UNIVERSITY OF
KWAZULU-NATAL

INYUVESI
YAKWAZULU-NATALI

The Cooperation of Heterogeneous Mobile Robot Configurations in Advanced Manufacturing Environments

Author:
Mr. Nicol Naidoo

Supervisor:
Prof. Glen Bright

Co-supervisor:
Dr. Riaan Stopforth

A dissertation submitted in fulfilment of the requirements for the degree of Master of Science in Engineering

in the

Mechatronics and Robotics Research Group
School of Engineering

November 2014

Declaration of Authorship

I, Mr. Nicol Naidoo, declare that this dissertation titled, 'The Cooperation of Heterogeneous Mobile Robot Configurations in Advanced Manufacturing Environments' and the work presented in it are my own. I confirm that:

- The research reported in this dissertation, except where otherwise indicated, is my original research.
- This dissertation has not been submitted for any degree or examination at any other university.
- This dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
- This dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then: (a) their words have been re-written but the general information attributed to them has been referenced, or (b) where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.
- This dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References sections.

Signed: ______________________________

Date: ______________________________

Declaration by Supervisor
As the candidate's Supervisor, I agree to the submission of this dissertation.

Signed: ______________________________

Date: ______________________________

Declaration of Publications

DETAILS OF CONTRIBUTION TO PUBLICATIONS that form part and/or include research presented in this dissertation (include publications in preparation, submitted, in press and published and give details of the contributions of each author to the experimental work and writing of each publication).

Publication 1: Conference Proceedings: 6th Robotics and Mechatronics Conference (RobMech), *"Material Flow Optimisation in Flexible Manufacturing Systems"*, 30-31 October 2013.
Mr. N. Naidoo: Research, simulation, application and main author.
Prof. G. Bright: Technical guidance, co-author.
Dr. R. Stopforth: Technical guidance, co-author.

Publication 2: Conference Proceedings: 8th International Conference on Intelligent Systems and Agents, *"Cooperative Autonomous Robot Agents in Flexible Manufacturing Systems"*, 15-17 July 2014.
Mr. N. Naidoo: Research, simulation, application and main author.
Prof. G. Bright: Technical guidance, co-author.
Dr. R. Stopforth: Technical guidance, co-author.

Publication 3: Journal Paper in Preparation: *"Support Vector Machine Learning in Multi-Robot Teams"*.
Mr. N. Naidoo: Research, simulation, application and main author.
Prof. G. Bright: Technical guidance, co-author.
Dr. R. Stopforth: Technical guidance, co-author.

Signed: ______________________________

Date: ______________________________

Abstract

Cooperation of Multiple Mobile Robot Systems (MMRS) have drawn increasing attention in recent years since these systems have the ability to perform complex tasks more efficiently compared to Single Mobile Robot Systems (SMRS). An implementation of a cooperative MMRS in a manufacturing environment can, for example, solve the issue of bottlenecks in a production line, whereas the limitations of a SMRS can lead to a lot of problems in terms of time wastage, loss of revenue, poor quality products and dissatisfied customers.

The study of cooperation in heterogeneous robot teams has evolved due to the engineering and economic benefits attribute as well as the existence of diversities in homogeneous robot teams. The challenge of cooperation in these systems is a result of the task taxonomies and fundamental abilities of each robot in the team; there is therefore a need for an Artificial Intelligence (AI) system that processes these heterogeneities to facilitate robot cooperation.

This dissertation focuses on the research, design and development of an artificial intelligence for a team of heterogeneous mobile robots. The application of the system was directed towards advanced manufacturing systems, however, it can be adapted to search and rescue tasks.

An essential component of the AI design is the machine learning algorithm which was used to predict suitable goal destinations for each mobile robot, given a set of input parameters. Mobile robot autonomy was achieved through the development of an obstacle avoidance and navigation system. The AI was also interfaced to a Supervisory Control and Data Acquisition System (SCADA) which facilitates end-user interaction – a vital ingredient to manufacturing automation systems.

Acknowledgements

I would like to acknowledge and express my sincere thanks to the following people and organisations:

- My supervisors, Professor Glen Bright and Dr. Riaan Stopforth for their supervision, guidance and provision of resources for the research. I thoroughly enjoyed every form of communication with them as it motivated me to do my best.
- To the directors, senior managers and HR staff at my company, Clifford Welding Systems, for their interest in developing my career as well as funding this degree.
- NAI Manufacturing for fabricating the metalwork for the robots at no cost.
- My family, for their unconditional love and support throughout this degree, especially my wife, Aurelle, who has played a pivotal role in the work presented in this research. I truly appreciate the sacrifices she endured during this degree and I commend her for the support and encouragement during the late nights and long weekends.
- My Lord and Saviour, Jesus Christ, who has always been with me and helped me through every challenge I faced in this degree. All praise and honour to Him for the multiple blessings He has graced me with and for the great privilege I have in getting to know Him.

Contents

List of Figures

List of Tables

Abbreviations

AGV	Automated Guided Vehicle
AMCL	Adaptive Monte Carlo Localisation
AI	Artificial Intelligence
BBS	Behaviour-Based Systems
BOM	Bill Of Materials
CAN	Control Area Network
CO	Cooperating Objects
ERP	Enterprise Resource Planning
GUI	Graphic User Interface
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IP	Internet Protocol
IR	Infra-Red
LMS	Laser Measurement System
LRF	Laser Range Finders
MCL	Monte Carlo Localisation
ML	Machine Learning
MMR	Multiple Mobile Robot
MRP	Manufacturing Resource Planning
ND	Nearness Diagram
Ni-MH	Nickel-Metal Hydride
PLC	Programmable Logic Controller
PT	Personal Transporter
RL	Reinforcement Learning
RMP	Robotic Mobility Platform

SAMCL	Self-Adaptive Monte Carlo Localisation
SCADA	Supervisory Control And Data Acquisition
SER	Similar Energy Region
SLA	Sealed Lead Acid
SLAM	Simultaneous Localisation And Mapping
SMR	Single Mobile Robot
SND	Smooth Nearness Diagram
SVM	Support Vector Machine
TCP	Transmission Control Protocol
URF	Ultrasonic Range Finders
VFH	Vector Field Histogram
VO	Visual Odometry
USB	Universal Serial Bus
WSN	Wireless Sensor Network

Chapter 1

Introduction

Every manufacturer faces a common challenge: to satisfy the delivery, quality and cost demands of the customer. Even though consumers do not directly influence the cost of products, manufacturers must maintain a competitive edge. These challenges always exist, since more companies have become advanced manufacturing plants by "making use of innovative technology to improve products or processes" [1].

Manufacturers have realised the insurmountable benefits of the practical implementation of robotics to their processes; one such genre of robots implemented in industry are Automated Guided Vehicles (AGVs), which have been used as material transporters within warehouses since the 1950's [2]. These mobile robots have proven to increase efficiencies and reduce costs while operating autonomously alongside humans.

Due to the complexities and variety of processes in an advanced manufacturing environment, different (heterogeneous) robots are assigned to perform specific tasks. These tasks are occasionally interrupted by plant disturbances, such as machine failures and inefficiencies. In this event, manufacturing bottlenecks are created which must be resolved quickly so that plant downtime and operational costs do not escalate. The research in this dissertation thus focuses on establishing cooperative teamwork behaviour among heterogeneous mobile robots as they contribute towards the relief of production bottlenecks. It is clear that this may be impractical in some scenarios due to the variation in task categories and robot capabilities but this is discussed further in chapter 3.

1.1 Robot heterogeneity in manufacturing environments

Multiple Mobile Robot (MMR) systems have drawn increasing attention in recent years since these systems have the capability to efficiently perform complex tasks through team work [3]. A Single Mobile Robot (SMR) system simply does not have the advantage of executing distributed tasks as a MMR would. Moreover, having a MMR system with resource-bounded robots is much more cost-effective and robust than implementing a single, powerful robot whose resources may be wasted.

A MMR team can either consist of homogeneous or heterogeneous members. The study of cooperation in heterogeneous robot teams has developed due to the following factors inherent in manufacturing environments [4]:

- Engineering advantages: there may be complex tasks that require the simultaneous use of combinations of sensors and robots; it can be challenging and impractical to design individual robots with multiple functionalities.
- Economic reality: It is practical and much more economical to distribute the specialised abilities of simple robots rather than employing a team of large, expensive robots.
- Diversity: A team of homogeneous robots can eventually become diverse over time due the differences in sensor tuning or calibration, wear and tear, and changes in construction due to robot maintenance.

Heterogeneous mobile robot teams can potentially find their way in many real-world applications such as search and rescue, mine and planetary exploration, security and surveillance, and hazardous waste clean-up. In relevance to manufacturing environments, areas include materials handling and transportation, and materials processing.

A materials handling and transportation application would involve the routing and transporting of products or materials between processing stations in the plant. An example of such an application is the Autonomous Materials Handling Robot for Reconfigurable Manufacturing Systems [5]; although this particular system involves a single mobile robot, the principle can be applied to a larger system of heterogeneous mobile robots where each robot must cooperate with the others so that individual and global goals are achieved while inhibiting path conflicts.

1.2 Bottlenecks in manufacturing environments

Bottlenecks occur when resources become overloaded. In relation to the manufacturing environment, this happens when the material build up rate at a process point (resource) is greater than the processing rate. The diagram in figure 1.1 gives an overview of some possible sources for production bottlenecks and a basic method used to relieve them.

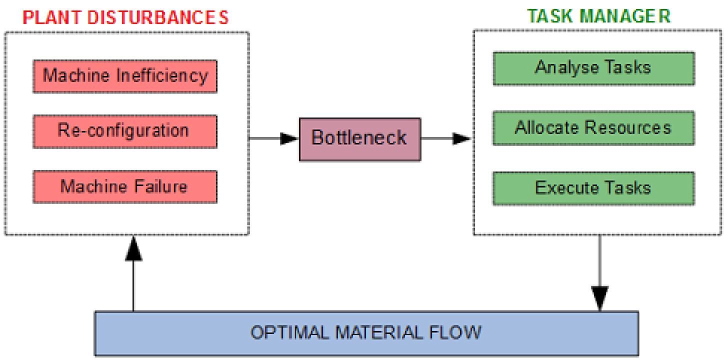

FIGURE 1.1: General representation of the bottleneck problem in manufacturing plants.

Reconfiguration and *machine inefficiency* plant disturbances can be identified in manufacturing environments which comprise of one or more process lines, whilst *machine failure* disturbances occur in plants which rely on multiple lines to increase product throughput. The bottleneck sources must be managed efficiently in order to re-establish an *optimal material flow* in the manufacturing process. The *task manager* must therefore determine the work requirements, allocate the competent resources to accomplish the tasks and finally ensure that they are executed.

1.3 Productivity and supply chain management

A typical "task manager" utilised by many manufacturers is the Manufacturing Resource Planning (MRP) system. This is a computer-based information system that uses sales orders, forecast demands, a production schedule as well as information such as the Bill of Materials (BOM) and available inventory to schedule and order dependent-demand inventories. The ordering of the precise quantity of inventory is a crucial step in the process of maintaining a good standard of supply chain management in the face of changes in market demand.

Supply chain management becomes complex when manufacturing plants comprise of more than one production line to increase throughput or produce various products. Redundant work stations can also exist to minimise bottleneck conditions or aid in the event of machine failures and preventative maintenance services. In these conditions, it is absolutely vital that the resources (robots and people) in the plant are efficiently managed so that productivity is increased and operational costs are minimised. The scheduling of robot resources is much more complicated to handle since robots are good at working to perform well on singular tasks but poor at being coordinated to multi-task, hence there is a need for the development of task management systems for robot resources. The design of such a system is much more convoluted when heterogeneous robots are involved.

1.4 Literature Survey

The focus of this literature survey pertains to indoor, heterogeneous mobile robot systems. The key components of typical autonomous navigation systems are discussed in each section.

1.4.1 The Mechatronic system

The discipline of Mechatronics originally included the combination of mechanics and electronics however, due to an advancement in technology throughout the years, Mechatronics now involves an integration of mechanical, electronic, computer and control systems as depicted in figure 1.2 [6].

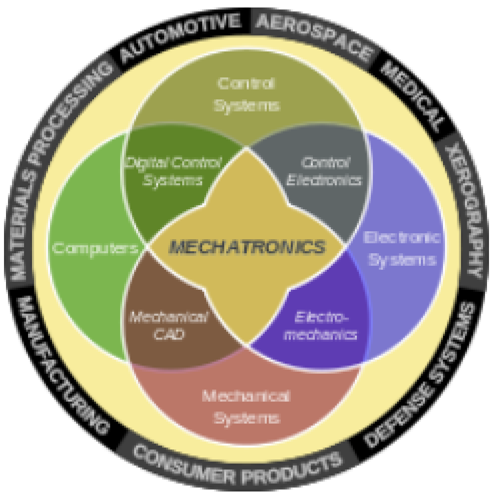

FIGURE 1.2: Venn diagram of Mechatronic systems

An autonomous industrial robot is a typical example of a Mechatronic system since it combines the disciplines of electronics, mechanics, control theory and computing systems to function in an advanced manufacturing environment. In this context, a well designed robotic system can significantly contribute to an improvement in manufacturing processes by enhancing product throughput and maintaining high levels in system reliability.

Navigation is one of the most challenging competences required of a mobile robot; four building blocks of navigation are identified [7]:

- Perception: the robot must interpret the raw data extracted from the sensors to obtain meaningful data.
- Localisation: the robot must identify its position in the environment.
- Cognition: the robot must decide how to achieve its goal position in the environment by planning a path.
- Motion Control: the robot must drive its motor outputs to achieve mobility in the planned direction.

In addition to these building blocks, the robot must be able to avoid obstacles in route to its goal. Figure 1.3 illustrates a structure of the components required in the design of an autonomous mobile robot system; they are further discussed in the subsequent sections of this chapter.

1.4.2 Perception

A core component of any autonomous mobile robot application is the ability of the robot to gather information about its environment. This is accomplished through the extraction and interpretation of meaningful data from sensors which are usually installed on the robotic platform. Low performance sensors that generate signals with low data resolution or susceptible to signal interference will cause an inaccurate representation of the environment. This negatively impacts the robot's autonomous performance in the environment, thus it is vital that the correct sensors are chosen in the design process.

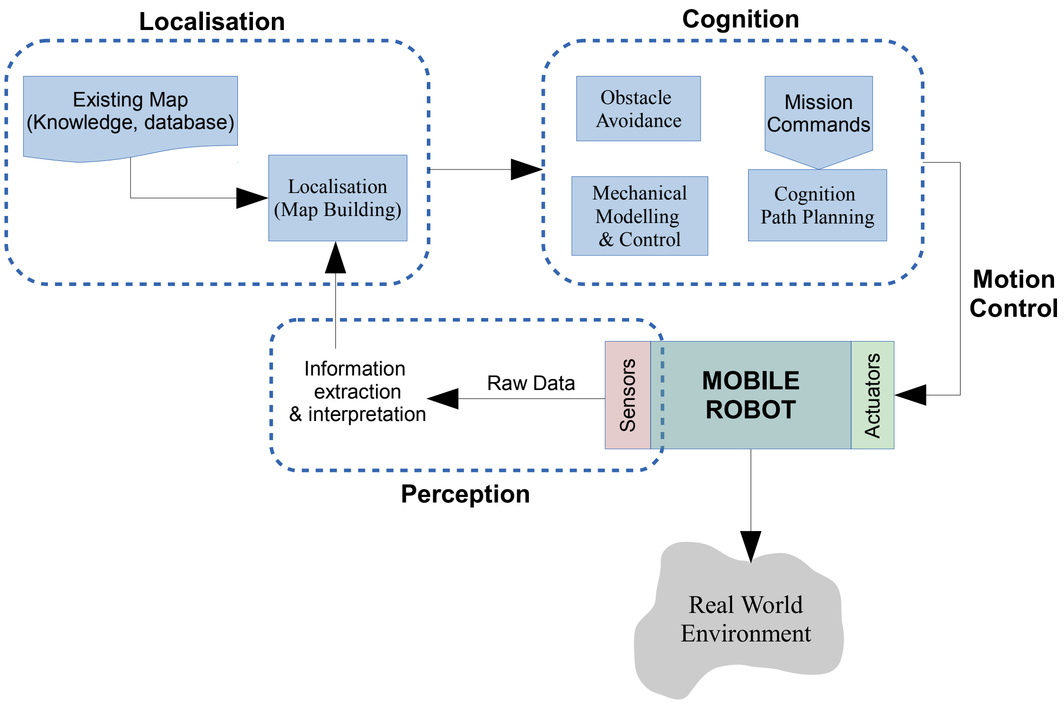

FIGURE 1.3: Control structure for autonomous mobile robots

Sensors can be classified using two functional axes [7]: 1) active or passive and 2) proprioceptive or exteroceptive. Proprioceptive sensors measure values internal to the robotic system whilst exteroceptive sensors gather information from the robot's environment. The following sections discuss some types of sensors used in mobile robotic applications.

1.4.2.1 Odometry

Odometry is a sensor system that works by integrating incremental data over time. It is a relative positioning system where the *dead reckoning* method determines the current position of the robot from previous positions. The distance and direction travelled by the robot are calculated by rotary encoders which are used to count the number of revolutions of each wheel. Odometry is a popular sensor system since it provides good short-term accuracy, is a relatively inexpensive option, and allows for high sampling rates [8].

There are a few disadvantages in the odometry system such as the error integration due to drift and slippage [9]. Figure 1.4 [10] shows an example of how the odometry error can grow as the robot progresses in a straight line. The ellipses represent position uncertainties of the robot in the x, y direction; the y-error is more prominent due to the uncertainty about the orientation of the robot.

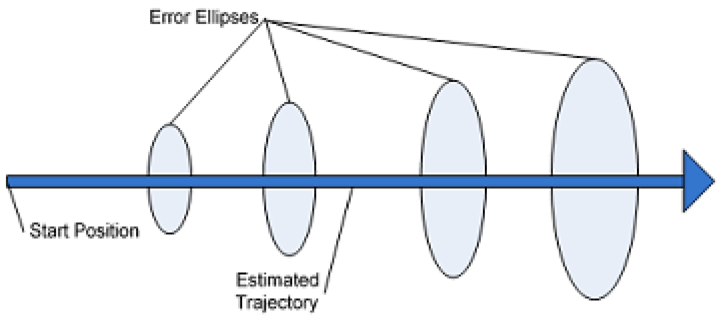

FIGURE 1.4: Robot odometry straight line error accumulation

A further disadvantage to odometry is its sensitivity to terrain (bumps or uneven floors) as the system cannot detect surface irregularities; other disadvantages include uncertain robot geometry, differences in wheel diameters, kinematic imperfections, and non-systematic errors [11].

Another area of research in the odometry sensing domain is Visual Odometry (VO) where the position and direction of the robot are determined through an analysis of changes that movement induces on the images taken by cameras. Research into VO began in the early 1980's with H. Moravec's work on the Stanford cart [12] but was popularised by the research of D. Nister et al. [13]. The advantages VO over conventional wheel odometry is that VO is not affected by slippage or uneven terrain; it has also been demonstrated that VO provides more accurate trajectory estimates, with the relative position error ranging from 0.1% to 2% [14]. The disadvantages of VO are the high processing loads on the computer system and the costly execution of the solution. There are many robotic applications of VO, however, its implementation on the Mars exploration rovers [15] is an outstanding example.

1.4.2.2 Range finder sensors

Range finder sensors are used to measure the distance from the observer to a particular target. In the field of robotics, there are three types of range finders commonly used; ultrasonic, laser, and infra-red.

Ultrasonic Range Finders (URF) operate on the time-of-flight principle: the ultrasonic signal is transmitted from the sender, reflected off an obstacle and then received by the receiver; the distance from the obstacle is calculated by using the time taken to receive the signal and the speed of sound in air. Despite their low cost and availability, ultrasonic sensors are sensitive to noisy environments and other sound waves [16]. Other factors which may influence measurements are temperature, humidity and barometric pressure of the environment since the speed of sound in air varies accordingly.

Laser Range Finders (LRF) are an excellent choice to use in mobile robotic applications due to the accurate range data provided [17]. The most common form of technology used to calculate distance in LRFs is, as with URFs, the time-of-flight principle; the difference is a laser beam is propagated through air and the speed of light (not sound) is used as the constant in the distance equation. Another technology also used involves measuring the phase shift of multiple frequencies on reflection and then applying simultaneous equations to solve for the actual distance. The drawback of using a LRF is the high cost implication; a good quality LRF can cost a minimum of R20 000. Research has been pioneered in the design of a low-cost laser distance sensor [18], however, it has not been released to the commercial market due to its integration on the NEATO XV-11 Robotic All-Floor Vacuum Cleaner [19].

Infrared (IR) sensors are low cost devices, commonly used as proximity detectors for obstacle avoidance in mobile robotic applications. Due to their non-linear characteristics and dependence on the reflectance from surrounding objects, the back-scattered IR signal is highly inaccurate for the purpose of ranging [20]. Some IR sensors produce medium distance resolution at long ranges [21], however, these products are very costly.

1.4.2.3 Inertial sensing and navigation

Like odometry, Inertial Navigation Systems (INS) also use the dead reckoning method to determine the position and orientation of the robot. The principle of operation in these systems involves the employment of an Inertial Measurement Unit (IMU) which contains three-axis accelerometers and gyroscopes. The accelerations in each of the three directional axes are continuously sensed and integrated over time to derive the position and velocity of the mobile robot at high rates, typically at 100 times per second [22]. The accelerometers thus sense the accelerations and the gyroscopes measure the angular rates. An integration of angular velocity with time gives angle data, whilst the double integration of acceleration with time yields distance travelled.

A significant disadvantage in INS applications is the amplification of low frequency noise and sensor biases due to the integrative nature of the system [23]. This accumulation error problem (as with odometry) produces long term inaccuracies which can be rectified by the use of additional external signals. In an aim to reduce the accumulation errors found in dead reckoning (without the utilisation of exteroceptive sensors), a dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding [24], combined odometry and the INS measurement data to determine position and orientation, then compensated for the errors by implementing a Kalman filter design (the use of Kalman filters are discussed further in section 1.4.3). The method showed correction of bias errors, inherent to inertial sensors, and compensated for the yaw angle errors that generate position errors in odometry.

1.4.3 Localisation and mapping

Localisation is a key component in autonomous mobile robot systems since it is imperative for the robot to identify its position in known or unknown environments. It has become a popular area of research in the past few decades and as a result, significant contributions have been made. Literature often discusses localisation and mapping (or map building) in conjunction with one another since the problem of determining where a robot is, is relative to a spatial model of the robot's environment.

There are five key challenges attributed to the robotic mapping problem [25]:

- Measurement noise: the statistical dependent nature of sensor measurement errors is a complicating factor and must be accommodated to ensure success in the map building process.
- Dimensionality of entities: the dimensionality of the mapping problem is determined by how many numbers it may take to describe the environment. A detailed two-dimensional floor plan often requires thousands of numbers whilst a 3D visual map can require millions of numbers; this increases the complexity of the mapping problem.

- Data association: sensor measurements taken at different points in time must correspond to the same object in the environment. An accumulated pose error of the robot causes the number of possible localisation hypothesis to increase exponentially over time, resulting in inaccurate mapping.
- Change in environment: map building in dynamic environments is a challenging problem and research in this area is relatively new. Most mapping techniques are designed for situations where the environment is static.
- Robot exploration: exploring robots work with partial, incomplete maps and must factor contingencies that may arise during map building.

Almost all popular algorithms used for localisation and mapping have one common feature: they are probabilistic [25]. The reason for this approach is due to the uncertainties characterised by the mapping problem as discussed above.

A common technique used in the localisation and mapping of mobile robotic applications is the Kalman filter [26] which uses a Gaussian probability density representation (figure 1.5 [7]) for the robot position and environmental data obtained from the sensors [27].

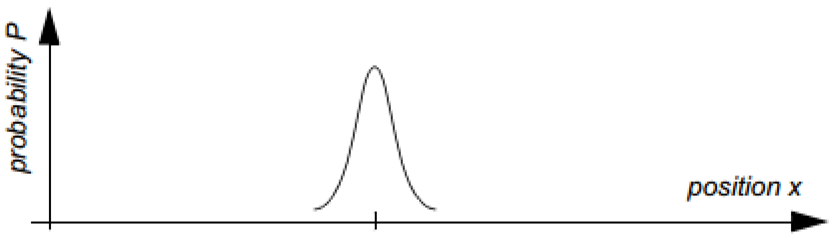

FIGURE 1.5: Gaussian probability density function centred at a single value

The filter records the uncertainties of state variables and minimises the overall mean square error in each update; as a result, it enables a Simultaneous Localisation and Mapping (SLAM) approach where the robot builds the map and uses the same dynamic map to localise. SLAM is actually a problem, not a solution, however, the term is closely affiliated with algorithms that use Kalman filters for estimating the map and pose of the robot [25].

Unlike the Kalman filter which represents the robot's belief state using a single Gaussian function, Markov localisation applies a probability distribution for every possible position of the robot on the map (figure 1.6 [7]). This method is challenged for larger maps since it would require a significant amount of computing resources to evaluate every possible position in the distribution, however, it can be optimised by randomised sampling methods such as particle filter [28] or Monte Carlo Localisation (MCL) algorithms [29] where only a random distribution of the positions are evaluated.

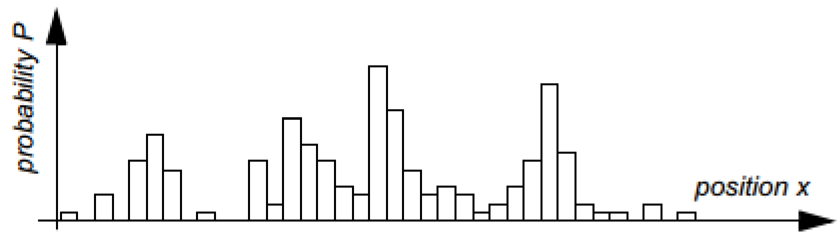

FIGURE 1.6: Grid map with probability values for all possible robot positions

Adaptive Monte Carlo Localisation (AMCL) is an improved method used to determine the robot's position by narrowing the search of local maxima areas of random position samples [30].

An advancement to the MCL algorithm termed Self-adaptive Monte Carlo Localisation (SAMCL) [31] has been investigated, where: 1) a pre-caching technique is employed to reduce the on-line computational burden, and 2) a concept of Similar Energy Region (SER) is defined, which is a set of poses that have similar energy with the robot in the robot space; samples are distributed in the SER instead of being randomly distributed in the map. The localisation simulations revealed a better performance from SAMCL compared to MCL and extended Kalman filter algorithms.

1.4.4 Cognition

The cognition component of an autonomous robot system represents the decision-making and execution processes required to achieve the goals of a particular mission. In mobile robotics, a robust autonomous system is directly linked to its cognitive competency. This is the level to which the robot system responds to mission commands by path planning goals, interpreting situations, and avoiding obstacles in the environment.

1.4.4.1 Path planning

Path planning involves the application of a strategic trajectory algorithm to achieve robot goal positions. The first step of the process is to transform the environmental model into a discrete map that is suitable for the planning algorithm; literature identifies three general strategies used in path planning algorithms [7]: road map, cell decomposition, and potential field.

In the road map approach, the environmental workspace is mapped to a one-dimensional graph containing vertices, including the start and goal locations; a collision-free path between vertices is then searched. Literature identifies some common road map methods such as *visibility graph* [32] and the *Voronoi diagram* [33]. A significant drawback of the visibility graph approach is that solution paths tend to take the robot as close to the obstacles as possible, whereas the Voronoi diagram, in contrast, maximises the distance between the robot and the obstacles in the environment [7].

Cell decomposition path planning involves the identification of areas or cells that are free and those that are occupied by objects; an appropriate path between the initial and goal locations is created by connecting adjacent free cells. There are two types of cell decomposition techniques used in planning: *exact cell decomposition* and *approximate cell decomposition*; the latter is more popular due to its grid-based environmental representation. The disadvantage of the exact cell approach is the dependence of computational efficiency upon the density and complexity of the objects in the environment [7].

The potential field strategy implements the robot as a point under the influence of an artificial potential field. A gradient is created across the robot's map which causes it to travel towards the goal positions from previous positions [34]. The goal positions act as attractive forces, whereas the obstacles act as repulsive forces. Literature discusses the potential field strategy as an elegant and simple approach to path planning [35], however, it can cause the robot to fall in a local minimum if this is not accommodated for in the high level software design.

1.4.4.2 Obstacle avoidance

While path planning involves a global-based strategy, obstacle avoidance techniques are applied to situations that are local to the mobile robot. The principle function of an obstacle avoidance algorithm is to use the current measurements from sensors to shape the direction of the robot's path when unexpected obstacles (e.g. humans or other robots) are encountered. It forms an integral component of a robot's navigation system since it significantly contributes to the safety aspect of the design.

A wide variety of obstacle avoidance algorithms are available, ranging from simple methods which stop the motion of the robot before an obstacle, to more sophisticated schemes that control the robot to detour obstacles.

The Bug 1 [36], and its improvement, Bug 2 [37] algorithms follow a simplified approach by moving directly towards the goal; if an obstacle is encountered, it is contoured by the robot until motion in the direction of the goal is again possible. Bug algorithms do not apply robot kinematics which is a shortcoming for non-holonomic robot systems. A further drawback is the impact of sensor noise on robot performance due to the most recent sensor data taken for these algorithms.

In known, static environments, potential field path planning algorithms can evaluate the potentials offline and provide the robot drive system with the calculated velocity vector; however, the algorithm can incorporate an obstacle avoidance component, thus operating in an online mode. The local minimum drawback of potential field algorithms can be mitigated by using an extended version of the strategy as discussed in [38]. In this approach, the repulsive force due to an obstacle is considered as a function of distance and orientation relative to the obstacle.

The Vector Field Histogram (VFH) method [39] creates a polar histogram of the environmental occupancy in close proximity of the robot. The polar histogram is first evaluated to select the best suitable sector among all sectors in the histogram with low polar obstacle densities and then proceeds to steer the robot in the direction of the sector chosen. An advantage of VFH over potential field methods is the absence of the local minima problem due to the non-existence of repulsive or attractive forces. The VFH+ method is an enhanced version of VFH and can be reviewed in [40].

1.4.5 Middleware in robotic networks

In robotic networks, the middleware is an abstraction layer found between the operating system and software application layer as shown in figure 1.7.

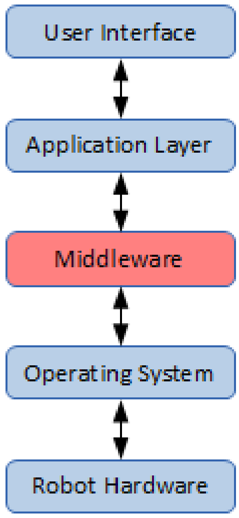

Figure 1.7: Middleware layer in robotic networks

The purpose of the middleware is to simplify the software design, manage robot hardware heterogeneities and reduce development costs [41]. In a network of robots where collaboration is required, the middleware is a key component of the system due to the following requirements:

- Hardware heterogeneity: robots and sensors in the network may be different; this creates a software development challenge to the higher layers in the network if the heterogeneity is not masked. The middleware is required to communicate with heterogeneous hardware and provide standard, simplified interfaces to the application layer. The implementation of such a system will eliminate low level complexities in the network, thus creating a simple, modular approach to the design of the application program.

- Self-operable: the middleware layer must be able to operate in the software tier without any direct intervention from the user. It should also reconfigure when required and optimise processes so that the robot network efficiencies are maintained.

- Collaborative interface: a robot team will require a common collaborative interface that is supported by the middleware. Each robot must be able to communicate through this interface so that the group task can be achieved efficiently.

- Modular design: the middleware must contain a modular software architecture so that an upgrade to a sensor driver, for instance, does not implicate a change in the design of the middleware.

Literature discusses a wide variety of middleware interfaces that are used in robotic applications; only the *Player Project* will be discussed in the remainder of this section. For a comprehensive survey on robot middleware, the reader is referred to *Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography* [41] and *A Review of Middleware for Networked Robots* [42].

The Player Project, developed by B. P. Gerkey et al. [43], is a robotic network server that provides a transparent interface for robot and sensor control. The project is supported by robotic researchers and is widely used all over the world [44]. It is a client-server based system consisting of the following components:

- Player: the server that communicates with the robot hardware (sensors and actuators) through device drivers and provides the client with standard interfaces to them.
- Stage: a two-dimensional simulator used to fast prototype development without the need for using actual robot hardware.
- Gazebo: a three-dimensional simulator.

The Player Project can be installed on a computer connected to the robot and provides interfaces to sensors and actuators over an IP network. Client application programs can communicate with the robot's hardware using standard interfaces via Player over a TCP socket. It is also possible for a client program, located anywhere in the network, to access any device [43]; thus a particular robot can view the environment by use of another robot's sensor.

1.4.6 Cooperation in mobile robot teams

In recent years, there has been a great research interest in cooperative mobile robotics. An advancement in industrial technology has seen the need for distributed applications in robotic systems where teams of robots are required to solve tasks intelligently and efficiently. The definitions of cooperation in robotics literature include [45]:

- "joint collaborative behaviour that is directed toward some goal in which there is a common interest or reward" [46]
- "a form of interaction, usually based on communication" [47]
- "(joining) together for doing something that creates a progressive result such as increasing performance or saving time" [48]

1.4.6.1 Control architectures

Research [4] describes a categorisation of multiple mobile robot networks into 1) collective swarm systems and 2) intentionally cooperative systems – classified further as being strongly cooperative or weakly cooperative solutions (figure 1.8).

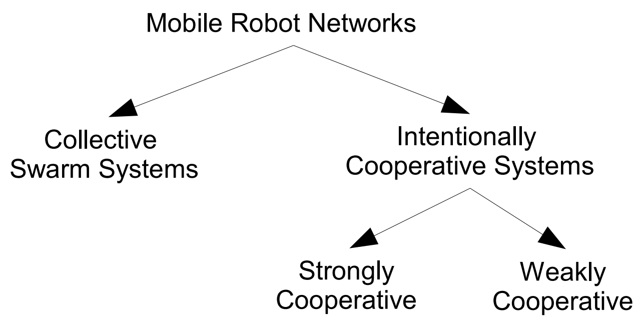

FIGURE 1.8: Classification of multiple mobile robot systems

Collective swarm systems are attributed to a team of homogeneous mobile robots that yield team behaviours while establishing little or no communication between each member. Swarm robotics [49] is an emerging area of research where robot teams tend to resemble the characteristics of biological societies such as ants, birds and bees.

Heterogeneous mobile robot systems are intentionally cooperative systems since the robots are aware of other robots in the team and work together based on the state, actions or capabilities of their team-mates to achieve a particular goal [4]. Depending on the design of the robotic network, solutions may form high levels of communication and synchronisation between robots (strongly cooperative), or allow for periods of functional independence among robots (weakly cooperative).

The method of cooperation system applied to a robotic team introduces the mention of some typical control architectures: centralised, decentralised, hierarchical, and hybrid systems.

- Centralised architectures control the entire team of robots from a single point. This approach is unreliable due to a single point of failure as well as the impracticality of maintaining a high communication frequency between robots for real-time control.
- Decentralised architectures is a common approach to most robot teams where each robot assumes control based on a knowledge base of their local situation. Unlike the centralised system, this type of control is highly robust, however, it may be a challenging task to maintain global control of the team due to the local behaviours of individual members.
- Hierarchical architectures resemble military systems where each robot oversees the control of a group of robots. It is better than centralised architectures in terms of reliability but is still susceptible to team failure due to the dependence on robots structured higher up in the hierarchy.
- Hybrid architectures is a combination of centralised and decentralised forms where local control and higher level plans are achieved, thus establishing a robust and potentially efficient control system. Hybrid architectures are applied in many multi-robot applications [4].

1.4.6.2 Related research

A great amount of research has been pioneered in the area of cooperation in multiple homogeneous mobile robot systems, however, since the focus of this thesis relates to heterogeneous robotic systems, a selected few of these approaches will be discussed.

The ALLIANCE architecture [50] is an early work studying cooperative behaviour in small to medium sized heterogeneous mobile robot teams. Local robot control (e.g. obstacle avoidance) and higher level task assignments are achieved by means of using "motivational behaviours" to activate "behavioural sets" belonging to each robot in the team.

The selection of the appropriate action is based on: 1) the robot's internal states, 2) the activities of other robots (obtained through broadcast communications), 3) environmental conditions, and 4) the requirements of the task. Successful fault tolerant implementations of the ALLIANCE architecture include a hazardous waste clean-up mission [51] and a cooperative box-pushing experiment [52]. An advancement to ALLIANCE is L-ALLIANCE [52] which improves the individual robot's action selection by using the knowledge learned from previous experience.

An integrated testbed for cooperation of heterogeneous mobile robots, Wireless Sensor Networks (WSNs) and other Cooperating Objects (COs) is discussed in [53]. The purpose of the testbed is to facilitate the development process of algorithms and techniques used in the research of heterogeneous COs. The architecture considers all COs at the same level, thus allows for various schemes such as multi-robot, WSN, and robot-WSN experiments.

A tiered architecture was investigated (illustrated in figure 1.9 [54]) that integrates, a graphical user interface, task planner, executive layer, and autonomous navigation system to control multiple heterogeneous robots [54].

