Obstacle Avoidance and Trajectory Optimisation for a Power Line Inspection Robot

Timothy Rowell

In fulfillment of the Master of Science in Engineering Degree (Electronic Engineering), undertaken at the School of Engineering, University of KwaZulu-Natal.

Supervisor:
Professor E.S. Boje

October 2012

As the candidate's Supervisor I agree/do not agree to the submission of this dissertation.

Signed:

Declaration

I, Timothy Rowell, declare that

(i) The research reported in this dissertation/thesis, except where otherwise indicated, is my original work.

(ii) This dissertation/thesis has not been submitted for any degree or examination at any other university.

(iii) This dissertation/thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This dissertation/thesis 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;

b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) Where I have reproduced a publication of which I am an author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.

(vi) This dissertation/thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation/thesis and in the References sections.

Signed:

Acknowledgements

This master's programme has been a journey of many challenges, and ultimately many rewards. Throughout this programme there have been key people who have supported me and encouraged me, making it possible for me to complete the journey I started.

I thank my supervisor and mentor, Prof. Edward Boje, for his support and guidance throughout this master's programme. He is always approachable and willing to impart valuable knowledge and wisdom to me, and gives generously of his time. Over and above this, he provides support in any way he can when challenges arise.

I also thank Trevor Lorimer and Yashren Reddi, my laboratory go-to guys. They are always willing to lend a hand and share their knowledge and expertise. Without them, many more hours of head-scratching would have been endured. Special acknowledgement goes to Trevor Lorimer for his support (and patience).

This work was funded by the National Research Foundation (NRF Grant-holder project 65860), the Eskom Tertiary Education Support Programme (TESP) and Eskom Sustainability and Innovation Department (Project D.R9004.07, Contract 4600011734). This assistance is greatly appreciated.

I thank my family, my biggest fans. They have always been there for me and their love gives me a solid foundation for all that I do in life. Finally, to my other biggest fan, Kate. She is always loving, caring and understanding (tolerant), and for that I am always grateful.

Abstract

This dissertation presents the research, development and application of trajectory creation, obstacle avoidance and trajectory optimisation methods for an existing serial manipulator power line inspection robot (PLIR). The obstacle avoidance implementation allows the robot to navigate around an obstacle obstructing its navigation along the line. The algorithm generated end effector trajectory waypoints autonomously based on bounding box obstacle descriptions in Cartesian space, and connected them with a fifth order basis-spline end effector trajectory command. The trajectories were created taking into account the dynamic torque and velocity constraints of the robot while ignoring non-linearities. Performance was inspected and evaluated in a simulated workspace environment.

The trajectory optimisation was designed to maximise the robot's operating range, with constraints on the battery power supply, by minimising charge consumed during obstacle avoidance trajectories. The temporal components of the basis-spline trajectories were optimised by minimising a time-energy type of cost function subject to the dynamic constraints of the robot. Cost function analyses are presented for a simple frictionless robot model based on the recursive Newton-Euler method, and for a more realistic model including viscous, Coulomb and static friction as well as gearbox backlash.

It is shown that the Nelder-Mead simplex method was appropriate for optimisation. For representative trajectories that were studied, the optimiser was capable of finding global minima with satisfactory speed and accuracy in simulation. The validity of trajectory optimisation with regard to the cost function behaviour was confirmed. This was based on experiments carried out on the robot hardware in the laboratory, examining the predicted and actual actuator current profiles.

The engineering design and implementation of hardware and software for the base station and on-board system is presented, together with the layout of the PLIR's control system and PID (proportional-integral-derivative) controller design. Trajectory commands are sent from the base station to the robot via Wi-Fi for execution. Furthermore, live video feed from the robot can be sent to the ground station computer. Furthermore, high voltage testing of the PLIR showed that the engineering design of the robot and communication platform is robust.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

3D	Three dimensional
AABB	Axis-aligned bounding box
ADC	Analogue to digital converter
C#	C# Microsoft .NET programming language
CD	Compact disc
COM	Component Object Model
CPU	Central processing unit
DDR	Double data rate
DLS	Damped least squares
DOF	Degree of freedom
FSB	Front-side bus
GB	Gigabyte (10^9 bytes)
GHz	Gigahertz (10^9 Hz)
GPU	Graphics processing unit
GUI	Graphical user interface
HV	High voltage
ID	Inverse dynamics
IK	Inverse kinematics
ITO	Interval time optimisation
kV	Kilovolt (10^3 V)
MHz	Megahertz (10^6 Hz)
.NET	Microsoft software framework
OS	Operating system

PC	Personal computer
PID	Proportional-integral-derivative
PLIR	Power line inspection robot
PWM	Pulse-width modulation
RAM	Random access memory
SPI	Serial peripheral interface
TRO	Time ratio optimisation
UART	Universal asynchronous receiver/transmitter
UAV	Unmanned aerial vehicle
WCF	Windows Communication Foundation

Roman Symbols

$B_j^d(t)$	Basis-function j of degree d
k_τ, k_ω, k_I	Cost penalty scaling factors for torque, velocity and current respectively
d	Basis-spline degree
$\vec{e}$	Cartesian position error between end effector position and target
$\mathrm{F_{pre}}$	Control system prefilter
G	Controller transfer function
g	Gravitational acceleration
$\mathbf{I}$	Identity matrix
I	Actuator current
I_C	Average component current
I_{peak}	Peak pulse current that the battery is capable of supplying
I_{total}	Total sum of the actuator and electronic component currents supplied by the battery
$\mathbf{J}$, $\mathbf{J}^\dagger$	IK Jacobian matrix and its pseudo-inverse respectively
$J(\bar{t}_{wp})$, $J^P(\bar{t}_{wp})$	Trajectory cost function, and trajectory cost function with limit violation penalties, respectively
J	Joint inertia
$\mathrm{k_p}$	Controller proportional gain
k_ϕ	Actuator motor/torque constant
L	Actuator armature inductance
l	Number of trajectory solver steps of size T

m	(Number of basis-spline control points) - 1
m_{ef}, m_c	End effector mass and chassis core mass respectively
n	(Number of waypoints) - 1
O_j	Cartesian coordinate frame for joint j
$\bar{p}$	Basis-spline control point vector
p_j	Basis-spline control point j
$\vec{p_j}$	Cartesian position of the j^{th} robot joint
Q_C	Charge consumed by robot components (excluding actuators)
R	Actuator armature resistance
r	Actuator gearbox ratio
$\mathbb{R}^3$, $\mathbb{R}^5$, $\mathbb{R}^n$	3-dimensional, 5-dimensional and n-dimensional Euclidean space respectively
$\bar{r}$, $\bar{r}'$	Scaled and unscaled trajectory interval time vectors respectively
$\vec{s}$	Cartesian end effector position vector
$\vec{s}(\bar{\theta}(t))$	End effector trajectory
$\mathbf{s}_{wp}$	End effector waypoint matrix
$\vec{t}$	Cartesian end effector target vector
T	Trajectory solver and control system time-step size
t_C	Time over which robot components consume charge
$\mathrm{T_i}$, $\mathrm{T_d}$	Controller integrator time constant and derivative time constant respectively
$\bar{t}_{wp}$, $\bar{t}'_{wp}$	Scaled and unscaled waypoint time vectors respectively
t_{wp_0}	Initial waypoint vector time
t_{wp_n}, t'_{wp_n}	Scaled and unscaled final waypoint vector times respectively
$\bar{u}$, $\bar{u}'$	Scaled and unscaled basis-spline time knot vectors respectively
$\vec{v}_j$	The unit vector pointing along the rotation axis of the j^{th} joint
V, V_{max}	Applied actuator voltage and actuator supply voltage limit respectively

Greek Symbols

$\frac{1}{\lambda_\tau}$, $\frac{1}{\lambda_\omega}$, $\frac{1}{\lambda_I}$	Scaling factors for torque, velocity and current limits respectively
γ	DLS damping factor
$\bar{\theta}$	Robot joint angle vector
$\bar{\theta}_{current}$, $\bar{\theta}_{next}$	IK iteration: current joint configuration and updated joint configuration respectively

$\dot{\theta}$, $\ddot{\theta}$, $\dddot{\theta}$	Angular velocity, acceleration and jerk respectively
$\dot{\theta}_{limit}$	Angular velocity limit
$\triangle\bar{\theta}$	IK iteration: incremental joint update value
θ_j	Joint angle for joint j
$\bar{\theta}_s$, $\bar{\theta}_t$	Robot joint configuration with end effector at $\vec{s}$ and at $\vec{t}$ respectively
$\bar{\theta}_{wp}$	Waypoint angle vector
κ	Initial time setting for t'_{wp_n}
λ	Trajectory time scaling factor
τ	Joint torque
τ_{elec}	Torque produced by the actuator
τ_{limit}	Joint torque limit
$\tau_{line_{friction}}$	Torque due to friction between the power line and the attached gripper wheels
τ_{static}, $\tau_{dynamic}$	Static and dynamic joint torque respectively
ϕ	Robot angle of rotation about the power line
ω, ω_{max}	Angular velocity and angular velocity limit respectively

Chapter 1

Introduction

1.1 Problem Statement

In order for an electricity utility to ensure reliability of supply and that their infrastructure does not pose a long term risk to society, maintenance and up-keep of its infrastructure is essential, and in South Africa is required by law (Lorimer, 2011). Forming an effective and efficient maintenance schedule requires thorough knowledge of the state of all components making up the infrastructure, which can be achieved through regular inspection. In the scope of this dissertation, the aspects of the infrastructure to be inspected are the power lines, pylons, power line insulators, power line hardware such as vibration dampers and the right-of-way.

There exist a number of traditional power line inspection methods, for example ground patrols, climbing patrols and detailed aerial inspections (Powerline Patrol Services, 2010). Ground patrols are time intensive and the upper parts of the infrastructure cannot be observed at close proximity. Also, they my not be able to reach all parts of the infrastructure easily. Aerial patrols are hazardous and costly (Powerline Patrol Services, 2010). For such reasons, robots have made their appearance in the field of power line inspection. Robots can inspect the power line infrastructure in great detail while keeping human inspection crews at safe distances from the lines, and reducing inspection costs.

While power line inspection robots are a logical progression in the technological advances of power line inspection, operating them can be a slow process due to their complex nature. Furthermore, due to the limited availability of power to the robot, energy efficiency is another issue that needs to be addressed. For this reason, there is a need to simplify the operation of power line inspection robots to make them faster and more energy efficient.

1.2 Objectives

The School of Engineering at the University of KwaZulu-Natal in collaboration with Eskom is undertaking a power line inspection robot development project that aims to provide a means for autonomous inspection of power line infrastructures. The first prototype for this project has been created by Trevor Lorimer of the University of KwaZulu-Natal as an MScEng research topic, and this dissertation makes some key advancements upon the existing work. An illustration of the existing robot prototype taken from Lorimer (2011) is given in Figure 1.1.

The objectives of this master's dissertation included the development of an autonomous obstacle avoidance system for the robot to improve operating speed and simplicity, and the optimisation of the obstacle avoidance trajectories' waypoint interval times in order to maximise energy efficiency. An autonomous obstacle avoidance system needed to be responsible for creating trajectories for the robot to execute if an obstacle is encountered on the line, allowing the robot to bypass the obstacle and continue traversing the line. Such obstacle avoidance trajectories require large amounts of electric current when compared with rolling along the line because one gripper is detached. Therefore the amount of charge used for these trajectories must be minimised to relieve demand on the power supply.

1.3 Specifications

The first step towards obstacle avoidance was to represent the robot's workspace using a 3D graph in Cartesian space. The robot must have knowledge of the obstacles in the workspace, and the obstacles are described using axis-aligned bounding boxes, commonly used in 3D graphics applications. Once the obstacles in the workspace are known, the robot must generate paths to get around the obstacles. The Lazy Theta* path finding algorithm by Nash *et al.* (2010) was used to find the shortest path between two points in a 3D graph taking obstacles into account.

Once obstacle avoidance paths have been found, trajectories must be created. The work presented in this dissertation made use of the basis-spline description as found in Biagiotti and Melchiorri (2008). These obstacle avoidance trajectories are to be optimised in order to minimise the amount of charge required for the end effector to follow the trajectory. This was done by adjusting the temporal components of the basis-splines using a simplex optimisation method. The optimisation of the shortest straight-line path however, does not take into account the optimisation of the spatial trajectory components. For this reason, the optimisation presented in this dissertation was not full global optimisation due to its intractable nature.

The above-mentioned functionality was to be encompassed within a software application with a

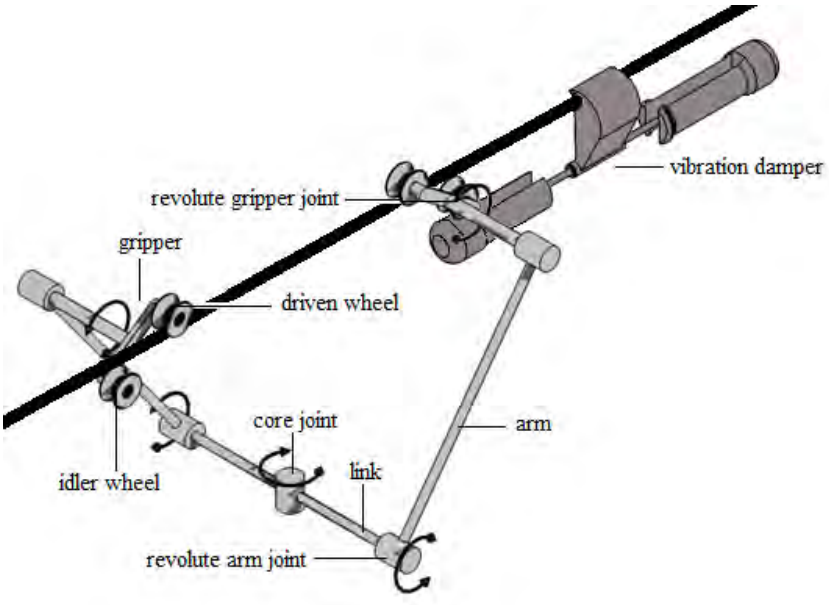

Figure 1.1: Illustration of the existing power line inspection robot prototype (Lorimer, 2011).

graphical user interface, that allows the user to analyse the entire obstacle avoidance procedure, from the fundamental level of describing the obstacles in the workspace, to the simulation of the optimised obstacle avoidance manoeuvres. These trajectories were to be transferred to the robot's on-board computer over a Wi-Fi communication link for execution. This software application has been designed with eventual autonomous operation in mind, and can be implemented on the on-board PC of the robot.

1.4 Layout of the Dissertation

In Chapter 2, six fundamental topics are discussed namely: inverse kinematics, robot modelling, trajectory planning, robot simulation, obstacle avoidance and trajectory optimisation. These sections provide the background required for the following chapters, and include literature surveys to present the current state of the art with regard to obstacle avoidance and trajectory optimisation.

Techniques for obstacle avoidance are presented in Chapter 3. Obstacle avoidance was implemented by simulating the workspace environment of the robot, and generating trajectories that allowed the robot to circumvent obstacles in this environment. These trajectories can then be used in the physical application, allowing the robot to avoid collisions with power line obstacles during inspection. The approach for selecting a valid collision free path around a given obstacle is presented in the sections of this chapter involving waypoint selection, path finding and collision detection. This dissertation has developed the obstacle avoidance capabilities of the robot to a level where these trajectories can be generated autonomously, provided that the workspace obstacle descriptions have been provided to the simulator.

Trajectory optimisation is discussed in Chapter 4. This dissertation developed a trajectory optimisation approach to minimise the total electric charge required and hence the demand placed on the robot's power supply. This is because the robot is battery powered, with plans for a future revision of the robot where the battery is charged from the power line itself using a clamp-on transformer core. Hence, the power supply is constrained by the payload mass of both the on-board battery pack and the clamp-on transformer core. A numerical simplex optimisation method has been implemented in this dissertation to minimise the time-energy type of cost function for obstacle avoidance trajectories.

Chapter 5 presents the underlying work that was involved in achieving the implementation of a functional power line inspection robot simulation environment for the creation and analysis of optimised obstacle avoidance trajectories. The software environment is run on a "ground station" computer that is, at this stage of development, responsible for supplying the robot with the required trajectories. These trajectories are sent to the robot's on-board computer for execution. The robot is also capable of sending video from an on-board camera to the ground station which can be viewed in the C# application. Communication is achieved using a Wi-Fi link. The robot's control system layout is discussed along with the design of the PID controllers used for joint actuator position control. This chapter also presents the Simulink PLIR model used to provide detailed system behaviour predictions. Finally, the hardware used to implement these systems is presented.

Experimental results are presented in Chapter 6. The first set of results show two optimised obstacle avoidance trajectories for a suspension clamp avoidance manoeuvre. The first trajectory was optimised using the Simulink robot model, and the second trajectory was optimised using the inverse dynamics model. Both trajectories were simulated and executed on a power line analogue in the laboratory and the results are analysed in this chapter. The second set of results show an

optimised obstacle avoidance trajectory for a jumper terminal manoeuvre that was executed on a power line segment in the laboratory. The PLIR was also tested in a high voltage laboratory to verify the robustness of the engineering design, and the results are presented in this chapter.

Chapter 7 concludes the dissertation, summarising the important aspects of the dissertation and commenting on the findings. This chapter is followed by the references and appendices. Appendix A presents design parameters for the Simulink model and the joint actuator PID controllers. Appendix B contains the software source code and Simulink model at the end of the dissertation on a CD. Finally, Appendix C contains the specifications of the ground station and on-board computers.

Chapter 2

Literature Review and Background

2.1 Kinematics

Kinematics for the PLIR refers to the relationship between the joint angles of the robot and the positions of each of the robot's links and joints. The forward kinematic solution for the PLIR yields part positions of the robot based on the angles of the revolute joints. The inverse kinematic solution for the PLIR yields the required joint angles to achieve a desired end effector position for the PLIR.

The kinematics solutions discussed in this section have already been implemented for the PLIR by Lorimer (2011), and have been revised and re-applied in this dissertation. A photograph of the robot is given in Figure 2.1.

The platform of the PLIR is a serial manipulator robot with six links and five revolute joints that connect the links. The naming convention of each joint axis coordinate frame is shown in Figure 2.2, and will be used in this dissertation.

The naming convention for the joints and links themselves (shown in Figure 2.3) has been adapted

Figure 2.1: A photograph of the power line inspection robot.

from Figure 2.2. These joint angle labels are used in the application of the inverse kinematics solution that will follow.

The configuration of the PLIR is dependent on the angles of each of the robot's joints. This set of joint angles can be viewed as a vector $\bar{\theta} = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4]$. Each angle in $\bar{\theta}$ is defined relative to each joint axis in Figure 2.2 as follows:

- θ_0: Line rotation angle about the z axis of O_0 (not controlled)
- θ_1: Base rotation angle about y axis of O_1
- θ_2: Attached arm rotation angle about y axis of O_2
- θ_3: Core rotation angle about x axis of O_3
- θ_4: Detached arm rotation angle about y axis of O_4

The joint naming convention will change depending on which gripper is attached.

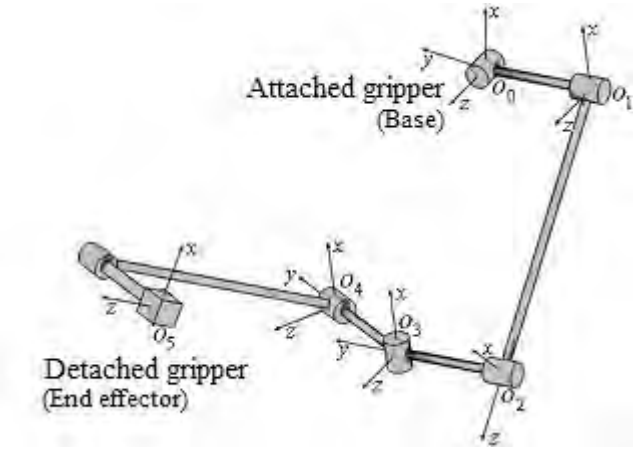

Figure 2.2: Axes coordinate frames (*ibid.*).

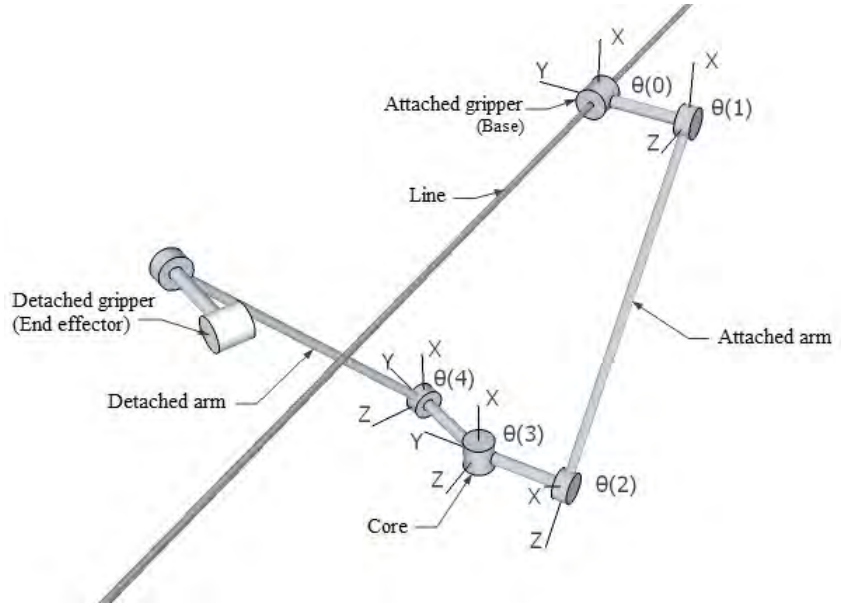

Figure 2.3: Joint and link naming convention.

2.1.1 Forward kinematics

For the forward kinematics solution for the PLIR, rotation matrices together with the link vectors describe the position of each of the joints relative to the base coordinate system.

The solution is achieved in three steps. The first step calculates the part positions relative to the base (shown in Figure 2.3) without taking the line rotation into account. The second step is to calculate the line rotation angle. The robot is able to rotate about the z axis of the power line coordinate frame O_0. This rotation angle (ϕ) depends on the torque balance about the z axis, simplified in (2.1) (*ibid.*). This equation assumes that all robot components except for the end effector and robot core have negligible mass. In (2.1), d_5 and d_3 are the perpendicular distances from the power line to the end effector and to the robot core respectively, g is the acceleration due to gravity, m_{ef} and m_c are the masses of the end effector and the core respectively, and $\tau_{line_{friction}}$ is the rotational friction between the power line and the attached gripper wheels. The angle α can be seen in Figure 2.4.

$$d_5 m_{ef} g \cos(\phi + \alpha) - d_3 m_c g \sin(\phi) + \tau_{line_{friction}} = 0 \tag{2.1}$$

Figure 2.4 shows the geometry used to calculate the line rotation angle, ϕ. It is assumed that the effects of static and kinematic friction between the grippers and the line are insignificant, i.e. $\tau_{line_{friction}} \ll d_5 m_{ef} g \cos(\phi + \alpha) - d_3 m_c g \sin(\phi)$, and that the robot is not swinging about the line. Finally, using the rotation matrix about the z axis together with the line rotation angle, the link and joint positions are calculated using the respective rotation matrices and link vectors.

2.1.2 Inverse kinematics

The inverse kinematics (IK) solution for the robot is not in closed form as there is no known set of equations that can be used in order to obtain the robot's joint angles analytically from given Cartesian coordinates of the end effector. Such set of closed form equations may exist, but in order to derive them, difficult geometric and trigonometric equations would have to be solved. Instead, a

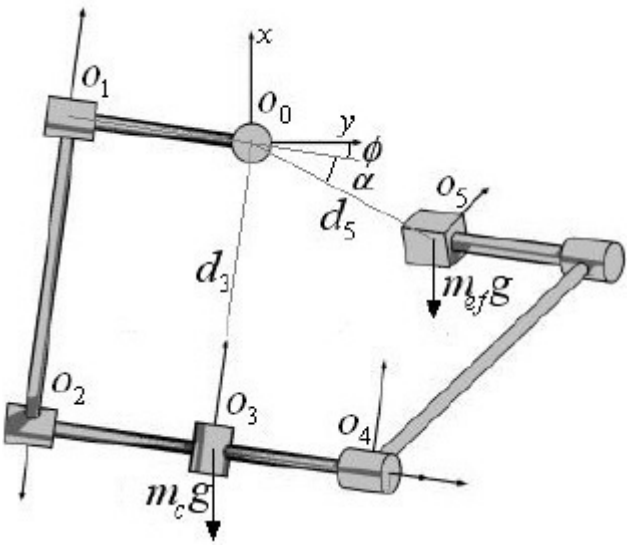

Figure 2.4: Line rotation geometry (*ibid.*).

numerical solution was sought since the computing power available can realise required solutions in real time. Real time in this context means that the IK solution is available in time to avoid any delay in the robot's operation.

Buss (2004) covers the development of the inverse kinematics solutions for bodies with multiple links and end effectors that are required to reach desired target positions. Although Buss (2004) deals with the development of an inverse kinematic solution for computer graphics and real-time animation, it is very well suited to the PLIR application.

Buss (2004) also presents robust inverse kinematics solutions for cases where the target positions for the end effectors are out of reach by ensuring that singularities do not lead to numerical instability whilst solving for the joint angles. This is applicable to the PLIR in cases where a certain target position (waypoint) for the end effector may not be reachable due to the physical limitations of the robot. In this dissertation, the inverse kinematic solution is translated into the PLIR application, and the resulting method for solving the problem is justified.

Let the position of the end effector be $\vec{s}$ in Cartesian coordinates, and let $\vec{t}$ be the target position in Cartesian coordinates, then $\vec{e} = \vec{t} - \vec{s}$ is the desired change in position to get the end effector to its target position. With the end effector position being a function of the joint angles, we can write $\vec{s} = \vec{s}(\bar{\theta}_s)$. Now we want to find the values for $\bar{\theta}_t$ such that

$$\vec{s} = \vec{t}(\bar{\theta}_t) \tag{2.2}$$

In the case of the PLIR, θ_2 and θ_4 are collinear when $\theta_3 = 0$, the solution for a desired end effector position is not unique. This non-uniqueness case is shown in Figure 2.5, which is a simplified side view of the robot showing the attached arm and detached arm joints as collinear. It clearly shows that there are two unique configurations for the joints that give the same end effector position. The solid line represents the solution with the robot below the power line, and the dashed line is the solution with the robot above the power line. The dashed solution is physically not possible since gravity would ensure that the centre of mass is kept below the power line. However, it can exist as far as the IK solver is concerned. If the IK solver of the PLIR is finding a solution using a sufficiently large damping factor and the starting configuration is sufficiently near a singularity, the damping factor may perturb it to the "wrong" side of the singularity resulting in a solution that may be one like the dashed line. The singularity in question would occur when the robot is extended to full reach at the limit of its workspace boundary, making the attached and detached arm parallel. The identification of such a singularity is covered in Spong *et al.* (2006), Section 4.9. In order to prevent the dashed solution from occurring, one can set operational limits for the joint angles of the PLIR to ensure solution uniqueness in such cases. If two solutions exist to get the end effector into the desired position, one may be outside of the operational limits for certain joints and can be rejected in favour of the "desired" solution.

In order to find a suitable solution to (2.2), we can use an iterative approach in order to approximate it using (2.3).

$$\vec{s}(\bar{\theta}_{current} + \triangle\bar{\theta}) \approx \vec{s}(\bar{\theta}_{current}) + \frac{\partial \vec{s}}{\partial \bar{\theta}} \triangle\bar{\theta} \tag{2.3}$$

In (2.3), $\bar{\theta}_{current}$ is the current joint configuration of the robot and $\triangle\bar{\theta}$ is the incremental joint angle movement for the iteration. Following Buss (2004), we begin an iteration by finding the Jacobian matrix $\mathbf{J}$ as a function of $\bar{\theta}$ for the end effector.

$$\mathbf{J}(\bar{\theta}) = \frac{\partial \vec{s}}{\partial \bar{\theta}} \tag{2.4}$$

Define $[\mathbf{J}]_j = \frac{\partial \vec{s}}{\partial \theta_j}$ as the j^{th} column entry in the Jacobian for joint j. In the case of the PLIR, $\mathbf{J}$ is a 3 by 5 matrix. Given current values for $\vec{s}$, $\vec{t}$ and $\bar{\theta}$, we can calculate the Jacobian and try and find update values $\triangle\bar{\theta}$ for $\bar{\theta}$ such that

$$\bar{\theta}_{next} = \bar{\theta}_{current} + \triangle\bar{\theta} \tag{2.5}$$

The values for $\triangle\bar{\theta}$ are found using (2.8) discussed in Section 2.1.2.2. After each iteration, $\bar{\theta}_{current}$ takes on the value of $\bar{\theta}_{next}$, and this process is continued over a number of iterations until a solution for $\bar{\theta}_{current}$ is found that satisfies $\left\|\vec{t}(\bar{\theta}_t) - \vec{s}(\bar{\theta}_{current})\right\| < e_{solution}$. The value of the solution error tolerance, $e_{solution}$, is discussed in Section 5.1.2. One way in which $\triangle\theta$ can be found is to try and find the solution to

$$\vec{t} - \vec{s} = \vec{e} \approx \mathbf{J}\triangle\bar{\theta} \tag{2.6}$$

by solving for $\triangle\bar{\theta}$ in the form of $\triangle\bar{\theta} = \mathbf{J}^{\dagger}\vec{e}$, where $\mathbf{J}^{\dagger}$ is the pseudo-inverse of $\mathbf{J}$. However, if $\mathbf{J}$ is near singular, the solution for the joint angles will be overly sensitive to changes in $\vec{e}$.

2.1.2.1 Calculating the Jacobian

Following Buss (2004), Section 2, each $\mathbb{R}^3$ entry in the Jacobian $\frac{\partial \vec{s}}{\partial \theta_j}$ can be calculated by

$$\frac{\partial \vec{s}}{\partial \theta_j} = \vec{v_j} \times (\vec{s} - \vec{p_j}). \tag{2.7}$$

In (2.7), $\vec{p_j}$ is the position of the j^{th} joint and $\vec{v_j}$ is unit vector pointing along the axis of rotation of the j^{th} joint. The angles for $\bar{\theta}$ are in radians, and the direction of rotation is given by the right-hand rule. The part positions $\vec{p_j}$ are found using the forward kinematics solution discussed in Section 2.1.1.

2.1.2.2 Damped least squares

The Jacobian inverse method ($\triangle\bar{\theta} = \mathbf{J^{-1}}\vec{e}$) is not applicable for solving (2.6) since non-square matrices cannot be inverted. Based on Buss (2004), the pseudo-inverse method ($\triangle\bar{\theta} = \mathbf{J}^{\dagger}\vec{e}$) is

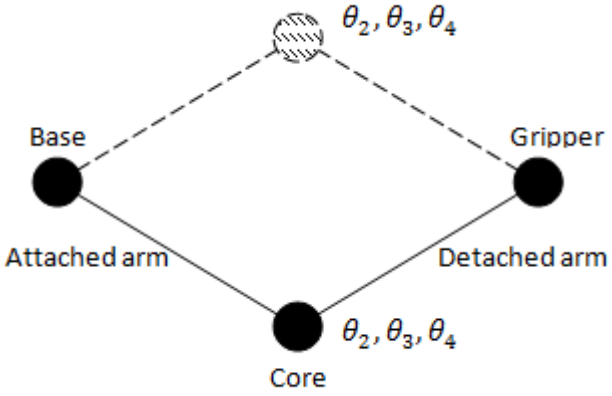

Figure 2.5: Non-unique solution for given end effector position.

not ideal for solving (2.6) either due to its ill conditioned behaviour near singularities. This so called "pathological" behaviour occurs when the matrix $\mathbf{J}$ is not of full rank. Currently the PLIR makes use of a damped least squares (DLS) algorithm. This is a method that behaves well near singularities. The DLS solution is taken from Section 5 of Buss (2004). The solution to $\triangle\bar{\theta}$ can be obtained from the following equation:

$$\triangle\bar{\theta} = (\mathbf{J}^T\mathbf{J} + \gamma^2\mathbf{I})^{-1}\mathbf{J}^T\vec{e} \tag{2.8}$$

where $\mathbf{I}$ is the identity matrix (in this application, a 5×5 matrix). Since $\mathbf{J}^T\mathbf{J}$ is non-negative definite, $(\mathbf{J}^T\mathbf{J} + \gamma^2\mathbf{I})$ is positive definite for $\gamma \neq 0$ and therefore has a positive determinant. This ensures that $(\mathbf{J}^T\mathbf{J} + \gamma^2\mathbf{I})$ always has an inverse. The damping constant γ must be made large enough to ensure that (2.8) is numerically robust. However, if the value for γ is too large, the convergence rate for the solution will be slow. The IK solution flowchart is given in Figure 2.6.

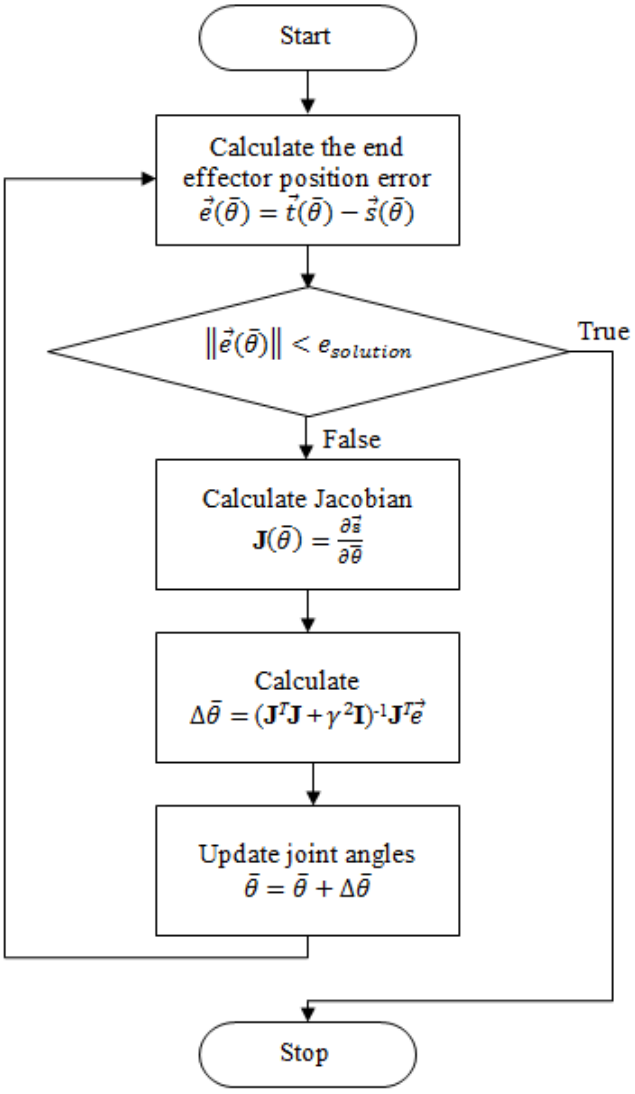

Figure 2.6: Flowchart of the iterative IK solution.

2.2 Robot Modelling

A model of the robot is required in order to predict how it will behave while executing a given trajectory. The model is used to determine whether or not the commanded trajectory causes the robot to exceed its designed performance limits as discussed in Section 2.3.4. The model is also used to predict the charge required to perform a manoeuvre so that the commanded trajectories can be optimised to minimise the cost (total charge) of the manoeuvre (discussed in Section 2.6).

2.2.1 Inverse dynamics model

The inverse dynamics of the robot reveals the relationship between the movement of the joints and the torque forces involved (Spong *et al.*, 2006). Once trajectories have been created for each of the robot's joints (discussed in Section 2.3), it is necessary to calculate the dynamics of the robot in order to find the torques required. In other words the task is to find the torque, $\tau(t)$, required to produce the desired trajectory $\theta(t)$ for a given joint (Spong *et al.*, 2006). Once $\tau(t)$ has been found for each joint, the torque profiles can be checked against the designed torque and current limitations of the robot, and appropriate trajectory time scaling can be performed if required (discussed in Section 2.3.4).

The inverse dynamics (ID) solution for the PLIR has already been implemented by Lorimer (2011), and is applied to the work of this dissertation. The method used to calculate the dynamics of the robot is known as the recursive Newton-Euler method. The torque solutions take into account the joint trajectory angular velocities, accelerations, gyroscopic effects and gravity, as well as the effect the links have on one another (Lorimer, 2011). The effect of gravity on the links provides the static torque experienced by each of the joints. The software implementation of the recursive Newton-Euler method is discussed in Section 5.1.2.

Although this model includes the coupling of joint torques from one joint to the next, it does not account for the effects of static, viscous and Coulomb friction, nor does it account for backlash in the actuator gearboxes. Initial comparison between this model and the actual robot revealed significant mismatch due to friction being omitted from the model. Over and above this, the model does not take into account the effects of having a closed loop control system controlling the position of the robot joints, and assumes perfect feed-forward control of the robot joints. Consequently, the charge consumption characteristics of the robot are not correctly predicted. For these reasons, a second model has been developed to capture the characteristics of the actual robot's behaviour in more detail. However, for the improved successor to this first robot prototype, it is expected that this model will be appropriate where the effect of friction is minimised due to superior actuator hardware.

2.2.2 Simulink model

The Simulink model of the robot comprises four discrete actuator models. These actuator models all have the same structure, but with slight variations of system parameters. This model allows for a more accurate representation of the real system, and includes friction, backlash and a closed-loop PID controller arrangement. The effect of static torque coupled between the joints is also modelled, but the coupling of the dynamic torque applied by the actuators is not. It was found during model tests that the dynamic torque coupling did not affect the behaviour of the robot significantly.

The Simulink model was created by including and modelling all of the robot's components at a fundamental level. The mechanical parameters for the actuator motors were found by conducting a set of system identification experiments using sine wave excitation and measuring the input to output behaviour of the plant. The other parameters such as static, viscous and Coulomb friction were also measured experimentally. The model parameters were then fine-tuned during comparative tests between the predicted and measured behaviour of the robot.

