Controller Development and Implementation for Path Planning and Following in an Autonomous Urban Vehicle

Matthew J. Barton

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Engineering (Mechatronics)

Australian Centre for Field Robotics (ACFR)
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney

November 2001

Declaration

I hereby declare that:

- I conducted the literature review for this thesis.
- Unless otherwise stated, I researched, developed and implemented the continuous curvature path planning techniques (both theoretically and in software) presented in Chapter 2 of this document.
- I researched the shortest path algorithms, developed the path replanning (obstacle avoidance) technique and implemented and performed the accompanying simulations myself.
- I developed and implemented the path tracking (steering) control system and simulator.
- I developed the steering model used in the controller simulations.
- I implemented and tested with the assistance of José Guivant (and his software) the steering controller and the MATLAB/Hyperkernel shared memory interface on the ute.
- I converted my path planning/following system from simulation to real-time operation on the ute.
- I wrote the GUI used for operating the path planning and tracking system (for both path entry and following).
- I assisted Ken Lee with the theoretical design of the intermodule communications system.
- The obstacle avoidance technique implemented on the ute was jointly developed by Ken Lee and myself.

- With the assistance of Ken Lee, I developed and implemented the complete path planning/tracking and obstacle avoidance system on the ute.
- The complete system, as described in Chapter 5, has been successfully demonstrated to my supervisor, Associate Professor Eduardo Nebot.
- All testing was performed by myself, except for the obstacle avoidance field trials which were carried out with the help of Ken Lee.
- All results derived from both simulations and field tests are presented without alteration.
- The analysis of the data and the conclusions reached are my own.

Matthew Barton

November 10, 2001

Abstract

Autonomous Land Vehicles (ALVs), due to their considerable potential applications in areas such as mining and defence, are currently the focus of intense research at robotics institutes worldwide. Control systems that provide reliable navigation, often in complex or previously unknown environments, is a core requirement of any ALV implementation. Three key aspects for the provision of such autonomous systems are: 1) path planning, 2) obstacle avoidance, and 3) path following.

The work presented in this thesis, under the general umbrella of the ACFR's own ALV project, the 'High Speed Vehicle Project', addresses these three mobile robot competencies in the context of an ALV based system. As such, it develops both the theoretical concepts and the practical components to realise an initial, fully functional implementation of such a system. This system, which is implemented on the ACFR's (ute) test vehicle, allows the user to enter a trajectory and follow it, while avoiding any detected obstacles along the path.

Acknowledgements

The work presented in this thesis would not have been possible without the assistance of countless individuals who gave up their time (often when they would rather be doing something else), for no reward other than the transfer of knowledge. So, to all these people, I extend my eternal gratitude. However, despite being a dangerous practice, I would like to single out a few, select, individuals for special recognition.

Firstly, I would like to thank my supervisor, Associate Professor Eduardo Nebot, for providing project direction, encouragement and advice over the past 9 months - it has been very much appreciated.

To the HSV project Ph.D. students, José 'this guy' Guivant, Trevor 'Rodrigues' Fitzgibbons and the other members of the 'Boca Juniors Back Four', I would like to express my gratitude for their good humour, technical expertise and advice, often at times of crisis.

Thanks is also owed to the other three members of 'Team HSV 2001', Tom, Mark and Ken, a more decent group of blokes you are never likely to meet. I would like to thank them for adding to the thesis experience (particularly Ken, for forcing me to spend two nights, hacking code in the 'sauna-like' cabin of the ute in the middle of St. Andrew's Oval).

Providing a university student (who by fourth year is in real danger of insolvency) with $2500 will always ensure you a mention on an acknowledgements page. Therefore, I would like to thank CMTE for their scholarship.

Speaking of cash for comment, I would also like to thank Richard Grover and my other friends who over the past year have always tried to make me laugh and think positively, often at times when I'd rather cry.

Finally, my biggest thank you goes to my family, for their unwavering love, support, guidance, education, life skills etc. . . . provided free of charge over the past 21 years. Words can never describe how grateful I am.

To my family, without you I am nothing

Lenny: Hey! Look, Homer's got one of those new robot cars!
[the car crashes]
Carl: Yeah! One of those American robot cars!
- The Simpsons, 1994

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Rise of the Robot

Over the last two decades, the world has witnessed a phenomenal expansion (in terms of capabilities, applications and research activity) in the field of robotics. Ever increasing computer speed, diverse (and accurate) sensing and actuation options (all at a decreasing cost), complimented by a concurrent development in the theoretical framework[1] required to transform collections of electromechanical components into functional robotic systems, represent three of the key driving forces responsible for this expansion.

As a result, robots (of one form or another) have been extensively employed for performing monotonous and/or dangerous tasks - such as space exploration and automated mass production - for nearly thirty years. However, it is fair to say that a paradigm shift in robot usage is presently underway. Across the globe, the potential of autonomous systems is being increasingly recognised and applied in many previously unexplored disciplines. Recent, novel, yet sophisticated applications include the following:

- **Gene Sequencers** - are used for rapidly mapping an organism's (for example, that of a human) genome. The arrival of DNA sequencers in the late 1990's pushed the anticipated completion date of the Human Genome Project (HGP) forward by approximately 5 years (Boyce and Coghlan 2000).

[1]Prominent examples include: modern control theory, the Kalman filter and image processing algorithms and techniques.

- **Automated Photonic Work Cell** - this world-first project is currently under way at the Optical Fibre Technology Centre (OFTC) in the Australian Technology Park (ATP). It's primary aim is to automate the manufacture of photonic components, such as thermally expanded core (TEC) optical fibre (Robinson 2001).

These examples are only indicative of the research and development projects currently in progress around the world. Such widespread interest and diverse applications have created numerous classes and sub-classes of robots[2], catapulting them to a point where such robotic systems are increasingly infiltrating daily life in the early part of this new century.

1.2 Autonomous Land Vehicles

One robot category that is presently the subject of considerable research and commercial interest is that of Autonomous Land Vehicles (ALVs). As with other expanding robot genres, ALVs offer numerous potential applications, particularly in dangerous and/or impractical environments, or where human operator costs are significant[3], some notable examples including:

- Space Exploration
- Stevedoring (that is, the loading and unloading of cargo ships)
- Military Reconnaissance
- Surveying
- Mining and Excavation

In response to interest in these and similar applications, most of the world's major robotic research institutes, including Carnegie Mellon University (CMU), Pittsburgh, the Massachusetts Institute of Technology (MIT), Boston, and the Australian Centre

[2]Note that this thesis focusses upon the development of a path planning and following system for a robot belonging to the car-like mobile robot subclass.

[3]It is worth noting that haulage accounts for approximately 50-60% of mining companies' costs (ACFR 2001).

for Field Robotics (ACFR) in Sydney have established projects to develop ALVs or related technologies. However, this research is quite diverse, as ALVs are a non-homogeneous class of robots, ranging from small interplanetary vehicles to excavators or bulldozers weighing many tonnes. Each of these sub-classes possess unique (and often limiting) motion characteristics, that must be understood and accounted for in any autonomous implementation.

One class of ALV, which the research presented in this thesis focusses upon, is that of car-like robots. For our purposes, a car-like robot was defined as a four-wheeled vehicle (all systems here were implemented on a late 1990's Holden utility), whose two rear wheels are fixed and whose two front wheels are capable of being steered[4] (Scheuer and Fraichard 1996*b*). Such a wheel arrangement and steering mechanism imposes significant motion restrictions upon a car-like robot, which are termed non-holonomic constraints whose intrinsic features and implications are discussed in Chapter 2 (Fraichard and Ahuactzin 2001).

The usage of the car-like vehicle configuration is very widespread, so the development of the necessary specialised automation techniques (such as control, path planning and localisation) represent a logical and necessary step towards ALVs achieving mainstream acceptance and deployment. There are innumerable projects currently investigating car-like robot systems, some relevant international examples being briefly outlined below:

- Researchers at CMU's Robotics Institute are involved in a number of ALV projects that tackle various aspects of the high speed autonomous navigation problem. Their FastNav and RANGER (Real Time Autonomous Navigator with a Geometric Engine) programs have attempted, under contract to the US Army, to investigate and develop a complete sensing, control and software system for the intelligent control of a car-like robot in 'off-road' environments (Kelly 1994*b*, Kelly 1995, Kelly 1994*a*).

  In addition, the RALPH (Rapidly Adapting Lateral Position Handler) project has developed a system, using machine vision techniques[5], that extract both road curvature and vehicle lane position information in real-time[6] (Pomerleau

[4]A car-like robot can, in fact, possess an arbitrary number of fixed or steerable wheels.

[5]Images are captured (via a frame-grabber) from a video camera mounted next to the rear-view mirror.

[6]Research similar to this was carried out in 2001 by Tom Mactier and is detailed in his undergraduate thesis entitled 'Vision Based Road Following'.

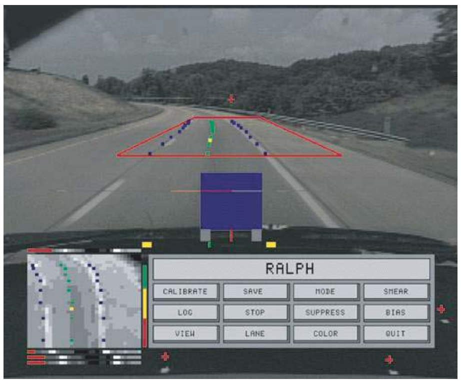


Figure 1.1: Example image from the RALPH system

1995). A screenshot of the RALPH system in action is shown Figure 1.1. It is hoped that this system will prove to be a valuable driver 'assistant' and/or motor accident prevention device, able to provide a warning to the driver, or even commandeer the vehicle's steering and take evasive action directly, if the vehicle has strayed from its lane.

Continuous Curvature Path Planning (CCPP) techniques[7] for car-like robots are also currently under development within the Robotics Institute at CMU(Nagy and Kelly 2001). Such techniques are a major focus of the research presented in this thesis.

- The Center for Intelligent Machines and Robotics (CIMAR), at the University of Florida, is another example of a research centre contracted by the military to develop ALVs. A number of vehicles have been automated over the last decade, including an autonomous survey vehicle for detecting buried, unexploded ordnance and a bulldozer, shown in Figure 1.2, capable of clearing a 50 metre square area of mines and other obstructions to provide a safe landing site for the US Marines and their supplies (Wit 2001).

  Like the Robotics Institute at CMU, CIMAR's research is very broad, concerning itself with all aspects (such as environmental perception, localisation, planning and control) of ground vehicle automation. An example of their recent work is the development of a path-tracking technique for a car-like robot, based upon 'screw theory' and implemented using two fuzzy reference model-learning controllers. Results of recent field trials have shown negligible path-tracking

[7]These are methods for path planning that respect the nonholonomic motion constraints of a car-like vehicle (see Chapter 2).

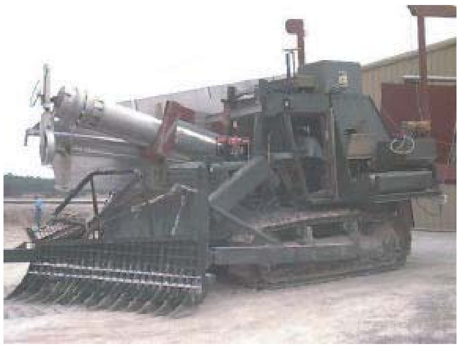


Figure 1.2: CIMAR's autonomous D7G bulldozer

errors[8] for speeds of up to 6 mps (21.6 km/h) being reported(Wit 2001).

- The INRIA[9], in Montbonnot, France has, over the last 10 years, been heavily involved with solving the car-like robot navigation problem, with particular interest in CCPP. Researchers such as Thierry Fraichard and Alexis Scheuer, have been formulating the mathematics and numerical techniques for specifying, generating and optimising continuous curvature paths (Scheuer and Fraichard 1996*b*, Scheuer and Fraichard 1997, Fraichard and Ahuactzin 2001). INRIA's path-planning system, using the 'Ariadne's clew algorithm' (ACA), is capable of operating, very successfully, in a cluttered (that is, obstacle ridden) environment (Mazer, Ahuactzin and Bessiere 1998). Figure 1.3 displays the results of one of the CCPP implementations developed at INRIA. However, at present, this has only ever been implemented in the guise of an off-line planner (i.e. once the path is generated, it is not updated based upon additional environmental information)[10].

A more detailed and rigorous discussion of the technical content and merits of these and other relevant research projects can be found in the critical literature reviews, located in Chapters 2, 3 and 4.