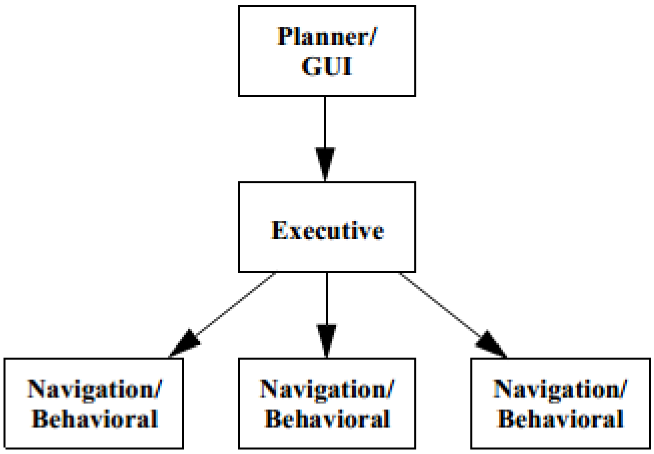

FIGURE 1.9: Tiered architecture for autonomous robots

The top layer in the tier consists of a task planner together with a graphical user interface that facilitates a user-friendly interaction in the task assignment process.

The bottom layer is a component in each robot and performs probabilistic methods for its autonomous navigation in the environment. Navigation consists of three components:

localisation (using MCL as discussed in section 1.4.3), motion planning, and obstacle avoidance.

The middle, executive, layer is an interface between the high and low levels in the tier. It coordinates the robots by distributing the tasks accordingly.

A multi-robot system [54], consisting of four robots, was tested in an indoor environment. The deployment proved to be efficient and robust, however, explicit coordination behaviours had to be implemented to achieve these results.

1.5 Problem statement

Identifying and managing bottlenecks in a manufacturing environment can have a significant impact on a company's product throughput and profit margins. In the context of an advanced manufacturing environment where dissimilar mobile robots are used in discrete processes, the idea of cooperation between robots when there is a need can prevent bottlenecks, improve material flow and thus contribute to the upkeep of a good supply chain management system. A middleware intelligence system is to be developed for this specific task. The system will aid any member in a team of heterogeneous robots in task decision making, particularly in a manufacturing environment. Each robot in the system must be capable of moving autonomously in the known environment while avoiding obstacles and maintaining a teamwork approach in the resolution of common goals.

1.6 Research objectives and contributions

The primary objectives of this research include:

1. Investigate the need for heterogeneous mobile robot cooperation, using the Segway RMP200, RMP400 and the Performance PeopleBot.

2. To research the system requirements for the design of a cooperative heterogeneous mobile robot team.

3. To research and design a Mechatronic control system for the cooperation of different mobile robots. This system involves the development of the middleware intelligence and integration of electronic hardware interfaces.

4. To test and validate the performance of the control system in simulated and real-world environments against the following key indicators:

- Robustness: The ability of the system to continue its operation when a failure occurs with a mobile robot.
- Reliability: The ability of the mobile robots to efficiently detect and avoid obstacles, and to be able to operate in a "safe mode" under certain fault conditions, thus minimising possible human injuries and damage to the environment.
- Plant integration: The user friendly design of the system to the plant engineer as well as the ability of the system to provide useful data to the plant MRP and/or Enterprise Resource Planning (ERP) networks.

1.7 Design specifications

The design specifications for this research topic are classified below according to mechanical, electronic, Artificial Intelligence (AI), communication and Graphical User Interface (GUI) categories; the AI specifications are split into expected individual and group robot behaviours.

Mechanical specifications (individual)

The following mechanical specifications need to be considered:

- Operate at linear velocity no greater than 0.2 m/s.
- Operate at angular velocity no greater than 0.4 rad/sec.

Electronic specifications (individual)

The following electronic specifications must be met:

- Position range finder sensors at front of robots to measure forward (and not rear) distances.
- Sensors must be powered from rechargeable batteries.

AI specifications (individual)

The following AI specifications for each robot need to be considered:

- Emergency stop (estop) if another robot or human is within a radius of 0.25 meters.
- Turn in tight spaces within an estop distance (0.25 meters) from obstacle.
- Plan and navigate along an efficient path towards the goal, avoiding obstacles.
- Teach the robot agent on the best task to execute, in either online or offline modes.

AI specifications (group)

The following AI specifications for the robot team need to be considered:

- Demonstrate team work when bottlenecks exist.
- Demonstrate robot assistance during robot failures.

Communication specifications

The following communication specifications for system need to be considered:

- Wireless communication between each robot and SCADA.
- TCP socket protocol exchange between each robot and SCADA.
- No direct inter-robot communication allowed.

GUI specifications

The following GUI specifications for the Supervisory Control and Data Acquisition (SCADA) system need to be considered:

- Show the manufacturing plant process, data and bottlenecks.
- Start the manufacturing process from GUI.
- Show the locations of robots in the plant.

1.8 Research publications

1. N. Naidoo, G. Bright, R. Stopforth. "Material Flow Optimisation in Flexible Manufacturing Systems". In *Proceedings of the 6th Robotics and Mechatronics Conference (RobMech)*. Durban, South Africa. 30-31 October 2013.

2. N. Naidoo, G. Bright, R. Stopforth. "Cooperative Autonomous Robot Agents in Flexible Manufacturing Systems". In *Proceedings of the 8th International Conference on Intelligent Systems and Agents*. Lisbon, Portugal. 15-17 July 2014.

1.9 Chapter summary

This chapter introduced the area of heterogeneous mobile robotics in manufacturing environments. The issue of bottlenecks was discussed and the literature survey addressed the necessary components of an autonomous mobile robot system, namely: perception, localisation, and cognition. In addition to these, the importance of implementing a robotic middleware layer, particularly in heterogeneous robot systems, was outlined. The problem statement was defined to summarise the constraints of the research problem space and the research objectives were also listed to establish a clear representation of the work involved in this dissertation. The chapter concluded by categorically listing the design specifications of the research topic and a list of publications by the author.

Chapter 2

Mechatronic Architecture Design

This chapter discusses the control architecture design as well as the mechanical and electronic hardware associated with the research topic.

2.1 Architecture design overview

The primary objective of this research is to establish a cooperation mechanism between different robots so that bottlenecks in manufacturing environments can be mitigated. The first step in the process of achieving the primary objective is to identify the control architecture required in the design of such a system. In section 1.4.6.1, strongly and weakly cooperative systems were discussed and four control architectures were identified, namely: centralised, decentralised, hierarchical, and hybrid systems. The scheme chosen in this design is a weakly cooperative hybrid system where majority of the control (for robot motion, obstacle avoidance, path planning and goal decision making) is decentralised; the centralised portion is necessary for the integration of the robot system to the manufacturing plant data acquisition system, an element of the design that will be discussed in the next chapter. The key benefits of the weakly cooperative hybrid architecture are:

- Localised control allows fast response time for robots to react to obstacles and changes in its path during navigation.

- Robots do not depend on a synchronised communication scheme between team members which promotes operational independence.
- The decentralised control characteristic strengthens the robot's ability to assist another team member during robot-failure conditions, this is a requirement for the *robustness* key indicator (outlined in section 1.6).

The diagram in figure 2.1 gives an overview of the architectural design of the research. Each robot in the team (conceptually not limited to three members) contains a local computer. The computer contains the operating system and middleware responsible for the decentralised motion control of the robot by obtaining data from sensors and accordingly driving the actuators (or motors). Robots are assigned to primary tasks (indicated by the black solid lines) and/or secondary tasks (indicated by the black dashed lines); secondary tasks are executed by a robot when there is a bottleneck contributed by machine inefficiencies or failures, reconfiguration, or robot failures. The centralised control facet of the architecture is between the local computers and the remote computer and occurs whenever the middleware of the robot is ready to make a global decision concerning the source and destination goals (tasks).

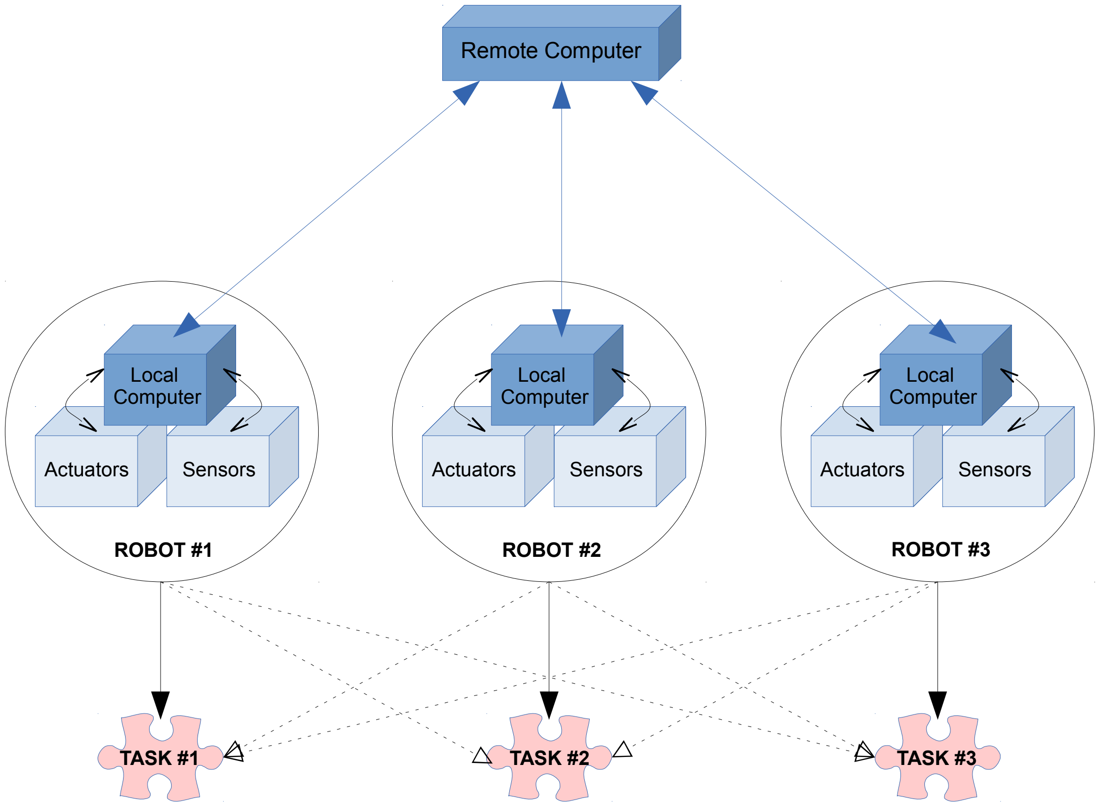

FIGURE 2.1: Robot cooperation architecture

2.2 Robot hardware

The three mobile robots used in this research topic are the Performance PeopleBot, Segway RMP200, and Segway RMP400. The platforms were chosen on the basis of their availability, they are mainly used for research purposes and not suited for manufacturing environment applications, which is acceptable for this research since the objective is to establish the concept of a cooperating team of heterogeneous mobile robots, irrespective of their abilities and functionality.

The subsections below discuss the functionality of each robot and outlines the technical specifications as well.

2.2.1 Performance PeopleBot

The Performance PeopleBot (shown in figure 2.2 [55]) is a mobile robot that was designed by *Activ*Media [56] for the purpose of human-robot interactive research and applications. The robot is equipped with a reversible two-wheel differential motor drive system and a balancing caster; this ensures that the steering control can be achieved by varying the relative rate of rotation of each wheel. The Peoplebot is also installed with four

FIGURE 2.2: Performance PeopleBot robotic platform

sonar arrays of eight sonars in each array, bumper switches, a two degree of freedom gripper, a pan/tilt/zoom colour camera, and on-board computer with wireless ethernet and monitor.

Table 2.1 [55] lists some of the technical specifications of the PeopleBot. The robot operates on three rechargeable batteries with a typical run time of eight hours. Like its fellow family members of Pioneer mobile robots [56], the PeopleBot has a relatively small footprint, thus capable of maneuvering in small, tight spaces.

Table 2.1: Performance PeopleBot technical specifications

Parameter	**Specification**
Base robot weight	21 kg
Footprint	51.5 x 42.9 cm
Tire diameter	19.5 cm
Operating payload	8 kg
Turn radius	0 cm
Maximum speed	0.8 m/s
Rotation speed	150°/s
Run time	8 hours (3 batteries)
Battery recharge time	2.4 hours
Battery voltage	12 V
Battery capacity (each)	7.2 Ah
Battery chemistry	SLA

2.2.2 Segway RMP200

In 2001, the Segway Personal Transporter (PT) was introduced to the market as the first self-balancing, zero emissions personal transportation vehicle [57]. The Segway PT operates on the concept of dynamic stabilisation (similar to the classic control problem, the inverted pendulum); it has two wheels which are rotated in the correct speed and torque to prevent the user from falling when leaning forwards or backwards.

A few years after the release of the Segway PT, the Robotic Mobility Platform (RMP) was developed by Segway, based on the design of the PT, for scientific and engineering research purposes. The RMP was designed for an integration into a system that has a control processor to communicate velocity and steering commands via the USB or CAN bus interface. Segway has developed balancing and non-balancing models of the RMP in a variety of 2, 3 or 4 wheel configurations [58].

The RMP200 (shown in figure 2.3 [59]) is a two-wheeled differential drive robot that is capable of maintaining its balance while carrying a heavy payload. The robot can operate in one of two modes: 1) *Tractor mode* causes the RMP to function in a static stable condition with a large footprint and low payload height, whereas 2) *Balance mode* is a dynamic stable operation with a smaller footprint and higher payload.

FIGURE 2.3: Segway RMP200

Table 2.2 [59] lists the technical specifications of the RMP200. The robot operates on two rechargeable batteries with a typical run time of 8 hours. Unlike the PeopleBot, the RMP200 has a larger tire diameter and footprint but it has the ability to operate in tight spaces due to its turn radius specification of 0 cm.

TABLE 2.2: Segway RMP200 technical specifications

Parameter	**Specification**
Weight	64 kg
Footprint	64 x 61 cm
Tire diameter	48 cm
Operating payload	45 kg
Turn radius	0 cm
Maximum speed	16 km/hr
Data update rate	100 Hz
Run time	8 hours (2 batteries)
Battery recharge time	6 hours
Battery voltage	72 V
Battery capacity (total)	380 watt-hours
Battery chemistry	Ni-MH

Mechanically, the RMP200 consists of a base plate which houses the batteries, motors, drives and control box. The payload plate is supported by two side plates. Electronically, the robot has five gyroscope sensors which measure the following:

- Yaw angle and yaw rate
- Pitch angle and pitch rate

- Roll angle and roll rate

In addition to the above itemised quantities, the RMP interface also returns left and right wheel speeds as well as a calculated odometry. The RMP200 does not contain an on-board computer so a laptop is used in this research as the control processor. The purpose of the laptop is to establish a communication interface with the robot to obtain data (yaw, pitch, roll, speed and odometry) and control its movement through speed and yaw commands.

2.2.3 Segway RMP400

The RMP400 is one of Segway's most powerful mobile robots, it is capable of carrying loads up to 181 kg for long distances over rugged terrains. The robot operates on four independent drive motors and five rechargeable batteries. A graphic image of the RMP400 is shown in figure 2.4 [60] and its technical specifications are listed in table 2.3 [60].

FIGURE 2.4: Segway RMP400

The RMP400 was designed for research applied in outdoor environments, therefore it does not function well in indoor, manufacturing environments. For instance, the turning radius of this robot is dependent on load conditions, operating surface type and tire friction, hence tire skid and wheel slippage can be expected over smooth surfaces pertinent to manufacturing or research lab environments. Despite these realities, the objectives of this research are not compromised.

The RMP400, like the RMP200, returns measurement data such as yaw, pitch, roll, speed, and odometry and once again, a laptop is used as the control processor to control the robot's movement through speed and yaw commands.

Table 2.3: Segway RMP400 technical specifications

Parameter	**Specification**
Weight	109 kg
Footprint	76 x 112 cm
Tire diameter	53 cm
Operating payload	181 kg
Maximum speed	29 km/hr
Data update rate	100 Hz
Run time	8 hours (4 batteries)
Battery recharge time	8 hours
Battery voltage	72 V
Battery capacity (total)	1600 watt-hours
Battery chemistry	Li-Ion

2.3 Sensor hardware

This section discusses the sensor hardware used in the research topic for the purpose of robot navigation and obstacle avoidance.

2.3.1 Laser range finder sensors

Section 1.4.2.2 surveyed the various types of range finder sensors that are commonly used in the field of robotics; among them, the LRF sensor was considered in the design due to the accurate range data provided [17]. Figure 2.5 gives an illustration of a LRF used in the robotic middleware platform, Player/Stage.

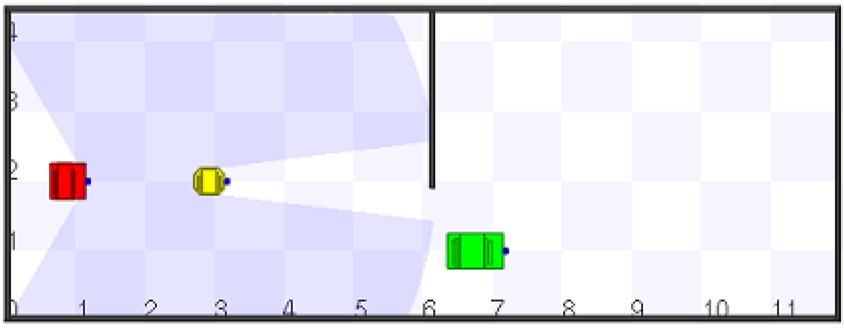

Figure 2.5: LRF illustration in Player/Stage simulation

Two types of LRF sensors were researched before being purchased for the research: the SICK LMS200 (figure 2.6 [61]) and the Hokuyo URG-04LX (figure 2.7 [62]). Various LRF models from each manufacturer were also reviewed but are not mentioned in this dissertation due to their lack of relevance to this research: some were not suited for the research application and others were too expensive to even consider.

FIGURE 2.6: SICK LMS200

FIGURE 2.7: Hokuyo URG-04LX

Table 2.4 draws a comparison of the technical specifications between the SICK LMS200 [61] and Hokuyo URG-04LX [62].

TABLE 2.4: SICK and Hokuyo technical specifications

Parameter	**LMS200**	**URG-04LX**
Range	80 m	4 m
Scanning angle	180°	240°
Angular resolution	0.5°; 1.0°	0.36°
Measurement accuracy	±10/±15 mm	20–1000 mm: ±10 mm 1000–4000 mm: ±1% of measurement
Resolution	10 mm	1 mm
Interface	RS232 or RS422	RS232 or USB
Power source	24 VDC	5 VDC
Current consumption	1.8 A	0.5 A
Environment use	indoor	indoor

The LMS200 has been widely used in robotic applications for localisation and navigation. The sensor can be configured in software to operate in a low range of 8 meters to the full range of 80 meters. The angular resolution for the 8 and 80 meter ranges are 0.5° and 1.0° respectively.

In comparison to the LMS200, the URG-04LX has a single, much smaller, operating range of 4 meters, with angular resolution of 0.36°. The scanning angle, however, is 240°, a 60° more area coverage than the LMS200. A major advantage that the Hokuyo laser has over the SICK laser is the extreme price difference. The URG-04LX is priced at less than half the cost of a LMS200; this is the main reason for choosing the Hokuyo product as the LRF for the RMP200 and the RMP400.

The URG-04LX can not be powered from the USB port of a laptop, therefore the 5V regulated power supply (figure 2.8 [63]) was sourced from Netram Technologies. The power supply board takes in an input of 6–12VDC and supplies a selectable regulated output voltage of 3.3 or 5 VDC. Due to the standalone operation of the LRF, a rechargeable battery was also purchased to supply power to the voltage regulator. The SLA battery is specified at 6V, 3.2Ah. The simple circuit diagram design of the LRF and power supply is shown in figure 2.9.

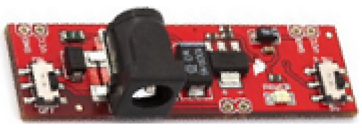

FIGURE 2.8: 5V regulated power supply

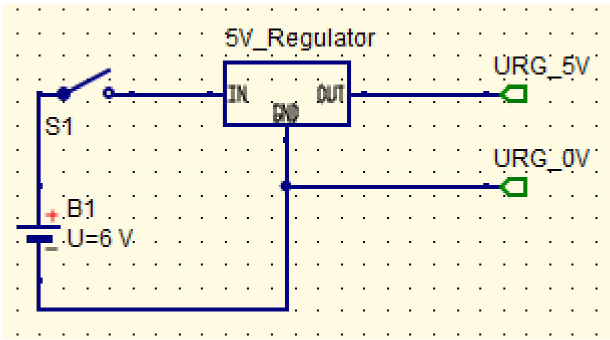

FIGURE 2.9: Power supply design for URG-04LX

2.3.2 Sonar range finder sensors

The SICK LMS200 (supplied as an accessory to the PeopleBot) was a part of the initial design, to be used on the PeopleBot as the range finder; however, during the implementation phase of the research, the LRF did not function as expected and returned spurious data signals which negatively impacted the robot's navigation algorithm. Various attempts were made to find the root cause of the problem and even filter the data but none of these attempts yielded any success. A final decision was made to use the sonar array sensors which are mounted at the top and bottom positions of the robot. The sensors returned accurate, reliable data which contributed to the successful application of the navigation algorithm on the PeopleBot, more of which will be discussed in the succeeding chapters.

2.4 Chapter summary

This chapter began by discussing the control architecture used in this research. A hybrid, weakly cooperative solution was chosen to utilise the benefits of localised control as well as allow for periods of centralised communication for dependent data. The robot hardware and technical specifications of the PeopleBot, Segway RMP200 and RMP400 were then discussed. The chapter concluded with the sensor hardware used in this research, namely the Hokuyo LRF sensors for the RMP robots and the sonar range finders for the PeopleBot.

Chapter 3

Mechatronic System Design

This chapter discusses the general system design and artificial intelligence aspect of the research.

3.1 System design overview

An overview of the system design is shown in figure 3.1. The scope of the design consists

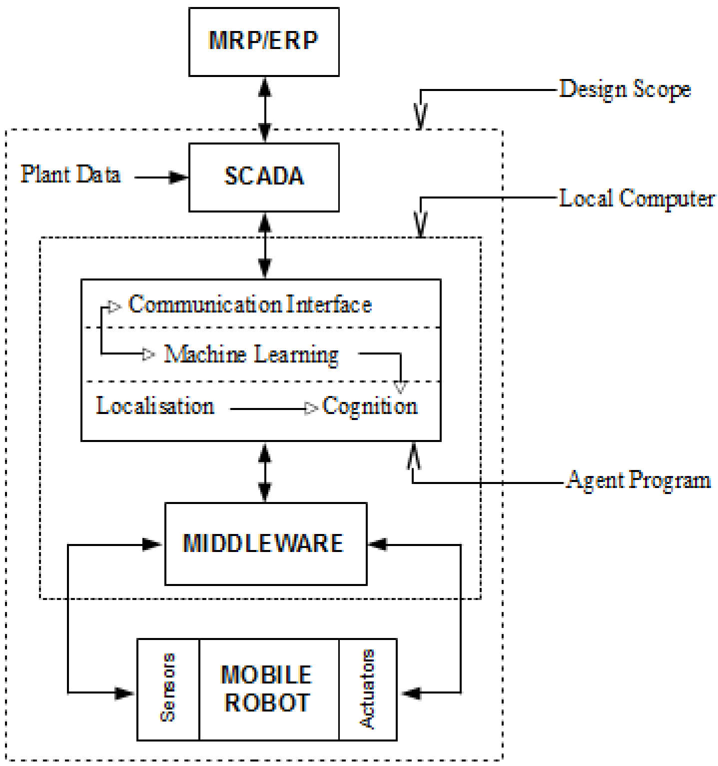

FIGURE 3.1: Design overview of the Mechatronic system

of an integration of the following components:

- Robot hardware
- Middleware
- Agent program
- Supervisory Control and Data Acquisition (SCADA)

The *robot hardware* comprises of the mechanical robot (PeopleBot, RMP200, RMP400), the sensors (LRF, sonars), and the actuators (drives, motors). The *middleware* layer is necessary since it is responsible for interpreting the high level (agent program) commands and presents them to the sensors and actuators through the use of low level software driver modules.

The *agent program* is the robot's decision making component in the system design as it determines which task (primary or secondary) is required by the robot at a specific point in time. In addition to the *localisation* and *cognition* modules (discussed in section 1.4.3 and section 1.4.4), the agent program contains a *machine learning* module which was incorporated in the system design due to the following benefits:

- Robot heterogeneity and task taxonomy: due to the different capabilities of each robot together with the variations in tasks, the system is required to identify whether or not a particular robot can perform a secondary task when required. An integrated learning system will ensure that each robot goes through an engineering teaching process so that the robot "agent" can identify itself as a helping agent when the need (bottleneck) arises.
- Manufacturing environment reconfiguration: changes in the environment, caused by the manufacturing of different products or the implementation of new machinery, will have a minimal impact on the cooperative function of each robot since the learning module ensures that robot agents are re-taught accordingly. A further advantage is the saving of money and resources that would have been required to reconfigure the robots to adapt to the new environment.

The *agent program* also comprises a *communication interface* which sends, receives and processes data packets to/from the plant SCADA system. The SCADA is a vital component in the manufacturing plant automation system since it makes process information available to operators and engineers for the purpose of monitoring and control.

3.2 Artificial intelligence design

In this section, the middleware and agent program components of the research will be discussed further.

3.2.1 The Player Project middleware

In section 1.4.5 the Player Project was discussed as a robotic network server that provides the client (agent program) with an interface to communicate and/or control the sensors and actuators of a robot. Player was chosen as the middleware layer for this research due to: 1) its free open source nature, 2) the availability of a large software library for device driver implementations, 3) the Stage simulation package that can be used to prototype the software development process, and 4) its popularity in the robotics community.

Figure 3.2 is a diagram of the Player architecture, illustrating the communication links between the Player Server and the other components in the robotic network.

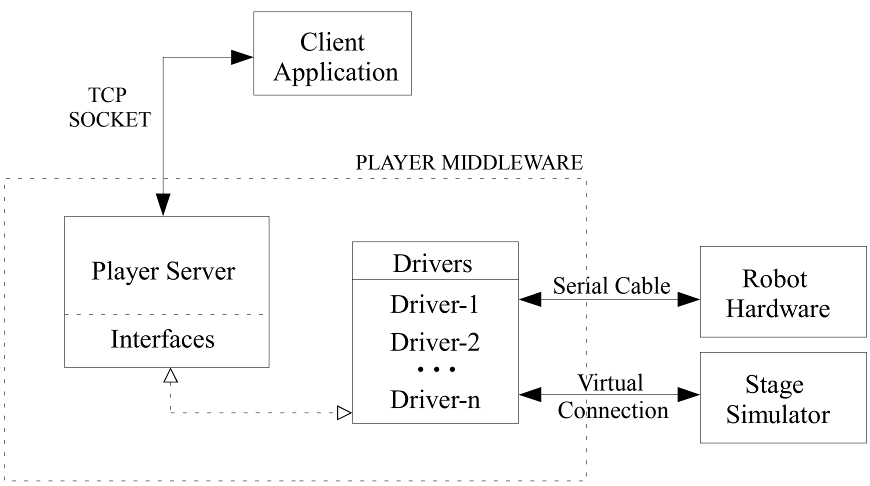

Figure 3.2: Player middleware architecture

3.2.1.1 Client-server framework

The client-server framework, provided by Player, uses the TCP socket protocol for communications between the *client application* and the *Player server*. The server is installed on a computer which is directly connected to the hardware devices of the robot; hence multiple clients (residing on local or remote computers) can connect to the Player server, each on a different socket, and have access to the *robot hardware* on that particular node. In this research topic, the *client application* (or *agent program* in figure 3.1) will reside on the local computer and only have control over the local hardware, prohibiting remote access of sensors and actuators. The use of remote access to sensors can be useful when a robot agent utilises another robot's LRF, for instance, to gain a knowledge of the environment map for the purpose of localisation and navigation. This requires the design and application of advanced localisation and mapping algorithms which is beyond the scope of this research topic.

Client applications can be written in any programming language that provides TCP socket support; to mention a few: C, C++, Python and Java. In this research, the client programs were developed in C++.

3.2.1.2 Interfaces and drivers

The *interfaces* and *drivers* in the Player middleware are key components in the client-server framework since they are used to establish the high-level interface between the *client application* and the sensors or actuators.

A device *interface* is a specification of data for commands and configuration formats, and a device *driver* is a program that controls the device, providing a standard interface to it. An example of a popular Player interface is the `position` interface, used to control a mobile robot base. The `position` interface specifies a command format for velocity and position targets, and specifies a data format for velocity and position status. An implementation of the `position` interface is in Player's `p2os` driver, which is used to control ActivMedia research robots. There are also other drivers which use the `position` interface to control other robots, hence they all accept commands and produce data in the same format. Player thus allows for multiple drivers to support the same interface and single drivers to support multiple interfaces. [43]

3.2.1.3 Stage

The *Stage* simulation package works with the Player server to allow the end-user to simulate the behaviours of robots, sensors and actuators without the need for actual physical hardware. The attributes of the robots and devices, such as size, shape and even colour, can be created in Stage model files and a map of the environment can be added to the "Stage world" as an image. The end-user can therefore re-create the environment and test the behaviour of robots in different scenarios which reduces the total software development time for a particular robotics project.

3.2.1.4 File types

There are three types of files that can be used in the Player-Stage environment:

- the ".world" file: is a programmable description of the environment or "world", where the robots, items and layout of the map are defined. The world file is used when the programmer is working with Stage simulations; it is not required for the core Player server communications with the actual hardware.
- the ".inc" file: is similar to the .world file, the only difference is it can be *included* in the .world file; this allows for modular programming in Stage simulations. Robot models can be used in other worlds and if there are changes required for a particular model, they can be easily modified in the .inc file.
- the ".cfg" file: is required in both simulated (Stage) and real-world environments. The .cfg file is structured to give the Player server all the information about the robot being used: the necessary drivers (if it is a simulated environment, the driver will be Stage) and the format in which the data from the drivers will be presented to the *client application*.

The use of these files will be discussed further in chapter 4. References to the Appendix will also be made in chapter 4 so that the coding and structures of the files can be viewed.

3.2.2 Localisation

The localisation driver included in the Player Project driver library is the AMCL algorithm (discussed in section 1.4.3). Fundamentally, the driver determines the robot's pose by using a predefined map of the environment and a probability distribution over a set of all possible poses. The distribution set is updated from odometry, sonar and/or LRF data.

In the development phase of this research the AMCL driver was tested in the Stage environment and yielded poor results due to the driver's inability to estimate the correct pose when the robot becomes lost (possibly attributed to accumulated errors). The developers of the Player AMCL driver make mention of this estimation problem and discuss the use of more advanced techniques which have not yet been implemented [64].

A design decision was made to calculate the localised pose of the robot by using a x-y coordinate system map of the environment together with the odometry data returned from the internal sensors of the robot. The sole use of odometry in localisation calculations has its own problems in terms of error integration due to drift and slippage (see section 1.4.2.1 and figure 1.4), however, this approach to robot localisation does not compromise the objectives of this research topic. Future work to this research (discussed more in chapter 6) will see the use of SLAM algorithms or landmark methods in robot localisation.

3.2.3 Cognition

The obstacle avoidance and local navigation drivers provided by Player include: 1) the VFH algorithm, 2) Nearness Diagram (ND), and 3) an improvement to the ND, the Smooth Nearness Diagram (SND). The drivers were tested during the development phase of this research in both simulated and real-world cases and they each revealed satisfactory results for the PeopleBot and to some extent the Segway RMP200; however, when implemented on the RMP400 the navigation performance was poor since the robot did not respond well to turns in tight spaces (due to its geometric shape and large size). For this purpose, a *cognition* module was designed and developed as part of this research topic. The design choice was also based on the idea of developing the module/driver further to include an advanced localisation algorithm as part of the future work to this research topic.

The cognition module is a component of the *agent program* and contains obstacle avoidance and local navigation routines. The state definitions in table 3.1 and the state diagram in figure 3.3 depict the design methodology used in the development of the cognition module, which is further discussed in chapter 4.

TABLE 3.1: State definitions for the cognition module

State no.	State description
0	Avoid near obstacles (estop, turn, reverse)
1	Position robot towards goal
2	Positioning done. Clear to proceed?
3	Plan path to goal using way-points
4	Clear path to goal– move to goal
5	Goal reached

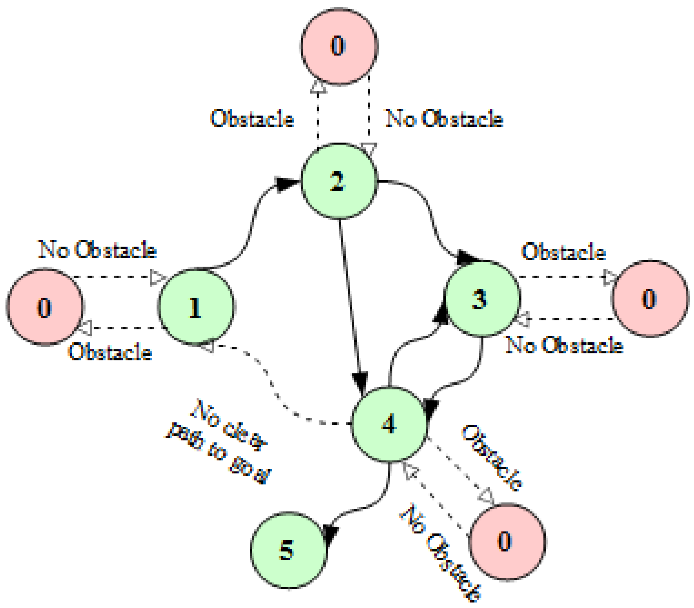

FIGURE 3.3: State machine diagram for the cognition module

3.2.4 Machine learning

The benefits of including a Machine Learning (ML) module to the system were discussed in section 3.1. The ML system forms the core of the intelligence involved in determining the immediate goals for each robot agent. Literature on learning models for robotic systems are vast and cover topics such as Behaviour-Based Systems (BBS) [50], Reinforcement Learning (RL), artificial neural networks [65], and genetic algorithms [66], amongst many others. In this research, the Support Vector Machine (SVM) algorithm is used as the learning platform for the robot agent intelligence. SVM learning is a supervised, classification or inductive learning scheme where the computing system learns from the database of past experiences to predict future outcomes; it was selected as the learning engine in this research due to the following reasons:

- The inductive learning characteristics of SVMs allow for the teaching of robot agents in an offline, simulated environment before the robots are called to action in the real-world.
- SVMs have proven to be successful in many applications such as bioinformatics, and text and image recognition.
- SVM software libraries are available as open source packages which assist developers to focus on their primary research.

3.2.4.1 SVM background

SVM learning is related to statistical theory [67] and was first introduced as a classification method in 1992 [68]. It is widely used in bioinformatics due to its accuracy and ability to work with high-dimensional space data. The standard SVM is a binary linear classifier (commonly referred to as the linear SVM) which predicts whether an input belongs to one of two possible classes; this is accomplished by first building a model from a set of training examples, each consisting of input data that are mapped to the corresponding class label. SVM non-linear classifiers can be created by using non-linear kernel functions, further discussed in section 3.2.4.3.

3.2.4.2 SVM linear classifiers

In order to gain an intuition on what support vectors actually are and how they are used to create learning models a few preliminary mathematical terms will now be introduced. Given some training data set, D, with n points:

$$D = \{(\mathbf{x}_i, \mathbf{y}_i), \mathbf{x}_i \in R_m, \mathbf{y}_i \in \{-1, 1\}\}_{i=1}^{n} \tag{3.1}$$

The boldface $\mathbf{x}$ term is a vector with training example inputs $\mathbf{x}_i$; each $\mathbf{x}_i$ has an m–dimensional size of m features. The classifier term, $\mathbf{y}_i$, is either -1 or 1 and indicates the class to which each point $\mathbf{x}_i$ belongs.

In figure 3.4 (a), the training examples are classified into positive and negative classes. The hyperplane, H, is the *decision boundary* that divides the regions between positive and negative classes. The decision boundary is said to be *linear* since the examples are linearly separable and a classifier with a linear decision boundary is called a *linear classifier*. *H1* and *H2* are lines that intersect the *support vectors*, these are the training examples that are closest to the decision boundary and they determine the margin (*d1* and *d2*) at which the two classes are separated from the hyperplane (or decision boundary). The SVM algorithm is also termed as the *large margin classifier* since its goal is to maximise the margin d for a set of classified training examples.