Since this model provides a high level of detail in its prediction of the actual system's behaviour compared to the ID model discussed in Section 2.2.1, it is more appropriate for the purpose of trajectory optimisation for this robot prototype. The design and implementation of the Simulink model is discussed in Chapter 5.

2.3 Trajectories

2.3.1 B-spline functions

In order for the PLIR end effector to get from one point to another in its work space, a trajectory for each joint is required. One can either create a Cartesian space trajectory for the robot end effector to get the desired motion, or one can create individual trajectories for each of the joints.

In the first case where a trajectory is used to control the end effector position, three sub trajectories would be required (for each of the x, y and z coordinate components). From these sub trajectories, IK can be used to translate a Cartesian coordinate in $\mathbb{R}^3$ into a set of joint angles in $\mathbb{R}^5$ to achieve the desired position for each trajectory element. If a trajectory is created for each of the joints, five trajectories would be required. The initial and final points of the end effector would be converted into initial and final joint angles for each joint using IK and from there, the trajectories would be generated.

The PLIR uses the second method for creating trajectories; i.e. trajectories are calculated for each joint. This is done because each of the joint actuators must be commanded with the desired angle at each instant of the trajectory. Therefore this approach avoids the on-the-fly IK calculations that would be required if a Cartesian space trajectory was used. Furthermore, the joint trajectories could be checked before they are executed on the robot to ensure they do not result in undesirable or impossible robot configurations.

In Biagiotti and Melchiorri (2008), many different methods for mathematically describing trajectories are available. Biagiotti and Melchiorri (2008) include polynomial trajectories of varying order, trigonometric trajectories of different types and trajectories based on Fourier series expansion. The authors also provide extensive information on multi-point trajectories using approaches such as trigonometric polynomials and cubic splines. These approaches are however not ideal for polynomials of degree higher than three. Instead one can make use of basis-spline functions to create trajectories with continuous jerk, and continuity at even higher derivatives.

Splines are piecewise polynomials that can be used to either interpolate data points or to approximate them. These splines can be efficiently computed based on basis-splines (B-splines), and the reason for this name is because the spline can be developed from a linear combination of basis functions of degree d, $(B_j^d(t))$ (*ibid.*). A B-spline curve for a given joint trajectory can be defined as

$$\theta(t) = \sum_{j=0}^{m} p_j B_j^d(t), \quad t_{wp_0} \leq t \leq t_{wp_n}. \tag{2.9}$$

In (2.9), p_j are the $(m+1)$ control points (elements of control vector $\bar{p}$) that define the curve $\theta(t)$ and can be defined for waypoint interpolation or approximation based on the waypoints given in the joint angle waypoint vector $\bar{\theta}_{wp}$. The construction of the control vector can be found in Biagiotti and Melchiorri (2008), Section 4.5. $B_j^d(t)$ represents the j^{th} basis function of degree d. Finally, t_{wp_0} and t_{wp_n} are the initial and final times respectively of the waypoint time vector $\bar{t}_{wp} = [t_{wp_0}, ..., t_{wp_n}]$, with each i^{th} element satisfying $t_{wp_i} \leq t_{wp_{i+1}}$; $(n+1)$ is the total number of waypoints.

Each $\bar{\theta}_{wp}$ is derived from the end effector waypoint matrix $\mathbf{s}_{wp}$ using IK. $\mathbf{s}_{wp}$ is an $(n+1) \times 3$ matrix and has Cartesian coordinate entries for each of the $(n+1)$ waypoints. Note that the number of control points $m+1$, and the number of waypoints $n+1$ are not equivalent (see Biagiotti and Melchiorri (2008), Section 4.5). Section 2.3.4 explains how the vector $\bar{t}_{wp}$ is calculated.

The trajectory joint angle at any given time instant within $\bar{t}_{wp}$ can be found using (2.9). The B-spline in (2.9) is defined as follows. Let $\bar{u} \in \{t_{wp_0}, ..., t_{wp_n}\}$ be a vector of time values known as knots. A knot vector for a trajectory of odd degree d has the form

$$\bar{u} = [\underbrace{t_{wp_0}, ..., t_{wp_0}}_{d+1}, t_{wp_1}, ..., t_{wp_i}, ..., t_{wp_{n-1}}, \underbrace{t_{wp_n}, ..., t_{wp_n}}_{d+1}], \tag{2.10}$$

where the number of t_{wp_0} and t_{wp_n} entries equals $d+1$. A knot vector for a trajectory of even degree d has the form

$$\bar{u} = [\underbrace{t_{wp_0}, ..., t_{wp_0}}_{d+1}, (t_{wp_0} + t_{wp_1})/2, ..., (t_{wp_{n-1}} + t_{wp_n})/2, \underbrace{t_{wp_n}, ..., t_{wp_n}}_{d+1}]. \tag{2.11}$$

The value of the j^{th} basis function of degree d at time t is then found recursively using (2.12) and (2.13).

$$B_j^0(t) = \begin{cases} 1, & \text{if } u_j \leq t < u_{j+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

$$B_j^d(t) = \frac{t - u_j}{u_{j+d} - u_j} B_j^{d-1}(t) + \frac{u_{j+d+1} - t}{u_{j+d+1} - u_{j+1}} B_{j+1}^{d-1}(t), \quad d > 0. \tag{2.13}$$

Using the incorrect knot vector definition for an even or odd degree polynomial results in trajectories that are not practicable (*ibid.*, Page 202, Example 4.17). The main properties of B-splines that apply to this project are (*ibid.*):

- $\theta(t)$ is differentiable infinitely many times within a knot interval, and can be continuously differentiated d times at a knot. So, for example, in order to obtain continuous $\frac{d^4\theta}{dt^4}$ (snap), $d = 5$.
- Time scaling of a B-spline curve can be done by scaling the knot vector $\bar{u}$.

The velocity, acceleration and further derivatives at a given time instant along the trajectory can be found by

$$\theta^{(i)}(t) = \sum_{j=0}^{m} p_j B_j^{d(i)}(t), \quad t_{wp_0} \leq t \leq t_{wp_n} \tag{2.14}$$

where $B_j^{d(i)}(t)$ represents the i^{th} derivative of the basis function $B_j^d(t)$. The definition of $B_j^{d(i)}$ can be found in (*ibid.*), Appendix B.1.

As mentioned before, a trajectory is developed for each of the joint actuators using the B-splines (or B-form) technique. By applying forward kinematics, these individual joint trajectories form a single end effector trajectory $\vec{s}(\bar{\theta}(t))$. In the case of a mechanical application, a B-spline curve of $d = 5$ is desirable in order to achieve continuous snap thus reducing stress on the components in the form of shock loading. The decision to use fifth order B-splines is supported by the literature review in Section 2.6 and the findings in Section 2.3.3.

2.3.2 Fitting method

When creating a B-spline trajectory, either waypoint interpolation or waypoint approximation can be used. Waypoint interpolation ensures that the waypoints are intersected by the trajectory. This allows for more precise position control of the end effector as it moves through the waypoints. This is shown in Figure 2.7.

The use of waypoint approximation does not allow for such strict waypoint-oriented control of the end effector as compared with the interpolation method since the trajectory only "approximates" the waypoints. This means that aside from the first and last waypoints, the end effector does not strictly pass through the waypoints (although they may) but rather forms an approximate fit to the control vector $\bar{p}$. This can be a problem in that the end effector may collide with the obstacle, even if the waypoints are outside of the obstacle space. This is shown in Figure 2.8.

Since obstacle avoidance for the PLIR is of primary importance, trajectories are created using waypoint interpolation to ensure the position command of the end effector passes through the desired points in space.

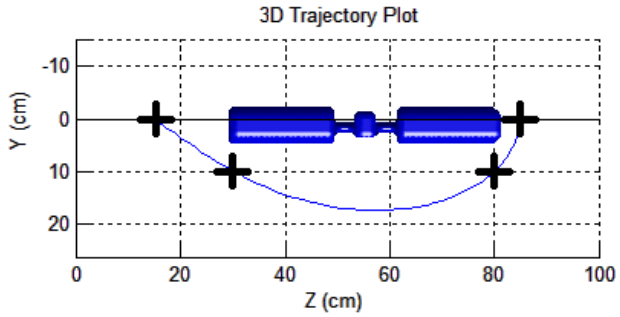

Figure 2.7: Top view of 3D trajectory plot using interpolation.

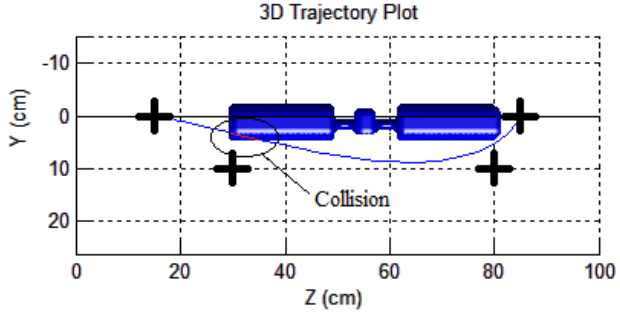

Figure 2.8: Top view of 3D trajectory plot using approximation.

2.3.3 B-spline order

Another attribute of the B-spline trajectory that needs to be set is the degree (order) of the B-spline fit. It is common practice to make use of cubic (3^{rd} order) splines for robotic manipulator trajectory fitting. The papers by Zhang and Sobh (2003) and Cao *et al.* (1994) show robot manipulator position control solutions using cubic polynomials.

Cubic splines allow for continuous acceleration, but not continuous jerk (derivative of acceleration). This may impact negatively on the hardware of the robot, particularly the actuators. To investigate the torque expected at the joint actuators, a cubic B-spline created using four waypoints was simulated with the PLIR model discussed in Section 2.2.1. The four waypoints used to create the end effector trajectory are given below in Cartesian coordinates (x, y, z):

1. $\vec{s} = (0,\ 0,\ 0.2)$
2. $\vec{s} = (0,\ 0.1,\ 0.4)$
3. $\vec{s} = (0,\ 0.1,\ 0.6)$
4. $\vec{s} = (0,\ 0,\ 0.8)$

Cubic B-spline trajectories are not very hardware-friendly due to the instantaneous non-zero acceleration/torque applied at $t = t_{wp_0} = 0\,\text{s}$ and $t = t_{wp_n}$ (Figure 2.9). In the real world, this translates to instantaneous current through the motors, which is not possible. The current will start off at zero and ramp to the initial value with a large gradient determined by the supply voltage and the actuator characteristics. As far as the hardware is concerned, this will still appear to be an instantaneous change in acceleration and as a result may cause damage to the gearboxes and other hardware due to the shock loading experienced.

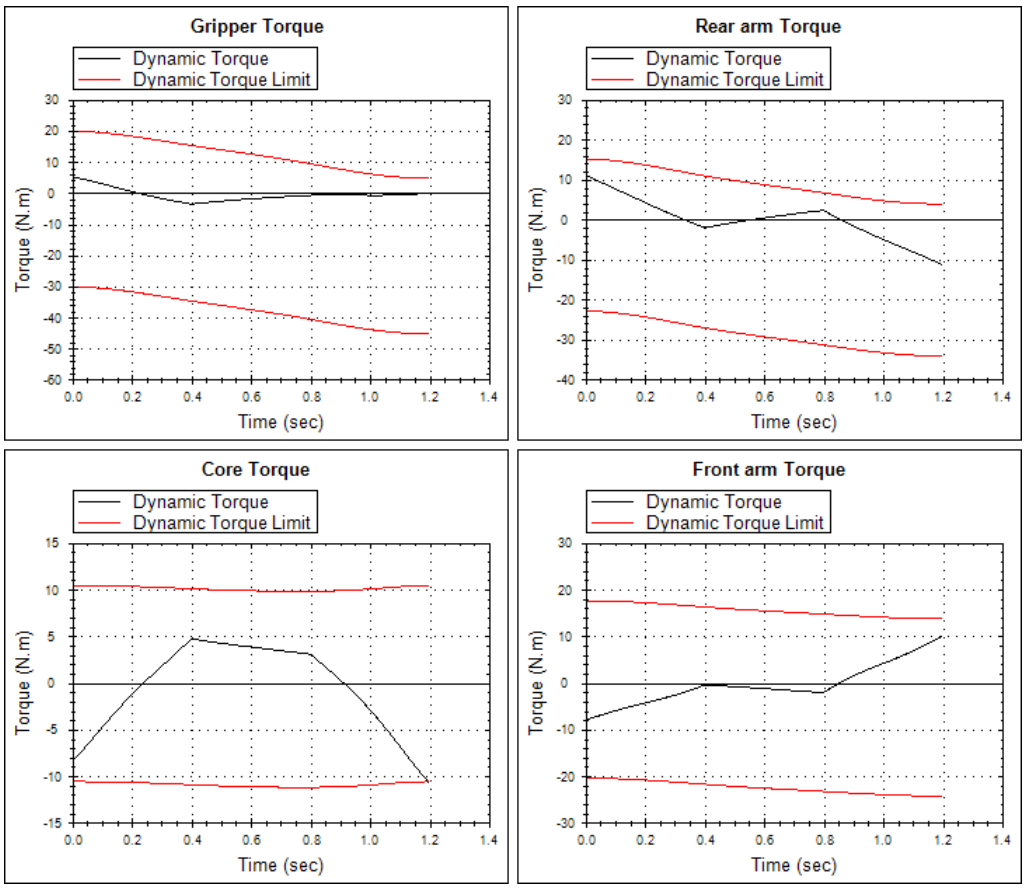

Figure 2.9: Torque graphs for third order waypoint approximation.

For a robotics application such as the PLIR, it is appropriate to create trajectories using fifth order B-splines. The simulated torque using the same waypoints above connected with a fifth order B-spline is shown in Figure 2.10. The mechanical shock loading at the start and end of a trajectory expected using cubic B-splines is no longer present when using fifth order B-splines. Smoother trajectories will reduce the risk of hardware failure, and increase the longevity of the robot's hardware. For this reason, fifth order B-spline trajectories are used to command the PLIR.

2.3.4 Time scaling

The time interval over which trajectories are executed must take the physical limitations of the PLIR into account. Time scaling is incorporated into the PLIR trajectory creation process to ensure that the physical limitations of the robot are not exceeded. Three limitations of the PLIR are taken into account: the torque limits of each robot joint, the angular speed limit of each joint actuator, and the peak instantaneous current that the power supply (battery) can deliver.

Before a joint trajectory can be generated and time scaling performed, the waypoint angle vector $\bar{\theta}_{wp_j}$ and corresponding waypoint time vector $\bar{t}'_{wp_j}$ must be set, and the time knot vector $\bar{u}'$ created. The superscript prime denotes a time element that has not been scaled according to the limitations of the robot. $\mathbf{s}_{wp}$ is a matrix containing each Cartesian coordinate end effector waypoint, and $\bar{\theta}_{wp_j}$ for each joint is generated from the waypoints in $\mathbf{s}_{wp}$ using IK as discussed earlier. The subscript j will be discarded from here on in this subsection for simplicity.

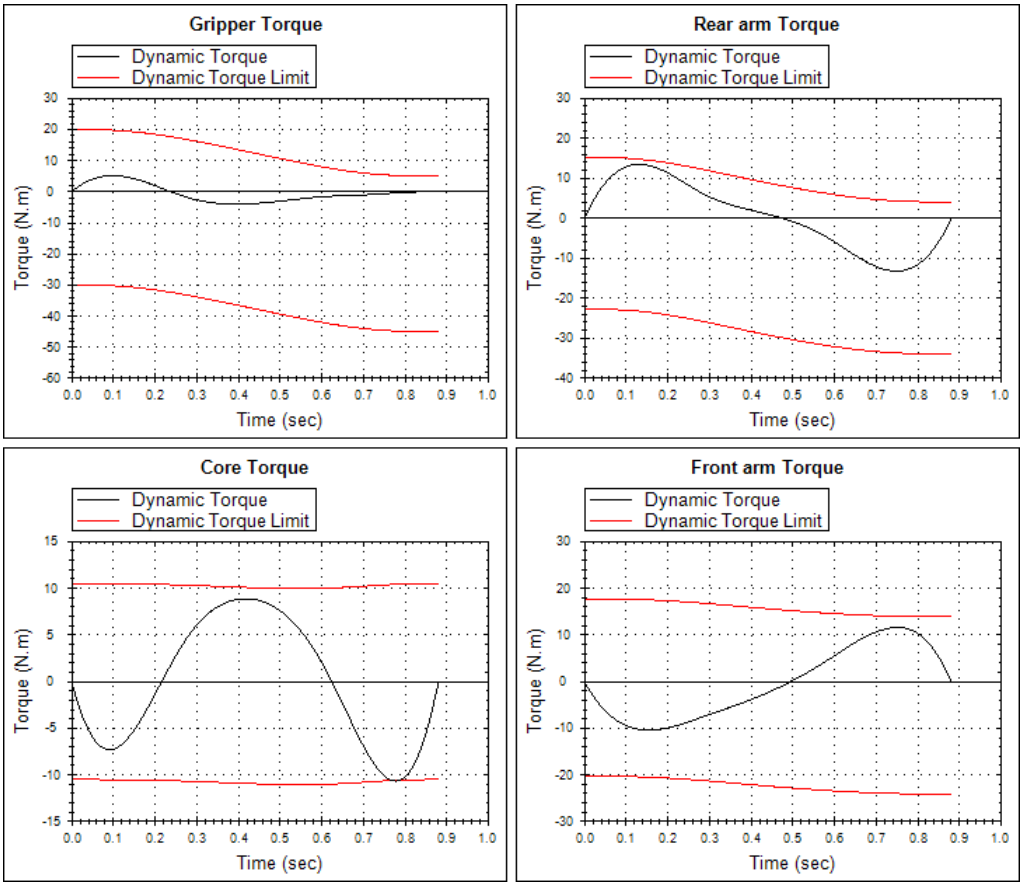

Figure 2.10: Torque graphs for 5th order waypoint approximation.

As a starting point, each time interval between waypoints is set to the same value, i.e. the time allocated to get from one waypoint to the next is uniform throughout. The final waypoint time, which is also the time interval over which the trajectory is executed, is t'_{wp_n}. The trajectory generating algorithm initially sets $t'_{wp_n} = \kappa$, a constant chosen to ensure that the limitations of the robot are exceeded for the initial trajectory solution at least at one point along the trajectory, based on the inverse dynamics model of the robot (discussed in Section 2.2.1). The selection of κ is discussed below. Following the initial solution, time scaling is performed on the time knot vector $\bar{u}'$ so that the limitations on torque, velocity and current for the robot are brought within the bounds of the robot's physical ability. The elements in $\bar{t}'_{wp}$ that are used to generate $\bar{u}'$ and subsequent trajectory are equispaced:

$$t'_{wp_i} = \frac{i}{n} \times \kappa, \tag{2.15}$$

where t'_{wp_i} is the i^{th} waypoint time vector entry in $\bar{t}'_{wp}$ and $n+1$ is the total number of waypoint vector entries which is equal to the number of waypoints.

The limits on torque, velocity and current for the robot will be exceeded if the trajectory time, κ, is sufficiently short. From this point, the algorithm can scale $\bar{u}'$, as mentioned before, to generate a feasible solution. However, because the trajectories and related dynamics of the robot are solved using a time step of $T = 15\,\text{ms}$, κ must be significantly larger than T for the solution to be found with sufficient accuracy. Experimentally, it was found that $\kappa = 100\,\text{ms}$ ensures that the robot's limits are exceeded on the first pass of the algorithm, and that T provides sufficient resolution to solve for the required scaling of $\bar{u}'$ and the subsequent trajectory and robot dynamics recalculation.

Once the trajectory and dynamics have been recalculated, a second check is done against the trajectory limits to ensure they are satisfied. If not, $\bar{u}$ is rescaled in an iterative manner until the limits are satisfied. However, only two passes are required in practice. A B-spline trajectory is generated for each of the joint actuators in this way. A flowchart for this algorithm is given in Figure 2.11.

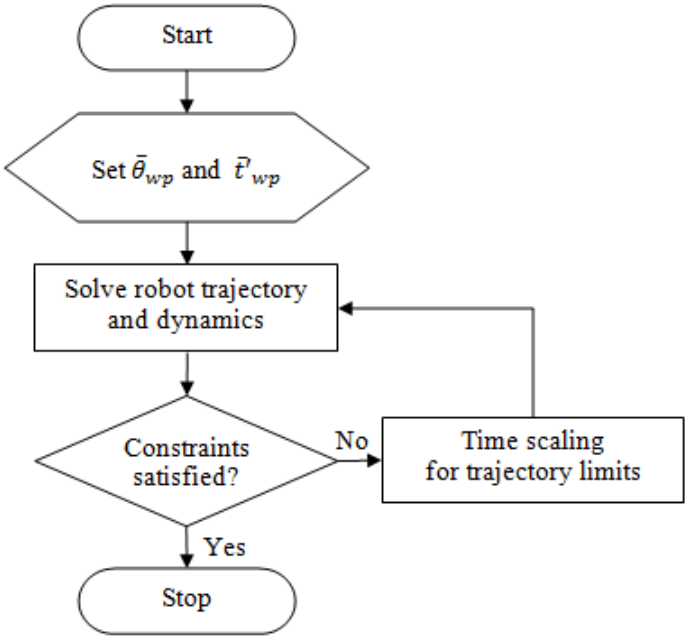

Figure 2.11: Flowchart of the trajectory solver algorithm.

2.3.4.1 Torque limiting

The joint torque limits imposed on the mechanical design in Lorimer (2011) have since been upgraded. The actuators used in Lorimer (2011) failed numerous times during testing and the gearboxes required frequent repair/replacement. This was because the actuator gearboxes could not endure the torque rating as stated in the manufacturer's data sheet. These actuators were replaced with units that are stronger (and larger). The torque ratings of the new actuators (tried and tested) are as follows:

- Gripper joint: 33 N.m
- Rear arm joint: 33 N.m
- Core joint: 10.5 N.m
- Front arm joint: 33 N.m

These torque limits are set based on the torque capabilities of the actuators that drive the revolute joints as well as the mechanical design of the links. In order to ensure reliable mechanical operation of the robot, these torque limits must not be exceeded. A joint's torque must be less than or equal to its corresponding torque limit at all stages of the manoeuvre: $|\tau_{joint}| = |\tau_{static} + \tau_{dynamic}| \leq |\tau_{limit}|$.

For a certain PLIR configuration, the static torque values and hence the dynamic torque limits must be calculated to find the maximum allowable dynamic torques that can be applied to prevent torque limit violation (Biagiotti and Melchiorri, 2008). The design of the PLIR is such that $|\tau_{static}| < |\tau_{limit}| \, \forall \, \bar{\theta}$ which ensures that dynamic torque can always be applied to effect movement (i.e. $|\tau_{dynamic}| > 0$).

Following Biagiotti and Melchiorri (2008), in order to achieve the required time scaling, one can use a linear scaling λ to translate the original time t'_{wp_n}, into a scaled time t_{wp_n} by the relationship

$$t_{wp_n} = \frac{t'_{wp_n}}{\lambda}. \tag{2.16}$$

The correct scaling factor $\frac{1}{\lambda}$ ensures the planned trajectory does not violate the PLIR's torque limits. From Biagiotti and Melchiorri (2008), a suitable choice for $\frac{1}{\lambda}$ is

$$\frac{1}{\lambda} = \sqrt{\frac{\left|\ddot{\theta}(t_{max})\right|_{max}}{\ddot{\theta}_{limit}}}, \tag{2.17}$$

where $\left|\ddot{\theta}(t_{max})\right|_{max}$ is the peak joint acceleration occurring at time t_{max}, and $\ddot{\theta}_{limit}$ is the joint acceleration limit. This is based on the acceleration profile for a given trajectory. There is a more appropriate form for the PLIR application that can be derived. Since $\tau = J\ddot{\theta}$, where τ is torque, J is inertia and $\ddot{\theta}$ is joint acceleration; substituting torques for accelerations in (2.17), $\frac{1}{\lambda}$ can be written as

$$\frac{1}{\lambda} = \sqrt{\frac{|\tau(t_{max})|_{max}}{\tau_{limit}}}, \tag{2.18}$$

where $|\tau(t_{max})|_{max}$ is the peak torque along along the entire trajectory, occurring at a time t_{max}, and τ_{limit} is the torque limit of that joint. Literature by Biagiotti and Melchiorri (2008) as well as Gasparetto and Zanotto (2007) make the assumption that the constraints for the dynamics of a trajectory are constant, regardless of joint positions. This is not the case with the PLIR where the dynamic torque limits vary based on the position of the end effector. This is because the static

torque experienced by the joints due to gravity varies as the joint angles change. For this reason, the method of time scaling is slightly different. Instead of using (2.18), $\frac{1}{\lambda}$ is found by

$$\frac{1}{\lambda} = \max_{j \in \{1,2,3,4\}} \left(\max_{t_{wp_0} \leq t \leq t_{wp_n}} \left(\sqrt{\frac{|\tau_{dynamic_j}(t)|}{\tau_{limit_j}(\theta(t))}} \right)_j \right), \tag{2.19}$$

where j represents the j^{th} joint of the robot (1 to 4 since line rotation is not controlled), $\tau_{dynamic_j}(t)$ and $\tau_{limit_j}(\theta(t))$ are the dynamic torque and the dynamic torque limit respectively of joint j at time t_{max}. The scaling factor $\frac{1}{\lambda}$ is found for each of the unscaled joint trajectories by searching for the largest limit violation in each one using (2.19). The initial value of $\frac{1}{\lambda}$ is always greater than 1, due to initial dynamic saturation ensured by κ. The knot vector $\bar{u}'$, which is the temporal component common to all the B-spline trajectories, is then scaled with the largest $\frac{1}{\lambda}$ of all the trajectories, i.e.

$$\bar{u} = \frac{\bar{u}'}{\lambda}. \tag{2.20}$$

Figure 2.12 shows the rear arm dynamic torque being limited for a sample trajectory, with the torque limit varying with the position of the end effector (to account for the changing static torque applied during the manoeuvre). The torque graph in Figure 2.12 is an output of the PLIR model simulation discussed in Section 2.4. The dynamic torque limit is generated by subtracting the static torque experienced by the joint from the designed total torque limit.

2.3.4.2 Angular speed limiting

When developing a model for the system, it is necessary to take the speed limits into account so that trajectories generated using the model do not request unrealistic performance from the robot. The Simulink model introduced in Section 2.2.2 does not require speed limiting since it is inherent in the actuator model. However, in the case of the inverse dynamics model discussed in Section 2.2.1, the following derivation is used to check the trajectory commands for speed violations.

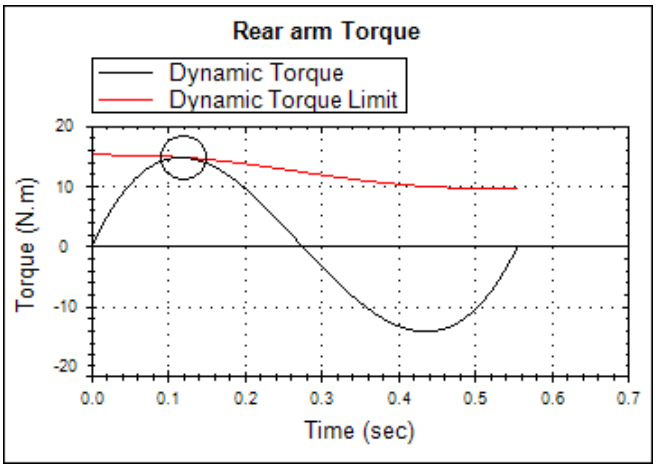

Figure 2.12: Rear arm torque graph showing torque optimisation for a simple manoeuvre.

The angular speed ($|\omega|$, in rad/s) for a given actuator can be derived from the approximate mathematical model for a DC motor (ignoring $L\frac{di}{dt}$):

$$|\omega| = \left|\frac{V - IR}{rk_\phi}\right|, \tag{2.21}$$

where V is the voltage applied to the actuator, I is the current drawn by the actuator, R is the actuator electrical resistance, k_ϕ is the back emf constant and r is the gearbox ratio. According to Lorimer (2011), the inductance of the actuator (L) was experimentally determined by measuring the electrical time constant of the motor and found to be in the order of 10^{-5} H and consequently neglected. The angular speed limit ($|\omega_{max}|$) occurs at $V = V_{max}$ where V_{max} is the maximum supply voltage to the motor. The torque for a DC motor is related to the current drawn by the motor by

$$\tau_{elec} = Ik_\phi, \tag{2.22}$$

where τ_{elec} is the torque produced by the motor. From (2.21), one can see that the angular speed limit for an actuator is dictated by the supply voltage and the current drawn by the motor to generate the required actuator torque. If the angular speed of the actuator at the output shaft of the gearbox is greater than the limit described in (2.21), a suitable value for $\frac{1}{\lambda}$ must be found to scale the knot vector $\bar{u}'$ as in (2.20).

Consider the angular speed limit violation depicted in Figure 2.13. The angular speed limit is dependent on current (each dependent on time), scaling the knot vector $\bar{u}'$ will cause the maximum angular speed of the joint (a) to be reduced towards some value c, and cause the angular speed limit (b) to increase towards the same value of c as depicted in Figure 2.13.

In order for the angular speed limit to be satisfied, both a and b have to move to some new value c so that $a = b = c$. To find a suitable value for $\frac{1}{\lambda}$, the value of c must be found and then $\frac{1}{\lambda}$ can be calculated as

$$\frac{1}{\lambda} = \frac{a}{c}. \tag{2.23}$$

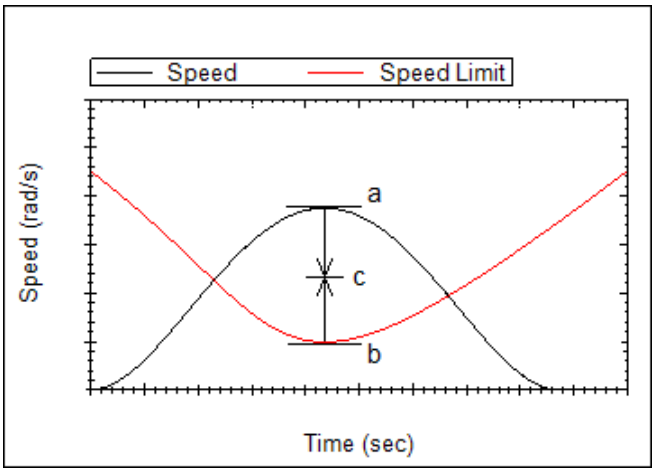

Figure 2.13: Speed limit violation.

In order to find c, it must be noted that as a moves down by an amount proportional to $\frac{1}{\lambda}$, b (dependent on actuator current) moves up by an amount proportional to $\left(\frac{1}{\lambda}\right)^2$. This is the same behaviour exhibited by the motor torque (Section 2.3.4.1) and is directly linked to the relationship between motor acceleration and time. Hence, one could alternatively calculate $\frac{1}{\lambda}$ as

$$\frac{1}{\lambda} = \sqrt{\frac{I_b}{I_c}}, \tag{2.24}$$

where I_b and I_c are the currents for the angular speed limits at b and c respectively. Now equating the right hand sides of (2.23) and (2.24) yields

$$\frac{a}{c} = \sqrt{\frac{I_b}{I_c}}. \tag{2.25}$$

From (2.21), one gets expressions for the currents:

$$\begin{aligned} |I_b| &= \tfrac{|V_{max}| - |b| r k_\phi}{R}, \\ |I_c| &= \tfrac{|V_{max}| - |c| r k_\phi}{R}. \end{aligned} \tag{2.26}$$

Combining (2.25) and (2.26), yields the quadratic formula

$$(|V_{max}| - |b|\, r k_\phi)c^2 + (a^2 r k_\phi)c + (-a^2 V_{max}) = 0. \tag{2.27}$$

Solving (2.27) for c using the quadratic formula gives

$$c = \frac{-a^2 r k_\phi \pm \sqrt{(a^2 r k_\phi + 4(|V_{max}| - |b|\, r k_\phi)(a^2\, |V_{max}|)}}{2(|V_{max}| - |b|\, r k_\phi)}. \tag{2.28}$$

The discriminant is non-negative due to $(|V_{max}| - |b|\, r k_\phi) \geq 0$ at all times, ensuring that the solutions to c are real. This is because from (2.21) it can be shown that $b r k_\phi$ has an upper limit of $V_{max} = b r k_\phi$ when $I = 0$. Division by zero is also avoided because $(|V_{max}| - |b|\, r k_\phi) > 0$ while the actuator is running with $I \neq 0$. Since $sign(V) = sign(I) = sign(a) = sign(b) = sign(c)$ at all times, all speed values are treated as positive such that the resulting problem is the same sign as that in Figure 2.13. The solution to c must therefore be given by the positive root. The positive root is

$$c = \frac{-a^2 r k_\phi + \sqrt{(a^2 r k_\phi + 4(|V_{max}| - |b|\, r k_\phi)(a^2\, |V_{max}|)}}{2(|V_{max}| - |b|\, r k_\phi)}. \tag{2.29}$$

Since $a = |\omega(t)|$, the value of $\frac{1}{\lambda}$ is given by

$$\frac{1}{\lambda} = \max_{j \in \{1,2,3,4\}} \left(\max_{t_{wp_0} \leq t \leq t_{wp_n}} \left(\frac{|\omega_j(t)|}{c_j(t)} \right)_j \right), \tag{2.30}$$

where j represents the j^{th} joint; $|\omega_j(t)|$ and $c_j(t)$ are the angular speed and value of c respectively for joint j at time t. The methodology of (2.30) is the same as that for (2.19). Finally, $\bar{u}'$ can be scaled using (2.20) to satisfy the velocity constraint.

2.3.4.3 Instantaneous current limiting

Limiting the instantaneous current draw of the robot is the simplest of the three limitations to be handled. The PLIR battery can supply a peak pulse current up to a specified magnitude. Current limiting simply requires that the sum of the absolute values of all actuator currents at any given instant does not exceed this battery specification. This limit is to protect the battery, and is unlikely to be reached under normal operation. The value for $\frac{1}{\lambda}$ needed to satisfy the current limit is

$$\frac{1}{\lambda} = \underset{t_{wp_0} \leq t \leq t_{wp_n}}{max} \left(\sqrt{\frac{I_{total}(t)}{I_{peak}}} \right), \tag{2.31}$$

where $I_{total}(t)$ is the total sum of the actuator and electronic component currents supplied by the battery at time t, and I_{peak} is the peak pulse current that the battery is capable of supplying. The time knot vector can then be scaled using (2.20).

2.4 Simulation

Simulation of the PLIR is necessary to predict the robot's performance. The trajectory solutions for the PLIR are used in conjunction with the robot models to form a simulated PLIR entity that can be used to generate feasible trajectories. The trajectory simulation data is used to predict the behaviour of the physical robot when performing a manoeuvre, and to ensure that the robot will operate within its designed limitations.

The simulated PLIR model is used to analyse all aspects of end effector trajectories covered in this dissertation. A trajectory, $\vec{s}(\bar{\theta}(t))$, is simulated in the following way:

1. Convert Cartesian end effector waypoints into joint angle waypoints using inverse kinematics.
2. Create trajectories for each joint using B-splines.
3. Use the trajectories to command PLIR model and find the total charge consumption.
4. Scale trajectory time if design limitations are violated.
5. Calculate the robot part positions at each point along the trajectory using forward kinematics.

The trajectories and model outputs are solved using a fixed time step of $T = 15\,\text{ms}$. This time step is the same as the physical controller's loop time of 15 ms. The trajectory simulation allows one to view many aspects of the calculated trajectory. It provides output of the following data: joint angles, velocities, accelerations, torques (dynamic and static) and currents; part positions; end effector path; trajectory time and charge consumption.

The part positions and end effector path are visualised in a 3D simulated environment, shown in Figure 2.14. This simulated 3D workspace environment is discretised into 1 cm intervals, yielding over 300 000 points in the robot's immediate workspace volume of $0.4\,\text{m} \times 0.8\,\text{m} \times 0.9\,\text{m}$ for the purpose of waypoint placement and obstacle descriptions with acceptable accuracy. Furthermore, the robot model is not restricted to move along the grid and is free to move in continuous space between waypoints. The 3D environment is graphed by exporting the part position and path data to a MATLAB plotting class created for C# using the MATLAB .NET Builder toolbox. The software implementation of the simulated PLIR is discussed in Section 5.1.2.8.

2.5 Obstacle avoidance

Obstacle avoidance in the scope of this PLIR project involves being able to manoeuvre around power line obstacles unassisted along a line span. It also involves the PLIR being able to move from one power line span to another, which requires navigation onto, along, and off of jumpers at pylons.

Up to this point, methods for IK and trajectory solutions have been discussed. These methods make use of given waypoints to generate desired end effector trajectories. The PLIR is able to get from one point to the next, but requires an algorithm to command it where to go in the first place. Obstacle avoidance in this context, also referred to as collision avoidance, focuses on generating end effector waypoints without human assistance. This dissertation makes a contribution towards the PLIR solution in terms of autonomous obstacle avoidance.

Many different approaches have been developed for generating paths for robot manipulators in order to avoid colliding with obstacles while navigating between two points. The different aspects of obstacle avoidance include obstacle description, analysis of the collision space of the obstacles, and the development of unobstructed paths within the free space.

2.5.1 Existing work

There are many existing path planning applications that involve collision avoidance. Those dealing with collision avoidance for robot manipulators are of specific interest and are outlined below.