[8]Position errors of less than 10 cm and heading errors of 0.5-1.0 deg.

[9]Inst. Nat. de Recherche en Informatique et en Automatique

[10]No clear indication has been provided by INRIA, as to why a real-time path planner based upon this, clearly promising, method remains unimplemented. One possible explanation, gleaned from recent papers, is that lengthy computation times associated with this and similar planning techniques render it impractical at present (Mazer et al. 1998) (Fraichard, Scheuer and Desvigne 1999) (Fraichard and Ahuactzin 2001).

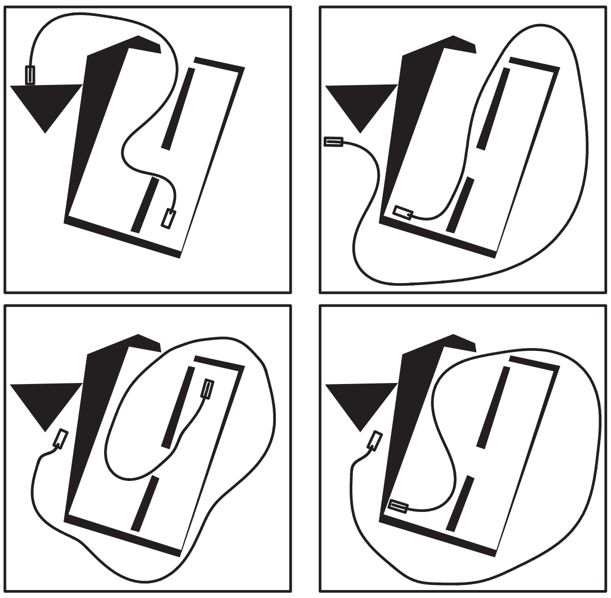


Figure 1.3: Results of CCPP techniques developed at INRIA

1.3 The High Speed Vehicle Project

Currently the ACFR, at The University of Sydney, operates its own ALV project, similar, in both research carried out and desired outcomes, to those conducted at overseas robotics institutes. This endeavour was initiated in the mid-1990s and is known as the High Speed Vehicle (HSV) project[11]. It has a strong focus on the automation of car-like vehicles, and forms the general umbrella under which the research and development outlined in this thesis was performed.

1.3.1 Background and Areas of Research

The over-arching aim of the HSV project is to develop technologies for the automation of land vehicles operating, at 'high speed'[12] in a variety of 'real' environments. Applications currently under investigation include:

- Navigation in urban environs
- Semi and full automation of mining[13], military and agricultural vehicles

[11]The HSV project is lead by Associate Professor Eduardo Nebot.

[12]Taken as meaning speeds of up to 90 km/h.

[13]The HSV project is carried out in collaboration with the Cooperative Research Centre for Mining Technology and Equipment (CMTE). The CMTE sponsored (via their undergraduate internship program) the four HSV project undergraduate theses in 2001.

Developing the robust and reliable control systems required by an autonomous vehicle for high speed operation/navigation (in often unstructured, real-world environments) is a challenge presenting many potential bottlenecks. Techniques and algorithms implemented on low speed (often indoor) mobile robots working in static, structured environments, are often completely inappropriate for a high speed autonomous vehicle. The 'high speed' element increases system complexity, requiring the vehicle to look further ahead and necessitating the consideration of a multitude of future motion outcomes - all of which needs to be implemented within a software architecture possessing multiple layers of redundancy and 'safe' failure modes.

To make such systems a reality, the HSV project researches all aspects of automating vehicle navigation. This process can be broken down into four steps: 1) perceiving and modelling the environment, 2) localising the vehicle within it's environment, 3) planning and deciding the vehicle's desired motion and 4) executing the vehicle's desired motion (Wit 2001). Some examples of current research in these areas are outlined below:

- Development of **high integrity navigation systems** - high speed ALV implementations not only require accurate control and environmental sensing but also need to be 'failure safe' (Sukkarieh, Nebot and Durrant Whyte 1999), (for example, they need to be able to recognise and ignore poor GPS estimates).
- Performance of **Simultaneous Localisation and Mapping** (SLAM), using environmental information obtained from a variety of sensor sources, including, radar, laser and vision. This is a 'learning' technique whereby a robot enters an environment, of which it has no prior knowledge, and proceeds to both map and localise itself, within this environment using detectable landmarks (Guivant and Nebot 2001).
- **Sensor fusion** - AGVs typically utilise a suite of sensors for providing information such as vehicle position. The quality of information each sensor provides can be characterised statistically, and must be fused with data from other sensors using a stochastic technique (such as a Kalman filter) (Guivant, Nebot and Baiker 2000).
- **Vehicle modelling and control** - vehicle motion (particularly at high speed) must be accurately characterised using a complete kinematic model. This model

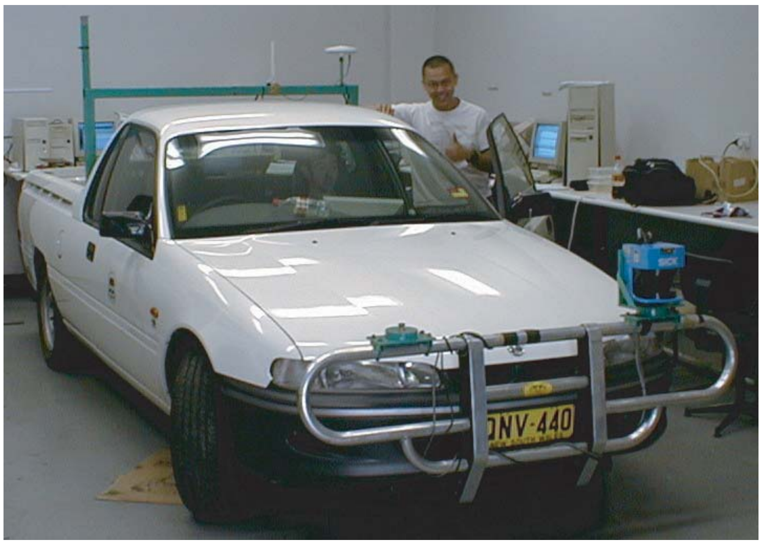


Figure 1.4: HSV Project's Test Vehicle: The 'Ute'

can then form part of a model-based predictive controller, used to precisely control the robot's motion by 'projecting its behaviour into the future'.

1.3.2 The Ute: Sensing and Actuation

Research and development work is implemented, tested, debugged, and evaluated on the HSV project's test vehicle, a mid-1990's Holden utility ('ute'), retrofitted with the computing, sensing and actuation hardware necessary for complete automation[14] (see Figure 1.4). A PC (Pentium II, 400MHz, 64MB RAM) running the MS Windows NT (with the Hyperkernel real-time extension) and QNX (real-time) operating systems is located in the ute's tray, along with data acquisition equipment (such as an A/D converter) for interfacing between the sensors and actuators[15]. Inside the cabin, on the passenger's side, a flat screen monitor has been mounted, to facilitate 'on-the-fly' monitoring of system performance during testing.

A substantial suite of sensors is currently available for use on the 'ute', these provideing several potential localisation and environmental sensing options and configurations. The sensors implemented are outlined as follows:

- Inertial Navigation Unit (INU) - Provides linear and rotational velocity data, also outputs unit pitch and roll angles.

[14]The 'ute' also serves as a test platform for other projects undertaken within the ACFR.

[15]All the computing hardware is protected by a waterproof cabinet, into which all the sensor, actuator and power wiring feeds.

- Global Positioning System (GPS) - The vehicle's position (to a nominal 1m error) and velocity on the surface of the Earth can be obtained from radio signals broadcast from global positioning satellites orbiting the Earth[16]. Increased accuracy estimates (down to a nominal 2cm error) are available if a local base station, of known position, is set up[17]. Such a GPS implementation is known as differential GPS (DGPS).

- Linear Variable Displacement Transducer (LDVT) - The role of the LVDT is to determine the ute's steering angle. This is interfaced to the steering rack, where its output voltage provides a linear relationship to the steering angle (Mok 1998).

- Optical Encoder - This is interfaced to the rear left wheel of the ute, and returns the wheel position[18]. When combined with a timestamp, this can be used to provide a dead-reckoning estimate of the vehicle's velocity.

- Potentiometers - These variable resistors return the position of the linear actuators used to control the vehicle's brake and throttle.

- Video Camera (with Frame Grabber) - This CCD camera allows 576x768 resolution images to be captured at a rate of up to 30 frames per second for input to a variety of image processing techniques.

- Bearing and Ranging Lasers - Two scanning lasers, mounted on the bull-bar (see Figure 1.5), typically in the horizontal plane, use time-of-flight measurements to determine the range of local obstacles in a 180 deg sweep[19].

- Compass - An alternative source of vehicle heading information, measured relative to magnetic north.

Three actuation systems (for steering, throttle and brake) are also in place, allowing the ute to operate under complete autonomous control. A brief description of the configuration of these three systems is now given (Mok 1998, Matheson 2000):

[16]This requires the reception of four satellite signals (or 'fixes') simultaneously.

[17]A base station improves accuracy by calculating the local error in the GPS estimates, which are then broadcast to the mobile GPS receiver for correction of vehicle position.

[18]The encoder has a resolution of 0.005 deg.

[19]The lasers' maximum range is 80m with a scanning resolution of 0.5 deg, i.e. 361 pulses/sweep.

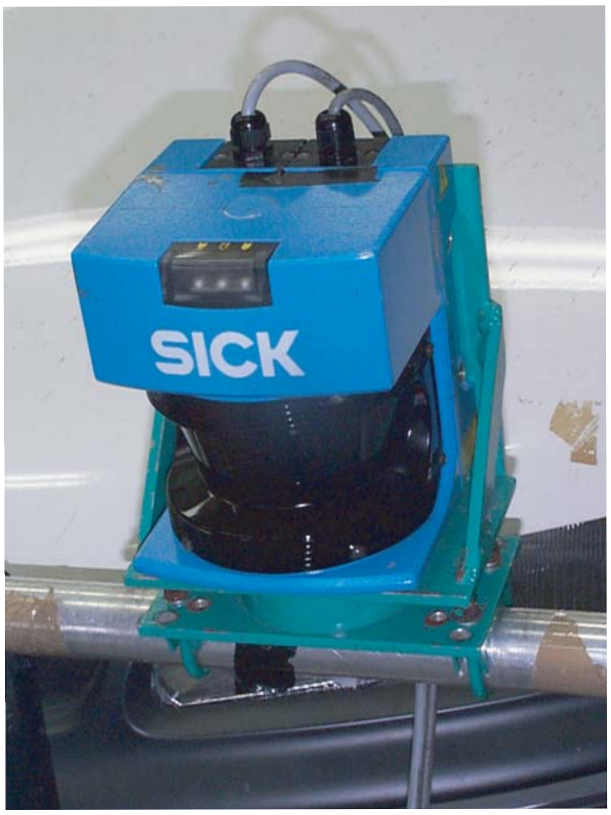


Figure 1.5: SICK scanning laser

- Steering - a DC motor, mounted inside the driver's foot-well, turns the steering column through a worm reduction gearbox and torque limiting clutch arrangement which is shown in Figure 1.6.
- Throttle - a linear actuator controls the displacement of the carburettor butterfly valve (which determines engine power output), via a guide cable.
- Brake - a second linear actuator (mounted under the driver's seat) controls the position of the brake pedal directly. The physical connection between the pedal and the actuator is via two cables, threaded through a bracket in the front of the foot-well.

1.3.3 Previous Undergraduate Theses

Each year, for the last 5 years, approximately 3-5 final year thesis students have been involved in the HSV project. Their research and development undertakings have covered many aspects relevant to the general high speed, car-like robot navigation problem.

