b0 on black r1 r2 on white go fwd
```

```
if ( (r0 < '9') && (b0 < '9') && (r1 == '9') && (r2 == '9') && (b1 == '9') && (b2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0 b1 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b1 < '9') && (r1 == '9') && (r2 == '9') && (b0 == '9') && (b2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0 b2 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b2 < '9') && (r1 == '9') && (r2 == '9') && (b0 == '9') && (b1 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r1 b0 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b0 < '9')  && (r2 == '9') && (b1 == '9') && (b2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r2 b0 on black r1 on white go fwd

if ( (r0 < '9') && (r2 < '9') && (b0 < '9')  && (r1 == '9') && (b1 == '9') && (b2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}


//r0 b0-b1 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b1 < '9') && (b0 < '9')  && (r2 == '9') && (r1 == '9') && (b2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0 b0-b2 on black r1 r2 on white go fwd

if ( (r0 < '9') && (b2 < '9') && (b0 < '9')  && (r2 == '9') && (r1 == '9') && (b1 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r1 b0-b1 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b1 < '9') && (b0 < '9')  && (r2 == '9') && (b2 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r1 b0-b2 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b2 < '9') && (b0 < '9')  && (r2 == '9') && (b1 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r2 b0-b1 on black r1 on white go fwd
```

```
if ( (r0 < '9') && (r2 < '9') && (b1 < '9') && (b0 < '9')  && (r1 == '9') && (b2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r2 b0-b2 on black r1 on white go fwd

if ( (r0 < '9') && (r2 < '9') && (b2 < '9') && (b0 < '9') && (r1 == '9') && (b1 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0 b0-b1-b2 on black r1 r2 on white go fwd

if ( (r0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (r1 == '9') && (r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r1 b0-b1-b2 on black r2 on white go fwd

if ( (r0 < '9') && (r1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0-r2 b0-b1-b2 on black r1 on white go fwd

if ( (r0 < '9') && (r2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (r1 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//r1 b0-b1-b2 on black r0 r2 on white go fwd

if ( (r1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (r0 == '9') && (r2 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 b0-b1-b2 on black r1 r0 on white go fwd

if ( (r2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (r1 == '9') && (r0 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 b0 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b0 < '9') && (r0 == '9') && (b1 == '9')&& (b2 == '9')&& (r2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 b0-b1 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b0 < '9') && (b1 < '9')&& (r0 == '9') && (b2 == '9')&& (r2 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 b0-b2 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b0 < '9') && (b2 < '9')&& (r0 == '9') && (b1 == '9')&& (r2 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 b0 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b0 < '9') && (r0 == '9') && (b1 == '9')&& (b2 == '9')&& (r1 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 b0 b1 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b0 < '9') && (b1 < '9')&& (r0 == '9') && (b2 == '9')&& (r1 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 b0 b2 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b0 < '9') && (b2 < '9')&& (r0 == '9') && (b1 == '9')&& (r1 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 b1 on black r1 r2 on white go fwd

if ( (r2 < '9') && (b1 < '9') && (b2 == '9')&& (r0 == '9') && (b0 == '9')&& (r1 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 b2 on black r1 r2 on white go fwd
```

```
if ( (r2 < '9') && (b2 < '9') && (b1 == '9')&& (r0 == '9') &&  (b0 == '9')&& (r1 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 b1 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b1 < '9') && (b2 == '9')&& (r0 == '9') && (b0 == '9')&& (r2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 b2 on black r1 r2 on white go fwd

if ( (r1 < '9') && (b2 < '9') && (b1 == '9')&& (r0 == '9') && (b0 == '9')&& (r2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideright45
//r1 on black r0 r2 on white go Right

else if ( (r1 < '9') && (r0 == '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideright315
//r2 on black r0 r1 on white go Left

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideright45
//r0/r1 on black r2 on white go right

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//Normal Sideright315
//r0/r2 on black r1 on white go Left

else if ( (r0 < '9') && (r1 == '9')&& (r2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Lmt integration s0
//On line SideRight only

//l0 on black l1 l2 on white go fwd

if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3 >=
'0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0'
&& ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{

    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
    printf(" Exit \n");
}

//l1 on black l0 l2 on white go Right

else if ( (s0 == 'C') && (r1 < '9' && r0 == '9'&& r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
    printf(" Exit \n");
}

//l2 on black l0 l1 on white go Left

else if ( (s0 == 'C') && (r2 < '9' && r0 == '9' && r1 == '9')&& ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
    printf(" Exit \n");
}

//l0/l1 on black l2 on white go right

else if ( (s0 == 'C')&& (r0 < '9' && r1 < '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
     //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
    Chill(ssp) ;
}

//l0/l2 on black l1 on white go Left

else if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 < '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    //Chill(ssp) ;
    TableSequence(ssp);
    sw = 0;
    Chill(ssp) ;
}

/*More Chills on line but switch closed
s6 s7
Reverse only
*/
```

```
if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    sw = 0;
```

```
}

return err;

}
```

1.4 Light Following Technique
Light Interface.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightFusion.h"
#include "LightRoutes.h"
#include "LightInterface.h"
#include "LightDock.h"
#include "MainInterface.h"

//#include "Directions.h"

int LightInterface(serial_struct* ssp)

{
   Flush(ssp) ;

   aErr err = aErrNone;
   extern int sw ;

   int route ;

   printf("\nPlease select a Light route: \n\n"  ) ;

   printf("Conveyor to Lathe \t 1 \nLathe to Conveyor \t 2 \n\nConveyor to RM \t\t 3 \nRM to
Conveyor \t\t 4 \n\nLathe to RM  \t\t 5 \nRM to Lathe\t\t 6 \n\nExit to Main \t\t 7 \n\n") ;
   scanf("%d", &route) ;

   printf("\n") ;

   if (route == 1)
   Conveyor_Lathe_Light(ssp) ;

   else if (route == 2)
   Lathe_Conveyor_Light(ssp) ;

   else if (route == 3)
   Conveyor_RM_Light(ssp) ;
```

```
   else if (route == 4)
   RM_Conveyor_Light(ssp) ;

   else if (route == 5)
   Lathe_RM_Light(ssp) ;

   else if (route == 6)
   RM_Lathe_Light(ssp) ;

   else if (route == 7)
   MainInterface(ssp) ;

   else printf("\nUnknown route. Please enter new route\n") ;

   return err;
}
```

Light Routes.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightFusion.h"
#include "LightDock.h"

//Conveyor_Lathe

int Conveyor_Lathe_Light(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("\nInitialising Conveyor to Lathe sequence \n\n\n") ;

   while(sw == 1)
   SideLeftLight(ssp) ;

   return err;
}

//Lathe_Conveyor

int Lathe_Conveyor_Light(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;
```

```
    printf("Starting Lathe_Conveyor sequence \n\n\n") ;

    while(sw == 1)
    SideRightLight(ssp) ;

    return err;
}

//Lathe_RM

int Lathe_RM_Light(serial_struct* ssp)

{
    aErr err = aErrNone;
    extern int sw ;

    printf("Starting Lathe_RM sequence \n\n\n") ;

    while(sw == 1)
    Lathe_RM_Light1(ssp) ;

    return err;
}



//RM_Lathe

int RM_Lathe_Light(serial_struct* ssp)

{
    aErr err = aErrNone;
    extern int sw ;

    printf("Starting RM_Lathe sequence \n\n\n") ;

    while(sw == 1)
    RM_Lathe_Light1(ssp) ;

    return err;
}

//RM_Conveyor

int RM_Conveyor_Light(serial_struct* ssp)

{
    aErr err = aErrNone;
    extern int sw ;
```

```
   printf("Starting RM_Conveyor sequence \n\n\n") ;

   while(sw == 1)
   ReverseLight(ssp) ;

   return err;
}

int Conveyor_RM_Light(serial_struct* ssp)

{
   aErr err = aErrNone;
   extern int sw ;

   printf("Starting Conveyor_RM sequence \n\n\n") ;

   while(sw == 1)
   ForwardLight(ssp) ;

   return err;
}
```

Light Dock.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "RobotMotor.h"
#include "LightDock.h"
#include "LightTestInterface.h"

int LatheChillLight(serial_struct* ssp)

/*
Docked at Lathe
Prompts user for route by Interface
*/

{
   aErr err = aErrNone;
   Chill(ssp);

   printf("\n0.5m in front of Light.\n") ;
   printf("At Lathe. Begin manual transfer. \n") ;

   MainInterface(ssp) ;

   return err;
```

```
}

int RMChillLight(serial_struct* ssp)

/*
Docked at RM
Prompts user for route by Interface
*/

{
   aErr err = aErrNone;
   Chill(ssp);

   printf("\n0.5m in front of Light.\n") ;
   printf("At RM. Begin manual transfer \n") ;

   MainInterface(ssp) ;

   return err;
}

int EmergencyStopLight(serial_struct* ssp)

{
   aErr err = aErrNone;
   Chill(ssp);

   printf("\n") ;
   printf("Crashed!! User intervention required. \n") ;

   MainInterface(ssp) ;

   return err;
}

int ChargingLight(serial_struct* ssp)

{
   //disconnect power

   aErr err = aErrNone;
   Chill(ssp);

   printf("\nCharging \n") ;
   MainInterface(ssp) ;

   return err;
}

int ConveyorChillLight(serial_struct* ssp)
```

```
/*
Docked at Conveyor
Prompts user for route by Interface
*/

{
   aErr err = aErrNone;

   Chill(ssp);

   printf("\n0.5 m in front of Light. \n") ;
   printf("At Conveyor. Begin manual transfer. \n") ;

   MainInterface(ssp) ;

   return err;
}

int TableSequenceLight(serial_struct* ssp)

{
   aErr err = aErrNone;

   Chill(ssp) ;

   int w = 1 ;

   printf("\n") ;
   printf("Transfer sequence activated \n") ;
   printf("Loading pallet . . . \n") ;

   while(w==1)

{
   Flush(ssp) ;
   Write(ssp, 17, "#%+MOT5SPF0000 *\\");
   sleep(400) ;
   Write(ssp, 17, "#%+MOT5SPF0500 *\\");
   sleep(400) ;

   //Check if s2 3 4 5 compressed

   Flush(ssp) ;

   Write(ssp, 17, "#%+SWTCH       *\\");
   sleep(400) ;

   err = Read(ssp, 19);

   int n ;
   for (n = 0; n < 19 ; n++)
```

```
    {
    printf("%d \t %c\n", n , ssp->pinputbuff[n]);
    }

    char s2 = ssp->pinputbuff[11] ;
    char s3 = ssp->pinputbuff[12] ;
    char s4 = ssp->pinputbuff[13] ;
    char s5 = ssp->pinputbuff[14] ;

    printf("s2 = %c \n" , s2);
    printf("s3 = %c \n" , s3);
    printf("s4 = %c \n" , s4);
    printf("s5 = %c \n" , s5);
    printf("\n");


//Stop Table

 if (s2 == 'C' || s3 == 'C' || s4 == 'C' || s5 == 'C')
{

    Flush(ssp) ;
    Write(ssp, 17, "#%+MOT5SPF0000 *\\");
    sleep(400) ;

    printf("\n") ;
    printf("Pallet loaded \n") ;
    MainInterface(ssp) ;
    w = 0 ;
}

}
    return err;
}


int RobotEmergencyStopLight(serial_struct* ssp)

/*
Docked at Lathe
Prompts user for route by Interface
*/

{
    aErr err = aErrNone;
    RobotChill(ssp);

    printf("\n0.5 m in front of Light.\n") ;
    printf("At Lathe. Begin manual transfer. \n") ;
```

```
    MainInterface(ssp) ;
    return err;
}
```

LightFusion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightFusion.h"
#include "LightDock.h"
#include "LineDock.h"

/* ForwardLight */

int ForwardLight(serial_struct* ssp)

//dock RM 8

{
    Flush(ssp) ;
    aErr err = aErrNone ;
    extern int sw ;

    Write(ssp, 17, "#%+LITER       *\\");
    sleep(400) ;
    err = Read(ssp, 50);

    //r2

    char f21 = ssp->pinputbuff[32] ;
    char f22 = ssp->pinputbuff[33] ;

    //r3

    char f31 = ssp->pinputbuff[44] ;
    char f32 = ssp->pinputbuff[45] ;

    printf("\nf21 = %c \n" , f21);
    printf("f22 = %c \n" , f22);
    printf("\nf31 = %c \n" , f31);
    printf("f32 = %c \n" , f32);

     sleep(400) ;

     //Ultrasonic Process Forward

    Flush(ssp) ;
```

```
    Write(ssp, 17, "#%+ULTSS       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Forward     left      m1  2       3   m2       right
//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

Forward(ssp) ;

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

Right(ssp) ;

//More light toward r22 left
```

```
else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Left(ssp) ;

else if ( (f21 == '1') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp) ;

else if ( (f21 == '1' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Left(ssp) ;

else if ( (f21 == '0') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Right(ssp) ;

//No light

else if ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//Limit Switch
//6 7

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' )  && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
```

```
    sw = 0;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

else if ( ( (f21 == '1') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}


else if ( ( (f21 == '1' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}


else if ( ( (f21 == '0') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )
```

```
{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    sw = 0;
}
//8

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s8 == 'C') && ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s8 == 'C') && ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s8 == 'C')&& ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    sw = 0;
}

else if ( ( (f21 == '1') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0')
|| (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0'
&& un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    sw = 0;
```

```
}

else if ( ( (f21 == '1' && f31 == '0') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') ||
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    sw = 0;
}

else if ( ( (f21 == '0') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0')
|| (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0'
&& un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
   RMChill(ssp);
    sw = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') ) && (s8 == 'C') && ( (un1 >=
'1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2
> '0' && un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2
== '0') ) )

{
    printf("/nLight search to be continued/n") ;
    RMChill(ssp);
    sw = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right
```

```
else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 == '1') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 == '1' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 == '0') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//No light
```

```
else if ( (f21 == '0' && f22 <= '8' && f21 == '0' && f31 == '0') && (f31 == '0' && f32 <= '8') &&
( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0'
&& un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

return err;

}
/* ReverseLight */

int ReverseLight(serial_struct* ssp)

//dock conveyor 9

{
   Flush(ssp) ;
   aErr err = aErrNone ;
   extern int sw ;

   Write(ssp, 17, "#%+LITER       *\\\"");              //LITEL
   sleep(400) ;
   err = Read(ssp, 50);

   //Reverse
   //b2

   char b21 = ssp->pinputbuff[32] ;
   char b22 = ssp->pinputbuff[33] ;

   //r3

   char b31 = ssp->pinputbuff[44] ;
   char b32 = ssp->pinputbuff[45] ;


   printf("\nb21 = %c \n" , b21);
   printf("b22 = %c \n" , b22);

   printf("\nb31 = %c \n" , b31);
   printf("b32 = %c \n" , b32);

   sleep(400) ;

    //Ultrasonic Process Reverse

   Flush(ssp) ;
```

```
    Write(ssp, 17, "#%+ULTSS       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis       9 = Light

//Reverse           left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0') )

Reverse(ssp) ;

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0') )

Reverseright(ssp) ;
```

```
//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0') )

Reverseleft(ssp) ;

else if ( (b21 == '1') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverse(ssp) ;

else if ( (b21 == '1' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseleft(ssp) ;

else if ( (b21 == '0') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseright(ssp) ;

//No light

else if ( ( (b21 == '0' && b22 <= '8' && b31 == '0' && b32 <= '8') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0') )

{
    Reverseright(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

/*Limit fusion
//Following Light & switch closes
6 7*/

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
```

```
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}


//50cm from light hit something

else if ( (b21 == '1') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

else if ( (b21 == '1' && b31 == '0')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' &&
us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1
== '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

else if ( (b21 == '0') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//no light
```

```
else if ( ( (b21 == '0' && b22 <= '8') && (b31 == '0' && b32 <= '8') ) && (s6 == 'C' || s7 == 'C' )
&& ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 ==
'0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0'
&& us2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    sw = 0;
}

//9
//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C' ) && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )

{
    ConveyorChill(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (s9 == 'C' ) && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )

{
    ConveyorChill(ssp);
    sw = 0;
}

else if ( (b21 == '1') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >=
'0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0'
&& us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )
```

```
{
    ConveyorChill(ssp);
    sw = 0;
}

else if ( (b21 == '1' && b31 == '0')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    sw = 0;
}

else if ( (b21 == '0') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    sw = 0;
}

//No light

else if ( ( (b21 == '0' && b22 <= '8') && (b31 == '0' && b32 <= '8') ) && (s9 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    ConveyorChill(ssp);
    sw = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear
//Equal light

if ( ( b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right
```

```
else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '1') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '1' && b31 == '0')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '0') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light
```

```
else if ( ((b21 == '0') && (b22 <= '8')) && ((b31 == '0') && (b32 <= '8')) && ( (us1 == '0' && us2
== '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 ==
'0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

return err;

}

/* SideRightLight */

int SideRightLight(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone ;
   extern int sw ;

   Write(ssp, 17, "#%+LITEL       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

    //Sideright
   //r0

   char r01 = ssp->pinputbuff[8] ;
   char r02 = ssp->pinputbuff[9] ;

   //r1

   char r11 = ssp->pinputbuff[20] ;
   char r12 = ssp->pinputbuff[21] ;

   printf("\nr01 = %c \n" , r01);
   printf("r02 = %c \n" , r02);
   printf("\nr11 = %c \n" , r11);
   printf("r12 = %c \n" , r12);

    sleep(400) ;

   //Ultrasonic Process SideRight

   Flush(ssp) ;

   Write(ssp, 17, "#%+ULTSW       *\\");                    //E
   sleep(400) ;
   err = Read(ssp, 17);
```

```
    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("\nue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s0 = ssp->pinputbuff[9] ;
    char s1 = ssp->pinputbuff[10] ;
    char s2 = ssp->pinputbuff[11] ;
    char s3 = ssp->pinputbuff[12] ;
    char s4 = ssp->pinputbuff[13] ;
    char s5 = ssp->pinputbuff[14] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns0 = %c \n" , s0);
    printf("s1 = %c \n" , s1);
    printf("s2 = %c \n" , s2);
    printf("s3 = %c \n" , s3);
    printf("s4 = %c \n" , s4);
    printf("s5 = %c \n" , s5);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis       9 = Light

//Sideright    315      m2 r0     r1 m3       45

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3
>= '0'))
```

```
Sideright(ssp) ;

//More light toward r12 right

else if ((r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright45(ssp) ;

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright315(ssp) ;

else if ( (r01 == '1' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
   Sideright(ssp) ;
   printf("\nApprox 50 cm in front of Light.\n") ;

}

else if ( (r01 == '1' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Sideright315(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;

}

else if ( (r01 == '0' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
     Sideright45(ssp) ;
     printf("\nApprox 50 cm in front of Light.\n") ;

}
// 1000 chill or all 0 no light

else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//Lmt
```

```
//6 7

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}
```

```
else if ( (r01 == '1' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

// No light

else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    sw = 0;
}

//0

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    sw = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    TableSequenceLight(ssp);
    sw = 0;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    sw = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    sw = 0;
}

else if ( (r01 == '1' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    sw = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    sw = 0;
}

// no light

else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    printf("/nLight search to be continued/n") ;
    TableSequenceLight(ssp);
    sw = 0;
}

//Obstacle detected. Waiting for clear
//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// no light

else if ( ( (r01 == '0') && (r02 <= '8')) && ((r11 == '0') && (r12 <= '8')) && ( (ue1 == '0' && ue2
== '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 ==
'0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (r01 == '1' && r11 == '0')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}



else if ( (r01 == '0' && r11 == '1') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

return err;

}

//test

int SideLeftLight(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone ;
   extern int sw ;

   Write(ssp, 17, "#%+LITER       *\\\"");              //LITEL
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideleft
```

```
//l0

char l01 = ssp->pinputbuff[8] ;
char l02 = ssp->pinputbuff[9] ;

//l1

char l11 = ssp->pinputbuff[20] ;
char l12 = ssp->pinputbuff[21] ;


printf("\nl01 = %c \n" , l01);
printf("l02 = %c \n" , l02);
printf("\nl11 = %c \n" , l11);
printf("l12 = %c \n" , l12);

 sleep(400) ;

 //Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");                //W
sleep(400) ;
err = Read(ssp, 17);

char uw1 = ssp->pinputbuff[10] ;
char uw2 = ssp->pinputbuff[11] ;
char uw3 = ssp->pinputbuff[12] ;

printf("\nuw1 = %c \n" , uw1);
printf("uw2 = %c \n" , uw2);
printf("uw3 = %c \n" , uw3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s0 = ssp->pinputbuff[9] ;
char s1 = ssp->pinputbuff[10] ;
char s2 = ssp->pinputbuff[11] ;
char s3 = ssp->pinputbuff[12] ;
char s4 = ssp->pinputbuff[13] ;
```

```
    char s5 = ssp->pinputbuff[14] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns0 = %c \n" , s0);
    printf("s1 = %c \n" , s1);
    printf("s2 = %c \n" , s2);
    printf("s3 = %c \n" , s3);
    printf("s4 = %c \n" , s4);
    printf("s5 = %c \n" , s5);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis        9 = Light

//Sideleft     225  m1 l1       l0 m4     135

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') )

Sideleft(ssp) ;

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') )

Sideleft225(ssp) ;

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') )

Sideleft135(ssp) ;

//50cm from light

else if ( (l01 == '1' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}
```

```
else if ( (l01 == '1' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft135(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (l01 == '0' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft225(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}


//No light

else if ( ( (l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8') ) && (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') )

{
    Reverseright(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//Lmt
//6 7

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 ==
'0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 >
'0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
```

```
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' )&& (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//No light

else if ( ( (l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    sw = 0;
}

else if ( (l01 == '1' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

else if ( (l01 == '1' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

else if ( (l01 == '0' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
```

```
}

//1

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    sw = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C' ) && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    sw = 0;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    sw = 0;
}

else if ( (l01 == '1' || l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    sw = 0;
}

//more light l01 135
```

```
else if ( (l01 == '1' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    sw = 0;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    sw = 0;
}

//No light

else if ( ((l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8')) && (s1 == 'C') && ( (uw1 >= '1'
&& uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2
> '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    printf("/nLight search to be continued/n") ;
    LatheChillLight(ssp);
    sw = 0;
}

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//light 50cm away

else if ( (l01 == '1' || l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l01 135

else if ( (l01 == '1' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light
```

```
else if ( ( (l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8') ) && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

return err;
}

int Lathe_RM_Light1(serial_struct* ssp)

{
   //Sideright

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LITEL       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r0

   char r01 = ssp->pinputbuff[8] ;
   char r02 = ssp->pinputbuff[9] ;
   //r1

   char r11 = ssp->pinputbuff[20] ;
   char r12 = ssp->pinputbuff[21] ;

   sleep(400) ;

   //Forward

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");                    //LITEL
   sleep(400) ;
   err = Read(ssp, 50);

   //f2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //f3
```

```
char f31 = ssp->pinputbuff[44] ;
char f32 = ssp->pinputbuff[45] ;

//display data

printf("\nr01 = %c \n" , r01);
printf("r02 = %c \n" , r02);

printf("\nr11 = %c \n" , r11);
printf("r12 = %c \n" , r12);

printf("\nf21 = %c \n" , f21);
printf("f22 = %c \n" , f22);

printf("\nf31 = %c \n" , f31);
printf("f32 = %c \n" , f32);

sleep(400) ;

 //Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");       //N
sleep(400) ;
err = Read(ssp, 17);

char un1 = ssp->pinputbuff[10] ;
char un2 = ssp->pinputbuff[11] ;
char un3 = ssp->pinputbuff[12] ;

printf("\nun1 = %c \n" , un1);
printf("un2 = %c \n" , un2);
printf("un3 = %c \n" , un3);

sleep(400) ;

//Ultrasonic Process SideRight

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //E
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
```

```
    printf("ue3 = %c \n" , ue3);

    sleep(400) ;


    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("\n");

     sleep(400) ;

     //Ultrasonic Process South

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSS       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

//Analysis       9 = Light

//Sideright          m2 r0     r1 m3

//Light from Sideright

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31 == '0'
&& f21 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Sideright(ssp) ;
```

```
//Light from Sideright315

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31 == '0' && f21 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Sideright315(ssp) ;

//Light from Sideright45

else if ( (r12 == '9' && r02 < '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31 == '0' && f21 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Sideright45(ssp) ;

//Light from Sideright315

else if ( (r02 == '9' && f32 == '9' && r01 == '0' && r11 == '0' && r12 < '9' && f22 < '9' && f31 == '0' && f21 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Sideright315(ssp) ;

//Light from Sideright315 new not tested

else if ( (r02 == '9' && r12 == '9' && f32 == '9' && f22 < '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Sideright315(ssp) ;

//Light from Sideright315

else if ( (r02 == '9' && r12 == '9' && f32 == '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Sideright315(ssp) ;

//Transition

//Light from Right new

else if ( (r02 == '9' && r12 < '9' && f32 == '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Right(ssp) ;
```

```
//Light from Forward

else if ( (f22 == '9' && f32 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' &&
r12 < '9' && r02 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' )
)

Forward(ssp) ;

//Light from Right

else if ( (f22 < '9' && f32 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' && r12
< '9' && r02 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Right(ssp) ;

//Light from Left

else if ( (f32 < '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' && r12
< '9' && r02 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Left(ssp) ;

//1000 - 30 cm in front of Light or really really close

else if (r01 == '1' || r11 == '1' || f21 == '1' || f31 == '1')

{
    RMChillLight(ssp) ;
    sw = 0;
}


//Lmt 6 7

//Sideright         m2 r0     r1 m3
//Light from Sideright

else if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31
== '0' && f21 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//Light from Sideright315

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31 ==
'0' && f21 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1
```

```
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )  && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//Light from Sideright45 south

else if ( (r12 == '9' && r02 < '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31 ==
'0' && f21 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) /*&& ( (us1 >= '1' && us2 >= '0' &&
us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1
== '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) */)
{
    EmergencyStopLight(ssp);
    sw = 0;
}

//Light from Sideright315

else if ( (r02 == '9' && f32 == '9' && r01 == '0' && r11 == '0' && r12 < '9' && f22 < '9' && f31
== '0' && f21 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//Light from Sideright315 new not tested

else if ( (r02 == '9' && r12 == '9' && f32 == '9' && f22 < '9' && r01 == '0' && r11 == '0' && f31
== '0' && f21 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}
```

```
//Light from Sideright315

else if ( (r02 == '9' && r12 == '9' && f32 == '9' && f22 == '9' && r01 == '0' && r11 == '0' &&
f31 == '0' && f21 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >=
'0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0'
&& ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2
>= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&&
un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//Transition

//Light from Right new

else if ( (r02 == '9' && r12 < '9' && f32 == '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31
== '0' && f21 == '0' ) && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
{
    EmergencyStopLight(ssp);
    sw = 0;
}


//Light from Forward

else if ( (f22 == '9' && f32 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' &&
r12 < '9' && r02 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//Light from Right

else if ( (f22 < '9' && f32 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' && r12
< '9' && r02 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
```

```
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//Light from Left

else if ( (f32 < '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' && r12 < '9' && r02 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    sw = 0;
}

//1000 - 30 cm in front of Light or really really close

else if ( ( r01 == '1' || r11 == '1' || f21 == '1' || f31 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    printf("\n30 cm in front of Light or really really close.\n") ;
    EmergencyStopLight(ssp);
    sw = 0;
}

/* Ultrasonics
Obstacle detected. Waiting for clear */

//Light from Sideright
```

```
else if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31
== '0' && f21 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright315

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31 ==
'0' && f21 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3
> '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) || ( (un1
== '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2
> '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright45 south

else if ( (r12 == '9' && r02 < '9' && r01 == '0' && r11 == '0' && f32 < '9' && f22 < '9' && f31 ==
'0' && f21 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3
> '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) || ( (us1
== '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 >
'0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright315

else if ( (r02 == '9' && f32 == '9' && r01 == '0' && r11 == '0' && r12 < '9' && f22 < '9' && f31
== '0' && f21 == '0')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) || (
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && (s6 == 'O' && s7 == 'O' )
)

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright315 new not tested
```

```
else if ( (r02 == '9' && r12 == '9' && f32 == '9' && f22 < '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright315

else if ( (r02 == '9' && r12 == '9' && f32 == '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Transition

//Light from Right new

else if ( (r02 == '9' && r12 < '9' && f32 == '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' ) && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Forward

else if ( (f22 == '9' && f32 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' && r12 < '9' && r02 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//Light from Right

else if ( (f22 < '9' && f32 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' && r12
< '9' && r02 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&&
un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') )
&& (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//Light from Left

else if ( (f32 < '9' && f22 == '9' && r01 == '0' && r11 == '0' && f31 == '0' && f21 == '0' && r12
< '9' && r02 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&&
un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') )
&& (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//No Light Search for Light >:)

// No obstacle switch open


else if ( (r02 < '9' && r12 < '9' && f22 < '9' && f32 < '9'  && r01 == '0' && r11 == '0' && f21 ==
'0' && f31 == '0' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

{
    Right(ssp) ;
    printf("\nSearching for Light >:) \n") ;
}

//No obstacle switch closed

else if ( (r02 < '9' && r12 < '9' && f22 < '9' && f32 < '9'  && r01 == '0' && r11 == '0' && f21 ==
'0' && f31 == '0' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') && (s6 == 'C' || s7 == 'C' ) )

{
    printf("No Light \n") ;
    EmergencyStopLight(ssp);
    sw = 0;
}
```

```
//Obstacle switch open - stop

else if ( (r02 < '9' && r12 < '9' && f22 < '9' && f32 < '9' && r01 == '0' && r11 == '0' && f21 ==
'0' && f31 == '0' ) && (s6 == 'O' && s7 == 'O' )  &&  ( ( (ue1 == '0' && ue2 == '0' && ue3 == '0')
|| (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' &&
ue3 > '0' && ue2 == '0') ) || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) ) )

{
    Chill(ssp);
    printf("No Light. Obstacle detected. Waiting for clear \n") ;
}

//Obstacle switch closed - stop

else if ( (r02 < '9' && r12 < '9' && f22 < '9' && f32 < '9'  && r01 == '0' && r11 == '0' && f21 ==
'0' && f31 == '0' ) && (s6 == 'C' || s7 == 'C' ) && ( ( (ue1 == '0' && ue2 == '0' && ue3 == '0') ||
(ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' &&
ue3 > '0' && ue2 == '0') ) || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) ) )

{
    printf("No Light. Obstacle detected. \n") ;
    EmergencyStopLight(ssp);
    sw = 0;
}

return err;

}

int RM_Lathe_Light1(serial_struct* ssp)

{
   //Sideleft

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LITEL       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

    //l0

   char l01 = ssp->pinputbuff[8] ;
   char l02 = ssp->pinputbuff[9] ;
```

```
//l1

char l11 = ssp->pinputbuff[20] ;
char l12 = ssp->pinputbuff[21] ;

sleep(400) ;

//Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+LITER       *\\");
sleep(400) ;
err = Read(ssp, 50);

//b2

char b21 = ssp->pinputbuff[32] ;
char b22 = ssp->pinputbuff[33] ;

//b3

char b31 = ssp->pinputbuff[44] ;
char b32 = ssp->pinputbuff[45] ;