2.5.1.1 Obstacle description

The PLIR first requires a description of work space obstacles in a form that can be utilised by a path planner. It is common to describe the obstacles in configuration space as in Bohlin and Kavraki (2000), or to describe the obstacles in the workspace directly as in Lin *et al.* (2005) and Wang *et al.* (2004). The configuration space of a robot manipulator is the set of all joint configurations that the robot can achieve. In configuration space representation, an obstacle is expressed as all configurations of the robot's joints that would result in a collision with the obstacle. The workspace of a robot manipulator is simply the 3D volume representing all possible locations that the end effector can reach. In workspace representation, an obstacle is expressed as the volume in Cartesian

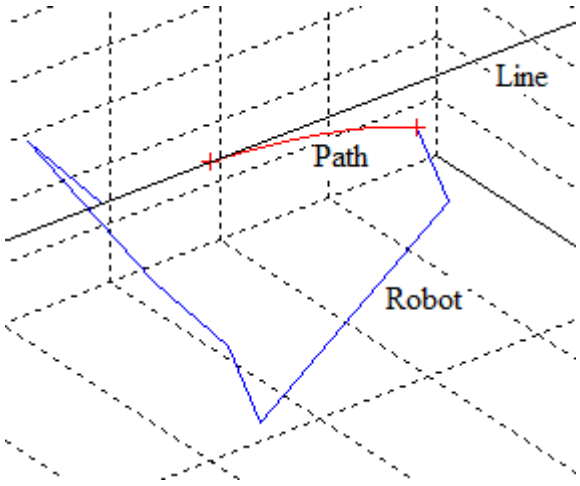

Figure 2.14: 3D simulated environment for the PLIR trajectory.

space that must not be intersected in order to avoid collision. Workspace representation is the preferred approach for obstacle avoidance in this dissertation. The obstacles exist in the workspace in reality and representing the obstacles in workspace makes visualisation of the problem easier. This will assist in the development of the obstacle avoidance algorithms needed for the robot. The configuration space of the PLIR joint angles would be in $\mathbb{R}^5$, and consequently difficult to visualise in application. As a result, configuration space obstacle avoidance methods would be difficult to analyse.

The obstacles in the workspace need to be defined in some way. The work of Zhang and Sobh (2003) makes use of spheres to describe the obstacles in workspace. This gives the obstacles a simple analytical description (Zhang and Sobh, 2003). While this will work well for obstacles that are approximately spherical in shape, it may not work well for other obstacles and will impact negatively on the perceived free space the robot is able to move in. Seshadri and Ghosh (1993) use regular polyhedra to represent obstacles in the workspace, which would be appropriate for obstacles that resemble prisms. Whatever the workspace obstacle may be, it is clearly important to model it appropriately to ensure that the free space seen by the path planner is a suitable representation of reality.

2.5.1.2 Path planning

There are numerous approaches that have been used in collision free path planning applications. The work of Yang and Meng (2001) makes use of neural networks to describe the workspace of the robot (mobile or manipulator) including the obstacles. The target of the path planner influences the neural network globally through neural activity propagation, attracting the robot to the goal, and the obstacles have a local repulsive effect on the robot. This approach is effective at finding collision-free paths between two points amongst obstacles, but it does not provide the waypoint information required to generate continuous trajectories. For this reason is not suitable for the obstacle avoidance application presented in this dissertation.

In Zhang and Sobh (2003), the initial and final configurations of the robot manipulator are known and an initial path is created assuming no obstacles exist in the workspace. Only once a path has been formed, does the path planner check for obstacle collisions. If collisions exist, the joint angles of the manipulator are adjusted. An algorithm is used to check for collisions and make the necessary adjustments in an analytical manner; a detailed description can be found in Zhang and Sobh (2003).

Another approach for path planning as applied by Bohlin and Kavraki (2000), uses a method known as probabilistic road map (PRM) planning. Their method is a slight variation on the traditional PRM method. They first create a road map using the known initial and final positions of the end-effector together with randomly generated nodes within the configuration space. A shortest path is found from the initial to the final position assuming no collisions ("Lazy PRM" (Bohlin and Kavraki, 2000)). Only once a path has been created are collision checks performed; then the road map is updated and the path is adjusted if necessary. A PRM approach was also used by Hrabar (2008) for a UAV (unmanned aerial vehicle) path planning solution in order to map the obstacle-free pathways through the robot's environment. The D* Lite path searching algorithm (used for finding shortest paths in a given graph) was then used on the PRM in order to find the shortest collision-free path from an initial to final location.

A PRM method is not desirable for the PLIR since it is not necessary to generate a road map first; the PLIR workspace is represented in a uniformly discretised manner. The 3D PLIR workspace

is discretised into a 3D graph with 1 cm intervals, and the obstacle space comprises a volume of these blocks in the workspace. Due to the limited reach of the robot, it is reasonable to represent the immediate workspace for obstacle avoidance in this way. The localisation of the obstacles in the case of the PLIR is well known (connected to the line). The PLIR will already have a clear description of the free space in which it can move (from a high-level image processing system), and a collision free path can easily be generated by adding waypoints that move the robot away from the obstacle. However, while the PRM approach may not be necessary, the use of a graph searching algorithm is applicable to the PLIR problem.

The D* Lite algorithm is not the best choice for finding shortest paths in uniform 3D graphs since it may not find the actual shortest path. This is because it finds a path by interconnecting nodes with only their neighbours, thus placing a restriction on the heading of a path from one node to the next (explained further in Section 3.2.1). The Lazy Theta* algorithm developed by Nash *et al.* (2010) however, does not restrict paths to be generated using a series of neighbouring nodes. Instead, it allows the interconnection of nodes whether they are neighbours or not. Lazy Theta* is designed to analytically find the shortest straight-line path between two nodes in a 3D graph whilst avoiding obstacles (filled graph blocks). Hence, the Lazy Theta* path finding solution is appropriate to the PLIR obstacle avoidance problem with the workspace and obstacle space being represented by a 3D graph.

2.5.1.3 Power line inspection robots

In the field of power line inspection robotics, numerous solutions exist for obstacle avoidance while moving along the power line. Further to this, some power line robots are able to manoeuvre around obstacles autonomously. A robot capable of autonomous obstacle avoidance manoeuvring is presented in Xiao *et al.* (2005), Xiao *et al.* (2008) and Wu *et al.* (2008). The authors show the obstacle avoidance procedure for known obstacles as a set of pre-programmed steps to be executed in a piece-wise fashion. The papers of Li *et al.* (2004) and Zhou *et al.* (2005) show how an expert system implemented using C Language Integrated Production System (CLIPS) can be used to govern obstacle avoidance manoeuvres. The expert system makes use of rule based movements that can be pieced together in order to get around known obstacles. A similar system used in Wang *et al.* (2006) and Sun *et al.* (2007) makes use of a high level inference planner based on sensory input together with a structural database of the entire power line infrastructure under inspection, including positions of the expected obstacles. Based on the estimated state of the robot (position on the line and position relative to the obstacles in the database), it combines basic movement elements to achieve obstacle avoidance.

These robots, together with others such as Cai *et al.* (2008) and Debenest and Guarnieri (2010) perform obstacle avoidance manoeuvres in a step-based manner. This is primarily due to the mechanical designs of the robots. In the case of the PLIR, such piecewise manoeuvres are not appropriate due to the robot's configuration. With one gripper detached from the line during a manoeuvre, the total torque experienced by an actuator is the sum of the static torque due to gravity and the dynamic torque providing the robot with the desired acceleration. Performing stop-start manoeuvres would require a gripper to be detached for longer periods of time, requiring the joint actuators to resist the force of gravity for longer than necessary. In order to counteract the static force of gravity, charge is required by the actuators to provide torque. Therefore, the robot must perform a continuous obstacle-passing manoeuvre that reduces the amount of time a gripper is detached from the line, and hence the amount of energy required to execute the manoeuvre.

2.5.2 Path planning implementation

The PLIR makes use of workspace obstacle descriptions in an attempt minimise computational effort (highlighted by Lin *et al.* (2005) and Wang *et al.* (2004)) and to allow for easier visualisation of the problem in 3D Cartesian space. Spheres have the benefit of simple mathematical representation as emphasised by Zhang and Sobh (2003) and only require a centre point and radius for their representation. They are appropriate for representing obstacles such as warning spheres on the line (unique to the overhead ground wires), but such obstacles are not of interest since they do not exist on the live lines that the PLIR is intended to work on (Lorimer, 2011). Rectangular prisms are suited to representing obstacles such as vibration dampers and insulators. So far, only rectangular prisms are used to represent obstacles in the workspace. The rectangular prism used is also known as an axis-aligned bounding box (AABB). The AABB is defined by Ericson (2005) as a rectangular six-sided box that has its face normals parallel to the axes of the given coordinate system. Since the AABB is axis-aligned, it requires only two points in $\mathbb{R}^3$ to be fully described. The explanation of this description is given in Section 3.1.

Within the 3D workspace graph, the 1 cm discrete blocks that are "filled" as obstacle space are described by the two-point sets in $\mathbb{R}^3$ as mentioned above. When the robot approaches an obstacle, it is intended that a high level image processing system of the PLIR will try to classify the obstacle. If the obstacle is successfully classified as a piece of known standard line hardware, a pre-calculated trajectory will be executed to get around the obstacle. If the obstacle is unknown, the image processing system will provide the trajectory planner with AABB information describing the obstacle. Based on this information, initial and final waypoints for obstacle avoidance can be set. The Lazy Theta* algorithm will therefore have a 3D graph description of the workspace and obstacles, and can find the shortest straight-line path from the initial to final waypoints. The nodes that make up the path found by the Lazy Theta* algorithm can be used as intermediate waypoints for the B-spline trajectory. This is done for both of the robot grippers so that the PLIR can get around the obstacle and continue along the line. The implementations of obstacle representation, collision detection and autonomous waypoint selection for the PLIR are discussed in Chapter 3.

2.6 Trajectory optimisation

2.6.1 Existing work

In the field of trajectory optimisation there are a number of approaches that have been pursued to optimise criteria such as time, energy, or time and energy for a given path. All such approaches have a common theme: minimise some "cost". The cost function for any optimisation must be chosen with the intended performance criteria in mind so that the desired effect of optimisation on these criteria can be realised. Papers covering the most relevant optimisation criteria as far as the PLIR is concerned, namely energy, time-energy, and jerk, are discussed below.

2.6.1.1 Energy minimisation

The principal behind energy minimisation is that in certain applications, the amount of energy consumed by a robot is more important than the time over which a particular trajectory is executed. This can be true in applications where robots, such as those in industry, consume large amounts of electrical energy. In such cases it makes sense to minimise energy consumption in order to bring

down electricity costs, as in Hirakawa and Kawamura (1997). Minimisation of energy consumption for the PLIR is necessary since the amount of energy available is limited due to it being battery powered, with limited charging from the line planned.

The work by Hirakawa and Kawamura (1997) aims to minimise the amount of electrical energy consumed for trajectories modelled using third order B-splines in the case of redundant manipulators. The so-called "variational approach" requires the adjusting of control points of the B-splines in position, velocity and acceleration subject to a performance index. The main criteria for the performance index is the position error between the desired end effector position and the actual position as the control points are varied. The second criteria for the performance index is the amount of electrical energy consumed. The impact of the energy criteria is governed by a weighting parameter. The results showed a significant reduction in the energy consumed.

Similarly in Martin and Bobrow (1997), the trajectories for a multi-link body are described using cubic B-splines. Again, the B-spline parameters are adjusted and optimised subject to the constraint that the end effector position follows a specified path. The optimisation goal in Martin and Bobrow (1997) is to minimise the sum of the torques experienced by each of the joints throughout a trajectory which the authors refer to as "minimum effort". This corresponds to minimisation of the sum of the currents required for each actuator of the joints.

The above approaches to minimising energy and effort for robot trajectories do not apply specifically to the PLIR. The main reason for this is that there is no redundancy in terms of joint positions with the PLIR; if it is to follow the specified path, the joint position parameters will have to remain fixed. Since the original end effector paths are generated by a high level path planner that compensates for obstacles, adjusting joint trajectories in terms of position would cause the end effector to follow a different path altogether and possibly result in a collision with an obstacle. However, the above approaches do reveal that the way in which joints are moved relative to one another does have an impact on the total energy cost of the trajectory. In order for the PLIR to benefit from this observation, the end effector would have to move along a different path from the one originally specified. This could be advantageous when the position of the end effector is not required to strictly follow a given path.

A different approach to energy minimisation is found in Bailó *et al.* (2010) where eighth degree polynomials are used to describe a certain trajectory for a six-DOF robot. Minimisation of the energy required by the trajectory with a specified time interval is achieved by searching for the optimal maximum trajectory velocity along the path. The maximum velocity is the parameter that is adjusted until the energy cost is minimised. For a given time interval, setting a certain velocity limit will have an effect on the shape of the velocity profile. The shape of the velocity profile will determine the behaviour of the trajectory, and hence the amount of energy that is required to perform the manoeuvre. So it makes sense that by optimising the shape of the velocity profile as a result of varying the velocity limit, the energy consumption can be minimised. Bailó *et al.* (2010) report that a genetic algorithm takes 100 generated trajectories with unique values for maximum velocity and calculates the energy cost of each trajectory by solving the cost function. The genetic algorithm then continues to generate and mutate population members until some convergence criterion is reached. Although the method for optimisation is significantly different to the work of Hirakawa and Kawamura (1997) and Martin and Bobrow (1997), the fundamental concept of optimisation by modifying trajectory parameters remains the same.

2.6.1.2 Time-energy minimisation

Another possible performance optimisation choice is that of time-energy, which lies between the two extremes of minimum energy optimisation and minimum time optimisation. With time-energy optimisation, the cost function is a weighted sum of the total time taken for a trajectory and the total energy required for the trajectory. By varying the weighting factors, one can find the ideal balance between time and energy optimisation for a particular application.

The time-energy optimisation by Saramago and Steffen (1998) describes trajectories using cubic splines. The trajectories are optimised by adjusting the interval times between the time values in the spline knot vector using sequential unconstrained minimisation performed by the Davidon-Fletcher-Powell method. The cost function for the optimiser is described as the weighted sum of the trajectory time and the trajectory energy subject to constraints on velocity, acceleration, jerk and torque.

The optimisation of trajectory time knots is the concept that is used in this dissertation in Chapter 4. However, the constraints used by Saramago and Steffen (1998) are constant, unlike the dynamic constraints of the PLIR as shown in Section 2.3.4. The other similarity between Saramago and Steffen (1998) and the optimisation used for the PLIR trajectories is the optimisation scheme; the PLIR uses an iterative approach to minimise the cost function by adjusting the B-spline time knot parameters, recalculating the model dynamics and solving the cost function at each iteration. The optimisation scheme for the PLIR is discussed in detail in Chapter 4.

The work by Shiller (1994) highlights the benefits of time-energy optimal control over time optimal control. They found that the smoother time-energy optimal solution to a given trajectory produced a smaller tracking error and resulted in better transient responses when used as input to PD position controllers. While this is not surprising since they are two different performance criteria, it is noted that a slight increase in trajectory time for a time-energy optimal solution compared with a pure time optimal one is insignificant compared to the improvement in energy efficiency. Another benefit of having a smoother trajectory, as pointed out by Shiller (1994), is that the currents to achieve the desired motion will be lower, thus prolonging the life of the robot actuators. Of course if one requires the trajectory to take the least amount of time possible, then time optimisation is best. However, these findings present a strong case for having a trade off between time minimisation and energy minimisation without sacrificing significant time performance.

2.6.1.3 Jerk minimisation

According to Gasparetto and Zanotto (2007), the main reasons for $\dddot{\theta}$ (jerk) minimisation of a trajectory are better path tracking, reduced stress on the actuators and the robot, and that the resonant frequencies of the robot are less likely to be excited. The approach to optimisation in Gasparetto and Zanotto (2007) is similar to that of Saramago and Steffen (1998) in that the time knot vector for the trajectory is optimised. In Gasparetto and Zanotto (2007), fifth order B-splines are used to describe the required trajectories for the six-joint robot, and the cost function is based on the B-spline formulation. It is a weighted sum of the trajectory time and the integral of the jerk squared, subject to velocity, acceleration and jerk constraints (similar to Saramago and Steffen (1998)).

The work of Piazzi and Visioli (2000) also uses the time knot vector (of a cubic spline in this case) as the optimisation variable in order to minimise the global jerk. Once again, the authors highlight the benefit gained in position tracking of the end effector.

2.6.2 Optimisation objectives

There are two components of a smooth polynomial trajectory that can be adjusted when it comes to optimisation for a particular cost function; the temporal component and the spatial component. The optimisation problem facing the PLIR is to minimise the total charge required for the trajectories that are executed. The reason for this is because the PLIR is powered by a battery, and not by a "limitless" power source such as for robots in manufacturing applications. It is in the best interest of the PLIR application to save as much charge as possible so that it can perform inspection missions for longer periods of time. In order to realise minimum charge consumption, both the manoeuvre path and the execution time would need to be optimised.

In the case of spatial optimisation, assuming the initial and final waypoints are given and fixed, any intermediate waypoints can be varied in their placement in an attempt to yield a lower cost trajectory subject to obstacle avoidance constraints. These intermediate waypoints can be placed in a configuration that is one of many possible positions within three dimensional space.

For example, consider a trajectory with fixed initial and final waypoints and a single intermediate waypoint to be placed somewhere within a $20\,\text{cm} \times 20\,\text{cm} \times 20\,\text{cm}$ volume in the PLIR workspace. With the workspace discretised at 1 cm intervals (for example), there are 9261 possible positions for that intermediate waypoint. As the number of intermediate waypoints increases, the number of possible configurations increases by the addition of an extra dimension, and it may become difficult for an optimiser to find a global minimum.

Adjusting the location of the intermediate waypoints influences the amount of charge required to get the end effector from the initial waypoint to the final waypoint, and waypoints can be placed to minimise charge subject to avoiding obstacles and other constraints. However, since there is no known analytical relationship between trajectory charge cost and the actual robot path taken, analytical minimisation cannot be done. Hence, a numerical method such as the Nelder-Mead simplex method must be used to find the optimal paths.

Investigating the validity of using such a method requires analyses of representative optimisation search spaces. Since the search spaces based on spatial optimisation are of order three (with just one intermediate waypoint) or higher, analysis is cumbersome. For these reasons, these analyses are not documented in this dissertation. However, investigations that have been done into the relationship between waypoint placement and charge consumption show that placing waypoints to produce shorter paths results in lower charge costs. Further to this, the shortest waypoint path approach results in sensible obstacle avoidance paths being generated.

In Section 2.3.4, trajectories are created with a B-spline time knot vector that has entries spaced at uniform intervals. Since the intervals between each waypoint are different in terms of space and constraints, the optimal times required to get from one waypoint to the next could be different. Optimisation can be performed on the temporal trajectory components (waypoint interval times) in order to minimise the amount of charge required for the trajectory.

In the case of adjusting trajectory interval times between waypoints, the only dimension to be adjusted is time as opposed to three dimensions in position. This means that an optimisation search space can be analysed visually for a set of three waypoints; enough to perform a simple obstacle avoidance manoeuvre. Therefore, for the purpose of this dissertation, optimisation is performed on the temporal components for a given fifth order B-spline trajectory that already satisfies spatial requirements.

The charge required for a trajectory does not only include the charge consumed by the actuators during a trajectory, as the charge consumed by the other electronic components of the robot must also be considered (including the microcontrollers and the on-board computer required to perform the control of the trajectory). So if one assumes that the current drawn by all of the electronic components (excluding the actuators) averages some value, I_C, and that all resources of these components are dedicated to trajectory control during a manoeuvre, it is necessary that I_C is integrated into a cost function along with the currents required for the actuators for a given trajectory. The optimisation objective can be defined as the minimisation of this cost function.

The trajectories of the PLIR are described using B-splines (Section 2.3.1), and the cost function can be minimised by optimising their temporal components; the elements of the waypoint time vector $\bar{t}_{wp} = [t_{wp_0}, ..., t_{wp_n}]$ from which $\bar{u}$ is created. The cost function, denoted by $J(\bar{t}_{wp})$, can be defined as

$$J(\bar{t}_{wp}) = \int_{t_{wp0}}^{t_{wp_n}} I_{total}(t)\, dt \tag{2.32}$$

where $I_{total}(t)$ is the total current supplied by the battery at time t. Due to the discretised nature of the trajectory solver, the cost cannot be calculated in the continuous-time domain. Therefore, a discrete-time implementation of $J(\bar{t}_{wp})$ can be defined as

$$J(\bar{t}_{wp}) = \sum_{i=1}^{\ell} \frac{I(t_i) + I(t_{i-1})}{2} T + \sum_{i=1}^{\ell} I_C T \tag{2.33}$$

where t_i are the discrete trajectory times with $t_i - t_{i-1} > 0$, ℓ is the total number of solver time steps of size T over the trajectory time t_{wp_n} and $I(t_i)$ is the total current supplied by the battery to all of the actuators at trajectory time t_i. The cost function in (2.33) is minimised subject to the constraints:

$$\begin{gathered} \dot{\theta}_j(t_i) \leq \dot{\theta}_{limit_j}(t_i) \quad \forall t_i : 0 \leq i \leq \ell,\, j = 1, ..., 4 \\ \tau_j(t_i) \leq \tau_{limit_j} \quad \forall t_i : 0 \leq i \leq \ell,\, j = 1, ..., 4 \\ I(t_i) + I_C \leq I_{peak} \quad \forall t_i : 0 \leq i \leq \ell \end{gathered} \tag{2.34}$$

For joint j: $\dot{\theta}_j$ is the velocity, $\dot{\theta}_{limit_j}$ is the velocity limit at time t_i, τ_j is the total torque and τ_{limit_j} is the total torque limit. I_{peak} is the maximum pulse current that can be provided by the battery. The limits required in (2.34) are as discussed in Section 2.3.4.

There are two extremes to the effect that I_C has on the optimisation. If $I_C \to 0$, the cost function will be dominated by the energy required by the actuators and this will effectively become an energy minimisation problem. On the other hand if $I_C \gg I(t_i) \quad \forall\, t_i : 0 \leq i \leq \ell$, the cost function will be dominated by I_C and the optimisation problem will become one of time minimisation since total charge will be proportional to time. An approximate value of $I_C \approx 0.75\,\text{A}$ has been measured. This means the trajectory optimisation problem is effectively one with a time-energy minimisation goal since the magnitude of $I(t_i)$ is usually under 5 A (based on experimental results presented in Chapter 6).

The trajectories being implemented using fifth order B-splines together with time-energy optimisation will have added advantages of reducing the stress on the actuators by requiring smaller jerks and improving tracking of the desired path. The implementation of time-energy optimisation for a power line inspection robot is discussed in Chapter 4, and is a primary contribution of this dissertation.

Chapter 3

Obstacle Avoidance

Obstacle avoidance is necessary for the PLIR to be able to move through its environment: the power line infrastructure. There is a variety of obstacles that will be encountered by the PLIR in the field. In order to get around these obstacles without human assistance, autonomous waypoint selection is required so that the robot can find its own way around. These waypoints are used to create obstacle avoidance trajectories as in Section 2.3. Most obstacles, can be easily represented using axis-aligned bounding boxes (AABBs). In some cases, it may be preferable to model an obstacle using a number of these boxes as "sub-components" of the model. Based on the obstacle description, decisions can be made by the autonomous waypoint selection system in order to get around it. It is important that obstacle descriptions are appropriate; the obstacle avoidance capability of the autonomous waypoint selection system is highly dependent on how a given obstacle is described. If an obstacle is described in a way that wastes a large amount of free-space, the autonomous waypoint selection system may not find a collision-free path around it. This may not have been the case if the obstacle was described in a more appropriate way.

For the set of known obstacles that are expected on a power line, obstacle representations and optimal collision-free trajectories will be pre-calculated and stored in the PLIR's on-board computer. It will be the task of a high-level image processing system to detect and classify these known obstacles, so that the PLIR can move into position and carry out the pre-programmed end effector trajectories to get around them. If the image processing system detects an obstacle that it cannot classify, the autonomous waypoint selection system will have to generate a customised obstacle avoidance solution. The image processing system will have to measure the obstacle, extract features, and then use an algorithm to fit AABBs around the obstacle (not in the scope of this dissertation) so that the autonomous waypoint selection system can take over.

The PLIR's workspace environment is represented by a 3D graph discretised at 1 cm intervals as mentioned in Section 2.4. This graph is used to describe the location of obstacles within the PLIR's workspace, and waypoint and obstacle positioning is restricted to graph nodes. For the purpose of obstacle avoidance, the PLIR is represented as a set of points connected with line segments. The robot representation is shown in Figure 3.1. The points are the joints/grippers and the line segments represent the robot links. Representing the robot as a series of points and lines makes collision detection and avoidance computationally simple.

The simplification of the robot representation means that the AABBs must take into account the size of the robot parts, mainly the robot gripper. With a gripper fully opened, the dashed line intersecting the two gripper wheels shown in Figure 3.2 is parallel with the x-axis. In this

configuration the dimensions of the gripper are 22 cm × 6 cm × 10 cm (x, y, z). The two gripper points at each end of the PLIR representation in Figure 3.1 correspond to the "centres" of the physical grippers. These "centres" exist where a line intersecting the centre point of each gripper wheel intersects the centre line of the gripper actuator as shown in Figure 3.2. In order to account for this, the AABB description must be enlarged by 11 cm along the x-axis, 3 cm along the y-axis and 5 cm along the z-axis in both the positive and negative directions of each axis. Over and above this, extra allowance is made so that the robot can pass the obstacle with a few centimetres of clearance.

3.1 Axis-aligned bounding box description

An AABB is a simple obstacle model with which objects can be defined. It can be used to model a complex obstacle that is relatively uniform about at least one axis, such as the z-axiz (along the line) in the case of a vibration damper. Instead of modelling an obstacle in a complex way, the PLIR autonomous waypoint selection system can model it as an AABB. For example, the vibration damper (modelled in relatively high detail in Section 2.3.2) can be modelled as an AABB. The dimensions and orientation of the box depend on the obstacle itself, and these parameters are chosen in order to appropriately enclose the obstacle. Figure 3.3 shows a vibration damper and the corresponding AABB.

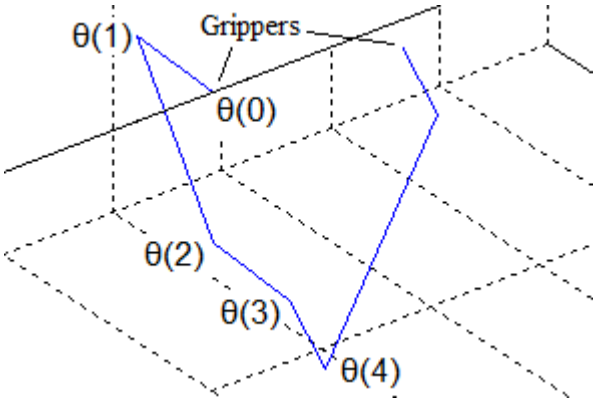

Figure 3.1: PLIR simulation model representation with joint angle labels.

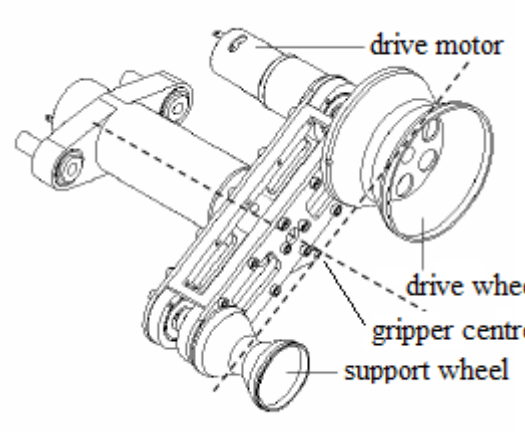

Figure 3.2: Figure showing the gripper "centre" point (Lorimer, 2011).

In this case, the AABB provides for a sufficient description of the vibration damper in order to generate waypoints for obstacle avoidance. It is not necessary for the obstacle to be described in more detail.

Figure 3.4 shows a worn Stockbridge damper on a power line. Such an obstacle can be appropriately modelled using two AABBs as "sub-components". A suitable representation for the damper in Figure 3.4 is shown in Figure 3.5.

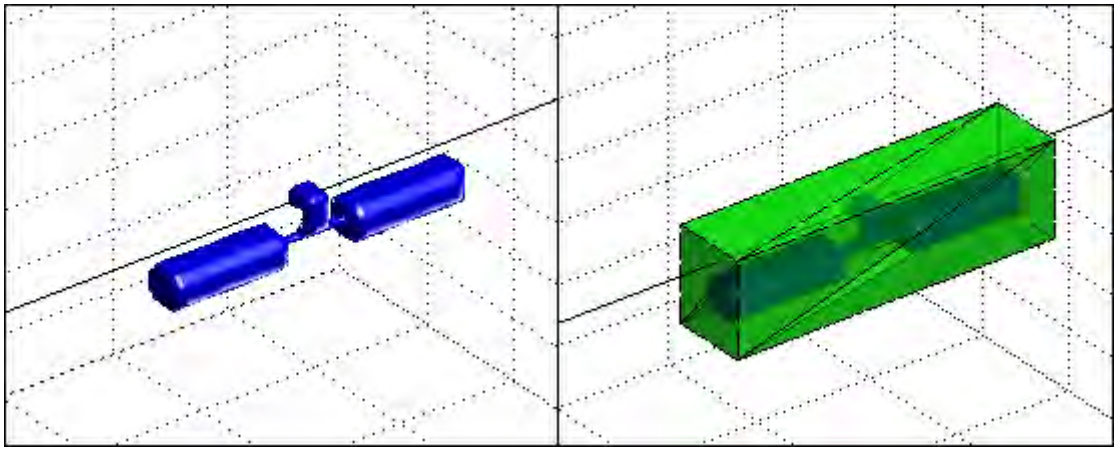

Figure 3.3: Figure showing complex model of vibration damper (left), and simple AABB model of vibration damper (right).

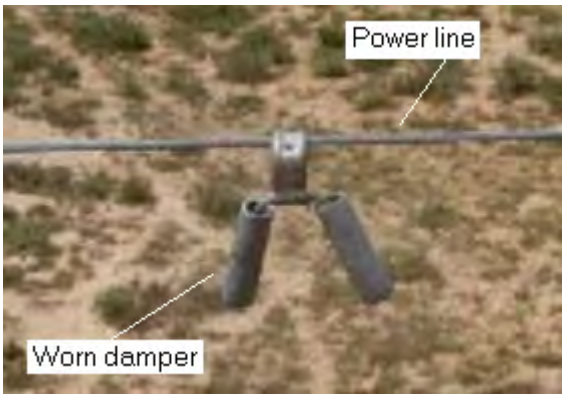

Figure 3.4: Photograph of a worn Stockbridge damper on a power line (Retief, No Date).

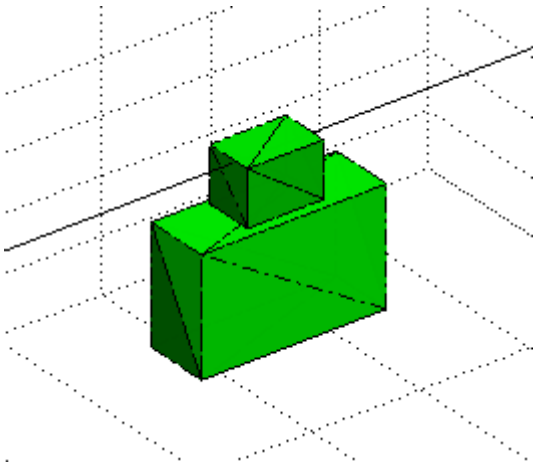

Figure 3.5: Figure showing multi-component representation of a worn Stockbridge damper.

Each AABB can be described using a pair of Cartesian points in $\mathbb{R}^3$. These points are referred to as the "minimum" and "maximum" coordinate values along the axes (Ericson, 2005), here referred to as boundaries (Figure 3.6). A similar concept for this type of boundary description can be found in Schouwenaars *et al.* (2001), Section 3.

3.2 Implementation

The PLIR traverses a power line with both grippers attached to the line. If an obstacle is detected, the PLIR must perform an obstacle avoidance manoeuvre in order to get around it and continue along the line. This requires each gripper to be removed from the line and reattached in turn to get around the obstacle. However, autonomous waypoint selection does not consider the actual detachment and reattachment procedures of the grippers for the time being. Due to the simplified robot representation as a set of points and lines, the grippers simply leave and join the power line as needed.

Although the PLIR has motorised wheels (rollers) attached to each gripper that allow the robot to traverse a power line, they are not used during obstacle avoidance manoeuvres at this stage. This is in order to simplify the problem at hand. The autonomous waypoint selection system creates obstacle avoidance trajectories based purely on the robot's ability to reach around an obstacle. In the case of "long" obstacles, the AABB description length (along the power line's z-axis) could be shortened if the robot is driven forwards during the obstacle avoidance manoeuvre.

For the creation of each gripper trajectory, it is assumed that a high-level image processing system has provided the necessary information about the obstacle in the way. It is also assumed that the PLIR is instructed by the image processing system to stop an appropriate distance from the obstacle. Once the PLIR has stopped, an obstacle avoidance manoeuvre needs to be carried out so that the PLIR can get around it.

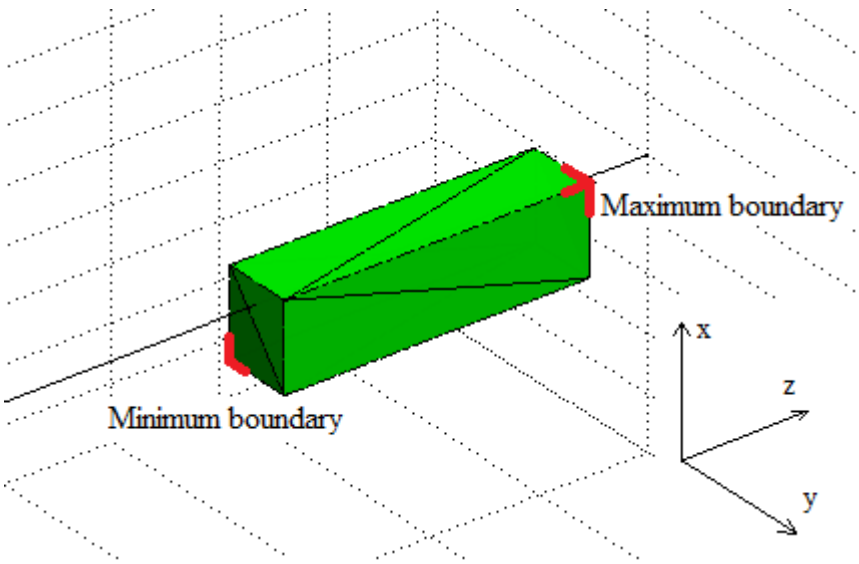

Figure 3.6: Figure showing the positions of the minimum and maximum boundaries of the AABB.

3.2.1 Waypoint selection

In order for a trajectory to be created, a set of waypoints is required as discussed in Section 2.3. For obstacles on a power line segment (such as dampers), the initial and final waypoints are assumed to be collinear on the power line's z-axis. Furthermore, the power line segment that is considered over the range of the manoeuvre is assumed to be straight. This assumption is supported by Figure 3.4 where the line appears to be straight over that particular segment.

Once the robot has stopped before the obstacle, the positions at which the front and rear gripper have come to rest are used as their initial waypoints. The simulated model has the robot come to rest 3 cm from the start of the obstacle representation. The rear gripper is 15 cm behind that since that is the closest the two grippers can be to one another in order to provide maximum reach. The final waypoint for the front gripper must be 18 cm past the far side of the obstacle representation, allowing the rear gripper to re-attach 3 cm clear of the far side of the obstacle representation. These measurements have been set experimentally and are found to yield desirable trajectories. As an example, the initial and final waypoint locations for the vibration damper representation are shown in Figure 3.7.

For obstacles that require initial and final waypoints that are not collinear (in the case of a jumper, shown in Chapter 6), the image processing system has to provide the initial and final waypoints explicitly (beyond the scope of this dissertation). Once the initial and final waypoints have been set, intermediate waypoints must be chosen so that a collision-free path can be created around the obstacle. There are many different paths that a gripper can take to get around a given obstacle. It is preferable to choose the path that requires the least amount of electric charge for the manoeuvre due to power supply constraints. However, since there is no analytical method for finding such a path due to the highly coupled nature of the robot's forward dynamics solution, it is sensible to find the "shortest distance" path and optimise the trajectory time instead. It is observed in simulation experiments that this will be close to the performance of the "minimum charge" path in terms of charge consumption.

The shortest path in the 3D workspace graph around an obstacle from a start node to an end node can be found using the Lazy Theta* algorithm (Nash *et al.*, 2010). The Lazy Theta* algorithm

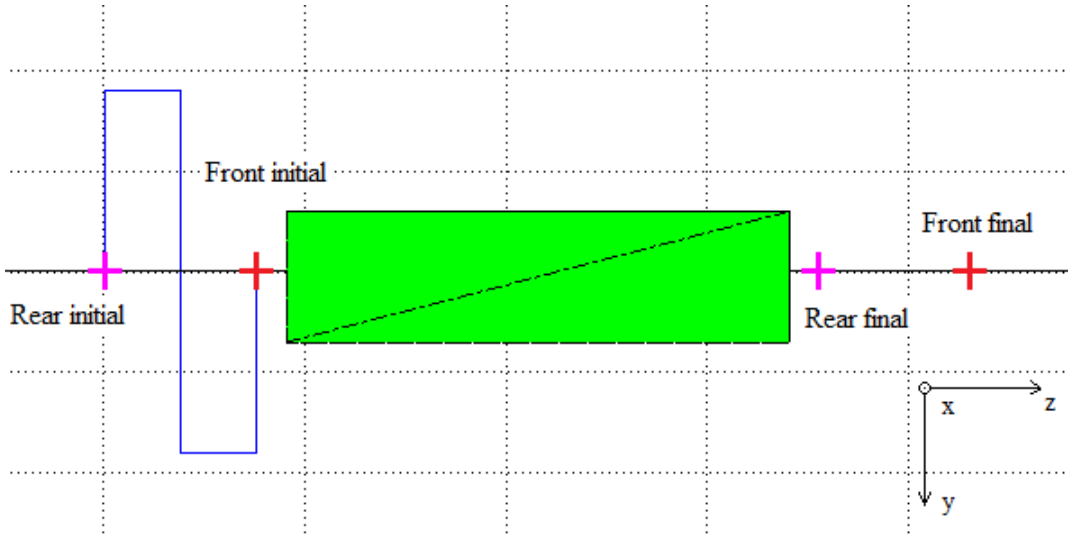

Figure 3.7: Figure showing the top view of the initial and final waypoints in the case of a vibration damper.

is based on the popular A* algorithm used for finding shortest paths between two graph nodes comprising of graph edges. Being restricted to graph edges is not desirable for the PLIR application, since a path confined to graph edges may not actually be the shortest path. A* finds the shortest path between two nodes by propagating path information along graph edges, and then linking the edges between the two nodes that result in the shortest path.