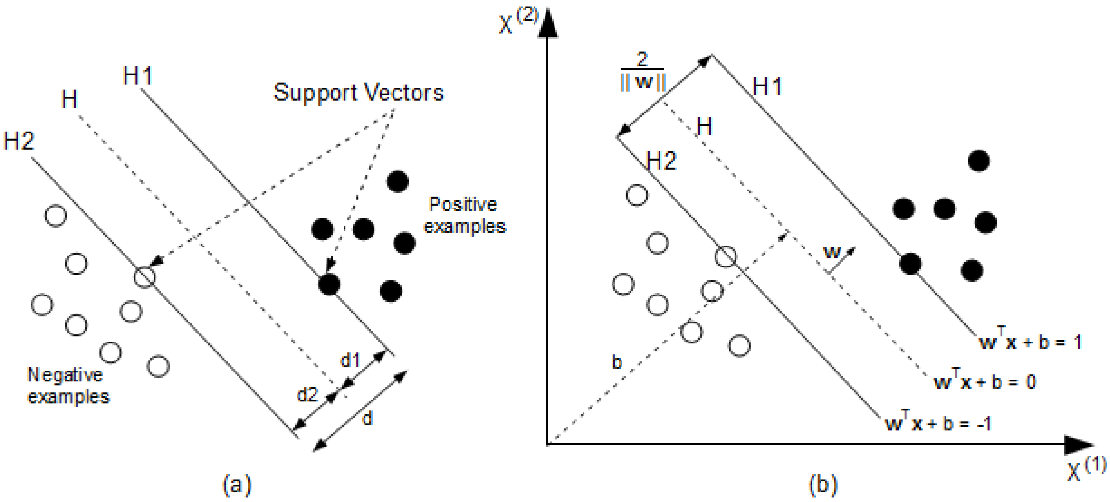

FIGURE 3.4: (a) Hyperplanes and margins. (b) Margin classifiers

Figure 3.4 (b) is an extension to (a) and shows the training examples on a two dimensional feature space with features $x^{(1)}$ and $x^{(2)}$. A linear classifier is based on a linear function of the form:

$$f(x) = \mathbf{w}^T\mathbf{x} + b \tag{3.2}$$

where $\mathbf{w}$ is commonly known as the weight vector and b is the bias. The product between $\mathbf{w}$ and $\mathbf{x}$ is known in linear algebra as the dot product and is defined as $\mathbf{w}^T\mathbf{x} = \sum_i w_i x_i$. The equation for the hyperplane is:

$$H : \mathbf{w}^T\mathbf{x} + b = 0 \tag{3.3}$$

where the purpose of the bias can be seen as moving the plane away from the origin, i.e. if b=*0* the hyperplane would go through the origin. Equations 3.4 and 3.5 are related to planes *H1* and *H2*:

$$H1 : \mathbf{w}^T\mathbf{x} + b = 1 \tag{3.4}$$

$$H2 : \mathbf{w}^T\mathbf{x} + b = -1 \tag{3.5}$$

and are equated to 1 and -1 respectively due to the definition of the classifier term, $\mathbf{y}_i$ in equation 3.1. Using geometry and referring to figure 3.4 (b), the margin between H and *H1* is $1/\|\mathbf{w}\|$, where $\|\mathbf{w}\|$ is the length of the vector $\mathbf{w}$ and is given by $\sqrt{\mathbf{w}^T\mathbf{w}}$; hence the margin between *H1* and *H2* is $2/\|\mathbf{w}\|$. In order to maximise the margin, $\|\mathbf{w}\|$ must be minimised subject to the following constraints which are added to prevent data points falling into the margin:

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1, \quad \{for\ \mathbf{y}_i = 1\} \tag{3.6}$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \{for\ \mathbf{y}_i = -1\} \tag{3.7}$$

Equations 3.6 and 3.7 can be combined to form:

$$\mathbf{y}_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \{for\ 1 \leq i \leq n\} \tag{3.8}$$

Minimising $\|\mathbf{w}\|$ subject to equation 3.8 is a *constrained optimisation problem* and solving it requires using the method of *Lagrange multipliers*. A method that can be used to obtain a dual formulation, expressed in terms of α_i variables [69]:

$$\text{maximise } \alpha\text{:} \quad \sum_{i=1}^{n}\alpha_i - \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T\mathbf{x}_j \tag{3.9}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} y_i\alpha_i = 0, \quad \alpha_i \geq 0 \tag{3.10}$$

The dual formulation also defines the weight vector in terms of the training examples:

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i \tag{3.11}$$

3.2.4.3 SVM non-linear classifiers

In most SVM classification problems the data set is not linearly separable. Literature [68] solves this challenge by mapping the original finite dimensional space into a higher dimensional space making the separation much easier in that space, as illustrated in figure 3.5.

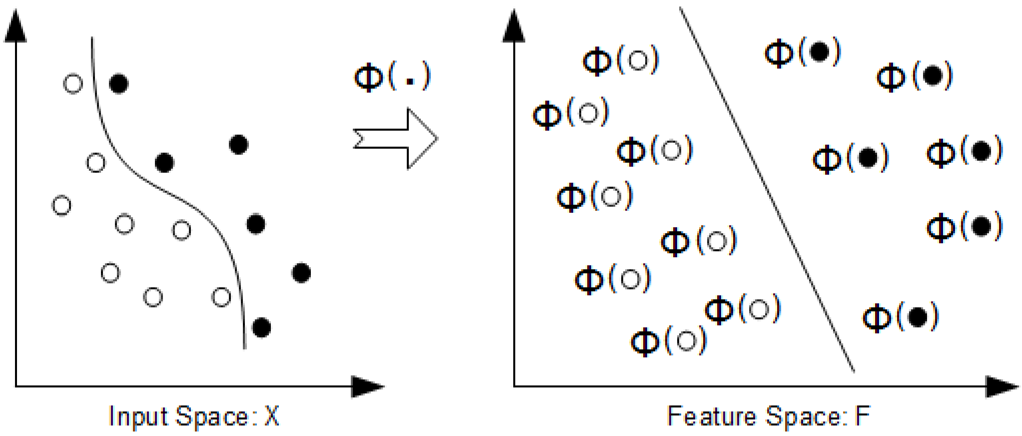

FIGURE 3.5: Non-linear classification mapping

The mapping is achieved by the use of *Kernel functions* and the dot product property in the linear SVM algorithm. The $\mathbf{x}_i^T\mathbf{x}_j$ terms in equation 3.9 are replaced by the kernel function, K:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j) \tag{3.12}$$

which can represent (among others) a *polynomial*, *gaussian*, or *hyperbolic function* [70]. The linear classifier is also known as the *linear kernel*.

3.2.4.4 Multi-class SVM

SVMs are inherently binary classifiers however, there are many applications where multiple classifications are required. The common method of solving the *M-class* problem is to divide it into multiple binary classification problems [71]:

- One-vs-All: This method constructs N binary SVM classifiers, where N represents the number of classes. Every i-th SVM is trained to differentiate the training examples of the i-th class from the examples of the other classes. At the classification phase, samples are classified in accordance to the highest output function among all the SVMs.

- One-vs-One: This strategy constructs one SVM for every pair of classes, hence for an *M-class* problem of N classes, $N(N\text{-}1)/2$ SVMs are trained. A maximum-wins voting concept is used where each SVM classifier assigns the sample to one of the two classes and the number of votes for the assigned class increases by one; in the end, the class with the most votes determines the classification of the sample.

Another approach to the *M-class* problem, which avoids the use of multiple binary classification problems, involves the application of a single optimisation model [72].

3.2.4.5 SVM software libraries

Over the past two decades there has been a wide interest in SVM algorithms which has led to the development of many solvers for SVM optimisation problems. Two popular open source solvers are LIBSVM [73] and SVM^{light} [74]. These solvers form excellent tools for researchers since they eliminate the vast quantity of time that could be spent on the complex software development of SVM optimisation algorithms and thus allow the scientist to focus on the primary components of the research. The LIBSVM library is used in this research.

3.3 SCADA design

SCADA systems are widely used in the manufacturing industry; they can vary in functionality and cost and are chosen to suite a company's application and design requirements. The criteria used to select a SCADA package for this research topic are:

1. The software package must be open source.
2. Support TCP socket programming (for communicating with the agent interface).
3. Allow for the reliable connectivity of multiple nodes on a network (for the communication of every robot agent).
4. Ability to develop an internal program for the purpose of simulating plant data.
5. Ability to develop a Graphic User Interface (GUI).
6. Trending functionality of plant data.

The Proview system [75] was chosen as the SCADA platform for this research since it satisfied the above mentioned criteria by having the following features:

1. Proview is an open source system for process control and can run on a standard PC with a Linux operating system.
2. Communication with other computers can be done via serial or ethernet means and the TCP socket protocol is supported.
3. The distributed system architecture of Proview allows it to be connected to multiple nodes in a network.
4. Simulation programs can be written in C, C++ or Java; the SCADA developer can switch between simulated and real environments allowing for an efficient process in the commissioning of a plant.
5. The Ge Editor is the environment in Proview where GUI's can be developed.
6. Proview supports trending and historical capturing of data.

In addition to the above features, Proview comes with the following benefits (some of which are not required for this research topic):

- Proview is based on a soft Programmable Logic Controller (PLC) solution, thus there is no need for hard-wired inputs.
- There is no limit to the number of I/O, PLC programs, timers, counters, etc.
- The minimum PLC loop cycle time is less than 1 ms and performance is limited by the hardware and operating system of the hosting PC.
- Proview is object orientated, thus complex blocks with methods and attributes can be created.

Section 4.4 covers the development of the SCADA system for this research, using Proview. Modules such as the PLC, GUI and simulation program will be discussed.

3.4 Operating system kernel

The main operating system used in this research is Linux, due to the compatibility of both Player and Proview in this kernel. The Linux distribution used on the computing processors for the Segway robots is Ubuntu 12.04 and the distribution used on the PeopleBot is ArchLinux.

3.5 System design overview (revisited)

A revised overview of the system design is given in figure 3.6 to include the components that were discussed throughout this chapter.

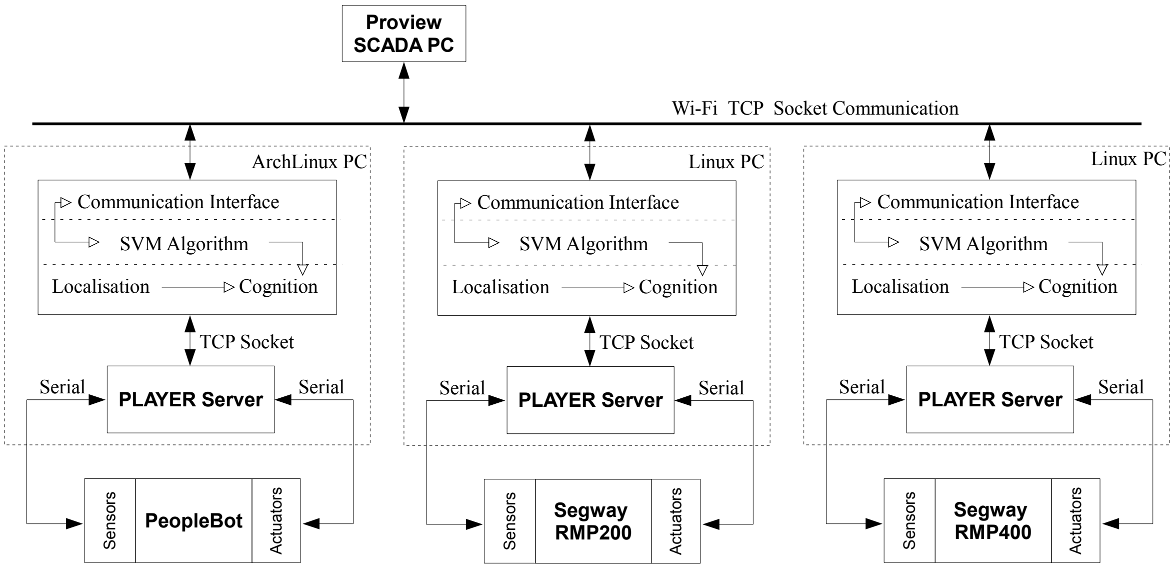

FIGURE 3.6: Revised design overview of the Mechatronic system

The form of communication between the SCADA and three Linux computers is wireless ethernet (Wi-Fi), this prevents the use of network cables connected to mobile robots.

3.6 Chapter summary

This chapter discussed the design of the software components involved in the research, namely the Player middleware, the agent program (consisting of localisation, cognition and SVM learning modules), and the Proview SCADA system. The chapter concluded by revisiting the system design overview to include the components discussed throughout the chapter.

Chapter 4

Software Development

The artificial intelligence software is the core element of this research topic since the research objectives are achieved here, through the development and system integration of multiple software components. The "parent" components are the *Player server* and *client applications* which are further subdivided into "child" components such as: drivers, interfaces, cognition and SVM modules, and communication protocols. The purpose of this chapter is to discuss the development and integration of these components in a structured manner so that the reader has a clear understanding of the AI process taken to achieve the objectives of this research topic. This chapter also discusses the development of the SCADA component of this research.

4.1 AI development overview

There are two development routes that were taken in this research: 1) the *simulation* process involves the use of Player's Stage platform, where robot models were created to interact in the two dimensional simulation environment, and 2) the *real-world* portion is the software implementation on the actual robots, in a robotic lab environment.

Figure 4.1 and figure 4.2 depict overviews of the software development for the simulation and real-world routes respectively.

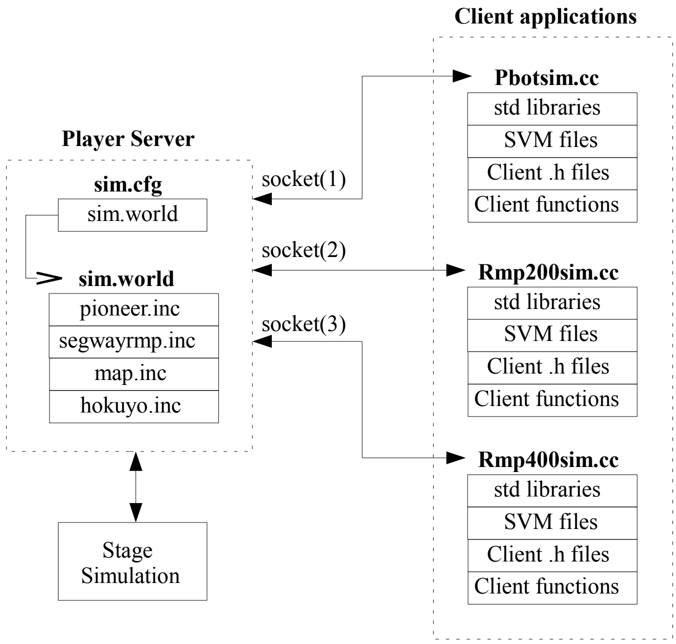

FIGURE 4.1: AI simulation development overview

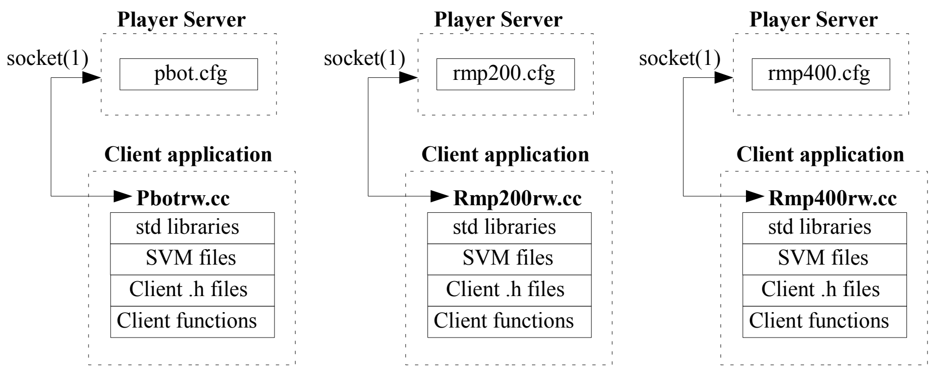

FIGURE 4.2: AI real-world development overview

The simulation development (with reference to figure 4.1) has the Player/Stage files and client application files that make up the system:

- *sim.cfg* is the Player configuration file which defines the drivers and interfaces (for each robot) used in the system. The file defines Stage as the simulation driver and includes the *sim.world* file. The code for *sim.cfg* is shown in Appendix A.1.

- *sim.world* defines the layout of the environment. The file includes an image of the environment map and other ".inc" files for the definition of robot and sensor hardware. The code for *sim.world* is shown in Appendix A.2.

- *Pbotsim.cc*, *Rmp200sim.cc* and *Rmp400sim.cc* are three client application files containing the cognition and SVM intelligence modules. The files also contain TCP socket and protocol coding for communication between the SCADA interface; *Pbotsim.cc* is shown in Appendix A.4. More detail on the client application code is given in the succeeding sections.

In the simulation development, all three client application files reside on one computer and use different socket connections to the Player server, although it is possible to run each client application on a separate computer. A single instance of Stage is used and all three robots can be seen in one Stage window during the simulation (see figure 2.5).

As shown in figure 4.2, the real-world development system comprises of *.cfg* files for each robot (the code is documented in Appendices A.5, A.6 and A.7) and do not contain ".world" or ".inc" files due to the lack of use of Stage. Each Player *.cfg* file and client *.cc* file executes on a separate local computer attached to the robot.

4.2 Player server configuration

As mentioned previously, the purpose of the Player configuration file is to define the drivers and interfaces used for each robot, in either simulated or real-world environments. This section discusses the various drivers and interfaces implemented in the research.

4.2.1 Simulation drivers and interfaces

The interfaces that were used in the simulation are `position2d`, `sonar`, and `laser`, as shown in table 4.1. The index (":x", where x is 0,1 or 2 in table 4.1) indicates which interface is going to be used in the driver, since there may be more than one device in the system.

TABLE 4.1: Player simulation drivers and interfaces

Robot	**model**	**Driver**	**Interface-1**	**Interface-2**
PeopleBot	r0	stage	position2d:0	sonar:0
RMP200	r1	stage	position2d:1	laser:1
RMP400	r2	stage	position2d:2	laser:2

The `position2d` interface is used by the client to retrieve the robot's two dimensional position data (such as x, y, and yaw odometry) and provides the client with control over actuators to set the linear and angular velocities of the robot. The `sonar` and `laser` interfaces are used to get range data from the sonars and lasers respectively.

4.2.2 Real-world drivers and interfaces

The driver and interface structure for the real-world development in Player is shown in figure 4.3, where the black boldface font give the names of the Player drivers used (e.g. `p2os`) and the red boldface font give the names of the final interfaces used (e.g. `position2d:0`).

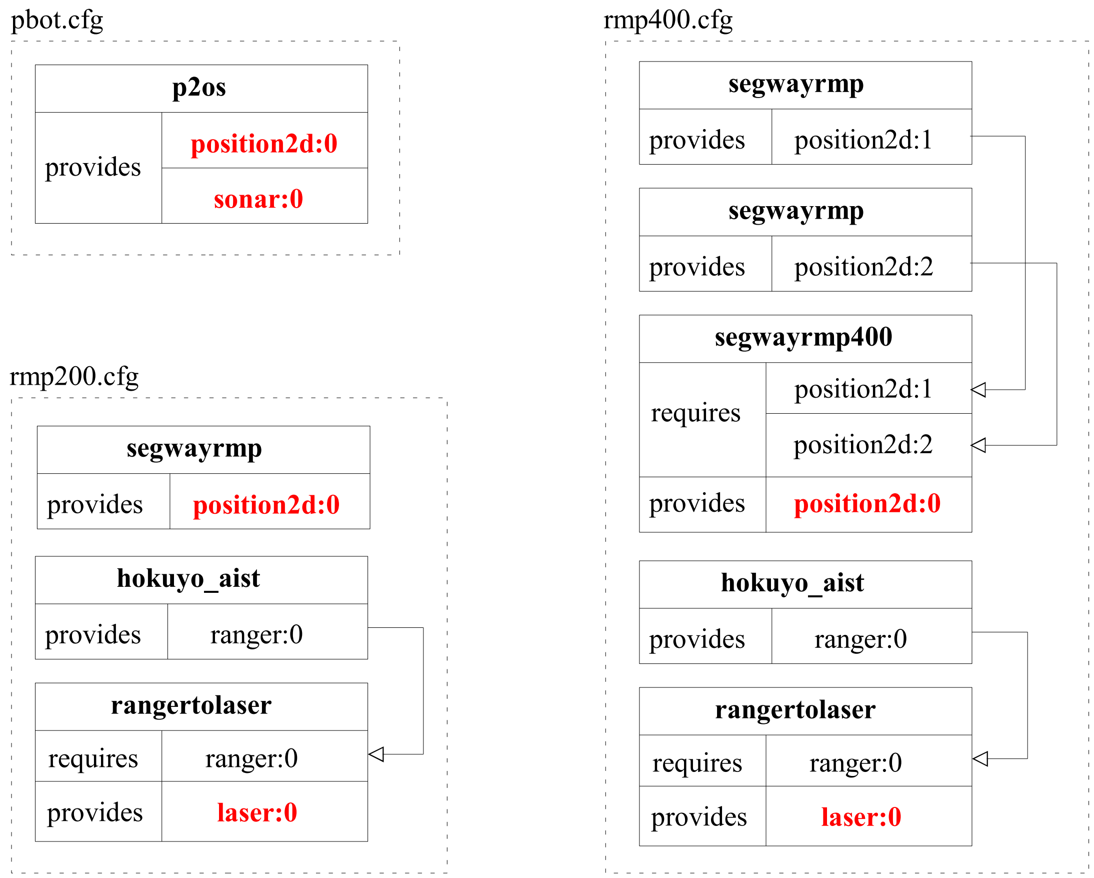

FIGURE 4.3: Player real-world drivers and interfaces

The `segwayrmp400` driver requires the position interfaces provided by two `segwayrmp` drivers, this is simply for the forward and rear sets of wheels on the robot. The driver for the Hokuyo laser, `hokuyo_aist`, provides a ranger interface which is converted to a laser interface by using the `rangertolaser` driver.

4.3 Client applications

The client application code for the simulation and real-world developments are almost the same, with just two sets of differences. The first difference is in the *config.h* files. This file contains the program constants which are used as factors for speed, obstacle avoidance, and navigation; the *config.h* file for the PeopleBot simulation is found in Appendix A.3. The reader is referred to the accompanying CD for a complete reference to the all the code files used in this research.

The second difference between simulation and real-world client code can be seen by comparing the code shown in figure 4.4, where a `SimulationProxy` is used for the simulation code only, a requirement for the Stage driver. Proxies are used in Player to subscribe the client to the interfaces of the drivers.

```
using namespace PlayerCc;
PlayerClient     robot("localhost");
SimulationProxy simproxy(&robot);
Position2dProxy pp0(&robot,0);
SonarProxy sp(&robot,0);
```

(a)

```
using namespace PlayerCc;
PlayerClient     robot("localhost");
Position2dProxy pp0(&robot,0);
SonarProxy sp(&robot,0);
```

(b)

FIGURE 4.4: (a) Simulation proxy code extract. (b) Real-world proxy code extract

4.3.1 Player proxies and methods

The `Position2dProxy` allows the client application programmer to use methods such as `GetXPos`, `GetYPos`, `SetSpeed`, among many others. In this research, the `Position2dProxy` methods used in the client application are:

- `GetXPos()` gets the robot's current x-coordinate in relation to its x-start position.
- `GetYPos()` gets the robot's current y-coordinate in relation to its y-start position.
- `GetYaw()` gets the robot's current yaw in relation to its yaw-start position.
- `SetMotorEnable(bool)` takes a boolean input to enable or disable the robot's motors.
- `SetSpeed(XSpeed,YawSpeed)` is the command which controls the linear and angular speed of the robot.

The `SonarProxy` and `LaserProxy` methods used are:

- `IsValid()` returns a boolean state declaring whether the sonar/laser scan was successful or not.
- `GetCount()` returns the number of points in the sonar/laser scan.
- `ProxyName[index]` returns the range data of the index for the sonar/laser scan.

4.3.2 Cognition module

Section 1.4.4 discussed the design overview of the cognition module; the state machine diagram (figure 3.3) illustrated the steps involved in the cognition process. In this section, a more detailed review of the cognition process will be discussed:

- Cognition program structure: flow chart illustration of the logic used in the programming of the cognition module.
- Turning in tight spaces: analysis of the solution to this problem.
- Local navigation: developmental discussion of the way in which each robot navigates to its immediate goal.

4.3.2.1 Cognition program structure

A flow diagram for the cognition program (in Appendix A.4) is given in figure 4.5. The sequence begins when the *GoalCmd* boolean variable is `true`. The variable is set when the SVM learning module evaluates a new goal location for the robot and the variable is cleared at the end of the cognition sequence (i.e. when the goal has been reached). The two procedures in the diagram responsible for the obstacle avoidance and local navigation components are *ObsAvoid* and *Calc waypoint* respectively.

The *ObsAvoid* routine ensures that 1) the robot goes into an emergency stop (estop) mode when another robot, obstacle or human is within a safety distance, 2) the robot reverses when safe to do so, 3) the robot turns in tight spaces, and 4) the robot waits until it is clear to proceed towards the goal.

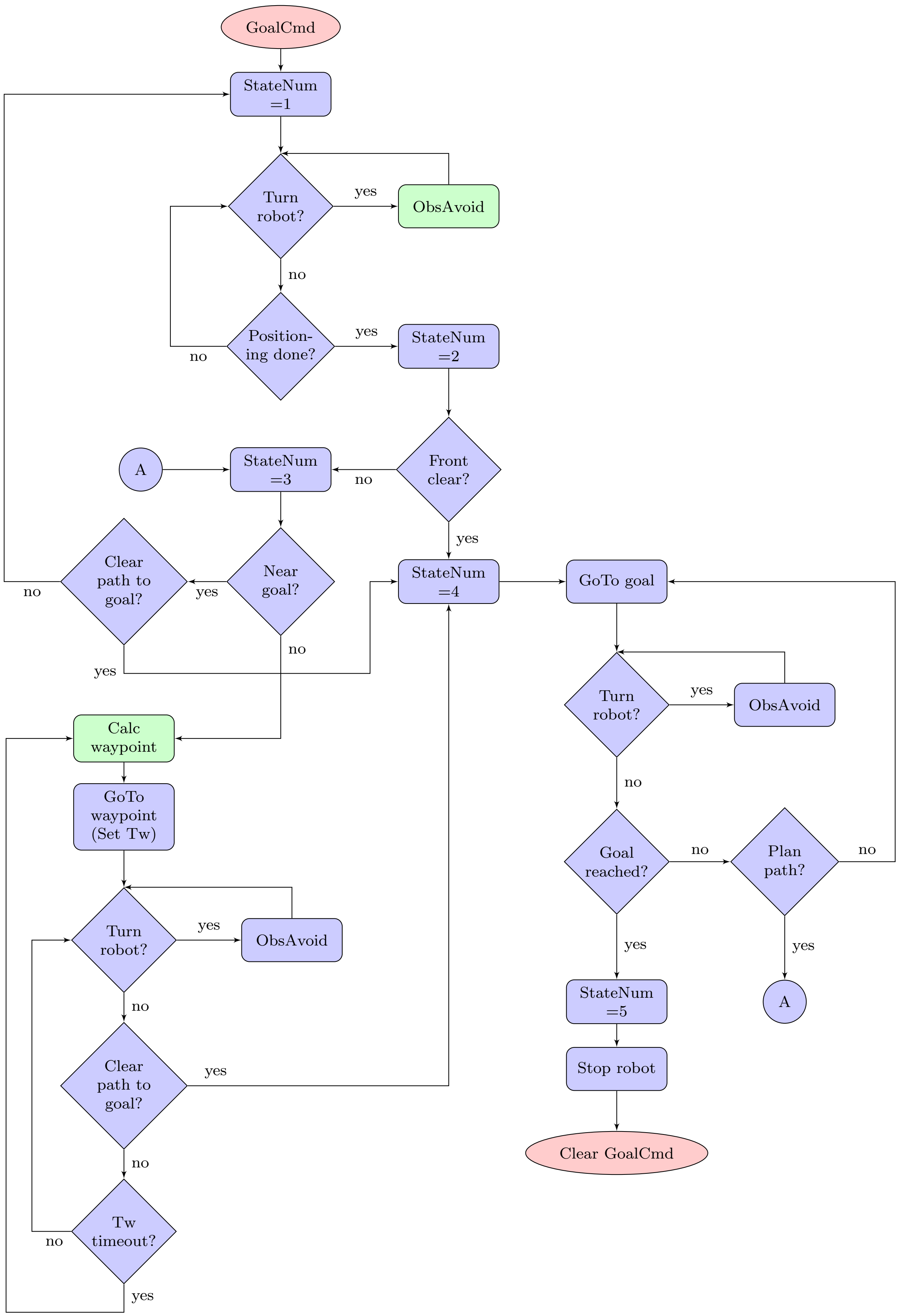

Figure 4.5: Cognition flow diagram

The *Calc waypoint* routine calculates the x-y coordinates of the "waypoint" to the goal and occurs whenever the robot is far away from the goal coordinates. The robot thus travels on a series of way-points en route to the goal.

4.3.2.2 Turning in tight spaces

The objective of the "turning" function in the cognition program is to control the angular speed (ω) of the robot so that the robot does not clash into objects in the confined space environment. In figure 4.6 a robot, R, with length R_L is shown x_{min} meters away from its nearest vertical obstacle and y_{min} meters away from its closest horizontal obstacle. The calculated ω must allow for a smooth trajectory of the robot without its 1) front colliding with the x_{min} obstacle, and 2) rear colliding with the y_{min} obstacle. The x_{min} and y_{min} distances are obtained from the sonar or laser range finders and the point of origin in the measurements is the front-center of the robot (labeled as c in figure 4.6).

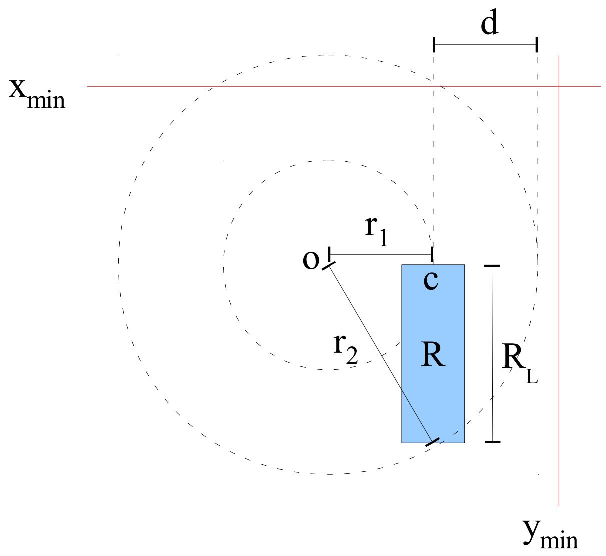

FIGURE 4.6: Robot parameters in confined space

The relationship between linear speed, ν and ω is:

$$\nu = r\omega \tag{4.1}$$

where r represents the radius r_1 in figure 4.6. Setting ν as a constant during robot turns will imply that ω can be calculated if r_1 is known. The outer radius, r_2, can be easily calculated using the theorem of Pythagoras:

$$r_2 = \sqrt{r_1^2 + R_L^2} \tag{4.2}$$

The distance, d is simply given by the difference between r_1 and r_2; this distance is used as a condition in the turning algorithm to ensure that the turn radius does not cause a collision with the horizontal obstacle:

$$d < y_{min} \tag{4.3}$$

The radius r_1 is determined through an iterative loop as shown by the code extract in figure 4.7.

```
bool wexit=false;
r1=xmin;
while ((r1>0)&&(wexit==false))
{
  r1-=0.05;
  r2=sqrt(sqr(r1)+sqr(RL));
  if ((r2-r1)<ymin)
  {
    w=(0.5*vmax)/r1;
    if (w>wmax) w=wmax;
    wexit=true;
  }
}
if (r1<=0) w=wmin;
```

FIGURE 4.7: Code extract for angular speed calculation

In the code, r_1 is initialised to x_{min} and reduced by 0.05 at every iteration, until the condition in equation 4.3 is satisfied. The r_2 radius is calculated from the iterated r_1 radius, hence d is determined by the difference between the radii. The resultant r_1 and the constant ν is then applied in equation 4.1 to determine ω.

4.3.2.3 Local navigation

The development for robot navigation is fundamentally achieved by choosing the correct sonar/laser sector in the direction of the goal. The number of sectors chosen during the implementation was 8, equidistantly spanned across 180° for the PeopleBot and 240° for the Segway RMP robots, as depicted in figure 4.8. The 180° span is due to the positioning of the 16 sonar sensors: 8 located at the top of the robot and 8 at the base. The 240° span is the maximum range of the Hokuyo LRF, since the laser is positioned at the front-center of the robot, the span is 180° with an additional 30° on either side of the robot.

FIGURE 4.8: Sonar/Laser segments for PeopleBot and Segway robots

The result of the local navigation algorithm is the way-point coordinate which is used as an intermediate goal for the robot, en route to the main goal defined by the SVM module. The idea of the navigation is to calculate the way-points such that the robot travels through gaps (free space) in the environment, avoiding obstacles; the gaps are determined by using the data returned from the sonar and laser scans. The process of calculating the way-point is given by the following steps:

1. Calculate the individual sector areas.

The area (A) of any sector, with radius R and angle θ, is given by [76]:

$$A = \frac{\theta}{2} R^2 \tag{4.4}$$

Using equation 4.4, a summation of the infinitesimal areas produced by each laser reading in the sector can determined by the following equation:

$$S_k = \sum_{i=1}^{n} \frac{\theta_p}{2} L_i^2 \tag{4.5}$$

where S_k represents the total area for the k-th sector ($k \in \{1, 8\}$), θ_p is the constant pitch angle between each laser reading (22.5° for the sonars and 0.36° for the Hokuyo LRF), L_i is the individual laser value in the sector of n laser readings.

2. Select the sectors by comparing the sector areas with a calculated threshold area.

The threshold area is calculated by use of equation 4.4, with a minor adjustment:

$$A_{thresh} = P_{fac} * \frac{\theta_{sector}}{2} * R_{dist}^2 \tag{4.6}$$

where P_{fac} is the probability constant, θ_{sector} is the constant angle of each sector (calculated by the sonar/laser span divided by 8— the number of sectors), R_{dist} is another constant which represents the maximum distance allowed from the obstacle.

3. Group the adjacent selected sectors and calculate the arc length spanned by these sectors.
The arc length, S, of any sector, with radius R and angle θ, is given by [76]:

$$S = R\theta \tag{4.7}$$

The total arc length for adjacent sectors is calculated by applying equation 4.7 as a summation over each sector in the group. The parameters used for R and θ in equation 4.7 are R_{dist} and θ_{sector} respectively, as applied in equation 4.6.

4. If the total arc length calculated in step 3 is above a constant threshold, calculate the start angle (θ_a) and end angle (θ_b) of the arc relative to the position of the robot.
The constant threshold is defined as 1.5*R_w, where R_w is the width of the robot; this definition ensures that the gap through which the robot travels is wide enough.

5. Calculate the best angle, θ_{best}, (from each selected arc) that give the smallest (ideally zero) angle difference relative to the goal.

6. Loop through all of the best angles and choose one with the least angle difference, in relation to the goal.

7. Calculate the x-y way-point coordinate from the best angle chosen in step 6, by applying the following two parametric equations of a circle:

$$x_w = x_c + (R_{dist} * \cos\theta_{best}) \tag{4.8}$$

$$y_w = y_c + (R_{dist} * \sin\theta_{best}) \tag{4.9}$$

where x_c and y_c are the coordinates of the current location of the robot, and x_w, y_w are the way-point coordinates.

4.3.3 Client-SCADA communication

The communication between the SCADA system and the client application is done through the use of TCP socket connections. The data packets in the communication are built and decoded by both the SCADA and client interfaces. An overview of the communication system is shown in figure 4.9.

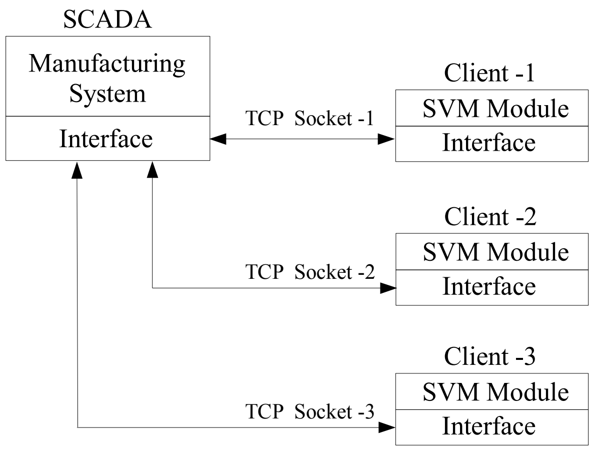

FIGURE 4.9: Client-SCADA communication overview

The following three sections discuss the development of the socket connection, the manufacturing application used for this research topic, and the format of the data packets.

4.3.3.1 Socket connection

A description of the socket functions used in the SCADA and client interface code is listed in table 4.2 and the sequence of the connection is illustrated in figure 4.10.

TABLE 4.2: TCP Socket functions

Function	Description
socket()	Create a new socket
bind()	Attach a socket with a port and address
listen()	Establish a queue for connection requests
accept	Accept a connection request
connect()	Attempt to establish a connection to a remote host
recv()	Receive data over the connection
send()	Send data over the connection
close()	Close the connection

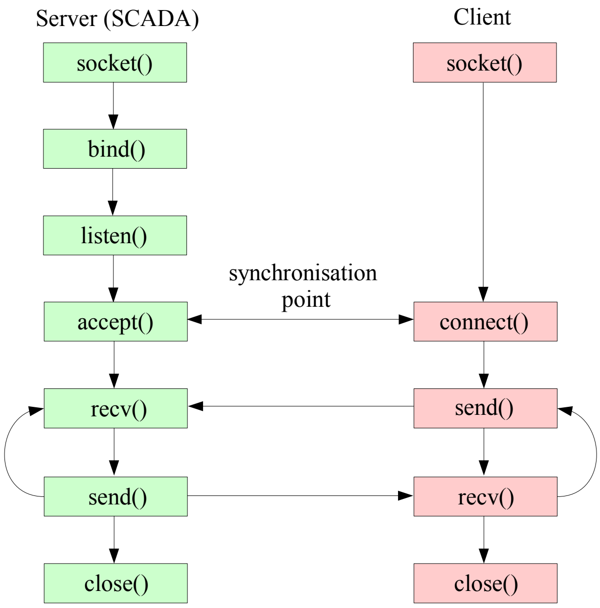

Figure 4.10: Client-SCADA TCP socket connection

The TCP server in the socket connection is the SCADA node and the TCP client is the client application. The server passively waits for and responds to clients, therefore the server socket is a passive one. In contrast, the TCP client socket is active, since the client initiates the communication and is required to know the address and port of the server. The TCP socket protocol supports multiple client connections to a single server, hence in this research, all three client applications connect simultaneously to the SCADA server.