//display data

printf("\nl01 = %c \n" , l01);
printf("l02 = %c \n" , l02);

printf("\nl11 = %c \n" , l11);
printf("l12 = %c \n" , l12);

printf("\nb21 = %c \n" , b21);
printf("b22 = %c \n" , b22);

printf("\nb31 = %c \n" , b31);
printf("b32 = %c \n" , b32);

sleep(400) ;
//Ultrasonic Process Sideleft

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");
sleep(400) ;
err = Read(ssp, 17);

char uw1 = ssp->pinputbuff[10] ;
char uw2 = ssp->pinputbuff[11] ;
char uw3 = ssp->pinputbuff[12] ;
```

```
printf("\nuw1 = %c \n" , uw1);
printf("uw2 = %c \n" , uw2);
printf("uw3 = %c \n" , uw3);

sleep(400) ;

//Ultrasonic Process Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");
sleep(400) ;
err = Read(ssp, 17);

char us1 = ssp->pinputbuff[10] ;
char us2 = ssp->pinputbuff[11] ;
char us3 = ssp->pinputbuff[12] ;

printf("\nus1 = %c \n" , us1);
printf("us2 = %c \n" , us2);
printf("us3 = %c \n" , us3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;

printf("\ns6 = %c \n" , s6);
printf("s7 = %c \n" , s7);

printf("\n");

sleep(400) ;

//Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSN       *\\");
sleep(400) ;
err = Read(ssp, 17);
```

```
    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

    sleep(400) ;

//Analysis        9 = Light
//Free movement

//Sideleft

//Light from Sideleft

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22 <
'9' && b32 < '9' ) && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Sideleft(ssp) ;

//Light from Sideleft135

else if ( ( l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22
< '9' && b32 < '9' ) && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') /*&& (us1 >= '1' && us2 >=
'0' && us3 >= '0')*/  && (s6 == 'O' && s7 == 'O' ) )

Sideleft135(ssp) ;

//Light from Sideleft225 North

else if ( (l12 == '9' && l02 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22
< '9' && b32 < '9' ) && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') /* && (un1 >= '1' && un2 >=
'0' && un3 >= '0') */ && (s6 == 'O' && s7 == 'O' ) )

Sideleft225(ssp) ;

//Light from Siderright135

else if ( (l02 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' &&
b22 < '9' && l12 < '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') /*&& (us1 >= '1' && us2
>= '0' && us3 >= '0')*/  && (s6 == 'O' && s7 == 'O' ) )

Sideleft135(ssp) ;

//Light from Siderright135 new
```

```
else if ( (l02 == '9' && l12 == '9' && b22 < '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') /*&& (us1 >= '1' && us2 >= '0' && us3 >= '0')*/  && (s6 == 'O' && s7 == 'O' ) )

Sideleft135(ssp) ;

//Light from Sideright135

else if ( (l02 == '9' && l12 == '9' && b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') /*&& (us1 >= '1' && us2 >= '0' && us3 >= '0')*/  && (s6 == 'O' && s7 == 'O' ) )

Sideleft135(ssp) ;

//Transition

//Light from Reverseright new

else if ( (l02 == '9' && l12 < '9' && b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Reverseright(ssp) ;

//Light from Reverse

else if ( (b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02 < '9' && l12 < '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Reverse(ssp) ;

//Light from Right

else if ( (b22 < '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02 < '9' && l12 < '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0')  && (s6 == 'O' && s7 == 'O' ) )

Reverseright(ssp) ;

//Light from Left

else if ( (b22 == '9' && b32 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02 < '9' && l12 < '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

Reverseleft(ssp) ;

//if going to dock have to take this out
//1000 - 30 cm in front of Light or really really close

else if (l01 == '1' || l11 == '1' || b21 == '1' || b31 == '1')

{
    LatheChill(ssp) ;
```

```
    sw = 0;
}

//Lmt 6 7

//Sideleft

//Light from Sideleft

else if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' &&
b22 < '9' && b32 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >=
'0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1
== '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Sideleft135

else if ( ( l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22
< '9' && b32 < '9' ) && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Sideleft225

else if ( (l12 == '9' && l02 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22
< '9' && b32 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Sideright135

else if ( (l02 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' &&
b22 < '9' && l12 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
```

```
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Sideright135 new

else if ( (l02 == '9' && l12 == '9' && b22 < '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21
== '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Sideright135

else if ( (l02 == '9' && l12 == '9' && b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' &&
b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >=
'0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1
== '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Reverseright new

else if ( (l02 == '9' && l12 < '9' && b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21
== '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )
```

```
{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Reverse

else if ( (b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' &&
l02 < '9' && l12 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Right

else if ( (b22 < '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02
< '9' && l12 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}

//Light from Left

else if ( (b22 == '9' && b32 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02
< '9' && l12 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    sw = 0;
}
//if going to dock have to take this out
//1000 - 30 cm in front of Light or really really close
```

```
else if ( (l01 == '1' || l11 == '1' || b21 == '1' || b31 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) &&  ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    printf("\n30 cm in front of Light or really really close.\n") ;
    sw = 0;
}

//Obstacle detected. Waiting for clear

//Sideleft

//Light from Sideleft

else if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22 < '9' && b32 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideleft135

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22 < '9' && b32 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') && (s6 == 'O' && s7 == 'O' ) ) /*|| ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') )*/ )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideleft225

else if ( (l12 == '9' && l02 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && b22 < '9' && b32 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) && (s6 == 'O' && s7 == 'O' ) /*|| ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') )*/ )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright135

else if ( (l02 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' &&
b22 < '9' && l12 < '9')&& ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) && (s6 == 'O' && s7 == 'O' ) /*|| ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') )*/ )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright135 new

else if ( (l02 == '9' && l12 == '9' && b22 < '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21
== '0' && b31 == '0')&& ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) && (s6 == 'O' && s7 == 'O' ) /* || ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') )*/ )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Sideright135

else if ( (l02 == '9' && l12 == '9' && b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' &&
b21 == '0' && b31 == '0')&& ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2
> '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) && (s6 == 'O' && s7 == 'O' ) /*|| ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') )*/ )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Transition

//Light from Reverseright new
```

```
else if ( (l02 == '9' && l12 < '9' && b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Reverse

else if ( (b22 == '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02 < '9' && l12 < '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') )&& (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Right

else if ( (b22 < '9' && b32 == '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02 < '9' && l12 < '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') )&& (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Light from Left

else if ( (b22 == '9' && b32 < '9' && l01 == '0' && l11 == '0' && b21 == '0' && b31 == '0' && l02 < '9' && l12 < '9')&& ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) && (s6 == 'O' && s7 == 'O' ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No Light Search for Light >:)
// No obstacle switch open
```

```
else if ( (l02 < '9' && l12 < '9' && b22 < '9' && b32 < '9' && l01 == '0' && l11 == '0' && b21 ==
'0' && b31 == '0' ) && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') && (us1 >= '1' && us2 >= '0'
&& us3 >= '0') && (s6 == 'O' && s7 == 'O' ) )

{
    Right(ssp) ;
    printf("\nSearching for Light >:) \n") ;
}

//No obstacle switch closed

else if ( (l02 < '9' && l12 < '9' && b22 < '9' && b32 < '9' && l01 == '0' && l11 == '0' && b21 ==
'0' && b31 == '0' ) && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') && (us1 >= '1' && us2 >= '0'
&& us3 >= '0') && (s6 == 'C' || s7 == 'C' ) )

{
    printf("No Light \n") ;
    EmergencyStop(ssp);
    sw = 0;
}


//Obstacle switch open - stop

else if ( (l02 < '9' && l12 < '9' && b22 < '9' && b32 < '9' && l01 == '0' && l11 == '0' && b21 ==
'0' && b31 == '0' ) && (s6 == 'O' && s7 == 'O' ) &&  ( ( (uw1 == '0' && uw2 == '0' && uw3 ==
'0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1
== '0' && uw3 > '0' && uw2 == '0') ) || ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) ) )

{
    Chill(ssp);
    printf("No Light. Obstacle detected. Waiting for clear \n") ;
}

//Obstacle switch closed - stop

else if ( (l02 < '9' && l12 < '9' && b22 < '9' && b32 < '9' && l01 == '0' && l11 == '0' && b21 ==
'0' && b31 == '0' ) && (s6 == 'C' || s7 == 'C' ) &&  ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) || ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')
) )

{
    printf("No Light. Obstacle detected. \n") ;
    EmergencyStop(ssp);
    sw = 0;
}
return err;
```

}

1.5 Line Variation Technique

Line Variation Interface.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "LineVariationInterface.h"
#include "LineLightSimConveyorRMForwardFusion.h"
#include "LineLightSimLatheConveyorSiderightFusion.h"
#include "LineLightSimLatheRMSiderightFusion.h"
#include "LineLightSimRMConveyorReverseFusion.h"
#include "LineLightSimRMLatheReverseFusion.h"
#include "LineLightSimRMLatheSideleftFusion.h"
#include "MainInterface.h"
#include "LineConfirm.h"

//#include "Directions.h"

int LineVariationInterface(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone;

   int route ;

   printf("\nPlease select a Line Variation route: \n\n"  ) ;

   printf("Conveyor to Lathe \t 1 \nConveyor to RM \t\t 2 \n\nLathe to Conveyor \t 3 \nLathe to RM
\t\t 4 \n \nRM to Lathe \t\t 5 \nRM to Conveyor \t\t 6 \nExit to Main \t\t 7 \n\n") ;
   scanf("%d" , &route) ;

   if (route == 1)
   {
   printf("\nInitialising conveyor to lathe sequence . . . \n\n\n\n") ;
   LineLightSimRMLatheSideleftFusion(ssp) ;
   }

   else if (route == 2)
   {
   printf("\nInitialising conveyor to RM sequence . . . \n\n\n\n") ;
   LineLightSimConveyorRMForwardFusion(ssp) ;
   }

   else if (route == 3)
   {
   LineConfirmLineLightSimLatheConveyorSiderightFusion(ssp) ;
```

```
    }

    else if (route == 4)
    {
    LineConfirmLineLightSimLatheRMSiderightFusion(ssp) ;
    }

    else if (route == 5)
    {
    LineConfirmLineLightSimRMLatheReverseFusion(ssp) ;
    }

    else if (route == 6)
    {
    LineConfirmLineLightSimRMConveyorReverseFusion(ssp) ;
    }

    else if (route == 7)
    {
    MainInterface(ssp) ;
    }

    else printf("\nUnknown route. Please enter new route\n") ;

return err;

}
```

Line Confirm.c

```
#define aWIN
#include "Access.h"
#include "LineConfirm.h"
#include "LineLightSimRMLatheReverseFusion.h"
#include "LineLightSimLatheRMSiderightFusion.h"
#include "LineLightSimLatheConveyorSiderightFusion.h"
#include "LineLightSimRMConveyorReverseFusion.h"

//for each route

//lathe - RM
int LineConfirmLineLightSimLatheRMSiderightFusion(serial_struct* ssp)

{

    aErr err = aErrNone;

    int c = 2;

    printf("\nLine Variation - Lathe to RM route\n\n") ;
```

```
   while (c == 2)
   {
   printf("\nPlease confirm docking station under conveyor is at 0.1m mark\n\n");
   printf("\nYes                1");
   printf("\nNo                 2\n");
   printf("\nExit to Main       3\n");

   scanf("%d", &c) ;

   //Load sequence

   if (c == 1)

   {
   printf("\nInitialising lathe to RM sequence . . . \n\n\n\n") ;
   LineLightSimLatheRMSiderightFusion(ssp);
   }

   //Load Main

   else if (c == 3)
   {
   MainInterface(ssp) ;
   }

  // Unknown selection. Made c = 2

   else if (c > 3 || c <= 0)
   {
   printf("\nUnknown selection.\n");
   c = 2 ;
   }

}
  return err;
}

//RM - lathe

int LineConfirmLineLightSimRMLatheReverseFusion(serial_struct* ssp)

{

   aErr err = aErrNone;
   int c = 2;

   printf("\nLine Variation - RM to Lathe route\n\n") ;

   while (c == 2)
   {
   printf("\nPlease confirm docking station at end of reverse track is at 3m mark\n\n");
```

```
   printf("\nYes                1");
   printf("\nNo                 2\n");
   printf("\nExit to Main       3\n");

   scanf("%d", &c) ;

   //Load sequence

   if (c == 1)

   {
   printf("\nInitialising RM to lathe sequence . . . \n\n\n\n") ;
   LineLightSimRMLatheReverseFusion(ssp) ;
   }

   //Load Main

   else if (c == 3)
   {
   MainInterface(ssp) ;
   }

  // Unknown selection. Made c = 2

   else if (c > 3 || c <= 0)
   {
   printf("\nUnknown selection.\n");
   c = 2 ;
   }

}
   return err;

}

//lathe - conveyor - high
//necessary because AGV has to dock at correct distance (adjacent) to conveyor

int LineConfirmLineLightSimLatheConveyorSiderightFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
   int c = 2;

   printf("\nLine Variation - Lathe to Conveyor route\n\n") ;

   while (c == 2)
   {
   printf("\nPlease confirm docking station under conveyor is at 0m mark\n\n");
   printf("\nYes                1");
   printf("\nNo                 2\n");
```

```
    printf("\nExit to Main        3\n");

    scanf("%d", &c) ;

    //Load sequence

    if (c == 1)

    {
    printf("\nInitialising lathe to conveyor sequence . . . \n\n\n\n") ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
    }

    //Load Main

    else if (c == 3)
    {
    MainInterface(ssp) ;
    }

   // Unknown selection. Made c = 2

    else if (c > 3 || c <= 0)
    {
    printf("\nUnknown selection.\n");
    c = 2 ;
    }
}

  return err;
}

//Line Variation - RM to Conveyor

int LineConfirmLineLightSimRMConveyorReverseFusion(serial_struct* ssp)

{
    aErr err = aErrNone;
    int c = 2;
    printf("\nLine Variation - RM to Conveyor route\n\n") ;

    while (c == 2)
    {
    printf("\nPlease confirm docking station at the end of reverse track is at 2.5m mark\n\n");
    printf("\nYes                 1");
    printf("\nNo                  2\n");
    printf("\nExit to Main        3\n");

    scanf("%d", &c) ;

    //Load sequence
```

```
   if (c == 1)

   {
   printf("\nInitialising RM to conveyor sequence . . . \n\n\n\n") ;
   LineLightSimRMConveyorReverseFusion(ssp);
   }

   //Load Main

   else if (c == 3)
   {
   MainInterface(ssp) ;
   }

  // Unknown selection. Made c = 2

   else if (c > 3 || c <= 0)
   {
   printf("\nUnknown selection.\n");
   c = 2 ;
   }

}
  return err;
}
```

LineLightSimConveyorRMForwardFusion.c

```
//follow line
//no line - light follow
//fwd dir only

// Foward Analysis                              front

/*                                          f1
                                            f0
                                            f2 */

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LineLightSimConveyorRMForwardFusion.h"

int LineLightSimConveyorRMForwardFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
```

```
int c = 1;

while(c==1)
{

//Foward                          LINER

 Flush(ssp) ;

 Write(ssp, 17, "#%+LINER       *\\");
 sleep(400) ;

 err = Read(ssp, 50);

char f0 = ssp->pinputbuff[9] ;
char f1 = ssp->pinputbuff[21] ;
char f2 = ssp->pinputbuff[33] ;

printf("\nf0 = %c \n" , f0);
printf("f1 = %c \n" , f1);
printf("f2 = %c \n" , f2);

sleep(400) ;

  //Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("\ns6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");
```

```
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

//normal foward
//f0 on black f1 f2 go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0')  )

Forward(ssp);

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Right(ssp) ;

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Right(ssp) ;

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Left(ssp) ;

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Left(ssp);

//No line  - light follow

//All on black

else if (f1 < '9' && f0 < '9' && f2 < '9')
```

```
{
    Chill(ssp) ;
    LineOwnLightSimConveyorRMForwardFusion(ssp) ;
    c = 0 ;
}

//All on white

else if (f1 == '9' && f0 == '9' && f2 == '9')

{
    Chill(ssp) ;
    LineOwnLightSimConveyorRMForwardFusion(ssp) ;
    c = 0 ;
}

//Lmt switch

//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7 == 'C') )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    c = 0;
}

//l2 l0 l1 on white

else if ((f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ))

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    c = 0;
}

//Offline and obstacle crash

//All on black

else if (  (f1 < '9' && f0 < '9' && f2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
```

```
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//On line obstacle detected waiting for clear

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' ) && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More Chills
//s6 s7

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >= '0'
&& un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0')
|| (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
    c = 0;
}
```

```
//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp) ;
    c = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp) ;
    c = 0;
}

//Docking at RM switch 8

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >=
'0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 ==
'0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
```

```
    RMChill(ssp);
    c = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

else if ( (f1 < '9' && f0 < '9' && f2 < '9')&& (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9')&& (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    RMChill(ssp);
    c = 0;
}
}

return err;

}
```

LineLightSimLatheConveyorSiderightFusion.c

```
//follow line
//no line - light follow
//sideright dir only

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LineLightSimLatheConveyorSiderightFusion.h"

int LineLightSimLatheConveyorSiderightFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   //Sideright                          LINER

    Flush(ssp) ;

    Write(ssp, 17, "#%+LINEL       *\\");
    sleep(400) ;

    err = Read(ssp, 50);

   char r0 = ssp->pinputbuff[9] ;
   char r1 = ssp->pinputbuff[21] ;
   char r2 = ssp->pinputbuff[33] ;

   printf("\nr0 = %c \n" , r0);
   printf("r1 = %c \n" , r1);
   printf("r2 = %c \n" , r2);

   sleep(500) ;
```

```
    //Ultrasonic Process SideRight

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\\");                    //E
    sleep(400) ;
    err = Read(ssp, 17);

    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("\nue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s0 = ssp->pinputbuff[9] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;

    printf("\ns0 = %c \n" , s0);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("\n");

//Line Analysis
//r0 on black r1 r2 on white && u >= 1 go Sideright              m2  r2 r0 r1

if ( (r0 < '9') && (r1 == '9') && (r2 == '9') && ue1 >= '1' && ue2 >= '0' && ue3 >= '0')

Sideright(ssp);

//r1 on black r0 r2 on white && u >= 1 go Sideright45

else if ( (r1 < '9') && (r0 == '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright45(ssp) ;

//r2 on black r0 r1 on white && u >= 1 go Sideright315
```

```
else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright315(ssp) ;

//r0/r1 on black r2 on white && u >= 1 go Sideright315

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//r0/r2 on black r1 on white && u >= 1 go Sideright315

else if ( ( r0 < '9' && r1 == '9'&& r2 < '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp);

//no line - light follow

//l2 l0 l1 on black

else if ( r1 < '9' && r0 < '9' && r2 < '9')

{
    Chill(ssp) ;
    LineOwnLightSimLatheConveyorSiderightFusion(ssp) ;
    a = 0 ;
}

//l2 l0 l1 on white
else if ( r1 == '9' && r0 == '9' && r2 == '9')

{
    Chill(ssp) ;
    LineOwnLightSimLatheConveyorSiderightFusion(ssp) ;
    a = 0 ;
}

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (s6 == 'C' || s7 == 'C'))

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' ))
```

```
{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//l2 l0 l1 on black
else if ( (r1 < '9' && r0 < '9' && r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//l2 l0 l1 on white
else if ( (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//l0 on black l1 l2 on
else if ( (r0 < '9' && r1 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9')&& ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9'&& r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Lmt integration s0

//l0 on black l1 l2 on white go fwd

if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3 >=
'0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0'
&& ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right
```

```
else if ( (s0 == 'C') && (r1 < '9' && r0 == '9'&& r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (s0 == 'C') && (r2 < '9' && r0 == '9' && r1 == '9')&& ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (s0 == 'C')&& (r0 < '9' && r1 < '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 < '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//More Chills
//s6 s7

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
```

```
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

}

return err;
```

}

LineLightSimLatheRMSiderightFusion.c

```
//line follow
//no line - light follow
//sr dir
//sense foward line - go to fwd

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LineLightSimLatheRMSiderightFusion.h"

int LineLightSimLatheRMSiderightFusion(serial_struct* ssp)

//going sideright sees black line starts going foward

/*                              E
                              r2 r0 r1

                          f1
                    N     f0
                          f2


  */

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   //1st board going right              LINEL

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LINEL       *\\");
   sleep(400) ;

   err = Read(ssp, 50);

   char r0 = ssp->pinputbuff[9] ;
   char r1 = ssp->pinputbuff[21] ;
   char r2 = ssp->pinputbuff[33] ;
```

```
printf("\nr0 = %c \n" , r0);
printf("r1 = %c \n" , r1);
printf("r2 = %c \n" , r2);

sleep(500) ;

 //2nd board foward                          LINER

 Flush(ssp) ;

 Write(ssp, 17, "#%+LINER       *\\");             //change to LITEL
 sleep(400) ;

 err = Read(ssp, 50);

char f0 = ssp->pinputbuff[9] ;
char f1 = ssp->pinputbuff[21] ;
char f2 = ssp->pinputbuff[33] ;

printf("\nf0 = %c \n" , f0);
printf("f1 = %c \n" , f1);
printf("f2 = %c \n" , f2);

sleep(400) ;

 //Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process SideRight

Flush(ssp) ;
```

```
    Write(ssp, 17, "#%+ULTSW       *\\");          //east
    sleep(400) ;
    err = Read(ssp, 17);

    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("\nue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSS       *\\");              //N
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

//Sideright Analysis                        m2  r2 r0 r1

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright(ssp);

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp) ;
```

```
//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright45(ssp) ;

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright315(ssp);

// Foward Analysis                              front

/*                                              f1
                                                f0
                                                f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0')  )

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
```

```
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Right(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}
```

```
//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') )

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Right(ssp) ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9'  && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))
```

```
{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r1 < '9'  && r0 == '9' && r2 == '9'&& f2 == '9' && f0 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Left(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}
```

```
//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f0-f2 on black f1 on white go Left
```

```
else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
{
    Left(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
```

```
    a = 0;
}
//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
```

```
{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9'  && r2 < '9' && r0 < '9' && r1 == '9' && f2 == '9' && f1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//no line - light follow sr dir
//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9')) /*|| (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') )*/

{
    Chill(ssp) ;
    //LineOwnLightSimLatheRMSiderightFusion(ssp);
    a = 0 ;
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9')) /*|| (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') )*/

{
    Chill(ssp) ;
    LineOwnLightSimLatheRMSiderightFusion(ssp);
    a = 0 ;
}

//f2 f0 f1 on black r1 r2 r0 on white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9')) /* || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 ==
```

```
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') )*/
{
    Chill(ssp) ;
    LineOwnLightSimLatheRMSiderightFusion(ssp);
    a = 0 ;
}

//f2 f0 f1 on white r1 r2 r0 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9')) /* || (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') )*/

{
    Chill(ssp) ;
   LineOwnLightSimLatheRMSiderightFusion(ssp);
    a = 0 ;
}

//Lmt switch

//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7
== 'C'|| s8 == 'C'|| s9 == 'C') )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 ==
'C' || s7 == 'C' || s8 == 'C' || s9 == 'C') )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white
```

```
else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9')&& (s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C') )
{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white
```

```
else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315
```

```
else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9') && (r1 == '9')&& (r2 < '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// Foward Analysis                                front

/*                                                f1
                                                  f0
                                                  f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}
```

```
//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
```

```
    a = 0;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r1 on black f2 r2 on white go Forward
```

```
else if ( (f1 < '9' && r1 < '9'  && r0 == '9' && r2 == '9' && f2 == '9' && f0 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}
```

```
//f2 r0-r1 r0-r2 combinations
//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
```

```
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r1 r0-r2 combinations
//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9' && r2 < '9' && r0 < '9' && r1 == '9'&& f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//More Chills
//s6 s7

if ( ((f0 < '9' && f1 == '9' && f2 == '9') || (r0 < '9' && r1 == '9' && r2 == '9')) && (s6 == 'C' || s7
== 'C'  ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0')
```

```
|| (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( ((f1 < '9' && f0 == '9' && f2 == '9') || (r1 < '9' && r0 == '9' && r2 == '9')) && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( ((f2 < '9' && f0 == '9' && f1 == '9')|| (r2 < '9' && r0 == '9' && r1 == '9')) && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( ((f0 < '9' && f1 < '9' && f2 == '9') || (r0 < '9' && r1 < '9' && r2 == '9')) && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( ( (f0 < '9' && f1 == '9' && f2 < '9') || (r0 < '9' && r1 == '9' && r2 < '9') ) && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    EmergencyStop(ssp) ;
    a = 0;
}

}

return err;

}
```

LineLightSimRMConveyorReverseFusion.c

```
//follow line

//no line - follow light Reverse dir
//dock conveyor 9

//chked

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LineLightSimRMConveyorReverseFusion.h"

//ReverseSideLeft

int LineLightSimRMConveyorReverseFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
   int c = 1;

   while(c==1)
   {

   Flush(ssp) ;

   //Reverse

   Write(ssp, 17, "#%+LINER       *\\\");          //change to LITER
   sleep(400) ;

   err = Read(ssp, 50);

   char b0 = ssp->pinputbuff[9] ;
   char b1 = ssp->pinputbuff[21] ;
   char b2 = ssp->pinputbuff[33] ;
```

```
    printf("b0 = %c \n" , b0);
    printf("b1 = %c \n" , b1);
    printf("b2 = %c \n" , b2);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s1 = ssp->pinputbuff[17] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[10] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s1 = %c \n" , s1);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    sleep(400) ;

    //Ultrasonic Process Reverse

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");          //S
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("us1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

//Line following

//Reverse Analysis
```

```
//b0 on black b1 b2 on white go fwd

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverse(ssp);

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') &&  (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b2 on black b0 b1 on white go Left

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9')  && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp) ;

//b0/b1 on black b2 on white go right

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9')  && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b0/b2 on black b1 on white go Left

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9')  && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp);

//No line - light follow

//All on black

else if ( b1 < '9' && b0 < '9' && b2 < '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMConveyorReverseFusion(ssp) ;
    c = 0 ;
}

//All on white

else if ( b1 == '9' && b0 == '9' && b2 == '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMConveyorReverseFusion(ssp) ;
    c = 0 ;
```

```
}

//Lmt Ultra integration


//Offline and switch 6 7 closed - crashed
//All on black

else if ((b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' || s7 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    c = 0;
}

//All on white

else if (( b1 == '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    c = 0;
}

//9 closed docked at conveyor

if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >=
'0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0'
&& us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    ConveyorChill(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s9 == 'C')  && ( (us1 >= '1' && us2 >= '0' &&
us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1
== '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    ConveyorChill(ssp);
    c = 0;
}

//l2 on black l0 l1 on white go Left
```

```
else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s9 == 'C')  && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    ConveyorChill(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s9 == 'C')  && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    ConveyorChill(ssp);
    c = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    c = 0;
}

//check this
//Offline and switch 9 closed - docked
//All on black

else if ((b1 < '9' && b0 < '9' && b2 < '9') && (s9 == 'C'))

{
    ConveyorChill(ssp);
    c = 0;
}

//All on white

else if (( b1 == '9' && b0 == '9' && b2 == '9') && (s9 == 'C') )

{
    ConveyorChill(ssp);
    c = 0;
}
```

```
//Offline and obstacle detected <1m stop

//All on black

else if ( (b1 < '9' && b0 < '9' && b2 < '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//All on white

else if ( (b1 == '9' && b0 == '9' && b2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0')))

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//On line obstacle detected waiting for clear

//Reverse Analysis

//b0 on black b1 b2 on white go Reverse

if ( (b0 < '9') && (b1 == '9') && (b2 == '9')  && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b1 on black b0 b2 on white go Reverseleft

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9')  && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//b2 on black b0 b1 on white go Reverseright

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') ||
(us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' &&
us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//b0/b1 on black b2 on white go Reverseleft

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9')  && ( (us1 == '0' && us2 == '0' && us3 == '0') ||
(us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' &&
us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b2 on black b1 on white go Reverseright

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1
== '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 >
'0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Emergency stops online 6/7

if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C') && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    EmergencyStop(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C')  && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
```

```
    EmergencyStop(ssp);
    c = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s6 == 'C' || s7 == 'C') && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStop(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s6 == 'C' || s7 == 'C')  && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStop(ssp);
    c = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s6 == 'C' || s7 == 'C') && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    c = 0;
}

}
return err;
}
```

LineLightSimRMLatheReverseFusion.c

```
//Line Variation - RM to Lathe route - Reverse

//follow line
//no line - follow light Reverse dir
//sense side-left line

#define aWIN
#include "Access.h"
```

```
#include "Motor.h"
#include "LineLightSimRMLatheReverseFusion.h"

//ReverseSideLeft

int LineLightSimRMLatheReverseFusion(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe


                                        b1
                                        b0      S
                                        b2
                                l2 l0 l1
                                   W
*/

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LINER       *\\");          //change to LITER
   sleep(400) ;

   err = Read(ssp, 50);

   char b0 = ssp->pinputbuff[9] ;
   char b1 = ssp->pinputbuff[21] ;
   char b2 = ssp->pinputbuff[33] ;

   printf("b0 = %c \n" , b0);
   printf("b1 = %c \n" , b1);
   printf("b2 = %c \n" , b2);

   sleep(400) ;

   //Sideleft

   Flush(ssp) ;

   Write(ssp, 17, "#%+LINEL       *\\");
   sleep(400) ;

   err = Read(ssp, 50);
```

```
char l0 = ssp->pinputbuff[9] ;
char l1 = ssp->pinputbuff[21] ;
char l2 = ssp->pinputbuff[33] ;

printf("l0 = %c \n" , l0);
printf("l1 = %c \n" , l1);
printf("l2 = %c \n" , l2);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s1 = ssp->pinputbuff[10] ;
char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("s1 = %c \n" , s1);
printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");
sleep(400) ;
err = Read(ssp, 17);

char us1 = ssp->pinputbuff[10] ;
char us2 = ssp->pinputbuff[11] ;
char us3 = ssp->pinputbuff[12] ;

printf("us1 = %c \n" , us1);
printf("us2 = %c \n" , us2);
printf("us3 = %c \n" , us3);

sleep(400) ;
```

```
    //Ultrasonic Process SideLeft

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char uw1 = ssp->pinputbuff[10] ;
    char uw2 = ssp->pinputbuff[11] ;
    char uw3 = ssp->pinputbuff[12] ;

    printf("uw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

//Line following

//Reverse Analysis

//b0 on black b1 b2 on white go fwd

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1
>= '1' && us2 >= '0' && us3 >= '0'))

Reverse(ssp);

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b2 on black b0 b1 on white go Left

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp) ;

//b0/b1 on black b2 on white go right

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b0/b2 on black b1 on white go Left
```

```
else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp);

//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}
//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}
```

```
//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
```

```
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
```

```
}

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )
```

```
{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b2 on black l1 l2 on white go fwd
else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right
```

```
else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft135(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft225(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft135(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft225(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Lmt Ultra integration

//No line light follow

//All on black
```

```
else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheReverseFusion(ssp) ;
    a = 0 ;
}

//All on white

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheReverseFusion(ssp) ;
    a = 0 ;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheReverseFusion(ssp) ;
    a = 0 ;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheReverseFusion(ssp) ;
    a = 0 ;
}

//Offline and switch 6 7 closed - crashed
//All on black

else if ((l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' || s7 ==
'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//All on white
```

```
else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle detected <1m stop

//All on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//All on white
```

```
else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//Reverse Analysis

//b0 on black b1 b2 on white go Reverse

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b1 on black b0 b2 on white go Reverseleft

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b2 on black b0 b1 on white go Reverseright

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b1 on black b2 on white go Reverseleft

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b2 on black b1 on white go Reverseright

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Sideleft analysis

//Normal Sideleft
```

```
//l0 on black l1 l2 on white go Sideleft

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}
```

```
//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
```

```
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
```

```
    a = 0;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft225
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

/*More Chills on line but switch closed
```

```
s6 s7
Reverse
*/

if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )
```

```
{
    EmergencyStop(ssp) ;
    a = 0;
}

}

return err;

}
```