Unlike A*, Lazy Theta* does not restrict the path itself to graph edges, it allows path segments to join any two nodes that can make up part of the shortest distance path as long as they have line-of-sight of each other (no obstacles between them). The initial and final waypoints for an obstacle avoidance manoeuvre will be selected by the high-level automation system. The nodes generated by the Lazy Theta* algorithm are used as the intermediate waypoints between the initial and final waypoints, all of which are used to create the trajectory. A comparison between A* and Lazy Theta* is given in Figure 3.8. The comparison is made in only two dimensions here for the purpose of clarity.

The path obtained by the A* algorithm is not the shortest path, whereas the Lazy Theta* solution is. The vertices used for the shortest path solution (marked by the "+"s in Figure 3.8) can be used as the waypoints from which obstacle avoidance trajectories can be created. The full description of the Lazy Theta* algorithm is given in Nash *et al.* (2010).

The line-of-sight checks for the PLIR application are performed using an algorithm from Ericson (2005), Chapter 5: "Basic Primitive Tests". The algorithm checks for an intersection between a line segment and an AABB. If no intersection occurs, the two points that describe the line segment have line-of-sight of one another. This line-of-sight algorithm is also used in Section 3.2.2 to detect collisions between the robot and the obstacles.

The selection of waypoints is restricted to the subset of the robot's workspace in the quadrant along the z-axis between the negative x-axis and the positive y-axis as shown in Figure 3.9. This admissible region is described relative to the gripper's coordinate frame. The restriction is due to the physical configuration of the robot and the method used for finding a collision-free path. Since the path is first found for the gripper only without considering other robot parts, crossing from one side of the line to the other or attempting to pass over an obstacle brings the robot parts closer to the obstacle instead of moving away from it, and increases the likelihood of a collision.

Once the waypoints have been generated, a trajectory is created with the waypoints using an interpolating fifth order B-spline. The reason an interpolating spline is chosen instead of an approximating one is because the trajectory must not enter the obstacle boundaries at any point along its path. Since the waypoints exist on the obstacle boundaries themselves, approximation could result in a collision with the obstacle as discussed in Section 2.3.2. The gripper trajectory is

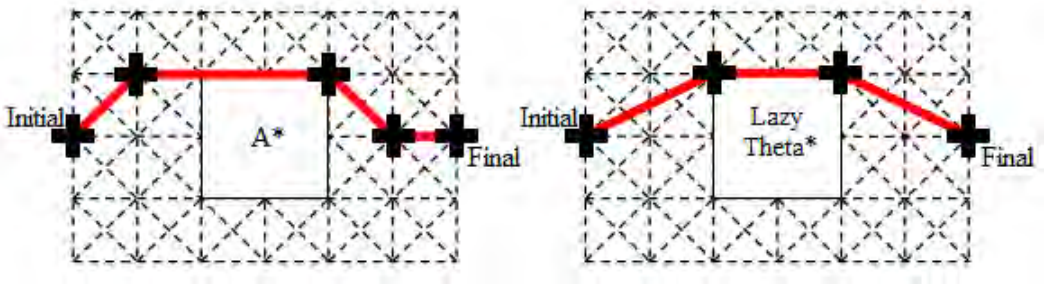

Figure 3.8: Figure showing the shortest path result of the A* algorithm (left) vs the Lazy Theta* algorithm (right).

checked for collisions between the robot and the obstacle as discussed in Section 3.2.2. If the front gripper trajectory is free from collisions, it is accepted and the entire trajectory creation procedure is carried out for the rear gripper.

3.2.2 Collision detection

Once obstacle avoidance trajectories have been generated, they are executed in a simulated environment to check for collisions between the obstacle and the robot body. Obstacle avoidance trajectories created for each of the grippers may be collision-free for the gripper itself, but not necessarily for the entire robot. Each joint actuator trajectory consists of a number of discrete gripper positions, and the robot configuration at each of these positions calculated at time steps of $T = 15\,\text{ms}$ as discussed in Section 2.4. It is necessary for the trajectory to be checked for collisions between the robot body and the obstacle at all of these configurations.

The collision checks are simple to perform due to the basic representation of the robot as mentioned at the beginning of the chapter. Since each of the robot parts are represented by a line segment, collisions can be investigated using the algorithm from Ericson (2005) that was used to perform line-of-sight checks. If any of the line segments making up the robot intersect an obstacle, a collision has occurred. If a collision is detected, the trajectory being tested is rejected. Depending on which part of the robot collides with the obstacle and where the collision occurs, the original AABB description must be enlarged to yield a new AABB description. With this, a new trajectory is formed that must then be checked against the original AABB description for collisions. If the new trajectory is collision free, it is used to command the robot. If the new trajectory is not viable, the robot will have to be assisted by a human operator at this point in the development stage, using a ground station computer (discussed in Chapter 5).

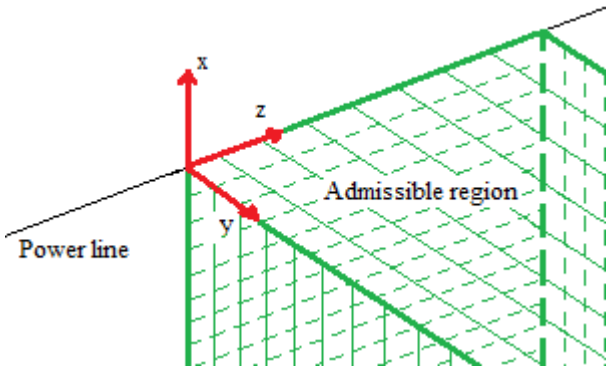

Figure 3.9: Figure showing the admissible waypoint insertion region in the robot's workspace.

Chapter 4

Trajectory Optimisation

Two approaches for temporal trajectory optimisation have been investigated, namely "time ratio optimisation" (TRO) and "interval time optimisation" (ITO). The TRO method which scales trajectory times is implemented using the inverse dynamics (ID) model only due to the inherent time scaling required when using the ID model to satisfy design constraints. Time scaling for the Simulink model is not applicable since the model inherently accounts for the robot's constraints. To compare the cost functions and optimisation viability of the ID and the Simulink models, the ITO method is implemented using both of the models.

The cost function in question for a fixed set of waypoints is $J(\bar{t}_{wp})$ as described in Section 2.6.2, (2.33). It is the cost associated with the end effector trajectory $\vec{s}(\bar{\theta}(t))$, where $\vec{s}(\bar{\theta}(t))$ incorporates each joint's B-spline trajectory. The trajectory and corresponding cost function for a joint are derived from a set of waypoints $\mathbf{s}_{wp}$ and respective unscaled time points $\bar{t}'_{wp}$. The reason $J(\bar{t}_{wp})$ is described here as a function of the interval times set in $\bar{t}_{wp}$ only is because the waypoint locations are fixed during optimisation. The optimisation is being performed on the temporal component of the trajectory $\bar{t}_{wp}$ alone. A vector $\bar{r}$ can be defined as the n-element vector containing the magnitudes of the interval times between each element in $\bar{t}_{wp}$ with entries

$$r_i = t_{wp_{i+1}} - t_{wp_i} \quad for\, i = 0, ..., n-1. \tag{4.1}$$

Based on the definition in (4.1), a cost function can be written as $J(\bar{r})$; a function of the interval times instead of the waypoint times.

4.1 Time ratio optimisation

The TRO investigation has been conducted with two different three-waypoint illustrative trajectories using the PLIR Trajectory Solver application discussed in Section 5.1. The cost function for each end effector trajectory is unique, so it is not feasible to investigate all possible trajectories.

The first sample trajectory $\vec{s}_1(\bar{\theta}(t))$ is generated using the following three waypoints:

- $\{\underbrace{(0,\ 0,\ 0.15)}_{\text{initial(x,}y\text{,}z\text{)}},\ \underbrace{(0,\ 0.1,\ 0.4)}_{\text{intermediate(x,y,z)}},\ \underbrace{(0,\ 0,\ 0.6)}_{\text{final(x,y,z)}}\}$

The second sample trajectory $\vec{s}_2(\bar{\theta}(t))$ is generated using the following three waypoints:

- $\{(0,\ 0,\ 0.15),\ (0,\ 0.05,\ 0.25),\ (0,\ 0,\ 0.6)\}$

Before a trajectory can be optimised with respect to time, it is first created with equal interval times as in Section 2.3.4. At this point, all trajectories satisfy the constraints on torque, speed and current and $\bar{t}_{wp}$ is such that $r_i = r_{i+1}$ for all i.

Trajectories generated with these equal interval times are not optimal, except in the possible case where equal interval times is indeed the optimal solution. The concept behind TRO is to find the optimal $\bar{r}'$ (unscaled interval times) for a trajectory of a particular set of waypoints so that once the unscaled knot vector $\bar{u}'$ has been scaled to satisfy the robot's constraints, the cost function $J(\bar{r})$ is at a minimum. Since t'_{wp_n} is always the same value, optimising the values of $\bar{r}'$ can be seen as optimising the ratios of the total trajectory time t_{wp_n} among the waypoint intervals; hence the name "time ratio optimisation".

4.1.1 Cost function analysis

Cost function analysis is important. It gives insight into the relationship between the interval times in $\bar{r}'$ used to generate the trajectory and the charge required to perform the trajectory. Insight into this relationship allows for decisions to be made on appropriate ways for minimising the objective cost function. A trajectory with three waypoints has an interval time vector $\bar{r}'$ with $n = 2$ elements. In this case, the cost function for a fixed set of waypoints can be described using a 3D graph with r'_0 on the x-axis, r'_1 on the y-axis and the associated $J(\bar{r})$ on the z-axis. Due to the nature of the waypoint time vector in (2.15), if only the first $n-1$ entries in $\bar{r}'$ are explicitly known, the n^{th} entry can be recovered uniquely as

$$r'_{n-1} = \begin{cases} \kappa & if\ n = 1 \\ \kappa - \sum_{j=0}^{n-2} r'_j & if\ n > 1 \end{cases} \tag{4.2}$$

This means that a cost function with $n = 2$ (as is the case for $\vec{s}_1(\bar{\theta}(t))$ and $\vec{s}_2(\bar{\theta}(t))$) can be expressed simply using a 2D line graph with r'_0 on the x-axis and $J(\bar{r})$ on the y-axis, where the value of r'_1 will be $\kappa - r'_0$ where $\kappa = 0.1\,\text{s}$.

To visualise the behaviour of a cost function $J(\bar{r})$, the solutions to a function $\vec{s}(\bar{\theta}(t))$ and subsequent $J(\bar{r})$ are enumerated for each possible r'_0 with $0.015\,\text{s} \leq r'_0 \leq 0.085\,\text{s}$ at $0.001\,\text{s}$ increments and plotted on the aforementioned graph.

4.1.1.1 Cost function for trajectory 1

The shape of $\vec{s}_1(\bar{\theta}(t))$ is not relevant in this analysis; of interest is the shape of $J_1(\bar{r})$ to see the relationship between the unscaled interval times $\bar{r}'$ and the cost function $J_1(\bar{r})$. Figure 4.1 is the graph of the cost function $J_1(\bar{r})$. From Figure 4.1, it can be seen that there is only one global minimum for $J_1(\bar{r})$. By inspection of the figure alone, it can be seen that the optimal value for r'_0 is

0.065 s, giving a minimum charge of 2.08 C once scaling of $\bar{u}'$ has been done. At the lower extreme of r'_0 (Figure 4.1), the trajectory interval time allocated to the first part of the trajectory is only 15%. For this reason, the cost of the trajectory is over 10 C due to the robot having to complete the first part of the trajectory so quickly (high joint torques required). At the other extreme where r'_1 is only allocated 15% of the trajectory time, a similar hike in cost occurs. This shows the importance of apportioning trajectory time appropriately.

4.1.1.2 Cost function for trajectory 2

Figure 4.2 shows the graph of the second cost function $J_2(\bar{r})$ for trajectory $\vec{s}_2(\bar{\theta}(t))$. Like Figure 4.1, Figure 4.2 shows a trajectory cost function that has one minimum. Again, this global minimum at $r'_0 = 0.035$ s of 1.43 C can easily be found by manual inspection of the figure.

4.1.2 Implementation

Since it is difficult (if not impossible) to find an analytical relationship between $\bar{r}$ and the cost of a trajectory, the optimal $\bar{r}$ must be found using numerical methods. Enumerated cost function analyses of typical trajectories were conducted and the cost functions for the PLIR trajectories were found to be reasonably smooth. Importantly for practical optimisation, the minima are not surrounded by steep gradients, and so the output of $J(\bar{r})$ will not be overly sensitive to small changes in $\bar{r}$, allowing for numerical robustness in terms of finding a minimum solution; i.e. small errors in $\bar{r}'$ will lead to only small errors in $J(\bar{r})$. In cases like these where the cost functions are smooth, it is probable that the Nelder-Mead simplex optimisation method will find the optimal solutions (Nelder and Mead, 1965).

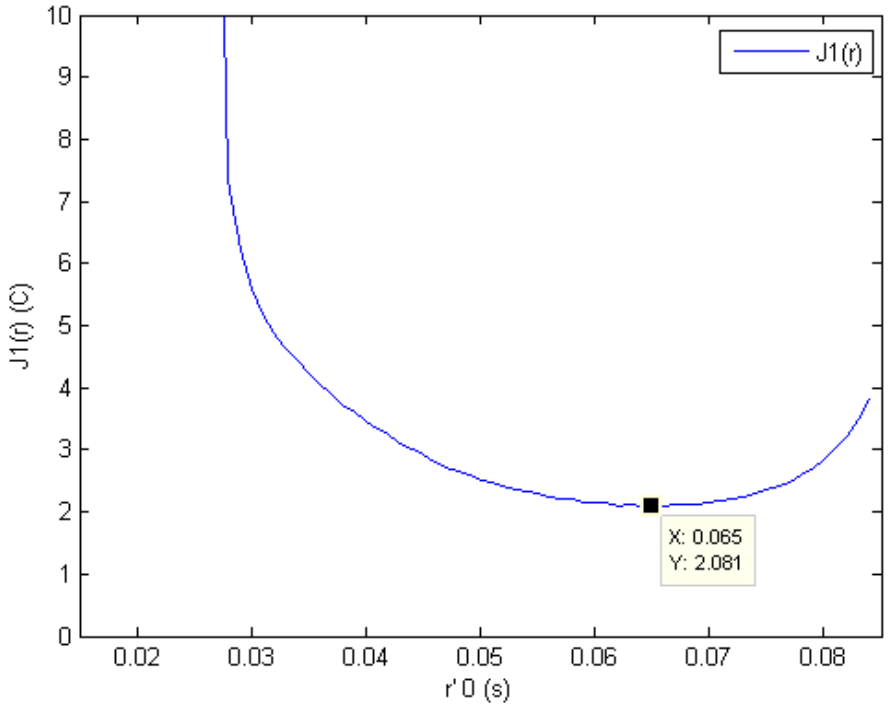

Figure 4.1: Line plot of $J_1(\bar{r})$ for $\vec{s}_1(\bar{\theta}(t))$ with $0.015\,\text{s} \leq r'_0 \leq 0.085\,\text{s}$ for TRO example 1.

The Nelder-Mead optimisation method is relatively simple and an explanation can be found in Barton and Ivey (1996). The optimiser evaluates the function $\vec{s}(\bar{\theta}(t))$ created using $\bar{r}'$ and then finds the associated cost $J(\bar{r})$ once $\bar{r}'$ has been scaled. Once the cost is calculated and returned to the simplex optimiser, the elements of $\bar{r}'$ are adjusted according to the simplex algorithm and the process is repeated until the cost $J(\bar{r})$ reaches a minimum (local or global). The flowchart for this optimisation process is given in Figure 4.3. The Nelder-Mead simplex optimisation is executed within the PLIR Trajectory Solver application using the Simplex Class of the DotNumerics V1.1 library for C# (De Santiago-Castillo, 2009); the software implementation is discussed in Section 5.1.2.

4.1.3 Optimisation performance

The performance of the optimisation process can be analysed by comparing the optimiser's cost function results with the enumerated cost function solution analysis. To compare the two, the cost function evaluation for each step of the optimiser as well as the final step is plotted onto the same set of axes as the cost function analysis. This way, the behaviour of the optimiser can be monitored as it tries to locate the global minimum. Furthermore, because the minima are exposed by the analyses, the success of the optimiser can be verified for a particular example.

For the optimisation of each of the above functions, $\vec{s}_1(\bar{\theta}(t))$ and $\vec{s}_2(\bar{\theta}(t))$, the simplex optimiser uses only one optimisation variable: r'_0. The other interval time variable is found by $r'_1 = 0.1 - r'_0$. The simplex optimiser is set up as follows:

- Optimisation variable: r'_0

  Lower limit = 0.015 s Initial guess = 0.001 s
  Upper limit = 0.085 s Initial step = 0.07 s

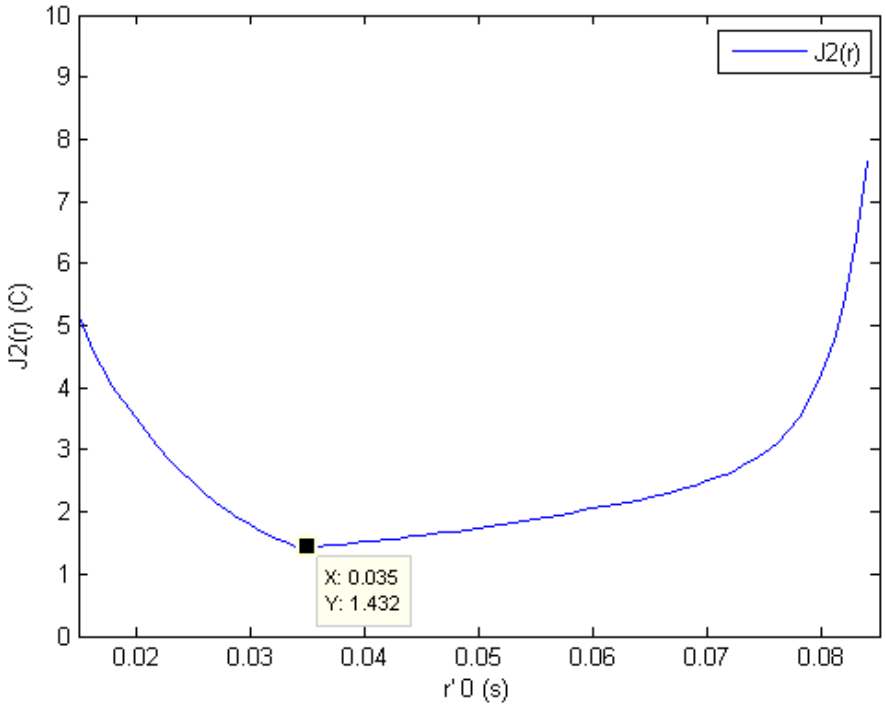

Figure 4.2: Line plot of $J_2(\bar{r})$ for $\vec{s}_2(\bar{\theta}(t))$ with $0.015 \leq r'_0 \leq 0.085$ for TRO example 2.

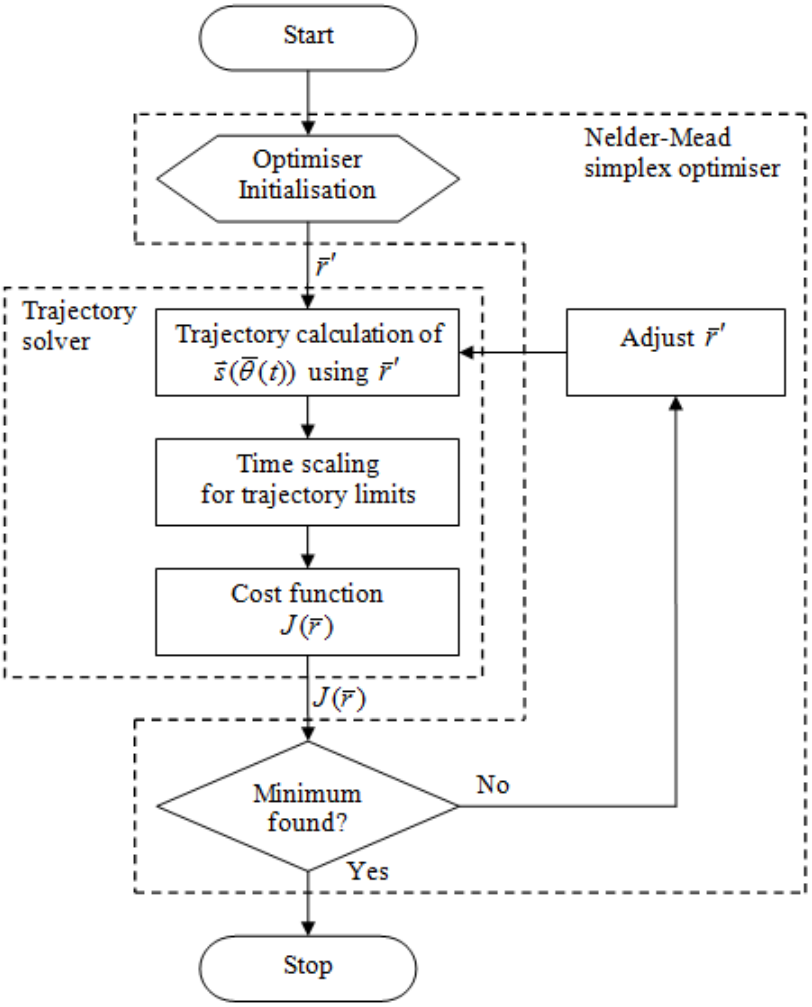

Figure 4.3: Flowchart of the TRO method.

The lower and upper limits are the constraints placed on the domain of values of r'_0. The initial guess is the value of r'_0 that is used at the beginning of the optimisation to calculate the first trajectory and cost. The initial step is the value added to the starting point of r'_0 to calculate the second trajectory and cost, and is set so that the simplex initially covers the entire domain of r'_0. The reason is to maximise the search region making it more likely that the global minimum will be found. However, there is no guarantee that this initial condition is going to result in the optimiser finding the global minimum for all trajectories. From here on, the Nelder-Mead optimiser proceeds to find a minimum in the cost function.

4.1.3.1 Trajectory 1 optimisation

The TRO result for $\vec{s}_1(\bar{\theta}(t))$ along with the analysis of $J_1(\bar{r})$ is given in Figure 4.4. Each evaluation of the cost function performed by the optimiser is marked by a "+", and some of the optimiser evaluations have been labelled to show the optimiser behaviour. The non-optimised cost for $\vec{s}_1(\bar{\theta}(t))$ with $r'_0 = 0.05\,\text{s}$ is 2.52 C with a trajectory time of 1.98 s once scaling has been applied.

The optimiser result is highlighted by a "$\circ$", and clearly coincides with the global minimum at $r'_0 = 0.065\,\text{s}$. In this case the optimiser succeeds after 17 iterations. The specifications of the machine used to perform these optimisations is given in Section 5.4.1. An optimised cost of 2.08 C with a trajectory time of 1.85 s gives a 17% reduction in charge over the non-optimised case.

4.1.3.2 Trajectory 2 optimisation

The TRO result for $\vec{s}_2(\bar{\theta}(t))$ together with the analysis of $J_2(\bar{r})$ is given in Figure 4.5. Again, some of the optimiser evaluations have been labelled. The non-optimised cost for $\vec{s}_2(\bar{\theta}(t))$ with $r'_0 = 0.05\,\text{s}$ is

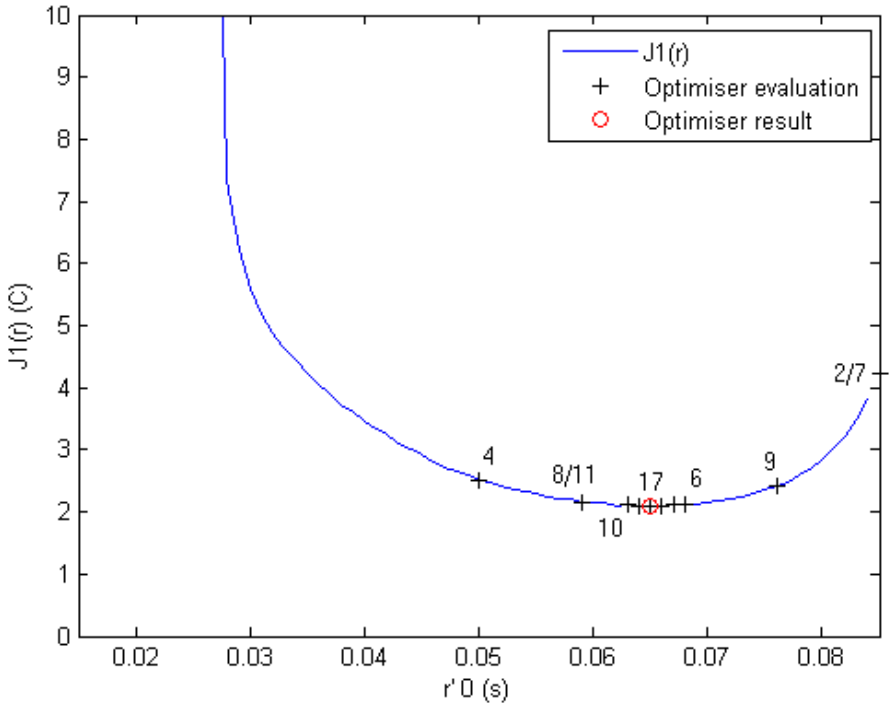

Figure 4.4: Line plot of $J_1(\bar{r})$ for $\vec{s}_1(\bar{\theta}(t))$ with the optimiser outputs for TRO example 1.

1.74 C with a trajectory time of 1.42 s. The optimiser was successful in finding the global minimum with 20 iterations giving an optimised cost of 1.43 C at $r'_0 = 0.035$ s with a trajectory time of 1.07 s. This is a charge reduction of 18% over the non-optimised cost.

4.2 Interval time optimisation

This approach does not involve the optimisation of trajectories by optimising initial time ratios and then scaling them to meet limit requirements. Instead, the optimised variables are the interval times in $\bar{r}$ themselves with each r_i defined by (4.1) at the beginning of the chapter. Interval times are generated by the optimiser directly and are not scaled before cost is calculated. In other words, $\bar{r}$ is not the result of scaling $\bar{r}'$ according to limits imposed on the trajectories in terms of torque, speed and current. These hard limits are not seen as impassable boundaries, but rather as expensive boundaries to violate. Violating a limit incurs a penalty that is added to the value of $J(\bar{r})$.

The ITO approach has been investigated using the ID and Simulink models so that the optimisation performance of both the models could be compared. The waypoints used to generate the trajectories are the same as those for the first TRO trajectory.

The sample trajectory $\vec{s}_1(\bar{\theta}(t))$ is generated using the following three waypoints (Trajectory 1 from Section 4.1):

- $\{\underbrace{(0,\ 0,\ 0.15)}_{\text{initial(x,}y,z\text{)}},\ \underbrace{(0,\ 0.1,\ 0.4)}_{\text{intermediate(x,y,z)}},\ \underbrace{(0,\ 0,\ 0.6)}_{\text{final(x,y,z)}}\}$

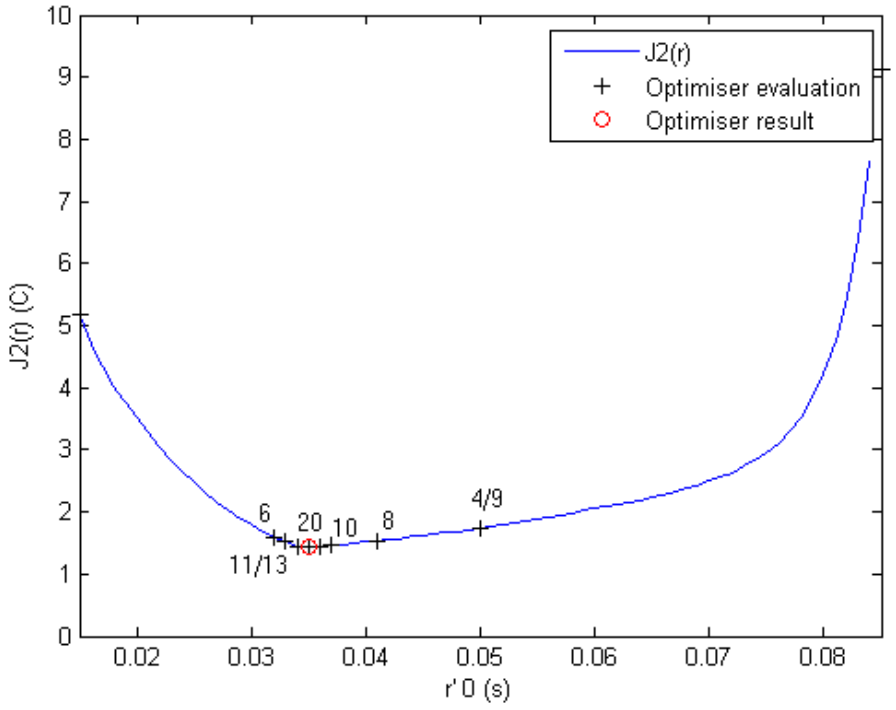

Figure 4.5: Line plot of $J_2(\bar{r})$ for $\vec{s}_2(\bar{\theta}(t))$ with the optimiser outputs for TRO example 2.

4.2.1 Cost function analysis

Just as in Section 4.1.1, cost functions are inspected in order to see what the relationship is between $\bar{r}$ and $J(\bar{r})$. However, because ITO does not perform scaling in order to satisfy trajectory constraints, the optimiser has to minimise a cost function that includes limit violation penalties as mentioned above. This is so that limits are obeyed when minimising the cost function. The violation penalties can be expressed as the scaling factors described in Section 2.3.4 multiplied by some constant. If a limit is violated, the equations in Section 2.3.4 are used to find the scaling factor, $\frac{1}{\lambda}$, for the corresponding violation of either torque, speed or current.

With ITO, each of the scaling factors are distinguished from one another and are represented as $\frac{1}{\lambda_\tau}$, $\frac{1}{\lambda_\omega}$ and $\frac{1}{\lambda_I}$ which are for the torque, speed and current limits respectively. If a violation does not occur for a certain parameter, the scaling factor is set to 1. Otherwise, it will be calculated as a number greater than 1 due to a violation of a torque/speed/current limit based on (2.19), (2.30) and (2.31). So, instead of inspecting the cost function $J(\bar{r})$, a new cost function is defined as

$$J^P(\bar{r}) = J(\bar{r}) + \left(\left(\frac{1}{\lambda_\tau} - 1\right) k_\tau + \left(\frac{1}{\lambda_\omega} - 1\right) k_\omega + \left(\frac{1}{\lambda_I} - 1\right) k_I\right) \tag{4.3}$$

where $J^P(\bar{r})$ is the cost function taking into account the violation penalties. The multipliers, k_τ, k_ω and k_I are used to make the time scaling factors a "cost" in Coulombs to be added to $J(\bar{r})$. The value of k must be designed so that the cost function space is suitable for the simplex optimisation algorithm. Experimentally, values of $k_\tau = k_\omega = 160\,\mathrm{C}$ and $k_I = 10\,\mathrm{C}$ were seen to provide suitably scaled "charge penalties" for typical trajectories when limits were violated. Clearly, if there are no limit violations the cost function will be

$$J^P(\bar{r}) = J(\bar{r}). \tag{4.4}$$

Since ITO optimises the values of $\bar{r}$ directly without the scaling of an initial time knot vector $\bar{u}'$, the cost function $J^P(\bar{r})$ needs to be evaluated over a domain of $\bar{r}$ that is not bounded between $0.015 < r_i < 0.085$ as in Section 4.1. Hence, the values of all elements in $\bar{r}$ need to be explicitly known, and $J^P(\bar{r})$ requires an $(n+1)$-dimensional representation.

For trajectories $\vec{s}_1(\bar{\theta}(t))$ and $\vec{s}_2(\bar{\theta}(t))$, $n = 2$, and so three dimensions are used to describe the cost functions and their relationships to $\bar{r}$. In this section, contour plots will be used to simplify the representation of the three dimensions (r_0, r_1 and $J^P(\bar{r})$).

The solutions to $J^P(\bar{r})$ within the domain limits are enumerated and represented on a contour plot, with the global minimum being found in the process. $J^P(\bar{r})$ is evaluated over a $4\,\mathrm{s} \times 4\,\mathrm{s}$ domain in a grid formation at 0.1 s intervals, with r_0 on the x-axis and r_1 on the y-axis. The values of $J^P(\bar{r})$ are represented by colour-mapped contours.

4.2.1.1 Cost function: ID model

The contour representation of the ID model cost function $J_1^P(\bar{r})$ for trajectory $\vec{s}_1(\bar{\theta}(t))$ is given in Figure 4.6. It shows the global minimum of the function $J_1^P(\bar{r})$ as well as the torque/speed/current limit boundary for trajectory $\vec{s}_1(\bar{\theta}(t))$. The global minimum is 2.10 C at $\bar{r} = [1.3\,\mathrm{s},\ 0.6\,\mathrm{s}]$ with a total trajectory time of 1.9 s. This is the same global minimum as in Section 4.1.1.1 as one would expect since the models used are the same.

The dash-dotted line in Figure 4.6 represents the torque/speed/current limit boundary. It can be seen that the plot of $J_1^P(\bar{r})$ is represented by (4.3) to the bottom left of the limit boundary, and by (4.4) to the upper right of the limit boundary. The global minimum is on the limit boundary itself; the significance of this will be discussed in Section 4.3.

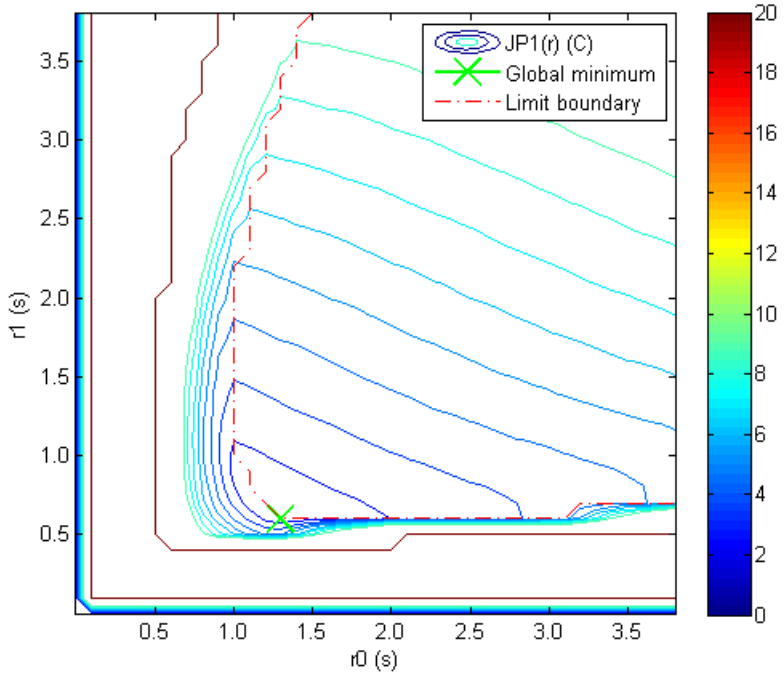

Figure 4.6: Contour plot of ID model cost $J_1^P(\bar{r})$ for $\vec{s}_1(\bar{\theta}(t))$ with $0.1\,\mathrm{s} \leq r_i \leq 4\,\mathrm{s}$, $for\, i = 0,\, 1$ using ITO.

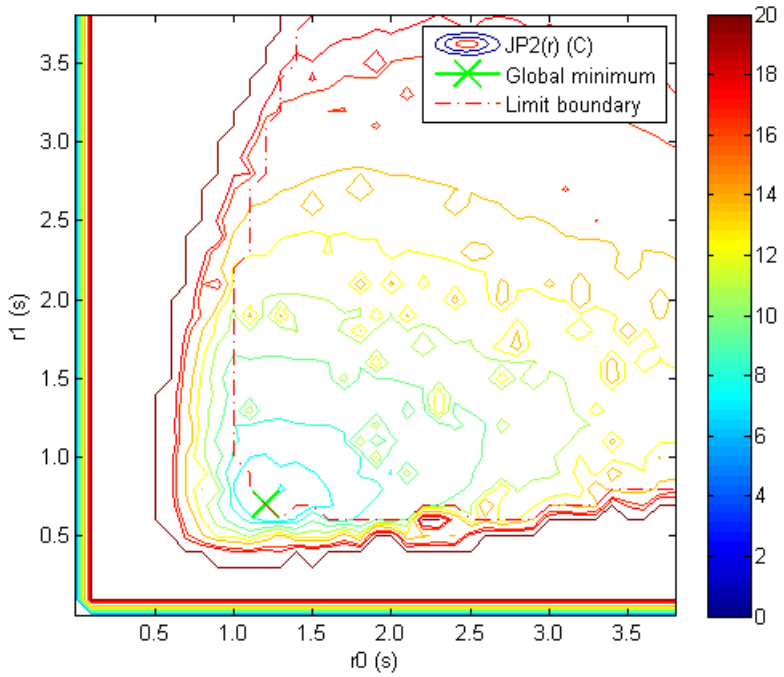

Figure 4.7: Contour plot of Simulink model cost $J_2^P(\bar{r})$ for $\vec{s}_1(\bar{\theta}(t))$ with $0.1\,\mathrm{s} \leq r_i \leq 4\,\mathrm{s}$, $for\, i = 0,\, 1$ using ITO.

4.2.1.2 Cost function: Simulink model

The contour representation of the Simulink model cost function $J_2^P(\bar{r})$ for trajectory $\vec{s}_1(\bar{\theta}(t))$ is given in Figure 4.7. Again, Figure 4.7 shows the global minimum of $J_2^P(\bar{r})$ as well as the torque/speed/current limit boundary for trajectory $\vec{s}_1(\bar{\theta}(t))$. The global minimum is 6.71 C at $\bar{r} = [1.2\,\text{s},\ 0.7\,\text{s}]$ with a trajectory time of 1.9 s (same as the ID model).