4.3.3.2 Manufacturing system (application)

The objectives of this research topic were tested in a material handling application, as illustrated by the SCADA screenshot shown in figure 4.11. The application shows a resource buffer ("R"), a storage buffer ("S"), 6 process buffers ("B1"–"B6"), 3 machines ("M1"–"M3"), and a conveyor; it was designed in this manner to demonstrate the co-operative ability of the system during bottleneck and fault conditions. The application was set up for the PeopleBot to transport material from "R" to "B1", the RMP200 move material from "B4" to "B5", and the RMP400 to finally move the end product from "B6" to "S".

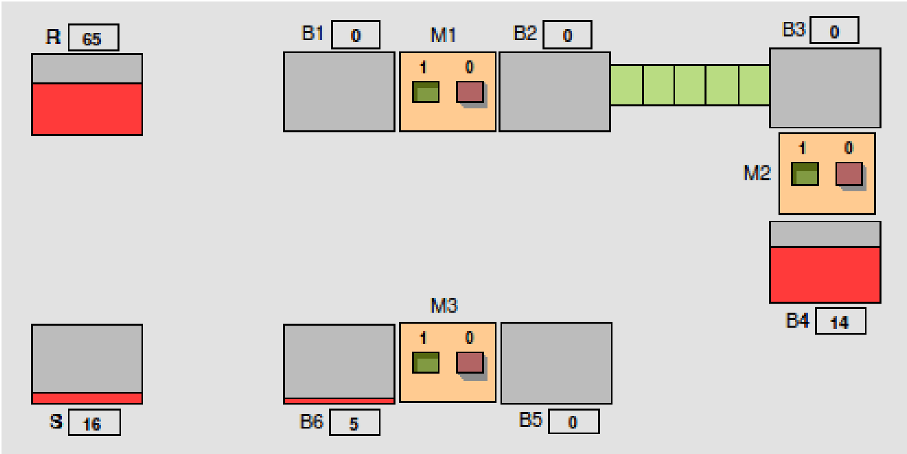

FIGURE 4.11: Material handling application for this research topic

The numbers within the blocks shown in figure 4.11 represent the quantity of material in the buffer and the buffer levels are illustrated as a percentage of their total capacity, thus the bottlenecks in the process can be seen at a glance during production. The calculations for the quantity of material, buffer capacities, and machine process rates are all done in the simulation program which is located in the SCADA *manufacturing system* component (see figure 4.9); more detail on the simulation program is discussed in section 4.4.4.

During the implementation and debug phase of this research, bottleneck conditions were intentionally created by altering: 1) the material handling capacities of the robots, 2) the machine efficiencies, and 3) the buffer capacities.

4.3.3.3 Data packet structure

Data packets are built and decoded at the *interface* (see figure 4.9) modules of the server and client nodes. The format of the data packets transmitted between the SCADA server and client application is shown in figure 4.12. Figure 4.12 (a) gives an example of the packet structure built at the client node and figure 4.12 (b) is an example of a server packet structure; both packets are decoded at the other node (i.e. a client packet is decoded at the server node and vice versa).

```
#client; aid=1; amode=0; agoal=0; apass=0; end#
```

(a)

```
#server; aid=1; svm=1; data=1 1:80 2:10 3:0 4:10 5:0 6:0 7:0 8:0; ago=0; aclr=0; tout=3; end#
```

(b)

FIGURE 4.12: (a) Client data packet structure (b) Server data packet structure

The packets contain vital information required by the simulation code in the SCADA system and the SVM module in the client application. The following list outlines the meaning of each parameter in the data packets:

- header: the start of the packet, describing where it was built, either *#client* or *#server*.
- *aid*: the robot agent identification, can range from 1 to 999. In this research, the *aid*'s used were 1, 2 and 3, for the PeopleBot, RMP200 and RMP400 respectively.
- *amode*: the agent mode, ranges from 0 to 6. Table 4.3 gives the description of each mode.

TABLE 4.3: Agent modes

Mode no.	Description
0	Wake up
1	Going home
2	At home
3	Going to source
4	At source
5	Going to destination
6	At destination

- *agoal*: refers to the agent goal location, ranges from 0 to 3 in this research. The "0" value is the robot agent's home location, "1" is the buffer service 1 location (the area between the resource and buffer 1), "2" is the buffer service 2 location (the area between buffers 4 and 5), and "3" is the buffer service 3 location (the area between buffer 6 and the storage).
- *apass*: the agent passport parameter, where a value of "0" means that the agent is not waiting to go to the buffer and "1" implies that the agent is waiting.

- *svm*: SVM mode, ranges from 0 to 2. Table 4.4 gives the description of each mode.

TABLE 4.4: SVM modes

Mode no.	**Description**
0	Do nothing
1	SVM learn mode
2	SVM train–predict mode

- *data*: this portion of the packet represents the number of materials in each buffer. The "x:" (where x is a number from 1 to 8) represents the buffer number and the value attached is the number of materials. For example, "1:80" represents 80 materials in the resource buffer. The first number after the *data* parameter is the value of the *agoal* parameter.
- *ago*: agent go/stop command, where a value of "0" means that the agent must stop and "1" commands the agent to go ahead.
- *aclr*: agent clear mutex, where a value of "0" means that the agent is not clear to proceed and "1" is the agent's clear-to-proceed flag.
- *tout*: timeout for the agent to try communications again with the SCADA server, ranges from 0 to 65535 seconds.
- tail: the end of the packet, *#end*.

The use of some of these parameters will be made clear in the next section since they are required by the SVM module.

4.3.4 SVM module

This section discusses the implementation of the SVM learning algorithm in the research. The LIBSVM library was used in the client program for the train and prediction algorithms, and the polynomial kernel was chosen as the non-linear SVM kernel function.

4.3.4.1 Program structure

The flow diagram for the SVM module is shown in figure 4.13, where the *Run SVM* block represents the SVM algorithm.

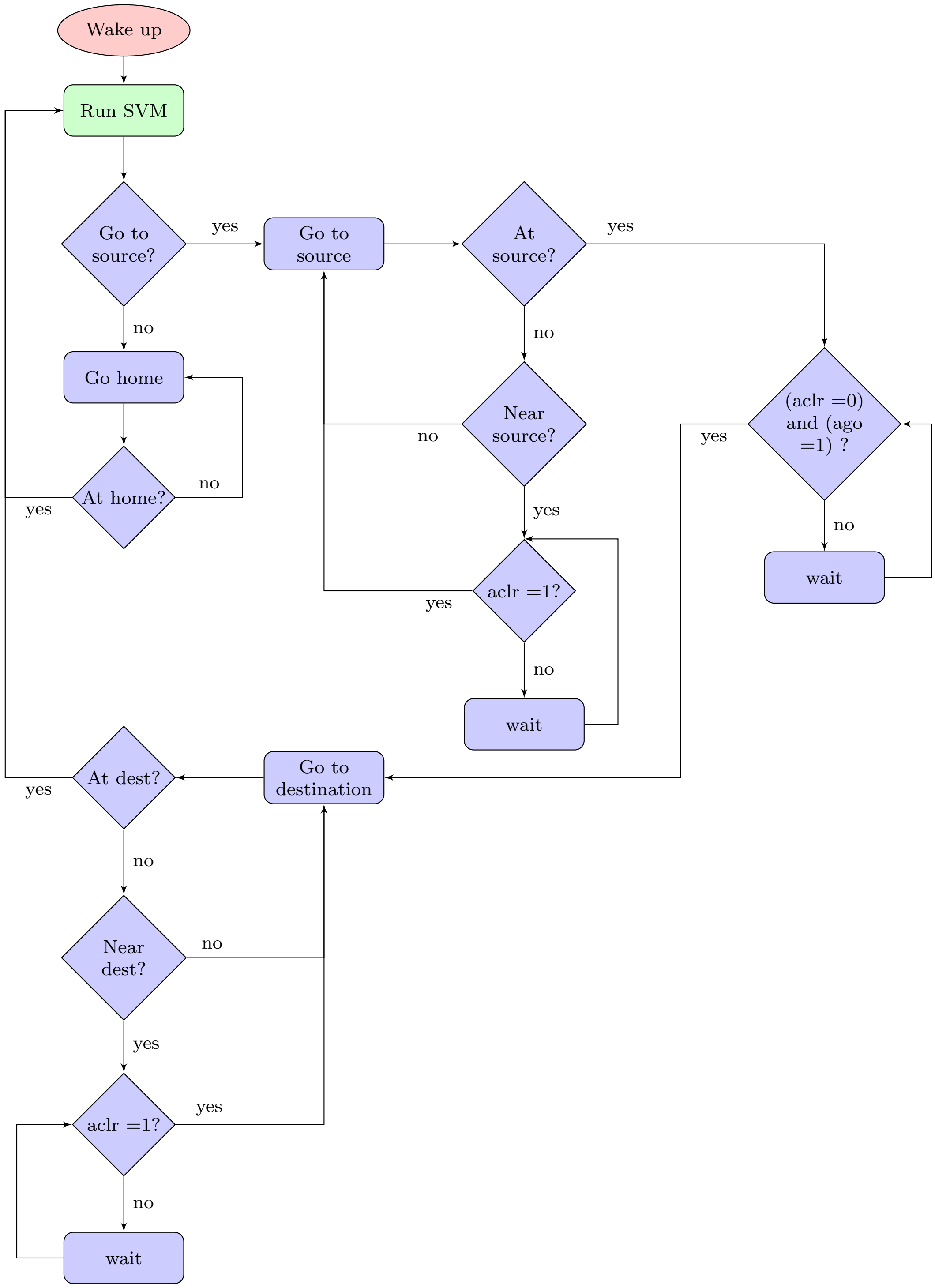

FIGURE 4.13: SVM module flow diagram for each robot agent

The home location is the "park-off" position of the robot, and the source and destination locations are the buffer service material pick-up and material drop–off positions respectively. For example, in figure 4.11, one of the source locations is the "R" buffer and the corresponding destination location is buffer "B1".

4.3.4.2 Learn, train and predict

There are two phases to the SVM algorithm:

- the *learning* phase, where agents are taught by the system on the best goal location to follow. The teaching process can take place in an offline (simulation) environment, or online through the GUI interface of the SCADA system.

  The objective of the learning phase is to build a knowledge database of SVM features with training examples. Figure 4.14 is an extract of the "train.txt" file that contains the training examples. The SVM features in the file are the buffers in the manufacturing application and the training examples are the number of materials in each buffer as well as the (output) goal location for the robot— shown as the first number in each line of the file.

```
1 1:100 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:20 3:0 4:0 5:0 6:0 7:0 8:0
1 1:60 2:0 3:0 4:0 5:15 6:1 7:4 8:0
1 1:60 2:0 3:0 4:0 5:15 6:0 7:5 8:0
```

FIGURE 4.14: Train.txt file extract with SVM features and training examples

  The robots are taught by going through a "teach mode" process, using the GUI interface (figure 4.22) of the SCADA system. The operator assigns each robot to a particular task by selecting the appropriate buffer service (BS) location during the simulation or the actual production process. The operator chooses the BS tasks for the robots by considering the buffer levels and the abilities of each robot.

- the *train–prediction* phase uses the data collated in the learning phase (i.e. the data contained in the train.txt file) to generate training models for each agent; the goal output for each agent is then accomplished by using the current data values (obtained from the data packet) as inputs to the prediction algorithm. The current data values represent the immediate status of the manufacturing process; they are stored as a string of data in the "test.txt" file which is used as an input to the

SVM prediction algorithm. Figure 4.15 illustrates the entire process of training, building the model, and predicting the goal output for each robot in the system.

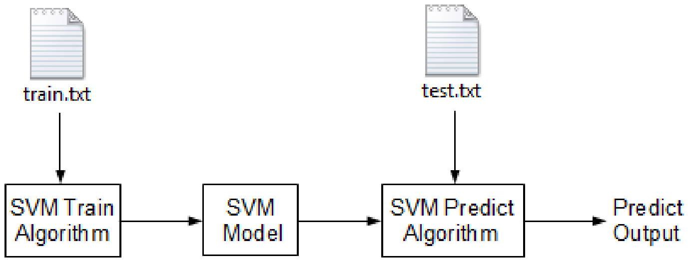

FIGURE 4.15: Process of the SVM train–predict phase

4.4 SCADA development

Chapter 3 discussed the design choice of using the Proview SCADA package for this research. The following sections cover the development and integration of the SCADA system with the rest of the modules already discussed in previous chapters.

4.4.1 Proview environment

This section introduces the Proview SCADA environment and the associated object orientated structures involved in the development of a SCADA project. Projects are created in the Proview "Project List" space, shown in figure 4.16.

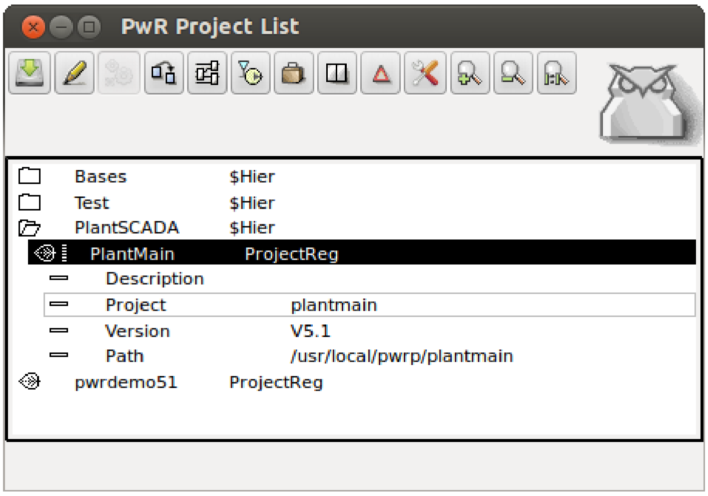

FIGURE 4.16: Proview project list

Each project has a directory volume in which the volumes and nodes of the project are created. Figure 4.17 is a screenshot of the directory volume used for the SCADA development of this research.

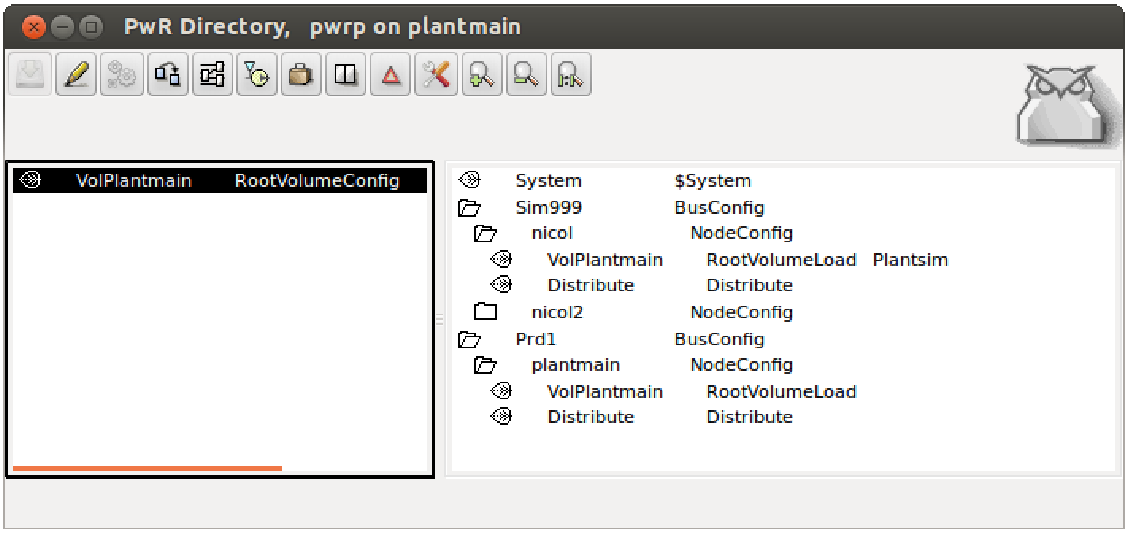

FIGURE 4.17: Proview volume directory

A volume is like a container that holds objects, ordered in a tree structure. There are various types of volumes that can be created in Proview, the one implemented in this research is the root volume. In the right window of figure 4.17, the nodes of the project are created: nodes are grouped by the QCOM bus port they communicate on. The two *BusConfig* nodes that were created are “Sim999” (simulation) and “Prd1” (production), each containing *NodeConfig* children objects that contain configurable node names and IP addresses. The simulation and production *BusConfig* nodes can contain objects that are assigned to the same root volume; in this way, SCADA development is done on one volume which can be tested in a simulation environment for prototyping/debugging, thereafter easily applied to the actual production environment.

Every volume can be configured and edited in a volume configuration space, shown in figure 4.18. The editor is split into two windows, the left window is the plant configuration and the right window is the node configuration.

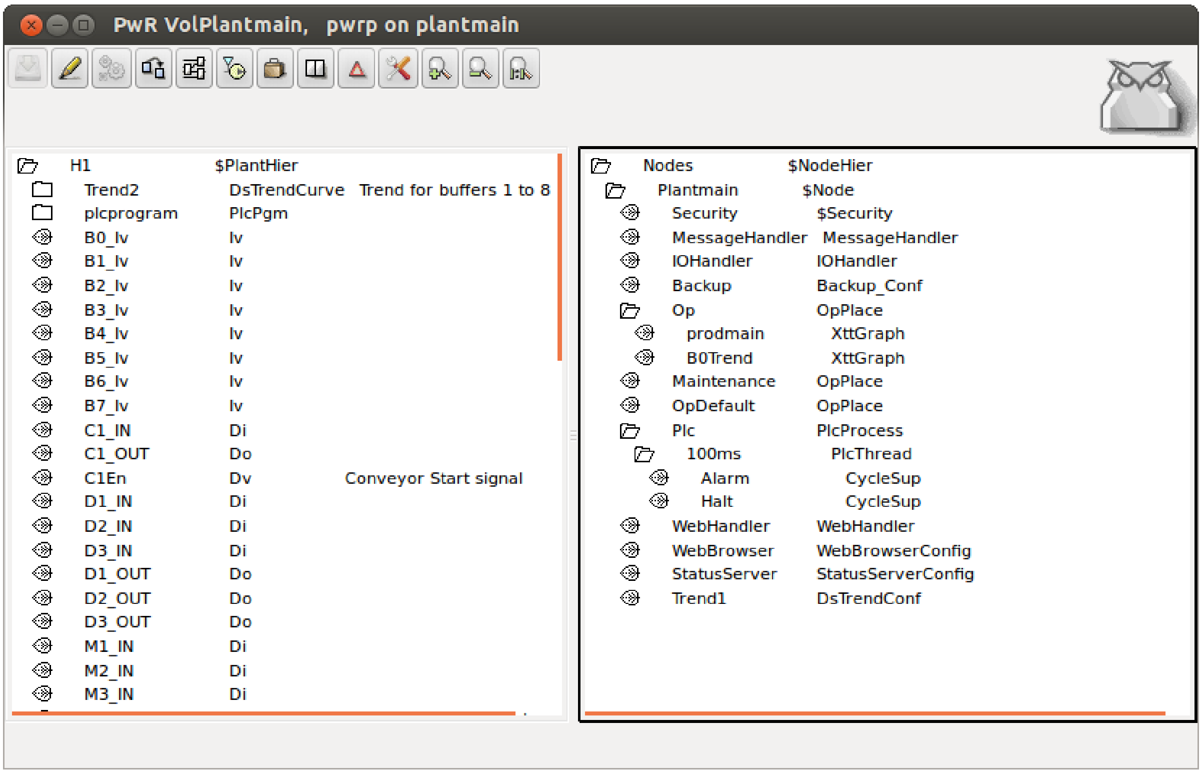

FIGURE 4.18: Proview volume configuration space

The plant configuration contains 1) signal objects, and 2) PLC program (*PlcPgm*) objects:

- signal objects represent the Input/Output (I/O) signals that are used somewhere in the process. Proview supports various I/O types, some of the commonly used ones are digital, analog, integer, and string.
- PLC objects define the PLC program elements, such as the I/O configuration and connections between I/O. Several PLC programs can be configured in a single plant setup.

The node configuration specifies the type of I/O system used in the project, such as rack and card, distributed I/O, or process and thread. The *OpPlace* object is also defined here; this parent object has *XttGraph* children objects which are used to configure GUI interfaces through Proview's Ge editor.

4.4.2 SCADA development overview

A development overview of the SCADA system for this research is given in figure 4.19. The Proview runtime environment calls the user application programs (*application interface* and *simulation program*), the project PLC program, and the GUI, developed in Ge editor.

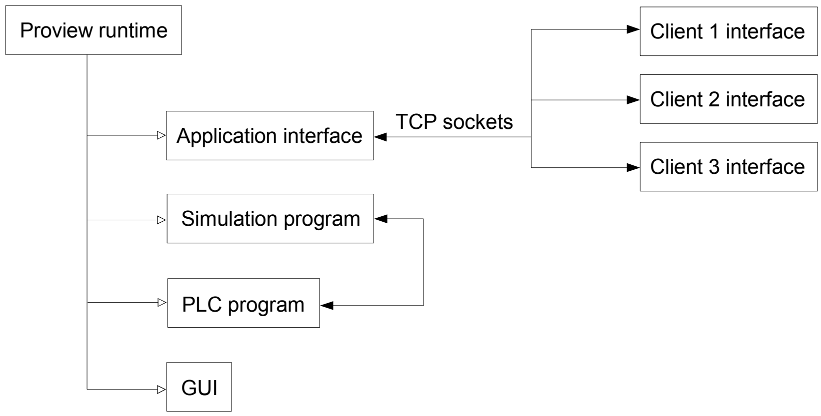

FIGURE 4.19: Proview SCADA development overview

4.4.3 Application interface

Proview supports the integration of user application programs to the SCADA system. The programs can be written in C, C++ or Java and the programmer can attach the real time database, `rtdb`, to read and write the I/O data.

The application interface code was written in C++ and can be viewed in appendix B.1. The purpose of this program is: 1) to function as the TCP server socket and communicate with the clients in the network, 2) to read and decode the data packets received from the clients, thereafter write the necessary data to associated SCADA signals, 3) to read data signals from the SCADA system, then build data packets before sending them to the corresponding client.

Section 4.3.3.3 discussed the structure of the data packets. The application interface builds the packet to send the *aid*, *svm*, *data*, *ago*, *aclr*, and *tout* parameters and encodes the packet received from the client to retrieve the *aid*, *amode*, *agoal*, and *apass* parameters.

4.4.4 Simulation program

The simulation program was written in C++ and can be viewed in appendix B.2. The program uses Proview's real time database to read and write I/O. The functionality of the simulation program is described by the following itemised list:

- get agent modes (*amode*): determine whether the robot agent is at, or going to, its home, source or destination location.
- get agent svm modes: identify whether a particular agent is in a teach or learn–predict SVM mode.
- get agent buffer goals when robot at home or at destination (*agoal*): determine which buffer service (BS) a particular agent is required to go to— this location is shown in the GUI.
- get agent passports (*apass*): check whether an agent is waiting to go to a buffer.
- set agent clear flags (*aclr*): control the "clear to proceed" flags for each agent.
- calculate buffer levels at source and destination locations: increase or reduce the material quantity in these buffers at the right time (determined by the PLC program). The amount of material that is loaded or off–loaded depends on the load carrying capacity of the robot as well as the amount of material the robot is actually carrying. The load carrying capacities of each robot agent are configured as constants in the simulation program.
- set agent "go/stop" flags (*ago*): control whether an agent can move ahead or is required to stop.
- calculate buffer levels effected by the machines and the conveyor: increase or reduce the material quantity in these buffers at the right time (determined by the PLC program). The efficiencies of the conveyor and the machines are set to constant values; these efficiencies can influence bottleneck conditions in the production process, however, they are not set as variables in this research. Bottleneck conditions were solely controlled by the load carrying capacities of the robots and the failure of individual robots.

4.4.5 PLC program

The Proview PLC program developed for this research is a simple one and could have been omitted from the SCADA system by being included in the simulation program. The two reasons for including the PLC program in the system are: 1) it is much easier and efficient to implement the use of timers in the PLC than writing timer code in the simulation program; 2) it was a research interest to integrate and test the Proview PLC module with the rest of the system.

Figure 4.20 is the PLC program developed in Proview. The *wait* function blocks are delay timers that are activated when the digital input signals are positively edge triggered. The output signals are set when the timer elapses. The purpose of this PLC program is to simulate the completion of: 1) material loading and off–loading at the source ("S" signals) and destination ("D" signals) buffers respectively, and 2) machine ("M" signals) and conveyor ("C" signals) process times.

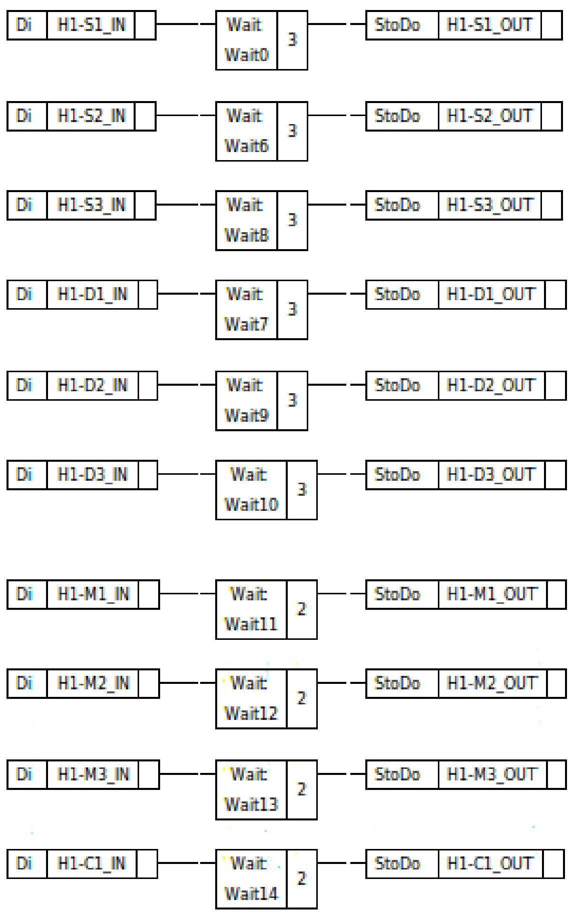

FIGURE 4.20: Proview PLC program

4.4.6 Graphic interface development

The GUI was developed in Proview's Ge editor package, shown in figure 4.21 and the final interface is illustrated in figure 4.22.

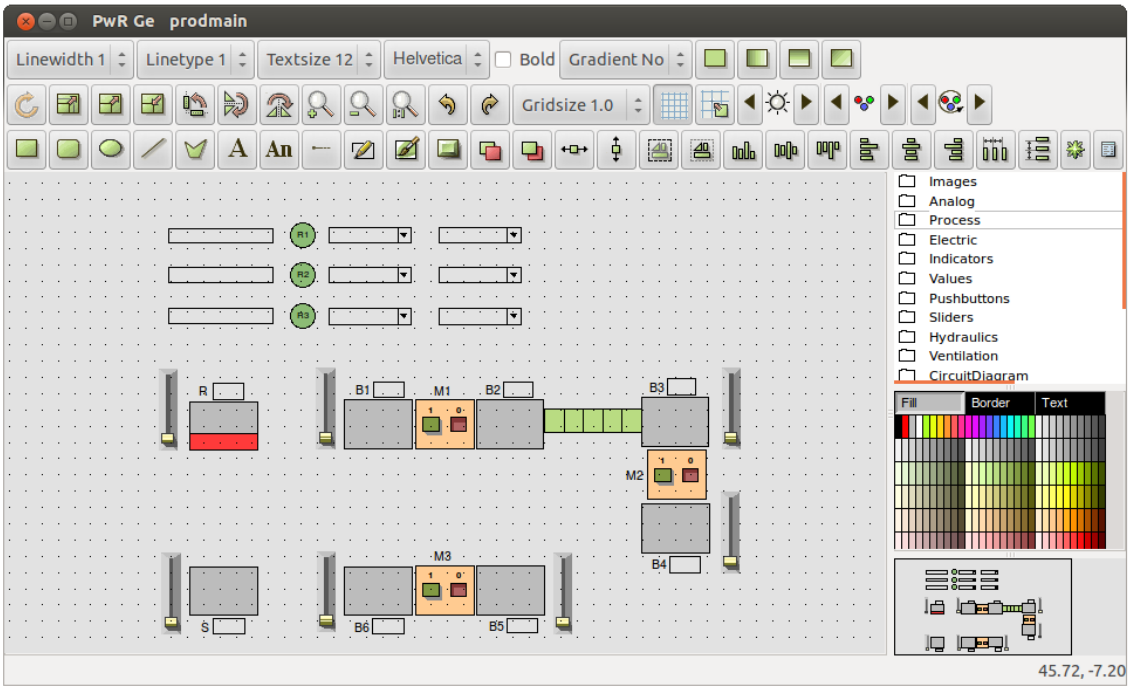

FIGURE 4.21: Ge editor development environment

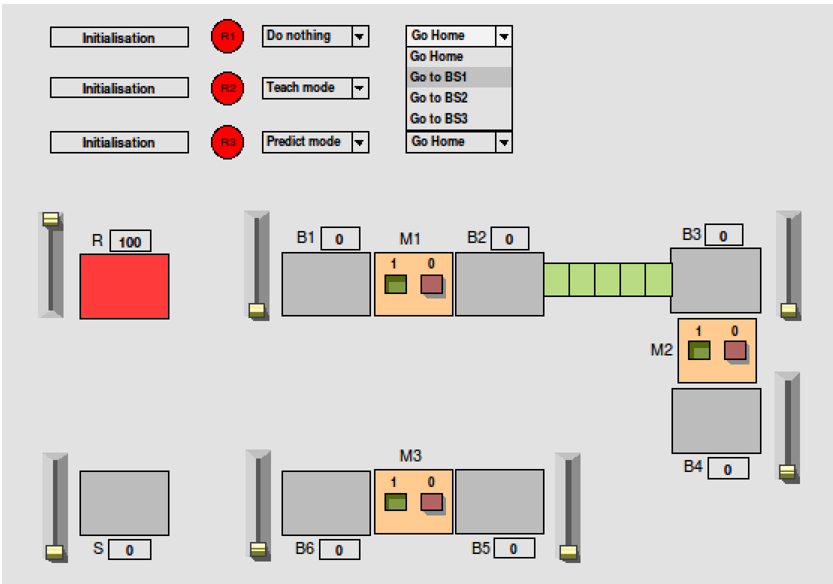

FIGURE 4.22: GUI screen in runtime

The GUI screen has the following characteristics:

- displays the present goal of the robot agent, whether it is going to the buffer source or destination.
- allows the operator to control the SVM mode of each agent, be it “teach mode” or “predict mode”.
- displays the current buffer service (“BS1”, “BS2’ or “BS3’) goal of each agent, or whether the agent is going home. The BS goal can also be a configurable parameter if the agent is operating in a SVM teach mode.
- displays the number of materials in each buffer.
- displays the buffer levels (as a percentage).

The GUI screen also shows vertical level control bars at each source and destination buffer; these were used to control the buffer levels during the test and implementation phase of the research.

4.5 Chapter summary

At the outset of this chapter, the AI development overview was given which discussed the two development routes taken in this research: the simulation and real-world developments. The Player, Stage and client configurations of each development was addressed. The chapter also discussed the cognition and SVM modules of the client application and code references to the appendix were made. The latter part of the chapter introduced the Proview SCADA development environment and discussed the components involved in the SCADA part of the research, namely the application interface, the simulation program, the PLC program, and the GUI.

Chapter 5

Results and Discussion

The Mechatronic design and development of the components discussed in this research were tested in simulated and real–world environments. This chapter begins by describing the laboratory layout of the two environments as well as the actual robot assemblies used in the tests; the sections that follow include the test results for robot cooperation and cognition.

5.1 Simulation environment

A map of the lab environment used in the Stage simulation is depicted in figure 5.1.

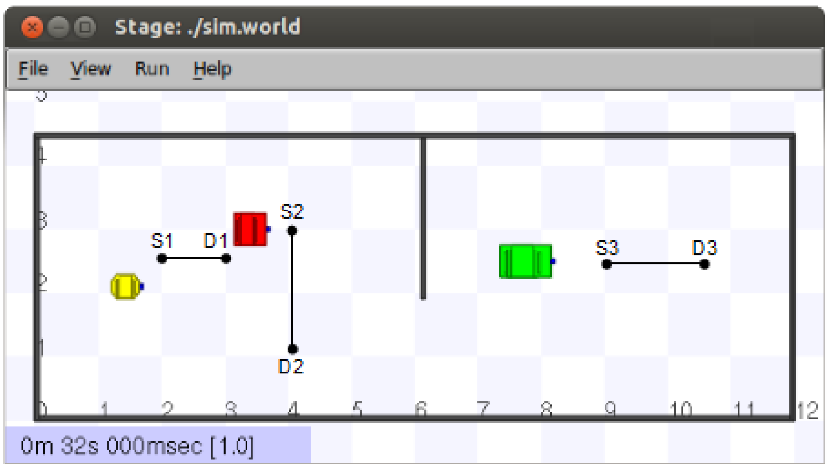

Figure 5.1: Stage simulation environment with robot and buffer locations

The map shows two rooms, separated by a wall and doorway. The room on the left is used for tests with the PeopleBot (yellow robot) and the RMP200 (red robot), whilst the room on the right is used for the RMP400 (green robot). The "S" and "D" labels in the figure are the source and destination locations for buffer service 1, 2 and 3. The PeopleBot's primary goal is to move material between "S1" and "D1", however it was also trained to be a helping agent to the RMP200, whose only task is to move material between "S2" and "D2". The RMP400 also has a single task of handling material from "S3" to "D3". The home locations for the robots are their current positions shown in figure 5.1 and, as mentioned in chapter 3, the localisation method used is odometry together with a memory of the robot's x–y coordinate position in the map.

Figure 5.2 is a screenshot of a simulation in progress. The picture shows 1) the Proview SCADA window, 2) the Stage simulation window and simulation terminal, and 3) three terminal windows— one for each client application. The clients connect to the Player server on the local IP address of the Ubuntu machine.

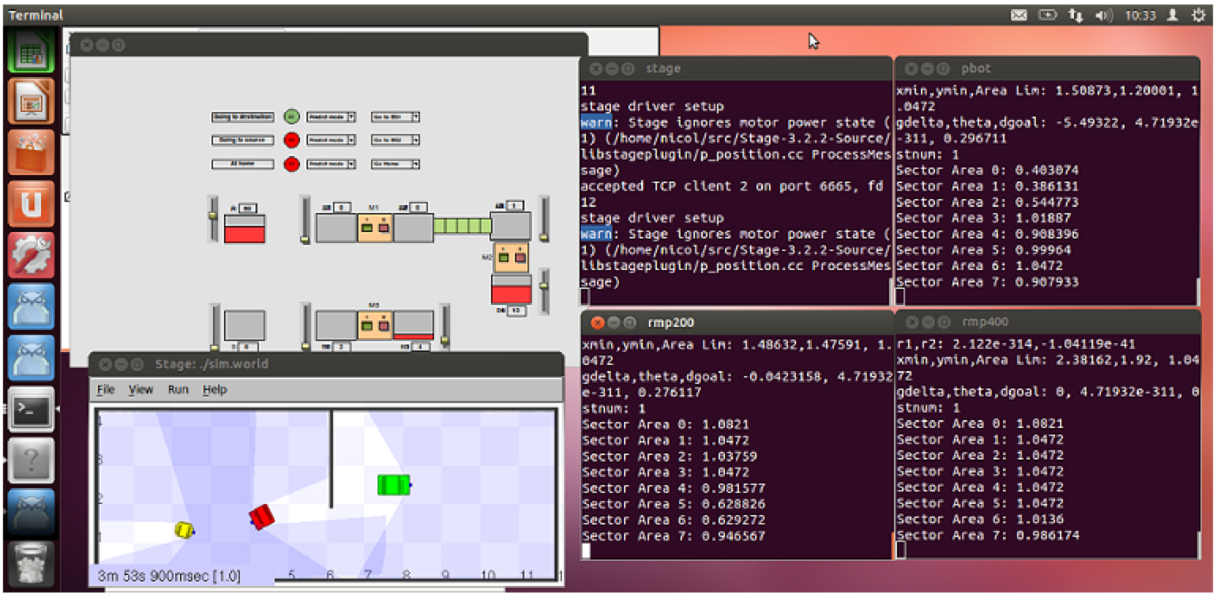

FIGURE 5.2: Simulation system: SCADA, Stage and client applications

5.2 Real–world environment

The real–world tests were performed in the Mechatronics lab at the University of KwaZulu-Natal. The three robots discussed in this research were used in the actual tests.