LineLightSimRMLatheSideleftFusion.c

```
//Line Variation - RM to Lathe route - Sideleft

//follow line
//no line - follow light sideleft dir

//ReverseSideLeft
//can take out reverse later
//use for conveyor to lathe - sideleft
//dock s=1

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineLightSimRMLatheSideleftFusion.h"

int LineLightSimRMLatheSideleftFusion(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe

                                        b1
                                        b0      S
                                        b2
                                l2 l0 l1
                                   W
*/

{
   aErr err = aErrNone;
   int b = 1;

   while(b==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LINER       *\\");          //change to LITER
   sleep(400) ;
```

```
   err = Read(ssp, 50);

   char b0 = ssp->pinputbuff[9] ;
   char b1 = ssp->pinputbuff[21] ;
   char b2 = ssp->pinputbuff[33] ;

   printf("b0 = %c \n" , b0);
   printf("b1 = %c \n" , b1);
   printf("b2 = %c \n" , b2);
sleep(400) ;

//Sideleft

   Flush(ssp) ;

   Write(ssp, 17, "#%+LINEL       *\\");
   sleep(400) ;

   err = Read(ssp, 50);

   char l0 = ssp->pinputbuff[9] ;
   char l1 = ssp->pinputbuff[21] ;
   char l2 = ssp->pinputbuff[33] ;

   printf("l0 = %c \n" , l0);
   printf("l1 = %c \n" , l1);
   printf("l2 = %c \n" , l2);

   sleep(400) ;

   //Limit Switch Analysis

   Flush(ssp) ;

   Write(ssp, 17, "#%+SWTCH       *\\");
   sleep(400) ;

   err = Read(ssp, 19);

   char s1 = ssp->pinputbuff[10] ;
   char s6 = ssp->pinputbuff[15] ;
   char s7 = ssp->pinputbuff[16] ;
   char s8 = ssp->pinputbuff[17] ;
   char s9 = ssp->pinputbuff[18] ;

   printf("s1 = %c \n" , s1);
   printf("s6 = %c \n" , s6);
   printf("s7 = %c \n" , s7);
   printf("s8 = %c \n" , s8);
   printf("s9 = %c \n" , s9);
```

```
    printf("\n");

    sleep(400) ;

    //Ultrasonic Process Reverse

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSS       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("us1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Ultrasonic Process SideLeft

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char uw1 = ssp->pinputbuff[10] ;
    char uw2 = ssp->pinputbuff[11] ;
    char uw3 = ssp->pinputbuff[12] ;

    printf("uw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

//Line following

//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))
Sideleft(ssp);

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))
```

```
Sideleft(ssp);

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft(ssp);

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);
```

```
//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b1 on black l1 l2 on white go fwd
```

```
else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )
  Sideleft(ssp);

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp);

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft135(ssp) ;

//Normal Sideleft225
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft225(ssp) ;

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft135(ssp) ;

//Normal Sideleft225
```

```
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

Sideleft225(ssp);

//Lmt Ultra integration

//No line light follow
//All on black

else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheSideleftFusion(ssp) ;
    b = 0 ;
}

//All on white

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheSideleftFusion(ssp) ;
    b = 0 ;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheSideleftFusion(ssp) ;
    b = 0 ;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9')

{
    Chill(ssp) ;
    LineOwnLightSimRMLatheSideleftFusion(ssp) ;
    b = 0 ;
}

//Offline and switch 6 7 closed - crashed
```

```
//All on black

else if ((l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' || s7 ==
'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    b = 0;

//All on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 ==
'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    b = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' ||
s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    b = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 == 'C' ||
s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    b = 0;
}

//Offline and obstacle detected <1m stop

//All on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0' &&
uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0'
&& uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3 ==
```

```
'0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0'
&& us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    b = 0;
}

//All on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0'
&& us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') ||
(us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    b = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3
== '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 ==
'0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    b = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3
== '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 ==
'0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    b = 0;
}
```

```
//On line obstacle detected waiting for clear

//Reverse Analysis

//b0 on black b1 b2 on white go Reverse

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1
== '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 >
'0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b1 on black b0 b2 on white go Reverseleft

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b2 on black b0 b1 on white go Reverseright

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b1 on black b2 on white go Reverseleft

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b2 on black b1 on white go Reverseright
```

```
else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go Sideleft

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0 on black l1 l2 on white go fwd
```

```
else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft225
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

/*More Chills on line but switch closed
s6 s7
Sideleft
*/

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    b = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    b = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    b = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    b = 0;
}

//l0/l2 on black l1 on white go Left
```

```
else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    b = 0;
}

/*More Chills
s1
Sideleft
*/

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s1 == 'C' ) && ( (uw1 == '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    b = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9')&& (s1 == 'C')&& ( (uw1 == '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    b = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    b = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    LatheChill(ssp) ;
    b = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    b = 0;
}

}

return err;

}
```

LineOwnLightSimConveyorRMForwardFusion.c

```
//light follow - search for light
//check for line return if found

//used for lathe-rm fwd

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LineOwnLightSimConveyorRMForwardFusion.h"

int LineOwnLightSimConveyorRMForwardFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);
```

```
  //r2

  char f21 = ssp->pinputbuff[32] ;
  char f22 = ssp->pinputbuff[33] ;

  //r3

  char f31 = ssp->pinputbuff[44] ;
  char f32 = ssp->pinputbuff[45] ;

  printf("\nf21 = %c \n" , f21);
  printf("f22 = %c \n" , f22);

  printf("\nf31 = %c \n" , f31);
  printf("f32 = %c \n" , f32);

   sleep(400) ;

  //line board

  //Foward                          LINER
Flush(ssp) ;

   Write(ssp, 17, "#%+LINER       *\\");              //change to LITEL
   sleep(400) ;

   err = Read(ssp, 50);

  char f0 = ssp->pinputbuff[9] ;
  char f1 = ssp->pinputbuff[21] ;
  char f2 = ssp->pinputbuff[33] ;

  printf("\nf0 = %c \n" , f0);
  printf("f1 = %c \n" , f1);
  printf("f2 = %c \n" , f2);

  sleep(400) ;

   //Ultrasonic Process Forward

  Flush(ssp) ;

  Write(ssp, 17, "#%+ULTSS       *\\");
  sleep(400) ;
  err = Read(ssp, 17);

  char un1 = ssp->pinputbuff[10] ;
  char un2 = ssp->pinputbuff[11] ;
  char un3 = ssp->pinputbuff[12] ;
```

```
    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Forward     left     m1  2       3   m2       right

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

Forward(ssp) ;

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

Right(ssp) ;

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

Left(ssp) ;

//approx 0.5 m in front of light
//only stops when docked s8 = C
```

```
else if ( (f21 == '1') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp) ;

else if ( (f21 == '1' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Left(ssp) ;

else if ( (f21 == '0') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Right(ssp) ;

//No light - search for light

else if ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}
//Limit Switch
//6 7

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' )  && ( (un1
>= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left
```

```
else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C
else if ( ( (f21 == '1') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( ( (f21 == '1' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( ( (f21 == '0') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}
```

```
//8 RM docking

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s8 == 'C' ) && ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s8 == 'C') && ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s8 == 'C')&& ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    a = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') ) && (s8 == 'C') && ( (un1 >=
'1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2
> '0' && un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2
== '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//approx 0.5 m in front of light
```

```
//only stops when docked s8 = C

else if ( ( (f21 == '1') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0')
|| (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0'
&& un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

else if ( ( (f21 == '1' && f31 == '0') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') ||
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

else if ( ( (f21 == '0') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0')
|| (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0'
&& un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
   RMChill(ssp);
    a = 0;
}
//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( (f21 == '0' && f22 <= '8' && f31 == '0' && f32 <= '8') && ( (un1 == '0' && un2 == '0'
&& un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '1') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 == '1' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 == '0') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//check for line

//normal foward
//f0 on black f1 f2 go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0')  )

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Right(ssp) ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Right(ssp) ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Left(ssp) ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f0-f2 on black f1 on white go Left
```

```
else if ( (f0 < '9' && f1 == '9' && f2 < '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Left(ssp) ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//Lmt switch

//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7 == 'C') )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white

else if ((f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ))
{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//All on black

else if (  (f1 < '9' && f0 < '9' && f2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//All on white
```

```
else if ( (f1 == '9' && f0 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
```

```
    a = 0;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' ) && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//More Chills
//s6 s7

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >= '0'
&& un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0')
|| (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

 {
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
```

```
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//Docking at RM switch 8

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >=
'0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 ==
'0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right
```

```
else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

else if ( (f1 < '9' && f0 < '9' && f2 < '9')&& (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}
```

```
//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9')&& (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

}

return err;

}
```

LineOwnLightSimLatheConveyorSiderightFusion.c

```
//fllw light sideright dir
//check for line if found return

#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LineOwnLightSimLatheConveyorSiderightFusion.h"

int LineOwnLightSimLatheConveyorSiderightFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITEL       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideright

   //r0

   char r01 = ssp->pinputbuff[8] ;
   char r02 = ssp->pinputbuff[9] ;

   //r1
```

```
char r11 = ssp->pinputbuff[20] ;
char r12 = ssp->pinputbuff[21] ;

printf("\nr01 = %c \n" , r01);
printf("r02 = %c \n" , r02);

printf("\nr11 = %c \n" , r11);
printf("r12 = %c \n" , r12);

 sleep(400) ;

 //Sideright                          LINER

 Flush(ssp) ;

 Write(ssp, 17, "#%+LINEL       *\\");
 sleep(400) ;

 err = Read(ssp, 50);

char r0 = ssp->pinputbuff[9] ;
char r1 = ssp->pinputbuff[21] ;
char r2 = ssp->pinputbuff[33] ;

printf("\nr0 = %c \n" , r0);
printf("r1 = %c \n" , r1);
printf("r2 = %c \n" , r2);

sleep(500) ;

//Ultrasonic Process SideRight

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //E
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
printf("ue3 = %c \n" , ue3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;
```

```
    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s0 = ssp->pinputbuff[9] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;

    printf("\ns0 = %c \n" , s0);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);

    printf("\n");

//Analysis        9 = Light

//Sideright    315      m2 r0     r1 m3       45

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3
>= '0'))

Sideright(ssp) ;

//More light toward r12 right

else if ((r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright45(ssp) ;

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright315(ssp) ;

//approx 0.5m in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp) ;

else if ( (r01 == '1' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp) ;
```

```
else if ( (r01 == '0' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//no light - search for light

else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//Lmt
//6 7
//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1
>= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
```

```
}

// No light

else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    a = 0;
}

//approx 0.5m in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (r01 == '1' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//dont need for lathe to RM must stop .5m in front of light
//0

//Equal light
```

```
if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}
//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

// no light

else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && (s0 == 'C') && ( (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 >
'0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 ==
'0') ) )

{
    printf("/nLight search to be continued/n") ;
    TableSequenceLight(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    TableSequenceLight(ssp);
    a = 0;
}

else if ( (r01 == '1' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//Obstacle detected. Waiting for clear

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//no light

else if ( ( (r01 == '0') && (r02 <= '8')) && ((r11 == '0') && (r12 <= '8')) && ( (ue1 == '0' && ue2
== '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 ==
'0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;

}

//approx 0.5m in front of light go sideleft
//only stops when docked s1 = C
else if ( (r01 == '1' && r11 == '1')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (r01 == '1' && r11 == '0')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (r01 == '0' && r11 == '1') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Check for line

if ( (r0 < '9') && (r1 == '9') && (r2 == '9') && ue1 >= '1' && ue2 >= '0' && ue3 >= '0')
```

```
{
    Sideright(ssp) ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//r1 on black r0 r2 on white && u >= 1 go Sideright45

else if ( (r1 < '9') && (r0 == '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

{
    Sideright45(ssp) ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//r2 on black r0 r1 on white && u >= 1 go Sideright315

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

{
    Sideright315(ssp);
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//r0/r1 on black r2 on white && u >= 1 go Sideright315

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Sideright45(ssp) ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//r0/r2 on black r1 on white && u >= 1 go Sideright315

else if ( ( r0 < '9' && r1 == '9'&& r2 < '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Sideright315(ssp);
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (s6 == 'C' || s7 == 'C'))
```

```
{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' ))

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )
{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//l2 l0 l1 on white
else if ( (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//l0 on black l1 l2 on
else if ( (r0 < '9' && r1 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
```

```
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9')&& ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9'&& r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
    LineLightSimLatheConveyorSiderightFusion(ssp);
    a = 0;
}

//Lmt integration s0

//l0 on black l1 l2 on white go fwd

if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3 >=
'0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0'
&& ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (s0 == 'C') && (r1 < '9' && r0 == '9'&& r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (s0 == 'C') && (r2 < '9' && r0 == '9' && r1 == '9')&& ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (s0 == 'C')&& (r0 < '9' && r1 < '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l0/l2 on black l1 on white go Left
```

```
else if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 < '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//More Chills
//s6 s7

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

}

return err;

}
```

LineOwnLightSimLatheRMSiderightFusion.c

```
//follow light sideright dir
//if light = 1 stop go foward
//no light - search for light
//search for line - line follow

#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LineOwnLightSimLatheRMSiderightFusion.h"
#include "LineOwnLightSimConveyorRMForwardFusion.h"

int LineOwnLightSimLatheRMSiderightFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   //light board

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITEL       *\\");
   sleep(400) ;
```

```
   err = Read(ssp, 50);

   //Sideright

//r0

   char r01 = ssp->pinputbuff[8] ;
   char r02 = ssp->pinputbuff[9] ;

   //r1

   char r11 = ssp->pinputbuff[20] ;
   char r12 = ssp->pinputbuff[21] ;


   printf("\nr01 = %c \n" , r01);
   printf("r02 = %c \n" , r02);

   printf("\nr11 = %c \n" , r11);
   printf("r12 = %c \n" , r12);

    sleep(400) ;

   //line board

    Flush(ssp) ;

    Write(ssp, 17, "#%+LINER       *\\");              //change to LITEL
    sleep(400) ;

    err = Read(ssp, 50);

   char f0 = ssp->pinputbuff[9] ;
   char f1 = ssp->pinputbuff[21] ;
   char f2 = ssp->pinputbuff[33] ;

   printf("\nf0 = %c \n" , f0);
   printf("f1 = %c \n" , f1);
   printf("f2 = %c \n" , f2);

   sleep(400) ;

   //siderright board

   //1st board going right                LINEL

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LINEL       *\\");
```

```
sleep(400) ;

err = Read(ssp, 50);

char r0 = ssp->pinputbuff[9] ;
char r1 = ssp->pinputbuff[21] ;
char r2 = ssp->pinputbuff[33] ;

printf("\nr0 = %c \n" , r0);
printf("r1 = %c \n" , r1);
printf("r2 = %c \n" , r2);

 sleep(500) ;

//Ultrasonic Process SideRight

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //E
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
printf("ue3 = %c \n" , ue3);

sleep(400) ;

  //Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSN       *\\");
sleep(400) ;
err = Read(ssp, 17);

char un1 = ssp->pinputbuff[10] ;
char un2 = ssp->pinputbuff[11] ;
char un3 = ssp->pinputbuff[12] ;

printf("\nun1 = %c \n" , un1);
printf("un2 = %c \n" , un2);
printf("un3 = %c \n" , un3);

sleep(400) ;

//Limit Switch Analysis
```

```
    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s0 = ssp->pinputbuff[9] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns0 = %c \n" , s0);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);

    printf("\n");

//Analysis        9 = Light

//Sideright    315      m2 r0     r1 m3       45

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3
>= '0'))

Sideright(ssp) ;

//More light toward r12 right

else if ((r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright45(ssp) ;

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright315(ssp) ;

//no light - line follow
```

```
else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && ( (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//approx 50 cm in front of light load forward
//ConveyorRM = LatheRM
//only stops when docked s1 = C

else if (r01 == '1' || r11 == '1')

{
   Chill(ssp);
   LineOwnLightSimConveyorRMForwardFusion(ssp);
   printf("\nApprox 0.5 m in front of Light. Loading Forward\n") ;
   a = 0 ;
}

//Lmt
//6 7

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1
>= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left
```

```
else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// No light

else if ( ( (r01 == '0' && r02 <= '8') && (r11 == '0' && r12 <= '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    a = 0;
}

//Obstacle detected. Waiting for clear

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left
```

```
else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Check for line

//Sideright Analysis                          m2  r2 r0 r1

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

{
     Sideright(ssp);
     LineLightSimLatheRMSiderightFusion(ssp);
     a=0;
}

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
Sideright45(ssp);
LineLightSimLatheRMSiderightFusion(ssp);
a=0;
}

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
Sideright315(ssp);
LineLightSimLatheRMSiderightFusion(ssp);
a=0;
}

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )
```

```
{
Sideright45(ssp);
LineLightSimLatheRMSiderightFusion(ssp);
a=0;
}

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

{
Sideright315(ssp);
LineLightSimLatheRMSiderightFusion(ssp);
a=0;
}

// Foward Analysis                              front

/*                                          f1
                                            f0
                                            f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0')  )

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
```

```
{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Right(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
```

```
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') )

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Right(ssp) ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9'  && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward
```

```
else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r1 < '9'  && r0 == '9' && r2 == '9'&& f2 == '9' && f0 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}
//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Left(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
```

```
}

//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f0-f2 on black f1 on white go Left
```

```
else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Left(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
```

```
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0-r1 on black go Forward
```

```
else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9'  && r2 < '9' && r0 < '9' && r1 == '9' && f2 == '9' && f1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//Lmt switch

//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7
== 'C'|| s8 == 'C'|| s9 == 'C') )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 ==
'C' || s7 == 'C' || s8 == 'C' || s9 == 'C') )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9')&& (s6 == 'C' ||
s7 == 'C'|| s8 == 'C'|| s9 == 'C') )
```

```
{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' ||
s7 == 'C' || s8 == 'C' || s9 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && ( (ue1 == '0' &&
ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' &&
ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white
```

```
else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )
  {
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheRMSiderightFusion(ssp);
    a=0;
}

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheRMSiderightFusion(ssp);
    a=0;
```

```
}

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheRMSiderightFusion(ssp);
    a=0;
}

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheRMSiderightFusion(ssp);
    a=0;
}

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9') && (r1 == '9')&& (r2 < '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimLatheRMSiderightFusion(ssp);
    a=0;
}

// Foward Analysis                                front

/*                                                 f1
                                                   f0
                                                   f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward
```

```
if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal right
```

```
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
```

```
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' && r1 < '9'  && r0 == '9' && r2 == '9' && f2 == '9' && f0 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}


//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0 on black f0 f1 on white go Forward
```

```
else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}
```

```
//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}


//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9' && r2 < '9' && r0 < '9' && r1 == '9'&& f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimConveyorRMForwardFusion(ssp);
    a = 0;
}

//More Chills
//s6 s7

if ( ((f0 < '9' && f1 == '9' && f2 == '9') || (r0 < '9' && r1 == '9' && r2 == '9')) && (s6 == 'C' || s7
== 'C'  ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0')
|| (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' &&
ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( ((f1 < '9' && f0 == '9' && f2 == '9') || (r1 < '9' && r0 == '9' && r2 == '9')) && (s6 == 'C' ||
s7 == 'C' )&& ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 ==
'0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0'
&& ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( ((f2 < '9' && f0 == '9' && f1 == '9')|| (r2 < '9' && r0 == '9' && r1 == '9')) && (s6 == 'C' ||
s7 == 'C' )&& ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 ==
'0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0'
&& ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( ((f0 < '9' && f1 < '9' && f2 == '9') || (r0 < '9' && r1 < '9' && r2 == '9')) && (s6 == 'C' || s7
== 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') ||
(ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' &&
ue3 > '0' && ue2 == '0') ) )
```

```
{
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( ( (f0 < '9' && f1 == '9' && f2 < '9') || (r0 < '9' && r1 == '9' && r2 < '9') ) && (s6 == 'C' ||
s7 == 'C' ) && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 ==
'0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0'
&& ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

}

return err;

}
```

LineOwnLightSimRMConveyorReverseFusion.c

```
//follow light - check for line
//s=9 dock conveyor

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightDock.h"
#include "LineOwnLightSimRMConveyorReverseFusion.h"

int LineOwnLightSimRMConveyorReverseFusion(serial_struct* ssp)

/*
                                    b1
                                    b0      S
                                    b2
                            */
{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {
   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
```

```
sleep(400) ;
err = Read(ssp, 50);

//Reverse
//b2

char b21 = ssp->pinputbuff[32] ;
char b22 = ssp->pinputbuff[33] ;

//r3

char b31 = ssp->pinputbuff[44] ;
char b32 = ssp->pinputbuff[45] ;

printf("\nb21 = %c \n" , b21);
printf("b22 = %c \n" , b22);

printf("\nb31 = %c \n" , b31);
printf("b32 = %c \n" , b32);

sleep(400) ;

 //Line board

Flush(ssp) ;

Write(ssp, 17, "#%+LINER       *\\\");          //change to LITER
sleep(400) ;

err = Read(ssp, 50);

char b0 = ssp->pinputbuff[9] ;
char b1 = ssp->pinputbuff[21] ;
char b2 = ssp->pinputbuff[33] ;

printf("b0 = %c \n" , b0);
printf("b1 = %c \n" , b1);
printf("b2 = %c \n" , b2);

sleep(400) ;

 //Ultrasonic Process Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\\");                  //S
sleep(400) ;
err = Read(ssp, 17);

char us1 = ssp->pinputbuff[10] ;
char us2 = ssp->pinputbuff[11] ;
```

```
    char us3 = ssp->pinputbuff[12] ;

    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    //sleep(400) ;

//Analysis       9 = Light

//Reverse          left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0'))

Reverse(ssp) ;

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseright(ssp) ;

//More light toward r22 left
```

```
else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseleft(ssp) ;

//No light <= 7 no obstacle follow line

else if ( ( (b21 == '0' && b22 <= '8'  && b31 == '0' && b32 <= '8') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0'))

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//approx 50 cm in front of light
//only stops when docked s9 = C

else if ( (b21 == '1') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverse(ssp) ;

else if ( (b21 == '1' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseleft(ssp) ;
 else if ( (b21 == '0') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseright(ssp) ;

/*Limit fusion
Following Light & switch closes - hit something
6 7*/

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ))
```

```
{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//  no light

else if ( ( (b21 == '0' && b22 <= '7') && (b31 == '0' && b32 <= '7') ) && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}
//50cm from light hit something

else if ( (b21 == '1') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 == '1' && b31 == '0')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}
```

```
else if ( (b21 == '0') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light - load line follow

else if ( ((b21 == '0') && (b22 <= '7')) && ((b31 == '0') && (b32 <= '7')) && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )
```

```
{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
    a = 0;
}

else if ( (b21 == '1') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '1' && b31 == '0')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '0') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//Docked at conveyor switch 9
//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)
```

```
{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//50cm from light hit something

else if ( (b21 == '1') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 == '1' && b31 == '0')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 == '0') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

//Check for line reverse

//Reverse Analysis

//b0 on black b1 b2 on white go fwd
```

```
if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverse(ssp);
   LineLightSimRMConveyorReverseFusion(ssp);
   a = 0;
}

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseleft(ssp) ;
   LineLightSimRMConveyorReverseFusion(ssp);
   a = 0;
}

//b2 on black b0 b1 on white go Left

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9')  && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseright(ssp) ;
   LineLightSimRMConveyorReverseFusion(ssp);
   a = 0;
}

//b0/b1 on black b2 on white go right

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseleft(ssp) ;
   LineLightSimRMConveyorReverseFusion(ssp);
   a = 0;
}

//b0/b2 on black b1 on white go Left

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseright(ssp);
   LineLightSimRMConveyorReverseFusion(ssp);
   a = 0;
}

//On line obstacle detected waiting for clear
```

```
//Reverse Analysis

//b0 on black b1 b2 on white go Reverse

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1
== '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 >
'0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMConveyorReverseFusion(ssp);
    a = 0;
}

//b1 on black b0 b2 on white go Reverseleft

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') ||
(us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' &&
us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMConveyorReverseFusion(ssp);
    a = 0;
}

//b2 on black b0 b1 on white go Reverseright

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9')  && ( (us1 == '0' && us2 == '0' && us3 == '0') ||
(us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' &&
us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMConveyorReverseFusion(ssp);
    a = 0;
}

//b0/b1 on black b2 on white go Reverseleft

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') ||
(us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' &&
us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMConveyorReverseFusion(ssp);
```

```
    a = 0;
}

//b0/b2 on black b1 on white go Reverseright

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9')  && ( (us1 == '0' && us2 == '0' && us3 == '0') ||
(us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' &&
us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMConveyorReverseFusion(ssp);
    a = 0;
}

//Emergency stops online 6/7

if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C') && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C')  && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s6 == 'C' || s7 == 'C') && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right
```

```
else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s6 == 'C' || s7 == 'C')  && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s6 == 'C' || s7 == 'C') && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//docking

//9 closed docked at conveyor

if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >=
'0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0'
&& us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    ConveyorChill(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right
else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s9 == 'C')  && ( (us1 >= '1' && us2 >= '0' &&
us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1
== '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    ConveyorChill(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s9 == 'C')  && ( (us1 >= '1' && us2 >= '0' &&
us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1
== '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    ConveyorChill(ssp);
```

```
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s9 == 'C')  && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    ConveyorChill(ssp);
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

}

return err;

}
```

LineOwnLightSimRMLatheReverseFusion.c

```
//follow light & check for line Reverse dir
//light = 1 go sideleft

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineOwnLightSimRMLatheReverseFusion.h"
#include "LightLineSimRMLatheSideleftFusion.h"

//ReverseSideLeft

int LineOwnLightSimRMLatheReverseFusion(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe

                                b1
```

```
                                        b0      S
                                        b2
                                l2 l0 l1
                                   W
*/

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Reverse

  //b2

   char b21 = ssp->pinputbuff[32] ;
   char b22 = ssp->pinputbuff[33] ;

   //r3

   char b31 = ssp->pinputbuff[44] ;
   char b32 = ssp->pinputbuff[45] ;

   printf("\nb21 = %c \n" , b21);
   printf("b22 = %c \n" , b22);

   printf("\nb31 = %c \n" , b31);
   printf("b32 = %c \n" , b32);

   sleep(400) ;

    //Ultrasonic Process Reverse

   Flush(ssp) ;

   Write(ssp, 17, "#%+ULTSS       *\\");
   sleep(400) ;
   err = Read(ssp, 17);

   char us1 = ssp->pinputbuff[10] ;
   char us2 = ssp->pinputbuff[11] ;
   char us3 = ssp->pinputbuff[12] ;
```

```
    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Line board

    Flush(ssp) ;

    Write(ssp, 17, "#%+LINER       *\\");          //change to LITER
    sleep(400) ;

    err = Read(ssp, 50);

    char b0 = ssp->pinputbuff[9] ;
    char b1 = ssp->pinputbuff[21] ;
    char b2 = ssp->pinputbuff[33] ;

    printf("b0 = %c \n" , b0);
    printf("b1 = %c \n" , b1);
    printf("b2 = %c \n" , b2);

    sleep(400) ;

    //Sideleft

    Flush(ssp) ;

    Write(ssp, 17, "#%+LINEL       *\\");
    sleep(400) ;
err = Read(ssp, 50);

char l0 = ssp->pinputbuff[9] ;
char l1 = ssp->pinputbuff[21] ;
char l2 = ssp->pinputbuff[33] ;

printf("l0 = %c \n" , l0);
printf("l1 = %c \n" , l1);
printf("l2 = %c \n" , l2);

sleep(400) ;

 //Ultrasonics

  Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");
 sleep(400) ;
 err = Read(ssp, 17);
```

```
    char uw1 = ssp->pinputbuff[10] ;
    char uw2 = ssp->pinputbuff[11] ;
    char uw3 = ssp->pinputbuff[12] ;

    printf("\n");
    printf("uw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s1 = ssp->pinputbuff[10] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns1 = %c \n" , s1);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

 //Analysis       9 = Light

//Reverse           left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0') )

Reverse(ssp) ;

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0') )

Reverseright(ssp) ;

//More light toward r22 left
```

```
else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0') )

Reverseleft(ssp) ;

//No light search for light

else if ( ( (b21 == '0' && b22 <= '8'  && b31 == '0' && b32 <= '8') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//50cm in front of light. Loading LineOwnLightSideleft
//will check for line while light following

else if ( (b21 == '1') || (b31 == '1') )

{
    Chill(ssp) ;
    printf("\n50cm in front of light. Loading Sideleft.\n") ;
    LineOwnLightSimRMLatheSideleftFusion(ssp) ;
    a = 0;
}

/*Limit fusion
Following Light & switch closes - hit something
6 7*/

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
```

```
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// 1000 chill or all 0 no light

else if ( ( (b21 == '0' && b22 <= '8') && (b31 == '0' && b32 <= '8') ) && (s6 == 'C' || s7 == 'C' )
&& ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 ==
'0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0'
&& us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ((b21 == '0') && (b22 <= '8')) && ((b31 == '0') && (b32 <= '8')) && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Check for line in Reverse Sideleft direction

//Reverse Analysis

//b0 on black b1 b2 on white go fwd

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverse(ssp);
   LineLightSimRMLatheReverseFusion(ssp);
   a = 0;
}

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseleft(ssp) ;
   LineLightSimRMLatheReverseFusion(ssp);
   a = 0;
}

//b2 on black b0 b1 on white go Left
```

```
else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseright(ssp) ;
   LineLightSimRMLatheReverseFusion(ssp);
   a = 0;
}

//b0/b1 on black b2 on white go right

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseleft(ssp) ;
   LineLightSimRMLatheReverseFusion(ssp);
   a = 0;
}

//b0/b2 on black b1 on white go Left

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
   Reverseright(ssp);
   LineLightSimRMLatheReverseFusion(ssp);
   a = 0;
}

//Sideleft analysis
//Normal Sideleft
//l0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
```

```
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0 b0-b1 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
```

```
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )
```

```
{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l2 b0 on black l1 l2 on white go fwd
```

```
else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l2 b1 on black l1 l2 on white go fwd
else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
```

```
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
   Sideleft(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft135(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft225(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right
```

```
else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft135(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
   Sideleft225(ssp);
   LineLightSimRMLatheSideleftFusion(ssp);
   a = 0;
}

//On line obstacle detected waiting for clear

//Reverse Analysis

//b0 on black b1 b2 on white go Reverse

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1
== '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 >
'0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheReverseFusion(ssp);
    a = 0;
}

//b1 on black b0 b2 on white go Reverseleft

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheReverseFusion(ssp);
    a = 0;
}
```

```
//b2 on black b0 b1 on white go Reverseright

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheReverseFusion(ssp);
    a = 0;
}

//b0/b1 on black b2 on white go Reverseleft

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheReverseFusion(ssp);
    a = 0;
}

//b0/b2 on black b1 on white go Reverseright

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheReverseFusion(ssp);
    a = 0;
}

//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go Sideleft

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b2 on black l1 l2 on white go fwd
else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}
//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}
//l2 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}
//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft225
//l2 on black l0 l1 on white go Left
```

```
else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    a = 0;
}

/*More Chills on line but switch closed
s6 s7
Reverse
*/

if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
```

```
}

//l1 on black l0 l2 on white go Right

else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}
//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

/*More Chills
s1
Sideleft
*/
```

```
if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s1 == 'C' ) && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9')&& (s1 == 'C')&& ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    a = 0;
```

```
}

}

return err;

}
```