The Simulink model predicts a minimum cost that is approximately three times higher than what the ID model predicts. This is because of the cost of friction and other real-world effects that the Simulink model takes into account. The global minimum is on the limit boundary as with the ID model.

4.2.2 Implementation

Due to the smooth nature of the ID model cost function contours, it is clear that the Nelder-Mead simplex approach for optimisation will be well suited to ITO as it was for TRO. In the case of the Simulink model cost function, the irregular contours may cause the optimiser to get "stuck" in a local minimum.

The Nelder-Mead optimisation method is implemented in the following way. From a given starting point (initial $\bar{r}$), the trajectory $\vec{s}(\bar{\theta}(t))$ is found and the scaling factors of the limit violations (if any) are calculated. The cost function $J^P(\bar{r})$ is subsequently found using (4.3). This cost result is then returned to the simplex optimiser, and a new $\bar{r}$ is generated by the optimiser. The new trajectory $\vec{s}(\bar{\theta}(t))$ is then calculated and the process continues iteratively until the optimiser settles on a minimum value for $J^P(\bar{r})$. The flowchart for this optimisation process is similar to that in Figure 4.3, but using a different trajectory solver unit and adjustment block, shown in Figure 4.8. The Nelder-Mead simplex optimisation for this method is executed within the PLIR Trajectory Solver application using the Simplex Class of the DotNumerics V1.1 library for C# (De Santiago-Castillo, 2009).

4.2.3 Optimisation performance

The performance of ITO is observed using the same concepts of Section 4.1.3. The vector $\bar{r}$ for each optimiser cost function evaluation is plotted on the same set of axes as the cost function analysis contour plot. The final optimised $\bar{r}$ is also marked on the same set of axes and this can be compared to the global minimum as well as the trajectory torque/speed/current limit in order to judge the success of the optimisation.

The simplex optimiser is set up with the same logic as put forward in Section 4.1.3. The initial steps are set in an attempt to get maximum coverage of the domain of $\bar{r}$ by the optimiser. The optimiser has the following setup:

- Optimisation variables: r'_0, r'_1

  Lower limit = 0.3 s Initial guess = 0.4 s
  Upper limit = 2 s Initial step = 1.6 s

4.2.3.1 ID model optimisation

The ID model ITO of $\vec{s}_1(\bar{\theta}(t))$ together with the cost function analysis contour of $J_1^P(\bar{r})$ is given in Figure 4.9. The starting point of the optimisation can be seen at $\bar{r} = [0.4\,\text{s},\ 0.4\,\text{s}]$. The optimiser's

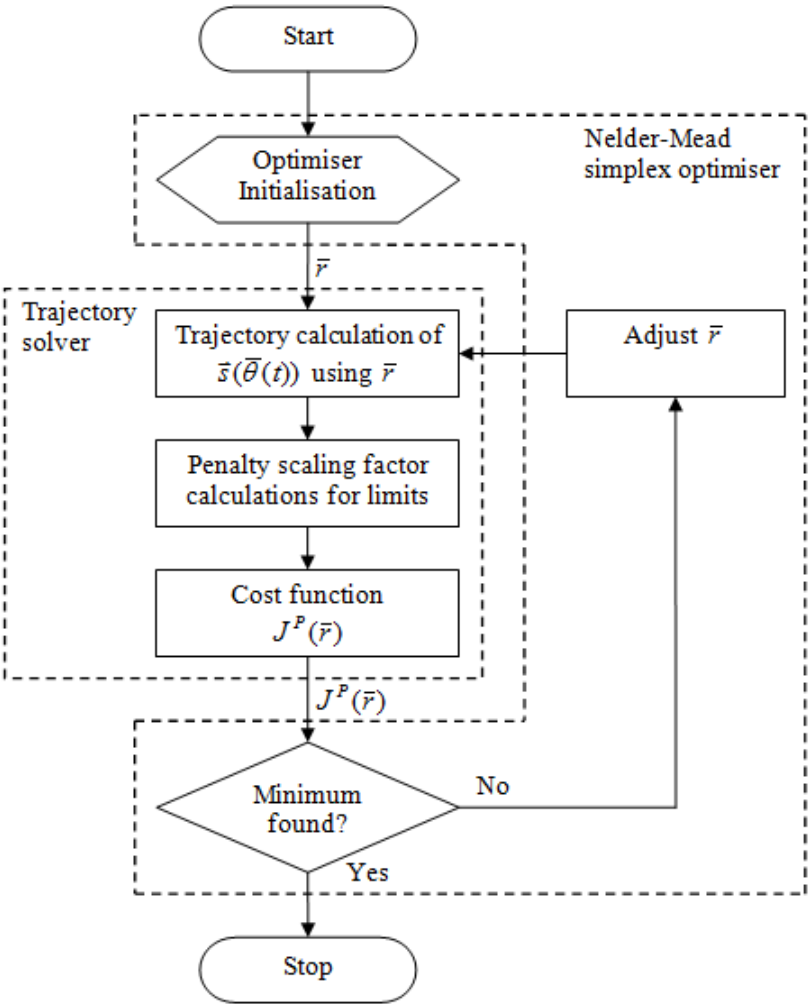

Figure 4.8: Flowchart of the ITO method.

final minimised value for $J_1^P(\bar{r})$ is at $\bar{r} = [1.2\,\text{s}, 0.7\,\text{s}]$ giving a total cost of 2.14 C and a trajectory time of 1.9 s, which is not the global minimum. The final output of the optimiser is on the limit boundary, which means that the trajectory limit of torque/speed/current is reached but not violated. This optimisation took a total of 41 iterations to complete.

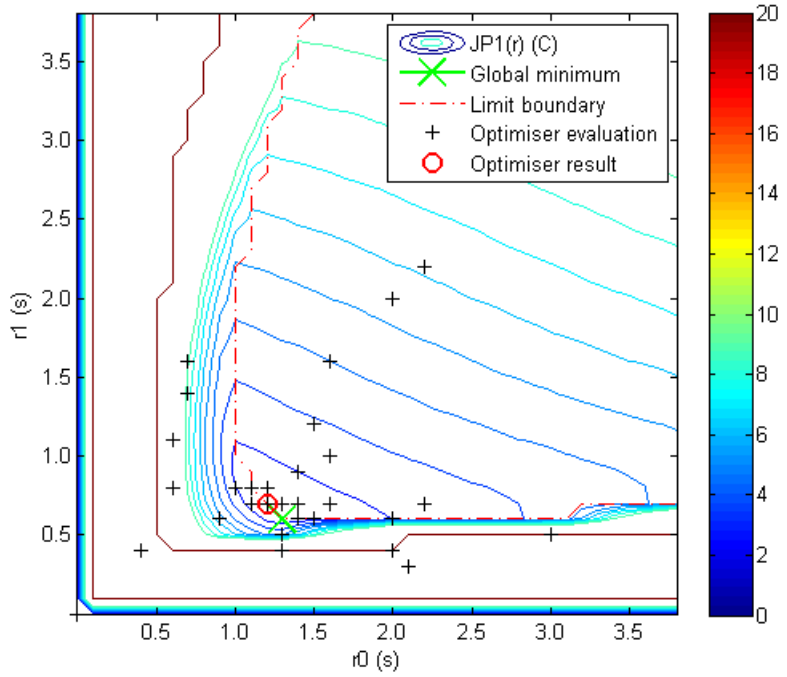

Figure 4.9: Contour plot of ID model cost $J_1^P(\bar{r})$ for $\vec{s}_1(\bar{\theta}(t))$ with the optimiser outputs using ITO.

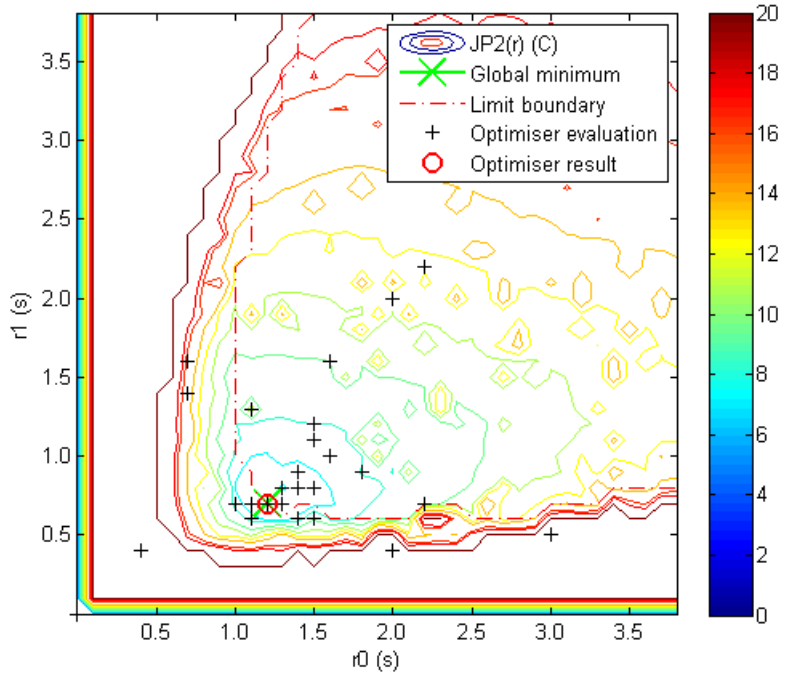

Figure 4.10: Contour plot of Simulink model cost $J_2^P(\bar{r})$ for $\vec{s}_1(\bar{\theta}(t))$ with the optimiser outputs using ITO.

4.2.3.2 Simulink model optimisation

The Simulink model ITO of $\vec{s}_1(\bar{\theta}(t))$ together with the cost function analysis contour of $J_2^P(\bar{r})$ is given in Figure 4.10. The optimiser's final minimised value for $J_2^P(r)$ after 32 iterations is at $\bar{r} = [1.2\,\text{s}, 0.7\,\text{s}]$, with a total cost of 6.71 C which is the global minimum. In this optimisation too, the limits of torque, speed or current were not exceeded.

4.3 Discussion and comparison

4.3.1 TRO vs ITO

The TRO method was successful in finding the global minimum in the case of sample trajectory $\vec{s}_1(\bar{\theta}(t))$ and sample trajectory $\vec{s}_2(\bar{\theta}(t))$. In the cost function inspections, the global minimum was clearly visible in both cases, making it easy to validate the time ratio optimiser's results. TRO places a restriction on the optimiser's result in that the limit of either torque, speed or current is always reached at some point along the trajectory due to the trajectory time scaling method. It does not consider solutions where the torques/speeds/currents are all below their respective limits when searching for minimum cost.

The ITO cost function analysis provided some insight into the locality of optimal values for $\bar{r}$ with respect to the trajectory torque/speed/current limits. Starting from the global minimum, if the trajectory is to be adjusted so that the torque/speed/current is not at the limit, one or more of the elements in $\bar{r}$ would have to be increased. This is clear from the work in Section 2.3.4; as the trajectory time is increased, the torques, velocities and currents decrease. If either r_0 or r_1 is increased in order to move away from the boundary, the contours clearly show that the cost for that particular trajectory will increase. This shows that for typical trajectories, the cost function minimum most likely exists on the limit boundary for some $\bar{r}$.

Comparing the performance of the two methods using the ID model, both of them successfully minimised the cost. However, the ITO method required 24 more iterations to complete the task (Table 4.1). This may be attributed to the fact that the TRO method was searching along the limit boundary only (where the optimal result is found), which drastically decreases the number of search points.

4.3.2 ID model optimisation vs Simulink model optimisation

The ID model has a smooth cost function contour when compared with the Simulink model's contour. This is because the ID model does not take non-linear friction and backlash into account. Although the two models are fundamentally different, the optimised interval times were the same (Table 4.2). For typical trajectories, this shows that the dominant factor influencing optimisation is the limit boundary (torque, velocity and current constraints) since the robot's constraints apply to both the ID model and the Simulink model.

Table 4.1: Table comparing the optimisation performance between TRO and ITO.

		TRO	ITO
Trajectory 1	Optimiser minimum	2.08 C	2.14 C
	Trajectory time	1.85 s	1.90 s
	No. of iterations	17	41

4.3.3 Online vs offline optimisation

An important consideration to be made about trajectory optimisation is whether or not it is feasible to implement online trajectory planning. In the case of known obstacles, predetermined optimised trajectories that have been calculated offline can be stored in the robot's on-board computer and recalled during online operation to get around the obstacle. In this case, no trajectory calculations will have to be performed online, which saves time (and ultimately charge).

In cases where the robot comes across an unknown obstacle, autonomous waypoint selection and trajectory calculations will be performed online. Such tasks require processing power and time to be carried out before the trajectory can be put into operation. There is no decision to be made about whether or not autonomous waypoint selection should be performed online or not, as this is a fundamental tool used to get around an unknown obstacle. The charge cost incurred whilst performing autonomous waypoint selection is unavoidable.

The remaining choice to be made is whether or not it is feasible to use an optimised trajectory over a non-optimised trajectory for an unknown obstacle. There is a possibility that the autonomous waypoint selection and optimisation of trajectories can be performed whilst undertaking other tasks such as rolling along the line and moving the robot closer to the obstacle to perform the manoeuvre. In this case, performing such calculations online will yield no time delays in operation and therefore negligible costs will be incurred by the robot components over and above what would already be expended. If these calculations cannot be performed while undertaking other tasks, extra charge costs will be incurred by the robot components while the optimised trajectory is being calculated.

As mentioned in Section 2.6.2, the computer and other electronic components on the robot consume charge at a rate of $I_C \approx 0.75\,\text{A}$ while the robot is on and the computer is not in sleep mode. The charge consumed by the robot components over time is calculated as

$$Q_C = I_C t_C \tag{4.5}$$

where t_C is the time over which the robot's components are powered. This can be seen as the cost of calculating the optimised trajectory if no other work can be performed.

Considering the trajectories that were optimised in this chapter, the largest charge saving achieved was that of trajectory 2. The trajectory cost was reduced from 1.74 C to 1.43 C, a reduction of 0.31 C. If the robot computer is of the same specification to the one used to perform the optimisation calculations in this chapter, the time taken to perform the optimisation for trajectory 2 is 2 s using the ITO method and ID model. If no other work is done during optimisation, this yields an optimisation calculation cost penalty of $0.75\,\text{A} \times 2\,\text{s} = 1.5\,\text{C}$ given by (4.5) before the trajectory can be carried out. Compared to the charge saving of 0.31 C, this would render the optimisation pointless, and indeed only increase the charge consumption over the non-optimised case rather than reduce it. The problem is exacerbated if more waypoints are to be optimised.

Table 4.2: Table comparing the optimisation performance between the ID model and the Simulink model.

		ID	Simulink
ITO	Global minimum	2.10 C	6.71 C
	Optimiser minimum	2.14 C	6.71 C
	Trajectory time	1.90 s	1.90 s
	No. of iterations	41	32

Ultimately, the feasibility of trajectory time optimisation is dependent on whether or not it can be performed while the robot is undertaking other tasks. If it can, this will mean that no extra time is wasted while the calculations are being performed, and online trajectory optimisation will be feasible. This will also benefit the PLIR in terms of the unavoidable autonomous waypoint selection calculations, ensuring that no extra charge cost is incurred for autonomous waypoint selection. If other tasks can not be done in parallel, trajectory optimisation will most likely cost more than it saves.

4.4 Conclusion

The TRO method searched for the minimum trajectory cost where it exists, i.e. along the limit boundary. This may make it more reliable in finding the global minimum due to a smaller search domain. Furthermore, it required the cost function to be minimised in $\mathbb{R}^{n-1}$, whereas the ITO method required optimisation in $\mathbb{R}^n$. This requirement for an extra dimension was the cause for the ITO method generally requiring more iterations to reach a solution compared with the TRO method. However, the ITO method was able to able to process iterations much quicker due to it not having to perform any time scaling of trajectories. So in reality, there is not much to separate the two methods.

There is one advantage, though, that the ITO method has over the TRO method; it is easier to upgrade to incorporate waypoint location optimisation parameters in future. Just as waypoint interval times were directly optimised by the ITO method using a simplex optimiser, so can the locations of the waypoints themselves be optimised directly if required. For this reason, ITO is the method of choice for this robot.

When comparing the ID and Simulink robot models in the Section 4.2.1, there was a stark difference between the two in terms of their cost function search spaces for the same set of waypoints. As mentioned before, the primary reason for this was due to the non-linear characteristics of the real robot that are incorporated only in the Simulink model.

Aside from this major difference, there was a clear congruity too. The location of the limit boundaries were almost identical which means that both models predicted almost the same limit violations for the matching interval times. Furthermore, this caused the optimised trajectories to have very similar solutions in terms of the waypoint interval times as seen in Section 4.3.2. This suggests that for the second robot prototype (which has superior components and much less significant non-linearities), the ID model will be appropriate for predicting the robot's behaviour when executing trajectories.

While both of these optimisation methods had success in finding optimal trajectory charge solutions in the temporal domain, they did have a clear weakness in that they did not solve the original optimisation problem for minimising global charge consumption. This would require optimisation both the temporal and spatial domains simultaneously. As discussed in Section 2.6.2, the inclusion of the spatial component makes global optimisation significantly more difficult to achieve, and is beyond the scope of this dissertation.

Regardless of whether or not the PLIR can carry out other tasks while performing trajectory optimisation calculations, it will be desirable to store optimised obstacle avoidance trajectories for known obstacles that have been solved offline on the PLIR computer's hard drive. These trajectories can be recalled and executed when a known obstacle is classified by the vision system. It is expected

that most obstacles encountered will be in a class of known obstacles (spacers, dampers, dead ends, splices, suspension clamps). This approach will reduce the robot's computational overhead, allowing more resources to be available for performing other tasks. In the case of unknown obstacles, it was found that calculating optimised trajectories online is only feasible if this does not cause a significant delay in the robot's inspection agenda. If it does cause a significant delay, it is beneficial to use non-optimised trajectories instead.

Chapter 5

Implementation

5.1 PLIR Trajectory Solver

The work covered in chapters 2 to 4 has been implemented in the form of a complete C# application: PLIR Trajectory Solver. It is a simulator for the PLIR that allows one to analyse the robot's behaviour within its workspace under various circumstances. This software is used on the ground station PC at this point in the development of the PLIR, and parts of it will be adapted to be implemented on the robot's on-board PC in the future as autonomy is improved. The development of this software using Microsoft Visual Studio 2010 Express edition and MathWorks MATLAB 2010b accounts for a large portion of engineering work done throughout this master's degree. This chapter reveals the scope of the software and hardware developed to support the research.

With this application, a user may simulate the robot's behaviour by adding waypoints, creating trajectories, performing obstacle avoidance simulations as well as trajectory optimisations. After a trajectory has been created, the entire set of trajectory data is available for analysis; from joint movements to charge consumption characteristics, right through to visualising the simulated trajectory in the virtual workspace. The software incorporates modules for:

- Inverse kinematics
- B-spline trajectory representation
- Robot modelling
- Obstacle avoidance
- Trajectory optimisation
- A simulated robot environment
- Communication protocols for trajectory and video information exchange between the ground station and the PLIR computer

All of these components are available through one application that is easy to understand and use. This section discusses the main features of each module and explains how each module is linked together. All of these modules work together to provide a rich simulated output for PLIR trajectory analysis. The code for this entire software application is available on the CD in Appendix B.1.

5.1.1 Graphical user interface

The graphical user interface (GUI) of the PLIR Trajectory Solver application allows access to all of the modules available in the application. It is intuitive and easy to navigate, and provides the user with a means to create and manipulate different PLIR trajectory scenarios. This includes everything from basic straight line manoeuvres, to autonomously generating obstacle avoidance manoeuvres with one click. The GUI allows the user to visualise how the robot will behave in reality by performing various simulations with the application. These simulations can then be compared to measured data from the PLIR (discussed in Section 5.2.1) that can be uploaded by clicking the "Load Trajectory" button. The simulation also provides the user with important information, such as:

- Trajectory execution time
- Robot joint angle profiles
- Joint velocity profiles
- Joint torque profiles
- Charge consumption characteristics

This information allows the user to gain insight in terms of operating the robot in the real world; whether the torque experienced by each joint should be reduced, or whether it is possible to avoid a certain type of obstacle on a power line segment. The GUI is shown in Figure 5.1. The following sections explain each of the GUI groups.

5.1.1.1 "End Effector Waypoints" group

This section of the GUI allows the user to select which gripper (front or rear) a trajectory must be created for. From here, the user can set a desired waypoint target position by using either the scroll bars or entering text directly into the "Target" text boxes for each of the Cartesian coordinates. Once the desired waypoint has been entered, clicking "Add Waypoint" places the waypoint into the waypoint list. The user may add as many waypoints as required, and can choose the order in which the waypoints enter the list by clicking on the list entry position. The waypoint will then enter the list at the desired position, and move the rest of the waypoints down the list. This can be seen in Figure 5.2. A waypoint can also be removed from the list by selecting it and the-n clicking "Delete Waypoint". The waypoint coordinates are measured in metres, and can be set at a resolution of 1 cm.

When trajectories are calculated, the global coordinate reference point for the simulated environment is the rear gripper at (0, 0, 0) (x, y, z). When the trajectory being created is for the front gripper, waypoint positions and obstacle descriptions are described using the global coordinate frame. When a rear gripper trajectory is being created, the coordinate reference based at the front gripper position as (0, 0, 0) (x, y, z). The two coordinate frames are shown in Figure 5.3. The coordinate frame used for rear trajectories is the global frame rotated 180° about the x-axis, with the origin at the front gripper position. The reason for using a different frame is so that the algorithms that have been implemented to solve trajectories do not need to be adjusted for different reference points. Due to the new reference, generating a rear trajectory is the same as generating a front trajectory with regard to the core functionality of the application.

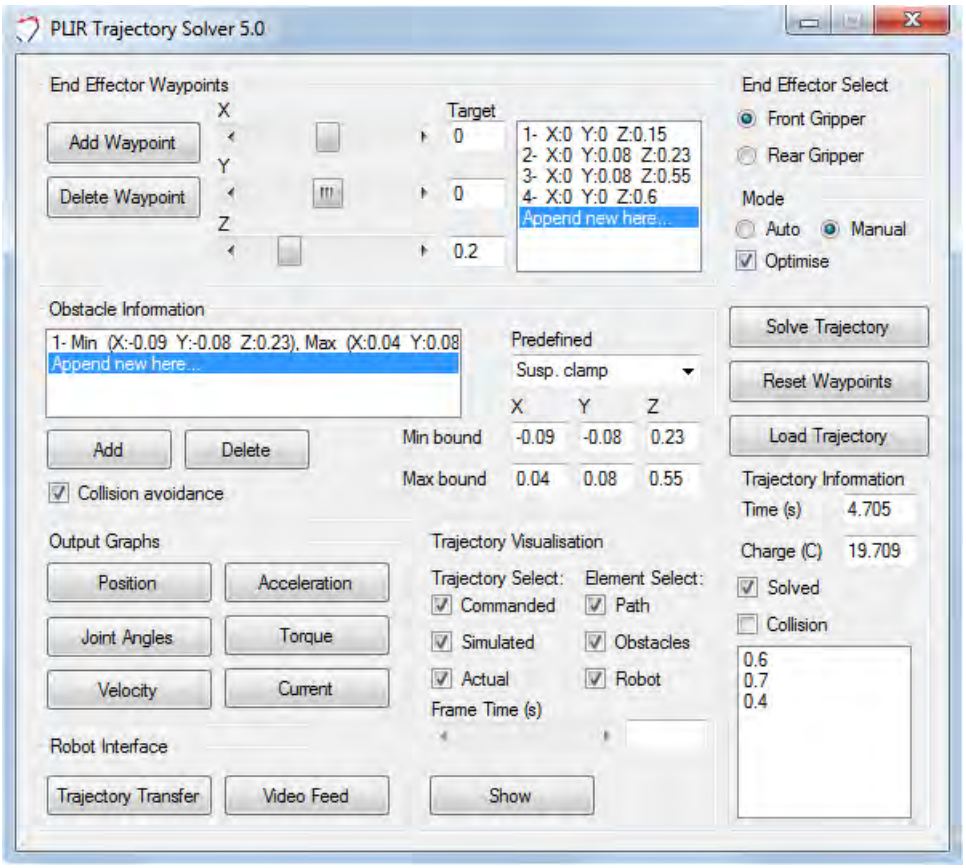

Figure 5.1: Graphical user interface of the PLIR Trajectory Solver application.

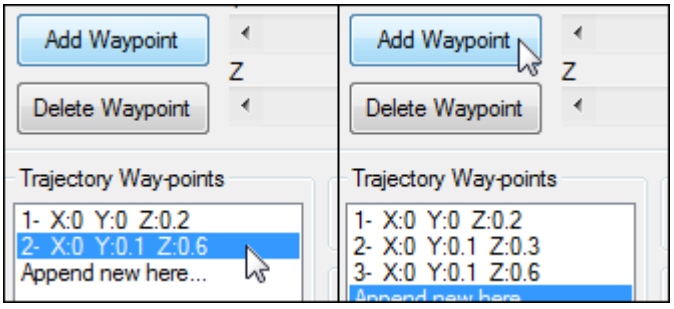

Figure 5.2: Figure showing a waypoint being inserted at a specific location in the list.

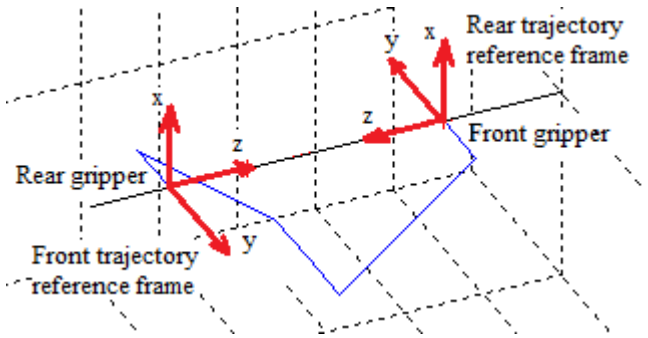

Figure 5.3: Figure showing the front trajectory coordinate reference frame (global frame) and rear trajectory coordinate reference frame.

5.1.1.2 "Mode" group

The "Mode" group allows the choice between manual and automatic trajectory creation. This is related to the obstacle avoidance aspect of the PLIR Trajectory Solver application. In manual mode, the user is able to add waypoints as required and to generate trajectories. The obstacle avoidance capabilities can still be activated in manual mode, however the user must first set the initial and final waypoint before the obstacle avoidance algorithm can be run. In automatic mode, the user does not enter any waypoint information. So long as obstacle information is specified, the application will generate entire obstacle avoidance trajectories for both the front and rear grippers with a single click, and then display the results. Depending on the mode, the button to the right is either labelled "Avoid Obstacle" for automatic, and "Solve Trajectory" for manual. The "Optimise" checkbox is used to select whether or not the manoeuvre should be optimised with respect to the temporal trajectory components.

5.1.1.3 "Obstacle Information" group

The "Obstacle Information" group is where the obstacle space is described. It allows the user to insert AABB elements into the workspace that represent a real-world obstacle. This information is used by the application to generate trajectories that will avoid the obstacle. In terms of autonomous operation of the robot on the power line, this part of the application is expected to receive obstacle information from an image processing system.

To enter an AABB description, the user enters the coordinates of the minimum and maximum boundaries (discussed in Section 3.1) into the "Min bound" and "Max bound" text boxes, and then clicks "Add". As with the waypoint list, the user can add as many AABB entries as they like, at any position in the list, and can delete any AABB from the list.

The "Predefined" drop-down box has some AABB obstacle descriptions that have been coded into the software. These have been used to test the obstacle avoidance capabilities of the software. Selecting a predefined obstacle populates the AABB list with the required boxes to describe the obstacle. The drop-down box includes the vibration damper and a worn vibration damper descriptions as used in Section 3.1.

The "Collision avoidance" check-box is used to enable/disable the collision avoidance functionality of the PLIR Trajectory Solver application. In automatic mode, obstacle avoidance is enabled regardless of the initial state of the check-box. If obstacle avoidance is enabled in manual mode, the initial and final waypoints will be set to the first and last in the waypoint list, and any necessary intermediate waypoints will be created by the obstacle avoidance system.

5.1.1.4 "Trajectory Information" group

Once a trajectory has been calculated, this group shows information of the trajectory solution. The time (in seconds) the trajectory manoeuvre is estimated to take, as well as an estimate on how much charge (in Coulombs) is required to perform the manoeuvre are shown in the two text boxes. The "Solved" and "Collision" check-boxes are read-only for the user. They indicate whether or not a feasible trajectory solution was found. When the "Solved" box is ticked, a feasible trajectory was found. If the "Collision" box is ticked, a collision has occurred in the simulation between the robot and the obstacle at some point along the generated trajectory. These two boxes are mutually exclusive.

The empty box at the bottom of the group is a list box that the application uses to write any extra information to the GUI, such as operational warnings. An example of such a warning would be if the user attempted to create a trajectory with fewer than two waypoints.

5.1.1.5 "Output Graphs" and "Trajectory Visualisation" groups

The "Output Graphs" group makes available all of the trajectory information in graphical form. The "Trajectory Select" checkboxes in the "Trajectory Visualisation" group are used to control which data are displayed in the output graphs. The user can select which of the "Commanded", Simulated" or "Actual" data they want to view, and then click on a button such as "Velocity". The "Commanded" (intended) trajectory refers to the generated B-Spline and ID model trajectory, "Simulated" (predicted) refers to the output of the Simulink model, and "Actual" refers to measured data from the robot during a manoeuvre.

When the user clicks a graph button, a window of graphs is displayed representing the trajectory data for each joint during a manoeuvre. For example, the velocity graph window is shown in Figure 5.4 showing the commanded data only. Similarly, graphs for acceleration, torque, current, joint angles and end effector position can be viewed.

The "Trajectory Visualisation" group also allows the user to view a virtual representation of the robot's manoeuvre. The simulated robot environment shows the power line segment, the obstacle AABBs, the robot and the end effector path together with waypoints. The three check-boxes in the "Element select" section allow the user to customise which aspects of the simulation are visible. The simulation of the trajectory is animated when the window is first opened by clicking the "Show" button, and the user can manually search through the trajectory animation frame by frame, using the scrollbar above the button. The manoeuvre can be observed from any position in 3D space. An example command trajectory in the simulated environment is shown in Figure 5.5.

5.1.1.6 "Robot Interface" group

The "Trajectory Transfer" button is used to transfer a generated trajectory to the PLIR for execution. This is covered in more detail in Section 5.1.2.9. The "Video Feed" button opens the

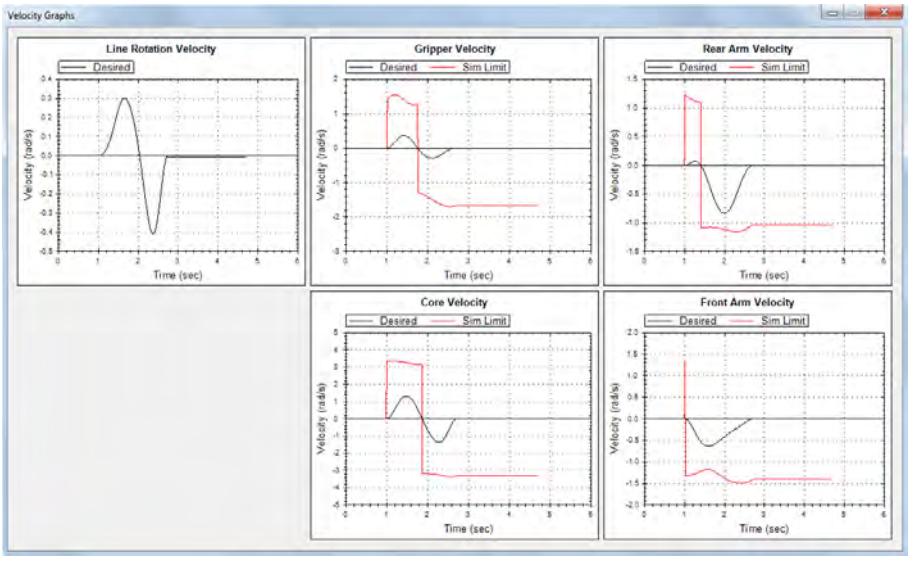

Figure 5.4: Figure showing velocity graph window.

PLIR camera window that shows the video feed from the PLIR. The video feed window is shown in Figure 5.6. The user starts the video feed by clicking “Start” (as long as the PLIR video feed is running) and can capture and save any video frame by clicking “Capture”. The window also provides information on the frame rate and the lag time of the video feed.

5.1.2 Code modules

This section describes the functionality of each of the major code modules in the PLIR Trajectory Solver application. Some of the code modules have been implemented as separate C# class objects. This allows some of the major code modules to be distinctly packaged (separate from one another) for ease of software design.

5.1.2.1 Inverse kinematics

The inverse kinematics solution for the PLIR is implemented in a separate `IK` class, developed based on the work of Buss (2004). When the application requires the joint angle solutions for a given end effector position for creating trajectories, the damped least squares (DLS) function of the `IK` class

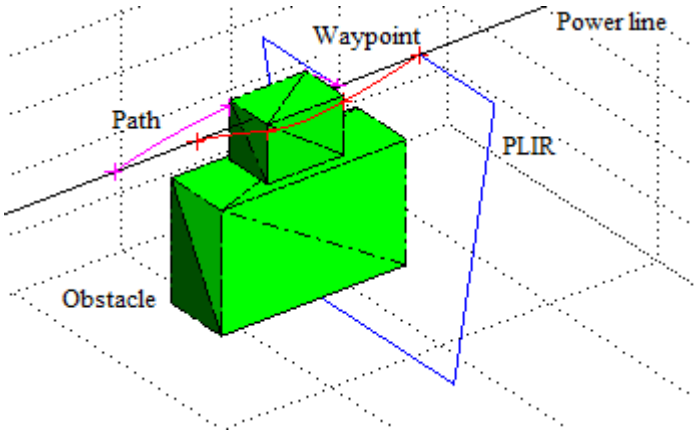

Figure 5.5: Figure showing simulated PLIR environment.

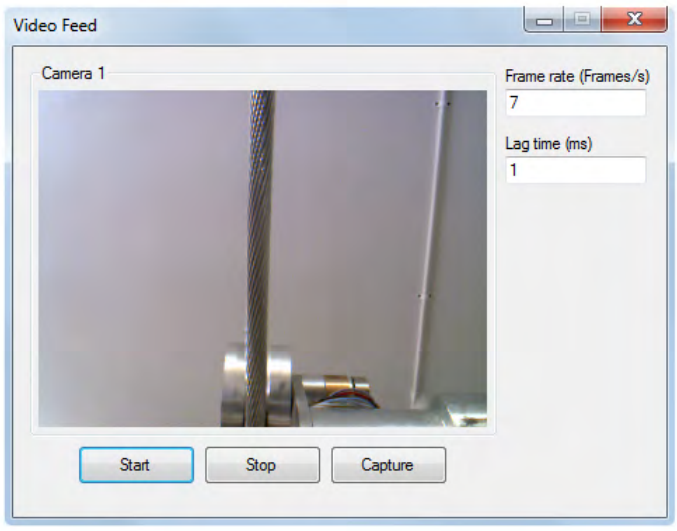

Figure 5.6: Figure showing the camera feed window.

is used (Appendix B, File: "IK.cs"). When the user adds a waypoint, this end effector position information is passed to a function called `GenAngle` which makes use of the `IK.DLS` function. The input to `GenAngle` is the end effector position in Cartesian coordinates, and the output is a set of joint angles used for a trajectory waypoint in the joint space.

The DLS function in `GenAngle` is called in the following way:

```
IK.DLS((double[])Target.Clone(), (double[])InitJointAngles.Clone(), DampingFactor,
MaxIterations, Tolerance);
```

The output of `IK.DLS` includes the coordinate transformation matrices, the rotation matrices, the robot part positions and the end effector position solution error. The DLS function takes as input the desired end effector position as `Target`, the starting joint angles for the DLS solution as `InitJointAngles`, the damping factor as `DampingFactor` = 0.1, the maximum number of DLS iterations as `MaxIterations` = 100, and the solution tolerance as `Tolerance` = 0.001. The damping factor of 0.1 has been set experimentally and was found to give the best solution speed/numerical stability ratio for finding an IK solution. The DLS solution tolerance allows the DLS iterations to terminate early if the solution error is less than 0.001 m, but with a limit of 100 iterations. These parameters were set experimentally and found to give solutions with zero error as long as the desired end effector position was within reach.

The `IK` class also contains a function that returns the robot part positions based on a given set of joint angles. This function is used in the `JointDynamics` function to retrieve part position information as well as the coordinate transform matrices in order to calculate the torques experienced by each of the robot joints at a given trajectory position. The input parameter is the set of robot joint angles as `JA`. The function is called as follows:

```
IK.ReturnPartPositions((double[])JA.Clone());
```

5.1.2.2 B-splines

The B-spline component has also been implemented in a separate class (Appendix B, file "BSpline.cs"). The class was developed based on the text by Biagiotti and Melchiorri (2008), from which fundamental code implementations have been taken. The `B-spline` class provides the application with a means of storing a B-spline trajectory, as well as all the functionality associated with B-spline trajectories such as creating them and obtaining trajectory position data and all relevant derivatives at specific time points along the trajectory. The main PLIR application uses an array of class instances to store B-spline trajectories for each of the robot's joints.

The function `CreateBSplines` in the PLIR application creates new instances of `B-spline` types, and stores them into a 4 element array. This is executed once the trajectory waypoints have been entered and the user clicks "Solve Trajectory" in the main application window. The code implementation is as follows:

```
BSplines[0] = new BSpline(WayPoints.Count, WayPointTime, WayPointAngleGripper, BSplineOrder,
Approximate);
BSplines[1] = new BSpline(WayPoints.Count, WayPointTime, WayPointAngleRearArm, BSplineOrder,
Approximate);
BSplines[2] = new BSpline(WayPoints.Count, WayPointTime, WayPointAngleCore, BSplineOrder,
Approximate);
BSplines[3] = new BSpline(WayPoints.Count, WayPointTime, WayPointAngleFrontArm, BSplineOrder,
Approximate);
```

The input for creating a new `B-spline` instance comprises of the number of trajectory waypoints as `WayPoints.Count`; the vector containing the waypoint times as `WayPointTime`; the set of waypoint angles for the desired trajectory as (for example) `WayPointAngleGripper`; the order of the desired B-splines as `BSplineOrder` and whether the B-spline solution must be found with waypoint interpolation or approximation as `ApproximateButton.Checked`. Note that all of the trajectories are formed with the same parameters except for the spatial waypoint information.