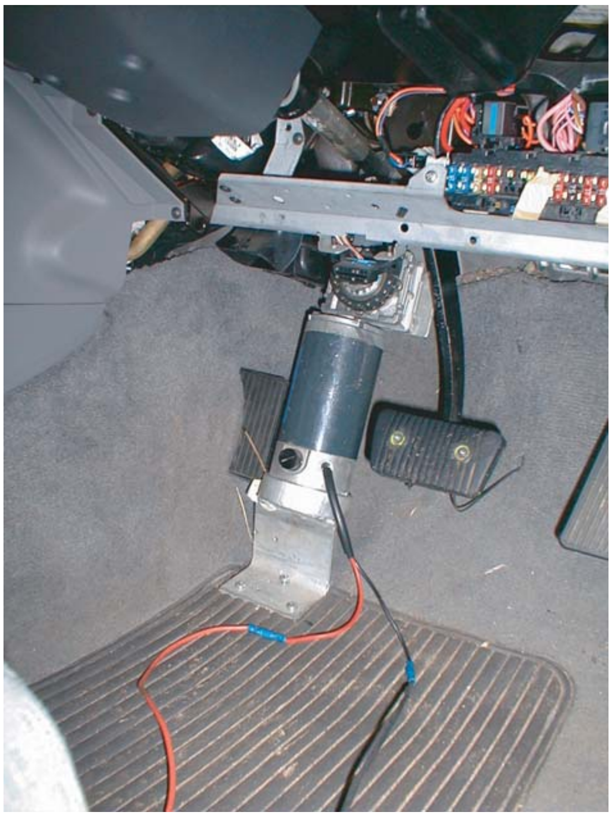


Figure 1.6: Steering motor arrangement

The theses of Mok (1998) and Lim (1998), undertaken during the formative stages of the HSV project, largely focussed upon hardware (both mechanical and electronic), low-level control and ensuring that the basic automation aspects of the ute were operational. More recent theses, such as Lin (1996), Matheson (2000) and Lai (2000) have investigated diverse topics including path planning, control and reactive navigation (in various guises, and with varying degrees of success).

One obvious concern with the most recent HSV thesis projects, is the retracing of ‘old ground’. However, the legacy of some undergraduate thesis students[20] is often minimal, meaning that previously undertaken research has to be redone, rather than expanded upon the following year[21]. This shortcoming has been well noted, and successful implementation, in a user friendly form, made a key objective (see section 1.4.1) of this thesis - with the hope of ensuring the continuation of this facet of research and development within the HSV project in 2002.

[20]Particularly in terms of readily reusable software.

[21]For example, quite similar path following techniques have been investigated in each of the last three years. However, none of this software remains operational on the ute today.

1.3.4 Undergraduate Theses in 2001

The four undergraduate theses in 2001 are quite diverse in scope and character. Two have focussed upon path planning, control (Matthew Barton) and reactive navigation (Ken Lee) - ‘traditional’ areas of research within the HSV project. By contrast, the other two theses have diverged into the areas of vision-based road following (Tom Mactier) and 3D synthetic mapping (Mark Nolde).

A brief outline of the other theses within the HSV project in 2001 is now given. Lee’s thesis ‘Reactive Navigation for an Autonomous Outdoor Vehicle’, develops a laser-based obstacle detection and characterisation system. This provides the obstacle inputs to the path replanner (obstacle avoidance system) developed in Chapter 3 for providing reactive navigation. The other two theses are somewhat unrelated to the work presented here. Mactier’s work entitled ‘Urban Road Following Using a Single CCD Camera’ used vision techniques for extracting road information (curvature and position) from an image, so as to provide steering controller inputs. Finally, Nolde’s thesis ‘3D Outdoor Synthetic Maps’ used the ute’s laser scanner (mounted in the vertical plane) to produce a 3D map of it’s environment, using efficient mesh representation techniques. For further information, the interested reader should consult the relevant thesis.

1.4 Thesis Specifications

The first section of Chapter 1 has introduced the concept of ALVs and placed the HSV project at ACFR in context, with respect to similar research projects underway around the world. Details of the HSV project in terms of research interests, automation hardware and recent undergraduate theses have also been outlined. With this necessary background information covered, this thesis project may now be specified.

1.4.1 Objectives

The central aim of this thesis was, given the computing, sensing and actuation capabilities available on the HSV project’s test vehicle (see section 1.3.2), to develop, implement and test a system for an autonomous urban vehicle (AUV) which facilitates the following:

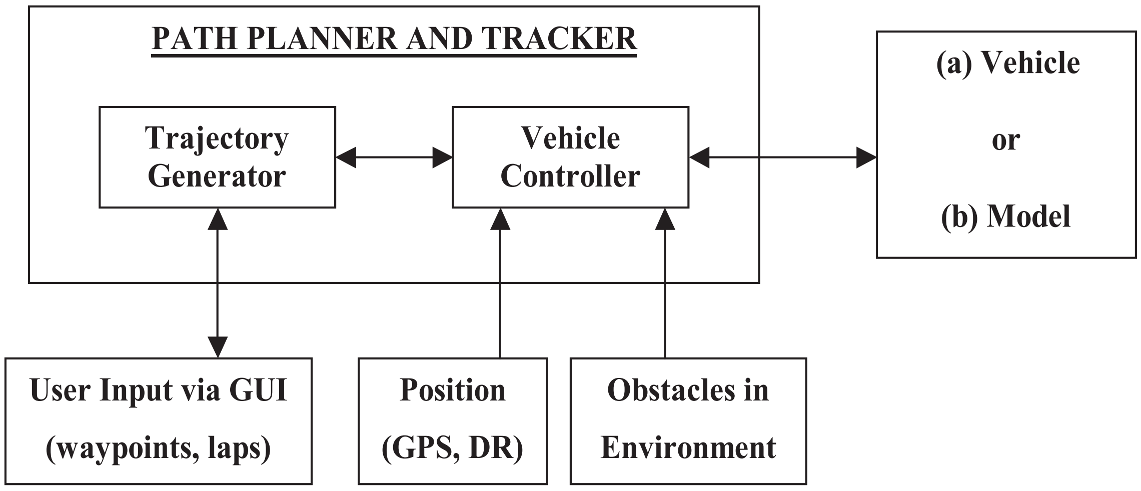


Figure 1.7: Overall path planning and tracking system structure

1. Path Planning
   (a) Allows the user (via a graphical user interface (GUI)), to specify the vehicle's trajectory as a set of waypoints.
   (b) Based upon these waypoints and using CCPP techniques, the path planning software should generate a physically trackable path that is, in turn, provided to the vehicle's path tracking controller.
2. Path Following (Tracking)
   (a) Tracks a path of arbitrary length and duration[22], reliably and accurately, controlling only the ute's steering[23].
   (b) Able to perform on-line path regeneration, so as to avoid obstacles present in the environment. These are to be detected and communicated to the vehicle controller by the reactive navigation module developed by Lee (2001).
   (c) The vehicle controller should be navigation platform independent, i.e this software module must be able to receive localisation information from a variety of sources, for example, from GPS, INS and dead-reckoning.

Achievement of these stated objectives would require the successful fusion of three key aspects of vehicle automation, namely: 1) localisation within the operating environment, 2) motion planning, based upon vehicle constraints and user/environmental

[22]It should be possible to specify a closed path, and to request that a number of laps of the trajectory be performed.
[23]Throttle and brake inputs are still be to provided by a 'safety driver' who rides in the vehicle at all times during testing.

inputs, and 3) the execution of this motion (control) (Wit 2001). Software modules[24] would be expected to reflect the multifaceted nature of this system. Note that the overall structure and interactions of the system can be seen in Figure 1.7. Clear documentation and annotated software were also seen as an imperative, so as to ensure that in future, system functionality can be both modified and/or expanded with relative ease if required.

1.4.2 Background and Motivation

There were a number of motivating factors behind the work conducted in this thesis. Firstly, as discussed in section 1.3.3, recent HSV related undergraduate theses have often retraced previous work and produced little in the way of tangible outcomes. This ensures minimal continuity between successive theses, and makes the development (over a number of years) of complex, multiple competency systems, challenging - to say the least. The research and development presented in this thesis attempts to 'tidy up' (and then expand significantly upon), a number of aspects (such as planning and control) already addressed in various theses over the last few years, and place them into a clear, readily accessible system.

Care was also taken to ensure that this path planning and tracking system was left in a state suitable to be built upon and, thus, provide the starting point for further developmental work in 2002. There exists a number of such potential 'follow on' projects, including the introduction of brake and/or throttle control, in addition to the eventual development of a complete C/C++ implementation of the entire system, these are all outlined in Chapter 6.

Finally, and perhaps most importantly, there is the system's 'utility value' (no pun intended). What was clearly identified at the start of this thesis was the need for a reliable system, available to all users of the ute, that would allow an operator to specify and closely track a path. Possible applications of such a robust system would include environmental mapping/surveying (used when performing SLAM), as well as the evaluation of data logging and controller/obstacle avoidance algorithms[25].

[24]Clearly, software design, implementation and testing is a core focus of this thesis.

[25]The system was developed so that additional software modules could be readily added to the basic system for testing under operational conditions.

1.4.3 Method of Approach

The developmental work can be divided into four subsections, namely, three of the key aspects of car-like robot automation as described in section 1.3.1, for which appropriate strategies for practical implementation need to be decided upon, and the software and hardware realisation of these modules on the ute.

A concurrent, incremental, approach to development and implementation was used, allowing the envisaged system to be developed on a number of fronts and evaluated (as an entire functional unit) at various levels of complexity. For example, the steering component of the overall system was first tested via simulation studies, then by 'in garage' tests[26], followed by field trials using a simple 'straight line tracker', before final field trials involving the tracking of a curved path (over a number of repetitive circuits) and the avoidance of obstacles (see Chapter 5 for implementation details.).

1.5 Thesis Overview

The structure of the remaining chapters reflect the four facet composition of the project, as discussed above. In Chapters 2, 3 and 4, all the technical content is presented and discussed, while Chapter 5 details the implementation, testing and evaluation of the 'theoretical' aspects of the thesis when fused together to form the complete path planning and following system. Finally, in Chapter 6, conclusions are outlined and recommendations for further work are made.

A short synopsis of each chapter is given below to assist in an appreciation of the scope and interconnectivity of the work carried out during the course of this thesis:

- Chapter 2 introduces the path planning problem for a car-like robot. A vehicle model is presented and its kinematic (nonholonomic) constraints explained. With this in mind, path representation and planning techniques for such vehicles are investigated. Finally two mathematical techniques for generating continuous curvature paths are developed for implementation in the path planning and tracking system.

[26]The front wheels of the vehicle are jacked up, and the steering controlled while vehicle motion is simulated.

- Chapter 3 investigates obstacle avoidance, a competency not provided in the path planning techniques from Chapter 2. Commonly used approaches to obstacle avoidance are introduced and the associated minimum cost algorithms necessary for extracting the shortest paths are explained. Combining these techniques with the output from the obstacle detection module developed by Lee (2001) a method for replanning the path to avoid obstacles is created.

- Chapter 4 takes the trajectories generated by the path (re)planning techniques from Chapters 2 and 3 and investigates the implementation of a path following controller. Firstly, car-like robot control is introduced and previously used techniques are discussed. Then the necessary low and high level controllers are developed and simulation results presented to verify their performance.

- Chapter 5 describes how the technical work presented in the three previous chapters is implemented on the vehicle. Aspects of software structure and design, including, real-time considerations, memory management, intermodule communications and the user interface are discussed. Finally the results of the system's field tests are presented and analysed.

- Chapter 6 concludes the thesis. Here a summary of the research and development undertaken is presented along with recommendations for possible future work.

- In addition to the thesis document, a software CD (attached to the inside back cover) has been included. This CD contains all the programs used in the development, testing and implementation of the system.

The reader should note that a conscious decision was made not to employ a single, extensive literature review - covering all aspects of the entire thesis. Rather, three smaller reviews have been placed within the relevant chapters (see Chapters 2, 3 and 4). It was felt that such an arrangement would assist the reader by providing the necessary background information 'in context', while also introducing key technical concepts utilised within a particular chapter. A block diagram (see Figure 1.8) has also been provided to act as a 'structural road map' for the thesis, showing concisely how the various components are related to each other.