5.2.1 Robot system assemblies

The PeopleBot assembly is shown in figure 5.3. The unused hardware are the Pan–Tilt–Zoom (PTZ) camera, 2D gripper, and the SICK LRF. The top and bottom sonar array sensors were used as the range finders in the navigation algorithm and the e-stop button was used to stop the robot during its autonomous operation. This robot contains an on-board PC, located in its base, and the wi-fi antenna (located behind the monitor) allowed access to wireless communication with the remote SCADA PC.

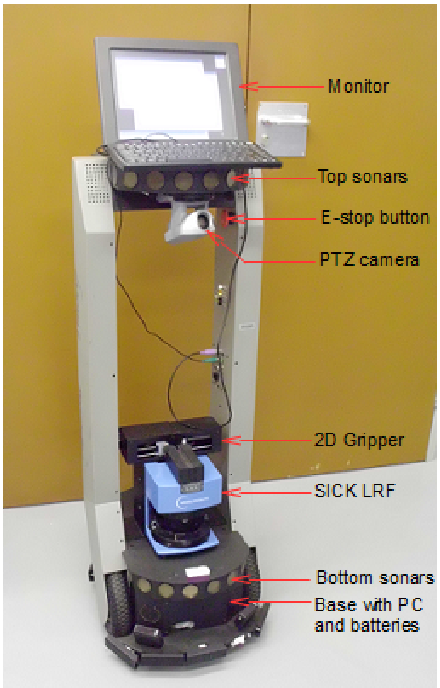

FIGURE 5.3: PeopleBot system assembly

The Segway RMP200 assembly is shown in figure 5.4. The client application for this robot was executed on the laptop PC, supported by the sheet metal bracket. The other hardware installed on the bracket is the Hokuyo LRF, located at the front of the robot, the regulated power supply circuit for the LRF, and the 6V battery which is used to supply power to the voltage regulator. The power supply circuit was neatly installed in an enclosure and the two-way switch was mounted outside the enclosure, allowing easy access to control power to the LRF. The user controls, used to switch the robot on and select balance or tractor mode, are located at the rear of the robot, hence they cannot be seen in figure 5.4, however, they are shown in figure 2.3. The balance mode was used during tests.

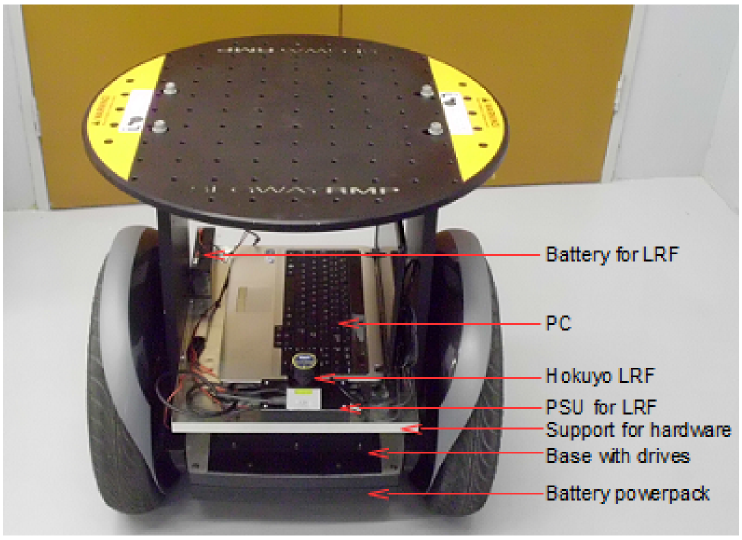

FIGURE 5.4: Segway RMP200 system assembly

The assembly of the Segway RMP400 system is almost identical to the RMP200 system in terms of the LRF and power supply arrangement, shown in figure 5.5. This assembly also executes the client application on the local PC and communicates with the SCADA system through the laptop's wi-fi adapter.

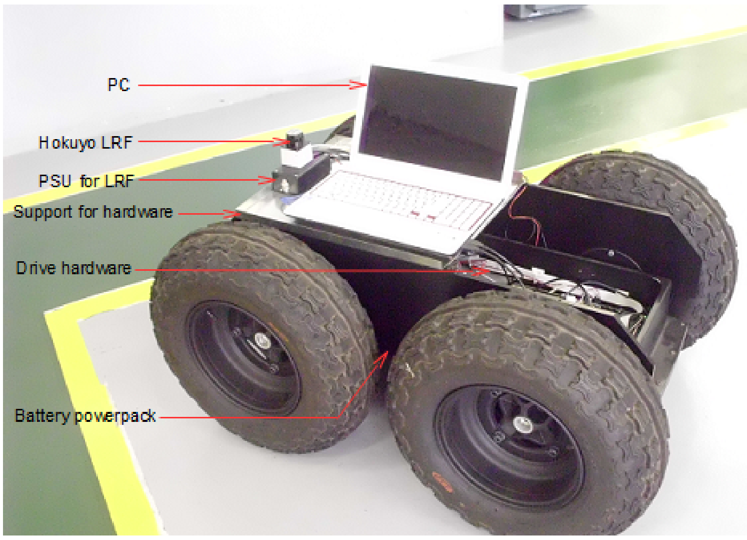

FIGURE 5.5: Segway RMP400 system assembly

5.2.2 Laboratory layout

The actual lab rooms used for the robotic system tests are shown in figure 5.6. The room at the top of the figure was used for tests with the PeopleBot and the RMP200, whilst the room at the bottom was used for tests with the RMP400. The rooms are separated by a wall and doorway, similar to the map used for the Stage simulation, in figure 5.1.

FIGURE 5.6: Lab rooms used during tests

The buffer source and destination locations are given in figure 5.7. The "H" labels represent the home positions for each robot. The x–y coordinate locations for the real–world tests were configured the same as the simulation system so that the performance of the robots in the real–world could be measured and debugged in relation to the simulation results. The short distance movement of the robots between source and destination locations may seem like a trivial exercise, however, the objective of the research is to use the concept to demonstrate cooperation among robots in bottleneck conditions.

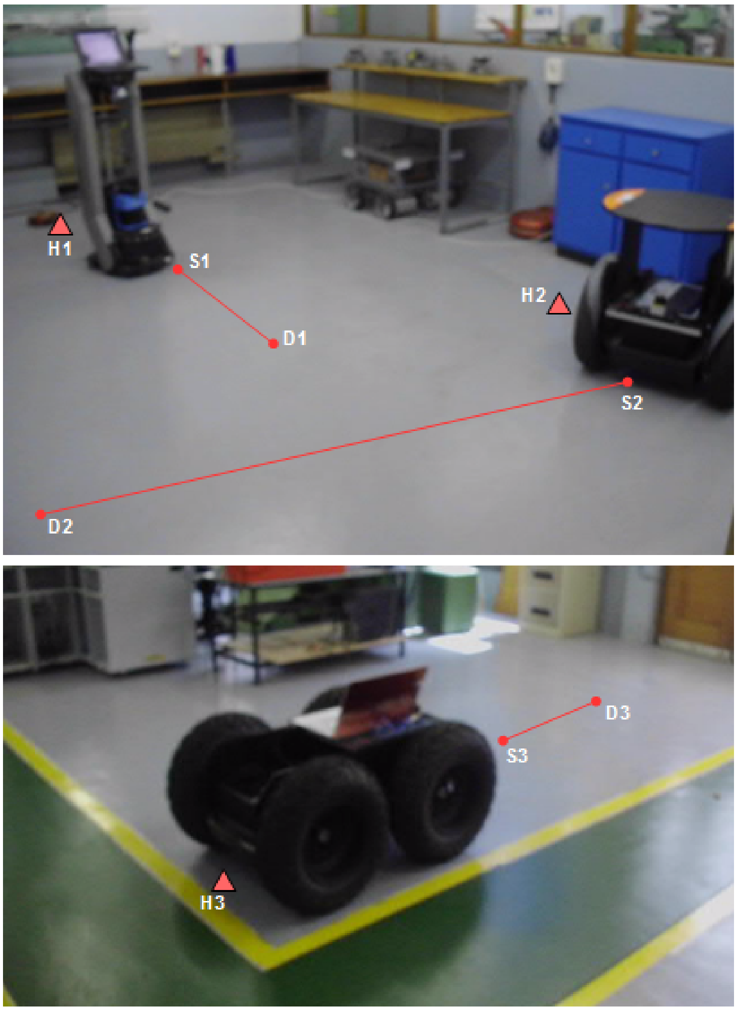

Figure 5.7: Lab environment with robot and buffer locations

5.3 Cooperation results and discussion

This section produces the results of the tests performed during simulation and real–world conditions. Bottlenecks were created by varying the load carrying capacities of the robots, however, there were other options by which this could have been done, namely: 1) vary the machine or conveyor efficiencies, and 2) change the buffer capacities.

During the SVM teach phase of the tests, the PeopleBot was taught to help the RMP200 at the bottleneck. Figure 4.11 showed a screenshot of the material handling application, where the PeopleBot's primary task is to move materials from the resource buffer ("R" or "B0") to "B1", the RMP200 has the single task of transporting material from "B4" to "B5", and the RMP400 also has a single task of moving the final product from "B6" to the storage buffer ("S" or "B7").

A bottleneck was created at "B4" by reducing the load carrying capacity of the RMP200 from 20 materials to 5 materials. The capacity of the PeopleBot remained the same (at 20 materials), this ensured that the material build up rate at B4 was greater than the buffer process rate, resulting in a bottleneck.

5.3.1 Simulation performance

Four types of simulation tests were performed:

- normal operation: the load carrying capacities of the robots were configured to prevent bottleneck conditions.
- bottleneck condition: the load carrying capacities of the robots were configured to promote bottleneck conditions.
- cooperation at the bottleneck: a robot agent was allowed to help another agent at the bottleneck.
- cooperation during a robot fault: a robot agent was allowed to take over the tasks of the faulty robot so that the possibility of the occurrence of a bottleneck is reduced.

5.3.1.1 Normal operation

The material distribution graph for the *normal operation* simulation test is given in figure 5.8. The graph has three axes: the x–axis represents the buffer locations, ranging from 0 (buffer B0) to 7 (buffer B7); the y–axis represents the time (in seconds) of the simulation; the z–axis gives the number of materials, in a percentage, at each buffer location. The percentage is calculated by the following equation:

$$B_{size} = \frac{B_{num}}{B_{cap}} * 100 \tag{5.1}$$

where B_{num} is the number of materials in the buffer and B_{cap} is a constant which represents the number of materials that the buffer can contain, i.e. the buffer capacity.

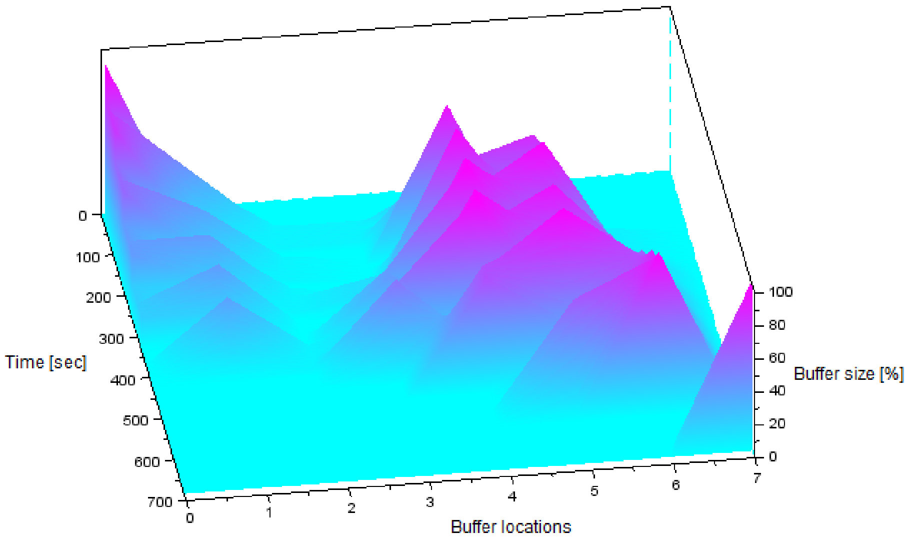

Figure 5.8: Material distribution graph: normal operation

The visual trend in the graph shows a decrease in material count at the resource buffer (which was initialised with 100 materials) and an increase in material count at the storage buffer, towards the end of the simulation. Table 5.1 gives more detail to the *normal operation* simulation and lists the values of some test parameters such as the total simulation (or production) time and the total operation time of each robot agent.

Table 5.1: Test parameter values during normal operation

Test parameter	**Value**
Total simulation time	683 sec
Agent 1 load capacity	20 materials
Agent 2 load capacity	20 materials
Agent 3 load capacity	100 materials
Agent 1 operation time	332 sec (48.6%)
Agent 2 operation time	434 sec (63.5%)
Agent 3 operation time	103 sec (15.1%)
Buffer 4 @100%	54 sec (7.9%)

Agents 1, 2 and 3 are the PeopleBot, RMP200 and RMP400 respectively. The values within brackets in the table are the percentages of the total simulation time. With the exception of the resource and storage buffers, the only other buffer that reached its full capacity during the simulation was the source buffer for the RMP200, B4: the buffer was at the total size of 100% for 7.9% of the total simulation time.

The navigation path of each robot in the simulation is given in figure 5.9 which illustrates the niche areas covered by the robots in the environment. The x–y positions of the robots were recorded every second during the simulation by writing the coordinates to a text file. The results shown in the figure depict a plot of the coordinates from the beginning to the end of the simulation. The figure also gives the x–y location of the robot's source and destination buffer locations so that the navigation path between the locations can be identified. The navigation paths in the figure give evidence that each robot performs a single task in the simulation.

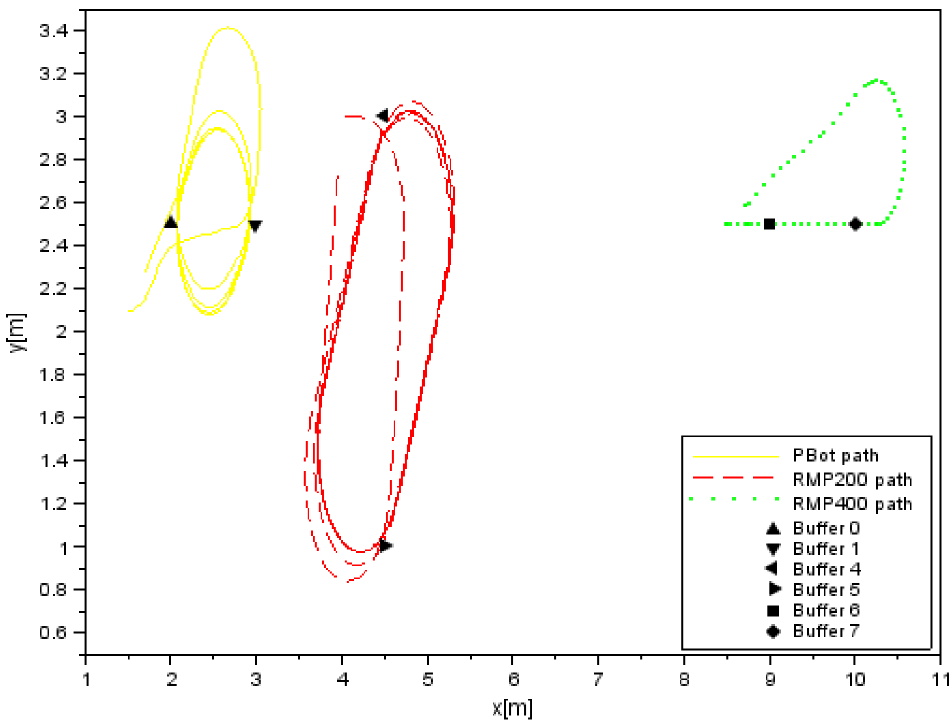

FIGURE 5.9: Robot navigation: normal operation

5.3.1.2 Bottleneck condition

In comparison to figure 5.8, figure 5.10 (and table 5.2) shows a significant change in the material distribution. The one modification made in this simulation was a change in the material load capacity of agent 2, the RMP200, from 20 materials to 5 materials which resulted in the total simulation time of 1763 seconds— a 158% increase in time from the previous simulation.

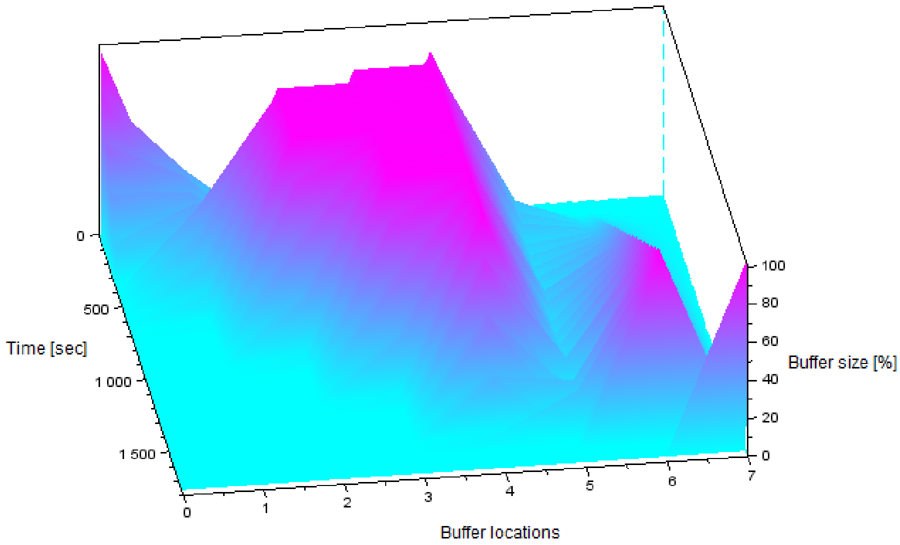

FIGURE 5.10: Material distribution graph: bottleneck condition

TABLE 5.2: Test parameter values during a bottleneck condition

Test parameter	**Value**
Total simulation time	1763 sec
Agent 1 load capacity	20 materials
Agent 2 load capacity	5 materials
Agent 3 load capacity	100 materials
Agent 1 operation time	349 sec (19.8%)
Agent 2 operation time	1520 sec (86.2%)
Agent 3 operation time	93 sec (5.3%)
Buffer 2 @100%	282 sec (16.0%)
Buffer 3 @100%	594 sec (33.7%)
Buffer 4 @100%	936 sec (53.1%)

The obvious reason for the large increase in simulation time is due to the bottleneck at buffer 4, where the RMP200 cannot transport the required amount of material to keep up with the incoming rate at the buffer. Table 5.2 further exemplifies the bottleneck problem by listing an increased time at which buffer 4 was at 100% in size; this caused a cascaded effect (depicted in figure 5.10) to fill up buffer 3 and buffer 2. The purpose of the *bottleneck condition* simulation was two–fold: 1) to emphasise the impact of the bottleneck on the production system, and 2) to set the stage for an implementation of the cooperative intelligence system in mitigating the bottleneck.

5.3.1.3 Cooperation at the bottleneck

The material distribution graph in figure 5.11 reflect the results of the cooperative intelligence system.

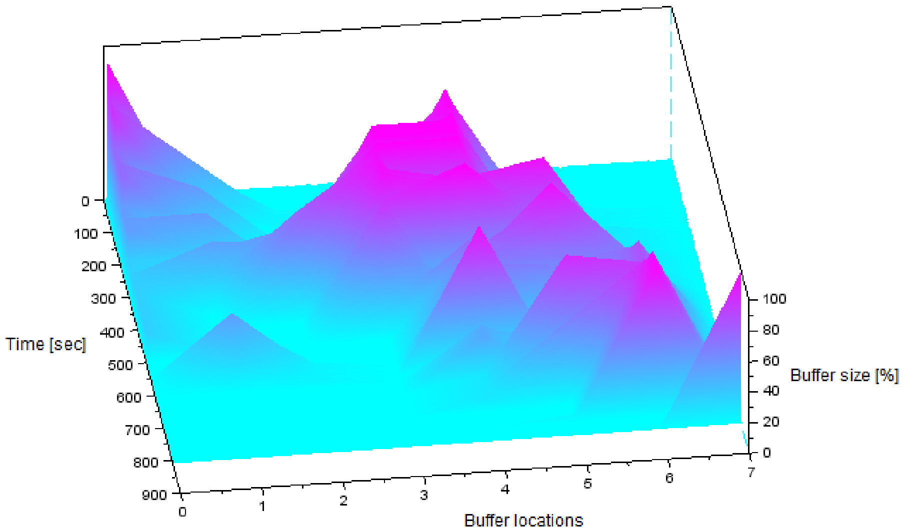

Figure 5.11: Material distribution graph: robot cooperation at bottleneck

The *cooperation at the bottleneck* simulation was performed by allowing the SVM–trained PeopleBot agent to assist the RMP200 agent at the bottleneck (buffer 4), hence the PeopleBot executes its primary task of transporting material from B0 to B1 as well as "cooperates" by effecting its secondary task of moving material from B4 to B5. The result of the cooperating agent is depicted by the navigation path illustration in figure 5.12 where the PeopleBot covers two areas in the environment as opposed to the one area in figure 5.9. An analysis of the SVM output results in the subplot of figure 5.13 gives an interesting perspective on the periods at which the algorithm determines the assistance of the PeopleBot at the bottleneck.

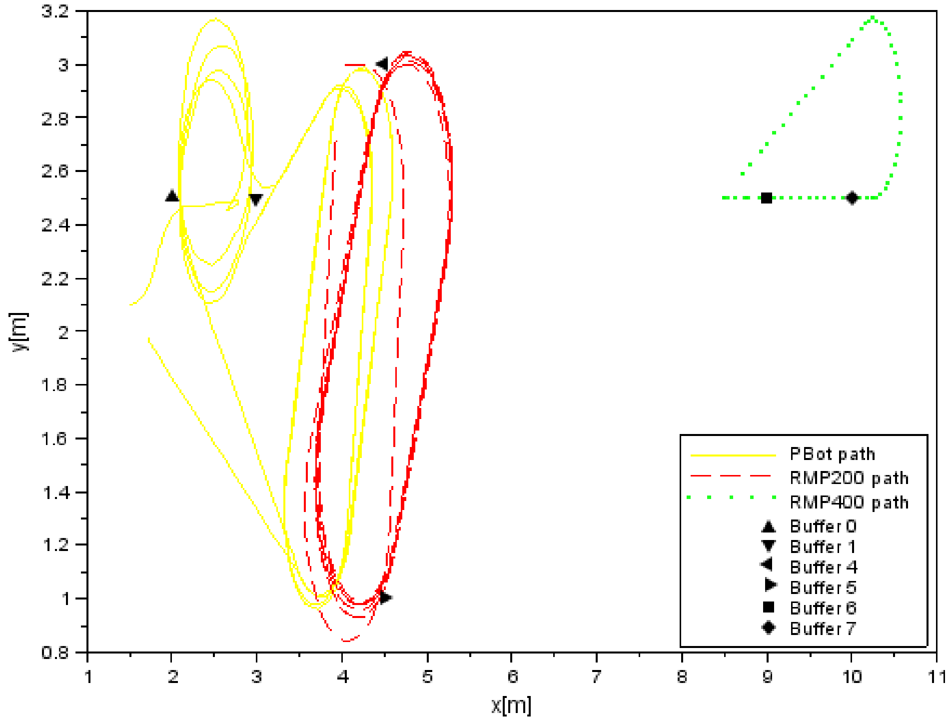

FIGURE 5.12: Robot navigation: cooperation at bottleneck

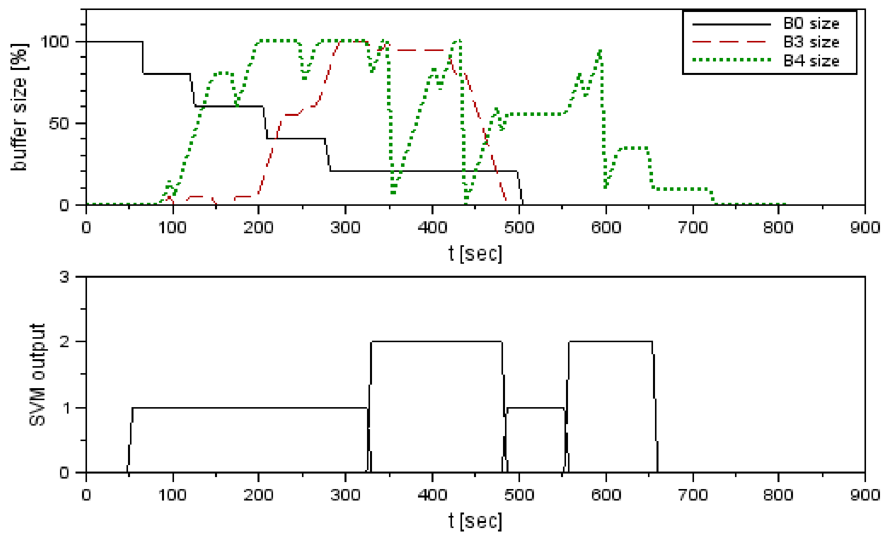

FIGURE 5.13: PeopleBot SVM outputs: cooperation at bottleneck

The SVM outputs for the PeopleBot agent are either "1" or "2", representing the primary or secondary task respectively. During the teach phase, the PeopleBot agent was taught to assist at B4 when the size of B0 is low and when the sizes of B4 and/or B3 are high. The effect of the teaching exercise is clearly shown in figure 5.13 since the SVM predictions are "2" during conditions where the test parameters of the SVM features (i.e. the buffer sizes) are approximately the same as the SVM training examples.

Table 5.3 lists the total simulation time of 809 seconds— a 54% reduction in comparison to the previous simulation case. The table also reflects the task distribution percentage for agent 1: the SVM algorithm determined the secondary goal for the PeopleBot 3 times out of a total of 8 iterations in the simulation, i.e. the PeopleBot spent 37.5% of its operation time on the secondary task and 67.5% on its primary task. The simulation also resulted in an elimination of buffer 2 from the bottleneck cascade and showed reduced buffer–full times of buffer 3 and buffer 4 to 5.2% and 17.1% respectively.

Table 5.3: Test parameter values during robot cooperation at the bottleneck

Test parameter	**Value**
Total simulation time	809 sec
Agent 1 load capacity	20 materials
Agent 2 load capacity	5 materials
Agent 3 load capacity	100 materials
Agent 1 operation time	600 sec (74.2%)
Agent 1 primary task	62.5%
Agent 1 secondary task	37.5%
Agent 2 operation time	678 sec (83.8%)
Agent 3 operation time	91 sec (11.3%)
Buffer 3 @100%	42 sec (5.2%)
Buffer 4 @100%	138 sec (17.1%)

During navigation, the robots do not arrive at the exact position of their goal location but rather reaches the position a few tenths of a meter short, due to the goal tolerance set point configured in the code. During the tests, the set point was configured to 0.3 meters, hence the agent will think it has reached the goal if it is localised within 0.3 meters of the goal coordinate. The results of the tolerance distances at each goal in the simulation is given by the interpolated plots in figure 5.14, none of the points go beyond the tolerance threshold of 0.3 meters. The figure shows the maximum distance margin from the goal for each robot, where the largest margin was 0.29 meters from the goal at 234 seconds into the simulation.

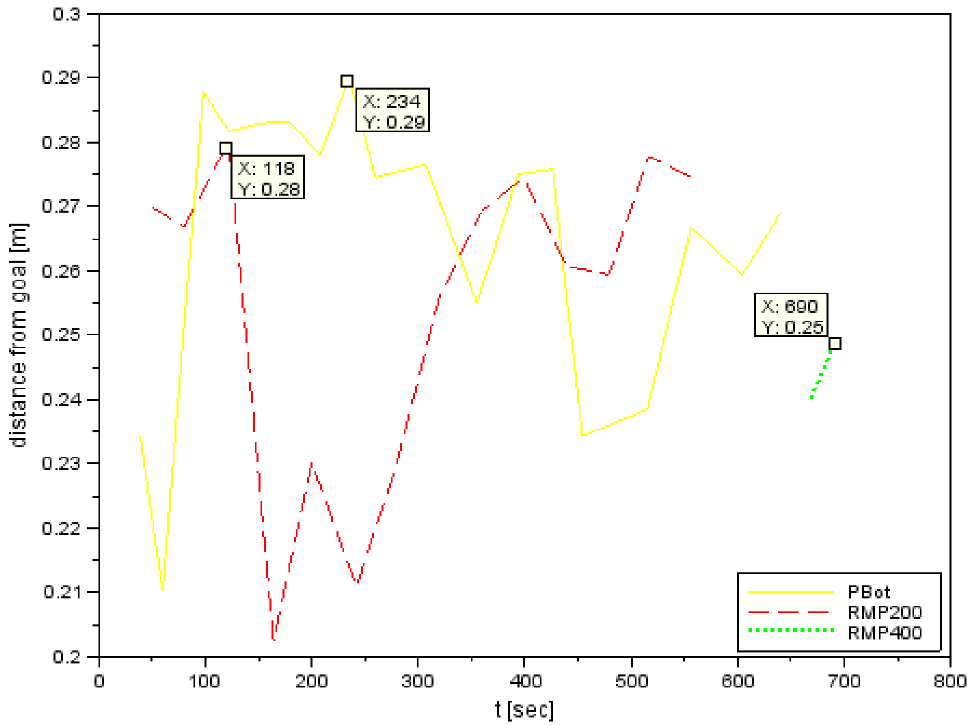

FIGURE 5.14: Robot goal tolerance simulation results

Figure 5.15 represents a plot of the distances between the PeopleBot and the RMP200 in the duration of the simulation. The smaller distances (less than 1 meter) in the plot indicate the periods in the simulation where the obstacle avoidance routines are carried out in each agent. The obstacle avoidance code was written such that a safe distance of 0.25 meters is maintained between the robot and the obstacle, however, the threshold was exceeded by 0.01 meters (a 4% error), 573 seconds into the simulation. The performance of the system can be improved by implementing a velocity controller that significantly reduces the speed of the robot as the obstacle approaches, yet this may come at the cost of an operationally slow robot if sensor positioning and sensor noise are not carefully considered in the design. Another approach in resolving obstacle avoidance issues between robots is the use of negotiation protocols, hinging towards the implementation of a strongly cooperative solution.

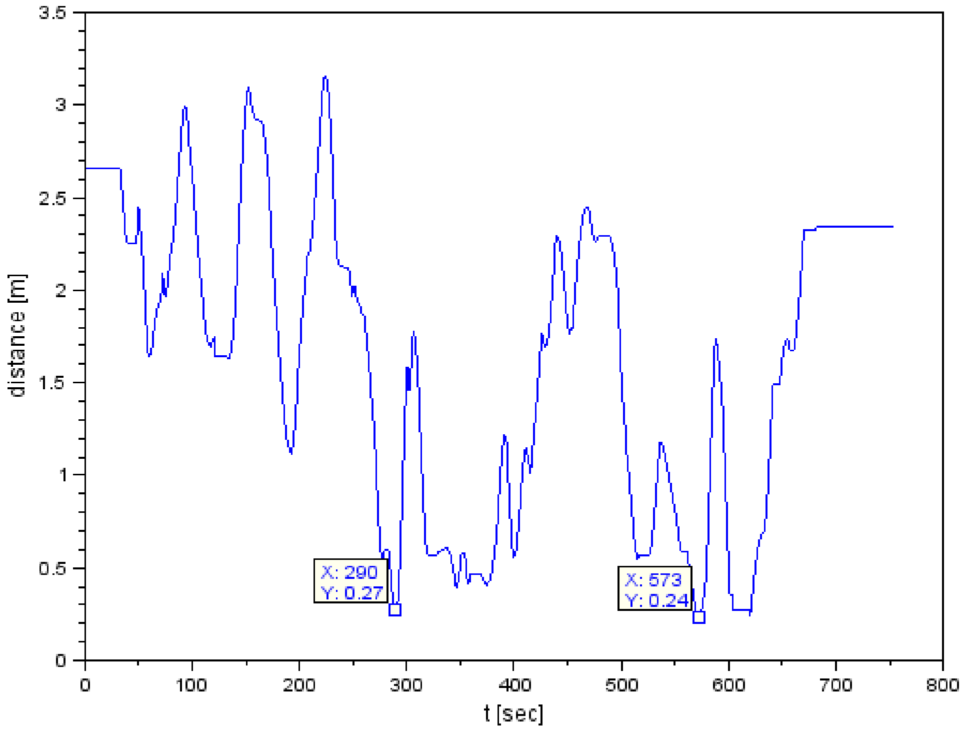

FIGURE 5.15: Obstacle avoidance simulation results between PeopleBot and RMP200

5.3.1.4 Cooperation during a robot fault

One of the objectives of this research is to test the robust ability of the system when a failure occurs with a robot. The results shown in figure 5.16 and in table 5.4 were obtained from the *cooperation during a robot fault* simulation. The operation time of the RMP200 was intentionally cut short to just 130 seconds out of the total simulation time of 844 seconds, allowing the PeopleBot to take over the task of moving material from B4 to B5. The system responded well to the robot failure since the production time increased by a minor 4% from the previous simulation case. Table 5.4 gives further information about the task distribution of the PeopleBot: a 50% split between primary and secondary tasks, this is expected due to the reduced work load performed by the RMP200, which had to be compensated by the PeopleBot.

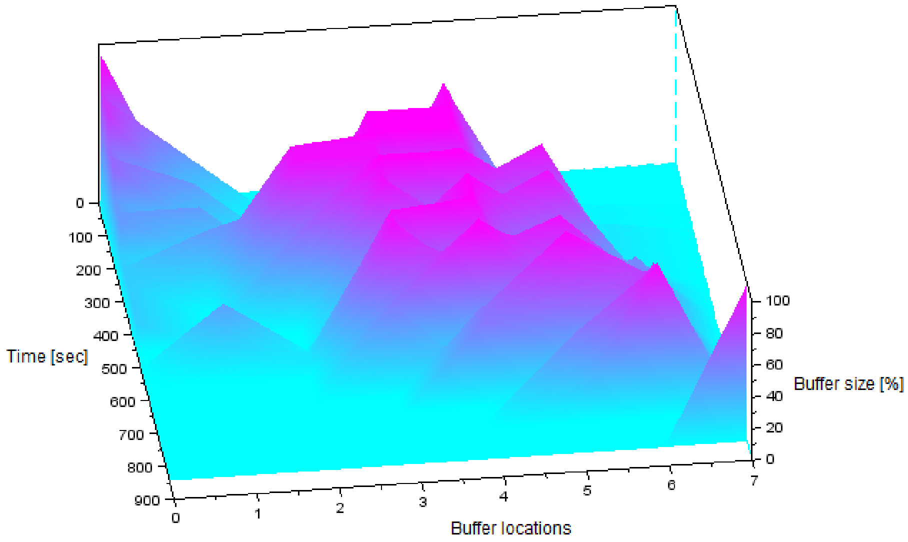

FIGURE 5.16: Material distribution graph: cooperation during robot fault

TABLE 5.4: Test parameter values during robot cooperation due to robot fault

Test parameter	**Value**
Total simulation time	844 sec
Agent 1 load capacity	20 materials
Agent 2 load capacity	5 materials
Agent 3 load capacity	100 materials
Agent 1 operation time	691 sec (81.9%)
Agent 1 primary task	50%
Agent 1 secondary task	50%
Agent 2 operation time	130 sec (15.4%)
Agent 3 operation time	93 sec (11.0%)
Buffer 3 @100%	90 sec (10.7%)
Buffer 4 @100%	252 sec (29.9%)

The SVM outputs for the PeopleBot and the buffer sizes for B0, B3 and B4 are shown in the two subplots of figure 5.17. The results are a similar representation of those discussed in figure 5.13, where the PeopleBot assists at the bottleneck and, in this case, takes over the task of the RMP200. The failure of the RMP200 in completing its task can actually be seen in the "B4 size" plot where the minor reductions in size (due to the small load carrying capacity of the RMP200) are stopped around 150 seconds into the simulation; thereafter the reductions are major due to the large load carrying capacity of the PeopleBot.

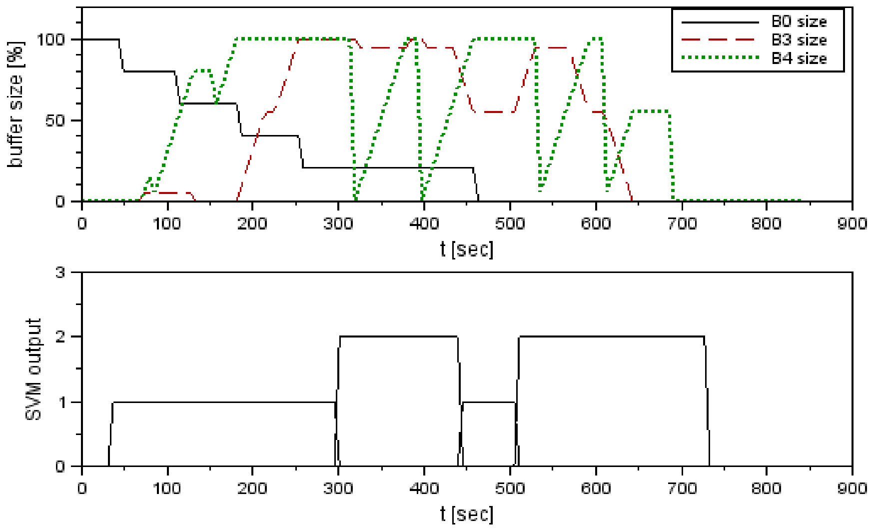

FIGURE 5.17: PeopleBot SVM outputs: cooperation during robot fault

5.3.2 Real–world performance

The only type of test performed in the real–world was *cooperation during a robot fault.* The other tests were not carried out to completion due to the inaccurate localisation calculation for the RMP200. The inaccuracy in the calculation was due to the accumulation of the odometry error (see figure 1.4), caused by the robot's wheel slippage. Attempts were made in software to compensate for the error, however none succeeded. The exercise proved that odometry is not a good method for long–term accuracy and further developments to this research will see the use of an advanced localisation technique with odometry as a short–term or secondary option in the calculation.