The trajectory information is retrieved from each `B-spline` instance using the built-in functions of the `B-spline` class. The application is able to access joint position, velocity, acceleration, jerk and further derivatives, as well as any temporal information needed about the trajectory. This information is used by the `JointDynamics` function so that, together with the IK data, the joint torques can be calculated. An example of retrieving joint position data is:

```
BSplineEval[0] = BSplines[0].EvaluateBSpline(TempTime);
```

This function has an input parameter for the time instant along the trajectory at which the data is required as `TempTime`, and returns the joint angle, velocity, acceleration and jerk for the attached gripper (element 0).

5.1.2.3 Inverse dynamics

Once the set of B-spline trajectories have been created by the `B-spline` class, the inverse dynamics of the robot must be solved for simulation purposes. The robot's inverse dynamics function is called `JointDynamics`. In this function, the joint trajectories are stepped through at 15 ms intervals; this is because the robot controller sampling time is 15 ms and the trajectory resolution must be at least the same to achieve sufficiently smooth control. All trajectory data is entered into rolling lists so that the data can be analysed at a later stage. The `JointDynamics` function incorporates the recursive Newton-Euler algorithm that calculates (and stores) the joint torques at each step of the trajectory. Based on the torque data, joint actuator current profiles are also generated and stored. The `JointDynamics` function does not require any input arguments and makes use of global application variables such as the B-spline trajectories. Data is appended to the lists as follows:

```
AngleGripperTemp.Add(list, JA[1]);
```

The time instant for the given trajectory position is given as `list` and the result to be stored is `JA[1]`. The variable type seen here as `AngleGripperTemp` is of type `RollingPointPairList` of the `ZedGraph` class that can be used to graph data in C# (Champion, 2005). This is done for each robot joint.

Once the inverse dynamics have been calculated and recorded along all of the trajectories, the `JointDynamics` function calls the `CheckLimits` function that checks the inverse dynamics of the simulated trajectories against the robot design limits. If the TRO optimisation method is used, `CheckLimits` makes necessary adjustments to the B-splines' temporal components to adhere to these limits. The `JointDynamics` function is called from a superior function named `SolveDynamics` that iteratively calls `JointDynamics` until the limits are obeyed. `SolveDynamics` is called once the B-splines have been created, which is done once the user clicks "Solve Trajectory".

5.1.2.4 Simulink model

Once the inverse dynamics of the robot has been solved using the `SolveDynamics` function, the Simulink model can be called to simulate the robot's behaviour. The Simulink model is incorporated into the PLIR Trajectory Solver application using a MATLAB COM object, which is discussed further in Section 5.1.3. The Simulink model can be run by calling the parent function `Simulate`, and the results are stored globally for use within the application.

5.1.2.5 Trajectory time scaling

Trajectory time scaling is required so that each of the calculated trajectories obeys the robot's mechanical design limitations as discussed in Section 2.3.4. The scaling is implemented across two functions. The first is responsible for calculating the limit violation margins and the time scaling factors required to bring the trajectories within the limits, and the second is responsible for making the necessary trajectory time adjustments based on all of the scaling factors calculated.

The first function, `CalcLimitViolations` is responsible for calculating the magnitude of velocity, torque and current limit violations if any, and solving for the required time scaling factor to avoid exceeding the limit(s). This function is called at the end of each `JointDynamics` step, and the calculated scaling factors are saved when they exceed the existing maximum. Once the entire trajectory has been checked and maximum scaling factors for each limit have been found, the `CheckLimits` function is called.

The `CheckLimits` function is called once the trajectories have been examined for limit violations and the maximum violations of each joint for each limit (torque/velocity/current) have been found. Once called, the `CheckLimits` function searches for the largest time scaling factor due to the largest limit violation by any joint for any limit. If the TRO optimisation method is used, this time scaling factor is applied to all of the joint trajectories. This is done by scaling the time knot vector for each trajectory using the `ScaleUknot` function of each `B-spline` instance. For example, the attached gripper trajectory time can be scaled by using the following function where the input parameter `MaxScalingFactor` represents the largest limit violation ratio:

```
BSplines[0].ScaleUknot(MaxScalingFactor);
```

However, if the ITO optimisation method is used, these scaling factors are used to calculate the violation penalties as described in Section 4.2.

5.1.2.6 Obstacle avoidance

Obstacle avoidance with the PLIR application can be implemented using one of two modes (automatic/manual) as discussed in Section 5.1.1. Both modes require the use of the `Obstacles` class for storage of AABB obstacle descriptions and the `ThetaStar` class that contains the algorithm for finding a shortest collision-free path between two points in a 3D graph.

A global variable of `Obstacles` class named `ObstacleBoxes` is used to store all obstacle data for the application. The class allows for obstacles to be added and removed, and for obstacle information to be retrieved once stored in `ObstacleBoxes`. The class also allows for AABB entries in `ObstacleBoxes` to be queried for certain information between themselves and other references in space. For example, the class contains a function that determines whether or not a query point in space lies in any of the AABBs, and if so, returns the identifier for the AABB.

Once an obstacle has been created by adding one or more AABBs, collision avoidance can be carried out. When the user clicks "Solve Trajectory" in manual mode with collision avoidance enabled, an instance of the `ThetaStar` class is created. This class allows the PLIR application to find a shortest path from the initial waypoint to the final waypoint around the obstacle. The class is initialised with a starting coordinate, an ending coordinate, and the obstacle information from `ObstacleBoxes`. From here, the `ThetaStar.Path` function of the `ThetaStar` class is called and will return the waypoints that make up the shortest distance path between the start and ending points. This is done using an implementation of the Lazy Theta* algorithm discussed in Section 3.2.1. The `ThetaStar` class was derived from the algorithm description found in Nash *et al.* (2010), except for the `LineOfSight` function for which the code was taken from Ericson (2005). The `LineOfSight` function is used by the Lazy Theta* algorithm to check for visibility between nodes in the 3D workspace graph.

Once the trajectory has been created with the new waypoints, a collision check is performed on the trajectory. The function that performs this check is named `CollisionCheck` and is called with no arguments since the trajectory and obstacle data are global. The entire end effector trajectory path is checked for collisions between all robot parts and the obstacle at each spatial point corresponding to a discrete trajectory time. This is achieved with the `ThetaStar.LineOfSight` function. The robot is described using a set of points connected by line segments, and if two neighbouring points do not have line-of-sight because of an obstacle's AABB at any point along the path, a collision has occurred. The `CollisionCheck` function returns a boolean result that is true if there is a collision, and false if there is not.

If a collision avoidance trajectory is being created for the rear gripper, the obstacle space must be described in the rear gripper's coordinate frame that is discussed in Section 5.1.1.1. The `Obstacles` class provides the option to set the rear gripper's coordinate frame origin (i.e. the front gripper position) and retrieve the obstacle space information relative to this coordinate frame so that a trajectory can be created and collisions can be detected as is done for the front trajectory.

When the user clicks "Avoid Obstacle" in the automatic mode, an entire obstacle avoidance solution is generated without user input. Firstly, the trajectory application is reset and all waypoints are cleared to allow for new trajectories to be created. Next, a preliminary end effector trajectory is created directly along the line from (0, 0, 0.15) to (0, 0, 0.87) (near the limit of the robot's reach), and collision checks are performed along the path. The end effector coordinate along the generated path where the robot configuration no longer results in a collision is used as a reference point for the final waypoint (as described in Section 3.2.1). Once the initial and final waypoints are set, the trajectory is generated as discussed for the manual mode, and the results are stored. For the rear gripper trajectory, the initial and final waypoints are set as in Section 3.2.1. The trajectory is then solved for the rear gripper and stored. As mentioned in Section 5.1.1.1, the trajectory is created using the rear gripper coordinate frame.

5.1.2.7 Trajectory optimisation

Trajectory optimisation is performed using the Nelder-Mead simplex optimisation method. The implementation of this simplex algorithm resides in the Simplex Class of the DotNumerics V1.1 library for C# (De Santiago-Castillo, 2009). The class provides a function for finding the minimum cost of a given function.

The function to be optimised is a slight variation of the `SolveTrajectory` function, named `SolveTrajectoryFunction`. The input argument for `SolveTrajectoryFunction` is the waypoint

interval time array that is given by the simplex optimiser at each iteration, and the function returns a trajectory cost to the optimiser. The cost at each iteration is calculated using the `CalculateCost` function. The main cost function implementation is as follows:

```
for (int ChargeCounter = 1; ChargeCounter < PosX.Count; ChargeCounter++)
{ TotalCharge += (CurrentTotal[ChargeCounter].Y + CurrentTotal[ChargeCounter - 1].Y); }
TotalCharge *= TimeStep / 2;
TotalCharge += AveComponentCurrent * MaxTime;
```

`PosX.Count` is the number of data points for the trajectory concerned. `CurrentTotal` is the total current required by all of the robot actuators at that particular point in the trajectory, `AveComponentCurrent` is the estimated average current required by all of the robot components excluding the actuators at any instant, and `MaxTime` is the total trajectory execution time. The `CalculateCost` function is called with a parameter that is used to select whether the cost should be based on the ID model, the Simulink model or the actual trajectory data measured from the PLIR after recorded manoeuvre data has been loaded.

`SolveTrajectoryFunctionSimplexOptimiser` is the function called to perform the entire optimisation process. This function is called once a trajectory has been generated if the "Optimise" check box is ticked. The function first sets up the optimisation parameters for each optimisation variable (waypoint interval time) as described in Section 4.1.3, and then starts the optimisation as follows:

```
simplex.ComputeMin(SolveTrajectoryFunction, OptVars, InitialStep);
```

The input parameters to this function are `SolveTrajectoryFunction` (the function to be minimised), the array of optimisation variables as `OptVars` and the initial step size for the solver as `InitialStep`. The choice of the initial step size is explained in Section 4.1.3. The function returns the optimal set of waypoint interval times or time ratios (optimisation method dependent) that minimise the cost function for the trajectory. The trajectory is finally solved using the optimal times and stored for analysis.

5.1.2.8 Trajectory simulation output

The trajectory simulation culminates in all of the trajectory data being displayed in a meaningful way so that it can be analysed. The trajectory simulation results can be viewed using the "Output Graphs", "Trajectory Visualisation" and "Trajectory Info" groups in the main application window. The "Trajectory Info" group has already been sufficiently explained.

The "Outputs Graphs" group provides the user with a means to visually analyse many trajectory aspects in dedicated C# windows such as that shown in Figure 5.4. As each graph window is opened, all of the necessary graph information is passed to the graph window as properties of that window. For example, the velocity graph window has properties to store each of the velocity profiles and the limit profiles. Below is an example of the attached gripper velocity data being passed to the velocity graph window as a property:

```
VelocityGraphs.VelocityGripperProperty = Velocity_2;
```

When the graph window loads, the graphs of the selected data (selected in the "Trajectory Select" section) are drawn onto their respective axes and all axes are then labelled. This is done for each of the following outputs:

- Position
- Joint Angles
- Velocity
- Acceleration
- Torque
- Current

The simulation of the workspace environment is done using MATLAB plotting functions that have been encapsulated in a .dll file and can be accessed like a regular C# class. The MATLAB interface is discussed in Section 5.1.3. When the user clicks "Show" in the "Trajectory Visualisation" group, the plotting functions in the `Plot` class enable the workspace environment to be visualised in 3D.

The first element to be plotted is the line segment, with the trajectory path added after that as follows:

```
Plot.Plot3(Z, Y, X, "red");
```

Both the line segment and trajectory path are plotted using the `Plot3` function. The input parameters are the three coordinate arrays as `Z`, `Y`, and `X`, and the desired colour of the trajectory path as `"red"`. The coordinate arrays hold the end effector positions at all points along the trajectory and are drawn onto the set of axes. For the rear trajectory, the x, y and z coordinate arrays have to be recalculated with respect to the rear gripper coordinate frame before they are drawn onto the axes so that they are correct relative to the global frame.

The obstacle is plotted onto the same set of axes using a computational geometry class that has also been derived from MATLAB functions. The AABBs of the obstacle are represented by Delaunay triangulations of the vertices that make up each AABB to form a convex hull of it. Each AABB is drawn separately as follows:

```
CompuGeom.FreeBoundaryPlot((double[,])Vertices, "g", Opacity, "-.", "k");
```

The input parameters are the vertices of the AABB as `Vertices`, the colour of the AABB as `"g"` (green), the opacity of the AABB faces as `Opacity` $= 0.99$ (99%), the line style of the AABB edges as `"-."` (dash-dotted) and the colour of the AABB edges as `"k"` (black).

The waypoints for each trajectory are drawn onto the axes in a similar way to the path itself, however the line style is specified as crosses so that they are clearly visible. As with the trajectory path, the coordinate arrays for the rear gripper path must be transformed into the rear trajectory reference frame to show correctly in the simulated workspace. They are drawn with the following code:

```
Plot.Plot3(Z, Y, X, "red", "+");
```

The final element to be drawn is the robot itself. When the trajectory graph window is opened for the first time, the robot trajectory is animated as the robot part positions are drawn at every

position along the trajectory from beginning to end. This is done by stepping through the part positions lists at each instant along the trajectory and drawing the robot parts as a set of line segments. As each new part position set is drawn, the previous one is deleted, giving the illusion of motion. The scrollbar in the "Trajectory Visualisation" group allows the user to select the particular robot position that is to be drawn in the simulated environment. In effect, this allows the user to visualise the robot configuration as it is moved through the trajectory manually, frame by frame. At any point, the graph may be rotated and zoomed providing a comprehensive view of the entire simulation.

5.1.2.9 Communication

The communication aspect of the application allows for the transfer of trajectories calculated by the ground station computer to the PLIR computer. It also provides a link between the two computers to provide a video feed from the robot to the PLIR Trajectory Solver application.

Once the required obstacle avoidance trajectories have been calculated, the trajectory data can be transferred. The transfer is done over a Wi-Fi link between the ground station computer and the PLIR computer. This communication has been implemented using the Windows Communication Foundation (WCF) of the .NET framework. It allows a server-client type communication protocol to be established that can be used to transfer data quickly between two computers on a network. The ground station computer is set up as a client that makes use of the PLIR computer service to transfer the trajectory data. The reason the ground station computer is set up as a client is so that it can initiate the service call when the data is ready to be sent, rather than the PLIR querying the ground station as to whether or not trajectory data is available.

The trajectory transfer window shown in Figure 5.7, is accessed by clicking the "Trajectory Transfer" button. The window is launched on a separate thread so that communication between the two computers does not interrupt the main application. The trajectory transfer window displays basic information about the trajectory solution for the selected gripper. The user is able to save the trajectory angle data to file for analysis in another application such as MATLAB. So long as the PLIR computer's controller application is running, the user can click "Transfer" to send the trajectory data.

The camera communication channel is implemented using a second WCF channel. As each frame is captured on the PLIR computer, it is transferred to the ground station. The communication link is set up on a separate thread so as to not disrupt the main PLIR trajectory solver application. Since the trajectory transfer and camera are on different communication ports and threads, video and trajectory data can be transferred simultaneously without affecting one another.

5.1.3 MATLAB integration

Part of the MATLAB® integration is achieved using the MATLAB Builder™ NE toolbox. MATLAB functionality can be incorporated into the .NET environment by using this toolbox that encrypts MATLAB programs and places them in .NET wrappers so that they can be used like a .NET component (Matlab, 2012).

The plotting and computational geometry components of the PLIR application are created using the MATLAB Builder™ NE toolbox, and each exists as a .dll file that is included in the project. Each file is made up of the desired functions that have been written as .m files and can be added to

the .dll to be deployed. In order to get the most basic functionality of MATLAB, such as plotting, one can simply encapsulate single MATLAB functions into stand-alone functions. Since MATLAB provides powerful tools for certain tasks such as 3D graphing, it makes sense to use this resource. Once a C# class has been created using the MATLAB Builder NE toolbox, the class can be added to the C# project by adding a reference to the .dll fille created by the toolbox. In order to use the class, an instance of the class can be created, and the functionality of the class can be accessed as with any regular C# class. For example, the "Plot" class created using the Builder NE toolbox used to display the robot's virtual environment can be created and used as follows:

1. Add the project reference to "Plot.dll".
2. Enter the reference into the "using" section of the project as "`using Plot;`".
3. Create an instance of the Plot class by declaring a new variable: "`PlotClass Plot = new PlotClass();`".
4. Make use of the necessary functionality, for example plotting 2D data: "`Plot.plot(X, Y);`".

The PLIR Trajectory Solver application can be deployed on any Windows XP (or later) based computer "royalty-free" (Matlab, 2012), even if it does not have MATLAB. The only requirement is that MATLAB Compiler Runtime 7.14 or later is installed on the machine.

The other method of integration used is known as Component Object Model (COM) technology that allows different software modules to communicate with each other. An instance of a MATLAB COM object is created, accessed and controlled from within the C# application. It essentially allows the PLIR Trajectory Solver application to incorporate MATLAB functionality.

For the PLIR Trajectory Solver application, the functionality of interest is that of the Simulink environment. The COM technology allows the Simulink model of the PLIR to be set up and executed from within the PLIR application, and the Simulink results can be retrieved from within the workspace of the MATLAB object. This model simulation output can be viewed in the PLIR application. The MATLAB COM object is also used to import the measured PLIR data that has been saved by the on-board PLIR Interface application discussed in Section 5.2.1.

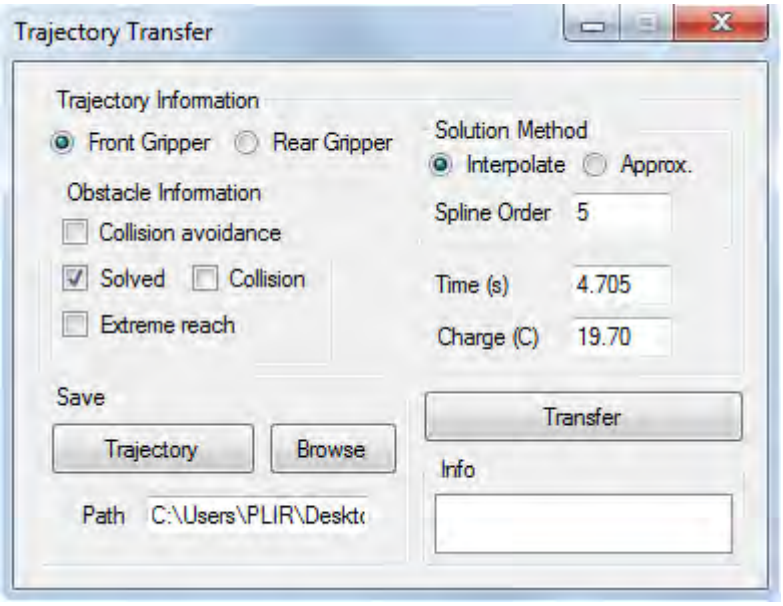

Figure 5.7: Figure showing the trajectory transfer window.

5.2 PLIR on-board control software

The PLIR On-board PC control application is of secondary importance with respect to this dissertation, and will therefore be briefly covered in this section for the purpose of context. The microcontroller software will also be covered here briefly.

5.2.1 On-board PC software

The "PLIR Interface" control application running on the on-board robot PC forms the interface between the ground station PC and the low-level microcontrollers on the robot. It is used to communicate with the the ground station PC using a Wi-Fi communication link. It also forms part of the crucial outer position control loop for the robot's joints.

At this stage of the robot's development, there is a requirement for a graphical user interface (GUI) for the on-board PC. This is so that crucial information regarding the status of the robot and the performance of trajectory execution can be monitored. The GUI of the PLIR Interface application is shown in Figure 5.8.

The GUI shows the measured position of each robot joint in real time in the "Measurements" group, as well as the initial trajectory angles of the desired trajectory in the "Initial Angles" group. In order to start/stop UART communication between the on-board PC and the microcontroller board (see Figure 5.9), the "Start Comms"/"Stop Comms" button must be clicked.

The PLIR Interface application also allows the operator to select between manual control or

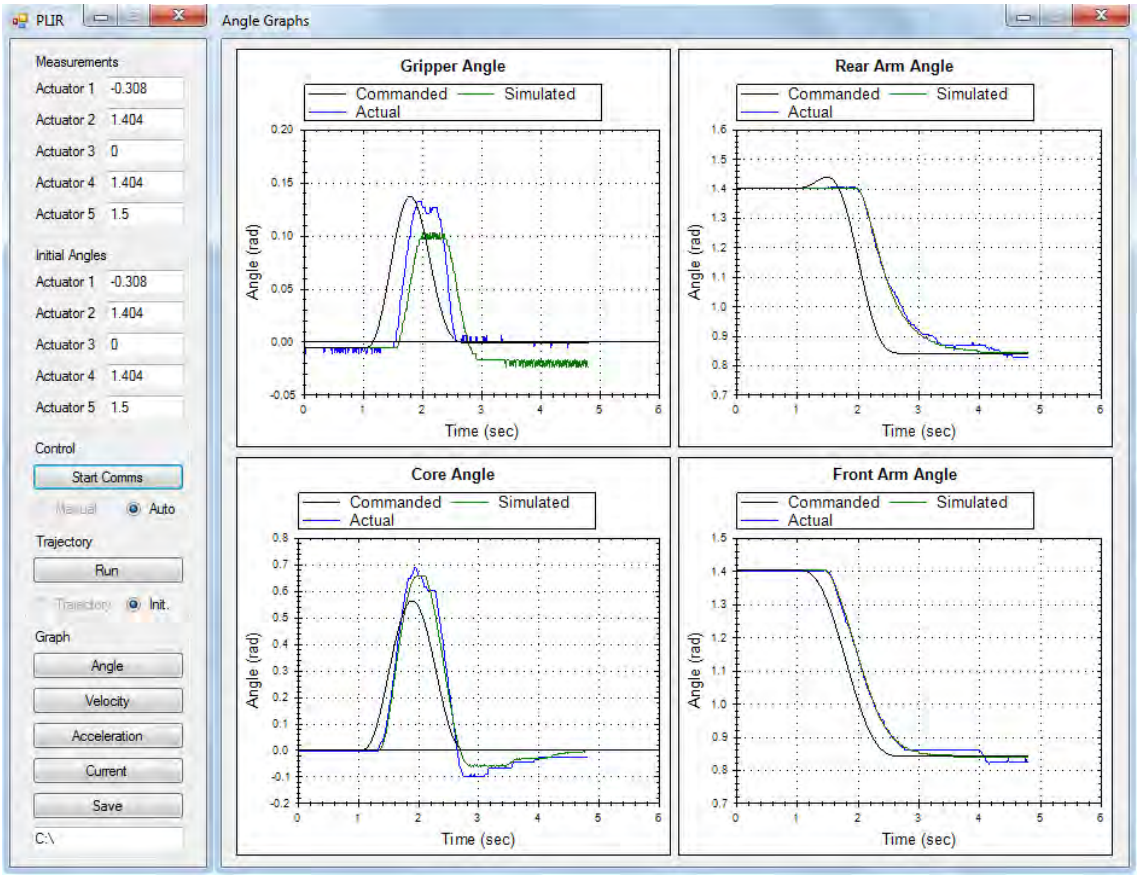

Figure 5.8: Figure showing the PLIR Interface GUI.

autonomous control. Manual control allows the operator to use a gamepad controller to control the individual robot joints. Autonomous control is used to execute the trajectories created by the PLIR Trajectory Solver application. The GUI allows the operator to initialise the robot before executing a trajectory (an important feature during the design phase), and then to execute the trajectory by clicking "Run".

The PLIR Interface application stores all of the trajectory data regarding joint positions, velocities and current profiles. This data can then be viewed by clicking on the relevant button in the "Graph" group. The commanded and measured components for each joint can be viewed on the same set of axes. Finally, the trajectory data that is collected during a manoeuvre can be saved to disk by clicking "Save". The data can then be imported into the PLIR Trajectory Solver application at a later stage for analysis. This allows for very fast and simple laboratory testing and performance evaluation.

5.2.2 Microcontroller software

There are two Atmel ATMega32 8-bit microcontrollers on board the PLIR. They are connected to each other in a master-slave configuration. The master microcontroller communicates with the on-board PC via UART as mentioned before. It receives new joint position commands at a rate of 67 Hz (every 15 ms) and sends back measured joint positions and actuator current information when each new command is received.

The master and the slave are each responsible for measuring and controlling position and current of the actuators connected to them. The slave sends its measurements to the master when it receives new commands from the on-board PC via the master. The two microcontrollers communicate using SPI (serial peripheral interface). The reason two microcontrollers are used is because of the limited ADC (analogue to digital converter) ports available on an ATMega32. The battery level is also monitored by one of the microcontrollers.

5.3 PLIR control system

In order to build the Simulink model of the PLIR for simulation purposes, the control structure of the robot must be modelled as well as the plant itself. This section covers the control system configuration, system identification of the actuator plant, verification of the Simulink model and finally, tuning of the robot controller using the Simulink model.

5.3.1 Control system configuration

The control setup for a single robot joint actuator is a closed-loop proportional-integral-derivative (PID) control system. Each joint actuator has been set up for closed-loop position control so that the robot joints can follow joint angle trajectories. Five control structure instances exist, one for each joint actuator of the robot. A simplified schematic of the control system configuration is given in Figure 5.9.

The control structure is made up of two control loops; an outer position control loop, and an inner current control loop. The PID controller is implemented in discrete form in the outer position control loop on the robot's on-board computer. A trajectory is transmitted from the base station via

Wi-Fi, and stored in the on-board computer. This trajectory is used as the controller reference for each joint. The angular position of each actuator (feedback signal) is provided by a potentiometer voltage that is measured by the microcontroller associated with the specific joint and transmitted via UART to the on-board computer. The PID controller has an input error of radians, and the controller output is a current command in units of Amperes. This output is transmitted to the microcontroller board as a reference set-point for the inner current control loop.

The inner current control loop is a simple proportional controller with saturation on the output, operating at a frequency of 14 kHz. The controller feedback is a 10-bit number from microcontroller ADC, measuring the current transducer's output voltage. Each current transducer is connected in series with the corresponding actuator armature winding. The feedback together with the set-point are used to create the output 8-bit PWM (pulse-width modulation) signal to drive the actuators using the H-bridges.

5.3.2 Simulink model

In order to create the Simulink model, the system parameters had to be determined. Some of the parameters such as armature resistance, armature inductance and motor torque constant were obtained from the datasheet. Other parameters such as friction coefficients needed to be measured experimentally. The parameter measurement procedures are not within the scope of this dissertation and are therefore not reported. The model parameters are provided in Appendix A.1. The model itself can be found in Appendix B.2.

With the correct parameters, the PLIR can be modelled using simple first order transfer functions to describe the physical components of the system. For example, the DC motor of the actuator can be modelled using two coupled first order differential equations; one to describe the electrical

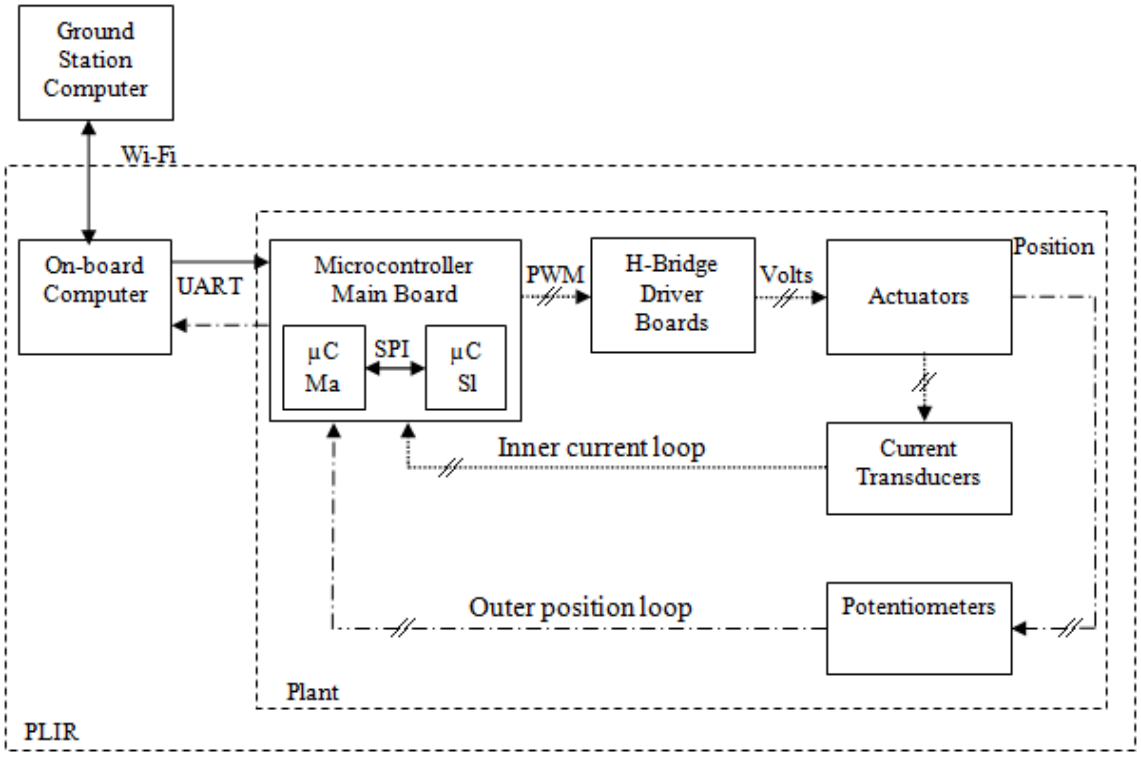

Figure 5.9: Simplified control system configuration.

characteristics of the system and another for the mechanical characteristics. All of these components are assembled in a Simulink model and this model is used to simulate the real system. The Simulink model takes into account some important system aspects:

- Discrete time controller including system delays
- Microcontroller and computer control characteristics
- Static, Coulomb and viscous friction
- Gearbox backlash (model adapted from Jukic and Peric (2001))
- Sensor noise and quantisation
- Static load torque coupling between joints

In order to verify all of the parameters, and ultimately the Simulink model, experiments were conducted to compare the performance of the actual and modelled systems. The Bode plot of the open-loop input (current command from the controller output) to output (robot joint position) transfer function of the plant was generated. This was then compared with the Bode plot of the linearised Simulink plant model at the same analysis points. The plant in this case comprises all components from the input of the microcontroller main board to the actuator position output as shown in Figure 5.9.

In order to obtain the input to output transfer function behaviour of the plant, sinusoidal signals of varying frequencies were applied to the position reference input of the entire closed-loop system at the PID controller summing junction input, and the plant position output was measured. The sinusoidal excitation frequency range was from 2.2 rad/s to 22 rad/s. The Ziegler-Nichols closed-loop PID tuning rule was used to achieve stable system behaviour before this test. The magnitude and phase relationship between the input and output was used to generate the Bode plots given in Figure 5.10.

The Bode plot shows that the Simulink model's behaviour is close to the physical system's behaviour, which means the Simulink model is sufficiently accurate to be used for controller design and system simulation (behaviour prediction). The slight difference between the physical system and the model in terms of phase shows that there is a small delay in the real system that has not been accounted for in the model.

5.3.3 PID controller tuning

Once a verified model of the robot has been created, a linearised version can be used to tune the PID controller using MATLAB's SISOTool. SISOTool (single-input-single-output tool) can be used to create controllers and filters for a given system using graphical techniques. In the case of the PLIR, the graphical tuning tools of interest were:

- Nichols chart - stability analysis
- Bode plot - input/output relationship
- Step response - system performance

The PID controller parameters were designed so that the system had satisfactory gain and phase margins, with sufficient performance. The open-loop Nichols chart, closed-loop Bode plot, and

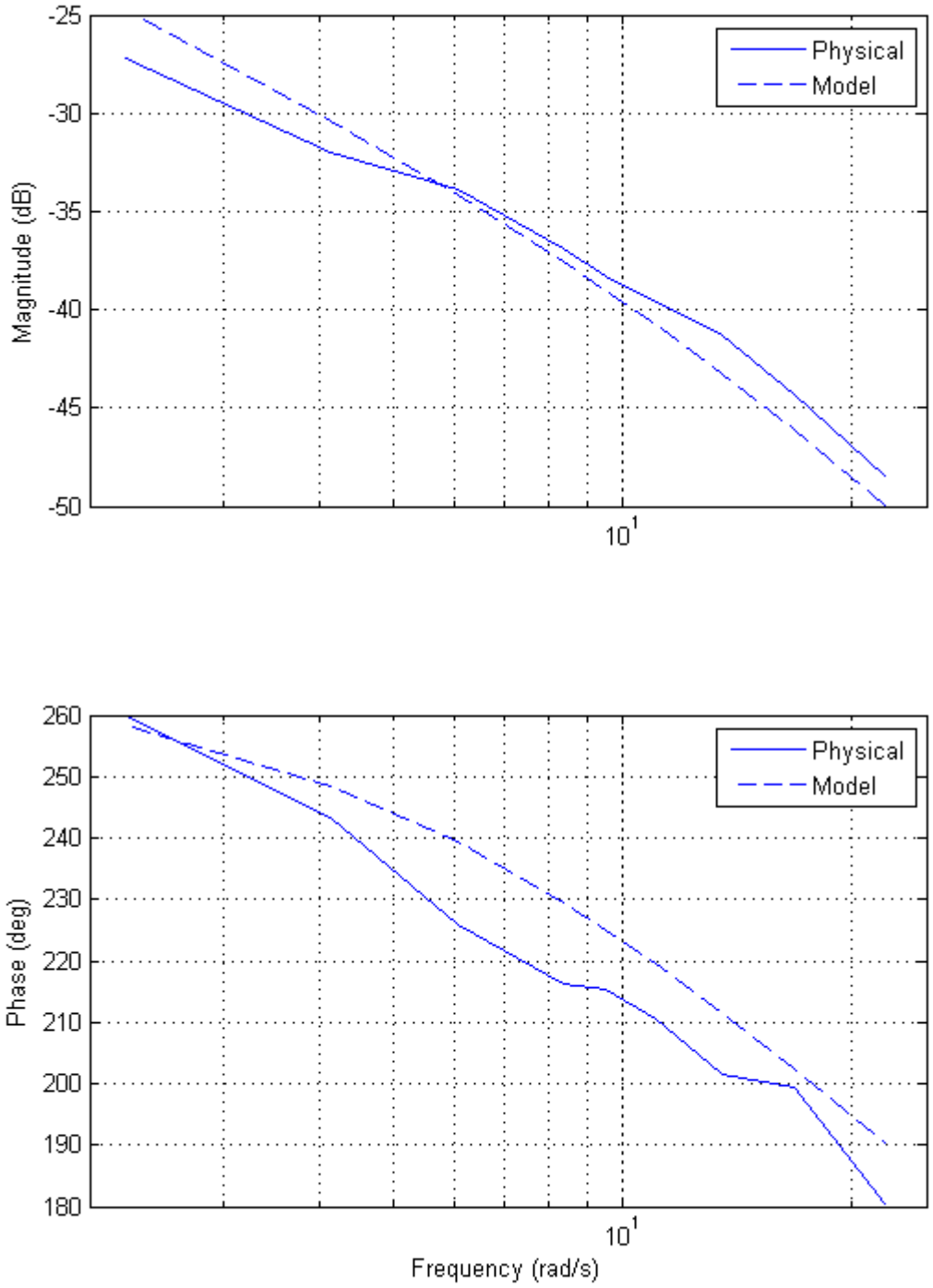

Figure 5.10: Bode response of physical and modelled systems.

closed-loop step response of the system are given in Figure 5.11, Figure 5.12 and Figure 5.13 respectively. The frequency labels in Figure 5.11 are in rad/s. The PID controller parameters can be found in Appendix A.2. The transfer function of the PID controller is given in series form in (5.1) and is implemented in discrete-time form on the robot's on-board PC. T_i is the integrator time constant, T_d is the derivative time constant and k_p is the proportional controller gain.

$$G(s) = k_p \left(\frac{sT_i + 1}{sT_i} \right) \left(\frac{sT_d + 1}{sT_d\alpha + 1} \right) \tag{5.1}$$

As shown in Figure 5.12, a prefilter was used to remove unnecessary high frequency gain to reduce the likelihood of exciting any mechanical/electronic resonant frequencies. The prefilter is placed before the summing junction input, as shown in Figure 5.14. It is a first order low pass filter with a corner frequency of 40 rad/s. The transfer function of the filter is given by $F_{pre}(s) = \frac{40}{s+40}$. The controller design provides good system performance and stability as shown in Figure 5.13.