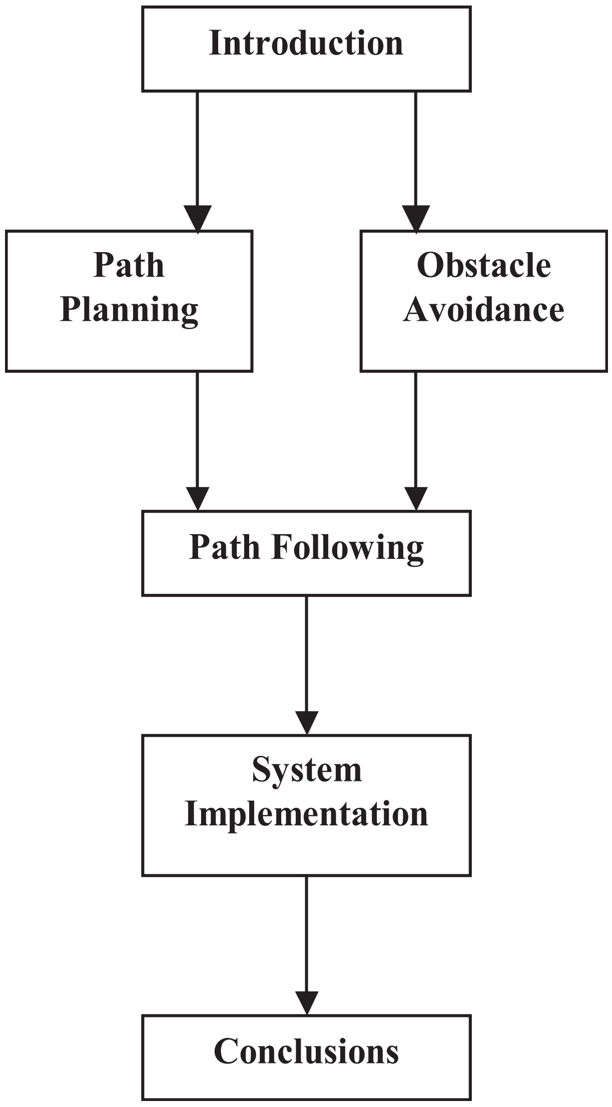


Figure 1.8: Thesis structure

Chapter 2

Path Planning

2.1 Introduction

Path planning, in the broadest sense, refers to the determination of how a robot will manoeuver within an environment (or workspace) to achieve its objectives. The type of robot being implemented places particular constraints on it's feasible motion. These must be accounted for when path planning techniques are developed to ensure that accurate and close path tracking is possible. Car-like robots, such as the 'ute', require specialised path generation techniques, as their motion is limited by the steering performance and wheel configuration (that is, by the nonholonomic constraints) of the vehicle.

In this chapter, the path planning problem for a car-like robot is outlined. A technique, known as continuous curvature path planning (CCPP), is then offered as a solution. The necessary mathematics for describing feasible paths using CCPP are introduced and used to formulate[1] two general path planning techniques. The second of these is closer to being optimal, and is used in the final system implementation[2].

[1]The path planning techniques were developed to meet the particular needs of the HSV project, i.e. for easy user path generation.

[2]In this chapter, path planning in an 'obstacle free' environment is considered, obstacle avoidance is introduced and discussed in Chapter 3.

2.2 Background and Critical Literature Review

Since the mid-1980's, when the pioneering work of researchers such as Kanayama and Miyake (1985) and Laumond (1986) was first published, a number of studies have addressed path planning for nonholonomic systems, of which the archetypal example is the car. Over the ensuing 15 years or so, the complexity of path representation and planning systems has increased astronomically. For example, a planner developed at INRIA in France uses genetic algorithms and is capable of trajectory generation inside 'obstacle dense' environments (Mazer et al. 1998, Fraichard and Ahuactzin 2001).

Much of the current work in this area focusses upon coupling optimised path generation directly with obstacle avoidance (Laumond, Sekhavat and Lamiraux 1998). For the purposes of the system being developed in this thesis, these abilities are distinct from one another, as the user will initially enter a desired path with no *a priori* knowledge of the environment available[3]. Obstacle avoidance (as discussed in Chapter 3) is implemented to be purely reactionary, i.e appropriate avoidance action is taken as obstacles are detected. Therefore, what is required of the basic 'off-line' path planning system is a method for specifying, representing and generating practical paths that acknowledge the kinematic constraints of a car-like vehicle. The remainder of this section introduces some of the important concepts and techniques required to perform this task.

2.2.1 Car-Like Robot Model

Firstly, a 'standard model' for the vehicle and its workspace is introduced. This is the representation of the car-like robot used throughout the development of the path planning, obstacle avoidance and tracking systems[4]. The model given below is based upon those described in Latombe (1991) and Scheuer and Fraichard (1996*b*).

Let $\mathcal{A}$ be a car-like robot, capable of only forward motion, modelled as a rigid rectangular body moving on a planar (two-dimensional) workspace, $\mathcal{W} \equiv \mathbb{R}^2$, that is free of obstacles. $\mathcal{A}$ is supported by four wheels making point contact with the ground, while it has two fixed rear wheels and two directional (steerable) front wheels. The

[3]Environmental maps represent a potential addition to the initial system configuration developed in this thesis.

[4]Note: this vehicle model is used as the nomenclature and workspace reference for the pose update model introduced in Chapter 4.

wheelbase (distance between front and rear wheels) is denoted by L. Λ is the configuration of $\mathcal{A}$ within $\mathcal{W}$ defined by the following quartet, (x, y, θ, κ) where (x, y) are the coordinates of the rear axle midpoint. θ (the vehicle's orientation) is the angle between the x-axis and the main axis of the vehicle $(-\pi \leqslant \theta \leqslant \pi)$[5]. Steering angle[6] is denoted as ϕ and determines the instantaneous centre of rotation Ω of the robot. The position of Ω determines the instantaneous radius of curvature ρ of the vehicle's path, given by $L/\tan\phi$ (Latombe 1991). The instantaneous curvature[7] κ of the robot is the inverse of the instantaneous radius of curvature $(\kappa = \rho^{-1})$. Finally, the vehicle's velocity at the centre of its rear axle is defined as V. This vehicle model is shown in Figure 2.1.

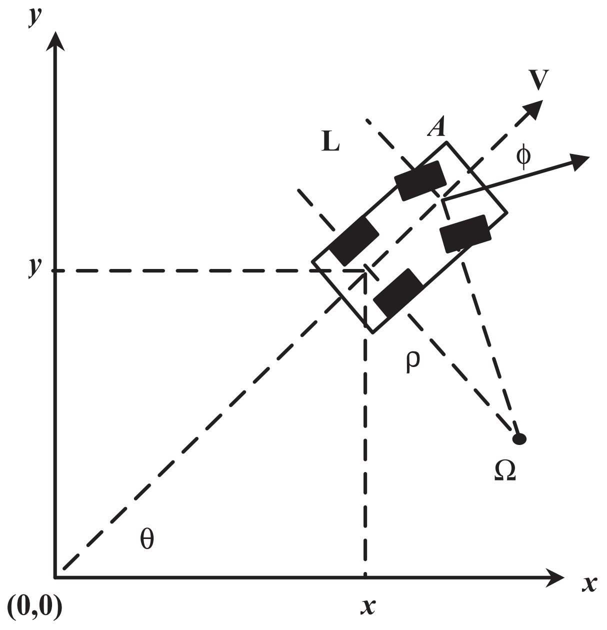


Figure 2.1: Car-Like robot model

2.2.2 Nonholonomic Motion Constraints

Based upon the definition of the car-like robot presented in the previous section, the vehicle, from a kinematic viewpoint, is subject to nonholonomic constraints that restrict its motion capabilities. These constraints are imposed by the vehicle's wheel

[5]Vehicle orientation is positive for angles measured counterclockwise from the x-axis and negative for clockwise angles.

[6]ϕ is the average orientation of the two front wheels of $\mathcal{A}$.

[7]Curvature κ is defined as the rate of change of orientation θ, w.r.t arc length, s, $\frac{d\theta}{ds}$ (Kreyszig 1999).

and steering configuration. In mathematical terms, nonholonomy arises when the derivative terms cannot be removed from a motion constraint (Scheuer and Fraichard 1996*b*). Two such constraints apply for a car-like vehicle.

The first nonholonomic constraint is based upon: (1) the fact that a single rigid body moving in a plane has only one centre of rotation (Bedford and Fowler 1996) and (2) the perfect rolling assumption for a wheel[8], i.e. a wheel must move in a direction normal to its axle (should roll, without sliding in a direction perpendicular to the axle). Therefore, when $\mathcal{A}$ is moving, the centre of rotation Ω of the vehicle must be located on the rear wheels' axle, at the point where the front and rear wheel axles intersect. Thus, the following constraint holds (Latombe 1991):

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0 \tag{2.1}$$

A second nonholonomic constraint results from the mechanical limitation of the steering angle. The robot's $\mathcal{A}$ steering angle is upper bounded[9]:

$$|\phi| \leqslant \phi_{max}, \tag{2.2}$$

Hence, the curvature of the robot's motion is also upper bounded by the following inequality:

$$|\kappa| \leqslant \kappa_{max} = L^{-1} tan\phi_{max} \tag{2.3}$$

These two analogous constraints can be rewritten to provide the second nonholonomic constraint for a car-like robot $\mathcal{A}$. This requires that the distance between R and Ω, the instantaneous radius of curvature ρ is lower bounded to a certain value $\rho_{min} = \kappa_{max}^{-1}$ and hence the following constraint, where v is the velocity of the vehicle, holds:

$$|\dot{\theta}| \leqslant \frac{|v|}{\rho_{min}} \tag{2.4}$$

which can be rearranged to give:

[8]This assumption is applicable, at least in terms of characterising vehicle motion, in most situations, although in practical implementations, particularly at high speeds, wheel slippage must be accounted for.

[9]The ute's maximum steering angle is approximately 35 deg.

$$\dot{x}^2 + \dot{y}^2 - \rho_{min}^2 \dot{\theta}^2 \geqslant 0 \tag{2.5}$$

Latombe (1991) shows that constraints 2.1 and 2.5 are indeed nonholonomic, because the derivative terms cannot be removed. As a result, the shapes of the feasible paths for $\mathcal{A}$ are restricted[10].

A third constraint is imposed by the performance characteristics of the steering controller itself. How fast it turns the front wheels (steering velocity) controls the derivative of curvature κ (w.r.t arc length). This quantity is known as the curve sharpness, and is a function of both vehicle and steering velocity, generally denoted by σ, where $\sigma = \dot{\kappa}$. Sharpness is also upper bounded:

$$|\sigma| \leqslant \sigma_{max}, \tag{2.6}$$

These motion limitations do not prevent the vehicle $\mathcal{A}$ from reaching a particular location, they 'merely' restrict the set of feasible paths $\mathcal{A}$ can follow to arrive there. Any path planning technique, such as those investigated in the next section and the two developed later in this thesis, must account for these constraints.

2.2.3 Continuous Curvature Path Planning

As already outlined, motion planning with nonholonomic constraints is a relatively new field of research, developed only in the last 15 years (Latombe 1991). All such path planning implementations require that any path generated meets the following three criteria:

1. Path curvature κ does not exceed, κ_{max}
2. Path sharpness σ does not exceed, σ_{max} [11]
3. Path curvature is continuous, i.e. the path is smooth

[10]The proofs that verify that constraints 2.1 and 2.5 are nonholonomic have been omitted, as they are lengthy and not a key area of this thesis. To develop a path planning system to meet the stated thesis specifications, what is required is an appreciation of the motion constraints imposed upon a car-like robot.

[11]Note: the apparent path sharpness is a function of velocity, i.e the faster the vehicle travels, the greater the path's sharpness will appear to be.

Criteria 1 and 2 have already been introduced in section 2.2.2 and are vehicle based constraints (with κ_{max} and σ_{max} becoming inputs to the path generator). While criteria 3, that of a continuous curvature (i.e. a continuous second derivative), is an intrinsic feature of the path generation method (and the mathematical functions used to describe the path) itself (Konishi and Takahashi n.d.).

Continuous curvature is an extremely important aspect of path planning for car-like robots, as a path possessing curvature discontinuities cannot be followed exactly unless the vehicle stops at this transition point and reorients its wheels[12] (Scheuer and Fraichard 1996*b*). In high speed vehicle implementations, stopping at curvature transitions points or accepting inaccurate path following is clearly not a practical option[13].

However, this aspect of the path planning problem was largely ignored until approximately 5 years ago when two researchers at INRIA, Scheuer and Fraichard (1996*b*) announced the world's first, obstacle avoiding, continuous curvature path planner (CCPP). Previously, path planning techniques typically created trajectories from the concatenation of straight lines and circular arcs. The associated curvature discontinuities (see Figure 2.3) were just accepted[14]. Such a path representation is often referred to as a 'Reeds and Shepp path' (Fraichard et al. 1999) - named after the 1990 paper where the authors expanded the optimal path planner for a forward moving car, developed by Dubins (1957), to a vehicle that can move both forwards and backwards (Reeds and Shepp 1990). Figure 2.2 shows how waypoints (vehicle poses) are connected using Reeds and Shepp path specification. Reeds and Shepp based techniques have remained popular until quite recently for two reasons:

- Optimal method (shortest path) for joining vehicle configurations.
- Computationally non-intensive.