A video of the complete demonstration is included in the accompanying CD, which shows the operation of the RMP200 for approximately 4 minutes, thereafter intentionally removed from the test to create a robot failure. Figure 5.18 gives a sequence of screenshots that were taken from the video.

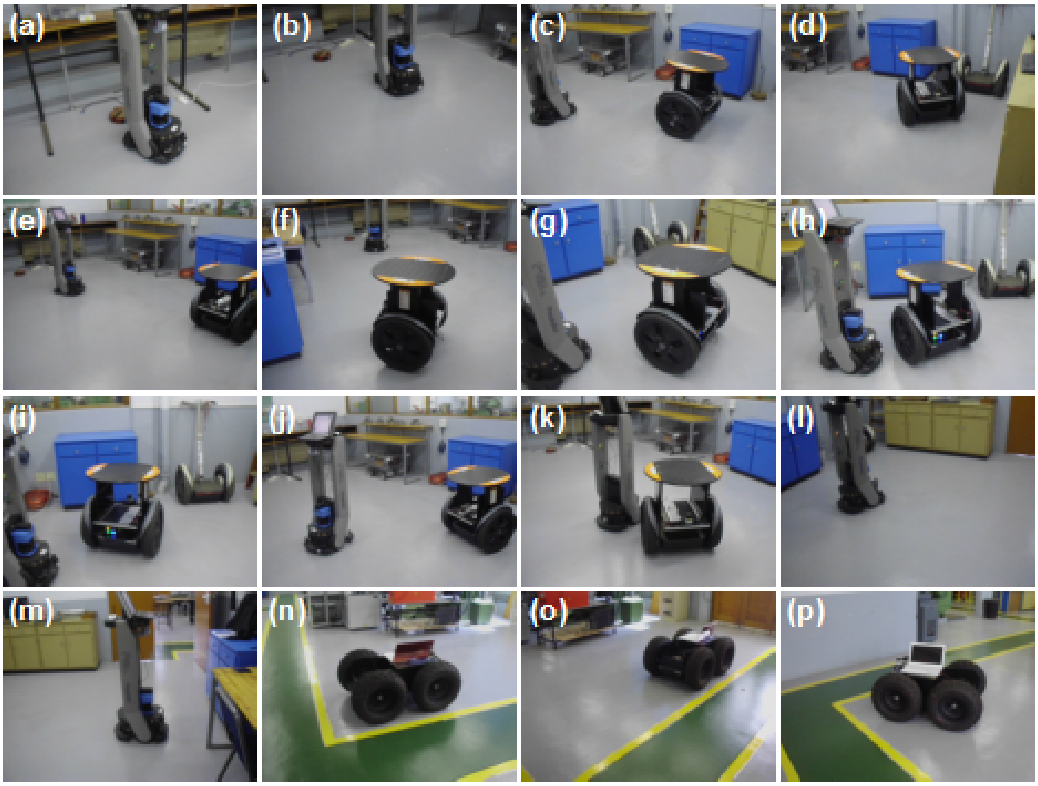

FIGURE 5.18: Screenshots from video of actual test

A description of each labeled picture in the figure is given by the following itemised list:

- *(a)*: the beginning of the test— the PeopleBot at its home position.
- *(b)*: PeopleBot at its primary source location (B0).
- *(c)*: PeopleBot moving away from its primary destination (B1) and the RMP200 at its home position.
- *(d)*: RMP200 at its source location (B4).
- *(e)*: PeopleBot at B1 and the RMP200 moving towards its destination location (B5).
- *(f)*: RMP200 at B5.
- *(g)*: PeopleBot assisting the RMP200 at B4.
- *(h)*: RMP200 moving to B4 and the PeopleBot going to B5.
- *(i)*: RMP200 at B4.

- *(j)*: both robots moving to B5.
- *(k)*: PeopleBot moving to B4 and RMP200 going to B5.
- *(l)*: PeopleBot at B4, completing the "failed" robot's task.
- *(m)*: PeopleBot at B5.
- *(n)*: RMP400 at its source location (B6).
- *(o)*: RMP400 at its destination location (B7).
- *(p)*: RMP400 at its home position— end of the test.

The results of the test are given in figure 5.19 and table 5.5. In comparison to the simulation results for the *cooperation during a robot fault* test, the real–world results improved the production completion time by 8%, this is attributed to the existence of various obstacles (desks and cupboards) in the real environment which assisted in the process of optimised solutions to the navigation algorithm; more on this matter is discussed in the next section. The task distribution of the PeopleBot, as shown in table 5.5, produced the same result as the simulation case: 50% for both primary and secondary tasks, however, the operation time of the PeopleBot is lower at 660 seconds, this is due to the increased work load performed by the RMP200.

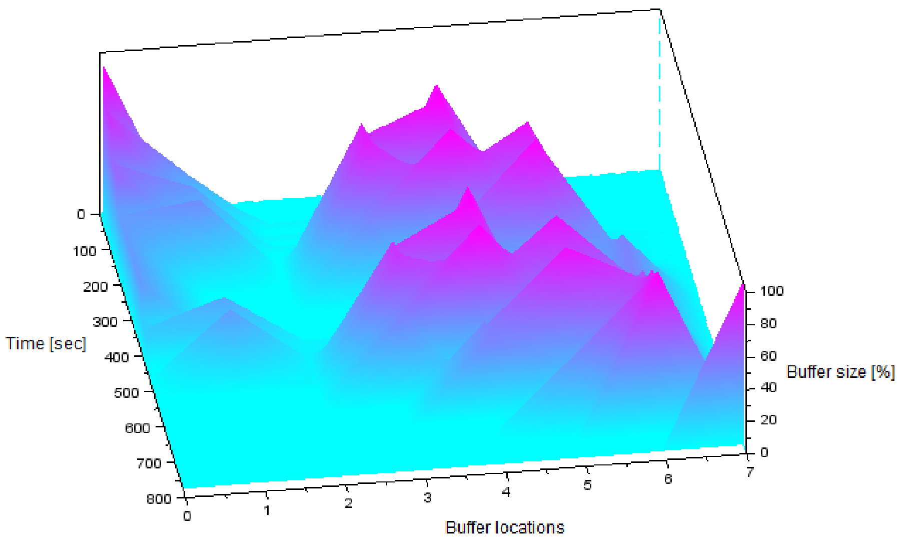

FIGURE 5.19: Material distribution graph: cooperation during robot fault (real–world)

TABLE 5.5: Real–world test parameter values

Test parameter	**Value**
Total simulation time	775 sec
Agent 1 load capacity	20 materials
Agent 2 load capacity	5 materials
Agent 3 load capacity	100 materials
Agent 1 operation time	660 sec (85.2%)
Agent 1 primary task	50%
Agent 1 secondary task	50%
Agent 2 operation time	222 sec (28.7%)
Agent 3 operation time	100 sec (12.9%)
Buffer 4 @100%	126 sec (16.3%)

The SVM outputs for the PeopleBot in the real–world test are given in figure 5.20. In comparison to figure 5.17, the SVM algorithm predicts an agent assist sooner (at approximately 200 seconds), and the evidence of the RMP200 in performing its task can be seen for a prolonged duration, up until 222 seconds into the actual test.

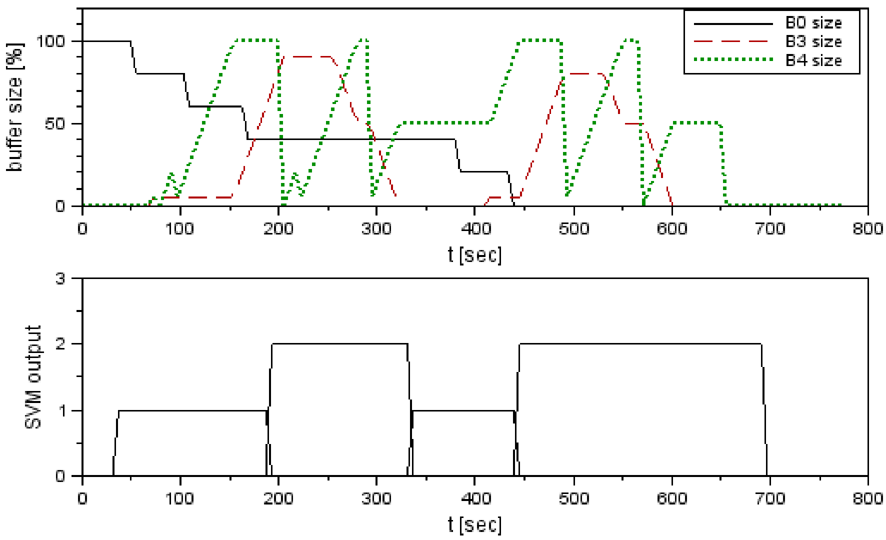

FIGURE 5.20: PeopleBot SVM outputs: cooperation during actual robot fault

5.4 Cognition results and discussion

The previous section discussed the results of the cooperative group performance of the robots; this section focuses on the test results of the individual robots, in particular, the navigation path and obstacle avoidance performance of each robot.

5.4.1 PeopleBot performance

The simulation and actual navigation path results for the PeopleBot (obtained during the *cooperation during a robot fault* test) are shown in figure 5.21. The figure also gives the x–y location of the robot's home position and the robot's source and destination buffer locations so that the navigation path between the locations can be identified.

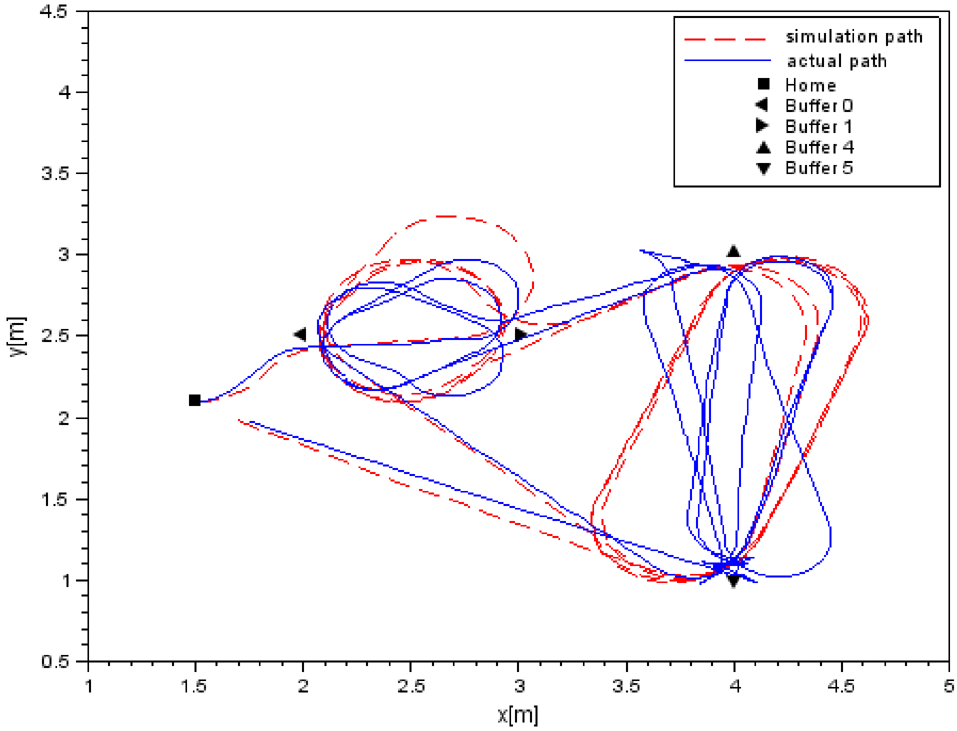

FIGURE 5.21: PeopleBot navigation in simulation and actual tests

In the analysis of the results, the following two observations were made:

1. The robot yaw–turn around path in the actual test is narrower, thus more efficient than the wide path taken by the robot in the simulation test, despite a configuration of the same angular velocity set points for the robot in both tests. The main reason for

these differences is due to the lack of obstacles designed in the simulation environment, hence creating more free–space for the navigation algorithm to use in the calculation of the way–points to the goal location. The obstacles in the actual environment are the desks and cupboards which "aid" the algorithm in determining the best way–point en-route to the goal. The second, and minor reason, can be attributed to an inaccurate Stage design model of the PeopleBot which will influence the accuracy of the angular speed.

2. Due to the goal tolerance set point of 0.3 meters, configured in the code, the actual and simulation results illustrate that the robot does not arrive at the exact position of its goal. Figure 5.14 showed the simulation results of the tolerance distances at each goal, whereas figure 5.22 gives the actual results of the tolerance distances, revealing the largest margin to be 0.28 meters from the goal, within the tolerance threshold of 0.3 meters.

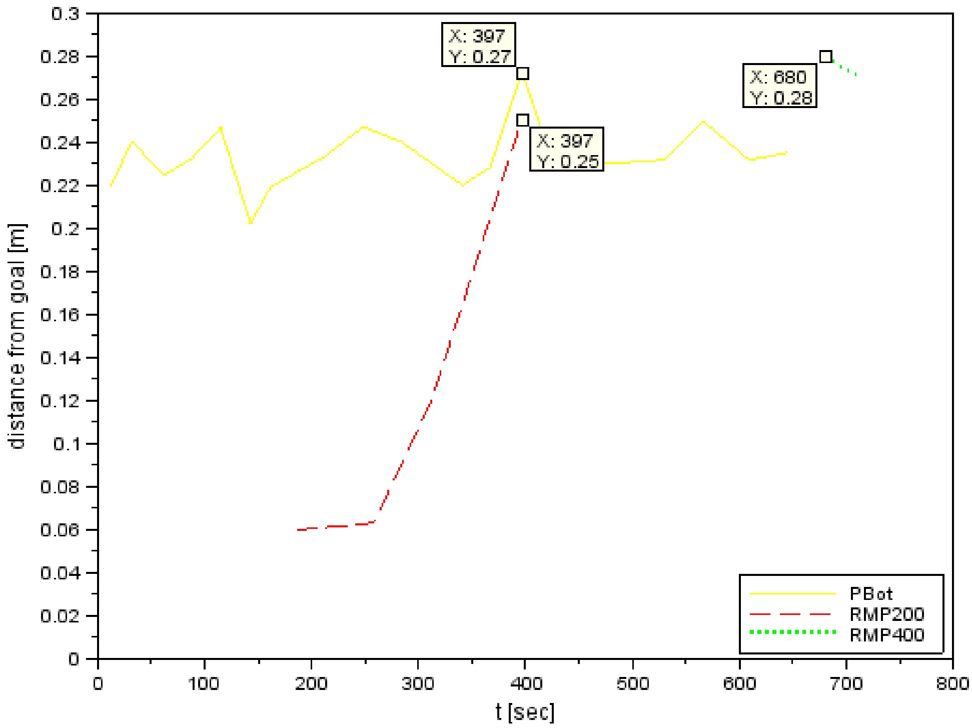

Figure 5.22: Robot goal tolerance actual results

Figure 5.23 shows a frame sequence (viewed from left to right) of the PeopleBot avoiding an obstacle in the actual environment. During the tests, there were cases when the robot moved in close towards the desk due to the inability of the sonar array in detecting the narrow legs of the desk.

FIGURE 5.23: Screenshots from video of the PeopleBot obstacle avoidance

The actual distance plot between the PeopleBot and the RMP200 is show in figure 5.24. The results reveal that the safe distance threshold of 0.25 meters was exceeded by 0.162 meters (a 64.8% error), 238 seconds into the actual test. A previous discussion on the simulation results of the plot mentioned the use of a velocity controller or negotiation protocols between the agents to improve the performance of the robots in an environment of "moving obstacles".

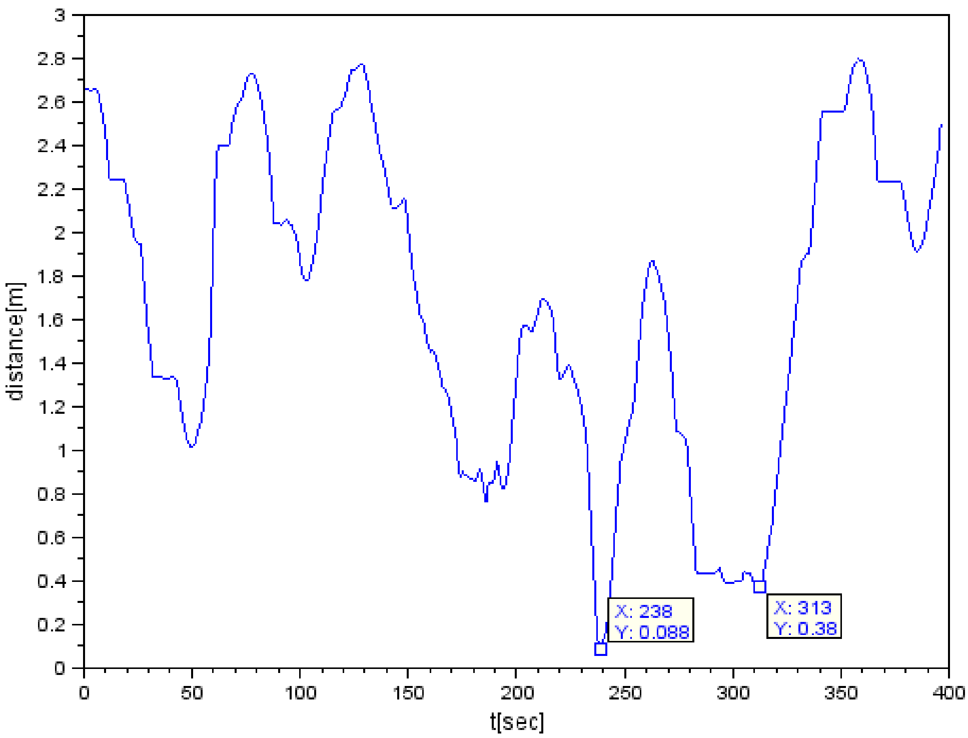

FIGURE 5.24: Obstacle avoidance actual results between PeopleBot and RMP200

5.4.2 RMP200 performance

The navigation path results of the RMP200 are depicted in figure 5.25. The results were obtained from the test where a failure was intentionally caused on the RMP200, thus the navigation paths are incomplete for the two tests. The simulation path is shorter than the actual path due to the robot operation time of 130 seconds (see table 5.4) and 222 seconds (see table 5.5) respectively. The irregularity of the actual navigation path (showed towards the lower half of figure 5.25) is due to the occurrence of the obstacle avoidance routine carried out by the RMP200 as it approaches the Peoplebot— these sequence of events are illustrated by the screenshots shown in figure 5.26.

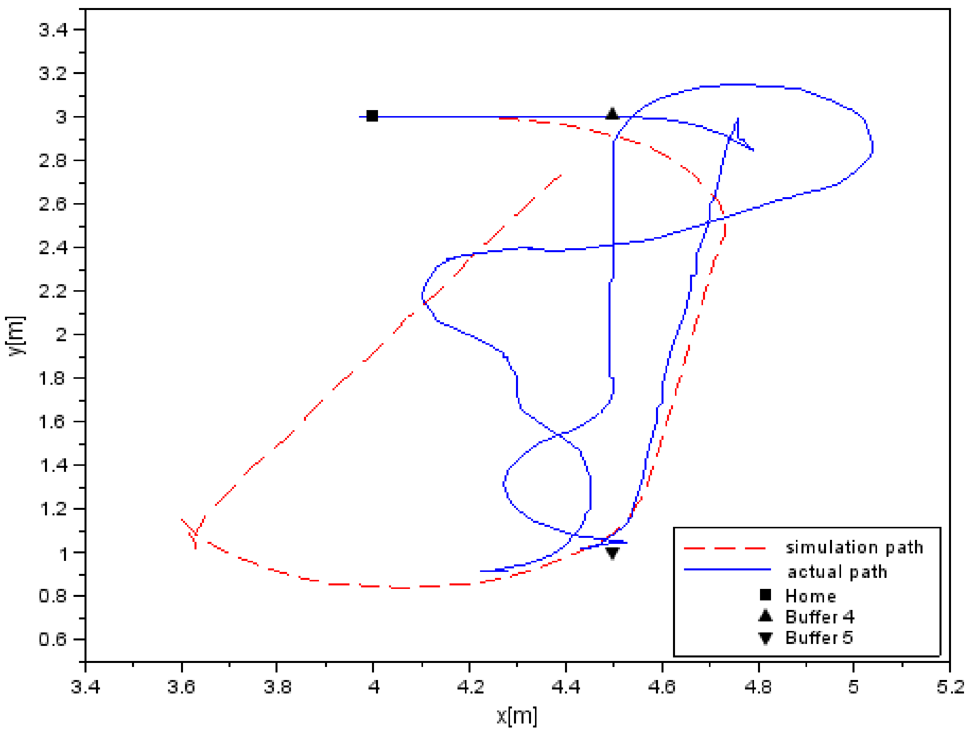

FIGURE 5.25: RMP200 navigation in simulation and actual tests

FIGURE 5.26: Screenshots from video of the RMP200 obstacle avoidance

5.4.3 RMP400 performance

The navigation path results of the RMP400 are illustrated in figure 5.27. A comparison of the navigation plots between the simulation and actual tests reveal the result of a difference in configuration of the maximum angular velocity limit set point. In the Stage simulation, the underlying RMP400 Player driver is an implementation of two RMP200 drivers, and the driver does not consider the RMP400's mechanical geometry when angular speed scaling factors are calculated; thus an angular speed limit set point in the simulation test does not yield the same result in the actual test. Figure 5.28 gives a sequence illustration of the turning portion of the actual path in figure 5.27.

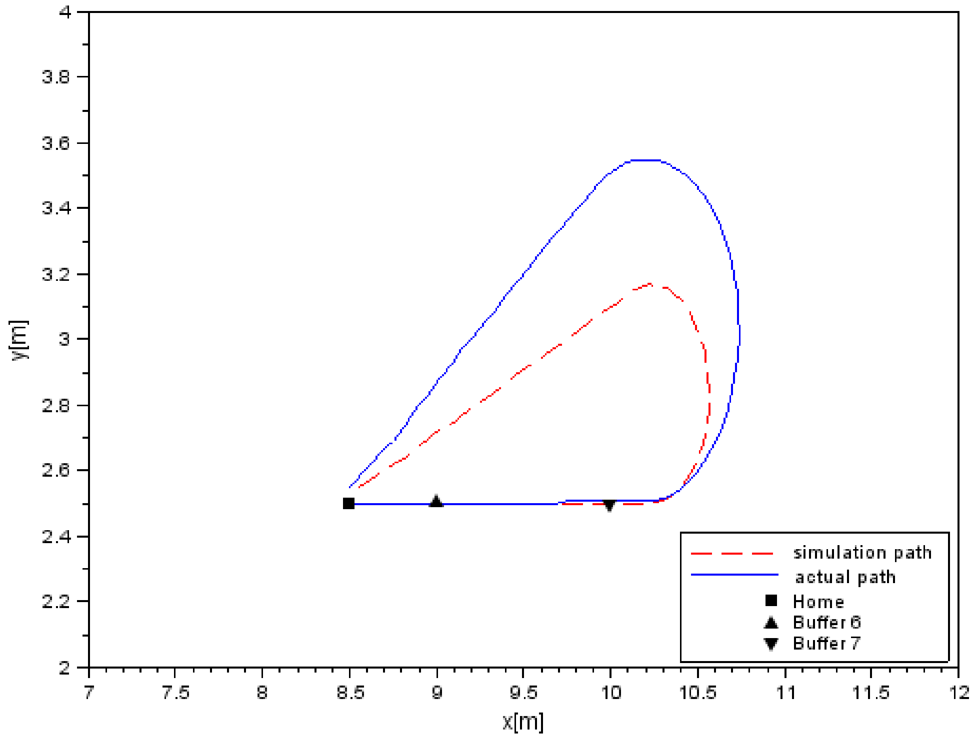

FIGURE 5.27: RMP400 navigation in simulation and actual tests

FIGURE 5.28: Screenshots from video of the RMP400 turning in a tight space

5.5 Chapter summary

This chapter discussed the layout of the simulation and real–world test environments where the buffer locations and home positions of the robots were shown. The actual robot assemblies that were used in the real–world tests were also reviewed. The major part of this chapter involved the discussion of the simulation and actual test results for robot cooperation performance and individual robot cognition performances.

Chapter 6

Conclusion and Further Research

This chapter concludes the dissertation by summarising the research work performed and includes a discussion on the achieved objectives outlined in section 1.6 . The contributions of this research are also mentioned and recommendations are given for further developments to the research topic.

6.1 Research conclusion

The motivation for this research topic stemmed around the problem of bottlenecks in advanced manufacturing environments where heterogeneous robots exist to perform specialised tasks. The need for a cooperative intelligent system in these environments was discussed. In this research, three heterogeneous mobile robots were used with the idea that each one had its primary task to perform and would only execute a secondary task if there was a need (i.e. a bottleneck) and if it had the ability to perform the secondary task.

The literature review and design phase of the research involved the analysis of strongly and weakly cooperative systems, together with the associated control architectures pertinent to robotic teams where four types of systems were discussed: centralised, decentralised, hierarchical, and hybrid (see section 1.4.6.1). The cooperation scheme that was chosen and implemented in the research was a weakly cooperative, hybrid one for reasons that were discussed in section 2.1.

The literature survey also addressed the purpose of the middleware layer in robotic networks; the Player server was implemented as the middleware layer in the AI system of the research and the Stage platform was used as the simulator to test and debug the software components. Another key element of the AI system is the SVM learning algorithm which essentially predicts and determines the goal tasks of each robot agent in the network by using a database of training examples.

The main objective of the research was the demonstration of a cooperative robot system in both simulated and real–world environments, validated against the key indicators outlined in section 1.6. This objective was achieved by the successful performance of the SVM algorithm (as discussed in chapter 5), where the bottlenecks were alleviated by the cooperating agent, significantly improving the manufacturing production times. The system was validated against the *robustness* key indicator since an intentional robot failure during the tests proved the ability of the system to continue operation and mitigate potential bottlenecks.

The cognition software module of the AI system in each robot agent allowed them to detect and avoid obstacles, thus preventing collisions and damages to the environment, establishing the *reliability* validation of the system. A shortcoming in this part of the research was the actual test results for the obstacle avoidance safety distance between the PeopleBot and the RMP200; the results did not exceed the safety distance design specification of 0.25 meters and it was discussed that a velocity controller or negotiation protocols between agents can be used to improve the performance of the system.

One of the design specifications of the research was the development of a GUI system that shows the manufacturing process and the associated data. This specification was achieved by the design and implementation of the Proview SCADA system which also fulfilled the *plant integration* requirement of the validation. The user friendly GUI permits system start up and SVM training of the robot agents and the Proview SCADA functionality allows for the integration of plant software packages (ERP and MRP) so that the data can be used at management levels.

6.2 Research contributions

The contributions of this research are vast and cover the following areas in industry and research:

- Productivity is the measure of the efficiency of a production system which is a direct implication on the time required to complete a job. The cooperation system discussed in this research has the potential to increase plant productivity due to the mitigation of bottlenecks in the production process where robotic tasks are concerned.
- An integration of the plant ERP system to the cooperation system can optimise delivery schedules and supply chain management processes by utilising the predictive data generated by the SVM algorithm of the AI. The cooperation system can be used in complex supply chain processes that comprise of multiple production lines and redundant work stations where the scheduling of (robotic) resources is critical.
- Search and rescue research can see the application of the cooperation system where the robots learn the dynamics of the environment and act accordingly, i.e. they either continue performing their "primary" task or cooperatively swarm towards a target in the environment.
- SVM learning has been widely used in the fields of bioinformatics and text/image recognition. The research discussed in this dissertation broadens the use of SVM algorithms (and potentially other supervised learning algorithms) in the area of multi–robot systems and manufacturing applications. The attraction of a learning based system is the semi–elimination of hard coded programmed solutions for specific scenarios; the learning system can adapt to dynamic environments and plant reconfiguration conditions.

6.3 Recommendations for further research

The localisation calculations for the robots in this research used odometry and, as a result, inherited the errors associated with odometry measurements. The performance of the actual system in this research will significantly improve if advanced localisation techniques are used such as Kalman filters and/or landmark detection methods.

Another recommendation for further research can be an inclusion of the load carrying efficiencies of the robots to the learning algorithm. The agents should determine the size of the task requirements at a work station since it may be inefficient to transport small loads (rather wait at the station for a longer period) or it may be unnecessary to operate at a capacity of 100% if the supply chain does not require fast production times. The ERP and supply chain management system should be incorporated into the cooperative learning system to optimise resources and establish a holistic solution to the manufacturing system.

The final recommendation is the implementation of a reinforced learning system where the agents dynamically learn the “positive” and “negative” examples from the environment without going through a training exercise facilitated by the robot operator. An addition to this recommendation is the use of an automated selection of a training database in a suite of databases, this is useful when an agent has to solve a variety of problems, requiring the employment of multiple sets of training data.

References

[1] Massachusetts Institute of Technology. Trends in advanced manufacturing, last accessed November 2014. URL http://web.mit.edu/deweck/Public/pie/TrendsinAdvancedManufacturingTechnologyResearch.pdf.

[2] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *In Proceedings of AI Magazine*, pages 9–20, AAAI, 2008.

[3] M. J. Mataric. New directions: Robotics: Coordination and learning in multirobot systems. *IEEE Intelligent Systems*, 13(2):6–8, IEEE, 1998.

[4] L. E. Parker. *Multiple Mobile Robot Systems. Chapter 40 of Springer Handbook of Robotics*. Springer-Verlag, Berlin Heidelburg, 2008.

[5] L. Butler. Autonomous materials handling robot for reconfigurable manufacturing systems. *In Proceedings of the 24th International Conference on CAD/CAM, Automation, Robotics, and Factories of the Future*, Koriyama, Japan, 2008.

[6] Pocobor. Mechatronics, last accessed November 2014. URL http://pocobor.com/blog/?m=200905.

[7] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, Cambridge, Massachusetts, 2004.

[8] J. Borenstein. Control and kinematic design of multi-degree-of-freedom robots with compliant linkage. *IEEE transactions on Robotics and Automation*, IEEE, 1995.

[9] J. Borenstein and L. Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE transactions on Robotics and Automation*, pages 869–880, IEEE, 1996.

[10] P. A. Bosscha. *Design and Construction of Meercat: An Autonomous Indoor and Outdoor Courier Service Robot*. University of KwaZulu-Natal, School of Mechanical Engineering, Durban, South Africa, 2011.

[11] S. Shoval and J. Borenstein. Measurement of angular position of a mobile robot using ultrasonic sensors. *Proceedings of the ANS Conference on Robotics and Remote Systems*, Pittsburgh, USA, 1999.

[12] H. Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Stanford, CA, Dept. of Computer Science, Stanford University, 1980.

[13] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. *In Proceedings of the International Conference in Computer Vision and Pattern Recognition*, pages 652–659, IEEE, Washington, DC, USA, 2004.

[14] F. Fraundorfer and D. Scaramuzza. Visual odometry, part ii: Matching, robustness, optimization, and applications. *IEEE Robotics and Automation Magazine*, pages 78–90, IEEE, 2012.

[15] M. Maimone, Y. Cheng, and L. Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, Wiley, 2007.

[16] V. Magori. Ultrasonic sensors in air. *IEEE Ultrasonics Symposium*, pages 471–481, IEEE, Cannes, France, 1994.

[17] M. C. Amann, T. Bosch, M. Lescure, R. Myllyla, and M. Rioux. Laser ranging: A critical review of usual techniques for distance measurement. *Optical Engineering*, 40(1):10–19, SPIE, 2001.

[18] K. Konolige, J. Augenbraun, N. Donaldson, and P. Shah. A low-cost laser distance sensor. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3002–3008, IEEE, Pasadena, CA, 2008.

[19] Neato Robotics, last accessed August 2013. URL http://www.neatorobotics.com/.

[20] G. Benet, F. Blanes, J. E. Simó, and P. Pérez. Using infrared sensors for distance measurement in mobile robots. *Robotics and Autonomous Systems 1006*, 40(4): 255–266, ELSEVIER, 2002.

[21] H.R. Everett. *Sensors for Mobile Robots: theory and application.* AK Peters, Ltd., Wellesley, MA, 1995.

[22] D. H. Titterton and J. L. Weston. *Strapdown inertial navigation technology.* The Institute of Electrical Engineers, 2nd Edition, United Kingdom, 2004.

[23] J. Vaganay, M. J. Aldon, and A. Fournier. Mobile robot attitude estimation by fusion of inertial data. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 277–282, IEEE, Atlanta, GA, 1993.

[24] B. Cho, W. Moon, W. Seo, and K. Baek. A dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding. *Journal of Mechanical and Science Technology*, 25(11):2907–2917, Springer, 2011.

[25] S. Thrun. *Robotic Mapping: A Survey.* Carnegie Mellon University, School of Computer Science, Pittsburgh, USA, 2002.

[26] J. Castellanos, J. Montiel, J. Neira, and J. Tardos. The spmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–952, IEEE, 1999.

[27] G. Dissanayake, H. Durrant-Whyte, and T. Bailey. A computationally efficient solution to the simultaneous localisation and map building (slam) problem. *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1009–1014, IEEE, San Francisco, CA, 2000.

[28] D. Fox. Kld-sampling: Adaptive particle filters and mobile robot localization. *Advances in Neural Information Processing Systems*, 14(1), MIT Press, 2001.

[29] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99–141, ELSEVIER, 2001.

[30] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 343–349, AAAI/IAAI, Orlando, Florida, USA, 1999.

[31] L. Zhang, R. Zapata, and P. Lépinay. Self-adaptive monte carlo localization for mobile robots using range finders. *Robotica*, 30(2):229–244, Cambridge University Press, 2012.

[32] J. Oommen, S. Iyengar, N. Rao, and R. Kashyap. Robot navigation in unknown terrains using visibility graphs: Part i: The disjoint convex obstacle case. *IEEE Journal on Robotics Automation*, 3(6):672–681, IEEE, 1987.

[33] H. Choset and J. Burdick. Sensor-based exploration: the hierarchical generalised voronoi graph. *International Journal of Robotics Research*, 19(2):96–125, SAGE, 2000.

[34] J. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwood, MA, 1991.

[35] D. Fu-guang, J. Peng, B. Xin-qian, and W. Hong-jian. Auv local path planning based on virtual potential field. *Mechatronics and Automation, IEEE International Conference*, 4:1711–1716, IEEE, Ontario, Canada, 2005.

[36] V. Lumelsky and T. Skewis. Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems Man and Cybernetics*, 20:1058–1068, IEEE, 1990.

[37] V. Lumelsky and A. Stepanov. Path-planning strategies for a point mobile automation amidst unknown obstacles of arbitrary shape. *Algorithmica*, pages 403–430, Springer, 1987.

[38] M. Khatib and R. Chatila. An extended potential field approach for mobile robot sensor-based motions. *In Proceedings of the Intelligent Autonomous Systems*, pages 490–496, IOS Press, Karlsruhe, Germany, 1995.

[39] J. Borenstein and Y. Koren. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Transaction on Robotics and Automation*, 7(3):278–288, IEEE, 1991.

[40] I. Ulrich and J. Borenstein. Vhf+: Reliable obstacle avoidance for fast mobile robots. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1572–1577, IEEE, Leuven, Belgium, 1998.

[41] A. Elkady and T. Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, Hindawi, 2012.

[42] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. A review of middleware for networked robots. *International Journal of Computer Science and Network Security*, 9(5): 139–148, IJCSNS, 2009.

[43] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. *In Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, pages 317–323, Coimbra, Portugal, 2003.

[44] About the Player Project, last accessed August 2013. URL http://playerstage.sourceforge.net/wiki/index.php/Main_Page.

[45] Y. Cao, A. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, Kluwer Academic Publishers Hingham, MA, USA, 1997.

[46] D. Barnes and J. Gray. Behaviour synthesis for co-operant mobile robot control. *International Conference on Control*, 2(1):1135–1140, IET, Edinburgh, 1991.

[47] M. Mataric. Interaction and intelligent behaviour, mit ai lab technical report, ai-tr-1495, August 1994.

[48] S. Premvuti and S. Yuta. Consideration on the cooperation of multiple autonomous mobile robots. *IEEE/RSJ IROS*, pages 59–63, IEEE, Ibaraki, Japan, 1990.

[49] E. Sahin and W. Spears. Swarm robotics, a state of the art survey. *Lecture notes in Computer Science*, 3342, 2005.

[50] L. E. Parker. Alliance: An architecture for fault tolerant multi robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, IEEE, 1998.

[51] L. E. Parker. An experiment in mobile robotic cooperation. *In Proceedings of the ASCE Specialty Conference on Robotics for Challenging Environments*, Albuquerque, NM, 1994.

[52] L. E. Parker. *Heterogeneous Multi-Robot Cooperation.* Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, USA, 1994.

[53] A. Jiménez-González, J. R Martínez de Dios, and A. Ollero. An integrated testbed for heterogeneous mobile robots and other cooperating objects. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3327–3332, IEEE, Taipei, Taiwan, 2010.