5.4 Hardware

The hardware used for this master's dissertation, specifically the power line inspection robot itself, existed before the start of this master's programme. However, during the course of the degree, many low level engineering issues were addressed during commissioning and testing of the PLIR. Some of the upgrades include:

- Improving communication reliability between the microcontroller board and the on-board PC
- Upgrading actuators (with the assistance of Trevor Lorimer) which failed repeatedly while being operated well within the manufacturer's specifications
- Adding current transducers to replace less accurate current sensing resistors
- Suppressing significant circuit noise to improve sensor readings

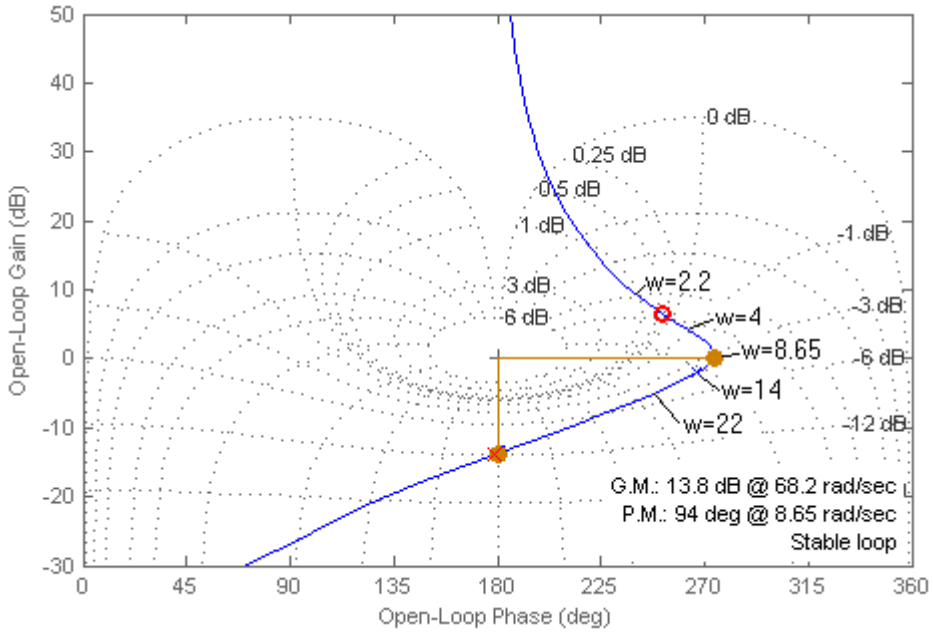

Figure 5.11: Open-loop Nichols chart of the actuator model.

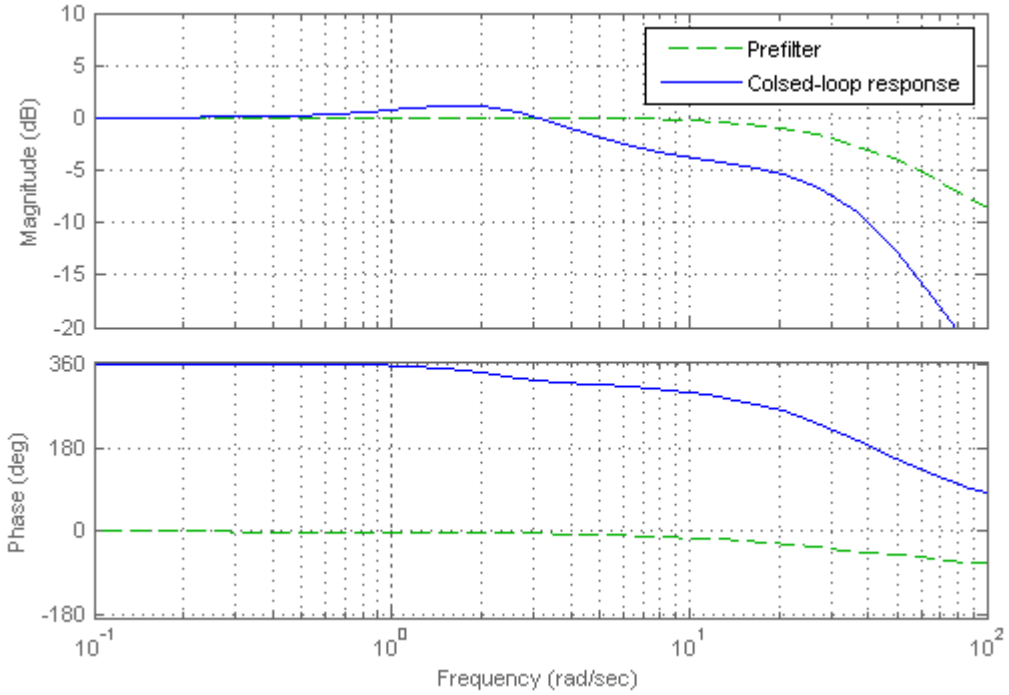

Figure 5.12: Closed-loop Bode plot of the actuator model showing the prefilter.

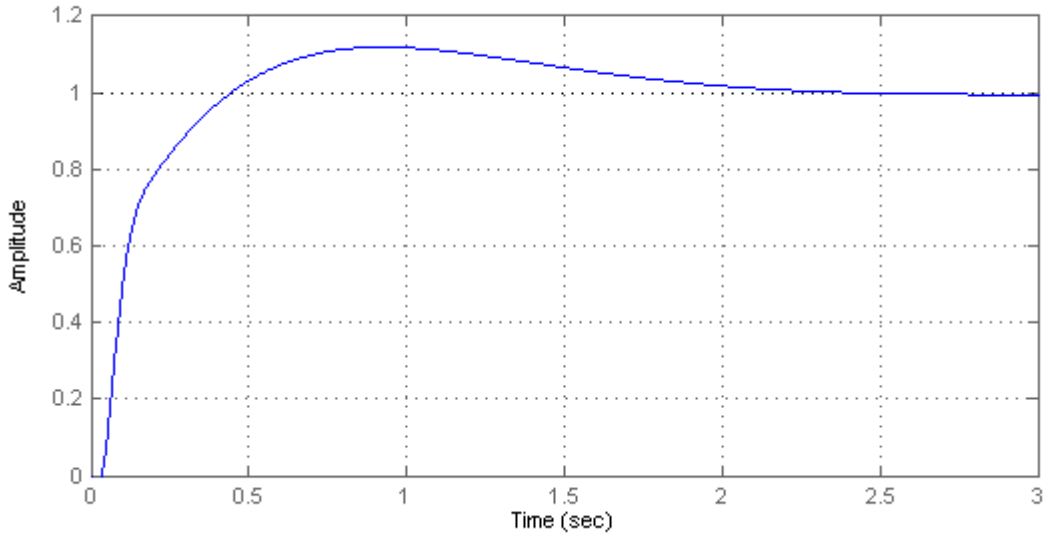

Figure 5.13: Closed-loop step response of the actuator model.

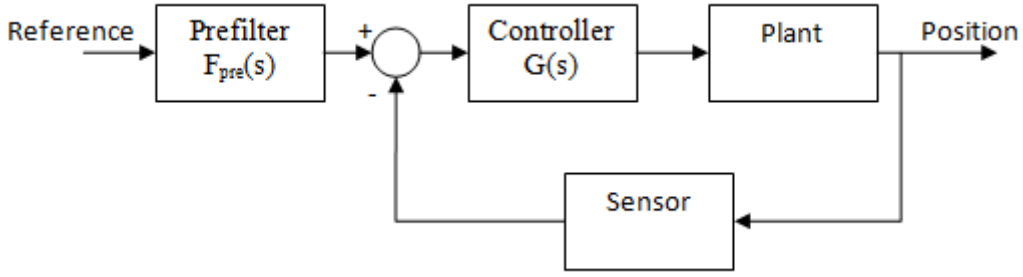

Figure 5.14: PLIR Control loop architecture.

A new hardware addition that has been used to implement the work described in this dissertation is the on-board PLIR computer. The PLIR computer is a fitPC-2i which is an entire computer in a $101 \times 115 \times 27\,\text{mm}$ fanless aluminium casing. The ground station computer used is simply a laptop PC (not quite as exciting).

5.4.1 Ground station computer

The laptop that was used for a ground station computer during this master's programme is a Toshiba Satellite Pro A300. It was used to develop all of the software for this project, and was used as the PLIR Trajectory Solver application PC for the robot simulation and testing. This computer has a Wi-Fi module to enable connectivity between itself and the PLIR computer. At this point in development, the ground station computer is responsible for all high-level processes such as trajectory computation and optimisation, and the robot's on-board computer merely carries out instructions provided by the ground station. However, as development of the robot continues with respect to autonomy, high-level tasks will be migrated to the fitPC-2i so that it is less reliant on the ground station computer. The ground station computer specifications are given in Table C.1 in Appendix C.1.

5.4.2 PLIR computer

The computer used on the PLIR is a fitPC-2i and is used for the high-level communication between the robot and the ground station, as well as the low-level control over the robot microcontrollers via RS-232. The fitPC-2i also runs a Windows operating system, which makes development of the communication and controller application relatively easy since the same Microsoft Visual Studio 2010 Express environment is used. During development work, it is connected to a monitor and has a mouse and keyboard. This allows the software to be developed on-board, and provides a debugging platform for the low-level controllers implemented on the microcontrollers. The computer is also host to the robot cameras. The fitPC-2i specifications are given in Table C.2 in Appendix C.2, along with a photograph of the computer.

Chapter 6

Experimental Results

In order to investigate the validity of the obstacle avoidance and trajectory optimisation techniques developed in this dissertation, obstacle avoidance trajectories were optimised and simulated in the PLIR Trajectory Solver application, and used to command the actual robot in the laboratory. The predicted trajectories were then compared with the actual measurements to determine whether or not:

1. The obstacle avoidance manoeuvre is collision-free (successful)
2. The predicted charge consumption characteristics are accurate (predicted and measured current correspond)

Two obstacles have been used to generate obstacle avoidance trajectories for the experiments. The first obstacle is a suspension clamp (Figure 6.4), and the second is a power line jumper cable pistol grip (Figure 6.13).

Section 6.1 presents the results for an optimised obstacle avoidance manoeuvre to get around a suspension clamp. Trajectory optimisation for this manoeuvre was achieved using both the ID and the Simulink model, and the measured behaviour is compared with the predicted behaviour. Section 6.2 presents the results for an optimised jumper cable manoeuvre. Section 6.3 presents the results of a high voltage test that was conducted in order to test the robustness of the PLIR systems in a high voltage environment.

6.1 Suspension clamp manoeuvre

As shown from Figure 6.4 to Figure 6.7 and in the accompanying videos (Appendix B.3), the suspension clamp test rig was set up with the robot placed on a wooden rod that represents a power line segment, and the suspension clamp attached to the wooden rod. A one metre ruler was used to provide scale in the video footage for verifying the Cartesian locations of the robot parts and the obstacle.

The trajectory required for the manoeuvre was created by the PLIR Trajectory Solver application (Section 5.1) with the suspension clamp obstacle representation entered into the GUI. The autonomous trajectory solver was set up to create a fifth order B-spline trajectory using interpolation, and the solution was then optimised using the interval time optimiser method presented in Chapter 4.

The suspension clamp AABB was defined using the following minimum and maximum boundary coordinates (in metres) (x, y, z):

- Min = (−0.09, −0.08, 0.23)
- Max = (0.04, 0.08, 0.55)

The waypoints that were calculated by the solver to create the obstacle avoidance trajectory $\vec{s}(\bar{\theta}(t))$ were (x, y, z):

- $\{\underbrace{(0,\ 0,\ 0.15)}_{\text{initial}},\ \underbrace{(0,\ 0.08,\ 0.23),\ (0,\ 0.08,\ 0.55)}_{\text{intermediate}},\ \underbrace{(0,\ 0,\ 0.6)}_{\text{final}}\}$

Only the front gripper trajectory was tested as a proof of concept, since the success of a front gripper trajectory indicated that the method will be successful for the rear gripper as well. The generated trajectories were sent to the PLIR Interface application using the Wi-Fi connection.

The position and current data taken directly from the robot microcontrollers was logged on the PLIR fitPC-2i computer using the PLIR Interface application discussed in Section 5.2.1. The velocity data was generated using a complementary filter with the potentiometer position measurements providing the low frequency component and the current measurements providing the high frequency component using the actuator motor model. The high frequency estimate is given by (6.1), where the back emf is estimated from V_{PWM} (applied motor voltage), I_{meas} (current measured by the current transducers), and R (armature resistance). k_ϕ is the motor constant and r is the gearbox ratio. The armature inductance is negligible and has been omitted.

$$\omega = \frac{V_{PWM} - I_{meas}R}{k_\phi r} \tag{6.1}$$

Before each trajectory is executed, the initial joint angle commands are applied to the joints for one second to provide a steady state starting point. Furthermore, since there is lag between the trajectory command and the robot's movement (visible below in Figure 6.1), the final joint trajectory position command was held steady at the end of each suspension clamp test for two seconds. This allowed the robot joints enough time to reach their final set points before the test terminated. The cause of this delay in movement is discussed below.

6.1.1 Simulink model test

The trajectory and simulation predictions were generated using the PLIR Trajectory Solver application with the detailed Simulink model. The results presented here show the commanded, simulated and measured performance of the PLIR. Figure 6.1 shows the commanded, simulated and actual movements of the robot's joints.

The robot followed the trajectory that was commanded, but with lag. There is a significant delay from the time the position command changes to the time the robot begins to move due to static friction (discussed below). Also, the rear arm joint does not keep up with the commanded trajectory due to it reaching its velocity limit, as marked in the "Rear Arm Velocity" graph of Figure 6.2.

The Simulink model of the PLIR is sufficiently accurate to predict this behaviour. The simulated and actual joint movements correspond closely, and this verifies the Simulink model. The noise present in the actual velocity measurements is due to noise in the current measurements, which is discussed below.

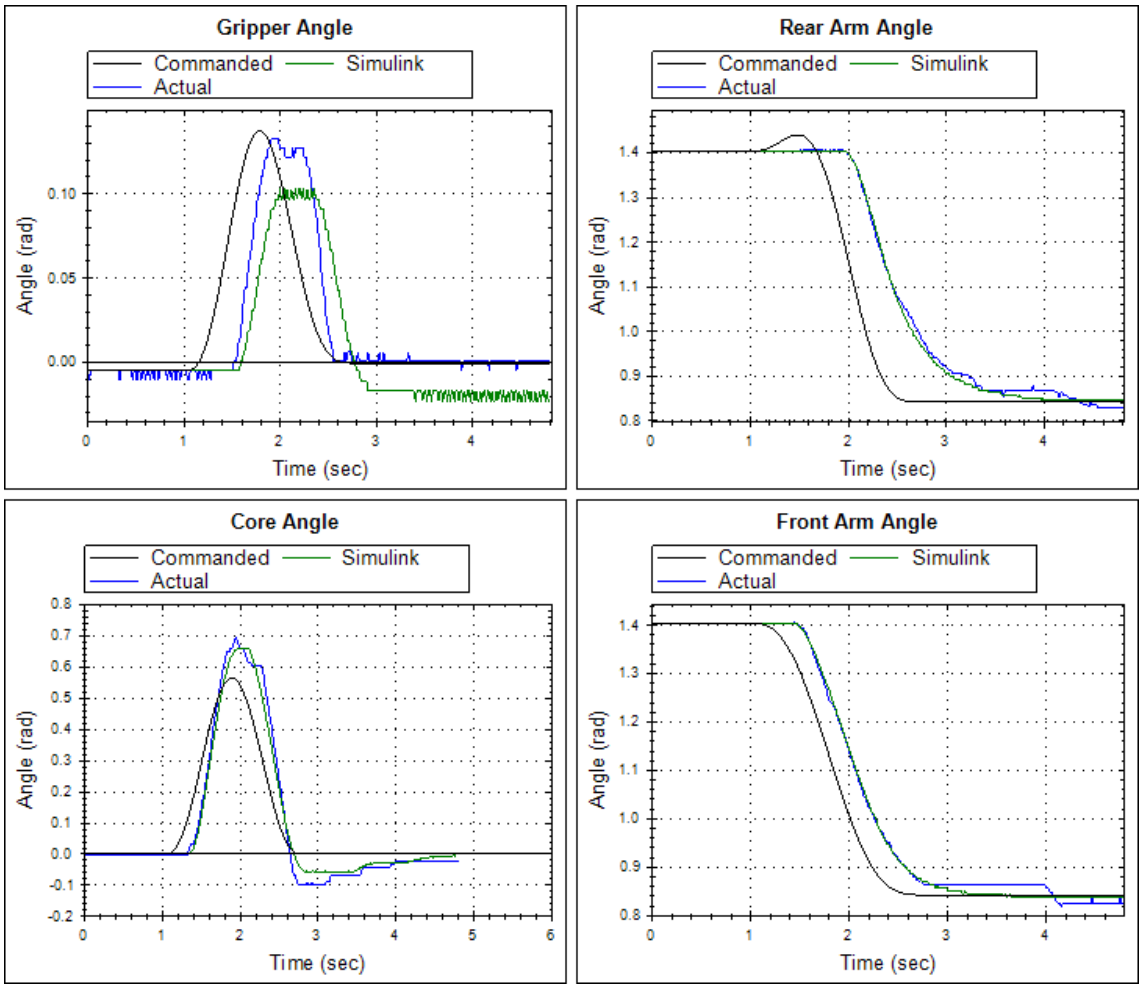

Figure 6.1: Simulink model optimised trajectory: commanded, simulated and actual robot joint angles for the suspension clamp manoeuvre.

The trajectory command has a time of 1.8 s and optimised interval times of $\bar{r} = [0.7\,\text{s}, 0.6\,\text{s}, 0.5\,\text{s}]$, with a 1 s and a 2 s trajectory command holding time at the beginning and end of the trajectory respectively.

The delay in movement can become a problem in obstacle avoidance situations when the joints do not move in a synchronised manner. In this particular case, the core joint completes its movement significantly sooner than the other joints. This means that the end effector moved back toward the power line before it had properly passed the AABB obstacle representation. This can be clearly seen in Figure 6.3.

In this experiment, the AABB violation was not a problem. The AABB had over bounded the suspension clamp in the first place, and the end effector did get around it as shown from Figure 6.4 to Figure 6.7. Also, the robot achieved the final set-points with sufficient accuracy to get the end effector back onto the line.

As mentioned above, static friction present in the robot joints is the primary reason for this delay in motion, with the integrator action in the PID controller not being fast enough to compensate. The position error at the controller input begins to drive the controller output and hence the actuator when the position command begins to change. However, due to the static friction, the joints can only move when the controller output (actuator current) is large enough to overcome the friction.

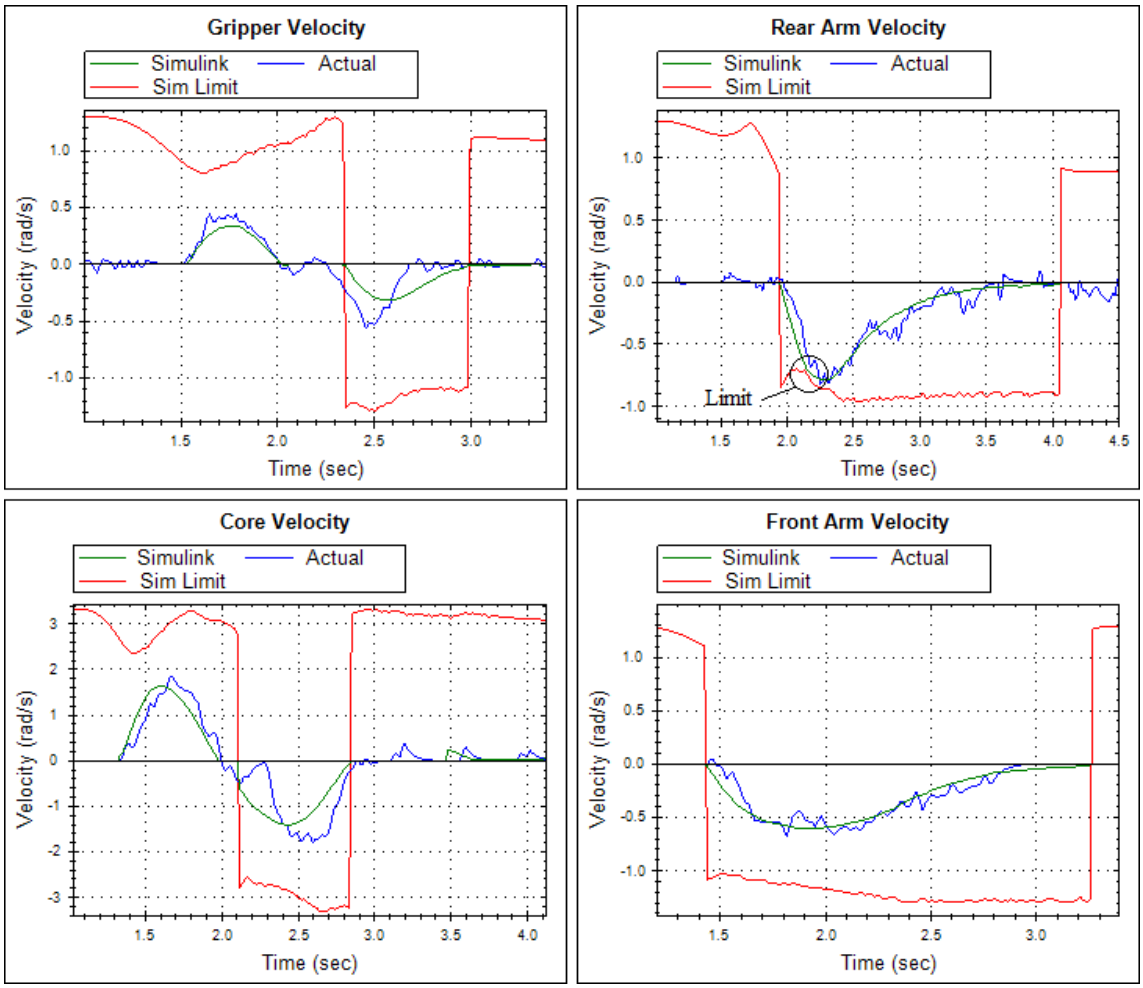

Figure 6.2: Simulink model optimised trajectory: simulated and actual robot joint velocities for the suspension clamp manoeuvre.

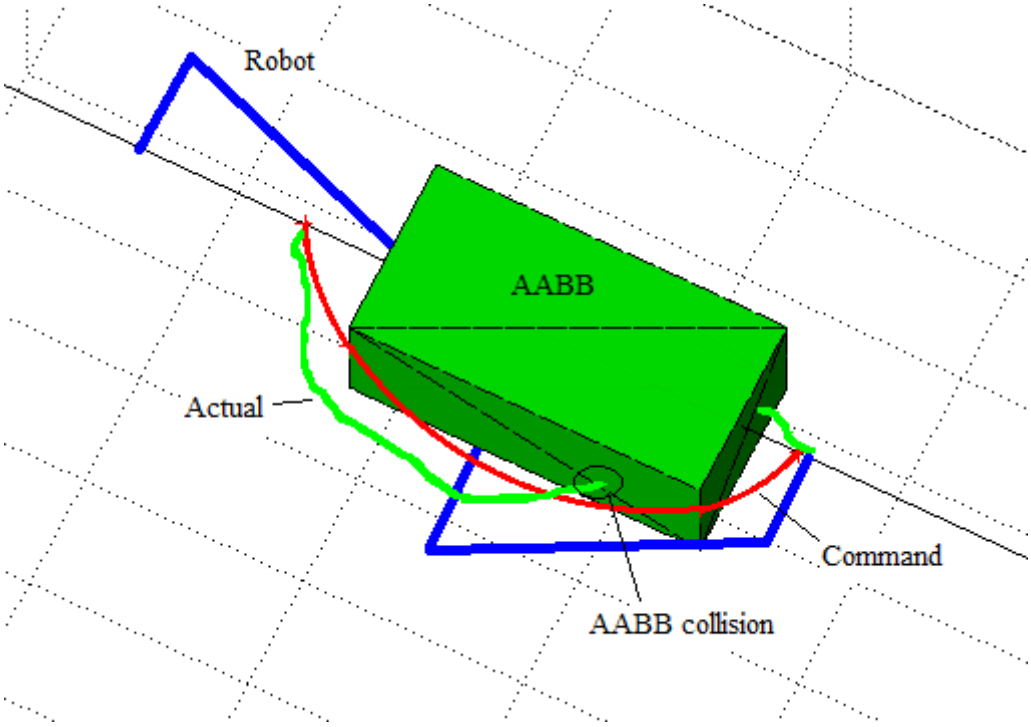

Figure 6.3: PLIR simulation environment showing AABB collision.

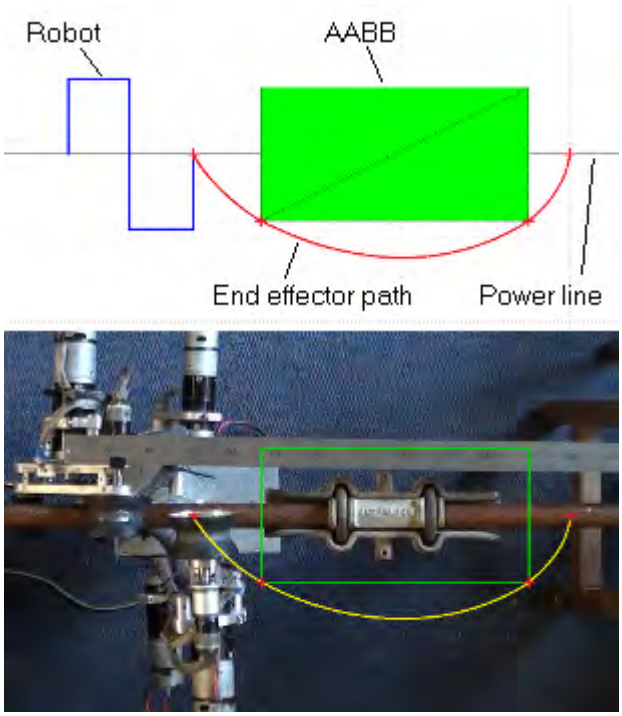

Figure 6.4: PLIR simulation together with the laboratory test video frame of the suspension clamp obstacle avoidance manoeuvre at time $t = 0$.

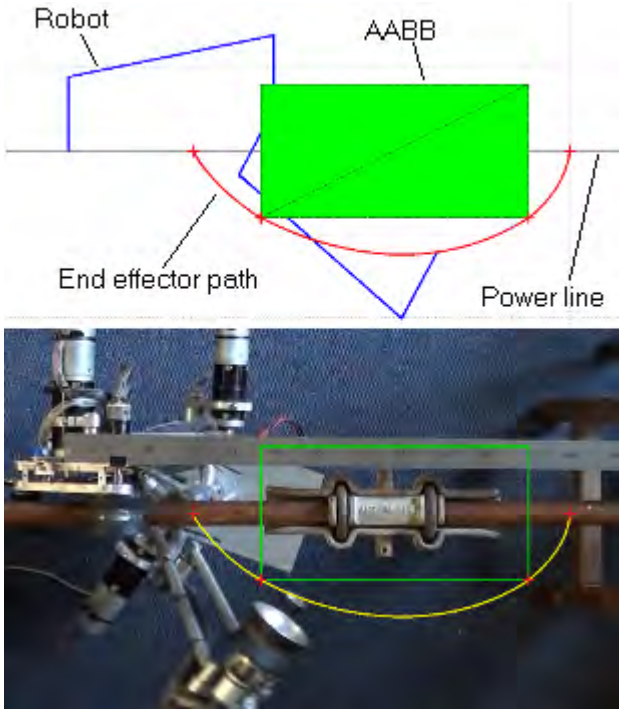

Figure 6.5: PLIR simulation together with the laboratory test video frame of the suspension clamp obstacle avoidance manoeuvre at time $t = 2.09$.

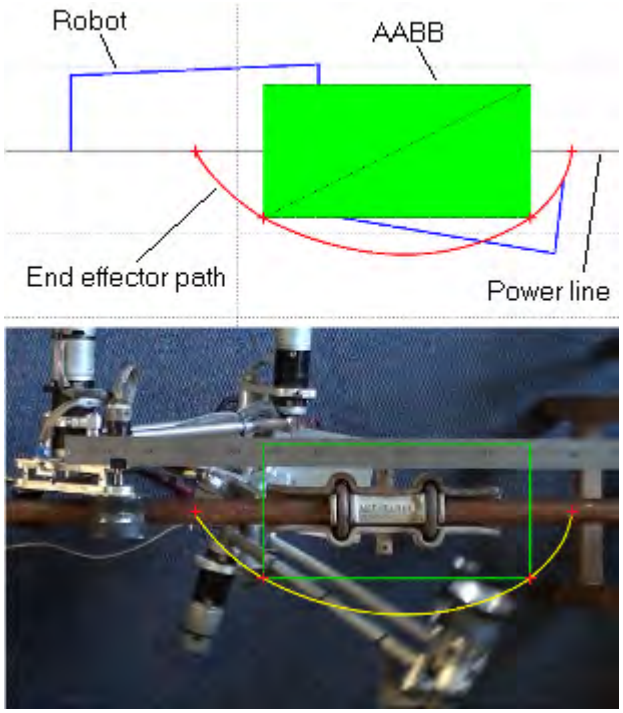

Figure 6.6: PLIR simulation together with the laboratory test video frame of the suspension clamp obstacle avoidance manoeuvre at time $t = 2.48$.

This effect is highlighted in Figure 6.8. The "Break-away" labels show where the static friction was overcome and the joints began to move. As seen in Figure 6.8, the current decreased after the break-away point because the torque due to viscous and Coulomb friction is less than that of static friction, and the position error was decreasing. The break-away points in Figure 6.8 can be matched up with the same time points in Figure 6.1 where the joints began to move.

Furthermore, the static friction plays a part at the end of the trajectories where the joints may have not met the end set-point exactly. There are clear cases in Figure 6.8 in the "Gripper Current" and "Front Arm Current" graphs; there is integrator wind-up in the PID controller due to the static friction. This will cause a limit cycle if the controller is left to hold the robot's position. An improvement to the PID controller would be to "switch-off" the integrator once the robot had met the desired set-point with sufficient accuracy.

From Figure 6.8, it is clear that the Simulink model of the robot is sufficiently accurate to predict the current profile for each actuator during a manoeuvre. From these profiles, the profile for the total current required can be calculated which can be used to determine the charge required from the battery for the manoeuvre. Not only does this verify the model used for charge requirement predictions, but in turn it verifies the optimisation techniques presented in Chapter 4 using the Simulink model. The cost at each optimisation iteration for a set of time intervals can be predicted with sufficient accuracy, and at each iteration, the interval time adjustments are being made according to a model that closely represents reality.

Developing the Simulink model provided much insight to the behaviour of the joint angle and current profiles seen in Figure 6.1 and Figure 6.8. Apart from the break-away current required to get the joints moving, there are some other noteworthy details in Figure 6.8.

The high frequency noise present in the actual current measurements is due to noise present in the position measurement signals. This is because the current command is derived from the

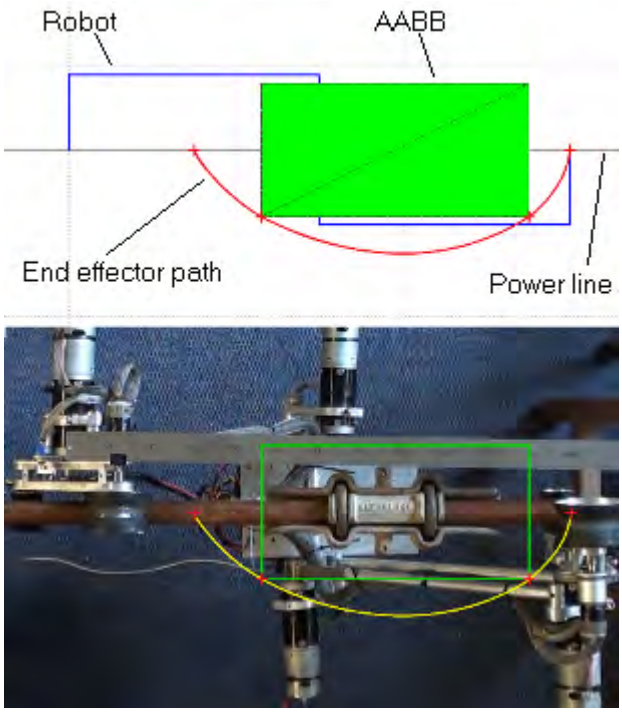

Figure 6.7: PLIR simulation together with the laboratory test video frame of the suspension clamp obstacle avoidance manoeuvre at time $t = 4.5$.

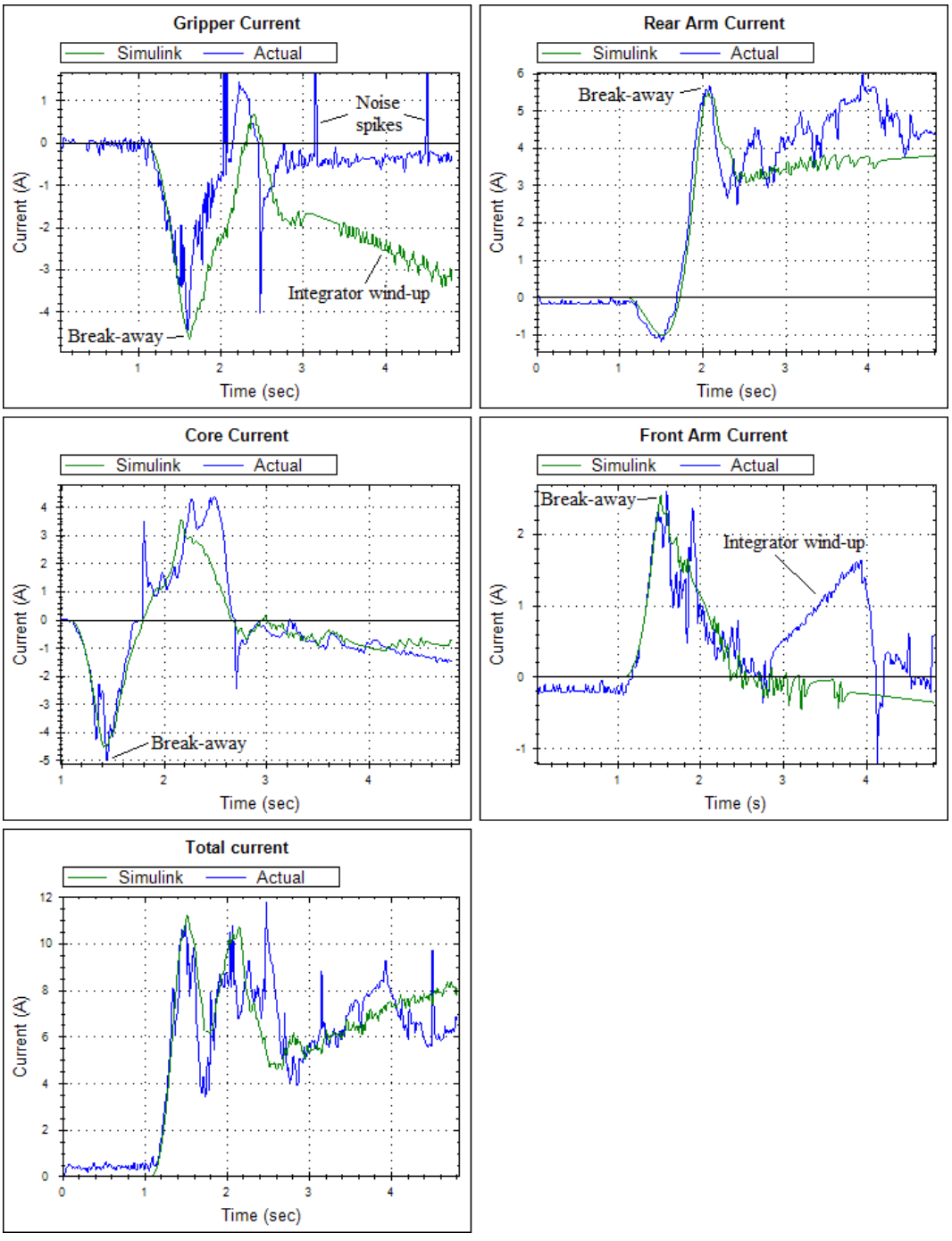

Figure 6.8: Simulink model optimised trajectory: Simulink model and actual robot actuator currents for the suspension clamp manoeuvre.

position error of the controller input, and of course, this noise propagates through the controller and out to the actuators. The cause of the position measurement noise is due to the existence of noise in the potentiometers' reference voltage. The noise source is the 3.6864 MHz oscillator on the microcontroller board. Attempts were made to filter out the noise, but with limited success. However, this is not of major concern, since the successor to this first PLIR prototype makes use of digital position encoders which are immune to such noise sources.

The "noise spikes" present in the "Gripper Current" window are due to momentary communication glitches between the master microcontroller and on-board PC. When a communication packet between the on-board computer and the master microcontroller is dropped (lost), the slave microcontroller receives an erroneous current set-point command, causing an ephemeral spike in the PWM voltage and hence the current. This bug was not a cause for major concern during these tests because it did not affect the overall performance of the robot. Furthermore, this robot is to be superseded by a superior prototype, and so fixing this bug would be pointless at this stage.

6.1.2 ID model test

The trajectory and simulation predictions for this test were generated using the PLIR Trajectory Solver application with the inverse dynamics model. The results presented here show the commanded, simulated and measured performance of the PLIR. Figure 6.9 shows the commanded, simulated and actual movements of the robot's joints. The simulated behaviour is based on the detailed non-linear Simulink model even though the trajectory design is based on the ID solution.

The execution time for this trajectory command is 1.8 s and has optimised interval times of $\bar{r} = [0.6\,\text{s}, 0.7\,\text{s}, 0.4\,\text{s}]$, once again with a 1 s and a 2 s trajectory command holding time at the beginning and end of the trajectory respectively. When comparing Figure 6.1 and Figure 6.9, it is clear that the generated trajectories are very similar regardless of the model used. This is because the predicted torque profiles of the two models are similar at the limits of the expected torque. Since the allowable joint torques form the dominant robot design limitations and trajectory optimisation constraints, the overall trajectory times will need to be similar to obey the same torque constraints. A comparison of the ID and Simulink models' expected torque is given in Figure 6.10.

The static cases of the two models' torques are very similar; at the initial steady state they coincide, and at the end of the trajectory they are less than 1 N.m apart (due to final set-point error). As shown in Figure 6.10, the gripper torque is very close to the design limit in both the ID and Simulink model cases. This would have been the limiting factor during trajectory optimisation.

On the other hand, the dynamic cases of the two models are different. The main reason for this is due to the fact that the Simulink model takes into account linear and non-linear friction, and backlash whereas the ID model does not. However, the ID model does take into account the dynamic coupling of joint torques. The effect of static friction can be clearly seen in the Simulink predictions, and the static friction break-away point for the rear arm joint is annotated in Figure 6.10. The noise present in the Simulink model's torque profiles is due to the modelled sensor noise as discussed in Section 6.1.1.

The ID model, Simulink model and actual current profiles for this test are shown in Figure 6.11. As in Figure 6.8, the Simulink model and the actual current profiles agree sufficiently to give confidence in the overall engineering solution. However, as with the torque predictions, the ID model current prediction is very different since friction is not accounted for in the ID model.