LineOwnLightSimRMLatheSideleftFusion.c

```
//follow light & check for line Sideleft dir

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineOwnLightSimRMLatheSideleftFusion.h"
#include "LightLineSimRMLatheSideleftFusion.h"

//ReverseSideLeft

int LineOwnLightSimRMLatheSideleftFusion(serial_struct* ssp)

{
   aErr err = aErrNone;
   int b = 1;

   while(b==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideleft

   //l0

   char l01 = ssp->pinputbuff[8] ;
   char l02 = ssp->pinputbuff[9] ;

   //l1

   char l11 = ssp->pinputbuff[20] ;
   char l12 = ssp->pinputbuff[21] ;

   printf("\n\nl01 = %c \n" , l01);
   printf("l02 = %c \n" , l02);
```

```
printf("\nl11 = %c \n" , l11);
printf("l12 = %c \n" , l12);

sleep(400) ;

//line board

 //Sideleft
Flush(ssp) ;

Write(ssp, 17, "#%+LINEL       *\\");
sleep(400) ;

err = Read(ssp, 50);

char l0 = ssp->pinputbuff[9] ;
char l1 = ssp->pinputbuff[21] ;
char l2 = ssp->pinputbuff[33] ;

printf("l0 = %c \n" , l0);
printf("l1 = %c \n" , l1);
printf("l2 = %c \n" , l2);

sleep(400) ;

 //Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");               //W
sleep(400) ;
err = Read(ssp, 17);

char uw1 = ssp->pinputbuff[10] ;
char uw2 = ssp->pinputbuff[11] ;
char uw3 = ssp->pinputbuff[12] ;

printf("\nuw1 = %c \n" , uw1);
printf("uw2 = %c \n" , uw2);
printf("uw3 = %c \n" , uw3);

sleep(500) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);
```

```
    char s1 = ssp->pinputbuff[17] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[10] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s1 = %c \n" , s1);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis        9 = Light

//Sideleft     225  m1 l1       l0 m4     135

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') && s1 == 'O' )

Sideleft(ssp) ;

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft225(ssp) ;

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft135(ssp) ;

//No light line follow no obstacle

else if ( ( (l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8') ) && (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') && s1 == 'O')

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C
```

```
else if ( (l01 == '1' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;

}
else if ( (l01 == '1' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft135(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (l01 == '0' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft225(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

//Lmt
//6 7

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 ==
'0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 >
'0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r02 left
```

```
else if ( (l02 == '9' && l12 < '9' )&& (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}
//No light

else if ( ( (l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '1' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '1' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '0' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//1

//light 50cm away
```

```
else if ( (l01 == '1' || l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}
//more light l01 135

else if ( (l01 == '1' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
```

```
    b = 0;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//No light

else if ( ((l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8')) && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left
```

```
else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ( (l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8') ) && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    b = 0;
}

//light 50cm away

else if ( (l01 == '1' || l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l01 135

else if ( (l01 == '1' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//check for line sideleft
//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft135(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft135(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9')  && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft225(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Normal Sideleft225
```

```
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9')  && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
    Sideleft225(ssp);
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Obstacle detected. Waiting for clear
//Normal Sideleft
//l0 on black l1 l2 on white go Sideleft

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9')  && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Normal Sideleft225
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
```

```
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9')  && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') ||
(uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0'
&& uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LineLightSimRMLatheSideleftFusion(ssp);
    b = 0;
}

/*More Chills on line but switch closed
s6 s7
Sideleft
*/

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 >
'0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStop(ssp);
    b = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)
```

```
{
    EmergencyStop(ssp);
    b = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp);
    b = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2
>= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp) ;
    b = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >=
'0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&&
uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') )
)

{
    EmergencyStop(ssp) ;
    b = 0;
}

/*More Chills
s1
Sideleft
*/

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s1 == 'C' ) && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
```

```
    LatheChill(ssp);
    b = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9')&& (s1 == 'C')&& ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    b = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    b = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    b = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    b = 0;
}

}

return err;
```

}

1.6 Light Variation Technique

Light Variation Interface.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"

#include "LightVariationInterface.h"
#include "LightLineSimConveyorRMForwardFusionHigh.h"
#include "LightLineSimRMConveyorReverseFusionHigh.h"
#include "LightLineSimRMLatheSideleftFusionHigh.h"

#include "LightLineSimConveyorRMForwardFusionLow.h"
#include "LightLineSimRMLatheSideleftFusionLow.h"
#include "LightLineSimRMConveyorReverseFusionLow.h"

#include "LightIntensityConfirm.h"

#include "MainInterface.h"

int LightVariationInterface(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone;

   int i ;

   printf("\nPlease select a Light Intensity: \n\n"  ) ;

   printf("High (20W, 35W, 50W) \t 1 \nLow (5W, 10W) \t\t 2 \nExit to Main \t\t 3 \n\n") ;
   scanf("%d" , &i) ;

   if (i == 1)
   LightVariationHighInterface(ssp) ;

   else if (i == 2)
   LightVariationLowInterface(ssp) ;

   else if (i == 3)
   MainInterface(ssp) ;

   else printf("\nUnknown Intensity. Please enter new Intensity\n") ;

   return err;
}
```

```
//LightVariationHigh

int LightVariationHighInterface(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone;

   int route ;

   printf("\nPlease select a High Light Variation route: \n\n"  ) ;

   printf("Conveyor to Lathe \t 1 \nConveyor to RM \t\t 2 \n\nLathe to Conveyor \t 3 \nLathe to RM
\t\t 4 \n \nRM to Lathe \t\t 5 \nRM to Conveyor \t\t 6 \nExit to Main \t\t 7 \n\n") ;
   scanf("%d" , &route) ;

   if (route == 1)
   {
   printf("\nInitialising conveyor to lathe sequence . . . \n\n\n\n") ;
   LightLineSimRMLatheSideleftFusionHigh(ssp) ;
   }

   else if (route == 2)
   {
   printf("\nInitialising conveyor to RM sequence . . . \n\n\n\n") ;
   LightLineSimConveyorRMForwardFusionHigh(ssp) ;
   }

   else if (route == 3)
   {
   LightIntensityConfirmLightLineSimLatheConveyorSiderightFusionHigh(ssp) ;
   }

   else if (route == 4)
   {
   LightIntensityConfirmLightLineSimLatheRMSiderightFusionHigh(ssp) ;
   }

   else if (route == 5)
   {
   LightIntensityConfirmLightLineSimRMLatheReverseFusionHigh(ssp) ;
   }

   else if (route == 6)
   {
   printf("\nInitialising RM to conveyor sequence . . . \n\n\n\n") ;
   LightLineSimRMConveyorReverseFusionHigh(ssp) ;
   }
```

```
   else if (route == 7)
   {
   MainInterface(ssp) ;
   }
   else printf("\nUnknown route. Please enter new route\n") ;

return err;
}

//LightVariationLow

int LightVariationLowInterface(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone;
   int route ;

   printf("\nPlease select a Low Light Variation route: \n\n"  ) ;

   printf("Conveyor to Lathe \t 1 \nConveyor to RM \t\t 2 \n\nLathe to Conveyor \t 3 \nLathe to RM
\t\t 4 \n \nRM to Lathe \t\t 5 \nRM to Conveyor \t\t 6 \nExit to Main \t\t 7 \n\n") ;
   scanf("%d" , &route) ;

   if (route == 1)
   {
   printf("\nInitialising conveyor to lathe sequence . . . \n\n\n\n") ;
   LightLineSimRMLatheSideleftFusionLow(ssp) ;
   }

   else if (route == 2)
   {
   printf("\nInitialising conveyor to RM sequence . . . \n\n\n\n") ;
   LightLineSimConveyorRMForwardFusionLow(ssp) ;
   }

   else if (route == 3)
   {
   LightIntensityConfirmLightLineSimLatheConveyorSiderightFusionLow(ssp) ;
   }

   else if (route == 4)
   {
   LightIntensityConfirmLightLineSimLatheRMSiderightFusionLow(ssp) ;
   }

   else if (route == 5)
   {
   LightIntensityConfirmLightLineSimRMLatheReverseFusionLow(ssp) ;
   }
```

```
   else if (route == 6)
   {
    printf("\nInitialising RM to conveyor sequence . . . \n\n\n\n") ;
   LightLineSimRMConveyorReverseFusionLow(ssp) ;
   }

   else if (route == 7)
   {
   MainInterface(ssp) ;
   }

   else printf("\nUnknown route. Please enter new route\n") ;

return err;

}
```

Light Intensity Confirm.c

```
#define aWIN
#include "Access.h"
#include "LightIntensityConfirm.h"
#include "LightLineSimLatheRMSiderightFusionHigh.h"
#include "LightLineSimLatheRMSiderightFusionLow.h"
#include "LightLineSimRMLatheReverseFusionHigh.h"
#include "LightLineSimRMLatheReverseFusionLow.h"
#include "LightLineSimLatheConveyorSiderightFusionHigh.h"
#include "LightLineSimLatheConveyorSiderightFusionLow.h"

//for each route
//lathe - RM - high

int LightIntensityConfirmLightLineSimLatheRMSiderightFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
   int c = 2;

   printf("\nHigh light following - Lathe to RM route\n\n") ;

   while (c == 2)
   {
   printf("\nPlease confirm docking station under conveyor is at 0.1m mark\n\n");
   printf("\nYes              1");
   printf("\nNo               2\n");
   printf("\nExit to Main      3\n");

   scanf("%d", &c) ;

   //Load sequence
```

```
    if (c == 1)

    {
    printf("\nInitialising lathe to RM sequence . . . \n\n\n\n") ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    }

    //Load Main

    else if (c == 3)
    {
    MainInterface(ssp) ;
    }

   // Unknown selection. Made c = 2

    else if (c > 3 || c <= 0)
    {
    printf("\nUnknown selection.\n");
    c = 2 ;
    }

}
     return err;
}

//lathe - RM - low

int LightIntensityConfirmLightLineSimLatheRMSiderightFusionLow(serial_struct* ssp)
{
    int c = 2;
    aErr err = aErrNone;
    printf("\nLow light following - Lathe to RM route\n\n") ;

    while (c == 2)
    {
    printf("\nPlease confirm docking station under conveyor is at -0.4m mark\n\n");
    printf("\nYes                1");
    printf("\nNo                 2\n");
    printf("\nExit to Main       3\n");

    scanf("%d", &c) ;

    //Load sequence

    if (c == 1)

    {
    printf("\nInitialising lathe to RM sequence . . . \n\n\n\n") ;
    LightLineSimLatheRMSiderightFusionLow(ssp) ;
```

```
    }

    //Load Main

    else if (c == 3)
    {
    MainInterface(ssp) ;
    }

   // Unknown selection. Made c = 2

    else if (c > 3 || c <= 0)
    {
    printf("\nUnknown selection.\n");
    c = 2 ;
    }

}
 return err;
}
//RM - lathe - high

int LightIntensityConfirmLightLineSimRMLatheReverseFusionHigh(serial_struct* ssp)

{

    aErr err = aErrNone;

    int c = 2;

    printf("\nHigh light following - RM to Lathe route\n\n") ;

    while (c == 2)
    {
    printf("\nPlease confirm docking station under conveyor is at 3m mark\n\n");
    printf("\nYes                1");
    printf("\nNo                 2\n");
    printf("\nExit to Main       3\n\n");

    scanf("%d", &c) ;

    //Load sequence

    if (c == 1)

    {
    printf("\nInitialising RM to lathe sequence . . . \n\n\n\n") ;
    LightLineSimRMLatheReverseFusionHigh(ssp);
    }

    //Load Main
```

```
   else if (c == 3)
   {
   MainInterface(ssp) ;
   }

  // Unknown selection. Made c = 2

   else if (c > 3 || c <= 0)
   {
   printf("\nUnknown selection.\n");
   c = 2 ;
   }

}
  return err;
}



//RM - lathe - low

int LightIntensityConfirmLightLineSimRMLatheReverseFusionLow(serial_struct* ssp)

{
   aErr err = aErrNone;

   int c = 2;

   printf("\nLow light following - RM to Lathe route\n\n") ;

   while (c == 2)
   {
   printf("\nPlease confirm docking station under conveyor is at 3.5m mark\n\n");
   printf("\nYes              1");
   printf("\nNo               2\n");
   printf("\nExit to Main      3\n");

   scanf("%d", &c) ;

   //Load sequence

   if (c == 1)

   {
   printf("\nInitialising RM to lathe sequence . . . \n\n\n\n") ;
   LightLineSimRMLatheReverseFusionLow(ssp);
   }

   //Load Main
```

```
   else if (c == 3)
   {
   MainInterface(ssp) ;
   }

  // Unknown selection. Made c = 2

   else if (c > 3 || c <= 0)
   {
   printf("\nUnknown selection.\n");
   c = 2 ;
   }

}

  return err;

}


//lathe - conveyor - high
//necessary because AGV has to dock at correct distance (adjacent) to conveyor

int LightIntensityConfirmLightLineSimLatheConveyorSiderightFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;

   int c = 2;
   printf("\nHigh light following - Lathe to Conveyor route\n\n") ;

   while (c == 2)
   {
   printf("\nPlease confirm docking station under conveyor is at 0m mark\n\n");
   printf("\nYes              1");
   printf("\nNo               2\n");
   printf("\nExit to Main      3\n");

   scanf("%d", &c) ;

   //Load sequence

   if (c == 1)

   {
   printf("\nInitialising lathe to conveyor sequence . . . \n\n\n\n") ;
   LightLineSimLatheConveyorSiderightFusionHigh(ssp);
   }

   //Load Main
```

```
   else if (c == 3)
   {
   MainInterface(ssp) ;
   }

  // Unknown selection. Made c = 2

   else if (c > 3 || c <= 0)
   {
   printf("\nUnknown selection.\n");
   c = 2 ;
   }

}
  return err;
}



//lathe - conveyor - low
//necessary because AGV has to dock at correct distance (adjacent) to conveyor & if user calls low
1st

int LightIntensityConfirmLightLineSimLatheConveyorSiderightFusionLow(serial_struct* ssp)

{
   aErr err = aErrNone;

   int c = 2;

   printf("\nLow light following - Lathe to Conveyor route\n\n") ;

   while (c == 2)
   {
   printf("\nPlease confirm docking station under conveyor is at 0m mark\n\n");
   printf("\nYes              1");
   printf("\nNo               2\n");
   printf("\nExit to Main      3\n");

   scanf("%d", &c) ;

   //Load sequence

   if (c == 1)

   {
   printf("\nInitialising lathe to conveyor sequence . . . \n\n\n\n") ;
   LightLineSimLatheConveyorSiderightFusionLow(ssp);
   }
```

```
    //Load Main

    else if (c == 3)
    {
    MainInterface(ssp) ;
    }

   // Unknown selection. Made c = 2

    else if (c > 3 || c <= 0)
    {
    printf("\nUnknown selection.\n");
    c = 2 ;
    }

}
   return err;
}



  /* neglible only stops when docks
//RM  - Conveyor

int LightIntensityConfirmLightLineSimRMConveyorReverseFusionHigh(serial_struct* ssp)

{

    int c = 2;
    printf("\nHigh light following - Conveyor to RM route\n\n") ;

    while (c == 2)
    {
    printf("\nPlease confirm docking station at the end of reverse track is at 3m mark\n\n");
    printf("\nYes                1");
    printf("\nNo                 2\n");
    printf("\nExit to Main       3\n");

    scanf("%d", &c) ;

    //Load sequence

    if (c == 1)

    {
    LightLineSimConveyorRMForwardFusionHigh(ssp)
    }

    //Load Main

    else if (c == 3)
```

```
   {
   MainInterface(ssp) ;
   }

  // Unknown selection. Made c = 2

   else if (c > 3 || c <= 0)
   {
   printf("\nUnknown selection.\n");
   c = 2 ;
   }

}
    return err;
}

*/
```

LightLineSimConveyorRMForwardFusionHigh.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LightLineSimConveyorRMForwardFusionHigh.h"
#include "LightLineSimConveyorRMForwardFusionLow.h"
#include "LightOwnLineSimConveyorRMForwardFusionHigh.h"

//use for lathe to RM
//follow light - no light follow line Reverse dir
//light == 8 change to low

//s=8 dock RM

int LightLineSimConveyorRMForwardFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;

   int a = 1;

   while(a==1)

   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);
```

```
//r2

char f21 = ssp->pinputbuff[32] ;
char f22 = ssp->pinputbuff[33] ;

//r3

char f31 = ssp->pinputbuff[44] ;
char f32 = ssp->pinputbuff[45] ;

printf("\nf21 = %c \n" , f21);
printf("f22 = %c \n" , f22);

printf("\nf31 = %c \n" , f31);
printf("f32 = %c \n" , f32);

 sleep(400) ;

 //Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSN       *\\");
sleep(400) ;
err = Read(ssp, 17);

char un1 = ssp->pinputbuff[10] ;
char un2 = ssp->pinputbuff[11] ;
char un3 = ssp->pinputbuff[12] ;

printf("\nun1 = %c \n" , un1);
printf("un2 = %c \n" , un2);
printf("un3 = %c \n" , un3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("\ns6 = %c \n" , s6);
```

```
   printf("s7 = %c \n" , s7);
   printf("s8 = %c \n" , s8);
   printf("s9 = %c \n" , s9);
   printf("\n");

   //sleep(400) ;

//Forward     left      m1  2       3   m2       right
//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

Forward(ssp) ;

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

Right(ssp) ;

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

Left(ssp) ;

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( (f21 == '1') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp) ;

else if ( (f21 == '1' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Left(ssp) ;

else if ( (f21 == '0') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Right(ssp) ;

//No light -line follow

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
   Chill(ssp) ;
```

```
   LightOwnLineSimConveyorRMForwardFusionHigh(ssp) ;
   a = 0;
}

//light == 8 change to low

else if ( (f21 == '0' && f22 == '8' && f31 == '0' && f32 == '8') && ( (un1 >= '1' && un2 >= '0'
&& un3 >= '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp) ;
    LightLineSimConveyorRMForwardFusionLow(ssp) ;
    a = 0;
}

//Limit Switch
//6 7

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' )  && ( (un1
>= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
```

```
    EmergencyStopLight(ssp);
    a = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '1') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}


else if ( ( (f21 == '1' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}


else if ( ( (f21 == '0') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//light == 8 crashes
```

```
else if ( (f21 == '0' && f22 == '8' && f31 == '0' && f32 == '8')&& (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//8 RM docking

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s8 == 'C' ) && ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s8 == 'C') && ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s8 == 'C')&& ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    a = 0;
}

//No light
```

```
else if ( ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') ) && (s8 == 'C') && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0' && un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '1') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

else if ( ( (f21 == '1' && f31 == '0') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

else if ( ( (f21 == '0') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
   RMChill(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light
else if ( (f21 == '0' && f22 <= '7' && f31 == '0' && f32 <= '7') && ( (un1 == '0' && un2 == '0'
&& un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp) ;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '1') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 == '1' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 == '0') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

}
return err;
}
```

LightLineSimConveyorRMForwardFusionLow.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LightLineSimConveyorRMForwardFusionLow.h"
#include "LightOwnLineSimConveyorRMForwardFusionHigh.h"

//follow light - no light follow line Reverse dir
//light == 8 change to low

//s=9 dock conveyor

//dnt go to high threshold 4 changing dir = 9
//all low light ffg must go to high line follow

int LightLineSimConveyorRMForwardFusionLow(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);
```

```
//r2

char f21 = ssp->pinputbuff[32] ;
char f22 = ssp->pinputbuff[33] ;

//r3

char f31 = ssp->pinputbuff[44] ;
char f32 = ssp->pinputbuff[45] ;

printf("\nf21 = %c \n" , f21);
printf("f22 = %c \n" , f22);

printf("\nf31 = %c \n" , f31);
printf("f32 = %c \n" , f32);

 sleep(400) ;

 //Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");
sleep(400) ;
err = Read(ssp, 17);

char un1 = ssp->pinputbuff[10] ;
char un2 = ssp->pinputbuff[11] ;
char un3 = ssp->pinputbuff[12] ;

printf("\nun1 = %c \n" , un1);
printf("un2 = %c \n" , un2);
printf("un3 = %c \n" , un3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;
```

```
   printf("\ns6 = %c \n" , s6);
   printf("s7 = %c \n" , s7);
   printf("s8 = %c \n" , s8);
   printf("s9 = %c \n" , s9);
   printf("\n");

//Forward     left     m1  2       3   m2       right

//Equal light

if ( (f22 == '8' && f32 == '8' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

Forward(ssp) ;

//More light toward r32 right

else if ( (f22 < '8' && f32 == '8' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

Right(ssp) ;

//More light toward r22 left

else if ( (f22 == '8' && f32 < '8' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

Left(ssp) ;

//approx 1 m in front of light
//only stops when docked s8 = C

else if ( (f21 == '9') && (f31 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Forward(ssp) ;

else if ( (f21 == '9' && f31 <= '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Left(ssp) ;

else if ( (f21 <= '9') && (f31 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

Right(ssp) ;

//No light -line follow

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
   Chill(ssp) ;
```

```
   LightOwnLineSimConveyorRMForwardFusionHigh(ssp) ;
   a = 0;
}

//Limit Switch
//6 7

//Equal light

if ( (f22 == '8' && f32 == '8' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' )  && ( (un1
>= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (f22 < '8' && f32 == '8' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (f22 == '8' && f32 < '8' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
```

```
    EmergencyStopLight(ssp);
    a = 0;
}

//approx 1 m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '9') && (f31 == '9') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( ( (f21 == '9' && f31 < '9') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( ( (f21 < '9') && (f31 == '9') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//8 RM docking

//Equal light

if ( (f22 == '8' && f32 == '8' && f21 == '0' && f31 == '0') && (s8 == 'C' ) && ( (un1 >= '1' &&
un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 >
'0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RMChill(ssp);
    a = 0;
}

//More light toward r32 right
```

```
else if ( (f22 < '8' && f32 == '8' && f21 == '0' && f31 == '0') && (s8 == 'C') && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (f22 == '8' && f32 < '8' && f21 == '0' && f31 == '0') && (s8 == 'C')&& ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') ) && (s8 == 'C') && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0' && un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//approx 1m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '9') && (f31 == '9') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}


else if ( ( (f21 == '9' && f31 < '9') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    RMChill(ssp);
    a = 0;
}

else if ( ( (f21 < '9') && (f31 == '9') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
   RMChill(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '8' && f32 == '8' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (f22 < '8' && f32 == '8' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (f22 == '8' && f32 < '8' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light
```

```
else if ( (f21 == '0' && f22 <= '7' && f31 == '0' && f32 <= '7') && ( (un1 == '0' && un2 == '0'
&& un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp) ;
}

//approx 1m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '9') && (f31 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


else if ( ( (f21 == '9' && f31 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( ( (f21 < '9') && (f31 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

}

return err;

}
```

LightLineSimLatheConveyorSiderightFusionHigh.c

```
#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LightLineSimLatheConveyorSiderightFusionHigh.h"
#include "LightLineSimLatheConveyorSiderightFusionLow.h"
#include "LightOwnLineSimLatheConveyorSiderightFusionHigh.h"
#include "LightLineSimLatheConveyorSiderightFusionLow.h"

//follow light - no light follow line
//light == 8 change to low
//(light == 1 change to forward lathe to RM )

//dock conveyor switch 0
//table sequence

int LightLineSimLatheConveyorSiderightFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)

   {
   Flush(ssp) ;

   Write(ssp, 17, "#%+LITEL       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideright
   //r0

   char r01 = ssp->pinputbuff[8] ;
   char r02 = ssp->pinputbuff[9] ;

   //r1

   char r11 = ssp->pinputbuff[20] ;
   char r12 = ssp->pinputbuff[21] ;

   printf("\nr01 = %c \n" , r01);
   printf("r02 = %c \n" , r02);

   printf("\nr11 = %c \n" , r11);
   printf("r12 = %c \n" , r12);
    sleep(400) ;

   //Ultrasonic Process SideRight

   Flush(ssp) ;
```

```
    Write(ssp, 17, "#%+ULTSW       *\\");                    //E
    sleep(400) ;
    err = Read(ssp, 17);

    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("\nue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s0 = ssp->pinputbuff[9] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;

    printf("\ns0 = %c \n" , s0);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("\n");

//Analysis       9 = Light

//Sideright    315      m2 r0     r1 m3       45

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3
>= '0'))

Sideright(ssp) ;

//More light toward r12 right

else if ((r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright45(ssp) ;
```

```
//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Siderright315(ssp) ;

//no light - line follow

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') )

{
   Chill(ssp) ;
   LightOwnLineSimLatheConveyorSiderightFusionHigh(ssp) ;
   a = 0;
}

// ==8 change to low  -docking station is in correct place already

else if ( (r01 == '0' && r02 == '8' && r11 == '0' && r12 == '8') && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
   Chill(ssp) ;
   LightLineSimLatheConveyorSiderightFusionLow(ssp) ;
   a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
   Siderright(ssp) ;
   printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (r01 == '1' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Siderright315(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (r01 == '0' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
     Siderright45(ssp) ;
```

```
     printf("\nApprox 50 cm in front of Light.\n") ;
}

//Lmt
//6 7

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1
>= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// No light

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
```

```
    EmergencyStopLight(ssp);
    a = 0;
}

// low light

else if ( ( (r01 == '0' && r02 == '8') && (r11 == '0' && r12 == '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (r01 == '1' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//dont need for lathe to RM must stop .5m in front of light
//0

//Equal light
```

```
if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

// no light

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (s0 == 'C') && ( (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 >
'0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 ==
'0') ) )

{
    printf("/nLight search to be continued/n") ;
    TableSequenceLight(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C
else if ( (r01 == '1' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    TableSequenceLight(ssp);
    a = 0;
}

else if ( (r01 == '1' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//Obstacle detected. Waiting for clear
//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// 1000 chill or all 0 no light

else if ( ( (r01 == '0') && (r02 <= '7')) && ((r11 == '0') && (r12 <= '7')) && ( (ue1 == '0' && ue2
== '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 ==
'0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimLatheConveyorSiderightFusionHigh(ssp) ;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (r01 == '1' && r11 == '0')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (r01 == '0' && r11 == '1') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
}

return err;
}
```

LightLineSimLatheConveyorSiderightFusionLow.c

```
#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LightLineSimLatheConveyorSiderightFusionLow.h"
#include "LightOwnLineSimLatheConveyorSiderightFusionHigh.h"

//follow light - no light follow line

//(light == 1 change to forward lathe to RM )

//dock conveyor switch 0

int LightLineSimLatheConveyorSiderightFusionLow(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)

   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITEL       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideright
   //r0

   char r01 = ssp->pinputbuff[8] ;
   char r02 = ssp->pinputbuff[9] ;

   //r1

   char r11 = ssp->pinputbuff[20] ;
   char r12 = ssp->pinputbuff[21] ;

   printf("\nr01 = %c \n" , r01);
   printf("r02 = %c \n" , r02);

   printf("\nr11 = %c \n" , r11);
   printf("r12 = %c \n" , r12);

    sleep(400) ;

   //Ultrasonic Process SideRight
```

```
    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");                    //E
    sleep(400) ;
    err = Read(ssp, 17);

    char ue1 = ssp->pinputbuff[10] ;
    char ue2 = ssp->pinputbuff[11] ;
    char ue3 = ssp->pinputbuff[12] ;

    printf("\nue1 = %c \n" , ue1);
    printf("ue2 = %c \n" , ue2);
    printf("ue3 = %c \n" , ue3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s0 = ssp->pinputbuff[9] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;

    printf("\ns0 = %c \n" , s0);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("\n");

//Analysis       9 = Light

//Sideright    315      m2 r0     r1 m3       45

//Equal light

if ( (r02 == '8' && r12 == '8' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3
>= '0'))

Sideright(ssp) ;

//More light toward r12 right

else if ((r02 < '8' && r12 == '8' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))
```

```
Sideright45(ssp) ;

//More light toward r02 left

else if ( (r02 == '8' && r12 < '8' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright315(ssp) ;

//no light - line follow

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') )

{
   Chill(ssp) ;
   LightOwnLineSimLatheConveyorSiderightFusionHigh(ssp) ;
   a = 0;
}

//approx 1m in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '9' && r11 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright(ssp) ;

else if ( (r01 == '9' && r11 < '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp) ;

else if ( (r01 < '9' && r11 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

 Sideright45(ssp) ;

//Lmt
//6 7

//Equal light

if ( (r02 == '8' && r12 == '8' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1
>= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right
```

```
else if ( (r02 < '8' && r12 == '8' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 == '8' && r12 < '8' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// No light

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    a = 0;
}

//approx 1m in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '9' && r11 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (r01 == '9' && r11 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    EmergencyStopLight(ssp);
    a = 0;
}
else if ( (r01 < '9' && r11 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//dont need for lathe to RM must stop .5m in front of light
//0

//Equal light

if ( (r02 == '8' && r12 == '8' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '8' && r12 == '8' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 == '8' && r12 < '8' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

// no light
```

```
else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    TableSequenceLight(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '9' && r11 == '9') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

else if ( (r01 == '9' && r11 < '9') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

else if ( (r01 < '9' && r11 == '9') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//Obstacle detected. Waiting for clear
//Equal light

if ( (r02 == '8' && r12 == '8' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (r02 < '8' && r12 == '8' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (r02 == '8' && r12 < '8' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// 1000 chill or all 0 no light

else if ( ( (r01 == '0') && (r02 <= '7')) && ((r11 == '0') && (r12 <= '7')) && ( (ue1 == '0' && ue2
== '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 ==
'0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimLatheConveyorSiderightFusionHigh(ssp) ;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '9' && r11 == '9')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
else if ( (r01 == '9' && r11 < '9')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
else if ( (r01 < '9' && r11 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

}

return err;

}
```

LightLineSimLatheRMSiderightFusionHigh.c

```
#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LightLineSimLatheRMSiderightFusionHigh.h"
#include "LightLineSimConveyorRMForwardFusionHigh.h"
#include "LightLineSimLatheRMSiderightFusionLow.h"
#include "LightOwnLineSimLatheRMSiderightFusionHigh.h"
#include "LightIntensityConfirm.h"

//follow light - no light follow line
//light == 8 change to low
//light == 1 change to forward lathe to RM
//chkd

int LightLineSimLatheRMSiderightFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITEL       *\\");
```

```
sleep(400) ;
err = Read(ssp, 50);

// Sideright
//r0

char r01 = ssp->pinputbuff[8] ;
char r02 = ssp->pinputbuff[9] ;