However, approaches based on a combination of straight lines and circular arcs are now being superseded by CCPPs which use mathematical functions that generate paths which respect the three, aforementioned, path planning criteria. A variety of

[12]Continuous path curvature equates to a continuous vehicle steering angle.

[13]Even in low speed environments, such as a car park (an application currently being researched), this is generally an unacceptable restriction (Fraichard and Ahuactzin 2001).

[14]At the transition between straight and circular segments, there exists a step change in path curvature, as shown in Figure 2.3, which requires an infinite steering velocity, which is theoretically unsound - even if such an arrangement can be made to work in practice.

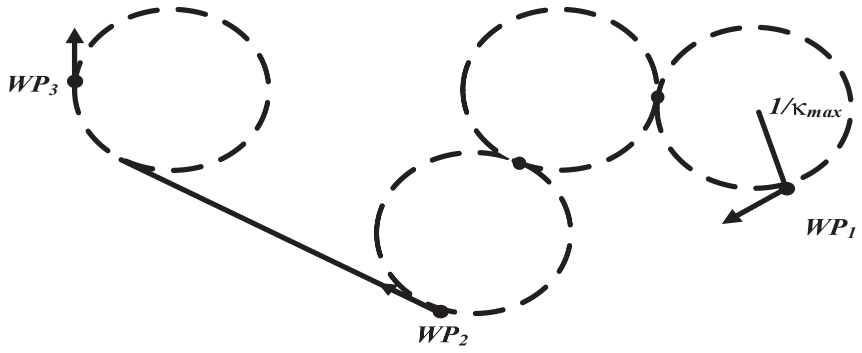


Figure 2.2: Joining waypoints by Reeds and Shepp paths

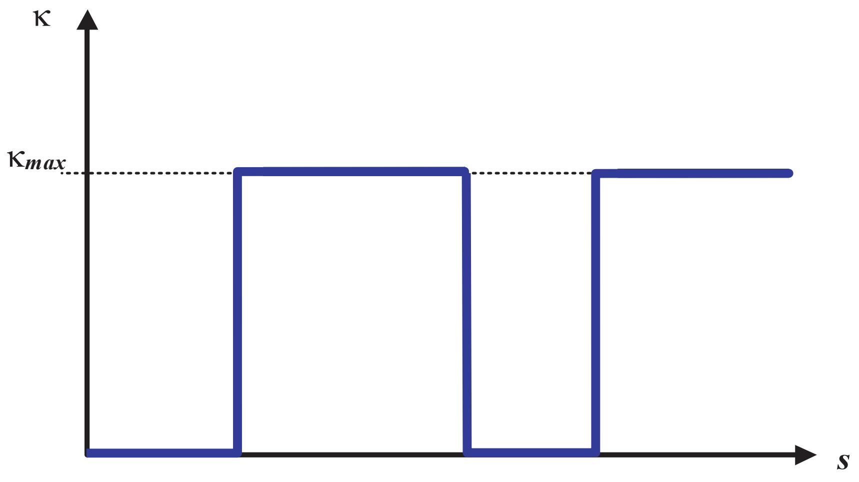


Figure 2.3: Curvature profile of a Reeds and Shepp path

curves are available for joining subgoals (waypoints) to create a complete continuous curvature path - recently used functions include the following:

- Quintic polynomials (Takahashi, Hongo and Ninomiya 1989)
- Polar splines (Nelson 1989)
- Clothoids, also known as cornu spirals (Scheuer and Fraichard 1996*b*)
- Cubic curvature polynomials (Nagy and Kelly 2001)

Upon reviewing the literature regarding CCPP techniques, it appears that the most 'popular' curves are those where curvature is a polynomial function of their arc length. These include clothoids, where curvature is a linear function of arc length (see Figure 2.5), given by the following expression:

$$\kappa(s) = \kappa_0 + as \tag{2.7}$$

Here s is the arc length, a is the clothoid coefficient and κ_0 is the initial curvature. Figure 2.4 shows a plot of a typical clothoid curve.

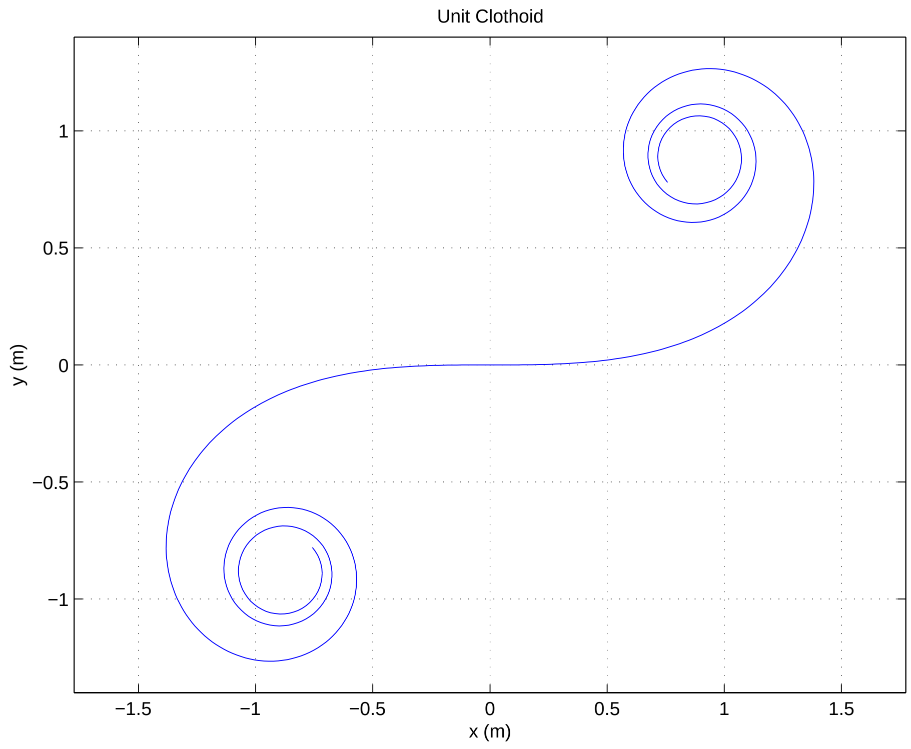


Figure 2.4: A unit clothoid

Another frequently used curve are the family of cubic curvature polynomials, whose curvature expression is given by:

$$\kappa(s) = \kappa_0 + a\kappa + b\kappa^2 + c\kappa^3 \tag{2.8}$$

Here s is the arc length, a, b and c are the curve sharpness coefficients, and κ_0 is the initial curvature. The use of cubic curvature curves has only recently come to prominence but has some very advantageous features including the ability to generate paths with continuous sharpness in addition to continuous curvature (i.e. providing 3rd derivative continuity). But these curves are difficult to implement in real-time and potentially not 'fail-safe', as they require the solution of four non-linear algebraic equations to generate each path segment.

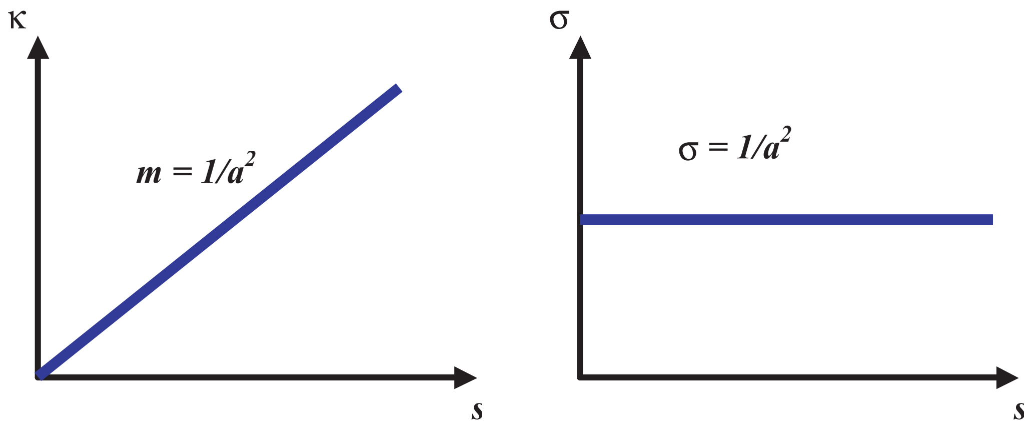


Figure 2.5: The curvature (left) and sharpness (right) profiles of a clothoid

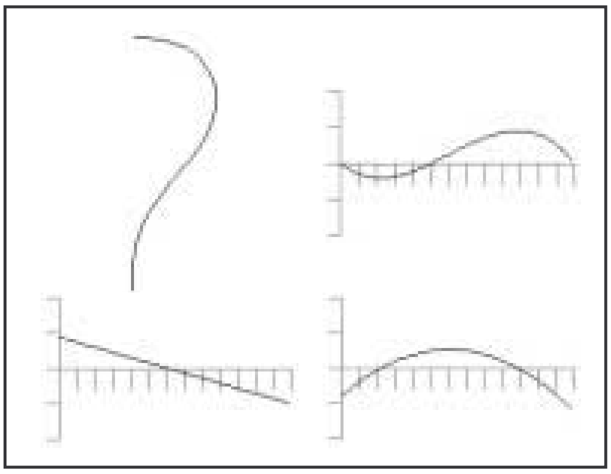


Figure 2.6: Clockwise from top left, cubic curve of polynomial curvature, its curvature and the first and second derivatives of curvature.

Hence, for the user entered path planner to be developed in this thesis, clothoids were selected to provide continuous curvature paths. Reasons for this selection include:

- They represent the 'simplest' curve which respects the three path planning requirements.
- Provide an 'easy' path for a steering controller to track, since curvature, and hence steering angle, vary linearly along its arc length[15].
- Such paths allows all necessary vehicle steering/motion variables to be matched to those of the actual car-like robot.
- They have been successfully implemented in previous CCPPs, while the underlying theory is well developed.

[15]This equates to a constant steering velocity.

In the following section, the mathematics that describes clothoids is introduced, then path specification is discussed along with two key continuous curvature path representations - the latter being central to the formulation of the two path planning techniques.

2.2.4 Clothoids

Clothoids, as already mentioned, are a family of curves whose curvature varies linearly with arc length[16], hence it can be constructed directly from its curvature profile, which is plotted in Figure 2.5. Such curves are widely used by civil engineers in the design of highway and railway routes for providing a smooth transition (so that lateral acceleration forces increase linearly) between straight and constant radius route segments (Öztan, Baykal, Müftüoglu and Sahin 1995). More recently (over the last 5 years), their applicability to CCPP for car-like robots, has seen clothoids become increasingly employed for the task of ALV path planning and trajectory smoothing.

The fundamental equations for a first-order clothoid and the notation used in the development of the path planner are now introduced.

A clothoid is a spiral curve which obeys the following rule:

$$\frac{s}{\kappa} = a^2 = constant \tag{2.9}$$

where κ is the curvature, s the clothoid arc length and a the clothoid coefficient.

Clearly, as arc length increases, the curvature of the clothoid must increase to maintain the constant ratio, leading to its characteristic spiral shape. The cartesian coordinates for the clothoid can be calculated via the following parametric equations (Spanier and Oldham 1987):

$$x[t] = a\sqrt{\pi} FresnelS[\frac{t}{\sqrt{\pi}}] \tag{2.10}$$

$$y[t] = a\sqrt{\pi} FresnelC[\frac{t}{\sqrt{\pi}}] \tag{2.11}$$

[16]A clothoid starts with $\kappa = 0$ (i.e. a straight line), then as the arc is traversed curvature increases linearly to create the characteristic spiral shape.

where x and y are the cartesian coordinates, t the parameter and $FresnelS$ and $FresnelC$ are the sine and cosine Fresnel integrals given by the following equations, respectively (note that z is the parameter of integration):

$$FresnelS[t] = \int_0^t sin[\frac{\pi z^2}{2}]dz \tag{2.12}$$

$$FresnelC[t] = \int_0^t cos[\frac{\pi z^2}{2}]dz \tag{2.13}$$

No analytic solution exists for either the sine or cosine Fresnel integrals, hence to calculate their value a numerical integration must be performed. Numerical integration routines such as MATLAB's 'quad8' are quite slow, which is an issue as generating a path requires that the two Fresnel integrals be evaluated at regular intervals (approximately every 0.5m). Calculation times can be reduced by using the Maclaurin series expansion for the Fresnel integrals (for $-2\pi \leqslant t^2/2 \leqslant 2\pi$), given below (Davis 1999):

$$FresnelS = t\sum_{i=0}^{\infty} \frac{(-1)^i (\frac{t^2}{2})^{2i}}{(4i+1)(2i)!} \tag{2.14}$$

$$FresnelC = t\sum_{i=0}^{\infty} \frac{(-1)^i (\frac{t^2}{2})^{2i+1}}{(4i+3)(2i+1)!} \tag{2.15}$$

Use of these power series approximations reduces calculation times by a factor of 4-5 by providing rapid convergence after the summation of only 5 or so terms[17]. Such time savings are especially important for 'on-line' (real-time) path regeneration, where calculation times must be minimised.