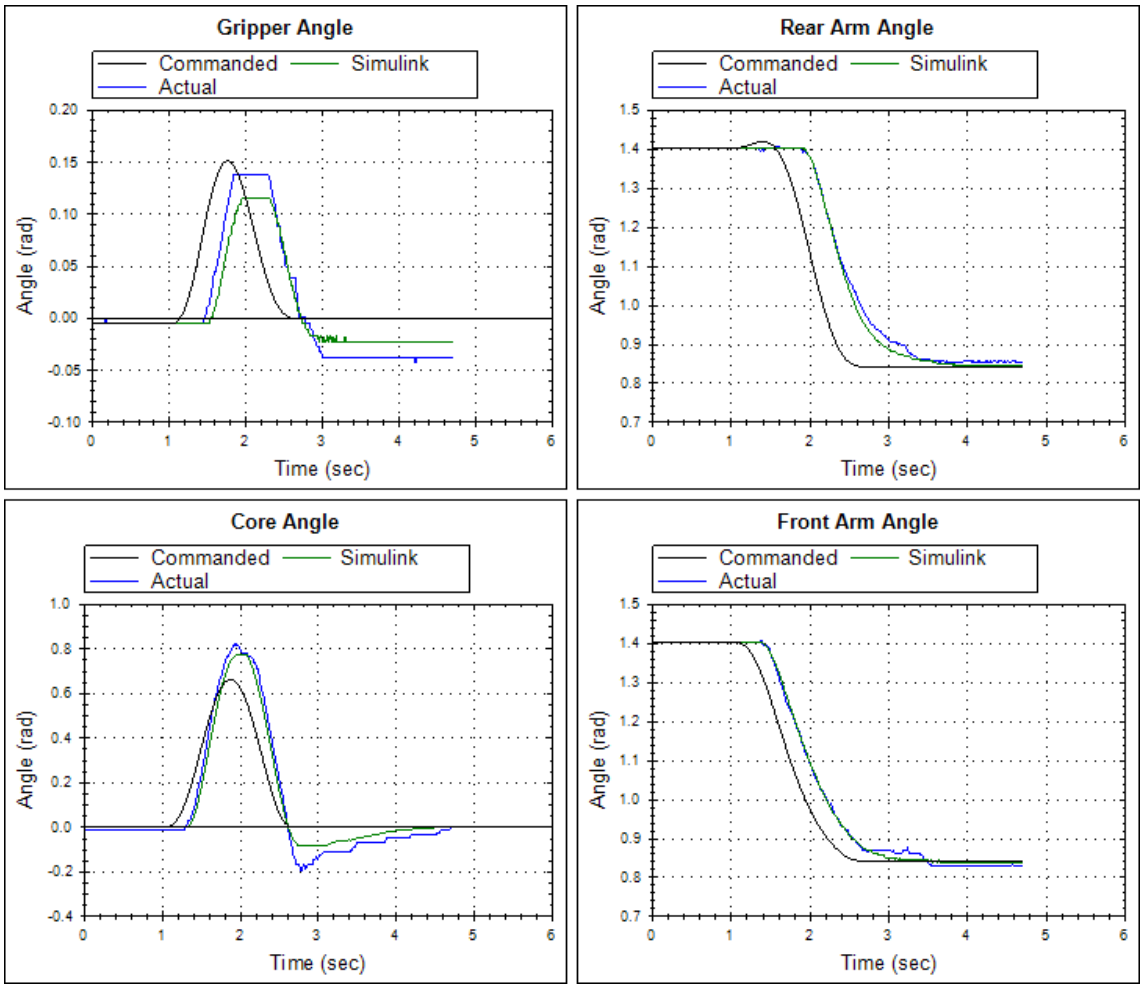

Figure 6.9: ID model optimised trajectory: commanded, simulated and actual robot joint angles for the suspension clamp manoeuvre.

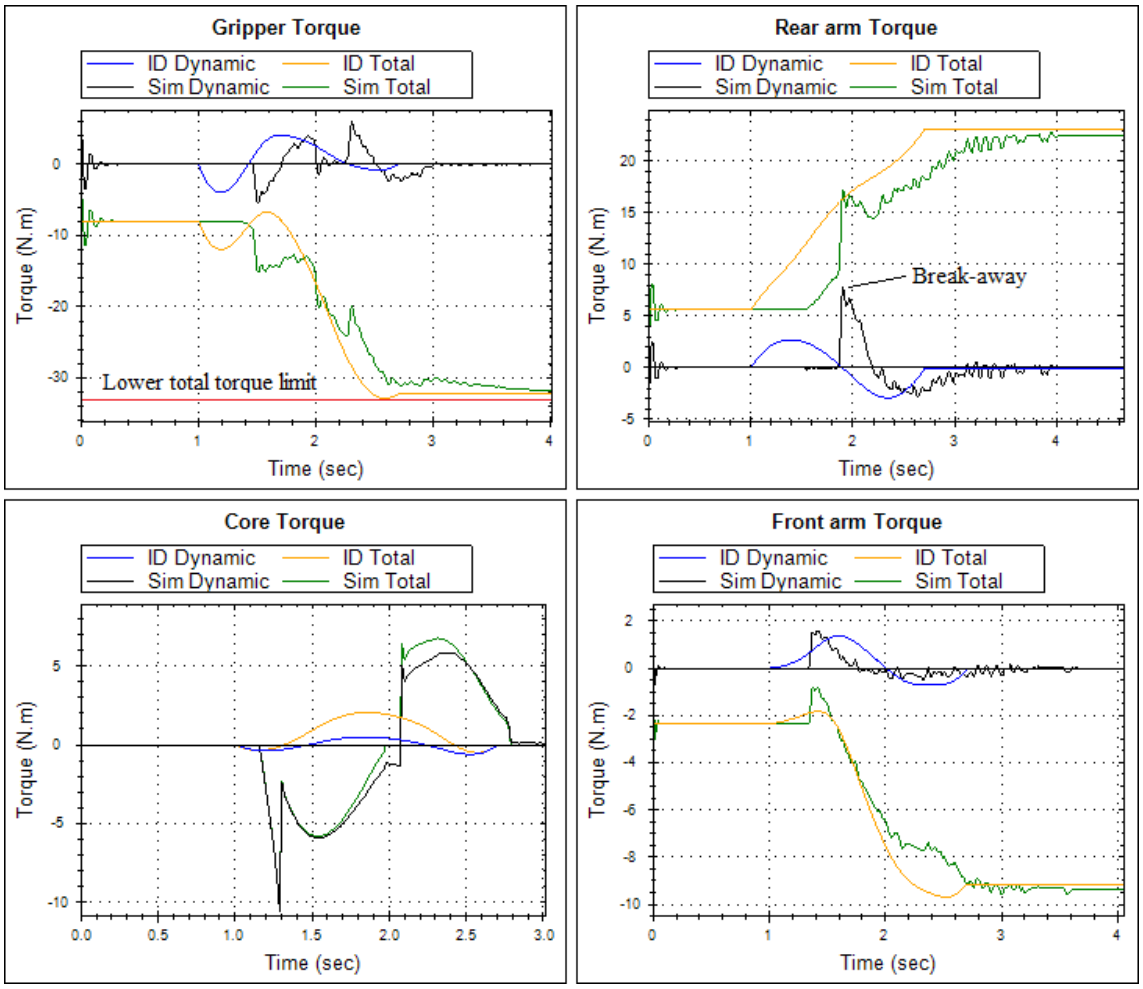

Figure 6.10: ID and Simulink model predicted joint torque profiles for the suspension clamp manoeuvre.

Comparing the dynamic components of the ID and Simulink model torques in Figure 6.10, it is clear that the two are in agreement with one another aside from the friction effects. For this reason, it is reasonable to assume that for an improved model of the PLIR with superior hardware and relatively low actuator friction, the ID model's predicted torque and current may be sufficiently accurate for trajectory optimisation.

The ID model iterations can be processed two orders of magnitude faster than the Simulink actuator model iterations. Especially when performing trajectory optimisations online, this reduction in computation time has obvious benefits.

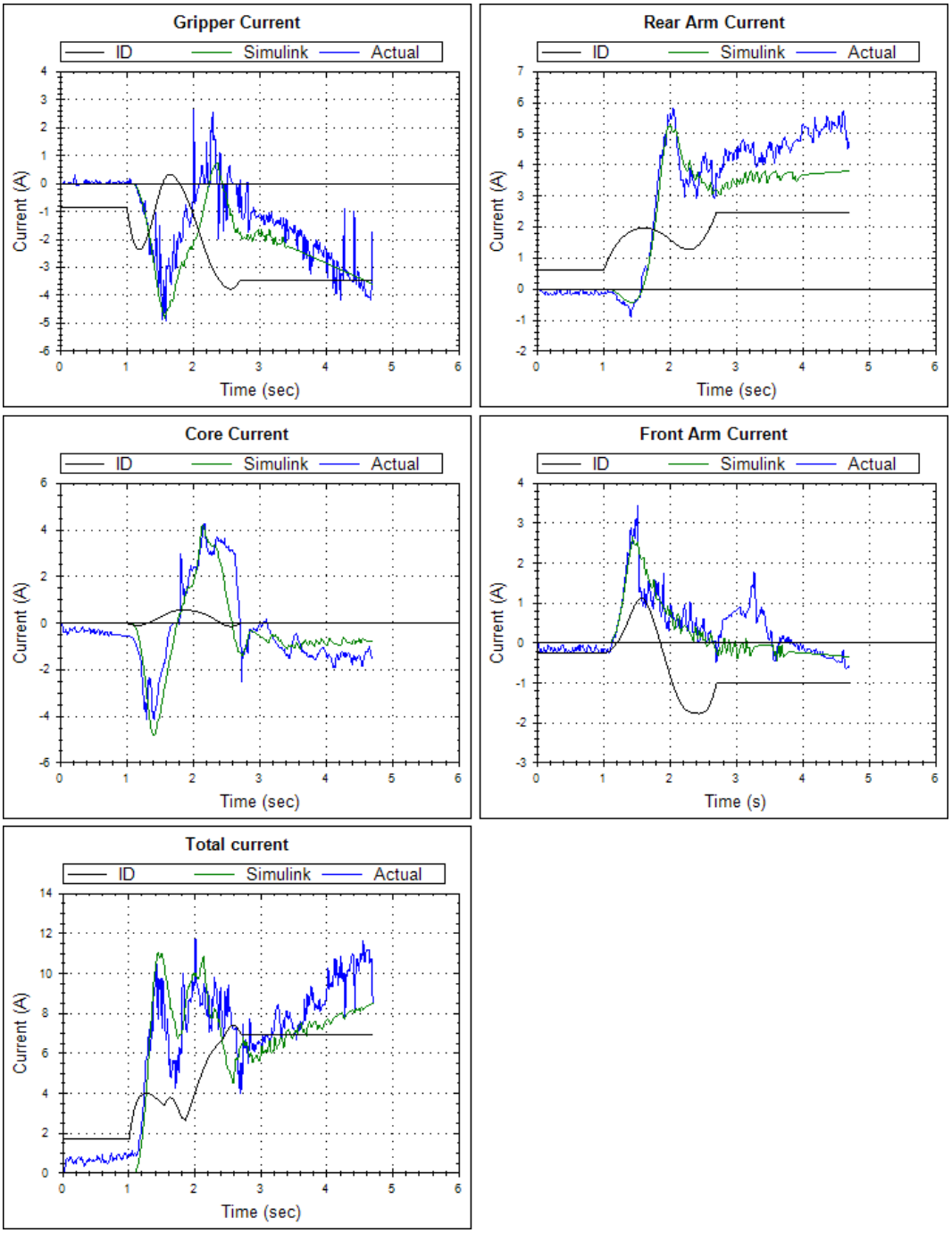

Figure 6.11: ID model optimised trajectory: simulated and actual actuator current for the suspension clamp manoeuvre.

6.2 Jumper manoeuvre

The jumper test was conducted in the laboratory on a small power line segment that has a jumper cable pistol grip. The trajectory required for the manoeuvre was created by the PLIR Trajectory Solver application, but this time the waypoints were manually entered into the GUI for testing. A trajectory was created and optimised, using the Simulink model, to be executed on the real robot. As with the suspension clamp test, a 1 s and a 2 s trajectory command holding time was applied to the beginning and the end of the trajectory respectively.

The waypoints used to create the obstacle avoidance trajectory $\vec{s}(\bar{\theta}(t))$ were (x, y, z):

- $\{\underbrace{(0,\ 0,\ 0.15)}_{\text{initial}},\ \underbrace{(-0.1,\ 0.1,\ 0.4)}_{\text{intermediate}},\underbrace{(-0.27,\ 0,\ 0.5)}_{\text{final}}\}$

Figure 6.12 shows the commanded, simulated and actual movements of the robot's joints for the jumper manoeuvre. As expected from Section 6.1, the behaviour of the robot was predicted by the Simulink model.

The robot manoeuvred the gripper past the jumper pistol grip obstacle, and positioned it on the jumper cable section ready for the gripper to re-attach to the line. Video footage frames of the jumper manoeuvre are given in Figure 6.13. For the video clip of the manoeuvre, refer to Appendix B.3.

6.3 High Voltage Testing

The PLIR was tested in the high voltage (HV) laboratory of the University of the Witwatersrand in collaboration with other members of the PLIR team and the Wits HV team. The purpose of the HV testing was to test the robustness of the engineering design of the power electronics, the control system, sensors, and the communication system (base station and on-board PC) in a high voltage environment. Standard engineering practice with respect to earthing and shielding of the PLIR was undertaken to prepare the robot for HV testing.

The test rig consisted of a mock live line segment, and an insulted rod (hot stick) connected to the line segment on one end and to ground at the other end. The experiment emulated the live line robot deployment procedure from a grounded tower. The PLIR was controlled remotely, driving it from zero potential (ground) to the energised line segment along the insulated rod. A photograph of the HV test is given in Figure 6.14, and a short video clip can be found in Appendix B.3.

The maximum line voltage reached during testing was 100 kV. The arc visible in Figure 6.14 appears as the PLIR gets close to the line, with the PLIR's potential (voltage) reaching that of the line's. During the test, remote control of the robot was not hindered, and the PLIR was successfully driven towards and onto the power line segment. All of the robot's systems still functioned after testing. The experience gained with respect to high voltage applications has been applied to the successor of this PLIR.

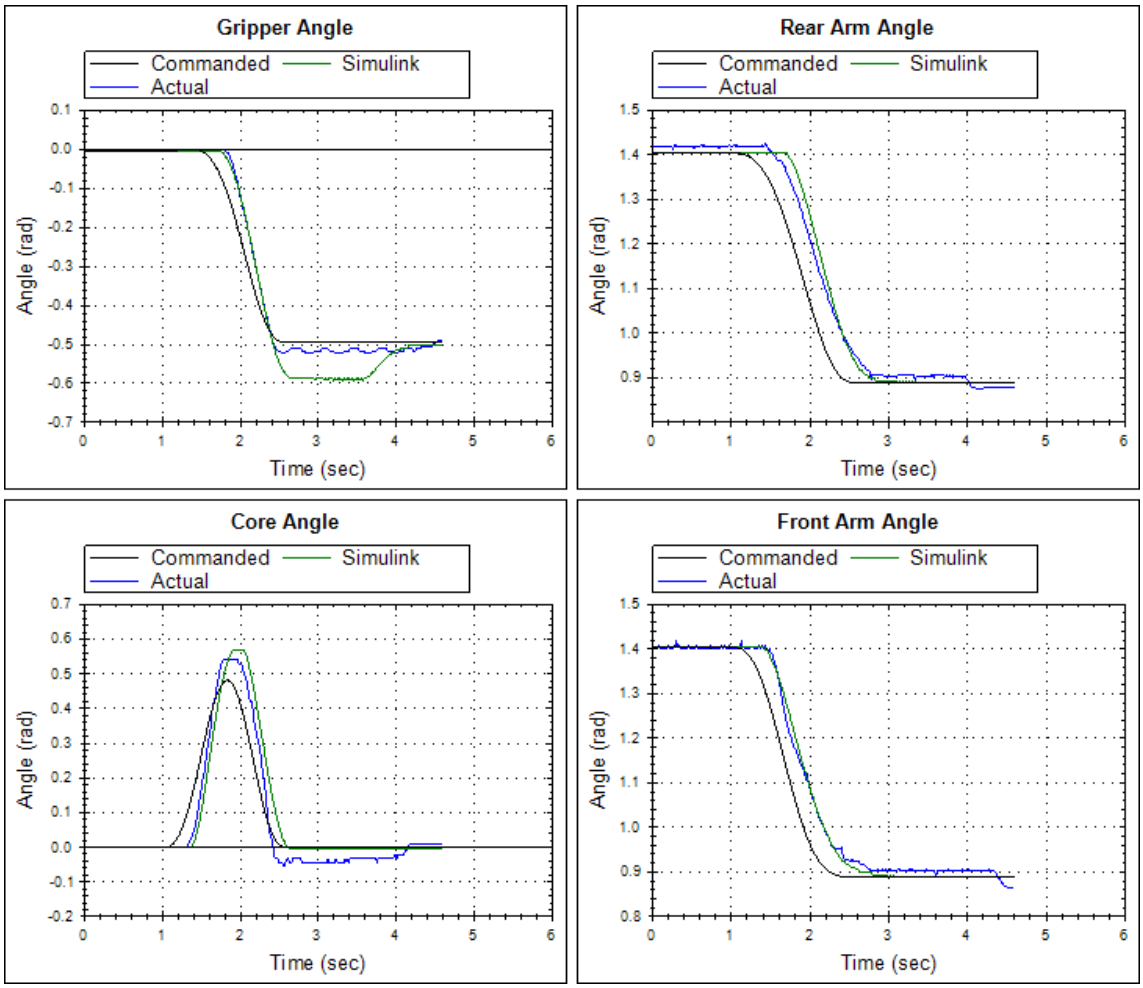

Figure 6.12: Simulink model optimised trajectory: commanded, simulated and actual robot joint angles for the jumper manoeuvre.

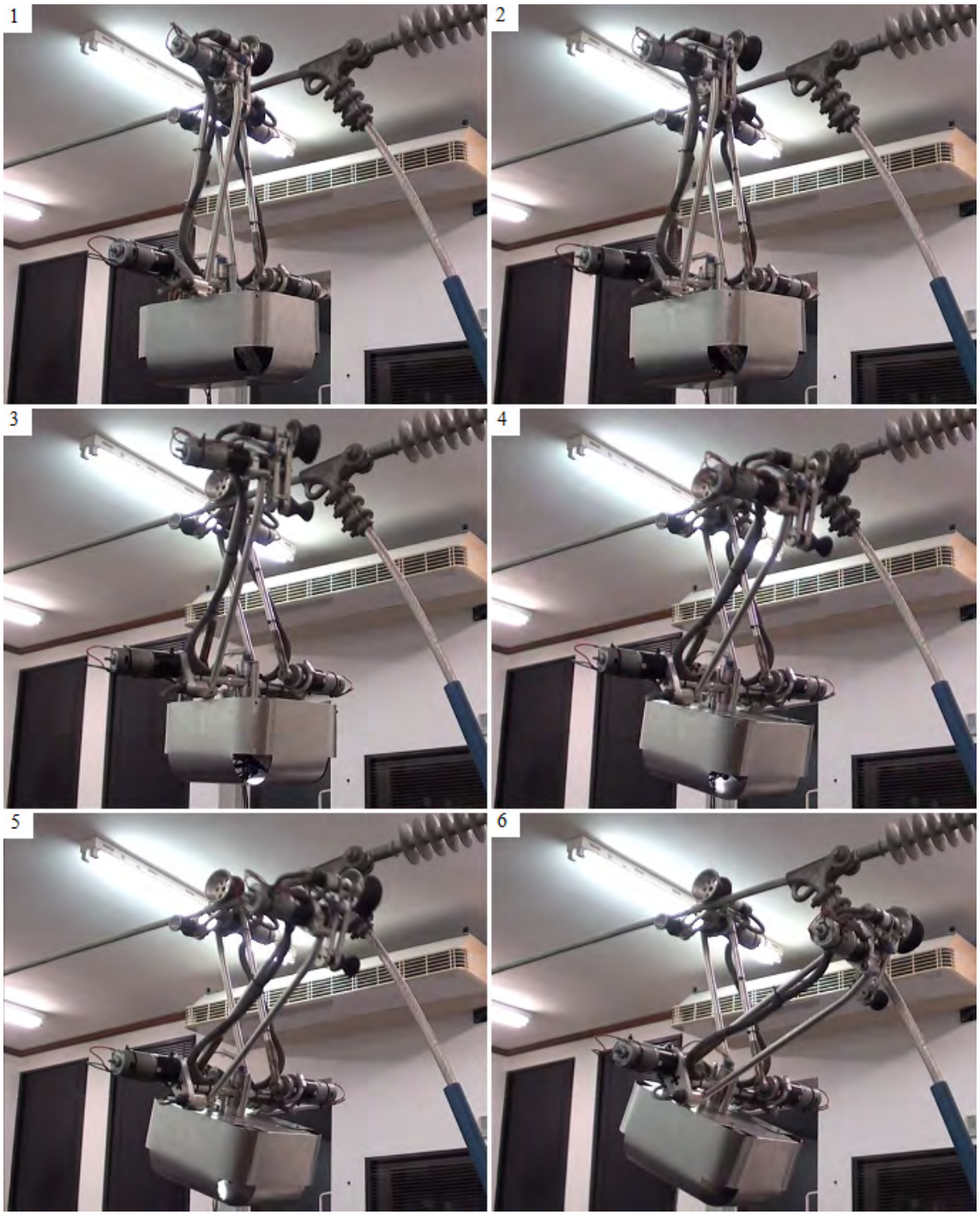

Figure 6.13: Video frames of the jumper manoeuvre test.

6.4 Conclusion

The experimental results in this chapter provided substantial verification of the methods developed in this dissertation for both obstacle avoidance and trajectory optimisation. Furthermore, the modelling of the robot and the implementation of the systems that are needed to control the robot were verified with the robot behaving as expected.

The Simulink model provided significant insight into the behaviour of the first PLIR prototype, highlighting flaws in the current robot design. The robot has many fundamentally undesirable attributes such as large friction effects and sensor noise. The successor to this robot is far superior in terms of both hardware and software, taking from the lessons learnt with the first prototype.

Figure 6.14: The PLIR approaching the energised line at 100 kV.

Chapter 7

Conclusion

This dissertation has presented the research and development of obstacle avoidance, trajectory optimisation and modelling techniques for an existing power line inspection robot, as well as their implementation and validation. The software and hardware implementations for each of the aforementioned solutions were also presented.

7.1 Evaluation of Objectives and Specifications

The obstacle avoidance solution that was developed for the PLIR comprises of three main objectives namely, obstacle description, waypoint selection and trajectory creation all represented in a 3D map description of the robot's workspace environment. The obstacle avoidance solution allowed the robot to create end effector trajectories autonomously to get around obstacles on the power line. Obstacle description was achieved using axis-aligned bounding boxes, which allows for simple representation of obstacles. A number of axis-aligned bounding boxes can be used together to describe a single obstacle.

The PLIR software has been developed to the point where it can select waypoints to get around an obstacle autonomously. The waypoints in this dissertation were selected using the Lazy Theta* algorithm (Nash *et al.*, 2010), finding the shortest path around an AABB obstacle description in the PLIR workspace map, given the initial and final waypoints. After the waypoints were selected, trajectories were created that interpolate the waypoints. It was shown that fifth order B-spline trajectories allow for smooth movement of the robot.

Optimisation of these obstacle avoidance trajectories was performed in order to reduce the demand on the robot's energy reserves. Since the robot is battery powered (with the possibility of being charged from the power line itself), preserving charge will increase the operating range of the robot. The optimisation goal presented in this dissertation was to minimise a time-energy type of cost function. The cost was based on the current drawn from the power supply by the robot components and the current required by the actuators while executing a manoeuvre. However, there was a common weakness in the optimisation approaches used in that they only minimised charge by optimising the trajectories' temporal space. In order to achieve true global charge minimisation, both the temporal and spatial components of the trajectories would have to be optimised, but this was deemed intractable in this dissertation.

Simulation results showed that the temporal B-spline components can be optimised to reduce such a cost function using a numerical simplex method. The predicted actuator cost function profiles using both models were continuous and had distinct minima around which the cost function was not overly sensitive to small variations in the input parameters.

The laboratory tests that were conducted, validated the work presented in this dissertation. The tests showed that the Simulink model of the robot was sufficiently detailed in order to predict the behaviour of the robot. This included predicting joint motion as well as the actuator current profiles. The techniques developed for trajectory interval time optimisation based on the Simulink model were verified by the accuracy of the current profile predictions. It was shown in the experiments that the PLIR was capable of executing these optimised trajectories in order to get around an obstacle successfully. Finally, high voltage testing that was conducted on the PLIR validated the engineering design robustness.

It is clear from the work presented that the specifications set out in the dissertation allowed the required objectives of the robot to be met. It has been shown that the charge requirements of the robot can be minimised to prolong potential inspection time. Furthermore, the use of an autonomous obstacle avoidance scheme makes the operation of the robot simple and energy efficient by removing direct human control.

7.2 Improvements

It is evident from the results that this first robot prototype's performance was restricted by undesirable effects. These include significant static friction that could lead to the robot being unsuccessful in some obstacle avoidance cases. Many practical problems encountered with the first prototype have been overcome by using higher quality actuators fitted with shaft encoders in the second prototype. Even though the second prototype is a different system, the fundamental principles remain the same, and the techniques developed in this dissertation have contributed greatly to the development of the second prototype.

7.3 Concluding Remark

The primary outcome of the laboratory experiments was confidence in all aspects of the engineering work. Greater insight into the behaviour of the entire robotic platform in terms of obstacle avoidance and trajectory optimisation was achieved, and the work presented closes the loop around the engineering "triangle" of theory, simulation and experimentation. The problems of energy efficiency and operating complexity were addressed with positive results.

Techniques developed in this dissertation are not only applicable to this power line inspection robot, but can be used in many different robotic applications, particularly where serial manipulators are used. Obstacle avoidance, trajectory optimisation and modelling techniques are at the forefront of development in the field of power line inspection robotics, and therefore the work presented in this dissertation is a useful contribution to the state of the art.

References

Bailó, W. P., Cardiel, E. B., Campos, I. J. and Paz, A. R. (2010). "Mechanical Energy Optimization in Trajectory Planning for Six DOF Robot Manipulators Based on Eighth-degree Polynomial Functions and a Genetic Algorithm". *International Conference on Electrical Engineering Computing Science and Automatic Control*. pp. 446 – 451.

Barton, R. R. and Ivey, J. S. J. (1996). "Nelder-Mead Simplex Modifications for Simulation Optimization". *Management Science*. Institute for Operations Research and the Management Sciences, pp. 954 – 973.

Biagiotti, L. and Melchiorri, C. (2008). *Trajectory Planning for Automatic Machines and Robots*. Springer. ISBN 978-3-540-85628-3.

Bohlin, R. and Kavraki, L. (2000). "Path Planning Using Lazy PRM". *IEEE International Conference on Robotics and Automation*. pp. 521 – 528.

Buss, S. (2004). "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods". Technical report, Department of Mathematics, University of California, San Diego.

Cai, L., Liang, Z., Hou, Z. and Tan, M. (2008). "Fuzzy Control of the Inspection Robot for Obstacle-Negotiation". *IEEE International Conference on Networking, Sensing and Control*. pp. 117 –122.

Cao, B., Doods, G. and Irwin, G. (1994). "Time-optimal and Smooth Constrained Path Planning for Robot Manipulators". *IEEE International Conference on Robotics and Automation*. pp. 1853 – 1858.

Champion, J. (2005). "ZedGraph Class for C#". Accessed: 13/02/2012.
URL: *http://zedgraph.sourceforge.net/index.html*

De Santiago-Castillo, J. A. (2009). "DotNumerics V1.1 library for C#". Accessed: 23/11/2011.
URL: *http://www.dotnumerics.com/*

Debenest, P. and Guarnieri, M. (2010). "Expliner - from Prototype Towards a Practical Robot for Inspection of High-voltage Lines". *International Conference on Applied Robotics for the Power Industry*. pp. 1 – 6.

Ericson, C. (2005). *Real-time Collision Detection*. Morgan Kaufmann series in interactive 3D technology, Elsevier. ISBN 978-1-55860-732-3.

Gasparetto, A. and Zanotto, V. (2007). "A New Method for Smooth Trajectory Planning of Robot Manipulators". *Mechanism and machine theory*, Elsevier. pp. 455 – 471.

Hirakawa, A. and Kawamura, A. (1997). "Trajectory Planning of Redundant Manipulators for Minimum Energy Consumption Without Matrix Inversion". *IEEE International Conference on Robotics and Automation*. pp. 2415 – 2420.

Hrabar, S. (2008). "3D Path Planning and Stereo-based Obstacle Avoidance for Rotorcraft UAVs". *IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 807 – 814.

Jukic, T. and Peric, N. (2001). "Model Based Backlash Compensation". *The American Control Conference*. pp. 775 – 780.

Li, T., Shuangfei, F., Lijin, F. and Hongguang, W. (2004). "Obstacle-navigation Strategy of a Wire-suspend Robot for Power Transmission Lines". *IEEE International Conference on Robotics and Biomimetics*. pp. 82 – 87.

Lin, C., Kuo, L. and Chuang, J. (2005). "Potential-based Path Planning for Robot Manipulators". *Journal of Robotic Systems*. John Wiley & Sons, Inc., pp. 313 – 322.

Lorimer, T. (2011). *The Design and Construction of a Robotic Platform for Power Line Inspection*. Master's thesis, Electrical, Electronic and Computer Engineering, University of KwaZulu-Natal.

Martin, B. and Bobrow, J. (1997). "Minimum Effort Motions for Open Chain Manipulators with Task-dependent End-effector Constraints". *IEEE International Conference on Robotics And Automation*. pp. 2044 – 2049.

Matlab (2012). "MATLAB Builder NE for Microsoft .NET Framework". Accessed: 26/02/2012.
URL: *http://www.mathworks.com/products/netbuilder/*

Nash, A., Koenig, S. and Tovey, C. (2010). "Lazy Theta *: Any-Angle Path Planning and Path Length Analysis in 3D". *AAAI Conference on Artificial Intelligence*. pp. 147 – 154.

Nelder, J. A. and Mead, R. (1965). "A Simplex Method for Function Minimization". *The Computer Journal*. Oxford University Press, pp. 308 – 313.

Piazzi, A. and Visioli, A. (2000). "Global Minimum-jerk Trajectory Planning of Robot Manipulators". *IEEE Transactions on Industrial Electronics*. pp. 140 – 149.

Powerline Patrol Services (2010). "Traditional Patrol Methods". Accessed: 19/02/2012.
URL: *http://www.powerlinepatrol.com/patrol-services/traditional-patrol-methods/*

Retief, J. (No Date). "Eskom Overhead Line Vibration System, 1994 - 1999 Audit Reports".
URL: *[CD-ROM]*

Saramago, S. and Steffen, V. J. (1998). "Optimization of the Trajectory Planning of Robot Manipulators Taking Into Account the Dynamics of the System". *Mechanism and Machine Theory*, Pergamon. pp. 883 – 894.

Schouwenaars, T., De Moor, B., Feron, E. and How, J. (2001). "Mixed Integer Programming for Multi-vehicle Path Planning". *European Control Conference*. pp. 2603 – 2608.

Seshadri, C. and Ghosh, A. (1993). "Optimum Path Planning for Robot Manipulators Amid Static and Dynamic Obstacles". *IEEE Transactions on Systems, Man and Cybernetics*, IEEE Institute of Electrical and Electronics Engineers. pp. 576 – 584.

Shiller, Z. (1994). "Time-energy Optimal Control of Articulated Systems with Geometric Path Constraints". *IEEE International Conference on Robotics and Automation*. pp. 2680 – 2685.

Spong, M. W., Hutchinson, S. and Vidyasagar, M. (2006). *Robot Modeling and Control*. John Wiley & Sons, Inc. ISBN 978-0-471-64990-8.

Sun, C., Wang, H. and Zhao, M. (2007). "Vibration Analysis of Obstacle-avoidance for EHV Power Transmission Lines Inspection Robot". *IEEE Industrial Electronics Society Conference*. pp. 2824 – 2828.

Wang, L., Fang, L., Wang, H. and Zhao, M. (2006). "Development and Control of an Autonomously Obstacle-Navigation Inspection Robot for Extra-High Voltage Power Transmission Lines". *International Joint Conference, SICE - ICASE*. pp. 5400 – 5405.

Wang, Y., Han, L., Li, M., Wang, Q., Zhou, J. and Cartmel, M. (2004). "A Real-time Path Planning Approach without the Computation of C-space Obstacles". *Robotica*, Cambridge University Press New York. pp. 173 – 187.

Wu, G., Xu, X., Xiao, H., Dai, J., Xiao, X., Huang, Z. and Ruan, L. (2008). "A Novel Self-navigated Inspection Robot along High-Voltage Power Transmission Line and Its Application". *International Conference on Intelligent Robotics and Applications*. ISBN 978-3-540-88516-0, pp. 1145 – 1154.

Xiao, X., Wu, G., Du, E. and Shi, T. (2005). "Dynamic Simulation and Experimental Study of Inspection Robot for High-voltage Transmission-line". *Journal of Central South University of Technology*, Central South University, co-published with Springer. pp. 726 – 731. 10.1007/s11771-005-0077-y.

Xiao, X., Wu, G., Xiao, H. and Dai, J. (2008). "An Inspection Robot for High Voltage Power Transmission Line and Its Dynamics Study". *Service Robot Applications*, InTech.

Yang, S. and Meng, M. (2001). "Neural Network Approaches to Dynamic Collision-free Trajectory Generation". *IEEE Transactions on Systems, Man, and Cybernetics*, IEEE Institute of Electrical and Electronics Engineers. pp. 302 – 318.

Zhang, W. and Sobh, T. M. (2003). "Obstacle Avoidance for Manipulators". *Systems Analysis Modelling Simulation*, Gordon & Breach / Harwood Academic Publishing. pp. 67 – 74.

Zhou, F., Wang, J., Li, Y., Wang, J. and Xiao, H. (2005). "Control of an Inspection Robot for 110kV Power Transmission Lines Based on Expert System Design Methods". *IEEE Conference on Control Applications*. pp. 1563 – 1568.

Appendix A

PLIR Parameters

A.1 Simulink Model Parameters

The system parameters used in the Simulink model of a single joint actuator for the gripper and arm joints are listed in Table A.1. The parameters used to model the core joint are given in Table A.2. The parameters differ between the core joint and other joints because the actuators used are different.

Table A.1: Simulink model parameters for a single gripper or arm joint.

Parameter	Value	Units
Motor constant (k_ϕ)	0.019	N.m.A^{-1}
Armature inductance (L)	6.84×10^{-4}	H
Armature resistance (R)	1.0	Ω
Armature inertia (J)	2.0×10^{-5}	kg.m^2
Armature viscous friction (B)	5.71×10^{-5}	N.m.s.rad^{-1}
Coulomb friction torque (τ_{f_C})	0.0171	N.m
Static friction torque (τ_{f_s})	0.04	N.m
Gearbox ratio (r)	488:1	(none)
Gearbox backlash	0.033	rad

Table A.2: Simulink model parameters for the core joint.

Parameter	Value	Units
Motor constant (k_ϕ)	0.007	N.m.A^{-1}
Armature inductance (L)	7.7×10^{-4}	H
Armature resistance (R)	0.77	Ω
Armature inertia (J)	4.4×10^{-6}	kg.m^2
Armature viscous friction (B)	2.2×10^{-6}	N.m.s.rad^{-1}
Coulomb friction torque (τ_{f_C})	0.002	N.m
Static friction torque (τ_{f_s})	0.003	N.m
Gearbox ratio (r)	516:1	(none)
Gearbox backlash	0.033	rad

A.2 PID Controller Parameters

The PID controller parameters for the arm and gripper actuators are given in Table A.3, and the parameters for the core controller are given in Table A.4.

Table A.3: PID parameters for arm and gripper joints.

Parameter	Value
Gain	364.28
T_i	0.56 s
T_d	0.13 s
α	0.1

Table A.4: PID parameters for the core joint.

Parameter	Value
Gain	28.41
T_i	0.19 s
T_d	0.47 s
α	0.034

Appendix B

Software

B.1 PLIR Trajectory Solver software

Please find attached the CD containing the PLIR Trajectory Solver code. It is best viewed using Microsoft Visual Studio 2010 by opening the main project file.

B.2 Simulink PLIR Model

Please find attached the CD containing the Simulink PLIR model. It can be viewed by opening the model in a MATLAB environment.

B.3 Videos

Please find attached the CD containing the video footage of the obstacle avoidance tests presented in Chapter 6. The videos for Section 6.1 are named “Suspension Clamp Manoeuvre - Simulink.wmv” and “Suspension Clamp Manoeuvre - ID.wmv”, and the video for Section 6.2 is named “Jumper Manoeuvre.wmv”. The HV test video is named “PLIR HV Test.wmv”.

Appendix C

Hardware

C.1 Ground Station Computer Specifications

Table C.1: Ground station computer specifications.

Processor	Intel® Core™ 2 Duo T8300 2.4 GHz
FSB	800 MHz
RAM	4 GB DDR 2 (667 MHz)
GPU	ATI Radeon 3650 with 512 MB RAM
Wi-Fi	Intel® Wireless Wi-Fi Link 4965AGN 802.11n
OS	Windows 7 SP 1 32-Bit

C.2 On-board Computer Specifications

Table C.2: fitPC-2i (robot on-board PC) specifications.

Processor	Intel® Atom™ Z550 2 GHz
FSB	533 MHz
RAM	2 GB DDR2 (533 MHz)
GPU	Intel® GMA500
Wi-Fi	RaLink RT3070 802.11n
OS	Windows XP SP 3 32-Bit
Power	
Supply	12 V DC (8 V to 15 V tolerant)
Full CPU load	8W
Low CPU load	6W
Standby	1W

Figure C.1: The fitPC-2i.