//r1

char r11 = ssp->pinputbuff[20] ;
char r12 = ssp->pinputbuff[21] ;

printf("\nr01 = %c \n" , r01);
printf("r02 = %c \n" , r02);

printf("\nr11 = %c \n" , r11);
printf("r12 = %c \n" , r12);

 sleep(400) ;

//Ultrasonic Process SideRight

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //E
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
printf("ue3 = %c \n" , ue3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s0 = ssp->pinputbuff[9] ;
char s6 = ssp->pinputbuff[15] ;
```

```
   char s7 = ssp->pinputbuff[16] ;

   printf("\ns0 = %c \n" , s0);
   printf("s6 = %c \n" , s6);
   printf("s7 = %c \n" , s7);
   printf("\n");

//Analysis       9 = Light

//Sideright    315      m2 r0     r1 m3       45

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3
>= '0'))

Sideright(ssp) ;

//More light toward r12 right

else if ((r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright45(ssp) ;

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright315(ssp) ;

//no light - line follow

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') )

{
   Chill(ssp) ;
   LightOwnLineSimLatheRMSiderightFusionHigh(ssp) ;
   a = 0;
}

// ==8 change to low

else if ( (r01 == '0' && r02 == '8' && r11 == '0' && r12 == '8') && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
   Chill(ssp) ;
```

```
   LightIntensityConfirmLightLineSimLatheRMSiderightFusionLow(ssp) ;
   a = 0;
}

//approx 50 cm in front of light load forward
//ConveyorRM = LatheRM
//only stops when docked s1 = C

else if (r01 == '1' || r11 == '1')

{
   Chill(ssp);
   LightLineSimConveyorRMForwardFusionHigh(ssp);
   printf("\nApprox 0.5 m in front of Light. Loading Forward\n") ;
   a = 0 ;
}

//Lmt  //6 7
//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1
>= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
```

```
    a = 0;
}

// No light

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    a = 0;
}

// low light

else if ( ( (r01 == '0' && r02 == '8') && (r11 == '0' && r12 == '8') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}
//Obstacle detected. Waiting for clear

//Equal light

if ( (r02 == '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (r02 < '9' && r12 == '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//More light toward r02 left

else if ( (r02 == '9' && r12 < '9' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// no light

else if ( ( (r01 == '0') && (r02 <= '7')) && ((r11 == '0') && (r12 <= '7')) && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimLatheRMSiderightFusionHigh(ssp) ;
}
}
return err;
}
```

LightLineSimLatheRMSiderightFusionLow.c

```
#include "Motor.h"
#include "LightDock.h"
#include "LineDock.h"
#include "LightLineSimLatheRMSiderightFusionLow.h"
#include  "LightLineSimConveyorRMForwardFusionLow.h"
#include "LightOwnLineSimLatheRMSiderightFusionHigh.h"

//follow light - no light follow line
//light == 8 change to low
//(light == 1 change to forward lathe to RM )

int LightLineSimLatheRMSiderightFusionLow(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)

   {

   Flush(ssp) ;
```

```
Write(ssp, 17, "#%+LITEL       *\\");
sleep(400) ;
err = Read(ssp, 50);

//Sideright
//r0

char r01 = ssp->pinputbuff[8] ;
char r02 = ssp->pinputbuff[9] ;

//r1

char r11 = ssp->pinputbuff[20] ;
char r12 = ssp->pinputbuff[21] ;


printf("\nr01 = %c \n" , r01);
printf("r02 = %c \n" , r02);

printf("\nr11 = %c \n" , r11);
printf("r12 = %c \n" , r12);

 sleep(400) ;

//Ultrasonic Process SideRight
Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //E
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
printf("ue3 = %c \n" , ue3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s0 = ssp->pinputbuff[9] ;
```

```
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;

    printf("\ns0 = %c \n" , s0);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("\n");

//Analysis        9 = Light

//Sideright    315      m2 r0     r1 m3       45

//Equal light

if ( (r02 == '8' && r12 == '8' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3
>= '0'))

Sideright(ssp) ;

//More light toward r12 right

else if ((r02 < '8' && r12 == '8' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright45(ssp) ;

//More light toward r02 left

else if ( (r02 == '8' && r12 < '8' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

Sideright315(ssp) ;

//no light - line follow

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') )

{
    Chill(ssp) ;
    LightOwnLineSimLatheRMSiderightFusionHigh(ssp) ;
    a = 0;
}

//approx 1m in front of light load forward
//ConveyorRM = LatheRM
//only stops when docked s1 = C

else if (r01 == '9' || r11 == '9')

{
```

```
   Chill(ssp);
   LightLineSimConveyorRMForwardFusionLow(ssp);
   printf("\nApprox 1 m in front of Light. Loading Forward\n") ;
   a = 0 ;
}

//Lmt
//6 7

//Equal light

if ( (r02 == '8' && r12 == '8' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1
>= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' &&
ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 < '8' && r12 == '8' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 == '8' && r12 < '8' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// No light

else if ( ( (r01 == '0' && r02 <= '7') && (r11 == '0' && r12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )
```

```
{
    printf("/nLight search to be continued/n") ;
    EmergencyStopLight(ssp);
    a = 0;
}

//Obstacle detected. Waiting for clear

//Equal light

if ( (r02 == '8' && r12 == '8' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//More light toward r12 right

else if ( (r02 < '8' && r12 == '8' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//More light toward r02 left

else if ( (r02 == '8' && r12 < '8' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// no light

else if ( ( (r01 == '0') && (r02 <= '7')) && ((r11 == '0') && (r12 <= '7')) && ( (ue1 == '0' && ue2
== '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 ==
'0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimLatheRMSiderightFusionHigh(ssp) ;
    a = 0;
```

```
}

}
return err;
}
```

LightLineSimRMConveyorReverseFusionHigh.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightDock.h"
#include "LightLineSimRMConveyorReverseFusionHigh.h"
#include "LightLineSimRMConveyorReverseFusionLow.h"
#include "LightOwnLineSimRMConveyorReverseFusionHigh.h"

//follow light - no light follow line Reverse dir
//light == 8 change to low
//light == 9 change to high

//s=9 dock conveyor

//chked

int LightLineSimRMConveyorReverseFusionHigh(serial_struct* ssp)

/*
                                        b1
                                        b0      S
                                        b2
                                */
{

    aErr err = aErrNone;
    int a = 1;

    while(a==1)
    {

    Flush(ssp) ;

    Write(ssp, 17, "#%+LITER       *\\");
    sleep(400) ;
    err = Read(ssp, 50);

    //Reverse
    //b2

    char b21 = ssp->pinputbuff[32] ;
    char b22 = ssp->pinputbuff[33] ;
```

```
//r3

char b31 = ssp->pinputbuff[44] ;
char b32 = ssp->pinputbuff[45] ;


printf("\nb21 = %c \n" , b21);
printf("b22 = %c \n" , b22);

printf("\nb31 = %c \n" , b31);
printf("b32 = %c \n" , b32);

sleep(400) ;

 //Ultrasonic Process Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //S
sleep(400) ;
err = Read(ssp, 17);

char us1 = ssp->pinputbuff[10] ;
char us2 = ssp->pinputbuff[11] ;
char us3 = ssp->pinputbuff[12] ;

printf("\nus1 = %c \n" , us1);
printf("us2 = %c \n" , us2);
printf("us3 = %c \n" , us3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("\ns6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");
```

```
   //sleep(400) ;


//Analysis       9 = Light

//Reverse          left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0'))

Reverse(ssp) ;

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseright(ssp) ;

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseleft(ssp) ;

//No light <= 7 no obstacle follow line

else if ( ( (b21 == '0' && b22 <= '7'  && b31 == '0' && b32 <= '7') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0'))

{
    Chill(ssp) ;
   LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

//light == 8 change to low

else if ( (b21 == '0' && b22 == '8' && b31 == '0' && b32 == '8') && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') ||
(us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    LightLineSimRMConveyorReverseFusionLow(ssp);
    a = 0;
}
```

```
//approx 50 cm in front of light go sideleft
//only stops when docked s9 = C

else if ( (b21 == '1') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )
Reverse(ssp) ;

else if ( (b21 == '1' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseleft(ssp) ;

else if ( (b21 == '0') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseright(ssp) ;

/*Limit fusion
Following Light & switch closes - hit something
6 7*/

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ))

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
```

```
    EmergencyStopLight(ssp);
    a = 0;
}

// no light

else if ( ( (b21 == '0' && b22 <= '7') && (b31 == '0' && b32 <= '7') ) && (s6 == 'C' || s7 == 'C' )
&& ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 ==
'0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0'
&& us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//50cm from light hit something

else if ( (b21 == '1') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 == '1' && b31 == '0')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' &&
us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1
== '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 == '0') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light
```

```
if ( ( b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light - load line follow

else if ( ((b21 == '0') && (b22 <= '7')) && ((b31 == '0') && (b32 <= '7')) && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

else if ( (b21 == '1') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
else if ( (b21 == '1' && b31 == '0')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '0') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Docked at conveyor switch 9
//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)
```

```
{
    ConveyorChill(ssp);
    a = 0;
}

//50cm from light hit something

else if ( (b21 == '1') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 == '1' && b31 == '0')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 == '0') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

}
return err;
}
```

LightLineSimRMConveyorReverseFusionLow.c

```
//follow light - no light follow line Reverse dir
// == 9 stops in front of light loads sideleft
//doesnot go back to high
//change all 9's

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMConveyorReverseFusionLow.h"
```

```
#include "LightOwnLineSimRMConveyorReverseFusionHigh.h"

//ReverseSideLeft

//chkd

int LightLineSimRMConveyorReverseFusionLow(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe


                                        b1
                                        b0      S
                                        b2
                                l2 l0 l1
                                   W
*/

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Reverse
   //b2

   char b21 = ssp->pinputbuff[32] ;
   char b22 = ssp->pinputbuff[33] ;

   //r3
   char b31 = ssp->pinputbuff[44] ;
   char b32 = ssp->pinputbuff[45] ;


   printf("\nb21 = %c \n" , b21);
   printf("b22 = %c \n" , b22);

   printf("\nb31 = %c \n" , b31);
   printf("b32 = %c \n" , b32);

   sleep(400) ;
```

```
    //Ultrasonic Process Reverse

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");                    //S
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    //sleep(400) ;
//Analysis       8 = Light

//Reverse           left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0'))

Reverse(ssp) ;

//More light toward r32 right
```

```
else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseright(ssp) ;

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseleft(ssp) ;

//No light <= 7 follow line

else if ( ( (b21 == '0' && b22 <= '7'  && b31 == '0' && b32 <= '7') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0'))

{
    Chill(ssp) ;
    LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s9 = C

else if ( (b21 == '9') && (b31 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

{
    Reverse(ssp) ;

}

else if ( (b21 == '9' && b31 < '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

{
    Reverseleft(ssp) ;

}

else if ( (b21 < '9') && (b31 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

{
    Reverseright(ssp) ;

}

/*Limit fusion
Following Light & switch closes - hit something
6 7*/
```

```
//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ))

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// no light


else if ( ( (b21 == '0' && b22 <= '7') && (b31 == '0' && b32 <= '7') ) && (s6 == 'C' || s7 == 'C' )
&& ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 ==
'0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0'
&& us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//1m from light hit something
```

```
else if ( (b21 == '9') && (b31 == '9')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 == '9' && b31 < '9')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 < '9') && (b31 == '9')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ((b21 == '0') && (b22 <= '7')) && ((b31 == '0') && (b32 <= '7')) && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

//1m from light

else if ( (b21 == '9') && (b31 == '9')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '9' && b31 < '9')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 < '9') && (b31 == '9')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Docked at conveyor switch 9
//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//50cm from light hit something

else if ( (b21 == '9') && (b31 == '9')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >=
'0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0'
&& us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}
```

```
else if ( (b21 == '9' && b31 < '9')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 < '9') && (b31 == '9')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}
}
return err;
}
```

LightLineSimRMConveyorReverseFusionLow.c

```
//follow light - no light follow line Reverse dir
// == 9 stops in front of light loads sideleft
//doesnot go back to high
//change all 9's

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMConveyorReverseFusionLow.h"
#include "LightOwnLineSimRMConveyorReverseFusionHigh.h"

//ReverseSideLeft

//chkd

int LightLineSimRMConveyorReverseFusionLow(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe


                        b1
                        b0      S
                        b2
                l2 l0 l1
```

```
                                W
*/

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Reverse
   //b2

   char b21 = ssp->pinputbuff[32] ;
   char b22 = ssp->pinputbuff[33] ;

   //r3
   char b31 = ssp->pinputbuff[44] ;
   char b32 = ssp->pinputbuff[45] ;


   printf("\nb21 = %c \n" , b21);
   printf("b22 = %c \n" , b22);

   printf("\nb31 = %c \n" , b31);
   printf("b32 = %c \n" , b32);

   sleep(400) ;

    //Ultrasonic Process Reverse

   Flush(ssp) ;

   Write(ssp, 17, "#%+ULTSW       *\\");                    //S
   sleep(400) ;
   err = Read(ssp, 17);

   char us1 = ssp->pinputbuff[10] ;
   char us2 = ssp->pinputbuff[11] ;
   char us3 = ssp->pinputbuff[12] ;

   printf("\nus1 = %c \n" , us1);
   printf("us2 = %c \n" , us2);
   printf("us3 = %c \n" , us3);
```

```
    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    //sleep(400) ;
//Analysis        8 = Light

//Reverse           left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0'))

Reverse(ssp) ;

//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseright(ssp) ;

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseleft(ssp) ;

//No light <= 7 follow line
```

```
else if ( ( (b21 == '0' && b22 <= '7'  && b31 == '0' && b32 <= '7') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0'))

{
    Chill(ssp) ;
    LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s9 = C

else if ( (b21 == '9') && (b31 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

{
    Reverse(ssp) ;

}

else if ( (b21 == '9' && b31 < '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

{
    Reverseleft(ssp) ;

}

else if ( (b21 < '9') && (b31 == '9') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

{
    Reverseright(ssp) ;

}

/*Limit fusion
Following Light & switch closes - hit something
6 7*/

//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right
```

```
else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// no light


else if ( ( (b21 == '0' && b22 <= '7') && (b31 == '0' && b32 <= '7') ) && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//1m from light hit something

else if ( (b21 == '9') && (b31 == '9')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 == '9' && b31 < '9')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
```

```
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 < '9') && (b31 == '9')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light
```

```
else if ( ((b21 == '0') && (b22 <= '7')) && ((b31 == '0') && (b32 <= '7')) && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

//1m from light

else if ( (b21 == '9') && (b31 == '9')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '9' && b31 < '9')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 < '9') && (b31 == '9')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Docked at conveyor switch 9
//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    ConveyorChill(ssp);
    a = 0;
}
```

```
//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//50cm from light hit something

else if ( (b21 == '9') && (b31 == '9')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >=
'0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0'
&& us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 == '9' && b31 < '9')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') ||
(us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 < '9') && (b31 == '9')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0')
|| (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' &&
us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
```

```
    a = 0;
}
}
return err;
}
```

LightLineSimRMLatheReverseFusionLow.c

```
//follow light - no light follow line Reverse dir
// == 9 stops in front of light loads sideleft
//doesnot go back to high

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMLatheReverseFusionLow.h"
#include "LightOwnLineSimRMLatheReverseFusionHigh.h"
#include "LightLineSimRMLatheSideleftFusionLow.h"

//ReverseSideLeft

int LightLineSimRMLatheReverseFusionLow(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe


                                    b1
                                    b0      S
                                    b2
                               l2 l0 l1
                                  W
*/

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)

   {
   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Reverse
   //b2

   char b21 = ssp->pinputbuff[32] ;
```

```
char b22 = ssp->pinputbuff[33] ;

//r3

char b31 = ssp->pinputbuff[44] ;
char b32 = ssp->pinputbuff[45] ;

printf("\nb21 = %c \n" , b21);
printf("b22 = %c \n" , b22);

printf("\nb31 = %c \n" , b31);
printf("b32 = %c \n" , b32);

sleep(400) ;

 //Ultrasonic Process Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //S
sleep(400) ;
err = Read(ssp, 17);

char us1 = ssp->pinputbuff[10] ;
char us2 = ssp->pinputbuff[11] ;
char us3 = ssp->pinputbuff[12] ;

printf("\nus1 = %c \n" , us1);
printf("us2 = %c \n" , us2);
printf("us3 = %c \n" , us3);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("\ns6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
```

```
   printf("\n");

//Analysis       8 = Light

//Reverse           left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0'))

Reverse(ssp) ;

//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseright(ssp) ;

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseleft(ssp) ;

//No light <= 7 follow line

else if ( ( (b21 == '0' && b22 <= '7'  && b31 == '0' && b32 <= '7') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0'))

{
    Chill(ssp) ;
    LightOwnLineSimRMLatheReverseFusionHigh(ssp) ;
    a = 0;
}

//1m in front of light. Loading Sideleft

else if ( (b21 == '9') || (b31 == '9') )

{
    Chill(ssp) ;
    LightLineSimRMLatheSideleftFusionLow(ssp) ;
    printf("\n1m in front of light. Loading Sideleft.\n") ;
    a = 0;
}

/*Limit fusion
Following Light & switch closes - hit something
```

```
6 7*/

//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ))

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// no light

else if ( ( (b21 == '0' && b22 <= '7') && (b31 == '0' && b32 <= '7') ) && (s6 == 'C' || s7 == 'C' )
&& ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 ==
'0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0'
&& us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
```

```
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ((b21 == '0') && (b22 <= '7')) && ((b31 == '0') && (b32 <= '7')) && ( (us1 == '0' && us2
== '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 ==
'0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMLatheReverseFusionHigh(ssp) ;
    a = 0;
}

}
return err;
}
```

LightLineSimRMLatheSideleftFusionHigh.c

```
//follow light - no light follow line Sideleft dir
//use for conveyor to lathe

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMLatheSideleftFusionHigh.h"
#include "LightLineSimRMLatheSideleftFusionLow.h"
#include "LightOwnLineSimRMLatheSideleftFusionHigh.h"

//Reverse & Sideleft
//sort out 1

int LightLineSimRMLatheSideleftFusionHigh(serial_struct* ssp)


{
   aErr err = aErrNone;
   int b = 1;

   while(b==1)
   {
   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideleft

   //l0

   char l01 = ssp->pinputbuff[8] ;
   char l02 = ssp->pinputbuff[9] ;

   //l1

   char l11 = ssp->pinputbuff[20] ;
   char l12 = ssp->pinputbuff[21] ;

   printf("\n\nl01 = %c \n" , l01);
   printf("l02 = %c \n" , l02);

   printf("\nl11 = %c \n" , l11);
   printf("l12 = %c \n" , l12);
```

```
    sleep(400) ;

     //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");              //W
    sleep(400) ;
    err = Read(ssp, 17);

    char uw1 = ssp->pinputbuff[10] ;
    char uw2 = ssp->pinputbuff[11] ;
    char uw3 = ssp->pinputbuff[12] ;

    printf("\nuw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

    sleep(500) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s1 = ssp->pinputbuff[17] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[10] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s1 = %c \n" , s1);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis       9 = Light

//Sideleft     225  m1 l1       l0 m4     135

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') && s1 == 'O' )
```

```
Sideleft(ssp) ;

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft225(ssp) ;

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft135(ssp) ;

//No light line follow no obstacle

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') && s1 == 'O')

{
    Chill(ssp) ;
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//light == 8 change to low

else if ( (l01 == '0' && l02 == '8' && l11 == '0' && l12 == '8') && ( (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 ==
'0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) && s1 == 'O' )

{
    Chill(ssp) ;
    LightLineSimRMLatheSideleftFusionLow(ssp) ;
    b = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (l01 == '1' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (l01 == '1' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )
```

```
{
    Sideleft135(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (l01 == '0' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft225(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

//Lmt
//6 7

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 ==
'0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 >
'0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' )&& (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//No light
```

```
else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '1' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '1' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '0' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//1

//light 50cm away

else if ( (l01 == '1' || l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}
```

```
//more light l01 135
else if ( (l01 == '1' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C' ) && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )
```

```
{
    LatheChillLight(ssp);
    b = 0;
}

//No light

else if ( ((l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8')) && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//No light

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//light 50cm away

else if ( (l01 == '1' || l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l01 135

else if ( (l01 == '1' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

}
return err;
}
```

LightLineSimRMLatheSideleftFusionLow.c

```
//follow light - no light follow line Sideleft dir

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMLatheSideleftFusionLow.h"
#include "LightOwnLineSimRMLatheSideleftFusionHigh.h"

//Reverse & Sideleft

int LightLineSimRMLatheSideleftFusionLow(serial_struct* ssp)

{
   aErr err = aErrNone;
   int b = 1;

   while(b==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideleft

   //l0

   char l01 = ssp->pinputbuff[8] ;
   char l02 = ssp->pinputbuff[9] ;

   //l1

   char l11 = ssp->pinputbuff[20] ;
   char l12 = ssp->pinputbuff[21] ;

   printf("\n\nl01 = %c \n" , l01);
   printf("l02 = %c \n" , l02);

   printf("\nl11 = %c \n" , l11);
   printf("l12 = %c \n" , l12);

   sleep(400) ;

    //Ultrasonic Process Forward

   Flush(ssp) ;
   Write(ssp, 17, "#%+ULTSW       *\\");              //W
   sleep(400) ;
```

```
    err = Read(ssp, 17);

    char uw1 = ssp->pinputbuff[10] ;
    char uw2 = ssp->pinputbuff[11] ;
    char uw3 = ssp->pinputbuff[12] ;

    printf("\nuw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

    sleep(500) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s1 = ssp->pinputbuff[17] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[10] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s1 = %c \n" , s1);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis       9 = Light

//Sideleft     225  m1 l1       l0 m4     135

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') && s1 == 'O' )

Sideleft(ssp) ;

//More light toward r12 right

else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')
Sideleft225(ssp) ;
```

```
//More light toward r02 left

else if ( (l02 == '8' && l12 < '8' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft135(ssp) ;

//No light line follow no obstacle

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') && s1 == 'O')

{
    Chill(ssp) ;
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//approx 1m in front of light go sideleft
//only stops when docked s1 = C

else if ( (l01 == '9' && l11 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp) ;

else if ( (l01 == '9' && l11 < '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft135(ssp) ;

else if ( (l01 < '9' && l11 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft225(ssp) ;

//Lmt
//6 7

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r12 right
```

```
else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}
//More light toward r02 left

else if ( (l02 == '8' && l12 < '8' )&& (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//No light

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '9' && l11 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '9' && l11 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}
```

```
else if ( (l01 < '9' && l11 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//1

//light (1m away)

else if ( (l01 == '9' || l11 == '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//more light l01 135

else if ( (l01 == '9' && l11 < '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//more light l11 225

else if ( (l01 < '9' && l11 == '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
     LatheChillLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s1 == 'C' ) && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//More light toward r02 left

else if ( (l02 == '8' && l12 < '8' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//No light

else if ( ((l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7')) && (s1 == 'C') && ( (uw1 >= '1'
&& uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2
> '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//More light toward r12 right

else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (l02 == '8' && l12 < '8' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && ( (uw1 == '0' && uw2 ==
'0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3
== '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//light 1m away

else if ( (l01 == '9' || l11 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' &&
uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' &&
uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l01 135
```

```
else if ( (l01 == '9' && l11 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l11 225

else if ( (l01 < '9' && l11 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

}
return err;
}
```

LightOwnLineSimConveyorRMForwardFusionHigh.c

```
//return to light follow senses light
//search for line if cant find it

//used to line follow for low light as well
//goes to high light
//high light goes to low if low

//>7 change to high no low

//chked

// Foward Analysis                          front

/*                                      f1
                                        f0
                                        f2 */

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LightOwnLineSimConveyorRMForwardFusionHigh.h"
#include "LightLineSimConveyorRMForwardFusionHigh.h"

int LightOwnLineSimConveyorRMForwardFusionHigh(serial_struct* ssp)
```

```
{
   aErr err = aErrNone;
   int c = 1;

   while(c==1)
   {

   //Foward                          LINER

    Flush(ssp) ;

    Write(ssp, 17, "#%+LINER       *\\");              //change to LITEL
    sleep(400) ;

    err = Read(ssp, 50);

   char f0 = ssp->pinputbuff[9] ;
   char f1 = ssp->pinputbuff[21] ;
   char f2 = ssp->pinputbuff[33] ;

   printf("\nf0 = %c \n" , f0);
   printf("f1 = %c \n" , f1);
   printf("f2 = %c \n" , f2);

   sleep(400) ;

   //Light board forward

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //r3

   char f31 = ssp->pinputbuff[44] ;
   char f32 = ssp->pinputbuff[45] ;

   printf("\nf21 = %c \n" , f21);
   printf("f22 = %c \n" , f22);

   printf("\nf31 = %c \n" , f31);
   printf("f32 = %c \n" , f32);
```

```
     sleep(400) ;

      //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    sleep(400) ;
    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSS       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

//normal foward
//f0 on black f1 f2 go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0')  )

Forward(ssp);

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))
```

```
Right(ssp) ;

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Right(ssp) ;

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Left(ssp) ;

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

Left(ssp);

//No line search for line

//All on black

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//Lmt switch
//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7 == 'C') )
```

```
{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    c = 0;
}

//l2 l0 l1 on white

else if ((f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ))

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    c = 0;
}

//Offline and obstacle crash

//All on black

else if (  (f1 < '9' && f0 < '9' && f2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//On line obstacle detected waiting for clear

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 ==
'0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' ) && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More Chills
//s6 s7

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >= '0'
&& un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0')
|| (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
    c = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s6 == 'C' || s7 == 'C') && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >=
'0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 >
'0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp) ;
    c = 0;
}

//l0/l2 on black l1 on white go Left
```

```
else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStop(ssp) ;
    c = 0;
}

//Docking at RM switch 8

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
```

```
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

else if ( (f1 < '9' && f0 < '9' && f2 < '9')&& (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3
>= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1
== '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//All on white

else if ( (f1 == '9' && f0 == '9' && f2 == '9')&& (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' &&
un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') ||
(un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//Check for light in fwd dir return if found

//Forward     left     m1  2       3   m2       right

//Equal light

if ( (f22 > '7' && f32 > '7' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3 >=
'0') )

{
    Forward(ssp) ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

//More light toward r32 right
```

```
else if ( (f22 <= '7' && f32 > '7' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

{
    Right(ssp) ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

//More light toward r22 left

else if ( (f22 > '7' && f32 <= '7' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

{
    Left(ssp) ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( (f21 == '1') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )
{
    Forward(ssp) ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

else if ( (f21 == '1' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    Left(ssp) ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

else if ( (f21 == '0') && (f31 == '1') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    Right(ssp) ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

//Limit Switch
//6 7

//Equal light
```

```
if ( (f22 > '7' && f32 > '7' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' )  && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    c = 0 ;
}

//More light toward r32 right

else if ( (f22 <= '7' && f32 > '7' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    c = 0;
}

//More light toward r22 left

else if ( (f22 > '7' && f32 <= '7' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    c = 0;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '1') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    c = 0;
}

else if ( ( (f21 == '1' && f31 == '0') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
```

```
    EmergencyStopLight(ssp);
    c = 0;
}

else if ( ( (f21 == '0') && (f31 == '1') && (s6 == 'C' || s7 == 'C' ) && (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    EmergencyStopLight(ssp);
    c = 0;
}

//8 RM docking

//Equal light

if ( (f22 > '7' && f32 > '7' && f21 == '0' && f31 == '0') && (s8 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}
//More light toward r32 right

else if ( (f22 <= '7' && f32 > '7' && f21 == '0' && f31 == '0') && (s8 == 'C') && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//More light toward r22 left

else if ( (f22 > '7' && f32 <= '7' && f21 == '0' && f31 == '0') && (s8 == 'C')&& ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C
```

```
else if ( ( (f21 == '1') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0')
|| (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0'
&& un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

else if ( ( (f21 == '1' && f31 == '0') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0') ||
(un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' &&
un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RMChill(ssp);
    c = 0;
}

else if ( ( (f21 == '0') && (f31 == '1') && (s8 == 'C') && (un1 >= '1' && un2 >= '0' && un3 >= '0')
|| (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0'
&& un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
   RMChill(ssp);
   c = 0;
}
//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 > '7' && f32 > '7' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

//More light toward r32 right

else if ( (f22 <= '7' && f32 > '7' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

//More light toward r22 left

else if ( (f22 > '7' && f32 <= '7' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

//approx 0.5 m in front of light
//only stops when docked s8 = C

else if ( ( (f21 == '1') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

else if ( ( (f21 == '1' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
    c = 0 ;
}

else if ( ( (f21 == '0') && (f31 == '1') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) ))

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimConveyorRMForwardFusionHigh(ssp) ;
```

```
    c = 0 ;
}


}
return err;
}
```