In addition to the cartesian equations, there exists a number of 'useful' equations based upon the clothoid parameter t (Gu 1999).

Clothoid arc length, s is given by the following equation:

$$s(t) = at \tag{2.16}$$

[17]These results were obtained on a PC running a 600MHz Pentium III with 128Mb of RAM.

The linear relationship between curvature and arc length can be seen in the equation below relating κ to t:

$$\kappa(t) = \frac{t}{a} \tag{2.17}$$

Integrating equation 4.18 w.r.t parameter t gives the equation for clothoid orientation (or gradient)[18]:

$$\theta(t) = \frac{t^2}{2} \tag{2.18}$$

Equations 2.16 and 4.18 can be combined to give an expression for κ as a function of arc length s as shown below:

$$\kappa(s) = \frac{s}{a^2} \tag{2.19}$$

Equation 2.19 can then be differentiated w.r.t s which results in the following expression for clothoid sharpness σ [19]:

$$\sigma = \frac{1}{a^2} \tag{2.20}$$

Finally, the maximum allowable sharpness σ_{max} can be approximated for a maximum vehicle velocity V_{max} by the following difference equation:

$$\sigma_{max} = \frac{\Delta\kappa}{V_{max}\Delta t} \tag{2.21}$$

Where $\Delta\kappa$ is the change in curvature (given by a relationship between steering angle and curvature, see section 2.2.1) and Δt is the time over which the change occurred.

2.3 Specifying and Constructing a Path

Prior to the formulation of a path generation strategy, trajectory specification and construction must be considered. The method of path specification was outlined in

[18]Note: the clothoid coefficient a is dropped from the orientation equation since a does not change the clothoid's orientation at a particular value of t, it merely acts as a scaling parameter.

[19]This shows that a clothoid's sharpness (i.e. its 3rd derivative) is indeed constant.

the thesis objectives - this being done by means of waypoints (i.e. a series of points that the vehicle must pass through) which the user would enter into the path planning and following system by means of a GUI[20]. Once entered, the trajectory generation software would interpolate between consecutive waypoints to construct a complete path for the vehicle to follow. The path planner would output the trajectory as a series of cartesian coordinates (x,y) within the local frame of reference. This path information would then become an input to the path following and obstacle avoidance modules.

2.3.1 Straight Lines

In the initial (i.e. first pass) implementation of the planning and tracking system, the user entered 'position only waypoints', i.e. each waypoint was an (x,y) position couplet. Paths were then constructed by fitting straight lines between consecutive waypoints. Hence, for a series of user specified waypoints, $\mathcal{WP}_{sl}$:

$$\mathcal{WP}_{sl} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_k & y_k \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

Each path segment, $\mathcal{P}$ has the following general (straight line) equation:

$$\mathcal{P}_k : \quad \left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right) x - y + y_k - \left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right) x_k = 0 \qquad (2.22)$$

Clearly, a trajectory composed only of straight lines does not possess the desired continuous curvature profile, in fact it is continuous in position only. It is discontinuous in both orientation (1st derivative) and curvature (2nd derivative). Hence, to track such a path exactly the vehicle would need to stop at each segment intersection and be picked up and rotated to the next segment's orientation (a somewhat impractical approach!). However, such a trajectory construction technique was only used as an interim means of testing the correct operation of the path tracking control algorithm.

[20]Waypoints were chosen as they are a simple and quick means for a user to specify a trajectory.

As discussed in Chapter 4, the control law implemented here performs a linearisation on the nearest path element.

Figure 5.4.1 shows a path consisting of only straight line segments. The orientation and curvature discontinuities at the waypoint (path segment) boundaries are easily recognised. The inadequacies of straight line based paths are further highlighted by the relatively poor path tracking displayed by the vehicle in simulation and field test results presented in Chapters 4 and 5 respectively.

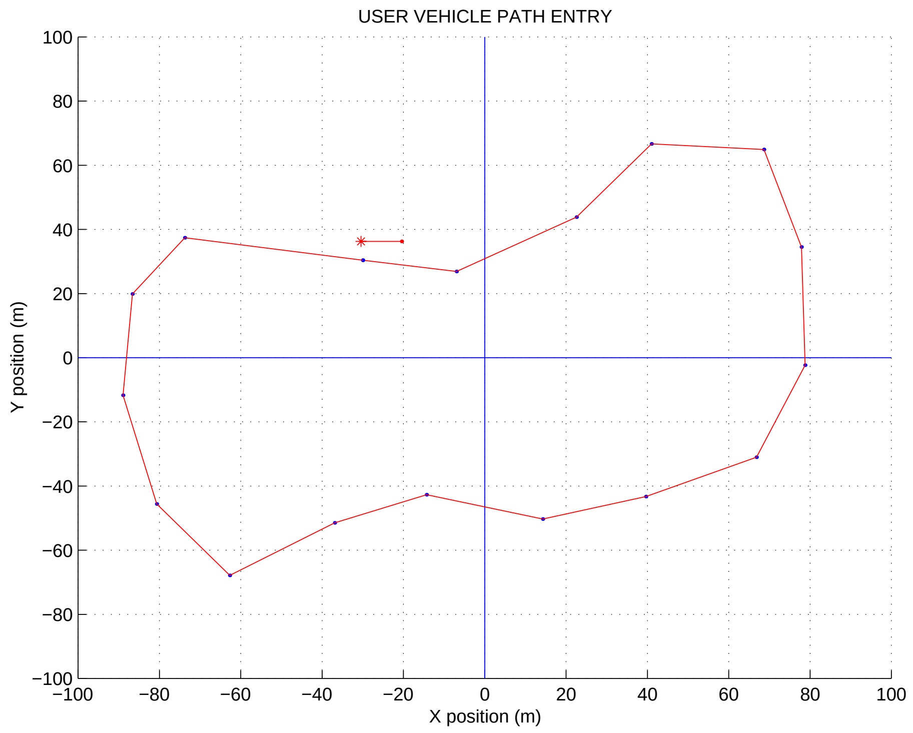


Figure 2.7: Example of a path consisting of waypoints joined by straight lines

2.3.2 Continuous Curvature Paths

For the generation of continuous curvature paths, more information (additional boundary conditions) is required at each particular waypoint[21]. A waypoint is now specified by the following quartet, known as the ‘posture’, (x, y, θ, κ) (Nagy and Kelly 2001). The user is required to enter the ‘pose’ (x, y, θ) for each path subgoal while the path curvature κ at each waypoint is set to 0. Therefore, a path must be generated from a series of waypoints, $\mathcal{W}_{cc}$ of the form:

[21]Unlike the straight line paths, a continuous curvature path must be continuous in position, orientation (1st derivative) and curvature (2nd derivative).

$$\mathcal{WP}_{cc} = \begin{bmatrix} x_1 & y_1 & \theta_1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_k & y_k & \theta_k & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & \theta_n & 0 \end{bmatrix}$$

In the following sections, two methods of path representation are outlined for joining posture specified waypoints while maintaining the continuous curvature conditions. These allow paths to be constructed from a series of line segments, clothoid arcs and circular arcs, and form the basis of the path planning techniques developed in sections 2.4 and 2.5.

2.3.3 Elementary Paths

A commonly used approach for connecting two posture configurations in a continuous curvature manner, is that of an 'elementary path' (see Figure 2.8), formulated by Kanayama and Hartman (1989). These are formed from the concatenation of two symmetric clothoid arcs[22] and produce a path Π (based upon equation 2.19) of total length l, such that (Scheuer and Fraichard 1996*b*):

$$\Pi = \begin{cases} \kappa(s) = \sigma s, & 0 \leqslant s \leqslant l/2 \\ \kappa(s) = \sigma(l - s), & l/2 \leqslant s \leqslant l \end{cases}$$

Over the first clothoid segment of an elementary path, the curvature κ increases linearly from 0 at the initial waypoint, to a maximum value of $\sigma l/2$, halfway along the path at the end of the first clothoid. The second clothoid segment is a mirror image of the first, with a linear curvature profile that decreases in curvature from its maximum value back to 0 at the second waypoint. This curvature profile is shown in Figure 2.9. The path Π that results is clearly symmetric, hence for an elementary path to exist between two vehicle configurations, the start and end postures, defined as $q_a = (x_a, y_a, \theta_a, 0)$ and $q_b = (x_b, y_b, \theta_b, 0)$ respectively, must be symmetric w.r.t the line that joins the initial and final postures, i.e. $\beta_a = \beta_b = \Delta\theta/2$, where $\Delta\theta$, the

[22]Note that the two clothoid arcs that are joined together to form an elementary path have sharpness values that are opposite in sign to each other. This is because the first segment turns the wheel in one direction while the second turns it in the other.

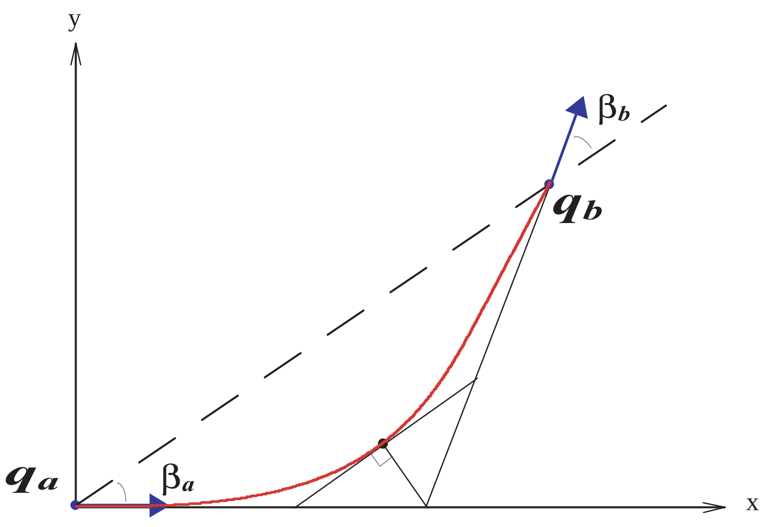


Figure 2.8: Elementary path construction

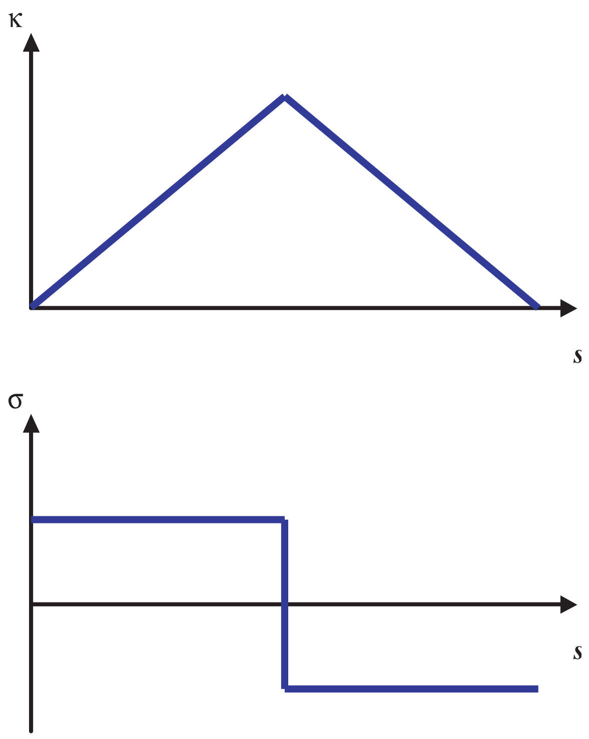


Figure 2.9: Curvature and sharpness profiles of an elementary path

change in orientation between the two waypoints is given by $|\theta_b - \theta_a|$. Mathematically, for two vehicle configurations to be symmetric the following equality must be true:

$$(x_b - x_a)\sin\left(\frac{\theta_b + \theta_a}{2}\right) = (y_b - y_a)\cos\left(\frac{\theta_b + \theta_a}{2}\right) \tag{2.23}$$

In addition to the requirement that elementary paths only join symmetric waypoints, such a path will only be considered feasible if it does not exceed the maximum curvature κ_{max} and sharpness σ_{max} applicable to the car-like robot $\mathcal{A}$, as defined in section 2.2.1. If these conditions are met, the path that results will conform to the three path planning criteria for a car-like robot introduced in section 2.2.3, and represent a physically trackable path for the vehicle.