[54] R. Simmons, D. Apfelbaum, D. Fox, R. Goldman, K. Haigh, D. Musliner, M. Pelican, and S. Thrun. Coordinated deployment of multiple, heterogeneous robots. *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2254–2260, IEEE, Takamatsu, Japan, 2000.

[55] ActivMedia. Peoplebot, last accessed October 2014. URL www.mobilerobots.com/Libraries/Downloads/PeopleBot-PPLB-RevA.sflb.ashx.

[56] ActivMedia. Mobile robots, last accessed October 2014. URL http://www.mobilerobots.com/Mobile_Robots.aspx.

[57] Segway. About segway, last accessed October 2014. URL http://www.segway.com/about-segway.

[58] Segway. Segway robotics, last accessed October 2014. URL http://rmp.segway.com/.

[59] Segway. Rmp200, last accessed October 2014. URL http://rmp.segway.com/downloads/RMP_200_Specsheet.pdf.

[60] Segway. Rmp400, last accessed October 2014. URL http://rmp.segway.com/downloads/RMP_400_Specsheet.pdf.

[61] SICK. Lms200, last accessed October 2014. URL http://sicktoolbox.sourceforge.net/docs/sick-lms-technical-description.pdf.

[62] Hokuyo. Urg-04lx, last accessed October 2014. URL http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_spec_en.pdf.

[63] Netram Technologies. 5v power supply, last accessed October 2014. URL http://netram.co.za/938-breadboard-power-supply-5v-33v.html.

[64] Player Project. Amcl driver, last accessed October 2014. URL http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__driver__amcl.html.

[65] S. Bhattacharya and S. Talapatra. Robot motion planning using neural networks: A modified approach. *International Journal of Lateral Computing*, 2(1):9–13, World Federation on Lateral Computing, 2005.

[66] N. Naidoo, G. Bright, and R. Stopforth. Material flow optimisation in flexible manufacturing systems. *In Proceedings of the 6th IEEE Robotics and Mechatronics Conference (RobMech)*, pages 1–5, IEEE, Durban, South Africa, 2013.

[67] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer–Verlag, New York, 2000.

[68] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. *In Proceedings of the 5th annual workshop on Computational learning theory*, pages 144–152, ACM, New York, USA, 1992.

[69] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20(3): 273–297, Kluwer Academic Publishers, Boston, 1995.

[70] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[71] C. Hsu and C. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, IEEE, 2002.

[72] K. Crammer and Y. Singer. Algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(1):265–292, ACM, 2002.

[73] C-C. Chang and C-J. Lin. Libsvm: a library for support vector machines. last accessed October 2014. URL http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf.

[74] T. Joachims. *Making large-scale support vector machine learning practical: Advances in Kernel Methods*. MIT Press, 1998.

[75] Proview. About proview, last accessed October 2014. URL http://www.proview.se/.

[76] Mathisfun. Circle sector and segment, last accessed October 2014. URL http://www.mathsisfun.com/geometry/circle-sector-segment.html.

Appendix A

Robot Agent Code

A.1 Agent simulation program: Player .cfg file

```
# load the Stage plugin simulation driver
driver
(
  name "stage"
  provides [ "simulation:0" ]
  plugin "stageplugin"

  # load the named file into the simulator
  worldfile "sim.world"
)

# Create a Stage driver and attach position2d and laser interfaces
# to the model "r0"
driver
(
  name "stage"
  provides [ "position2d:0" "laser:0" "graphics2d:0" "graphics3d:0"]
  model "r0"
)

# Create a Stage driver and attach position2d and laser interfaces
# to the model "r1"
driver
(
  name "stage"
  provides [ "position2d:1" "laser:1" "graphics2d:1" "graphics3d:1"]
  model "r1"
)

# Create a Stage driver and attach position2d and laser interfaces
# to the model "r2"
driver
(
  name "stage"
  provides [ "position2d:2" "laser:2" "graphics2d:2" "graphics3d:2"]
  model "r2"
)
```

A.2 Agent simulation program: Player .world file

```
include "pioneer.inc"
include "segwayrmp.inc"
include "map.inc"
include "hokuyo.inc"

# time to pause (in GUI mode) or quit (in headless mode (-g)) the
    simulation
quit_time 3600 # 1 hour of simulated time

paused 1

resolution 0.02

# configure the GUI window
window
(
  size [ 435.000 466.000 ] # in pixels
  scale 43.481   # pixels per meter
  center [ 4  2 ]
  rotate [ 0  0 ]

  show_data 1              # 1=on 0=off
)

# load an environment bitmap
floorplan
(
  name "cave"
  size [9.000 4.500 0.800]
  pose [4.5 2.25 0 0]
  bitmap "material_app9.png"
)

define product0 model
(
  #picture of a black circle
  bitmap "circle.png"
  size [0.35 0.5 0.10]
  pose [-0.15 0 0 0]
  color "red"
)

define product1 model
(
  #picture of a black circle
  bitmap "circle.png"
  size [0.35 0.35 0.10]
  pose [-0.07 0 0 0]
  color "yellow"
)

define product2 model
(
  #picture of a black circle
  bitmap "circle.png"
  size [0.60 0.50 0.10]
  pose [-0.3 0 0 0]
  color "green"
)
```

```

rmp200
(
  # can refer to the robot by this name
  name "r0"
  pose [ 1 2 0 0]
  color "red"
  hokuyolaser()
  product0()
)

pioneer2dx
(
  # can refer to the robot by this name
  name "r1"
  pose [ 3 2 0 0]
  color "yellow"
  hokuyolaser()
  product1()
)

rmp400
(
  # can refer to the robot by this name
  name "r2"
  pose [ 5 3 0 0]
  color "green"
  hokuyolaser()
  product2()
)
```

A.3 Agent simulation program: config.h file

```
/*
Filename: config.h
Author: N. Naidoo
Date: 20/09/2014
Revision: 4e
Description: Header configuration file for the PeopleBot
*/

#define a_id 1

#define MAXLINE 4096 //max text line length
#define SERV_PORT 3000 //port

#define Dnum 120 //number of elements in data training set
#define dthresh 0.3 //threshold distance between robot pose and goal
#define dnear 2.0 //waiting distance between robot and goal
#define RobTurn 1.0 //0 degrees per second
#define RobSpeed 0.3
#define htime 10 //no. of secs to wait at home before checking for next
    goal

//gap-nav
#define estopDist 0.3
#define minObsDist 1.0
```

```
#define maxObsDist 2.0
#define safeDist 0.30
#define goaltol 0.3
#define angtol 0.10
#define vmin 0.0
#define vmax 0.10 //0.1
#define wmin 0.0
#define wmax 0.10
#define ewaitsp 3 //wait time setpoint in estop
#define revsp 2 //robot reverse time setpoint
#define fwdsp 0 //robot forward time setpoint
#define wayDist 1.0 //waypoint distance
#define rangespan 240 //240 degrees
#define pitchang 1 //0.36 pitch angle of laser in degrees
#define angoffset -30 //offset start angle in scan [degrees]
#define secnum 8 //number of sectors in scan
#define Rw 0.4 //robot width
#define RL 0.4 //robot length
#define navfreq 5 //navigation frequency in seconds
#define sectmask 126 //binary mask for estop sectors
#define rfac 0.1
#define pfac 0.70 //probability factor

//svm
#define learnc 1
#define trnpredc 2

//robot home positions
#define hx 3.0;
#define hy 2.0;
#define hyaw 0.0;

//buffer robot goal locations
#define Lnum 3 //number of possible locations
#define Lmask 2 //location bit mask (e.g 7->all 3 locations)
#define s1x 1.0
#define s1y 4.0
#define d1x 2.0
#define d1y 1.0
#define s2x 4.0
#define s2y 1.0
#define d2x 4.0
#define d2y 3.0
#define s3x 5.0
#define s3y 2.0
#define d3x 8.0
#define d3y 1.0

//agent modes
#define wakeup 0
#define gohome 1
#define athome 2
#define gosrc 3
#define atsrc 4
#define godest 5
#define atdest 6
```

A.4 Agent simulation program: Client file

```
/*
Filename: Pbotsim.cc
Author: N. Naidoo
Date: 20/09/2014
Revision: 4e
Description: Client application program for the PeopleBot
*/
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <utilities.h>
#include <config.h>
#include <string.h>
#include <libplayerc++/playerc++.h>
#include <ctype.h>
#include <errno.h>
#include "svm.h"
#include "svm-train.h"
#include "svm-predict.h"
#include "agent.h"
//#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

//global variables
double xg, yg, yawg; //goal locations
char str1[50];
int amode, agoal, aid, apass; //amode is the state number
int svmode, tout, ago, aclr; //go if ago=1, else stop
int imode;
char sendline[MAXLINE], recvline[MAXLINE], inbuf[MAXLINE];
int sockfd;
bool goalcmd=false;
struct sockaddr_in servaddr;

bool opensocket()
{
  bool tempb=true;
  //Create a socket for the client
  //If sockfd<0 there was an error in the creation of the socket
  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) <0)
  {
    perror("Problem in creating the socket");
    tempb=false;
  }
  //Creation of the socket
  memset(&servaddr, 0, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr= inet_addr("192.168.43.171"); //192.168.43.125
  servaddr.sin_port =  htons(SERV_PORT); //convert to big-endian order
  //Connection of the client to the socket
  if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr))<0)
  {
    perror("Problem in connecting to the server");
    tempb=false;
```

```
  }
  if (tempb==true) return true;
  else return false;
}

void updatepkt() //update send packet to server
{
  sprintf(str1,"%s%d%s%d%s%d%s%d%s","#client;aid=",aid,";amode=",amode,";agoal=",agoal,";apass=",apass,";end#");
  //clear the send and receive buffers:
  memset(&sendline[0], 0, sizeof(sendline));
  memset(&recvline[0], 0, sizeof(recvline));
  memset(&inbuf[0], 0, sizeof(inbuf));
  //set the send buffer:
  snprintf(sendline,MAXLINE,"%s",str1);
}

int getaid() //get agent id
{
  char *parstr="aid";
  snprintf(recvline,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
  int tempi=GetParam(recvline,parstr);
  std::cout << "aid: " << tempi << std::endl;
  if (tempi==a_id)
  {
    return (1); //success
  }
  else
  {
    return (0); //failure
  }
}

int getsvm() //get svm mode
{
  char *parstr="svm";
  snprintf(recvline,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
  int tempi=GetParam(recvline,parstr);
  std::cout << "svmode: " << tempi << std::endl;
  if ((tempi>=0)&&(tempi<=2)) //check svmode
  {
    svmode=tempi;
    return (1); //success
  }
  else
  {
    return (0); //failure
  }
}

int getago() //get ago
{
  char *parstr="ago";
  snprintf(recvline,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
  int tempi=GetParam(recvline,parstr);
  std::cout << "ago: " << tempi << std::endl;
  if ((tempi==0)||(tempi==1)) //check ago
  {
    ago=tempi;
    return (1); //success
  }
  else
```

```
  {
    return (0); //failure
  }
}

int getaclr() //get aclr
{
  char *parstr="aclr";
  snprintf(recvline,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
  int tempi=GetParam(recvline,parstr);
  std::cout << "aclr: " << tempi << std::endl;
  if ((tempi==0)||(tempi==1)) //check aclr
  {
    aclr=tempi;
    return (1); //success
  }
  else
  {
    return (0); //failure
  }
}

int gettout() //get tout
{
  char *parstr="tout";
  snprintf(recvline,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
  int tempi=GetParam(recvline,parstr);
  std::cout << "tout: " << tempi << std::endl;
  if (tempi>0) //check tout
  {
    tout=tempi;
    return (1); //success
  }
  else
  {
    return (0); //failure
  }
}

int getbufdat() //get buffer data
{
  char *parstr="data";
  snprintf(recvline,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
  GetData(recvline,parstr);//storedata[0] should have the location now!
  int tempi=atoi(&storedata[0]);
  std::cout << "target: " << tempi << std::endl;
  if ((tempi>=0)&&(tempi<=Lnum)) //check location number
  {
    agoal=tempi; //set agent goal
    return (1); //success
  }
  else
  {
    return (0); //failure
  }
}

//Set x-y goal:
void SetGoal(double x1, double y1, double yaw1)
{
  xg=x1;
  yg=y1;
```

```
  yawg=yaw1;
  goalcmd=true;
}

//******************begin main program*********************
int main(int argc, char *argv[])
{
  imode=atoi(argv[argc-1]); //0:gapnav only; any other number:gapnav and
    svm/server comms
  std::cout << "imode: " << imode << std::endl;
  if (argc==2)//&&((imode>=learnc)&&(imode<=trnpredc)))
  {
    using namespace PlayerCc;
    PlayerClient    robot("localhost");
    SimulationProxy simproxy(&robot);
    Position2dProxy pp0(&robot,1);
    //RangerProxy rp(&robot,0);
    LaserProxy lp(&robot,1);

    pp0.SetMotorEnable(true);

    //variable list
    int elapsec, prevsec, tsec;
    int waitime; //wait time in seconds
    int waitcount; //wait time counter
    //int svmloc; //goal location for agent determined by svm algorithm
    double xc, yc, yawc; //current x-y and yaw locations of agent
    double Home[2]; //x-y locations of home position
    double Src[3][2]; //x-y locations of source positions
    double Dest[3][2]; //x-y locations of destination positions
    bool wait=false; //true if robot must wait before proceeding to next
    state
    bool svmflag; //=1 if we have an output from svm algorithm
    bool svmready; //=1 if ready to run svm algorithm
    bool servercom, serverok; //servercom:initiate comms with server;
    serverok:good packet reply from server
    bool sendflag; //=1 when we want to send packets to the server
    time_t start = time(0);
    //gap-nav variables:
    double sensdata[100]; //integrated sensor data, 8 sectors
    double ThetaArr[secnum][2]; //chosen theta ranges for routing: theta1,
    theta2
    double v=vmin;
    double w=wmin;
    double vin, win; //linear and angular velocity user inputs
    double xw, yw; //x-y waypoints
    double yawn; //normalised yaw
    double dgoal; //distance to goal
    double rdist=maxObsDist;
    double wdelta, gdelta; //waypoint and goal angle diff
    double xt,yt,theta, thetat; //x&y temp vars
    double xmin,ymin;
    double wdir; //w direction
    double wgap; //w for moving in tight spaces
    double arealim;
    double r1,r2;
    int lasercount; //laser scan number
    int cellnum;
    int navtimer=0; //navigation timer
    int stopnum=0; //number of estops
    int revnum=0; //number of reverses
    int stnum=1; //state machine number
```

```
    bool estop=false;
    bool revrobot=false;
    bool oneshot=true;
    bool turning=false;
    bool navrun=false;
    bool slowdown=false; //slow down robot flag
    bool rtight=false; //tight space flag
    bool obsavoid=false; //obstacle avoidance flag
    bool waycalc=false; //waypoint calc done flag
    bool frontclr=false; //robot front clear to move
    uint ewaitsec=0;
    uint revsec=0;
    uint fwdsec=0;
    long totsec=0; //total no. of seconds since start
    //end variable list

    //Initialisations
    //initialise sample counter
    tsec=0;
    aid=a_id;
    amode=0; //wake up
    agoal=0; //home
    ago=0; //not ready to go
    aclr=0; //not clear to proceed
    apass=0; //not waiting to go to buffer source/destination
    waitime=htime;
    waitcount=0;
    //svmloc=0; //home location
    svmflag=false; //we dont have an output from svm algorithm
    svmready=false; //not ready to run svm algorithm
    servercom=false; //not ready to communicate with server
    serverok=false; //no good reply yet from server
    sendflag=false; //dont send data to server
    Home[0]=hx;
    Home[1]=hy;
    Src[0][0]=s1x;
    Src[0][1]=s1y;
    Src[1][0]=s2x;
    Src[1][1]=s2y;
    Src[2][0]=s3x;
    Src[2][1]=s3y;
    Dest[0][0]=d1x;
    Dest[0][1]=d1y;
    Dest[1][0]=d2x;
    Dest[1][1]=d2y;
    Dest[2][0]=d3x;
    Dest[2][1]=d3y;
    updatepkt(); //initialise send buffer packet
    if (imode==0)
    {
      xg=d2x; //Home[0];
      yg=d2y; //Home[1];
      goalcmd=true;
    }
    // end initialisations

    while(1) //main loop: gapnav and/or svm
    {
      //get the elapsed seconds from start
      elapsec = difftime(time(0),start);
      if (elapsec != prevsec)
      {
```

```
        //tick = true;
        if (imode!=0) //svm-server
        {
          if (wait==true)
          {
              waitcount++;
              if (waitcount>=tout) wait=false; //not waiting any longer
          }
          else waitcount=0;
        }
        if (ewaitsec>0) ewaitsec--; //decrement estop timer if neccessary
        if (revsec>0) revsec--; //decrement reverse timer if neccessary
        if (fwdsec>0) fwdsec--; //decrement forward timer if neccessary
        navtimer++; //increment navigation timer

        if (wait==true) std::cout << "waiting!!!" << std::endl;

        totsec++; //increment total seconds
        //store x-y locations:
        xystore(totsec,xc,yc);

        std::cout << "x,y,yaw,yawn: " << xc << "," << yc << ", " << yawc << ", " << yawn << std::endl;
        std::cout << "rdist,v,w: " << rdist << ", " << v << ", " << w << std::endl;
        std::cout << "xg,yg: " << xg << "," << yg << std::endl;
        std::cout << "xw,yw: " << xw << "," << yw << std::endl;
        std::cout << "r1,r2: " << r1 << "," << r2 << std::endl;
        std::cout << "xmin,ymin,Area Lim: " << xmin << "," << ymin << ", " << arealim << std::endl;
        std::cout << "gdelta,theta,dgoal: " << gdelta << ", " <<theta << ", " << dgoal << std::endl;
        std::cout << "stnum: " << stnum << std::endl;
        for (int i=0; i<secnum; i++)
        {
          std::cout << "Sector Area " << i << ": " << sensdata[i] << std::endl;
        }
        for (int i=0; i<lasercount; i++)
        {
          //std::cout << "LasArr: " << i << ": " << lp[i] << std::endl;
        }
        if (estop) std::cout << "Robot in E-STOP!!!" << std::endl;
        prevsec = elapsec;
      }
      //else tick = false;

      robot.Read(); //read from the proxies
      xc=pp0.GetXPos()+hx; //get current x-location of robot
      yc=pp0.GetYPos()+hy; //get current y-location of robot
      yawc=pp0.GetYaw()+hyaw;

      //***********************Begin GapNav************************
      if ((lp.IsValid()==true) && (goalcmd==true))
      {
        //robot.Read(); //read from the proxies
        //normalise yawc to [0;2pi]
        if (yawc<0) yawn = (2*M_PI) + yawc;
        else yawn = yawc;

        lasercount = lp.GetCount();

```

```
        //clear sensor data
        for (int i=0; i<secnum; i++)
        {
          sensdata[i] = 0;
        }

        //get xmin and ymin:
        xmin=100.0;
        ymin=100.0;
        for (int i=0; i<lasercount; i++)
        {
          double angt=DTOR(angoffset)+DTOR(i*pitchang);
          if ((angt>(0.3*M_PI))&&(angt<(0.8*M_PI))) xt=lp[i]; //xt=fabs(lp[i]*sin(angt));
          if ( ((angt>=0)&&(angt<=(0.3*M_PI))) || ((angt>=(0.8*M_PI))&&(angt<=M_PI)) ) yt=lp[i]; //yt=fabs(lp[i]*cos(angt));
          if (xt<xmin) xmin=xt;
          if (yt<ymin) ymin=yt;
        }

        //check estop condition
        if (((xmin<estopDist)||(ymin<estopDist))&&(estop==false)&&(revrobot==false))
        {
          estop=true;
          obsavoid=true; //handle estop in obsavoid routine
          ewaitsec=ewaitsp;
          stopnum++; //increment number of stops
        }

        //check if robot must slow down
        if ((xmin>estopDist)&&(xmin<minObsDist)) slowdown=true;
        else slowdown=false;

        //check if robot must wait
        if (wait==true) obsavoid=true; //handle estop in obsavoid routine

        //integrate laser data
        int k=1;
        double lpt; //temp laser var
        for (int i=0; i<lasercount; i++)
        {
          if (lp[i]>rdist) lpt=rdist;
          else lpt=lp[i];
          //lpt=lp[i];
          sensdata[k-1] += (DTOR(pitchang/2.0))*sqrd(lpt); //area=(theta/2)*R^2.
          if (i== int((lasercount/secnum)*k) ) k++;
        } //end for

        //check if robot front clear:
        arealim=(DTOR((rangespan/secnum)/2))*sqrd(maxObsDist); //Area=(theta/2)*R^2
        if (((sensdata[secnum/2])>(pfac*arealim))&&((sensdata[(secnum/2)-1])>(pfac*arealim))) frontclr=true;
        else frontclr=false;

        //obstacle avoidance routine
        //---------------------------------
        if (obsavoid==true)
        {
          //turning:
```

```
        if ((turning==true)&&(estop==false)&&(revrobot==false)) //turning required?
        {
          wdelta=AngDiff(xw,xc,yw,yc,yawc); //get angle diff between robot and waypoint
          wdir=GetDirn(wdelta); //get dirn to turn cw or acw
          //check if robot in tight space and calc wgap:
          if ( (xmin<(1*Rw)) || (ymin<(1*Rw)) )
          {
            //begin iteration to find r1,wgap:
            bool wexit=false;
            r1=xmin; //-estopDist;
            while ((r1>0)&&(wexit==false))
            {
              r1-=0.05;
              r2=sqrt(sqrd(r1)+sqrd(Rw));
              if ((r2-r1)<ymin)
              {
                wgap=(0.5*vmax)/r1;
                if (wgap>wmax) wgap=wmax;
                wexit=true;
              }
            }
            if (r1<=0) wgap=wmin;
            w=wdir*wgap;
          }
          else w=wdir*0.5*wmax; //robot not in tight space
          v=0.5*vmax;
          if (fabs(wdelta)<=angtol) turning=false;
        }
        else //turning==false or estop==true
        {
          if (revrobot==false)
          {
            //stop robot:
            v=vmin;
            w=wmin;
          }
        }
        //slowdown:
        //if (slowdown==true) v=0.5*vmax; //halve the speed
        //else v=vmax;

        //estop:
        if (estop==true)
        {
          if ((revrobot==false)&&(stopnum>1))
          {
            revrobot=true;
            revsec=revsp; //start reverse timer
            estop=false;
            v=-1*vmax;
            w=wmin;
          }
          else if (ewaitsec<=0) estop=false;//clear estop flag
        }
        //reverse:
        if (revrobot==true)
        {
          if ((revsec<=0)&&(stopnum>0))
          {
            revrobot=false;
```

```
                stopnum=0;
                //stop robot:
                v=vmin;
                w=wmin;
              }
            }
            //waiting:
            if (wait==true)
            {
              v=vmin;
              w=wmin;
            }

            if ((turning==false)&&(estop==false)&&(revrobot==false)&&(wait==
    false)) obsavoid=false; //exit routine
          } //end if (obsavoid==true)

          //State#1: Position robot towards goal
          //------------------------------------
          if (stnum==1)
          {
            gdelta=AngDiff(xg,xc,yg,yc,yawc);
            if ((fabs(gdelta)>angtol)&&(obsavoid==false)) //need to turn
    towards goal
            {
              xw=xg;
              yw=yg;
              turning=true;
              obsavoid=true;
            }
            if (fabs(gdelta)<=angtol) //in direction of goal - positioning
    almost done
            {
              //wait for obsavoid routine to complete
              if (obsavoid==false) stnum=2; //positioning done
            }
          }

          //State#2: Positioning done
          //-------------------------
          if (stnum==2)
          {
            if (obsavoid==false)
            {
              if (frontclr==true) stnum=4; //clear path to goal
              else stnum=3; //ready to plan path
            }
          }

          //State#3: Plan path using gaps
          //-----------------------------
          if (stnum==3)
          {
            dgoal = dcalc(xg,xc,yg,yc);
            gdelta=AngDiff(xg,xc,yg,yc,yawc);
            if (obsavoid==false)
            {
              if ( (dgoal>goaltol) && (dgoal<(4*goaltol)) ) //are we close to
     goal?
              {
                if (fabs(gdelta)<=angtol) stnum=4; //clear path to goal
                else stnum=1; //positon robot towards goal
```

```
            }
            else
            {
              if ((dgoal<=goaltol)&&(fabs(gdelta)<=angtol)) stnum=5; //goal reached
              if ((dgoal<=goaltol)&&(fabs(gdelta)>angtol)) stnum=1; //positon robot towards goal
              else //far away from goal - plan paths
              {
                if ((waycalc==true)&&(navtimer<navfreq))
                {
                  wdelta=AngDiff(xw,xc,yw,yc,yawc);
                  if ((fabs(wdelta)>angtol)&&(obsavoid==false)) //need to turn towards waypoint
                  {
                    turning=true;
                    obsavoid=true;
                  }
                  if ((fabs(wdelta)<=angtol)&&(obsavoid==false)) //travel straight towards waypoint
                  {
                    w=wmin;
                    v=vmax;
                  }
                }
                else
                {
                  if (navtimer>=navfreq)
                  {
                    waycalc=false;
                    navrun=true; //calculate waypoints in nav routine
                    navtimer=0; //reset timer for next iteration
                  }
                }
              }
            }
          }
        }

        //State#4: Clear path to goal
        //---------------------------
        if (stnum==4)
        {
          dgoal = dcalc(xg,xc,yg,yc);
          gdelta = AngDiff(xg,xc,yg,yc,yawc);
          if (obsavoid==false)
          {
            w=wmin;
            v=vmax;
            if ((dgoal<=goaltol)&&(fabs(gdelta)<=angtol)) stnum=5; //goal reached
            if ((frontclr==false)||(fabs(gdelta)>angtol)) stnum=3; //not positioned to goal or no clear path to goal
          }
        }

        //State#5: Goal reached
        //---------------------
        if (stnum==5)
        {
          //stop robot:
          v=vmin;
```

```
          w=wmin;
          std::cout << "Goal reached!" << std::endl;
          //for test purposes:
          //xg=s3x;
          //yg=s3y;
          stnum=1;
          goalcmd=false;
        }

        //Path planning/ navigation routine:
        //------------------------------------------
        if (navrun==true)
        {
          navrun=false;
          waycalc=true;
          xt=1000;
          yt=1000;
          rdist=maxObsDist;
          arealim=(DTOR((rangespan/secnum)/2))*sqrd(rdist); //Area=(theta/2)*R^2
          gdelta=AngDiff(xg,xc,yg,yc,yawc);
          dgoal = dcalc(xg,xc,yg,yc);

            if (xmin<rdist)
            {
              int sidx=0; //clear sector index
              int csector=0; //clear chosen sector index
              int scount; //sector counter
              while (sidx<secnum)
              {
                scount=0;
                if (sensdata[sidx]>(pfac*arealim))
                {
                  //if (((sidx>1)&&(sidx<6))&&(xmin<RL)) scount=0;
                  scount++;
                  for (int i=(sidx+1); i<secnum; i++)
                  {
                    //if (((sidx>1)&&(sidx<6))&&(xmin<RL)) i=secnum;
                    //else
                    //{
                      if (sensdata[i]>(pfac*arealim)) scount++;
                      else i=secnum;
                    //}
                  }
                  if ((maxObsDist*scount*DTOR(rangespan/secnum))>=(2.5*Rw)) //(S=R*theta) > 1.5*Rw?
                  {
                    ThetaArr[csector][0] = yawn + ((sidx-(secnum/2))*DTOR(rangespan/secnum)); //theta_1 (start)
                    ThetaArr[csector][1] = yawn + ((scount-(secnum/2))*DTOR(rangespan/secnum)); //theta_2 (end)
                    //Normalise to [0;2pi]
                    if (ThetaArr[csector][0]<0) ThetaArr[csector][0]+=(2*M_PI);
                    if (ThetaArr[csector][0]>(2*M_PI)) ThetaArr[csector][0]-=(2*M_PI);
                    if (ThetaArr[csector][1]<0) ThetaArr[csector][1]+=(2*M_PI);
                    if (ThetaArr[csector][1]>(2*M_PI)) ThetaArr[csector][1]-=(2*M_PI);
                    //add in safety distance:
                    ThetaArr[csector][0]+=safeDist;
```

```
                      ThetaArr[csector][1]-=safeDist;
                      csector++;
                    }
                    sidx+=scount;
                    std::cout << "sidx: " << sidx << std::endl;
                  }
                  else sidx++;
                } //end while
                for (int i=0; i<csector; i++)
                {
                  std::cout << "CSector " << i << ": " << ThetaArr[i][0] << ", " << ThetaArr[i][1] << std::endl;
                }
                //now determine the best theta and waypoints:
                int j=0;
                int tnum;
                bool gpath=false; //path to goal flag
                for (int i=0; i<csector; i++)
                {
                  tnum=BestThetaGoal(xc,yc,xg,yg,rdist,ThetaArr[i][0],ThetaArr[i][1]);
                  if (tnum==2) //path to goal
                  {
                    gpath=true;
                    xw=xg;
                    yw=yg;
                  }
                  else
                  {
                    if (tnum==0) ThetaArr[i][0]=ThetaArr[i][0];//-safeDist;
                    else ThetaArr[i][0]=ThetaArr[i][1];//+safeDist;
                    j++;
                  }
                  if (gpath==true) break; //exit for loop
                }
                //clear path to goal not found, so iterate to get best overall theta:
                if (gpath==false)
                {
                  thetat=ThetaArr[0][0];
                  for (int i=1; i<j; i++)
                  {
                    tnum=BestThetaPair(xc,yc,xg,yg,rdist,thetat,ThetaArr[i][0]);
                    if (tnum!=0) thetat=ThetaArr[i][0]; //replace new best theta
                  }
                  //might have to tweak point here so that sides of robot dont collide with obstacle!
                  xw = xc + (rdist*cos(thetat));
                  yw = yc + (rdist*sin(thetat));
                }
              }
              else
              {
                xw=xg;
                yw=yg;
              }
          } //end if (navrun==true)

          pp0.SetSpeed(v,w);
        }//end (lp.IsValid()==true)
```

```
//************************End GapNav*************************

//*****************Begin SVM and server comms******************
if (imode!=0) //svm and server comms
{
  //***********Agent interface***********
  //communicate with the server
  if (servercom==true)
  {
    std::cout << "<<<<Agent Interface>>>>" << std::endl;
    servercom=false; //reset flag
    //check server message here (for #server,end#) and only proceed if message is good!!!

    //Level 0 request (agent at home or destination):
    if ((amode==athome)||(amode==atdest))
    {
      if (getaid()==1)
      {
        if (getsvm()==1)
        {
          if (getago()==1)
          {
            if (getaclr()==1)
            {
              if (gettout()==1)
              {
                if (getbufdat()==1)
                {
                  std::cout << "<<<<Server packet is good!!!>>>>" << std::endl;
                  serverok=true; //server string is good
                  svmready=true; //ready to run svm algorithm
                }
                else serverok=false;
              }
              else serverok=false;
            }
            else serverok=false;
          }
          else serverok=false;
        }
        else serverok=false;
      }
      else serverok=false;
    } //level 0

    //Level 1 request (agent going to source or destination):
    if ((amode==gosrc)||(amode==godest))
    {
      if (getaid()==1)
      {
        if (getaclr()==1)
        {
          if (gettout()==1)
          {
            std::cout << "<<<<Server packet is good!!!>>>>" << std::endl;
            serverok=true; //server string is good
          }
          else serverok=false;
        }
```

```
            else serverok=false;
         }
         else serverok=false;
      } //level 1

      //Level 2 request (agent going at source):
      if (amode==atsrc)
      {
         if (getaid()==1)
         {
            if (getago()==1)
            {
               if (gettout()==1)
               {
                  std::cout << "<<<<Server packet is good!!!>>>>" << std::endl;
                  serverok=true; //server string is good
               }
               else serverok=false;
            }
            else serverok=false;
         }
         else serverok=false;
      } //level 2

   }//end if (servercom==true)

   //send packet to server
   if ((wait==false)&&(sendflag==true))
   {
      if (opensocket()==true) //connected to server
      {
         sendflag=false; //reset flag
         updatepkt(); //update packet before send to server
         std::cout << "Sending string to server..." << std::endl;
         send(sockfd, sendline, strlen(sendline), 0);
         if (recv(sockfd, inbuf, MAXLINE,0) == 0)
         {
            //error: server terminated prematurely
            perror("The server terminated prematurely");
            close(sockfd); //close the connection
         }
         else
         {
            printf("%s", "String received from the server: ");
            puts(inbuf);
            servercom=true; //ready to analyse packet received from server
            close(sockfd); //close the connection
         }
      }
      else
      {
         tout=htime;
         wait=true; //wait for tout seconds before trying to connect to server again
      }
   }
   //********End Agent interface**********


   //**********Learning component************
```

```
        if (svmready==true) //this flag is set after checking packet quality from server!
        {
          std::cout << "<<<<Learning component>>>>" << std::endl;
          svmready=false; //clear flag
          char *ftrain="train";
          char *fmod="model";
          char *fin="test";
          char *fout="predict";
          switch (svmode)
          {
            case learnc://learning mode
              strbuf(svmode); //append data string to train file
              svmflag=true;
              break;
            case trnpredc://train-prediction mode
              strbuf(svmode); //write data string to test file
              train_main(ftrain,fmod); //generate the model file
              predict_main(fin,fmod,fout); //predict target location and output to "predict" file
              //get the target location from the "predict" file and evaluate location
              if (get_loc()<=0) agoal=0;
              else
              {
                if ((Lmask & int(pow(2,get_loc()-1))) > 0) agoal=get_loc();
                else agoal=0;
              }
              svmflag=true;
              break;
            default://do nothing
              svmflag=false;
              break;
          }
        }
        //*******End Learning component**********


        //*********** Central Process ************
        //robot source and destination path planning

        //robot.Read(); //read from the proxies
        if (lp.IsValid()==true)
        {
        switch (amode)
        {
          case 0: //agent just woke up
            amode=1; //next state
            break;
          case 1: //go home
            SetGoal(Home[0],Home[1],0);
            if (dcalc(xc,Home[0], yc,Home[1])<dthresh)
            {
              amode=2; //next state
              wait=true; //set wait flag
              tout=htime; //timeout=htime
              sendflag=true; //ready to send packet to server
            }
            break;
          case 2: //at home
            if ((wait==false)&&(sendflag==false))
            {
```

```
        if ((svmflag==true)&&(ago==1)&&(serverok==true))
        {
          svmflag=false;
          serverok=false; //reset flag
          //if (agoal==0) amode=1; //go home
          if (agoal>0) amode=3; //go to source
          ago=0; //clear go flag
          sendflag=true; //ready to send packet to server
        }
        else
        {
          wait=true; //wait timeout
          sendflag=true; //ready to send packet to server
        }
      }
      break;
    case 3: //go to source
      if (dcalc(xc,Src[agoal-1][0], yc,Src[agoal-1][1])>dnear) //far away from source
      {
        SetGoal(Src[agoal-1][0],Src[agoal-1][1],0); //continue to source
      }
      else
      {
        if ((dcalc(xc,Src[agoal-1][0], yc,Src[agoal-1][1])<=dnear)&&(dcalc(xc,Src[agoal-1][0], yc,Src[agoal-1][1])>=dthresh)&&(wait==false)&&(sendflag==false)) //near source
        {
          apass=1; //set pass flag so that agent waits
          if ((aclr==1)&&(serverok==true))
          {
            SetGoal(Src[agoal-1][0],Src[agoal-1][1],0); //continue to source
          }
          else
          {
            //stay here and wait for tout seconds
            wait=true;
            sendflag=true;
          }
        }
        else
        {
          if (dcalc(xc,Src[agoal-1][0], yc,Src[agoal-1][1])<dthresh) //robot is at source - goto next state
          {
            amode=4; //next state
            apass=0; //clear agent pass flag
          }
        }
      }
      break;
    case 4: //at source
      if ((wait==false)&&(sendflag==false))
      {
        if ((ago==1)&&(serverok==true))
        {
          serverok=false; //reset flag
          amode=5; //next state
          ago=0; //clear go flag
          sendflag=true; //ready to send packet to server
```

```
          }
          else
          {
            wait=true; //wait timeout
            sendflag=true; //ready to send packet to server
          }
        }
        break;
      case 5: //go to destination
        if (dcalc(xc,Dest[agoal-1][0], yc,Dest[agoal-1][1])>dnear) //far away from destination
        {
          SetGoal(Dest[agoal-1][0],Dest[agoal-1][1],0); //continue to destination
        }
        else
        {
          if ((dcalc(xc,Dest[agoal-1][0], yc,Dest[agoal-1][1])<=dnear)&&(dcalc(xc,Dest[agoal-1][0], yc,Dest[agoal-1][1])>=dthresh)&&(wait==false)&&(sendflag==false)) //near destination
          {
            apass=1; //set pass flag so that agent waits
            if ((aclr==1)&&(serverok==true))
            {
              SetGoal(Dest[agoal-1][0],Dest[agoal-1][1],0); //continue to destination
            }
            else
            {
              //stay here and wait for tout seconds
              wait=true;
              sendflag=true;
            }
          }
          else
          {
            if (dcalc(xc,Dest[agoal-1][0], yc,Dest[agoal-1][1])<dthresh) //at destination
            {
              amode=6; //next state
              apass=0; //clear agent pass flag
            }
          }
        }
        break;
      case 6: //at destination
        if ((wait==false)&&(sendflag==false))
        {
          if ((svmflag==true)&&(ago==1)&&(serverok==true))
          {
            svmflag=false;
            serverok=false; //reset flag
            if (agoal==0) amode=1; //go home
            if (agoal>0) amode=3; //go to source
            ago=0; //clear go flag
            sendflag=true; //ready to send packet to server
          }
          else
          {
            wait=true; //wait timeout
            sendflag=true; //ready to send packet to server
          }
```

```
            }
            break;
          default:
            amode=0;
            break;
        }// end switch
        }// end if (lp.IsValid()==true)

        //end source and destination path planning
        //*********** End Central Process ************

      }//end if (imode!=0)
      //*****************End SVM and server comms*******************

      }//end while(1) - main loop
    }
    else
    {
      printf("Incorrect input parameters\n");
      exit(1);
    }//end if-else
    return 0;
}//main
```