LightOwnLineSimLatheConveyorSiderightFusionHigh.c

```
//return to light follow senses light
//search for line if cant find it

//used to line follow for low light as well
//goes to high light
//high light goes to low if low

//>7 change to high no low

//chked
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LightOwnLineSimLatheConveyorSiderightFusionHigh.h"
#include "LightLineSimLatheConveyorSiderightFusionHigh.h"

int LightOwnLineSimLatheConveyorSiderightFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
  int a = 1;

   while(a==1)
   {

   //Sideright                          LINER

    Flush(ssp) ;

    Write(ssp, 17, "#%+LINEL       *\\");
    sleep(400) ;

    err = Read(ssp, 50);

   char r0 = ssp->pinputbuff[9] ;
   char r1 = ssp->pinputbuff[21] ;
   char r2 = ssp->pinputbuff[33] ;

   printf("\nr0 = %c \n" , r0);
```

```
printf("r1 = %c \n" , r1);
printf("r2 = %c \n" , r2);

sleep(500) ;

 //light board

 Flush(ssp) ;
Write(ssp, 17, "#%+LITEL       *\\");
sleep(400) ;
err = Read(ssp, 50);

//Sideright
//r0

char r01 = ssp->pinputbuff[8] ;
char r02 = ssp->pinputbuff[9] ;

//r1

char r11 = ssp->pinputbuff[20] ;
char r12 = ssp->pinputbuff[21] ;

printf("\nr01 = %c \n" , r01);
printf("r02 = %c \n" , r02);

printf("\nr11 = %c \n" , r11);
printf("r12 = %c \n" , r12);

 sleep(400) ;

//Ultrasonic Process SideRight

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");                    //E
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
printf("ue3 = %c \n" , ue3);

sleep(400) ;

//Limit Switch Analysis
```

```
    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s0 = ssp->pinputbuff[9] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;

    printf("\ns0 = %c \n" , s0);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("\n");

//Line Analysis
//r0 on black r1 r2 on white && u >= 1 go Sideright              m2  r2 r0 r1

if ( (r0 < '9') && (r1 == '9') && (r2 == '9') && ue1 >= '1' && ue2 >= '0' && ue3 >= '0')

Sideright(ssp);

//r1 on black r0 r2 on white && u >= 1 go Sideright45

else if ( (r1 < '9') && (r0 == '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright45(ssp) ;

//r2 on black r0 r1 on white && u >= 1 go Sideright315

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0')  )

Sideright315(ssp) ;

//r0/r1 on black r2 on white && u >= 1 go Sideright315

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//r0/r2 on black r1 on white && u >= 1 go Sideright315

else if ( ( r0 < '9' && r1 == '9'&& r2 < '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp);

//Main Chills

//l2 l0 l1 on black
```

```
else if (( r1 < '9' && r0 < '9' && r2 < '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//l2 l0 l1 on white
else if (( r1 == '9' && r0 == '9' && r2 == '9') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (s6 == 'C' || s7 == 'C'))

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' ))
{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//l2 l0 l1 on white
```

```
else if ( (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear
//l0 on black l1 l2 on

else if ( (r0 < '9' && r1 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1
== '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 >
'0' && ue2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9')&& ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 ==
'0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0'
&& ue2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}
//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9'&& r2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Lmt integration s0

//l0 on black l1 l2 on white go fwd

if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (s0 == 'C') && (r1 < '9' && r0 == '9'&& r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (s0 == 'C') && (r2 < '9' && r0 == '9' && r1 == '9')&& ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right
```

```
else if ( (s0 == 'C')&& (r0 < '9' && r1 < '9' && r2 == '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}


//l0/l2 on black l1 on white go Left

else if ( (s0 == 'C') && (r0 < '9' && r1 == '9' && r2 < '9') && ((ue1 >= '1' && ue2 >= '0' && ue3
>= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1
== '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequence(ssp);
    a = 0;
}

//More Chills
//s6 s7

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//Check for high light

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >=
'0'))

{
    Sideright(ssp) ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r12 right

else if ((r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

{
    Sideright45(ssp) ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r02 left
```

```
else if ( (r02 > '7' && r12 <= '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

{
    Sideright315(ssp) ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}


//approx 0.5 m in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Sideright(ssp) ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

else if ( (r01 == '1' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Sideright315(ssp) ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Sideright45(ssp) ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

//Lmt
//6 7

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
```

```
}

//More light toward r12 right

else if ( (r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}
//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7'  && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (r01 == '1' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') ||
(ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
```

```
    EmergencyStopLight(ssp);
    a = 0;
}

//dont need for lathe to RM must stop .5m in front of light
//0

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' && ue2
>= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3
> '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7'  && r01 == '0' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' &&
ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&&
ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') ||
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}
```

```
else if ( (r01 == '1' && r11 == '0') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && (s0 == 'C') && ( (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    TableSequenceLight(ssp);
    a = 0;
}

//Obstacle detected. Waiting for clear

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r02 left
```

```
else if ( (r02 > '7' && r12 <= '7'  && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (r01 == '1' && r11 == '1')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

else if ( (r01 == '1' && r11 == '0')  && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}

else if ( (r01 == '0' && r11 == '1') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheConveyorSiderightFusionHigh(ssp);
    a = 0;
}
}

return err;

}
```

LightOwnLineSimLatheRMSiderightFusionHigh.c

```
//return to light follow senses light
//search for line if cant find it
//detect line in fwd dir

//chked

//going sideright sees black line starts going foward

/*                                  E
                                  r2 r0 r1

                              f1
                        N     f0
                              f2
  */

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LightOwnLineSimLatheRMSiderightFusionHigh.h"
#include "LightLineSimLatheRMSiderightFusionHigh.h"
#include "LightOwnLineSimConveyorRMForwardFusionHigh.h"

int LightOwnLineSimLatheRMSiderightFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   //1st board going right                LINEL

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LINEL       *\\");
   sleep(400) ;

   err = Read(ssp, 50);

   char r0 = ssp->pinputbuff[9] ;
   char r1 = ssp->pinputbuff[21] ;
   char r2 = ssp->pinputbuff[33] ;

   printf("\nr0 = %c \n" , r0);
   printf("r1 = %c \n" , r1);
```

```
printf("r2 = %c \n" , r2);

 sleep(500) ;

 //2nd board foward                          LINER

 Flush(ssp) ;

 Write(ssp, 17, "#%+LINER       *\\");             //change to LITEL
 sleep(400) ;

 err = Read(ssp, 50);

char f0 = ssp->pinputbuff[9] ;
char f1 = ssp->pinputbuff[21] ;
char f2 = ssp->pinputbuff[33] ;

printf("\nf0 = %c \n" , f0);
printf("f1 = %c \n" , f1);
printf("f2 = %c \n" , f2);

sleep(400) ;

//light board

 Flush(ssp) ;

Write(ssp, 17, "#%+LITEL       *\\");
sleep(400) ;
err = Read(ssp, 50);

//Sideright
 //r0

 char r01 = ssp->pinputbuff[8] ;
 char r02 = ssp->pinputbuff[9] ;

 //r1

 char r11 = ssp->pinputbuff[20] ;
 char r12 = ssp->pinputbuff[21] ;


 printf("\nr01 = %c \n" , r01);
 printf("r02 = %c \n" , r02);

 printf("\nr11 = %c \n" , r11);
 printf("r12 = %c \n" , r12);

  sleep(400) ;
```

```
 //Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process SideRight

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");          //east
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
printf("ue3 = %c \n" , ue3);

//Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSS       *\\");
sleep(400) ;
err = Read(ssp, 17);

char un1 = ssp->pinputbuff[10] ;
char un2 = ssp->pinputbuff[11] ;
char un3 = ssp->pinputbuff[12] ;

printf("\nun1 = %c \n" , un1);
```

```
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

//Sideright Analysis                        m2  r2 r0 r1

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright(ssp);

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp) ;

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright45(ssp) ;

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright315(ssp);

// Foward Analysis                                front

/*                                                f1
                                                  f0
                                                  f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0')  )
```

```
{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
   Right(ssp) ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}
```

```
//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') )

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Right(ssp) ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward
else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
```

```
{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9'  && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r1 < '9'  && r0 == '9' && r2 == '9'&& f2 == '9' && f0 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
```

```
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Left(ssp) ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r1 on black f0 f1 r2 on white go Forward
```

```
else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Left(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))
{
    Forward(ssp);
```

```
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}


//f1 [r0 r1 r2] on black go Forward
```

```
else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9'  && r2 < '9' && r0 < '9' && r1 == '9' && f2 == '9' && f1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//Chill time - obstacle < 1m or crashed - obstacle

//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
```

```
    printf("\nSearching for line :) \n");
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (un1 >=
'1' && un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//f2 f0 f1 on black r1 r2 r0 on white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//f2 f0 f1 on white r1 r2 r0 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//Lmt switch

//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7
== 'C'|| s8 == 'C'|| s9 == 'C') )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white
```

```
else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C') )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9')&& (s6 == 'C' || s7 == 'C'|| s8 == 'C'|| s9 == 'C') )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
```

```
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )  )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )  )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )  )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9') && (r1 == '9')&& (r2 < '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && ( (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )  )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

// Foward Analysis                              front

/*                                          f1
                                            f0
                                            f2 */
 //normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward
```

```
if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward
```

```
else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' && r1 < '9'  && r0 == '9' && r2 == '9' && f2 == '9' && f0 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward
```

```
else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}


//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9' && r2 < '9' && r0 < '9' && r1 == '9'&& f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More Chills
//s6 s7
```

```
if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
{
    EmergencyStop(ssp) ;
    a = 0;
}
```

```
//More Chills
//s8 dock at RM

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s8 == 'C' || s9 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}



//l0/l2 on black l1 on white go Left
```

```
else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s8 == 'C' || s9 == 'C' )&& ((ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//Check for high light

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

{
    Sideright(ssp) ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r12 right

else if ((r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

{
    Sideright45(ssp) ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0'))

{
    Sideright315(ssp) ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//approx 50 cm in front of light load forward
//ConveyorRM = LatheRM
//only stops when docked s1 = C

else if (r01 == '1' || r11 == '1')
{
   Chill(ssp);
```

```
   LightLineSimConveyorRMForwardFusionHigh(ssp);
   printf("\nApprox 0.5 m in front of Light. Loading Forward\n") ;
   a = 0 ;
}

//Lmt
//6 7

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 >
'0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 ==
'0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Obstacle detected. Waiting for clear

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

}

return err;

}
```

LightLineSimRMConveyorReverseFusionHigh.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightDock.h"
#include "LightLineSimRMConveyorReverseFusionHigh.h"
#include "LightLineSimRMConveyorReverseFusionLow.h"
#include "LightOwnLineSimRMConveyorReverseFusionHigh.h"
```

```
//follow light - no light follow line Reverse dir
//light == 8 change to low
//light == 9 change to high

//s=9 dock conveyor

//chked

int LightLineSimRMConveyorReverseFusionHigh(serial_struct* ssp)

/*
                                        b1
                                        b0      S
                                        b2
                                */
{

    aErr err = aErrNone;
    int a = 1;

    while(a==1)
    {

    Flush(ssp) ;

    Write(ssp, 17, "#%+LITER       *\\");
    sleep(400) ;
    err = Read(ssp, 50);

    //Reverse
    //b2

    char b21 = ssp->pinputbuff[32] ;
    char b22 = ssp->pinputbuff[33] ;

    //r3

    char b31 = ssp->pinputbuff[44] ;
    char b32 = ssp->pinputbuff[45] ;


    printf("\nb21 = %c \n" , b21);
    printf("b22 = %c \n" , b22);

    printf("\nb31 = %c \n" , b31);
    printf("b32 = %c \n" , b32);

    sleep(400) ;

     //Ultrasonic Process Reverse
```

```
    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");                    //S
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    //sleep(400) ;


//Analysis       9 = Light

//Reverse           left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0'))

Reverse(ssp) ;

//More light toward r32 right
```

```
else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseright(ssp) ;

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseleft(ssp) ;

//No light <= 7 no obstacle follow line

else if ( ( (b21 == '0' && b22 <= '7'  && b31 == '0' && b32 <= '7') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0'))

{
    Chill(ssp) ;
   LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

//light == 8 change to low

else if ( (b21 == '0' && b22 == '8' && b31 == '0' && b32 == '8') && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') ||
(us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    LightLineSimRMConveyorReverseFusionLow(ssp);
    a = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s9 = C

else if ( (b21 == '1') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )
Reverse(ssp) ;

else if ( (b21 == '1' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseleft(ssp) ;

else if ( (b21 == '0') && (b31 == '1') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

Reverseright(ssp) ;

/*Limit fusion
```

```
Following Light & switch closes - hit something
6 7*/

//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ))

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// no light

else if ( ( (b21 == '0' && b22 <= '7') && (b31 == '0' && b32 <= '7') ) && (s6 == 'C' || s7 == 'C' )
&& ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 ==
'0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0'
&& us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}
```

```
//50cm from light hit something

else if ( (b21 == '1') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 == '1' && b31 == '0')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

else if ( (b21 == '0') && (b31 == '1')  && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light - load line follow

else if ( ((b21 == '0') && (b22 <= '7')) && ((b31 == '0') && (b32 <= '7')) && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMConveyorReverseFusionHigh(ssp) ;
    a = 0;
}

else if ( (b21 == '1') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


else if ( (b21 == '1' && b31 == '0')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

else if ( (b21 == '0') && (b31 == '1')   && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
```

```
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Docked at conveyor switch 9
//Equal light

if ( (b22 == '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1' &&
us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&&
us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r32 right

else if ( (b22 < '9' && b32 == '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '9' && b32 < '9' && b21 == '0' && b31 == '0') && (s9 == 'C') && ( (us1 >= '1'
&& us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    ConveyorChill(ssp);
    a = 0;
}

//50cm from light hit something

else if ( (b21 == '1') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >=
'0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0'
&& us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}
```

```
else if ( (b21 == '1' && b31 == '0')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

else if ( (b21 == '0') && (b31 == '1')  && (s9 == 'C') && ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    ConveyorChill(ssp);
    a = 0;
}

}
return err;
}
```

LightOwnLineSimRMLatheReverseFusionHigh.c

```
//follow line & check for light Reverse dir

//add ultra if necessary to searching line

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightOwnLineSimRMLatheReverseFusionHigh.h"
#include "LightLineSimRMLatheReverseFusionHigh.h"
#include "LightOwnLineSimRMLatheSideleftFusionHigh.h"

//ReverseSideLeft

int LightOwnLineSimRMLatheReverseFusionHigh(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe

                                    b1
                                    b0      S
                                    b2
                              l2 l0 l1
                                 W
*/

{
   aErr err = aErrNone;
```

```
int c = 1;

while(c==1)
{

Flush(ssp) ;

Write(ssp, 17, "#%+LINER       *\\");          //change to LITER
sleep(400) ;

err = Read(ssp, 50);

char b0 = ssp->pinputbuff[9] ;
char b1 = ssp->pinputbuff[21] ;
char b2 = ssp->pinputbuff[33] ;

printf("b0 = %c \n" , b0);
printf("b1 = %c \n" , b1);
printf("b2 = %c \n" , b2);

sleep(400) ;
 //Sideleft

Flush(ssp) ;

Write(ssp, 17, "#%+LINEL       *\\");
sleep(400) ;

err = Read(ssp, 50);

char l0 = ssp->pinputbuff[9] ;
char l1 = ssp->pinputbuff[21] ;
char l2 = ssp->pinputbuff[33] ;

printf("l0 = %c \n" , l0);
printf("l1 = %c \n" , l1);
printf("l2 = %c \n" , l2);

sleep(400) ;

//Light board

Flush(ssp) ;

Write(ssp, 17, "#%+LITER       *\\");
sleep(400) ;
err = Read(ssp, 50);

//Reverse
//b2
```

```
char b21 = ssp->pinputbuff[32] ;
char b22 = ssp->pinputbuff[33] ;

//r3

char b31 = ssp->pinputbuff[44] ;
char b32 = ssp->pinputbuff[45] ;

printf("\nb21 = %c \n" , b21);
printf("b22 = %c \n" , b22);

printf("\nb31 = %c \n" , b31);
printf("b32 = %c \n" , b32);

sleep(400) ;

//Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s1 = ssp->pinputbuff[17] ;
char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[10] ;
char s9 = ssp->pinputbuff[18] ;

printf("s1 = %c \n" , s1);
printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process Reverse

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");          //S
sleep(400) ;
err = Read(ssp, 17);

char us1 = ssp->pinputbuff[10] ;
char us2 = ssp->pinputbuff[11] ;
char us3 = ssp->pinputbuff[12] ;
```

```
    printf("us1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Ultrasonic Process SideLeft

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char uw1 = ssp->pinputbuff[10] ;
    char uw2 = ssp->pinputbuff[11] ;
    char uw3 = ssp->pinputbuff[12] ;

    printf("uw1 = %c \n" , uw1);
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

//Line following

//Reverse Analysis

//b0 on black b1 b2 on white go fwd

if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && (us1
>= '1' && us2 >= '0' && us3 >= '0'))

Reverse(ssp);

//b1 on black b0 b2 on white go Right

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b2 on black b0 b1 on white go Left

else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp) ;

//b0/b1 on black b2 on white go right
```

```
else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseleft(ssp) ;

//b0/b2 on black b1 on white go Left

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') &&
(us1 >= '1' && us2 >= '0' && us3 >= '0'))

Reverseright(ssp);

//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{

Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0 b2 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0-l1 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
```

```
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )
```

```
{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l1 b0 on black l1 l2 on white go fwd
```

```
else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
```

```
c = 0 ;
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l2 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )
```

```
{
Sideleft(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft135(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft225(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft135(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
c = 0 ;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') &&
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0'))

{
Sideleft225(ssp);
LightOwnLineSimRMLatheSideleftFusionHigh(ssp);
```

```
c = 0 ;
}

//Lmt Ultra integration

//No line search for line

//All on black

else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//All on white

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9')
{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//l2 l0 l1 on white b0 b1 b2 on black

else if (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9')

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//l2 l0 l1 on black b0 b1 b2 on white

else if (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9')

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//Offline and switch 6 7 closed - crashed
//All on black

else if ((l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' || s7 ==
'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
```

```
    c = 0;
}

//All on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 ==
'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    c = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && (s6 == 'C' ||
s7 == 'C'|| s8 == 'C'|| s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    c = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && (s6 == 'C' ||
s7 == 'C' || s8 == 'C' || s9 == 'C' || s1 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    c = 0;
}

//Offline and obstacle detected <1m stop

//All on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0' &&
uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0'
&& uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3 ==
'0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0'
&& us3 > '0' && us2 == '0') ))

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}
```

```
//All on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0'
&& us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') ||
(us1 == '0' && us3 > '0' && us2 == '0')))

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//l2 l0 l1 on white b0 b1 b2 on black

else if ( (l1 == '9' && l0 == '9' && l2 == '9' && b1 < '9' && b0 < '9' && b2 < '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3
== '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 ==
'0' && us3 > '0' && us2 == '0') ))

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//l2 l0 l1 on black b0 b1 b2 on white

else if ( (l1 < '9' && l0 < '9' && l2 < '9' && b1 == '9' && b0 == '9' && b2 == '9') && ( (uw1 == '0'
&& uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 >
'0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') || (us1 == '0' && us2 == '0' && us3
== '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 ==
'0' && us3 > '0' && us2 == '0')) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    c = 0;
}

//On line obstacle detected waiting for clear

//Reverse Analysis

//b0 on black b1 b2 on white go Reverse
```

```
if ( (b0 < '9') && (b1 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b1 on black b0 b2 on white go Reverseleft

else if ( (b1 < '9') && (b0 == '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b2 on black b0 b1 on white go Reverseright
else if ( (b2 < '9') && (b0 == '9') && (b1 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b1 on black b2 on white go Reverseleft

else if ( (b0 < '9') && (b1 < '9') && (b2 == '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//b0/b2 on black b1 on white go Reverseright

else if ( (b0 < '9') && (b1 == '9')&& (b2 < '9') && (l0 == '9') && (l1 == '9') && (l2 == '9') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//Sideleft analysis

//Normal Sideleft
//l0 on black l1 l2 on white go Sideleft

else if ( (l0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b0 < '9') && (l1 == '9') && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && (b0 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0 on black l1 l2 on white go fwd
```

```
else if ( (l0 < '9') && (l1 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b0 < '9')  && (l1 == '9') && (b1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9')  && (l2 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//l0-l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b2 < '9') && (b0 < '9')  && (l2 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b1 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9')  && (l1 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b2 < '9') && (b0 < '9') && (l1 == '9') && (b1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l1 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0-l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l0 < '9') && (l2 < '9') && (b1 < '9') && (b0 < '9') && (b2 < '9')&& (l1 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l0 == '9') && (l2 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0-b1-b2 on black l1 l2 on white go fwd

else if ( (l2 < '9')&& (b1 < '9') && (b0 < '9') && (b2 < '9') && (l1 == '9') && (l0 == '9') && ( (uw1
== '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' &&
uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b1 on black l1 l2 on white go fwd
```

```
else if ( (l1 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b0-b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (l0 == '9') && (b1 == '9')&& (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b1 < '9')&& (l0 == '9') && (b2 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b0 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b0 < '9') && (b2 < '9')&& (l0 == '9') && (b1 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

//l2 b1 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 b2 on black l1 l2 on white go fwd

else if ( (l2 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l1 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b1 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b1 < '9') && (b2 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 b2 on black l1 l2 on white go fwd

else if ( (l1 < '9') && (b2 < '9') && (b1 == '9')&& (l0 == '9') && (b0 == '9')&& (l2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft135
//l1 on black l0 l2 on white go Right
```

```
else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft225
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft135
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//Normal Sideleft225
//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (b1 == '9') && (b0 == '9') && (b2 == '9') && (
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

/*More Chills on line but switch closed
s6 s7
Reverse
*/
```

```
if ( (b0 < '9' && b1 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ( (us1 >= '1' && us2 >= '0'
&& us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') ||
(us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))  )

{
    EmergencyStop(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (b1 < '9' && b0 == '9' && b2 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStop(ssp);
    c = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (b2 < '9' && b0 == '9' && b1 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))
 {
    EmergencyStop(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (b0 < '9' && b1 < '9' && b2 == '9') &&  (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2
>= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3
> '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ))

{
    EmergencyStop(ssp) ;
    c = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (b0 < '9' && b1 == '9'&& b2 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >= '1' && us2 >=
'0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 >
'0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    EmergencyStop(ssp) ;
    c = 0;
}
```

```
/*More Chills
s1
Sideleft
*/

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s1 == 'C' ) && ( (uw1 == '1' && uw2 >= '0' && uw3
>= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') ||
(uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    c = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9' && l0 == '9' && l2 == '9')&& (s1 == 'C')&& ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    c = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9' && l0 == '9' && l1 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp);
    c = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9' && l1 < '9' && l2 == '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    c = 0;
}

//l0/l2 on black l1 on white go Left
```

```
else if ( (l0 < '9' && l1 == '9'&& l2 < '9') && (s1 == 'C') && ( (uw1 == '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChill(ssp) ;
    c = 0;
}

//Check for light

//Equal light

if ( (b22 > '7' && b32 > '7' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0') )

{
    Reverse(ssp) ;
    LightLineSimRMLatheReverseFusionHigh(ssp);
    c = 0;
}

//More light toward r32 right

else if ( (b22 <= '7' && b32 > '7' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))
{
    Reverseright(ssp) ;
    LightLineSimRMLatheReverseFusionHigh(ssp);
    c = 0;
}

//More light toward r22 left

else if ( (b22 > '7' && b32 <= '7' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3 >= '0'))

{
    Reverseleft(ssp) ;
    LightLineSimRMLatheReverseFusionHigh(ssp);
    c = 0;
}

//50cm in front of light. Loading Sideleft

else if ( (b21 == '1') || (b31 == '1') )

{
    Chill(ssp) ;
    LightLineSimRMLatheSideleftFusionHigh(ssp) ;
    printf("\n50cm in front of light. Loading Sideleft.\n") ;
```

```
    c = 0;
}

/*Limit fusion
Following Light & switch closes - hit something
6 7*/

//Equal light

if ( (b22 > '7' && b32 > '7' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1 >=
'1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 >
'0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0'))
)

{
    EmergencyStopLight(ssp);
    c = 0;
}

//More light toward r32 right

else if ( (b22 <= '7' && b32 > '7' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    c = 0;
}

//More light toward r22 left

else if ( (b22 > '7' && b32 <= '7' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    c = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 > '7' && b32 > '7' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3
== '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 ==
'0' && us3 > '0' && us2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimRMLatheReverseFusionHigh(ssp);
    c = 0;
}

//More light toward r32 right

else if ( (b22 <= '7' && b32 > '7' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimRMLatheReverseFusionHigh(ssp);
    c = 0;
}

//More light toward r22 left

else if ( (b22 > '7' && b32 <= '7' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimRMLatheReverseFusionHigh(ssp);
    c = 0;
}

}

return err;

}
```

LightLineSimRMLatheReverseFusionLow.c

```
//follow light - no light follow line Reverse dir
// == 9 stops in front of light loads sideleft
//doesnot go back to high

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMLatheReverseFusionLow.h"
```

```
#include "LightOwnLineSimRMLatheReverseFusionHigh.h"
#include "LightLineSimRMLatheSideleftFusionLow.h"

//ReverseSideLeft

int LightLineSimRMLatheReverseFusionLow(serial_struct* ssp)

/*    Going reverse then sees line and starts going sideleft
RM to Lathe


                                        b1
                                        b0      S
                                        b2
                                l2 l0 l1
                                   W
*/

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)

   {
   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Reverse
   //b2

   char b21 = ssp->pinputbuff[32] ;
   char b22 = ssp->pinputbuff[33] ;

   //r3

   char b31 = ssp->pinputbuff[44] ;
   char b32 = ssp->pinputbuff[45] ;

   printf("\nb21 = %c \n" , b21);
   printf("b22 = %c \n" , b22);

   printf("\nb31 = %c \n" , b31);
   printf("b32 = %c \n" , b32);

   sleep(400) ;

    //Ultrasonic Process Reverse
```

```
    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSW       *\\");                    //S
    sleep(400) ;
    err = Read(ssp, 17);

    char us1 = ssp->pinputbuff[10] ;
    char us2 = ssp->pinputbuff[11] ;
    char us3 = ssp->pinputbuff[12] ;

    printf("\nus1 = %c \n" , us1);
    printf("us2 = %c \n" , us2);
    printf("us3 = %c \n" , us3);

    sleep(400) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis       8 = Light

//Reverse           left      m3  b2          b3    m4        right

//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' && us3
>= '0'))

Reverse(ssp) ;

//More light toward r32 right
```

```
else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseright(ssp) ;

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && (us1 >= '1' && us2 >= '0' &&
us3 >= '0'))

Reverseleft(ssp) ;

//No light <= 7 follow line

else if ( ( (b21 == '0' && b22 <= '7'  && b31 == '0' && b32 <= '7') ) && (us1 >= '1' && us2 >= '0'
&& us3 >= '0'))

{
    Chill(ssp) ;
    LightOwnLineSimRMLatheReverseFusionHigh(ssp) ;
    a = 0;
}

//1m in front of light. Loading Sideleft

else if ( (b21 == '9') || (b31 == '9') )

{
    Chill(ssp) ;
    LightLineSimRMLatheSideleftFusionLow(ssp) ;
    printf("\n1m in front of light. Loading Sideleft.\n") ;
    a = 0;
}

/*Limit fusion
Following Light & switch closes - hit something
6 7*/

//Equal light

if ( (b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (us1
>= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' &&
us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2
== '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r32 right
```

```
else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ))

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8'&& b21 == '0' && b31 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0'
&& us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' &&
us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

// no light

else if ( ( (b21 == '0' && b22 <= '7') && (b31 == '0' && b32 <= '7') ) && (s6 == 'C' || s7 == 'C' )
&& ( (us1 >= '1' && us2 >= '0' && us3 >= '0') || (us1 == '0' && us2 == '0' && us3 == '0') || (us1 ==
'0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0'
&& us2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( ( b22 == '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' &&
us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1
== '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right
```

```
else if ( (b22 < '8' && b32 == '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (b22 == '8' && b32 < '8' && b21 == '0' && b31 == '0') && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ((b21 == '0') && (b22 <= '7')) && ((b31 == '0') && (b32 <= '7')) && ( (us1 == '0' && us2 == '0' && us3 == '0') || (us1 == '0' && us2 > '0'&& us3 > '0') || (us1 == '0' && us2 > '0' && us3 == '0') || (us1 == '0' && us3 > '0' && us2 == '0') ) )

{
    Chill(ssp) ;
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMLatheReverseFusionHigh(ssp) ;
    a = 0;
}

}
return err;
}
```

LightLineSimRMLatheSideleftFusionHigh.c

```
//follow light - no light follow line Sideleft dir
//use for conveyor to lathe

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMLatheSideleftFusionHigh.h"
#include "LightLineSimRMLatheSideleftFusionLow.h"
#include "LightOwnLineSimRMLatheSideleftFusionHigh.h"
```

```
//Reverse & Sideleft
//sort out 1

int LightLineSimRMLatheSideleftFusionHigh(serial_struct* ssp)


{
   aErr err = aErrNone;
   int b = 1;

   while(b==1)
   {
   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //Sideleft

   //l0

   char l01 = ssp->pinputbuff[8] ;
   char l02 = ssp->pinputbuff[9] ;

   //l1

   char l11 = ssp->pinputbuff[20] ;
   char l12 = ssp->pinputbuff[21] ;

   printf("\n\nl01 = %c \n" , l01);
   printf("l02 = %c \n" , l02);

   printf("\nl11 = %c \n" , l11);
   printf("l12 = %c \n" , l12);

   sleep(400) ;

    //Ultrasonic Process Forward

   Flush(ssp) ;

   Write(ssp, 17, "#%+ULTSW       *\\");              //W
   sleep(400) ;
   err = Read(ssp, 17);

   char uw1 = ssp->pinputbuff[10] ;
   char uw2 = ssp->pinputbuff[11] ;
   char uw3 = ssp->pinputbuff[12] ;

   printf("\nuw1 = %c \n" , uw1);
```

```
    printf("uw2 = %c \n" , uw2);
    printf("uw3 = %c \n" , uw3);

    sleep(500) ;

    //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s1 = ssp->pinputbuff[17] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[10] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s1 = %c \n" , s1);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis       9 = Light

//Sideleft     225  m1 l1       l0 m4     135

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') && s1 == 'O' )

Sideleft(ssp) ;

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft225(ssp) ;

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft135(ssp) ;
```

```
//No light line follow no obstacle

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') && s1 == 'O')

{
    Chill(ssp) ;
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//light == 8 change to low

else if ( (l01 == '0' && l02 == '8' && l11 == '0' && l12 == '8') && ( (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 ==
'0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) && s1 == 'O' )

{
    Chill(ssp) ;
    LightLineSimRMLatheSideleftFusionLow(ssp) ;
    b = 0;
}

//approx 50 cm in front of light go sideleft
//only stops when docked s1 = C

else if ( (l01 == '1' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (l01 == '1' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft135(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

else if ( (l01 == '0' && l11 == '1') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

{
    Sideleft225(ssp) ;
    printf("\nApprox 50 cm in front of Light.\n") ;
}

//Lmt
//6 7
```

```
//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 ==
'0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 >
'0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' )&& (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//No light

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '1' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '1' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '0' && l11 == '1') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//1

//light 50cm away

else if ( (l01 == '1' || l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//more light l01 135
else if ( (l01 == '1' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0')
|| (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 ==
'0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && (s1 == 'C' ) && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//No light

else if ( ((l01 == '0' && l02 <= '8') && (l11 == '0' && l12 <= '8')) && (s1 == 'C') && ( (uw1 >= '1'
&& uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2
> '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
```

```
    b = 0;
}

//Equal light

if ( (l02 == '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (l02 < '9' && l12 == '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left

else if ( (l02 == '9' && l12 < '9' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && ( (uw1 == '0' && uw2 ==
'0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3
== '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//light 50cm away
```

```
else if ( (l01 == '1' || l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' &&
uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' &&
uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l01 135

else if ( (l01 == '1' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l11 225

else if ( (l01 == '0' && l11 == '1') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

}
return err;
}
```

LightLineSimRMLatheSideleftFusionLow.c

```
//follow light - no light follow line Sideleft dir

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSimRMLatheSideleftFusionLow.h"
#include "LightOwnLineSimRMLatheSideleftFusionHigh.h"

//Reverse & Sideleft

int LightLineSimRMLatheSideleftFusionLow(serial_struct* ssp)

{
```

```
aErr err = aErrNone;
int b = 1;

while(b==1)
{

Flush(ssp) ;

Write(ssp, 17, "#%+LITER       *\\");
sleep(400) ;
err = Read(ssp, 50);

//Sideleft

//l0

char l01 = ssp->pinputbuff[8] ;
char l02 = ssp->pinputbuff[9] ;

//l1

char l11 = ssp->pinputbuff[20] ;
char l12 = ssp->pinputbuff[21] ;

printf("\n\nl01 = %c \n" , l01);
printf("l02 = %c \n" , l02);

printf("\nl11 = %c \n" , l11);
printf("l12 = %c \n" , l12);

sleep(400) ;

 //Ultrasonic Process Forward

Flush(ssp) ;
Write(ssp, 17, "#%+ULTSW       *\\");               //W
sleep(400) ;
err = Read(ssp, 17);

char uw1 = ssp->pinputbuff[10] ;
char uw2 = ssp->pinputbuff[11] ;
char uw3 = ssp->pinputbuff[12] ;

printf("\nuw1 = %c \n" , uw1);
printf("uw2 = %c \n" , uw2);
printf("uw3 = %c \n" , uw3);

sleep(500) ;