A general solution was developed by Scheuer and Fraichard (1996*a*) for determining the existence of an elementary path between two symmetric posture configurations, q_a and q_b, where κ_a and $\kappa_b = 0$ and the maximum allowable curvature κ_{max} is not exceeded[23]. The mathematics that define this solution are outlined below in Theorem 1, see Scheuer and Fraichard (1996*a*) for the proof.

Theorem 1 *Given two symmetric configurations (postures) q_a and q_b, rewrite the vectors $(x_b - x_a, y_b - y_a)$ in polar coordinates as $(rcos(\theta_a + \alpha), rsin(\theta_a + \alpha))$ where $0 \leqslant r \leqslant +\infty$, $|\alpha| \leqslant \pi$ and:*

$$r = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

$$\alpha = \arccos((x_b - x_a)/r)$$

An elementary path that respects the minimum turning radius ρ_{min} (maximum curvature κ_{max}) constraint exists between q_a and q_b if and only if r and α respect the following constraints:

$$0 \leqslant |\alpha| \leqslant \theta_{root} \qquad \textit{and} \qquad r \geqslant 4\rho_{min}\sqrt{|\alpha|}D_1(|\alpha|)$$

where D_1 is the function defined over $[0, \pi]$ as:

$$D_1(\zeta) = \cos\zeta \int_0^{\sqrt{\zeta}} \cos u^2 du + \sin\zeta \int_0^{\sqrt{\zeta}} \sin u^2 du$$

and where θ_{root} ($\doteqdot 2.298$) is the unique root of D_1 over $[0, \pi]$. The function D_1 has been plotted over $[0, \pi]$ in Figure 2.10, note the position of θ_{root}.

As a demonstration, the 'reachable space' from an initial (null) configuration of $(0, 0, 0, 0)$ was generated for $\rho_{min} = 13m$ within a 100x50m region, the results being shown in Figure 2.11. The blue region is where a solution exists and the red region is 'inaccessible' i.e. no solution for an elementary path between the two points exists.

[23]This solution could potentially be used by the path entry module for determining if a feasible path exists between two user entered waypoints and providing feedback to the user.

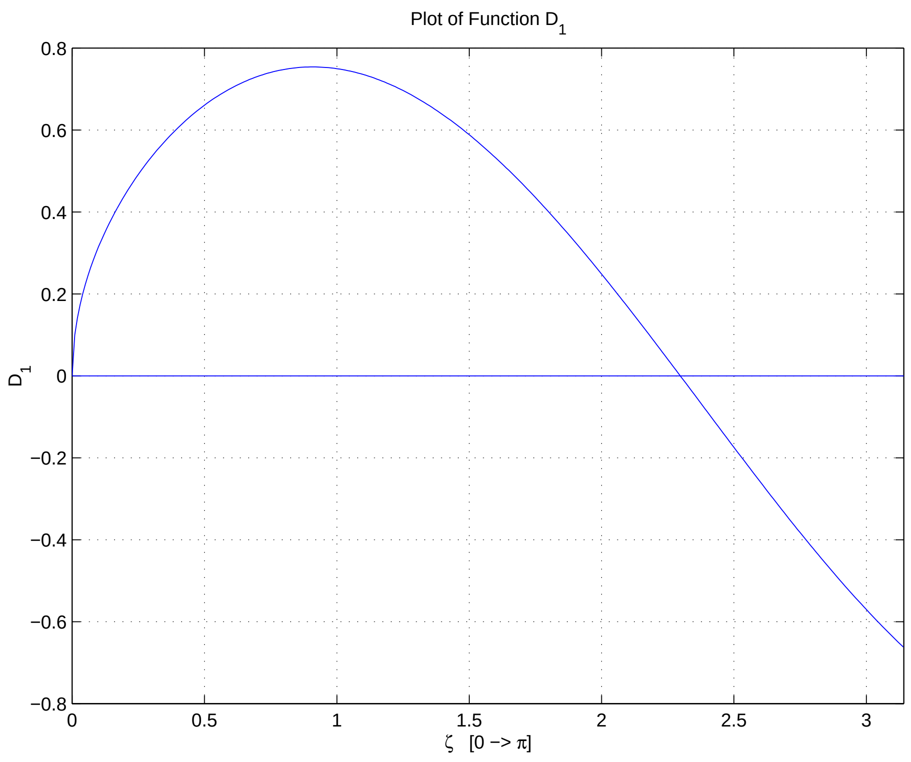


Figure 2.10: Plot of the elementary path function D_1

This investigation found that for this, very elegant, general solution, as expected, the existence of a feasible path was a function of: (1) the distance r between symmetric waypoints, (2) the orientation change α between symmetric waypoints and (3) their relative positions. However, this approach was deemed not applicable for the needs of the path planning system required as part of this thesis for two key reasons:

1. Maximum path sharpness σ_{max} was not a constraint accounted for by the general solution, despite being one of the three trajectory planning constraints.
2. Consecutive waypoints entered by the user will typically not be a symmetric pair, and the general solution offered by Scheuer and Fraichard (1996a) does not account for this.

In section 2.4 a path planning technique which validates the existence of, and constructs feasible paths from clothoid arcs and straight segments, while overcoming the aforementioned limitations is developed.

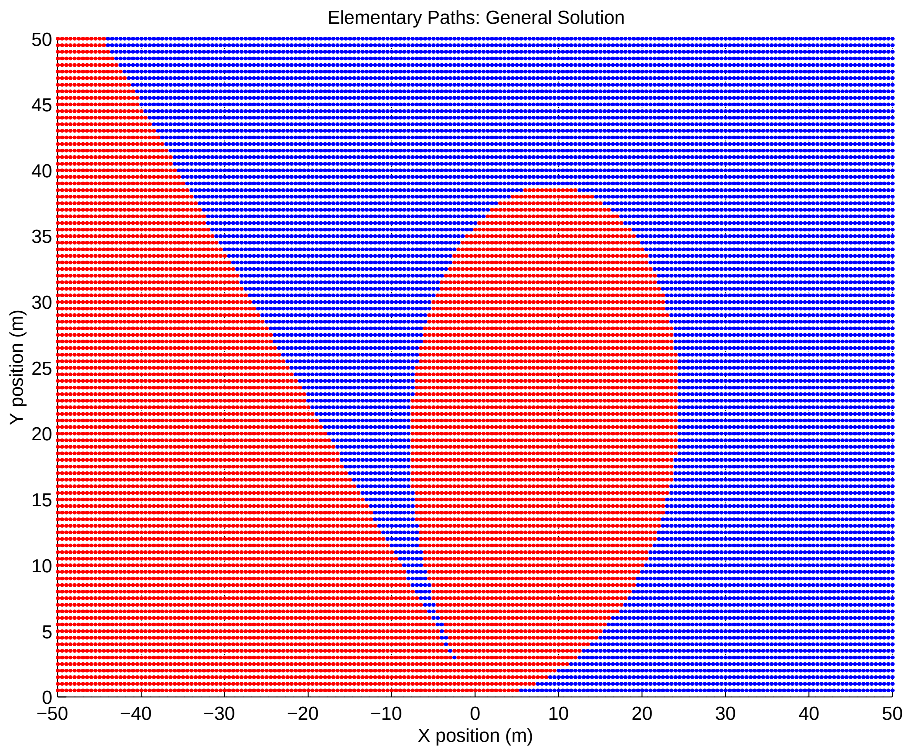


Figure 2.11: Results of elementary path general solution for symmetric waypoints

2.3.4 Smooth Reeds and Shepp Paths

In section 2.2.3, Reeds and Shepp paths were introduced. Such trajectories represent the optimal (shortest) means of joining two postures together, but they are not curvature continuous. Based upon these paths, a more flexible and optimal[24] means of joining symmetric postures together (than elementary paths) is a class of trajectories known as smooth Reeds and Shepp paths. Like elementary paths, these meet the three path planning constraints but consist of three components: straight lines, clothoid arcs and circular arcs. Figure 2.12 demonstrates how such a path is constructed.

Smooth Reeds and Shepp paths are defined in the following way (Fraichard et al. 1999), supposing that q_a and q_b are two symmetric postures as defined in section 2.3.3. Starting at q_a [25] the steerable wheels are turned at their maximum rate transcribing a clothoid (1) of sharpness σ_{max}. Once the maximum steering angle is reached at intermediate waypoint 1, $q_{int1} = (x_{int1}, y_{int1}, \theta_{int1}, \kappa_{max})$ a circular arc (2) of radius $1/\kappa_{max}$ is followed until intermediate waypoint 2, $q_{int2} = (x_{int2}, y_{int2}, \theta_{int2}, \kappa_{max})$ is

[24]Optimality is still to be proven for smooth Reeds and Shepp paths, but it is strongly suspected that they represent the shortest continuous curvature paths for joining waypoint postures (Fraichard and Ahuactzin 2001).

[25]Note: the path segments are numbered to correspond to the segments shown in Figures 2.12 and 2.13.

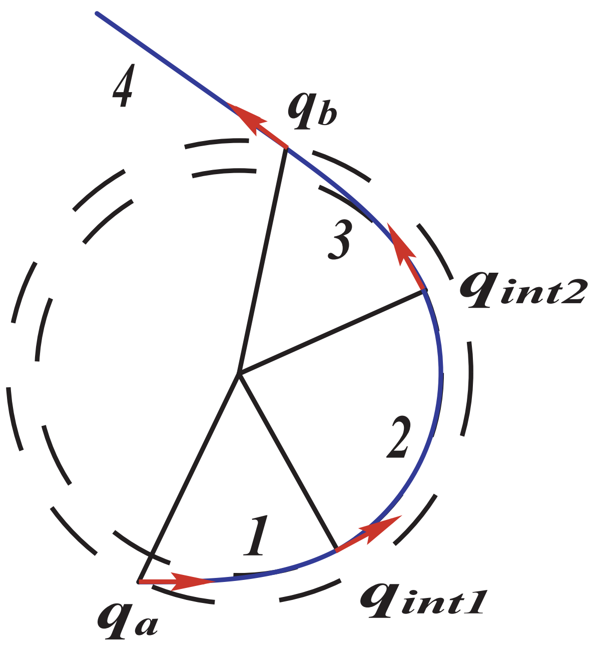


Figure 2.12: Construction of a smooth Reeds and Shepp path

reached. Finally, the wheels are turned back to a steering angle of 0 deg at their maximum rate, tracing out a clothoid arc (3) of sharpness $-\sigma_{max}$ until the second waypoint q_b is reached. Line segments (4) are used to join the clothoid/circular arc segments. This results in the curvature profile as shown in Figure 2.13.

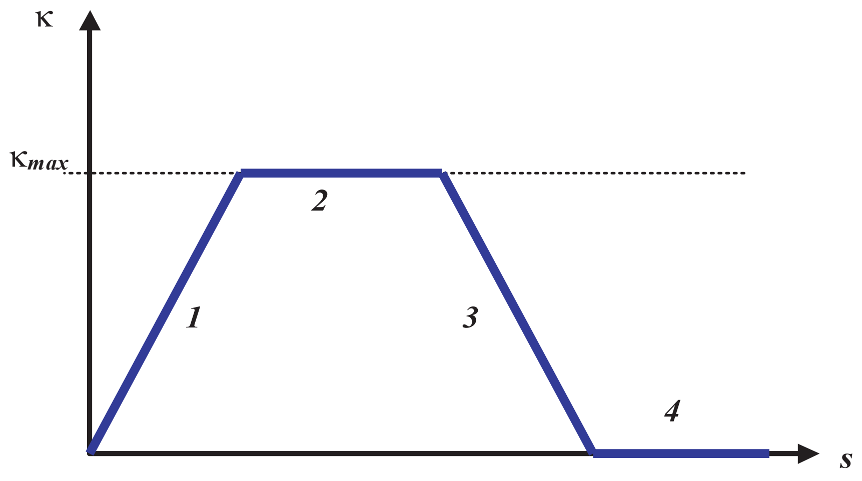


Figure 2.13: Curvature profile of a smooth Reeds and Shepp path

Two variants of smooth Reeds and Shepp paths exist when:

- The angle between q_a and q_b is 0 deg, which means that the waypoints are joined by a straight line.