A.5 PeopleBot real-world program: Player .cfg file

```
driver
(
  name "p2os"
  provides [ "odometry:::position2d:0" "sonar:0"]
  port "/dev/ttyS0"
)
```

A.6 RMP200 real-world program: Player .cfg file

```
driver
(
  name "segwayrmp"
  provides ["position2d:0" "position3d:0" "power:0" "ui:::power:1"]
  bus "usb"
  usb_device "/dev/ttyUSB0"
  max_xspeed 1.0
  max_yawspeed 40
)

driver
(
  name "hokuyo_aist"
  provides ["ranger:0"]
  portopts "type=serial,device=/dev/ttyACM0,timeout=1,baud=19200"
  min_dist 0.2 #0.15
```

```
  pose [0.25 0.0 0.0 0.0 0.0 0.0]
  alwayson 1
)

driver
(
  name "rangertolaser"
  requires ["ranger:0"]
  provides ["laser:0"]
)
```

A.7 RMP400 real-world program: Player .cfg file

```
driver
(
  name "segwayrmp"
  provides ["position2d:1" "position3d:1" "power:1"]
  bus "usb"
  usb_device "/dev/ttyUSB0"
  max_xspeed 0.5
  max_yawspeed 40
)

driver
(
  name "segwayrmp"
  provides ["position2d:2" "position3d:2" "power:2"]
  bus "usb"
  usb_device "/dev/ttyUSB1"
  max_xspeed 0.5
  max_yawspeed 40
)

driver
(
  name "segwayrmp400"
  provides ["position2d:0" "position3d:0"]
  requires ["front:::position3d:1" "back:::position3d:2" "front2d:::
    position2d:1" "back2d:::position2d:2"]
  fullspeed_data 1
)

driver
(
  name "hokuyo_aist"
  provides ["ranger:0"]
  portopts "type=serial,device=/dev/ttyACM0,timeout=1,baud=19200"
  min_dist 0.15 #0.25
  pose [0.25 0.0 0.0 0.0 0.0 0.0]
)

driver
(
  name "rangertolaser"
  requires ["ranger:0"]
  provides ["laser:0"]
)
```

Appendix B

Proview SCADA Code

B.1 Application interface program

```
/*
Filename: appServer3a.cpp
Author: N. Naidoo
Date: 25/04/2014
Revision: 3a
Description: Application server program for Proview SCADA
*/

/*
run these two commands to compile this code:

pwrp@nicol:/usr/local/pwrp/plantmain/src/appl$ g++ -g -c appServer3a.cpp -o
    /usr/pwr51/os_linux/hw_x86/exp/obj/appServer3a.o -I$pwr_inc -DOS_LINUX
    =1 -DOS=linux -DHW_X86=1 -DHW=x86 -DOS_POSIX

pwrp@nicol:/usr/local/pwrp/plantmain/src/appl$ g++ -g -o /usr/pwr51/
    os_linux/hw_x86/exp/exe/appServer3a /usr/pwr51/os_linux/hw_x86/exp/obj/
    appServer3a.o /usr/pwr51/os_linux/hw_x86/exp/obj/pwr_msg_rt.o -
    L$pwr_lib -lpwr_rt -lpwr_co -lpwr_msg_dummy -lrt

change "ld_appl_nicol_999.txt" load file in /usr/local/pwrp/plantmain/bld/
    common/load:
# User applications
# id,   name,   load/noload run/norun,  file,   prio,   debug/nodebug,  "
    arg"
appServer3a, appServer3a, noload, run, appServer3a, 12, nodebug, ""
#
# System processes
*/

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
```

```
#include <iostream>
#include "pwr.h"
#include "pwr_baseclasses.hpp"
#include "rt_gdh.h"
#include "rt_errh.h"

#define MAXLINE 4096 //max text line length
#define SERV_PORT 3000 //port
#define LISTENQ 8 //maximum number of client connections
#define rnum 3 //number of robot agents
#define bnum 8 //number of buffers
#define bsnum 3 //number of buffers being serviced
//agent modes
#define wakeup 0
#define gohome 1
#define athome 2
#define gosrc 3
#define atsrc 4
#define godest 5
#define atdest 6
//agent goals
#define nogoal -1
#define hgoal 0
//svm modes
#define nosvm 0
#define learnsvm 1
#define predictsvm 2
//go-stop command
#define rwait 0
#define rgo 1

//global variables
pwr_tBoolean bval;
pwr_tStatus sts;
pwr_tInt32 ival, ival2;
int aid, amode, agoal, aclr, apass;
int svmode, ago, tout, aloc;
char recbuf[MAXLINE], inbuf[MAXLINE]; //receive buffer
char sendbuf[MAXLINE]; //send buffer
bool pktQ; //packet quality flag. 1-> packet received from agent is good

int GetParam(char *buf1, char *buf2)
{
  char *delim="=";
  char *token, *strtemp;
  int tempi;

  strtemp=strstr(buf1,buf2);
  if (strtemp != NULL)
  {
    //printf("%s\n",strtemp);
    token = strtok(strtemp,delim);
    if (token != NULL)
    {
      delim=";";
      token = strtok(NULL,delim);
      if (token != NULL)
      {
        tempi=atoi(token);
      }
      else
      {
```

```
          tempi=-996;
        }
      }
      else
      {
        tempi=-997;
      }
    }
    else
    {
      tempi=-998;
    }
    std::cout << "test: " << tempi << std::endl;
    puts(recbuf);
    return tempi;
}

void updatedata()
{
    int tarr[bnum];
    char str1[20], str2[120], str3[20], str4[20], str5[30];

    //get buffer data
    sts = gdh_GetObjectInfo("H1-B0_Iv.ActualValue",&ival,sizeof(ival)); //GET the b0 buffer value
    tarr[0]=ival;
    sts = gdh_GetObjectInfo("H1-B1_Iv.ActualValue",&ival,sizeof(ival)); //GET the b1 buffer value
    tarr[1]=ival;
    sts = gdh_GetObjectInfo("H1-B2_Iv.ActualValue",&ival,sizeof(ival)); //GET the b2 buffer value
    tarr[2]=ival;
    sts = gdh_GetObjectInfo("H1-B3_Iv.ActualValue",&ival,sizeof(ival)); //GET the b3 buffer value
    tarr[3]=ival;
    sts = gdh_GetObjectInfo("H1-B4_Iv.ActualValue",&ival,sizeof(ival)); //GET the b4 buffer value
    tarr[4]=ival;
    sts = gdh_GetObjectInfo("H1-B5_Iv.ActualValue",&ival,sizeof(ival)); //GET the b5 buffer value
    tarr[5]=ival;
    sts = gdh_GetObjectInfo("H1-B6_Iv.ActualValue",&ival,sizeof(ival)); //GET the b6 buffer value
    tarr[6]=ival;
    sts = gdh_GetObjectInfo("H1-B7_Iv.ActualValue",&ival,sizeof(ival)); //GET the b7 buffer value
    tarr[7]=ival;

    //get and set other data for each agent
    if (aid==1)
    {
      sts=gdh_GetObjectInfo("H1-Svm1.ActualValue",&ival,sizeof(ival)); //r1 svm
      svmode=ival;
      sts=gdh_GetObjectInfo("H1-R1goal.ActualValue",&ival,sizeof(ival)); //r1 location
      aloc=ival;
      sts=gdh_GetObjectInfo("H1-R1go.ActualValue",&ival,sizeof(ival)); //R1go
      ago=ival;
      sts=gdh_GetObjectInfo("H1-Rclr1.ActualValue",&ival,sizeof(ival)); //r1 clr
      aclr=ival;
```

```
        sts=gdh_GetObjectInfo("H1-Rtout1.ActualValue",&ival,sizeof(ival)); //r1 tout
        tout=ival;
        //set amode:
        ival=amode;
        sts=gdh_SetObjectInfo("H1-R1loc.ActualValue",&ival,sizeof(ival)); //r1 mode
        //set apass:
        ival=apass;
        sts=gdh_SetObjectInfo("H1-R1pass.ActualValue",&ival,sizeof(ival)); //r1 pass
        //set agoal:
        if (svmode==predictsvm)
        {
          ival=agoal;
          sts=gdh_SetObjectInfo("H1-R1goal.ActualValue",&ival,sizeof(ival)); //r1 goal
        }
    }
    if (aid==2)
    {
        sts=gdh_GetObjectInfo("H1-Svm2.ActualValue",&ival,sizeof(ival)); //r2 svm
        svmode=ival;
        sts=gdh_GetObjectInfo("H1-R2goal.ActualValue",&ival,sizeof(ival)); //r2 location
        aloc=ival;
        sts=gdh_GetObjectInfo("H1-R2go.ActualValue",&ival,sizeof(ival)); //R2go
        ago=ival;
        sts=gdh_GetObjectInfo("H1-Rclr2.ActualValue",&ival,sizeof(ival)); //r2 clr
        aclr=ival;
        sts=gdh_GetObjectInfo("H1-Rtout2.ActualValue",&ival,sizeof(ival)); //r2 tout
        tout=ival;
        //set amode:
        ival=amode;
        sts=gdh_SetObjectInfo("H1-R2loc.ActualValue",&ival,sizeof(ival)); //r2 mode
        //set apass:
        ival=apass;
        sts=gdh_SetObjectInfo("H1-R2pass.ActualValue",&ival,sizeof(ival)); //r2 pass
        //set agoal:
        if (svmode==predictsvm)
        {
          ival=agoal;
          sts=gdh_SetObjectInfo("H1-R2goal.ActualValue",&ival,sizeof(ival)); //r2 goal
        }
    }
    if (aid==3)
    {
        sts=gdh_GetObjectInfo("H1-Svm3.ActualValue",&ival,sizeof(ival)); //r3 svm
        svmode=ival;
        sts=gdh_GetObjectInfo("H1-R3goal.ActualValue",&ival,sizeof(ival)); //r3 location
        aloc=ival;
        sts=gdh_GetObjectInfo("H1-R3go.ActualValue",&ival,sizeof(ival)); //R3go
        ago=ival;
```

```
      sts=gdh_GetObjectInfo("H1-Rclr3.ActualValue",&ival, sizeof(ival)); //r3 clr
      aclr=ival;
      sts=gdh_GetObjectInfo("H1-Rtout3.ActualValue",&ival, sizeof(ival)); //r3 tout
      tout=ival;
      //set amode:
      ival=amode;
      sts=gdh_SetObjectInfo("H1-R3loc.ActualValue",&ival, sizeof(ival)); //r3 mode
      //set apass:
      ival=apass;
      sts=gdh_SetObjectInfo("H1-R3pass.ActualValue",&ival, sizeof(ival)); //r3 pass
      //set agoal:
      if (svmode==predictsvm)
      {
        ival=agoal;
        sts=gdh_SetObjectInfo("H1-R3goal.ActualValue",&ival, sizeof(ival)); //r3 goal
      }
    }

    sprintf(str1,"%s%d%s","#server;aid=",aid,";");
    sprintf(str2,"%s%d%s%d%s%d%s%d%s%d%s%d%s%d%s%d%s%d%s%d%s","svm=",svmode,";data=",aloc,"  1:",tarr[0],"  2:",tarr[1],"  3:",tarr[2],"  4:",tarr[3],"  5:",tarr[4],"  6:",tarr[5],"  7:",tarr[6],"  8:",tarr[7],";");
    sprintf(str3,"%s%d%s%d%s","aclr=",aclr,";tout=",tout,";end#");
    sprintf(str4,"%s%d%s%d%s","ago=",ago,";tout=",tout,";end#");
    sprintf(str5,"%s%d%s%d%s%d%s","ago=",ago,";aclr=",aclr,";tout=",tout,";end#");

    if ((amode==athome)||(amode==atdest)) snprintf(sendbuf,MAXLINE,"%s%s%s",str1,str2,str5); //send all
    if ((amode==gosrc)||(amode==godest)) snprintf(sendbuf,MAXLINE,"%s%s",str1,str3); //send aclr
    if (amode==atsrc) snprintf(sendbuf,MAXLINE,"%s%s",str1,str4); //send ago
}

int main (int argc, char **argv)
{
 int listenfd, connfd, n;
 socklen_t clilen;
 struct sockaddr_in cliaddr, servaddr;

 sts = gdh_Init("appServer3a");
 if (EVEN(sts))
 {
   std::cout << "appServerXy Initialisation Failure! " << sts << std::endl;
   exit(0);
 }
 else
 {
   std::cout << "appServerXy Initialisation Success! " << sts << std::endl;
 }

 //creation of the socket
 listenfd = socket (AF_INET, SOCK_STREAM, 0);

 //preparation of the socket address
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
servaddr.sin_port = htons(SERV_PORT);

bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr)); //bind
listen(listenfd, LISTENQ); //listen

printf("%s\n","Server running...waiting for connections.");

//main loop
for (;;)
{
  clilen = sizeof(cliaddr);
  connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);
  printf("%s\n","Received request...");
  while ( (n = recv(connfd, inbuf, MAXLINE,0)) > 0)
  {
    printf("%s","String received from the client:");
    puts(inbuf);
    pktQ=false; //initialise packet quality to "bad"
    char *parstr="aid";
    snprintf(recbuf,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
    int tempi=GetParam(recbuf,parstr);
    if ((tempi>0)&&(tempi<=rnum)) //check aid
    {
      aid=tempi;
      std::cout << "aid: " << aid << std::endl;
      char *parstr="amode";
      snprintf(recbuf,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
      tempi=GetParam(recbuf,parstr);
      if ((tempi>=wakeup)&&(tempi<=atdest)) //check amode
      {
        amode=tempi;
        std::cout << "amode: " << amode << std::endl;
        char *parstr="agoal";
        snprintf(recbuf,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
        tempi=GetParam(recbuf,parstr);
        if ((tempi>=hgoal)&&(tempi<=bsnum)) //check agoal
        {
          agoal=tempi;
          std::cout << "agoal: " << agoal << std::endl;
          char *parstr="apass";
          snprintf(recbuf,MAXLINE,"%s",inbuf); //restore recbuf from inbuf
          tempi=GetParam(recbuf,parstr);
          if ((tempi==0)||(tempi==1)) //check apass
          {
            apass=tempi;
            std::cout << "apass: " << apass << std::endl;
            pktQ=true; //packet is good!
          }
        }
      }
    }
    if (pktQ==true) //only reply if packet is good
    {
      updatedata(); //get the data from SCADA and prepare packet
      send(connfd, sendbuf, strlen(sendbuf), 0); //send data to agent
      puts(sendbuf);
    }
  }//end while

  if (n < 0)
  {
    perror("Read error");
```

```
      exit(1);
    }
    close(connfd);

  }//end for (;;)
  //close listening socket
  close(listenfd);
}//end main
```

B.2 Simulation program

```
/*
Filename: prodsim3c.cpp
Author: N. Naidoo
Date: 26/09/2014
Revision: 3c
Description: Simulation program for Proview SCADA
*/

/*
run these two commands to compile this code:

pwrp@nicol:/usr/local/pwrp/plantmain/src/appl$ g++ -g -c prodsim3a.cpp -o /usr/pwr51/os_linux/hw_x86/exp/obj/prodsim3a.o -I$pwr_inc -DOS_LINUX=1 -DOS=linux -DHW_X86=1 -DHW=x86 -DOS_POSIX

pwrp@nicol:/usr/local/pwrp/plantmain/src/appl$ g++ -g -o /usr/pwr51/os_linux/hw_x86/exp/exe/prodsim3a /usr/pwr51/os_linux/hw_x86/exp/obj/prodsim3a.o /usr/pwr51/os_linux/hw_x86/exp/obj/pwr_msg_rt.o -L$pwr_lib -lpwr_rt -lpwr_co -lpwr_msg_dummy -lrt

change "ld_appl_nicol_999.txt" load file in /usr/local/pwrp/plantmain/bld/common/load:
# User applications
# id,  name,  load/noload run/norun,  file,  prio,  debug/nodebug,  "arg"
prodsim3a, prodsim3a, noload, run, prodsim3a, 12, nodebug, ""
#
# System processes
*/

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#include <iostream>
#include "pwr.h"
#include "pwr_baseclasses.hpp"
#include "rt_gdh.h"
#include "rt_errh.h"

//definitions
#define bnum 8
#define bsnum 3
```

```
#define rnum 3
#define load1 20
#define load2 5
#define load3 120
#define bs1 1
#define bs2 2
#define bs3 3
#define lwokeup 0
#define lgohome 1
#define lathome 2
#define lgosrc 3
#define latsrc 4
#define lgodest 5
#define latdest 6
#define b0cap 100
#define b1cap 50
#define b2cap 20
#define b3cap 20
#define b4cap 20
#define b5cap 20
#define b6cap 100
#define b7cap 100
#define go 1
#define stay 0
#define clear 1
#define notclear 0
#define nosvm 0
#define learnsvm 1
#define predictsvm 2

//global variables:
int bmtx[bsnum][2]; //mutex for each buffer service (src&dest). non -1
    value implies mutex is not available

void mtxtake(int anum, int bs_idx, int sd) //take mutex
{
  if ((bs_idx<=bsnum)&&(bs_idx>0)&&(bmtx[bs_idx-1][sd]==-1)) bmtx[bs_idx
    -1][sd]=anum;
}

bool mtxavail(int bs_idx, int sd) //check if mutex is available
{
  if ((bs_idx<=bsnum)&&(bs_idx>0)&&(bmtx[bs_idx-1][sd]==-1)) return true;
  else return false;
}

int mtxagent(int anum, int bs_idx) //check if mutex belongs to agent
{
  if ((bs_idx<=bsnum)&&(bs_idx>0))
  {
    if (bmtx[bs_idx-1][0]==anum) return 0; //src mutex
    else
    {
      if (bmtx[bs_idx-1][1]==anum) return 1; //dest mutex
      else return -1;
    }
  }
  else return -1;
}

void mtxrelease(int anum, int bs_idx, int sd) //release mutex
{
```

```
    if ((bs_idx<=bsnum)&&(bs_idx>0)&&(bmtx[bs_idx-1][sd]==anum)) bmtx[bs_idx
      -1][sd]=-1;
}

//main program
int main()
{
  pwr_tBoolean bval;
  pwr_tStatus sts;
  pwr_tInt32 ival, ival2;
  sts = gdh_Init("prodsim3a");
  if (EVEN(sts))
  {
    std::cout << "gdh_Init failure" << sts << std::endl;
    exit(0);
  }
  else
  {
    std::cout << "gdh_Init success!!! " << sts << std::endl;
  }

  //declarations
  int rloc[rnum]; //current pos of robot: 0=home, 1=source, 2=dest
  rloc[0]=0;
  rloc[1]=0;
  rloc[2]=0;
  int rlocp[rnum]; //previous pos of robot: 0=home, 1=source, 2=dest
  rlocp[0]=0;
  rlocp[1]=0;
  rlocp[2]=0;
  int rgoal[rnum]; //goal of robot: 0=nothing, 1=bs1, 2=bs2, 3=bs3
  rgoal[0]=0;
  rgoal[1]=0;
  rgoal[2]=0;
  int rload[rnum]; //load capacity of robot
  rload[0]=load1;
  rload[1]=load2;
  rload[2]=load3;
  int rprod[rnum]; //no. of products currently carried by robot
  rprod[0]=0;
  rprod[1]=0;
  rprod[2]=0;
  int rgo[rnum]; //robot ready to go {=1}
  rgo[0]=go;
  rgo[1]=go;
  rgo[2]=go;
  int rclr[rnum]; //robot clear flags
  rclr[0]=0; //robot not clear
  rclr[1]=0;
  rclr[2]=0;
  int rpass[rnum]; //robot passports
  rpass[0]=0; //robot not waiting
  rpass[1]=0;
  rpass[2]=0;
  int rsvm[rnum]; //robot svm modes
  rsvm[0]=nosvm;
  rsvm[1]=nosvm;
  rsvm[2]=nosvm;
  bmtx[0][0]=-1; //mutex for source is available to take
  bmtx[1][0]=-1;
  bmtx[2][0]=-1;
  bmtx[0][1]=-1; //mutex for destination is available to take
```

```
bmtx[1][1]=-1;
bmtx[2][1]=-1;
int tempi=0;

//main loop
while (1)
{
  //get robot locations:
  sts = gdh_GetObjectInfo("H1-R1loc.ActualValue",&ival, sizeof(ival));
  rloc[0]=ival;
  sts = gdh_GetObjectInfo("H1-R2loc.ActualValue",&ival, sizeof(ival));
  rloc[1]=ival;
  sts = gdh_GetObjectInfo("H1-R3loc.ActualValue",&ival, sizeof(ival));
  rloc[2]=ival;
  for (int i=0; i<rnum; i++)
  {
    if (rloc[i]!=rlocp[i])
    {
      if ( (rloc[i]==latsrc) || (rloc[i]==latdest) )
      {
        rgo[i]=stay; //robot must stay at source or destination
        rlocp[i]=rloc[i];
      }
    }
  }

  //get robot svm modes:
  sts=gdh_GetObjectInfo("H1-Svm1.ActualValue",&ival, sizeof(ival)); //r1 svm
  rsvm[0]=ival;
  sts=gdh_GetObjectInfo("H1-Svm2.ActualValue",&ival, sizeof(ival)); //r2 svm
  rsvm[1]=ival;
  sts=gdh_GetObjectInfo("H1-Svm3.ActualValue",&ival, sizeof(ival)); //r3 svm
  rsvm[2]=ival;

  //get robot buffer goals when robot at home or at destination:
  for (int i=0; i<rnum; i++)
  {
    if ( ((rloc[i]==lathome) || ( (rloc[i]==latdest) && (rgo[i]==go) )) || ((rsvm[i]==predictsvm)&&(rloc[i]==lgosrc)) )
    {
      if (i==0) sts = gdh_GetObjectInfo("H1-R1goal.ActualValue",&ival, sizeof(ival));
      if (i==1) sts = gdh_GetObjectInfo("H1-R2goal.ActualValue",&ival, sizeof(ival));
      if (i==2) sts = gdh_GetObjectInfo("H1-R3goal.ActualValue",&ival, sizeof(ival));
      rgoal[i]=ival;
    }
  }

  //get robot passports:
  sts = gdh_GetObjectInfo("H1-R1pass.ActualValue",&ival, sizeof(ival));
  rpass[0]=ival;
  sts = gdh_GetObjectInfo("H1-R2pass.ActualValue",&ival, sizeof(ival));
  rpass[1]=ival;
  sts = gdh_GetObjectInfo("H1-R3pass.ActualValue",&ival, sizeof(ival));
  rpass[2]=ival;

  //set robot clear flags:
```

```
      ival = rclr[0];
      sts = gdh_SetObjectInfo("H1-Rclr1.ActualValue",&ival, sizeof(ival));
      ival = rclr[1];
      sts = gdh_SetObjectInfo("H1-Rclr2.ActualValue",&ival, sizeof(ival));
      ival = rclr[2];
      sts = gdh_SetObjectInfo("H1-Rclr3.ActualValue",&ival, sizeof(ival));

      for (int i=0; i<rnum; i++)
      {
        //buffer source and destination calcs:-
        //work with buffer goal_1, source:
        if ( (rgoal[i]==bs1) && (rloc[i]==latsrc) && (rgo[i]==stay) )
        {
          bval=true;
          sts = gdh_SetObjectInfo("H1-S1_IN.ActualValue",&bval, sizeof(bval)); //set the wait-timer
          sts = gdh_GetObjectInfo("H1-S1_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
          if (bval==true)
          {
            bval=false;
            sts = gdh_SetObjectInfo("H1-S1_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
            sts = gdh_GetObjectInfo("H1-B0_Iv.ActualValue",&ival, sizeof(ival)); //GET the b0 buffer value
            if (ival>0)
            {
              if (rload[i]<=ival)
              {
                rprod[i]=rload[i];
                ival=ival-rprod[i];
              }
              else
              {
                rprod[i]=ival;
                ival=0;
              }
              sts = gdh_SetObjectInfo("H1-B0_Iv.ActualValue",&ival, sizeof(ival)); //SET the b0 buffer value
              rgo[i]=go; //robot is ready to leave source
            }//if (ival>0)
          }//if (bval==true)
        }//if ( (rgoal[0]==bs1) && (rloc[0]==lsrc) && (rgo[0]==stay) )

        //work with buffer goal_1, destination:
        if ( (rgoal[i]==bs1) && (rloc[i]==latdest) && (rgo[i]==stay) )
        {
          bval=true;
          sts = gdh_SetObjectInfo("H1-D1_IN.ActualValue",&bval, sizeof(bval)); //set the wait-timer
          sts = gdh_GetObjectInfo("H1-D1_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
          if (bval==true)
          {
            bval=false;
            sts = gdh_SetObjectInfo("H1-D1_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
            sts = gdh_GetObjectInfo("H1-B1_Iv.ActualValue",&ival, sizeof(ival)); //GET the b1 buffer value
            if (ival<b1cap)
            {
              if (rprod[i]<=(b1cap-ival))
```

```
                {
                  ival=ival+rprod[i];
                  rprod[i]=0;
                  rgo[i]=go; //robot is ready to leave destination
                }
                else
                {
                  tempi=rprod[i]-(b1cap-ival);
                  ival=b1cap;
                  rprod[i]=tempi;
                }
                sts = gdh_SetObjectInfo("H1-B1_Iv.ActualValue",&ival, sizeof(ival)); //SET the b1 buffer value
              }//if (ival<b1cap)
            }//if (bval==true)
          }//if ( (rgoal[i]==bs1) && (rloc[i]==ldest) && (rgo[i]==stay) )

          //------------------------------------------------------------------
          //work with buffer goal_2, source:
          if ( (rgoal[i]==bs2) && (rloc[i]==latsrc) && (rgo[i]==stay) )
          {
            bval=true;
            sts = gdh_SetObjectInfo("H1-S2_IN.ActualValue",&bval, sizeof(bval)); //set the wait-timer
            sts = gdh_GetObjectInfo("H1-S2_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
            if (bval==true)
            {
              bval=false;
              sts = gdh_SetObjectInfo("H1-S2_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
              sts = gdh_GetObjectInfo("H1-B4_Iv.ActualValue",&ival, sizeof(ival)); //GET the b4 buffer value
              if (ival>0)
              {
                if (rload[i]<=ival)
                {
                  rprod[i]=rload[i];
                  ival=ival-rprod[i];
                }
                else
                {
                  rprod[i]=ival;
                  ival=0;
                }
                sts = gdh_SetObjectInfo("H1-B4_Iv.ActualValue",&ival, sizeof(ival)); //SET the b4 buffer value
                rgo[i]=go; //robot is ready to leave source
              }//if (ival>0)
            }//if (bval==true)
          }//if ( (rgoal[0]==bs2) && (rloc[0]==lsrc) && (rgo[0]==stay) )

          //work with buffer goal_2, destination:
          if ( (rgoal[i]==bs2) && (rloc[i]==latdest) && (rgo[i]==stay) )
          {
            bval=true;
            sts = gdh_SetObjectInfo("H1-D2_IN.ActualValue",&bval, sizeof(bval)); //set the wait-timer
            sts = gdh_GetObjectInfo("H1-D2_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
            if (bval==true)
            {
```

```
          bval=false;
          sts = gdh_SetObjectInfo("H1–D2_IN.ActualValue",&bval, sizeof(bval)); //reset the wait–timer
          sts = gdh_GetObjectInfo("H1–B5_Iv.ActualValue",&ival, sizeof(ival)); //GET the b5 buffer value
          if (ival<b5cap)
          {
            if (rprod[i]<=(b5cap–ival))
            {
              ival=ival+rprod[i];
              rprod[i]=0;
              rgo[i]=go; //robot is ready to leave destination
            }
            else
            {
              tempi=rprod[i]–(b5cap–ival);
              ival=b5cap;
              rprod[i]=tempi;
            }
            sts = gdh_SetObjectInfo("H1–B5_Iv.ActualValue",&ival, sizeof(ival)); //SET the b5 buffer value
          }//if (ival<b5cap)
        }//if (bval==true)
      }//if ( (rgoal[i]==bs2) && (rloc[i]==ldest) && (rgo[i]==stay) )
      //––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

      //work with buffer goal_3, source:
      if ( (rgoal[i]==bs3) && (rloc[i]==latsrc) && (rgo[i]==stay) )
      {
        bval=true;
        sts = gdh_SetObjectInfo("H1–S3_IN.ActualValue",&bval, sizeof(bval)); //set the wait–timer
        sts = gdh_GetObjectInfo("H1–S3_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait–timer
        if (bval==true)
        {
          bval=false;
          sts = gdh_SetObjectInfo("H1–S3_IN.ActualValue",&bval, sizeof(bval)); //reset the wait–timer
          sts = gdh_GetObjectInfo("H1–B6_Iv.ActualValue",&ival, sizeof(ival)); //GET the b6 buffer value
          if (ival>0)
          {
            if (rload[i]<=ival)
            {
              rprod[i]=rload[i];
              ival=ival–rprod[i];
            }
            else
            {
              rprod[i]=ival;
              ival=0;
            }
            sts = gdh_SetObjectInfo("H1–B6_Iv.ActualValue",&ival, sizeof(ival)); //SET the b6 buffer value
            rgo[i]=go; //robot is ready to leave source
          }//if (ival>0)
        }//if (bval==true)
      }//if ( (rgoal[0]==bs3) && (rloc[0]==lsrc) && (rgo[0]==stay) )

      //work with buffer goal_3, destination:
      if ( (rgoal[i]==bs3) && (rloc[i]==latdest) && (rgo[i]==stay) )
```

```
      {
        bval=true;
        sts = gdh_SetObjectInfo("H1-D3_IN.ActualValue",&bval, sizeof(bval)); //set the wait-timer
        sts = gdh_GetObjectInfo("H1-D3_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
        if (bval==true)
        {
          bval=false;
          sts = gdh_SetObjectInfo("H1-D3_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
          sts = gdh_GetObjectInfo("H1-B7_Iv.ActualValue",&ival, sizeof(ival)); //GET the b7 buffer value
          if (ival<b7cap)
          {
            if (rprod[i]<=(b7cap-ival))
            {
              ival=ival+rprod[i];
              rprod[i]=0;
              rgo[i]=go; //robot is ready to leave destination
            }
            else
            {
              tempi=rprod[i]-(b7cap-ival);
              ival=b7cap;
              rprod[i]=tempi;
            }
            sts = gdh_SetObjectInfo("H1-B7_Iv.ActualValue",&ival, sizeof(ival)); //SET the b7 buffer value
          }//if (ival<b7cap)
        }//if (bval==true)
      }//if ( (rgoal[i]==bs1) && (rloc[i]==ldest) && (rgo[i]==stay) )

      //passport checks and mutex control:
      if ( (rpass[i]==1) && ((rloc[i]==lgosrc)||(rloc[i]==lgodest)) )
      {
        int temploc;
        if (rloc[i]==lgosrc) temploc=0;
        if (rloc[i]==lgodest) temploc=1;
        if (mtxavail(rgoal[i],temploc)==true)
        {
          mtxtake(i,rgoal[i],temploc); //take the mutex
          rclr[i]=clear;
          rgo[i]=stay; //clear rgo flag early - before robot reaches source or destination
        }
      }
      int tempstat;
      tempstat=mtxagent(i,rgoal[i]);
      if ((tempstat==0)||(tempstat==1)) //check if src or dest mutex held by agent
      {
        if (rpass[i]==0)
        {
          mtxrelease(i,rgoal[i],tempstat); //release the mutex
          rclr[i]=notclear;
        }
      }

    }//end for (int i=0; i<rnum; i++)

    //set rgo status for each robot:
```

```
    ival=rgo[0];
    sts = gdh_SetObjectInfo("H1-R1go.ActualValue",&ival, sizeof(ival));
    ival=rgo[1];
    sts = gdh_SetObjectInfo("H1-R2go.ActualValue",&ival, sizeof(ival));
    ival=rgo[2];
    sts = gdh_SetObjectInfo("H1-R3go.ActualValue",&ival, sizeof(ival));

    //machine & conveyor efficiencies:
    //machine 1:
    sts = gdh_GetObjectInfo("H1-M1En.ActualValue",&bval, sizeof(bval));//get machine start signal
    if (bval==true)
    {
      sts = gdh_GetObjectInfo("H1-B1_Iv.ActualValue",&ival, sizeof(ival));//get b1 value
      sts = gdh_GetObjectInfo("H1-B2_Iv.ActualValue",&ival2, sizeof(ival2));//get b2 value
      if ((ival>0)&&(ival2<b2cap))
      {
        sts = gdh_SetObjectInfo("H1-M1_IN.ActualValue",&bval, sizeof(bval)); //set the wait-timer
        sts = gdh_GetObjectInfo("H1-M1_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
        if (bval==true)
        {
          bval=false;
          sts = gdh_SetObjectInfo("H1-M1_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
          ival=ival-1;
          sts = gdh_SetObjectInfo("H1-B1_Iv.ActualValue",&ival, sizeof(ival));//set b1 value
          ival2=ival2+1;
          sts = gdh_SetObjectInfo("H1-B2_Iv.ActualValue",&ival2, sizeof(ival2));//set b2 value
        }
      }
    }
    //conveyor:
    sts = gdh_GetObjectInfo("H1-C1En.ActualValue",&bval, sizeof(bval));//get conveyor start signal
    if (bval==true)
    {
      sts = gdh_GetObjectInfo("H1-B2_Iv.ActualValue",&ival, sizeof(ival));//get b2 value
      sts = gdh_GetObjectInfo("H1-B3_Iv.ActualValue",&ival2, sizeof(ival2));//get b3 value
      if ((ival>0)&&(ival2<b3cap))
      {
        sts = gdh_SetObjectInfo("H1-C1_IN.ActualValue",&bval, sizeof(bval)); //set the wait-timer
        sts = gdh_GetObjectInfo("H1-C1_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
        if (bval==true)
        {
          bval=false;
          sts = gdh_SetObjectInfo("H1-C1_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
          ival=ival-1;
          sts = gdh_SetObjectInfo("H1-B2_Iv.ActualValue",&ival, sizeof(ival));//set b2 value
          ival2=ival2+1;
```

```
            sts = gdh_SetObjectInfo("H1-B3_Iv.ActualValue",&ival2, sizeof(ival2));//set b3 value
          }
        }
      }
      //machine 2:
      sts = gdh_GetObjectInfo("H1-M2En.ActualValue",&bval, sizeof(bval));//get machine start signal
      if (bval==true)
      {
        sts = gdh_GetObjectInfo("H1-B3_Iv.ActualValue",&ival, sizeof(ival));//get b3 value
        sts = gdh_GetObjectInfo("H1-B4_Iv.ActualValue",&ival2, sizeof(ival2));//get b4 value
        if ((ival>0)&&(ival2<b4cap))
        {
          sts = gdh_SetObjectInfo("H1-M2_IN.ActualValue",&bval, sizeof(bval));//set the wait-timer
          sts = gdh_GetObjectInfo("H1-M2_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
          if (bval==true)
          {
            bval=false;
            sts = gdh_SetObjectInfo("H1-M2_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
            ival=ival-1;
            sts = gdh_SetObjectInfo("H1-B3_Iv.ActualValue",&ival, sizeof(ival));//set b3 value
            ival2=ival2+1;
            sts = gdh_SetObjectInfo("H1-B4_Iv.ActualValue",&ival2, sizeof(ival2));//set b4 value
          }
        }
      }
      //machine 3:
      sts = gdh_GetObjectInfo("H1-M3En.ActualValue",&bval, sizeof(bval));//get machine start signal
      if (bval==true)
      {
        sts = gdh_GetObjectInfo("H1-B5_Iv.ActualValue",&ival, sizeof(ival));//get b5 value
        sts = gdh_GetObjectInfo("H1-B6_Iv.ActualValue",&ival2, sizeof(ival2));//get b6 value
        if ((ival>0)&&(ival2<b6cap))
        {
          sts = gdh_SetObjectInfo("H1-M3_IN.ActualValue",&bval, sizeof(bval));//set the wait-timer
          sts = gdh_GetObjectInfo("H1-M3_OUT.ActualValue",&bval, sizeof(bval)); //get output from wait-timer
          if (bval==true)
          {
            bval=false;
            sts = gdh_SetObjectInfo("H1-M3_IN.ActualValue",&bval, sizeof(bval)); //reset the wait-timer
            ival=ival-1;
            sts = gdh_SetObjectInfo("H1-B5_Iv.ActualValue",&ival, sizeof(ival));//set b5 value
            ival2=ival2+1;
            sts = gdh_SetObjectInfo("H1-B6_Iv.ActualValue",&ival2, sizeof(ival2));//set b6 value
          }
        }
```

```
      }
      sleep(1);
      std::cout << "R1 go,loc,goal:" << rgo[0] << "," << rloc[0] << "," <<
     rgoal[0] << std::endl;
   }//end while
   exit(0);
}
```