//Limit Switch Analysis
```

```
    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s1 = ssp->pinputbuff[17] ;
    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[10] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s1 = %c \n" , s1);
    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

//Analysis       9 = Light

//Sideleft     225  m1 l1       l0 m4     135

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' && uw3
>= '0') && s1 == 'O' )

Sideleft(ssp) ;

//More light toward r12 right

else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')
Sideleft225(ssp) ;

//More light toward r02 left

else if ( (l02 == '8' && l12 < '8' && l01 == '0' && l11 == '0') && (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') && s1 == 'O')

Sideleft135(ssp) ;

//No light line follow no obstacle

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (uw1 >= '1' && uw2 >= '0'
&& uw3 >= '0') && s1 == 'O')

{
    Chill(ssp) ;
```

```
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//approx 1m in front of light go sideleft
//only stops when docked s1 = C

else if ( (l01 == '9' && l11 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft(ssp) ;

else if ( (l01 == '9' && l11 < '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft135(ssp) ;

else if ( (l01 < '9' && l11 == '9') && (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') )

Sideleft225(ssp) ;

//Lmt
//6 7

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (uw1
>= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0'
&& uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0'
&& uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 ==
'0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 >
'0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}
//More light toward r02 left

else if ( (l02 == '8' && l12 < '8' )&& (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    EmergencyStopLight(ssp);
    b = 0;
}

//No light

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && (s6 == 'C' || s7 == 'C' ) &&
( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1
== '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' &&
uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '9' && l11 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 == '9' && l11 < '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

else if ( (l01 < '9' && l11 == '9') && (s6 == 'C' || s7 == 'C' ) && ( (uw1 >= '1' && uw2 >= '0' &&
uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0')
|| (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    EmergencyStopLight(ssp);
    b = 0;
}

//1

//light (1m away)

else if ( (l01 == '9' || l11 == '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )
```

```
{
    LatheChillLight(ssp);
    b = 0;
}

//more light l01 135

else if ( (l01 == '9' && l11 < '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//more light l11 225

else if ( (l01 < '9' && l11 == '9') && (s1 == 'C') && ( (uw1 >= '1' && uw2 >= '0' && uw3 >= '0') ||
(uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0'
&& uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
     LatheChillLight(ssp);
    b = 0;
}

//More light toward r12 right

else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && (s1 == 'C' ) && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}
```

```
//More light toward r02 left

else if ( (l02 == '8' && l12 < '8' && l01 == '0' && l11 == '0') && (s1 == 'C') && ( (uw1 >= '1' &&
uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 >
'0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//No light

else if ( ((l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7')) && (s1 == 'C') && ( (uw1 >= '1'
&& uw2 >= '0' && uw3 >= '0') || (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2
> '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2
== '0') ) )

{
    LatheChillLight(ssp);
    b = 0;
}

//Equal light

if ( (l02 == '8' && l12 == '8' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r12 right

else if ( (l02 < '8' && l12 == '8' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' &&
uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') ||
(uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r02 left
```

```
else if ( (l02 == '8' && l12 < '8' && l01 == '0' && l11 == '0') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( ( (l01 == '0' && l02 <= '7') && (l11 == '0' && l12 <= '7') ) && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightOwnLineSimRMLatheSideleftFusionHigh(ssp) ;
    b = 0;
}

//light 1m away

else if ( (l01 == '9' || l11 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l01 135

else if ( (l01 == '9' && l11 < '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//more light l11 225

else if ( (l01 < '9' && l11 == '9') && ( (uw1 == '0' && uw2 == '0' && uw3 == '0') || (uw1 == '0' && uw2 > '0'&& uw3 > '0') || (uw1 == '0' && uw2 > '0' && uw3 == '0') || (uw1 == '0' && uw3 > '0' && uw2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

}
return err;
}
```

LightOwnLineSimLatheRMSiderightFusionHigh.c

```
//return to light follow senses light
//search for line if cant find it
//detect line in fwd dir

//chked

//going sideright sees black line starts going foward

/*                                  E
                                  r2 r0 r1

                              f1
                        N     f0
                              f2
  */

#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineDock.h"
#include "LightOwnLineSimLatheRMSiderightFusionHigh.h"
#include "LightLineSimLatheRMSiderightFusionHigh.h"
#include "LightOwnLineSimConveyorRMForwardFusionHigh.h"

int LightOwnLineSimLatheRMSiderightFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
   int a = 1;

   while(a==1)
   {

   //1st board going right                LINEL

   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LINEL       *\\");
   sleep(400) ;
```

```
err = Read(ssp, 50);

char r0 = ssp->pinputbuff[9] ;
char r1 = ssp->pinputbuff[21] ;
char r2 = ssp->pinputbuff[33] ;

printf("\nr0 = %c \n" , r0);
printf("r1 = %c \n" , r1);
printf("r2 = %c \n" , r2);

 sleep(500) ;

 //2nd board foward                          LINER

 Flush(ssp) ;

 Write(ssp, 17, "#%+LINER       *\\");             //change to LITEL
 sleep(400) ;

 err = Read(ssp, 50);

char f0 = ssp->pinputbuff[9] ;
char f1 = ssp->pinputbuff[21] ;
char f2 = ssp->pinputbuff[33] ;

printf("\nf0 = %c \n" , f0);
printf("f1 = %c \n" , f1);
printf("f2 = %c \n" , f2);

sleep(400) ;

//light board

 Flush(ssp) ;

Write(ssp, 17, "#%+LITEL       *\\");
sleep(400) ;
err = Read(ssp, 50);

//Sideright
 //r0

 char r01 = ssp->pinputbuff[8] ;
 char r02 = ssp->pinputbuff[9] ;

 //r1

 char r11 = ssp->pinputbuff[20] ;
 char r12 = ssp->pinputbuff[21] ;
```

```
printf("\nr01 = %c \n" , r01);
printf("r02 = %c \n" , r02);

printf("\nr11 = %c \n" , r11);
printf("r12 = %c \n" , r12);

 sleep(400) ;

 //Limit Switch Analysis

Flush(ssp) ;

Write(ssp, 17, "#%+SWTCH       *\\");
sleep(400) ;

err = Read(ssp, 19);

char s6 = ssp->pinputbuff[15] ;
char s7 = ssp->pinputbuff[16] ;
char s8 = ssp->pinputbuff[17] ;
char s9 = ssp->pinputbuff[18] ;

printf("s6 = %c \n" , s6);
printf("s7 = %c \n" , s7);
printf("s8 = %c \n" , s8);
printf("s9 = %c \n" , s9);
printf("\n");

sleep(400) ;

//Ultrasonic Process SideRight

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSW       *\\");          //east
sleep(400) ;
err = Read(ssp, 17);

char ue1 = ssp->pinputbuff[10] ;
char ue2 = ssp->pinputbuff[11] ;
char ue3 = ssp->pinputbuff[12] ;

printf("\nue1 = %c \n" , ue1);
printf("ue2 = %c \n" , ue2);
printf("ue3 = %c \n" , ue3);

//Ultrasonic Process Forward

Flush(ssp) ;
```

```
   Write(ssp, 17, "#%+ULTSS       *\\");
   sleep(400) ;
   err = Read(ssp, 17);

   char un1 = ssp->pinputbuff[10] ;
   char un2 = ssp->pinputbuff[11] ;
   char un3 = ssp->pinputbuff[12] ;

   printf("\nun1 = %c \n" , un1);
   printf("un2 = %c \n" , un2);
   printf("un3 = %c \n" , un3);

//Sideright Analysis                        m2  r2 r0 r1

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright

if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright(ssp);

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright45(ssp) ;

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9' && r0 == '9' && r1 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') )

Sideright315(ssp) ;

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9' && r1 < '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright45(ssp) ;

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9' && r1 == '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (ue1 >= '1'
&& ue2 >= '0' && ue3 >= '0')  )

Sideright315(ssp);

// Foward Analysis                                front
```

```
/*                                          f1
                                            f0
                                            f2 */

//normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0')  )

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//normal right
//f1 on black f0 f2 on white go Right
```

```
else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
   Right(ssp) ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') )

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//normal right
//f0-f1 on black f2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
```

```
{
    Right(ssp) ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward
else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9'  && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r1 < '9'  && r0 == '9' && r2 == '9'&& f2 == '9' && f0 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}
```

```
//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Left(ssp) ;
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))
```

```
{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Left(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}
```

```
//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))
{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))
```

```
{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}


//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9'  && r2 < '9' && r0 < '9' && r1 == '9' && f2 == '9' && f1 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0'))

{
    Forward(ssp);
    LightOwnLineSimConveyorRMForwardFusionHigh(ssp);
    a = 0;
}
```

```
//Chill time - obstacle < 1m or crashed - obstacle

//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && (un1 >= '1' &&
un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (un1 >=
'1' && un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//f2 f0 f1 on black r1 r2 r0 on white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//f2 f0 f1 on white r1 r2 r0 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (un1 >= '1'
&& un2 >= '0' && un3 >= '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >= '0') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//Lmt switch

//Main Chills

//Offline and switch closed
//l2 l0 l1 on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && (s6 == 'C' || s7
== 'C'|| s8 == 'C'|| s9 == 'C') )
```

```
{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//l2 l0 l1 on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 ==
'C' || s7 == 'C' || s8 == 'C' || s9 == 'C') )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9')&& (s6 == 'C' ||
s7 == 'C'|| s8 == 'C'|| s9 == 'C') )

{
    Chill(ssp);
    printf("Crash limit. Check AGV. \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && (s6 == 'C' ||
s7 == 'C' || s8 == 'C' || s9 == 'C' ) )

{
    Chill(ssp) ;
    printf("Crash limit. Check AGV \n") ;
    a = 0;
}

//Offline and obstacle crash

//All on black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 < '9' && f0 < '9' && f2 < '9') && ( (ue1 == '0' &&
ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' &&
ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) ) )

{
```

```
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//All on white

else if ( (r1 == '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1
== '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 >
'0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on black r0-2 white

else if ( (f1 < '9' && f0 < '9' && f2 < '9') && (r1 == '9' && r0 == '9' && r2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp);
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//f0-2 on white r0-2 black

else if ( (r1 < '9' && r0 < '9' && r2 < '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') || ( (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) ) )

{
    Chill(ssp) ;
    printf("Crash obstacle. Check AGV \n") ;
    a = 0;
}

//On line obstacle detected waiting for clear

//r0 on black r1 r2 [f2 f0 f1] on white go Sideright
```

```
if ( (r0 < '9' && r1 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 == '0'
&& ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r1 on black r0 r2 [f2 f0 f1] on white go Sideright45

else if ( (r1 < '9' && r0 == '9' && r2 == '9') && (f1 == '9' && f0 == '9' && f2 == '9') && ( (ue1 ==
'0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0'
&& ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r2 on black r0 r1 [f2 f0 f1] on white go Sideright315

else if ( (r2 < '9') && (r0 == '9') && (r1 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r1 on black r2 [f2 f0 f1] on white go Sideright45

else if ( (r0 < '9') && (r1 < '9') && (r2 == '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//r0/r2 on black r1 [f2 f0 f1] on white go Sideright315

else if ( (r0 < '9') && (r1 == '9')&& (r2 < '9') && (f1 == '9') && (f0 == '9') && (f2 == '9') && (
(ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' &&
ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') )   )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
```

```
}

// Foward Analysis                              front

/*                                              f1
                                                f0
                                                f2 */
 //normal foward
//f0 on black f1 f2 r1 r0 r2 on white go Forward

if ( (f0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0 on black f1 f2 r1 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r1 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r1 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r2 on black f1 f2 r0 r2 on white stop sideright go Forward

else if ( (f0 < '9' && r2 < '9' && f1 == '9' && f2 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//normal right
//f1 on black f0 f2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r0 on black f0 f2 r1 r2 on white stop sideright go Forward

else if ( (f1 < '9' && r0 < '9' && f0 == '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r2-r0 r0-r1 combinations

//f1 r0-r2 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r0-r1 on black f0 f2 on white go Forward

else if ( (f1 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal right
//f0-f1 on black f2 on white go right
```

```
else if ( (f0 < '9' && f1 < '9' && f2 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0 on black f2 r1 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && f2 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0-r1 r0-r2

//f0-f1 r0-r1 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r1 < '9' && f2 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f1 r0-r2 on black f2 r2 on white go Forward

else if ( (f0 < '9' && f1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r1 on black f2 r2 on white go Forward

else if ( (f1 < '9' && r1 < '9'  && r0 == '9' && r2 == '9' && f2 == '9' && f0 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
```

```
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f1 r2 on black f2 r2 on white go Forward

else if ( (f1 < '9' &&  r2 < '9' && r0 == '9' && r1 == '9' && f2 == '9' && f0 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f2 on black f0 f1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 ==
'0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 >
'0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0 on black f0 f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && f0 == '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r0-r1 r0-r2 combinations

//f2 r0-r1 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//f2 r0-r2 on black f1 on white go Forward

else if ( (f2 < '9' && r0 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r1 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r1 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 r2 on black f0 f1 r2 on white go Forward

else if ( (f2 < '9' && r2 < '9' && f0 == '9' && f1 == '9' && r0 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//normal left
//f0-f2 on black f1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9' && r1 == '9' && r0 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 r0 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && f1 == '9' && r1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//f0-f2 r0-r1 r0-r2 combinations

//f0-f2 r0-r1 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r1 < '9' && f1 == '9' && r2 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0-f2 r0-r2 on black f1 on white go Forward

else if ( (f0 < '9' && f2 < '9' && r0 < '9' && r2 < '9' && f1 == '9' && r1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 [r0 r1 r2] on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && r2 < '9' && f2 == '9' && f1 == '9') && ( (un1 == '0'
&& un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0'
&& un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 < '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' &&
un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' &&
un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
//f0 f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 < '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}


//f1 [r0 r1 r2] on black go Forward

else if ( (f1 < '9' && f0 == '9' && f2 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f2 [r0 r1 r2] on black go Forward

else if ( (f2 < '9' && f0 == '9' && f1 == '9' && r1 < '9' && r0 < '9' && r2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0-r1 on black go Forward

else if ( (f0 < '9' && r1 < '9' && r0 < '9' && f1 == '9' && f2 == '9' && r2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//f0 r0-r2 on black go Forward

else if ( (f0 < '9' && r2 < '9' && r0 < '9' && r1 == '9'&& f2 == '9' && f1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )
```

```
{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More Chills
//s6 s7

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s6 == 'C' || s7 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    EmergencyStop(ssp) ;
    a = 0;
}
```

```
//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s6 == 'C' || s7 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
{
    EmergencyStop(ssp) ;
    a = 0;
}


//More Chills
//s8 dock at RM

if ( (f0 < '9' && f1 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C'  ) && ((ue1 >= '1' && ue2 >= '0'
&& ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0')
|| (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (f1 < '9' && f0 == '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (f2 < '9' && f0 == '9' && f1 == '9') && (s8 == 'C' || s9 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//l0/l1 on black l2 on white go right

else if ( (f0 < '9' && f1 < '9' && f2 == '9') && (s8 == 'C' || s9 == 'C' ) && ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )
```

```
{
    RMChill(ssp);
    a = 0;
}



//l0/l2 on black l1 on white go Left

else if ( (f0 < '9' && f1 == '9' && f2 < '9') && (s8 == 'C' || s9 == 'C' )&& ((ue1 >= '1' && ue2 >=
'0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 >
'0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 == '0') ) )

{
    RMChill(ssp);
    a = 0;
}

//Check for high light

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' && ue3 >=
'0'))

{
    Sideright(ssp) ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r12 right

else if ((r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

{
    Sideright45(ssp) ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7' && r01 == '0' && r11 == '0') && (ue1 >= '1' && ue2 >= '0' &&
ue3 >= '0'))

{
    Sideright315(ssp) ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
```

```
}

//approx 50 cm in front of light load forward
//ConveyorRM = LatheRM
//only stops when docked s1 = C

else if (r01 == '1' || r11 == '1')
{
   Chill(ssp);
   LightLineSimConveyorRMForwardFusionHigh(ssp);
   printf("\nApprox 0.5 m in front of Light. Loading Forward\n") ;
   a = 0 ;
}

//Lmt
//6 7

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && ( (ue1 >=
'1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0' && ue2 >
'0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' && ue2 ==
'0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7' && r01 == '0' && r11 == '0') && (s6 == 'C' || s7 == 'C' ) && (
(ue1 >= '1' && ue2 >= '0' && ue3 >= '0') || (ue1 == '0' && ue2 == '0' && ue3 == '0') || (ue1 == '0'
&& ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 == '0' && ue3 > '0' &&
ue2 == '0') ) )

{
    EmergencyStopLight(ssp);
    a = 0;
```

```
}

//Obstacle detected. Waiting for clear

//Equal light

if ( (r02 > '7' && r12 > '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' && ue3
== '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1 ==
'0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r12 right

else if ( (r02 <= '7' && r12 > '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

//More light toward r02 left

else if ( (r02 > '7' && r12 <= '7' && r01 == '0' && r11 == '0') && ( (ue1 == '0' && ue2 == '0' &&
ue3 == '0') || (ue1 == '0' && ue2 > '0'&& ue3 > '0') || (ue1 == '0' && ue2 > '0' && ue3 == '0') || (ue1
== '0' && ue3 > '0' && ue2 == '0') ) )

{
    Chill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    LightLineSimLatheRMSiderightFusionHigh(ssp);
    a = 0;
}

}

return err;

}
```

E.2 Robot Programming

2.1 Interfaces

Robot Main Interface.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMainInterface.h"
#include "RobotMainInterfaceEnvironment.h"

int RobotMainInterface(serial_struct* ssp)

{
   Flush(ssp) ;

   aErr err = aErrNone;

   int e ;

   printf("\nVariable Sensor System"  ) ;
   printf("\n----------------------\n") ;

   printf("\nPlease select an Environment: \n\n"  ) ;

   printf("Unsatisfactory Floor \t\t 1  \nLight Intensity \t\t 2 \nHigh Noise Level \t\t 3 \nConstant
Environment \t\t 4 \n\n") ;
   scanf("%d" , &e) ;

   printf("\n") ;

   if (e == 1)
   RobotUnsatisfactoryFloor(ssp) ;

   else if (e == 2)
   RobotLightIntensity(ssp) ;

   else if (e == 3)
   RobotHighNoiseLevelEnvironment(ssp) ;

   else if (e == 4)
   RobotConstantEnvironment(ssp) ;

   else printf("\nUnknown Environment. Please enter new Environment\n") ;

return err;

}
```

Robot Main Interface Environment.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "RobotMainInterfaceEnvironment.h"
#include "RobotLightFusion.h"
#include "RobotLineFusion.h"
#include "RobotLineLightFusionHigh.h"
#include "RobotLightLineFusionHigh.h"
#include "RobotLightLineFusionLow.h"

int RobotUnsatisfactoryFloor(serial_struct* ssp)

{
   //only load line var routes here

   aErr err = aErrNone;
   int t;

   printf("Unsatisfactory Floor");
   printf("\n--------------------\n") ;

   printf("\n\nPlease select an Navigation & Guidance Technique for the forward direction: \n\n"  ) ;

   printf("Line Variation \t\t\t 1 \nLight Following \t\t 2 \nExit to Main \t\t\t 3\n\n") ;
   scanf("%d", t);
   printf("\n") ;

   if (t == 1)
   RobotLineLightForwardFusionHigh(ssp) ;

   else if (t == 2)
   RobotForwardLightFusion(ssp) ;

   else if (t == 3)
   RobotMainInterface(ssp) ;

   else printf("\nUnknown technique. Please enter new technique\n") ;

   return err;
}

int RobotLightIntensity(serial_struct* ssp)

{

   aErr err = aErrNone;
```

```
    int t;

    printf("Light Intensity");
    printf("\n---------------\n") ;

    printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

    printf("Light Variation \t\t\t 1 \nLine Following \t\t 2 \nExit to Main \t\t\t 3\n\n") ;
    scanf("%d", t);

    printf("\n") ;

    if (t == 1)
    RobotLightVariationInterface(ssp) ;

    else if (t == 2)
    RobotForwardLineFusion(ssp) ;

    else if (t == 3)
    RobotMainInterface(ssp) ;

    else printf("\nUnknown technique. Please enter new technique\n") ;

    return err;
}

int RobotHighNoiseLevelEnvironment(serial_struct* ssp)

{
    aErr err = aErrNone;
    int t;

    printf("\nHigh Noise Level Environment") ;
    printf("\n---------------------------\n") ;

    printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

    printf("Line Variation \t\t\t 1  \nLight Variation \t\t 2 \nLine Following \t\t\t 3  \nLight Following
\t\t 4 \nExit to Main \t\t\t 5\n\n") ;
    scanf("%d", t);

    printf("\n") ;

    if (t == 1)
    RobotLineLightForwardFusionHigh(ssp) ;

    if (t == 2)
    RobotLightVariationInterface(ssp) ;

    else if (t == 3)
```

```
   RobotForwardLineFusion(ssp) ;

   else if (t == 4)
   RobotForwardLightFusion(ssp) ;

   else if (t == 5)
   RobotMainInterface(ssp) ;

   else printf("\nUnknown technique. Please enter new technique\n") ;

   return err;
}

int RobotConstantEnvironment(serial_struct* ssp)

{
   aErr err = aErrNone;
   int t;

   printf("\nConstant Environment \n\n"  ) ;
   printf("\n--------------------\n") ;

   printf("\n\nPlease select an Navigation & Guidance Technique: \n\n"  ) ;

   printf("Line Variation \t\t\t 1  \nLight Variation \t\t 2 \nLine Following \t\t\t 3  \nLight Following
\t\t 4 \nExit to Main \t\t\t 5\n\n") ;
   scanf("%d", t);

   printf("\n") ;

   if (t == 1)
   RobotLineLightForwardFusionHigh(ssp) ;

   if (t == 2)
   RobotLightVariationInterface(ssp) ;

   else if (t == 3)
   RobotForwardLineFusion(ssp) ;

   else if (t == 4)
   RobotForwardLightFusion(ssp) ;

   else if (t == 5)
   RobotMainInterface(ssp) ;

   else printf("\nUnknown technique. Please enter new technique\n") ;

   return err;

}
int RobotLightVariationInterface(serial_struct* ssp)
```

```
{
   Flush(ssp) ;

   aErr err = aErrNone;

   int i ;

   printf("\nPlease select a Light Intensity: \n\n"  ) ;

   printf("High (20W, 35W, 50W) \t 1 \nLow (5W, 10W) \t\t 2 \nExit to Main \t\t 3 ") ;
   scanf("%d" , &i) ;

    printf("\n") ;

   if (i == 1)
   RobotForwardLightLineFusionHigh(ssp) ;

   else if (i == 2)
   RobotForwardLightLineFusionLow(ssp) ;

   else if (i == 3)
   MainInterface(ssp) ;

   else printf("\nUnknown Intensity. Please enter new Intensity\n") ;

   return err;

}
```

Robot Motor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"

//      7           8
//      R-R         B-R

int RobotChill(serial_struct *ssp)

{
   aErr err = aErrNone;

   Write(ssp, 17, "#%+MOT7SPF0000 *\\");

   sleep(400);
```

```
    Write(ssp, 17, "#%+MOT8SPF0000 *\\");

    sleep(400);

    return err;
}

int RobotForward(serial_struct* ssp)

{
    RobotChill(ssp) ;

     /*
    1 2      front     order motors come on
    3 4      back
    */

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT7SPF0100 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT8SPF0100 *\\");

    sleep(400);

    return err;
}
int RobotReverse(serial_struct* ssp)

{

    /*
    3 4
    1 2
    */

    RobotChill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT7SPR0100 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT8SPR0100 *\\");

    sleep(400);
```

```
    return err;
}

int RobotLeft(serial_struct* ssp)

{

    /*
    2 1
    3 4

    1 & 4 reverse 2 & 3 forward
    */

    RobotChill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT7SPF0085 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT8SPR0080 *\\");

    sleep(400);

    return err;
}


int RobotRight(serial_struct* ssp)

{

     /*
    1 2
    4 3

    2 & 3 reverse 1 & 4 forward
    */

    RobotChill(ssp) ;

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT8SPF0100 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT7SPR0095 *\\");
```

```
    sleep(400);

    return err;
}

int RobotForwardLeft(serial_struct* ssp)

{
    RobotChill(ssp) ;

     /*
    1 2      front     order motors come on
    3 4      back
    */

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT7SPF0090 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT8SPF0085 *\\");

    sleep(400);

    return err;
}



int RobotForwardRight(serial_struct* ssp)

{
    RobotChill(ssp) ;

     /*
    1 2      front     order motors come on
    3 4      back
    */

    aErr err = aErrNone;

    Write(ssp, 17, "#%+MOT8SPF0090 *\\");

    sleep(400);

    Write(ssp, 17, "#%+MOT7SPF0085 *\\");

    sleep(400);

    return err;
```

```
}

int RobotReverseRight(serial_struct* ssp)

{

   /*
   4 3
   1 2

   1 & 4 reverse 2 & 3 forward
   */

   RobotChill(ssp) ;

   aErr err = aErrNone;

   Write(ssp, 17, "#%+MOT7SPR0150 *\\");

   sleep(400);

   Write(ssp, 17, "#%+MOT8SPF0100 *\\");

   sleep(400);

   return err;

}

int RobotReverseLeft(serial_struct* ssp)

{

   /*
   3 4
   2 1

   2 & 3 reverse 1 & 4 forward
   */

   RobotChill(ssp) ;

   aErr err = aErrNone;

   Write(ssp, 17, "#%+MOT8SPR0150 *\\");

   sleep(400);

   Write(ssp, 17, "#%+MOT7SPF0100 *\\");

   sleep(400);
```

```
   return err;
}

//robot slower
//light 8-9 10W


/*Reverse functions

2       3       u r here facing reverse     left hand       reverse left
1       4                                   right hand      reverse right

*/
```

2.2 Line Following Technique
Robot Line Fusion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"
#include "RobotLineFusion.h"
#include "LineDock.h"

/* white = '9'
   black < '9'
*/

int RobotForwardLineFusion(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */
{
   aErr err = aErrNone ;
   int sw = 1 ;

   while (sw==1)
   {

   Flush(ssp) ;
   Write(ssp, 17, "#%+LINER       *\\");
   sleep(400) ;

   err = Read(ssp, 50);
```

```
    char l0 = ssp->pinputbuff[9] ;
    char l1 = ssp->pinputbuff[21] ;
    char l2 = ssp->pinputbuff[33] ;

    printf("l0 = %c \n" , l0);
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

    sleep(400) ;

    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSN       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

     sleep(400) ;

/*//Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");  */

//On Line & > 1m - no obstacle
//Analysis
```

```
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotForward(ssp);

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotRight(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )
RobotLeft(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotRight(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotLeft(ssp);

//Chill

//l2 l0 l1 on black

else if ( (l1 < '9') && (l0 < '9') && (l2 < '9') )

{
    RobotChill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}

//l2 l0 l1 on white

else if ( (l1 == '9') && (l0 == '9') && (l2 == '9') )

{
    RobotChill(ssp) ;
    printf("Offline. Check AGV \n") ;
    sw = 0;
}
```

```
//On Line - Obstacle detected. Waiting for clear

//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l2 on black l1 on white go Left
```

```
else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

/*//Limit switch

//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) && ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) && ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) && ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  && ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
```

```
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  &&
((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )
{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}*/
}
return err;
}
```

2.3 Light Following Technique

Robot Light Fusion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"
#include "RobotLightFusion.h"
#include "LightDock.h"

/* ForwardLight */

int RobotForwardLightFusion(serial_struct* ssp)

{
   aErr err = aErrNone ;
   int sw = 1 ;

   while (sw==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);
```

```
    //r2

    char f21 = ssp->pinputbuff[32] ;
    char f22 = ssp->pinputbuff[33] ;

    //r3

    char f31 = ssp->pinputbuff[44] ;
    char f32 = ssp->pinputbuff[45] ;

    printf("\nf21 = %c \n" , f21);
    printf("f22 = %c \n" , f22);

    printf("\nf31 = %c \n" , f31);
    printf("f32 = %c \n" , f32);

     sleep(400) ;

     //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSN       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

    sleep(400) ;

/*
//Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;
```

```
    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n");

    //sleep(400) ;
*/

//Forward      left      m1  2        3   m2       right

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

RobotForward(ssp) ;

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotRight(ssp) ;


//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotLeft(ssp) ;

//No light

else if ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
    RobotRight(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//30 cm in front of light or really really close

else if  (f21 == '1' || f31 == '1')

{
    RobotChill(ssp) ;
    printf("\n30 cm in front of Light or really really close.\n") ;
    sw = 0;
```

```
}

/*
//Limit Switch
//6 7

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C' ||
s9 == 'C' )  && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') ) && (s6 == 'C' || s7 == 'C' || s8
== 'C' || s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
```

```
    RobotEmergencyStopLight(ssp);
    sw = 0;
}
*/

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '9' && f32 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2
> '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( (f21 == '0' && f22 <= '8') && (f31 == '0' && f32 <= '8') && ( (un1 == '0' && un2 == '0'
&& un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
```

```
}
return err;

}
```

2.4 Line Variation Technique

Robot Line Light Fusion High.c

```
//follow line
//no line - follow light

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"
#include "RobotLineFusion.h"
#include "LineDock.h"
#include "RobotLineLightFusionHigh.h"

/* white = '9'
   black < '9'
*/

int RobotLineLightForwardFusionHigh(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */
{
   aErr err = aErrNone;
   int sw = 1;

   while(sw==1)
   {

   Flush(ssp) ;
   Write(ssp, 17, "#%+LINER       *\\");
   sleep(400) ;

   err = Read(ssp, 50);

   char l0 = ssp->pinputbuff[9] ;
   char l1 = ssp->pinputbuff[21] ;
   char l2 = ssp->pinputbuff[33] ;

   printf("l0 = %c \n" , l0);
   printf("l1 = %c \n" , l1);
```

```
    printf("l2 = %c \n" , l2);

    sleep(400) ;

    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSN       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

     sleep(400) ;

    /*//Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n"); */

//On Line & > 1m - no obstacle
//Analysis

//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotForward(ssp);
```

```
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotRight(ssp) ;
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotLeft(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotRight(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotLeft(ssp);

//no line - light follow

//l2 l0 l1 on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotChill(ssp) ;
    RobotLineOwnLightForwardFusionHigh(ssp);
    sw = 0;
}

//l2 l0 l1 on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotChill(ssp) ;
    RobotLineOwnLightForwardFusionHigh(ssp);
    sw = 0;
}

//On Line - Obstacle detected. Waiting for clear
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )
```

```
{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

/*//Limit switch
```

```
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) && ((un1
>= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )
{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )
&& ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )
&& ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  &&
((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  &&
((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
```

```
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )
{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}*/
}return err;
}
```

Robot Line Own Light Forward Fusion High.c

```
//follow light
//no light - search for light
//check for line return if found

//only forward but RM to Lathe
//robotmotor

//no light line follow
//==8 low light follow

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"
#include "RobotLightLineFusionHigh.h"
#include "RobotLightLineFusionLow.h"
#include "RobotLightOwnLineFusionHigh.h"
#include " RobotLineOwnLightForwardFusionHigh.h"

#include "LightDock.h"

/* ForwardLight */

int RobotLineOwnLightForwardFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
   int sw = 1;

   while(sw==1)
   {

   Flush(ssp) ;
   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2
```

```
char f21 = ssp->pinputbuff[32] ;
char f22 = ssp->pinputbuff[33] ;

//r3

char f31 = ssp->pinputbuff[44] ;
char f32 = ssp->pinputbuff[45] ;
printf("\nf21 = %c \n" , f21);
printf("f22 = %c \n" , f22);

printf("\nf31 = %c \n" , f31);
printf("f32 = %c \n" , f32);

 sleep(400) ;

//line board

Flush(ssp) ;
Write(ssp, 17, "#%+LINER       *\\");
sleep(400) ;

err = Read(ssp, 50);

char l0 = ssp->pinputbuff[9] ;
char l1 = ssp->pinputbuff[21] ;
char l2 = ssp->pinputbuff[33] ;

printf("l0 = %c \n" , l0);
printf("l1 = %c \n" , l1);
printf("l2 = %c \n" , l2);

sleep(400) ;

 //Ultrasonic Process Forward

Flush(ssp) ;

Write(ssp, 17, "#%+ULTSN       *\\");
sleep(400) ;
err = Read(ssp, 17);

char un1 = ssp->pinputbuff[10] ;
char un2 = ssp->pinputbuff[11] ;
char un3 = ssp->pinputbuff[12] ;

printf("\nun1 = %c \n" , un1);
printf("un2 = %c \n" , un2);
printf("un3 = %c \n" , un3);

sleep(400) ;
```

```
  /* //Limit Switch Analysis

   Flush(ssp) ;

   Write(ssp, 17, "#%+SWTCH       *\\");
   sleep(400) ;

   err = Read(ssp, 19);

   char s6 = ssp->pinputbuff[15] ;
   char s7 = ssp->pinputbuff[16] ;
   char s8 = ssp->pinputbuff[17] ;
   char s9 = ssp->pinputbuff[18] ;

   printf("\ns6 = %c \n" , s6);
   printf("s7 = %c \n" , s7);
   printf("s8 = %c \n" , s8);
   printf("s9 = %c \n" , s9);
   printf("\n"); */

 //Forward      left      m1  2       3   m2       right

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

RobotForward(ssp) ;

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotRight(ssp) ;

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotLeft(ssp) ;

//No light - search

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
    RobotRight(ssp) ;
    printf("Searching for light. \n") ;
```

```
    sw = 0;
}

//Low light - low light follow

else if ( (f21 == '0' && f22 == '8') && (f31 == '0' && f32 == '8') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )
{
    RobotRight(ssp) ;
    printf("Searching for light. \n") ;
    sw = 0;
}

//0.5 m in front of light or really really close

else if  (f21 == '1' || f31 == '1')

{
    RobotChill(ssp) ;
    printf("\n0.5 m in front of Light.\n") ;
    sw = 0;
}

/*//Limit Switch
//6 7

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C' ||
s9 == 'C' )  && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r22 left
```

```
else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') ) && (s6 == 'C' || s7 == 'C' || s8
== 'C' || s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    RobotEmergencyStopLight(ssp);
    sw = 0;
} */

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left
```

```
else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//check for line
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotForward(ssp);
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotRight(ssp) ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotLeft(ssp) ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}
```

```
//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotRight(ssp) ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotLeft(ssp);
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//On Line - Obstacle detected. Waiting for clear

//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//l2 on black l0 l1 on white go Left
```

```
else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotLineLightForwardFusionHigh(ssp);
    sw = 0;
}

/*//Limit switch

//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) && ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}
```

```
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )
&& ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )
&& ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  &&
((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  &&
((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}*/

}
return err;
}
```

2.5 Light Variation Technique
Robot Forward Light Line Fusion High.c

```
//only forward but RM to Lathe
//robotmotor

//no light - line follow
//==8 low light follow

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"
#include "RobotLightLineFusionHigh.h"
#include "RobotLightLineFusionLow.h"
#include "RobotLightOwnLineFusionHigh.h"
#include "LightDock.h"


/* ForwardLight */

int RobotForwardLightLineFusionHigh(serial_struct* ssp)

{
   aErr err = aErrNone;
   int sw = 1;

   while(sw==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //r3

   char f31 = ssp->pinputbuff[44] ;
   char f32 = ssp->pinputbuff[45] ;

   printf("\nf21 = %c \n" , f21);
   printf("f22 = %c \n" , f22);
```

```
    printf("\nf31 = %c \n" , f31);
    printf("f32 = %c \n" , f32);

     sleep(400) ;

     //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSN       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

    sleep(400) ;

    /*//Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n"); */

 //Forward      left      m1  2        3   m2       right

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

RobotForward(ssp) ;
```

```
//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotRight(ssp) ;

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotLeft(ssp) ;

//No light - line follow

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
    RobotChill(ssp) ;
    RobotForwardLightOwnLineFusionHigh(ssp) ;
    sw = 0;
}

//Low light - low light follow

else if ( (f21 == '0' && f22 == '8') && (f31 == '0' && f32 == '8') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
    RobotChill(ssp) ;
    RobotForwardLightLineFusionLow(ssp) ;
    sw = 0;
}

//combine with limit
//0.5 m in front of light or really really close

else if  (f21 == '1' || f31 == '1')

{
    RobotChill(ssp) ;
    printf("\n0.5 m in front of Light.\n") ;
    sw = 0;
}



/*//Limit Switch
```

```
//6 7

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C' ||
s9 == 'C' )  && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') ) && (s6 == 'C' || s7 == 'C' || s8
== 'C' || s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    RobotEmergencyStopLight(ssp);
    sw = 0;
}*/
//Ultrasonics
```

```
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right

else if ( (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && ( (un1 == '0' && un2 == '0'
&& un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotForwardLightOwnLineFusionHigh(ssp) ;
}

}
return err;
}
```

Robot Forward Light Line Fusion Low.c

```
//only forward but RM to Lathe
//robotmotor

//no light line follow
//==8 low light follow

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"
#include "RobotLightLineFusionLow.h"
//#include "RobotLightOwnLineFusionLow.h"
#include "LightDock.h"

/* ForwardLight */

int RobotForwardLightLineFusionLow(serial_struct* ssp)

{
   aErr err = aErrNone;
   int sw = 1;

   while(sw==1)
   {

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //r3

   char f31 = ssp->pinputbuff[44] ;
   char f32 = ssp->pinputbuff[45] ;

   printf("\nf21 = %c \n" , f21);
   printf("f22 = %c \n" , f22);

   printf("\nf31 = %c \n" , f31);
   printf("f32 = %c \n" , f32);
   sleep(400) ;
```

```
    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSN       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

    sleep(400) ;


/* //Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("\ns6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n"); */

//Forward     left     m1  2       3   m2       right

//Equal light

if ( (f22 == '8' && f32 == '8' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3
>= '0') )

RobotForward(ssp) ;

//More light toward r32 right
```

```
else if ( (f22 < '8' && f32 == '8' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotRight(ssp) ;

//More light toward r22 left

else if ( (f22 == '8' && f32 < '8' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

RobotLeft(ssp) ;

//No light - line follow

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && (un1 >= '1' && un2 >= '0'
&& un3 >= '0') )

{
    RobotChill(ssp) ;
    RobotForwardLightOwnLineFusionHigh(ssp) ;
    sw = 0;
}

//0.5 m in front of light or really really close

else if  (f21 == '9' || f31 == '9')

{
    RobotChill(ssp) ;
    printf("\n0.5 m in front of Light.\n") ;
    sw = 0;
}

/*//Limit Switch
//6 7

//Equal light

if ( (f22 == '8' && f32 == '8' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C' ||
s9 == 'C' )  && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r32 right
```

```
else if ( (f22 < '8' && f32 == '8' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (f22 == '8' && f32 < '8' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//No light

else if ( ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') ) && (s6 == 'C' || s7 == 'C' || s8
== 'C' || s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    printf("/nLight search to be continued/n") ;
    RobotEmergencyStopLight(ssp);
    sw = 0;
} */

//Ultrasonics
//Obstacle detected. Waiting for clear

//Equal light

if ( (f22 == '8' && f32 == '8') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' && un2
> '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' && un2 ==
'0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r32 right
```

```
else if ( (f22 < '8' && f32 == '8') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//More light toward r22 left

else if ( (f22 == '8' && f32 < '8') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//No light

else if ( (f21 == '0' && f22 <= '7') && (f31 == '0' && f32 <= '7') && ( (un1 == '0' && un2 == '0'
&& un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotForwardLightOwnLineFusionHigh(ssp) ;
}

}

return err;

}
```

Robot Forward Light Own Line Fusion High.c

```
//follow line check for light
//>7
//return if found

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "RobotMotor.h"
#include "RobotLightOwnLineFusionHigh.h"
```

```
#include "RobotLightLineFusionHigh.h"
#include "LineDock.h"

//Test with ultra then add lmt switches crash only

/* white = '9'
   black < '9'
*/

int RobotForwardLightOwnLineFusionHigh(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */
{
    aErr err = aErrNone;
    int sw = 1;

    while(sw==1)
    {
    Flush(ssp) ;
    Write(ssp, 17, "#%+LINER       *\\");
    sleep(400) ;

    err = Read(ssp, 50);

    char l0 = ssp->pinputbuff[9] ;
    char l1 = ssp->pinputbuff[21] ;
    char l2 = ssp->pinputbuff[33] ;

    printf("l0 = %c \n" , l0);
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

    sleep(400) ;
//light board

    Flush(ssp) ;

    Write(ssp, 17, "#%+LITER       *\\");
    sleep(400) ;
    err = Read(ssp, 50);

    //r2
    char f21 = ssp->pinputbuff[32] ;
    char f22 = ssp->pinputbuff[33] ;

    //r3
    char f31 = ssp->pinputbuff[44] ;
```

```
    char f32 = ssp->pinputbuff[45] ;

    printf("\nf21 = %c \n" , f21);
    printf("f22 = %c \n" , f22);
    printf("\nf31 = %c \n" , f31);
    printf("f32 = %c \n" , f32);

     sleep(400) ;

    //Ultrasonic Process Forward

    Flush(ssp) ;

    Write(ssp, 17, "#%+ULTSN       *\\");
    sleep(400) ;
    err = Read(ssp, 17);

    char un1 = ssp->pinputbuff[10] ;
    char un2 = ssp->pinputbuff[11] ;
    char un3 = ssp->pinputbuff[12] ;

    printf("\nun1 = %c \n" , un1);
    printf("un2 = %c \n" , un2);
    printf("un3 = %c \n" , un3);

     sleep(400) ;

    /*//Limit Switch Analysis

    Flush(ssp) ;

    Write(ssp, 17, "#%+SWTCH       *\\");
    sleep(400) ;

    err = Read(ssp, 19);

    char s6 = ssp->pinputbuff[15] ;
    char s7 = ssp->pinputbuff[16] ;
    char s8 = ssp->pinputbuff[17] ;
    char s9 = ssp->pinputbuff[18] ;

    printf("s6 = %c \n" , s6);
    printf("s7 = %c \n" , s7);
    printf("s8 = %c \n" , s8);
    printf("s9 = %c \n" , s9);
    printf("\n"); */

//On Line & > 1m - no obstacle
//Analysis

//l0 on black l1 l2 on white go fwd
```

```
if ( (l0 < '9' && l1 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotForward(ssp);

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotRight(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotLeft(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotRight(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

RobotLeft(ssp);

//Chill

//l2 l0 l1 on black

else if ( (l1 < '9' && l0 < '9' && l2 < '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotRight(ssp) ;
    printf("Searching for line. \n") ;
    sw = 0;
}

//l2 l0 l1 on white

else if ( (l1 == '9' && l0 == '9' && l2 == '9') && (un1 >= '1' && un2 >= '0' && un3 >= '0') )

{
    RobotRight(ssp) ;
    printf("Searching for line. \n") ;
    sw = 0;
}
```

```
//On Line - Obstacle detected. Waiting for clear

//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') ||
(un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' &&
un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && ( (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )
```

```
{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
}

/*//Limit switch

//l0 on black l1 l2 on white go fwd

if ( (l0 < '9' && l1 == '9' && l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' ) && ((un1
>= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0' &&
un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0' &&
un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )
&& ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}
//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )
&& ((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1
== '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 >
'0' && un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  &&
((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
```

```
    RobotEmergencyStop(ssp) ;
    sw = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9 == 'C' )  &&
((un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 == '0') || (un1 == '0'
&& un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0' && un3 > '0'
&& un2 == '0') ) )

{
    RobotEmergencyStop(ssp) ;
    sw = 0;
}
*/

//Check for light
//Forward     left      m1  2       3   m2       right
//Equal light

if ( (f22 > '7' && f32 > '7' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' && un3 >=
'0') )

{
    RobotForward(ssp) ;
    RobotForwardLightLineFusionHigh(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 <= '7' && f32 > '7' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )
{
    RobotRight(ssp) ;
    RobotForwardLightLineFusionHigh(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (f22 > '7' && f32 <= '7' && f21 == '0' && f31 == '0') && (un1 >= '1' && un2 >= '0' &&
un3 >= '0') )

{
    RobotLeft(ssp) ;
    RobotForwardLightLineFusionHigh(ssp);
    sw = 0;
}
```

```
//0.5 m in front of light or really really close

else if  (f21 == '1' || f31 == '1')

{
    RobotChill(ssp) ;
    printf("\n0.5 m in front of Light.\n") ;
    sw = 0;
}

/*//Limit Switch
//6 7

//Equal light

if ( (f22 > '7' && f32 > '7' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C' || s9
== 'C' )  && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3 ==
'0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 == '0'
&& un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 <= '7' && f32 > '7' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (f22 > '7' && f32 <= '7' && f21 == '0' && f31 == '0') && (s6 == 'C' || s7 == 'C' || s8 == 'C'
|| s9 == 'C' ) && ( (un1 >= '1' && un2 >= '0' && un3 >= '0') || (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotEmergencyStopLight(ssp);
    sw = 0;
} */

//Ultrasonics
//Obstacle detected. Waiting for clear
```

```
//Equal light

if ( (f22 > '7' && f32 > '7' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' && un3
== '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') || (un1 ==
'0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotForwardLightLineFusionHigh(ssp);
    sw = 0;
}

//More light toward r32 right

else if ( (f22 <= '7' && f32 > '7'  && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )

{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotForwardLightLineFusionHigh(ssp);
    sw = 0;
}

//More light toward r22 left

else if ( (f22 > '7' && f32 <= '7' && f21 == '0' && f31 == '0') && ( (un1 == '0' && un2 == '0' &&
un3 == '0') || (un1 == '0' && un2 > '0'&& un3 > '0') || (un1 == '0' && un2 > '0' && un3 == '0') ||
(un1 == '0' && un3 > '0' && un2 == '0') ) )
{
    RobotChill(ssp);
    printf("Obstacle detected. Waiting for clear \n") ;
    RobotForwardLightLineFusionHigh(ssp);
    sw = 0;
}
}
return err;
}
```

E.3 Program progress and optimization

3.1 First Program: Line Variation

LineLight.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
```

```
#include "Motor.h"
#include "LineLight.h"

/* white = '9'
   black < '9'
*/

int LineLight(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */

{
    Flush(ssp) ;
    aErr err = aErrNone;
    extern int sw ;

    Write(ssp, 17, "#%+LINER       *\\");
    sleep(400) ;


     err = Read(ssp, 50);

    char l0 = ssp->pinputbuff[9] ;
    char l1 = ssp->pinputbuff[21] ;
    char l2 = ssp->pinputbuff[33] ;

    printf("l0 = %c \n" , l0);
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

     //sleep(500) ;

//Analysis
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9') && (l1 == '9') && (l2 == '9') )

Forward(ssp);

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') )

Right(ssp) ;

//l2 on black l0 l1 on white go Left
```

```
else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') )

Left(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') )

Right(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') )

Left(ssp);

//When goes offline stops and starts light following

//l2 l0 l1 on black

else if ( (l1 < '9') && (l0 < '9') && (l2 < '9') )

{
    Chill(ssp) ;
    LineOwnLight(ssp) ;
    sw = 0 ;
    printf(" Exit \n");
}

//l2 l0 l1 on white

else if ( (l1 == '9') && (l0 == '9') && (l2 == '9') )

{
    Chill(ssp) ;
    LineOwnLight(ssp) ;
    sw = 0 ;
    printf(" Exit \n");
}

return err;

}
```

LineOwnLight.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineOwnLight.h"

// This is a light following program that line following swaps to when no line
```

```
// Searches for light first  - forward dir

int LineOwnLight(serial_struct* ssp)

{
   aErr err = aErrNone;
   int x=1 ;

   while(x==1)
{

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //r3

   char f31 = ssp->pinputbuff[44] ;
   char f32 = ssp->pinputbuff[45] ;

   printf("\nf21 = %c \n" , f21);
   printf("f22 = %c \n" , f22);

   printf("\nf31 = %c \n" , f31);
   printf("f32 = %c \n" , f32);

//Analysis       9 = Light

//Forward     left      m1  2       3   m2       right

//Equal light

if (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Forward(ssp) ;
//More light toward r32 right

else if (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Right(ssp) ;

//More light toward r22 left

else if (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0')
```

```
Left(ssp) ;

//No light

else if ( ((f21 == '0') && (f22 <= '8')) && ((f31 == '0') && (f32 <= '8')) )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//30 cm in front of light or really really close

else if ( (f21 == '1') || (f31 == '1') )

{
    Chill(ssp) ;
    printf("\n30 cm in front of Light or really really close.\n") ;
    x = 0;
}

}

return err;

}
```

Light Variation
LightLine.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLine.h"

int LightLine(serial_struct* ssp)

{
   Flush(ssp) ;
   aErr err = aErrNone;
   extern int sw ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
```

```
    err = Read(ssp, 50);

    //r2

    char f21 = ssp->pinputbuff[32] ;
    char f22 = ssp->pinputbuff[33] ;

    //r3

    char f31 = ssp->pinputbuff[44] ;
    char f32 = ssp->pinputbuff[45] ;

    printf("\nf21 = %c \n" , f21);
    printf("f22 = %c \n" , f22);

    printf("\nf31 = %c \n" , f31);
    printf("f32 = %c \n" , f32);

//Analysis        9 = Light

//Forward      left      m1  2        3   m2       right

//Equal light

if (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Forward(ssp) ;

//More light toward r32 right

else if (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Right(ssp) ;

//More light toward r22 left

else if (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0')

Left(ssp) ;

else if ( (f21 == '1') || (f31 == '1') )

{
    Chill(ssp) ;
    printf("\n30 cm in front of Light or really really close.\n") ;
    sw = 0;
}

//When no light stops and starts line following

else if ( ((f21 == '0') && (f22 <= '8')) && ((f31 == '0') && (f32 <= '8')) )
```

```
{
    Chill(ssp) ;
    LightOwnLine(ssp) ;

    sw = 0 ;
    printf(" Exit \n");
}

return err;

}
```

LightOwnLine.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightOwnLine.h"

/* white = '9'
  black < '9'
*/

int LightOwnLine(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */
{
   aErr err = aErrNone;
   int y = 1 ;

   while(y==1)

   {
   Flush(ssp) ;
   Write(ssp, 17, "#%+LINER       *\\");
   sleep(400) ;

   err = Read(ssp, 50);

   char l0 = ssp->pinputbuff[9] ;
   char l1 = ssp->pinputbuff[21] ;
   char l2 = ssp->pinputbuff[33] ;

   printf("l0 = %c \n" , l0);
```

```
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

//Analysis
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9') && (l1 == '9') && (l2 == '9') )

Forward(ssp);


//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') )

Right(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') )

Left(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') )

Right(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') )

Left(ssp);

//has to look for line when no line

//l2 l0 l1 on black

else if ( (l1 < '9') && (l0 < '9') && (l2 < '9') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");

}

//l2 l0 l1 on white

else if ( (l1 == '9') && (l0 == '9') && (l2 == '9') )
```

```
{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

}

return err;

}
```

3.2 Second Program: InterLineLight.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "InterLineLight.h"

//starts of Line converts to Light if necessary and interchanges
//line - light - line - light . .. .

/* white = '9'
  black < '9'
*/

int InterLineLight(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */
{
   aErr err = aErrNone;
   int j = 1;

   while(j==1)
{

   Flush(ssp) ;
   Write(ssp, 17, "#%+LINER       *\\");
   sleep(400) ;

   err = Read(ssp, 50);

   char l0 = ssp->pinputbuff[9] ;
   char l1 = ssp->pinputbuff[21] ;
   char l2 = ssp->pinputbuff[33] ;
```

```
    printf("l0 = %c \n" , l0);
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

//Analysis
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9') && (l1 == '9') && (l2 == '9') )

Forward(ssp);

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') )

Right(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') )

Left(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') )

Right(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') )

Left(ssp);

//When goes offline stops and starts light following
//l2 l0 l1 on black
else if ( (l1 < '9') && (l0 < '9') && (l2 < '9') )

{
    Chill(ssp) ;
    InterLightLine(ssp) ;
    j = 0 ;
    printf(" Exit line\n");
}

//l2 l0 l1 on white

else if ( (l1 == '9') && (l0 == '9') && (l2 == '9') )
```

```
{
    Chill(ssp) ;
    InterLightLine(ssp) ;

    j = 0 ;
    printf(" Exit line\n");
}
}
return err;
}
```

InterLightLine.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "InterLightLine.h"

//starts of Light converts to Line if necessary and interchanges
//light - line - light - line . . . .

int InterLightLine(serial_struct* ssp)

{
   aErr err = aErrNone;

   int i = 1;

   while(i==1)

  {

  Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //r3

   char f31 = ssp->pinputbuff[44] ;
   char f32 = ssp->pinputbuff[45] ;

   printf("\nf21 = %c \n" , f21);
   printf("f22 = %c \n" , f22);
```

```
    printf("\nf31 = %c \n" , f31);
    printf("f32 = %c \n" , f32);

//Analysis        9 = Light

//Forward      left      m1  2        3   m2       right

//Equal light

if (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Forward(ssp) ;

//More light toward r32 right

else if (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Right(ssp) ;

//More light toward r22 left

else if (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0')

Left(ssp) ;

else if ( (f21 == '1') || (f31 == '1') )

{
    Chill(ssp) ;
    printf("\n30 cm in front of Light or really really close.\n") ;
    i = 0;
}

//When no light stops and starts line following

else if ( ((f21 == '0') && (f22 <= '8')) && ((f31 == '0') && (f32 <= '8')) )

{
    Chill(ssp) ;

    InterLineLight(ssp) ;
    i = 0 ;
    printf(" Exit light \n");
}

}

return err;

}
```

3.3 Third Program: Line Variation
LineLightSim.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineLightSim.h"

//calls two boards at a time

/* white = '9'
   black < '9'
*/

int LineLightSim(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */
{
    aErr err = aErrNone;
    int y = 1 ;

    while(y==1)
    {
    Flush(ssp) ;

    Write(ssp, 17, "#%+LINER       *\\");
    sleep(400) ;

    err = Read(ssp, 50);

    char l0 = ssp->pinputbuff[9] ;
    char l1 = ssp->pinputbuff[21] ;
    char l2 = ssp->pinputbuff[33] ;

    printf("l0 = %c \n" , l0);
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

//Analysis
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9') && (l1 == '9') && (l2 == '9') )

Forward(ssp);
```

```
//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') )

Right(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') )

Left(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') )

Right(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') )

Left(ssp);

//When goes offline stops and starts light following

//l2 l0 l1 on black

else if ( (l1 < '9') && (l0 < '9') && (l2 < '9') )

{
    Chill(ssp) ;
    LineOwnLightSim(ssp) ;
    y = 0 ;
}

//l2 l0 l1 on white

else if ( (l1 == '9') && (l0 == '9') && (l2 == '9') )

{
    Chill(ssp) ;
    LineOwnLightSim(ssp) ;
    y = 0 ;
}
}

return err;

}
```

LineOwnLightSim.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LineOwnLightSim.h"

// This is a light following program that line following swaps to when no line
// Searches for light first  - forward dir
//checks for line while light following

int LineOwnLightSim(serial_struct* ssp)

{
   aErr err = aErrNone;
   int x=1 ;

   while(x==1)
{

   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //r3

   char f31 = ssp->pinputbuff[44] ;
   char f32 = ssp->pinputbuff[45] ;

   printf("\nf21 = %c \n" , f21);
   printf("f22 = %c \n" , f22);

   printf("\nf31 = %c \n" , f31);
   printf("f32 = %c \n" , f32);

   sleep(400) ;

//Check for line while light following

   Flush(ssp) ;
   aErr err = aErrNone;

   Write(ssp, 17, "#%+LINER       *\\");
   sleep(400) ;
```

```
    err = Read(ssp, 50);

    char l0 = ssp->pinputbuff[9] ;
    char l1 = ssp->pinputbuff[21] ;
    char l2 = ssp->pinputbuff[33] ;

    printf("l0 = %c \n" , l0);
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

//Analysis        9 = Light

//Forward      left      m1  2        3   m2       right

//Equal light

if (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Forward(ssp) ;

//More light toward r32 right

else if (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Right(ssp) ;

//More light toward r22 left

else if (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0')

Left(ssp) ;

//No light

else if ( ((f21 == '0') && (f22 <= '8')) && ((f31 == '0') && (f32 <= '8')) )

{
    Right(ssp) ;
    printf("\nSearching for Light :) \n") ;
}

//30 cm in front of light or really really close

else if ( (f21 == '1') || (f31 == '1')  )

{
    Chill(ssp) ;
    printf("\n30 cm in front of Light or really really close.\n") ;
    x = 0;
}
```

```
/*check for line
if line found exit this loop
start line following
*/

//Analysis
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9') && (l1 == '9') && (l2 == '9') )

{
    Forward(ssp);
    LineLightSim(ssp);
    x = 0;
}

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') )

{
    Right(ssp) ;
    LineLightSim(ssp);
    x = 0;
}

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') )

{
    Left(ssp) ;
    LineLightSim(ssp);
    x = 0;
}

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') )

{
    Right(ssp) ;
    LineLightSim(ssp);
    x = 0;
}

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') )
{
```

```
    Left(ssp);
    LineLightSim(ssp);
    x = 0;
}

}
return err;
}
```

Light Variation
LightLineSim.c

```
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightLineSim.h"

int LightLineSim(serial_struct* ssp)

{
   aErr err = aErrNone;
  int b = 1;

   while(b==1)

   {
   Flush(ssp) ;

   Write(ssp, 17, "#%+LITER       *\\");
   sleep(400) ;
   err = Read(ssp, 50);

   //r2

   char f21 = ssp->pinputbuff[32] ;
   char f22 = ssp->pinputbuff[33] ;

   //r3

   char f31 = ssp->pinputbuff[44] ;
   char f32 = ssp->pinputbuff[45] ;

   printf("\nf21 = %c \n" , f21);
   printf("f22 = %c \n" , f22);

   printf("\nf31 = %c \n" , f31);
   printf("f32 = %c \n" , f32);

//Analysis       9 = Light

//Forward     left      m1  2       3   m2       right
```

```
//Equal light

if (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Forward(ssp) ;

//More light toward r32 right

else if (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0' )

Right(ssp) ;
//More light toward r22 left

else if (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0')

Left(ssp) ;

//30 cm in front of light or really really close

else if ( (f21 == '1') || (f31 == '1') )

{
    Chill(ssp) ;
    printf("\n30 cm in front of Light or really really close.\n") ;
    b = 0;
}

//When no light stops and starts line following

else if ( ((f21 == '0') && (f22 <= '8')) && ((f31 == '0') && (f32 <= '8')) )

{
    Chill(ssp) ;
    LightOwnLineSim(ssp) ;

    b = 0 ;
    printf(" Exit LightOwnLineSim \n");
}

}
return err;
}

//searches for line when no light no line
//if searching for line and detects light will follow light
```

LightOwnLineSim.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <errno.h>
#define aWIN
#include "Access.h"
#include "Motor.h"
#include "LightOwnLineSim.h"

// This is a line following program that light following swaps to when no line
// Searches for line first  - turns right
//checks for light while line following

//Sideleft

/* white = '9'
   black < '9'
*/

int LightOwnLineSim(serial_struct* ssp)

                    /*
                    l1
                    l0
                    l2
                    */
{
    aErr err = aErrNone;
    int a = 1 ;

    while(a==1)
   {
    Flush(ssp) ;
    Write(ssp, 17, "#%+LINER       *\\");
    sleep(400) ;

    err = Read(ssp, 50);

    char l0 = ssp->pinputbuff[9] ;
    char l1 = ssp->pinputbuff[21] ;
    char l2 = ssp->pinputbuff[33] ;

    printf("l0 = %c \n" , l0);
    printf("l1 = %c \n" , l1);
    printf("l2 = %c \n" , l2);

     sleep(400) ;

//light check
    Flush(ssp) ;

    Write(ssp, 17, "#%+LITER       *\\");
    sleep(400) ;
    err = Read(ssp, 50);
```

```
    //r2

    char f21 = ssp->pinputbuff[32] ;
    char f22 = ssp->pinputbuff[33] ;

    //r3

    char f31 = ssp->pinputbuff[44] ;
    char f32 = ssp->pinputbuff[45] ;

    printf("\nf21 = %c \n" , f21);
    printf("f22 = %c \n" , f22);

    printf("\nf31 = %c \n" , f31);
    printf("f32 = %c \n" , f32);

//Line Analysis
//l0 on black l1 l2 on white go fwd

if ( (l0 < '9') && (l1 == '9') && (l2 == '9') )

Forward(ssp);

//l1 on black l0 l2 on white go Right

else if ( (l1 < '9') && (l0 == '9') && (l2 == '9') )

Right(ssp) ;

//l2 on black l0 l1 on white go Left

else if ( (l2 < '9') && (l0 == '9') && (l1 == '9') )

Left(ssp) ;

//l0/l1 on black l2 on white go right

else if ( (l0 < '9') && (l1 < '9') && (l2 == '9') )

Right(ssp) ;

//l0/l2 on black l1 on white go Left

else if ( (l0 < '9') && (l1 == '9')&& (l2 < '9') )

Left(ssp);

//has to look for line when no line

//l2 l0 l1 on black
```

```
else if ( (l1 < '9') && (l0 < '9') && (l2 < '9') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

//l2 l0 l1 on white

else if ( (l1 == '9') && (l0 == '9') && (l2 == '9') )

{
    Right(ssp) ;
    printf("\nSearching for line :) \n");
}

/*check for light
if light
then stop line
light follow
*/

//Analysis        9 = Light

//Forward      left      m1  2        3   m2       right

//Equal light

if (f22 == '9' && f32 == '9' && f21 == '0' && f31 == '0' )

{
    Forward(ssp);
    LightLineSim(ssp);
    a = 0;
}

//More light toward r32 right

else if (f22 < '9' && f32 == '9' && f21 == '0' && f31 == '0' )

{
    Right(ssp) ;
    LightLineSim(ssp);
    a = 0;
}
//More light toward r22 left

else if (f22 == '9' && f32 < '9' && f21 == '0' && f31 == '0')

{
```

```
    Left(ssp) ;
    LightLineSim(ssp);
    a = 0;
}

//combine with limit
//30 cm in front of light or really really close

else if ( (f21 == '1') || (f31 == '1') )

{
    Chill(ssp) ;
    printf("\n30 cm in front of Light or really really close.\n") ;
    //LightLineSim(ssp);
    a = 0;
}
}

return err;
}
```