- The orientation change midpoint ($\Delta\theta/2$) is reached prior to the maximum curvature κ_{max} being reached. In this scenario, the circular arc segment is not required and hence the path is termed degenerate, with the smooth Reeds and Shepp path now being effectively an elementary path.

In section 2.5 a technique for checking for the existence of, and generating feasible smooth Reeds and Shepp paths from non-symmetric postures is developed for use in the path planning module.

2.4 CCPP: Planning Technique 1

Given a series of non-symmetrical (i.e. equality 2.23 is not true) posture waypoints $\mathcal{WP}_{cc}$ as defined in section 2.3.2, a feasible path that joins these waypoints and meets the three path planning criteria is required. The first CCPP technique developed fits a clothoid pair (i.e. an elementary path) of the greatest sharpness allowable without exceeding the maximum curvature of the vehicle $\mathcal{A}$ between the two waypoints. Elementary path segments are connected by straight lines to complete the trajectory. What follows is an explanation of how such a path is generated.

Let q_a and q_b be two consecutive non-symmetric waypoints (defined as in section 2.3.2) between which a continuous curvature path must be placed (see Figure 2.14). Based upon their pose (x, y, θ), the equation of the line passing through each waypoint, $\mathcal{L}_a$ and $\mathcal{L}_b$ is calculated. The start and end points of the elementary path will lie on $\mathcal{L}_a$ and $\mathcal{L}_b$ respectively, these equations being given by:

$$\begin{aligned} \mathcal{L}_a : \quad & \tan\theta_a x - y + (y_a - (\tan\theta_a)x_a) = 0 \\ \mathcal{L}_b : \quad & \tan\theta_b x - y + (y_b - (\tan\theta_b)x_b) = 0 \end{aligned} \tag{2.24}$$

Next, the orientation change $\Delta\theta$ between the two waypoints is calculated as the following difference[26]:

$$\Delta\theta = \theta_b - \theta_a \tag{2.25}$$

[26]Note: for all calculations performed involving the addition or subtraction of angles a pi to -pi 'wraparound check' is carried out to ensure that all angles remain within the bounds defined by the workspace of the vehicle $\mathcal{W}$ in section 2.2.1.

The intersection point (x_c, y_c) between $\mathcal{L}_a$ and $\mathcal{L}_b$ is found by solving the line equations 2.24 simultaneously:

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} tan\theta_b & -1 \\ tan\theta_a & -1 \end{bmatrix}^{-1} \begin{bmatrix} (tan\theta_b)x_b - y_b \\ (tan\theta_a)x_a - y_a \end{bmatrix} \tag{2.26}$$

The midpoint of the elementary path that joins the lines $\mathcal{L}_a$ and $\mathcal{L}_b$ will lie on the line $\mathcal{L}_{mp}$ that bisects the angle ϕ (where ϕ is the angle between $\mathcal{L}_a$ and $\mathcal{L}_b$), hence $\mathcal{L}_{mp}$ passing through the point (x_c, y_c). The orientation ψ of the bisector $\mathcal{L}_{mp}$ [27] is given by:

$$\psi = \theta_a + sign(\Delta\theta)(\pi/2 + |\Delta\theta/2|) \quad ^{28} \tag{2.27}$$

The equation for the bisecting line $\mathcal{L}_{mp}$ can now be derived as:

$$\mathcal{L}_{mp}: \qquad \tan\psi x - y + (y_c - (\tan\psi)x_c) = 0 \tag{2.28}$$

Prior to fitting the clothoid segment, the orientation γ perpendicular to $\mathcal{L}_{mp}$ is required, this angle and the preceding steps are illustrated in Figure 2.15. This is the orientation of the elementary path's midpoint and is the angle halfway between θ_a and θ_b, being equal to:

$$\gamma = \psi - sign(\Delta\theta)\pi/2 \tag{2.29}$$

To generate the clothoid segments for the elementary path, the clothoid parameter t is required. This can be obtained from equation 2.18 and the orientation change over the first clothoid segment between q_a and $\mathcal{L}_{mp}$. The clothoid parameter for the elementary path segments t_{ep} is thus given by:

$$t_{ep} = \sqrt{2(|\gamma - \theta_a|)} \tag{2.30}$$

A check is now performed to determine if a clothoid of parameter size t_{ep} exceeds the maximum curvature κ_{max} when the sharpness of the clothoid is at it's maximum

[27]The halfway point of the elementary path (where half the required orientation change has occurred) will lie on this line $\mathcal{L}_{mp}$.

[28]The 'sign(x)' operator returns -1 if $x < 0$ and $+1$ if $x > 0$.

value σ_{max}. If κ_{max} is exceeded, the clothoid's sharpness σ is set (reduced) to the value that ensures that the maximum curvature along the elementary path is κ_{max}. The criterion used in this check can be obtained by combining equations 4.18 and 2.20 and is performed as follows:

$$\begin{aligned} &if \quad \sqrt{\sigma_{max}} t_{ep} > \kappa_{max} \\ &\sigma = \left(\tfrac{\kappa_{max}}{t_{ep}}\right)^2 \end{aligned} \tag{2.31}$$

The clothoid segment (for $t : 0 \longrightarrow t_{ep}$) can now be generated from the equations 2.10 and 2.11, using either numerical integration, equations 2.12 and 2.13, or the power series approximations, equations 2.14 and 2.15, to calculate the Fresnel integrals to produce the clothoid profile for one half of the elementary path X_{cloth} and Y_{cloth} (see Figure 2.16). Note that the y coordinates of the clothoid segment must be multiplied by the $sign(\Delta\theta)$ to ensure that the segment turns in the correct direction. The x and y coordinates of the clothoid segment are now rotated by the matrix $R(\omega)$[29] to the orientation of the initial waypoint q_a:

$$\begin{bmatrix} X_{cloth_1} \\ Y_{cloth_1} \end{bmatrix} = R(\theta_a) \begin{bmatrix} X_{cloth} \\ Y_{cloth} \end{bmatrix} \tag{2.32}$$

$$where \quad R(\omega) = \begin{bmatrix} \cos\omega & -\sin\omega \\ \sin\omega & \cos\omega \end{bmatrix}$$

Now that the first clothoid segment of the elementary path has been generated, its start $(X_{cloth_1}(0), Y_{cloth_1}(0))$ and end points $(X_{cloth_1}(t_{ep}), Y_{cloth_1}(t_{ep}))$ must be fitted to the lines $\mathcal{L}_a$ and $\mathcal{L}_{mp}$, respectively. This is done by calculating the offset (x_{off_1}, y_{off_1}) which is added to the clothoid vectors $(X_{cloth_1}, Y_{cloth_1})$. The offset (or connection point) (x_{off_1}, y_{off_1}) must satisfy the equation for $\mathcal{L}_a$ while the end point of the clothoid segment (i.e. the midpoint of the elementary path) $(x_{off_1} + X_{cloth_1}(t_{ep}), y_{off_1} + Y_{cloth_1}(t_{ep}))$ must satisfy the line equation $\mathcal{L}_{mp}$. Solving these two linear conditions simultaneously gives the offset point:

[29]This matrix simply rotates a set of cartesian coordinates (x,y) about the origin (0,0) by an angle ω.

$$\begin{bmatrix} x_{off_1} \\ y_{off_1} \end{bmatrix} = \begin{bmatrix} tan\theta_a & -1 \\ tan\psi & -1 \end{bmatrix}^{-1} \begin{bmatrix} (tan\theta_a)x_a - y_a \\ (tan\psi)x_c - y_c - \tan\psi X_{cloth_1}(t_{ep}) + Y_{cloth_1}(t_{ep}) \end{bmatrix} \tag{2.33}$$

The offsets are now added to the clothoid segment $(X_{cloth_1}, Y_{cloth_1})$ to produce the first half of the elementary path (X_{EP1}, Y_{EP1}), as shown in Figure 2.17:

$$\begin{bmatrix} X_{EP1} \\ Y_{EP1} \end{bmatrix} = \begin{bmatrix} X_{cloth_1} \\ Y_{cloth_1} \end{bmatrix} + \begin{bmatrix} x_{off_1} \\ y_{off_1} \end{bmatrix} \tag{2.34}$$

The endpoint of (X_{EP1}, Y_{EP1}), i.e. the midpoint of the clothoid elementary path is defined as (x_{mp}, y_{mp}) where $x_{mp} = (X_{EP_1}(t_{ep})$ and $y_{mp} = Y_{EP_1}(t_{ep})$. The second half of the elementary path (X_{EP2}, Y_{EP2}) is generated in a similar manner to the first. Due to the symmetry of an elementary path, the path generation process begins with the basic clothoid segment (X_{cloth}, Y_{cloth}). Since the clothoids that comprise an elementary path have a sharpness opposite in sign to each other (one curves to the left, the other to the right) the y coordinates (Y_{cloth}) are multiplied by $-sign(\Delta\theta)$. The clothoid position vectors are rotated by an angle of $(\theta_b + \pi)$ [30] to produce X_{cloth_2} and Y_{cloth_2}. The new start $(X_{cloth_2}(0), Y_{cloth_2}(0))$ and end points $(X_{cloth_2}(t_{ep}), Y_{cloth_2}(t_{ep}))$ of the rotated clothoid segments must be fitted to the lines $\mathcal{L}_b$ and $\mathcal{L}_{mp}$. This is done by calculating the offset position (x_{off_2}, y_{off_2}) obtained from equation 2.33 [31]. Using equation 2.34 to add the offset, the second clothoid segment (X_{EP2}, Y_{EP2}) is obtained, and the creation of the elementary path is complete (see Figure 2.18).

To ensure that the path generated does actually fit between the two non-symmetric waypoints q_a and q_b, a geometrical verification is performed. Two triangles, $\triangle A$ (vertices: A_1, A_2 and MP) and $\triangle B$ (vertices: B_1, B_2 and MP) are defined in the following way:

- A_1: initial waypoint - (x_a, y_a)
- A_2: first clothoid segment connection point - (x_{off_1}, y_{off_1})

[30] The second clothoid segment is effectively 'in reverse', as it approaches the waypoint q_b with an orientation of θ_b (hence the rotation of $\theta_b + \pi$), rather than leaving the waypoint with an orientation of θ_b.

[31] The line coefficients and endpoint values must be changed appropriately.

- B_1: second waypoint - (x_b, y_b)
- B_2: second clothoid segment connection point - (x_{off_2}, y_{off_2})
- MP: elementary path midpoint - (x_{mp}, y_{mp})

The achievement of a proper and feasible elementary path fit is indicated by the longest side of both triangles ($\triangle A$ and $\triangle B$) being between the waypoint and the midpoint (see Figure 2.19). Or, in mathematical terms, the following four conditions should be true:

$$\triangle A: \quad Condition \quad 1: \quad \overrightarrow{A_1MP} \geqslant \overrightarrow{A_1A_2} \tag{2.35}$$

$$Condition \quad 2: \quad \overrightarrow{A_1MP} \geqslant \overrightarrow{A_2MP} \tag{2.36}$$

$$\triangle B: \quad Condition \quad 3: \quad \overrightarrow{B_1MP} \geqslant \overrightarrow{B_1B_2} \tag{2.37}$$

$$Condition \quad 4: \quad \overrightarrow{B_1MP} \geqslant \overrightarrow{B_2MP} \tag{2.38}$$

If these four conditions are evaluated as ‘true’, then the path generation process is completed by joining the waypoints and elementary paths together using line segments, this is shown in Figure 2.20. This is possible because the end and beginning of consecutive elementary paths have the same orientation and lie on the same line.

To illustrate the typical paths created, and allow comparison between the two CCPP techniques developed, two reference waypoint sets were defined. Prior to trajectory generation the maximum allowable curvature κ_{max} (steering angle) and path sharpness σ_{max} must be defined, typical values for a ute travelling at a maximum velocity of 20 km/h were selected as follows:

- $\kappa_{max} = 0.1m^{-1} \qquad (\phi \doteqdot 16deg)$
- $\sigma_{max} = 0.0156m^{-2} \quad or \quad a_{max} = 8m \qquad (Equation \quad 2.21)$

The paths and their corresponding curvature profiles generated from the two reference sets are shown in Figures 2.21, 2.22, 2.23 and 2.24. Clearly, the trajectories possess the desired continuous curvature profile without exceeding κ_{max} or σ_{max}, and hence meet the path planning criteria for a car-like robot outlined in 2.2.3. As stipulated

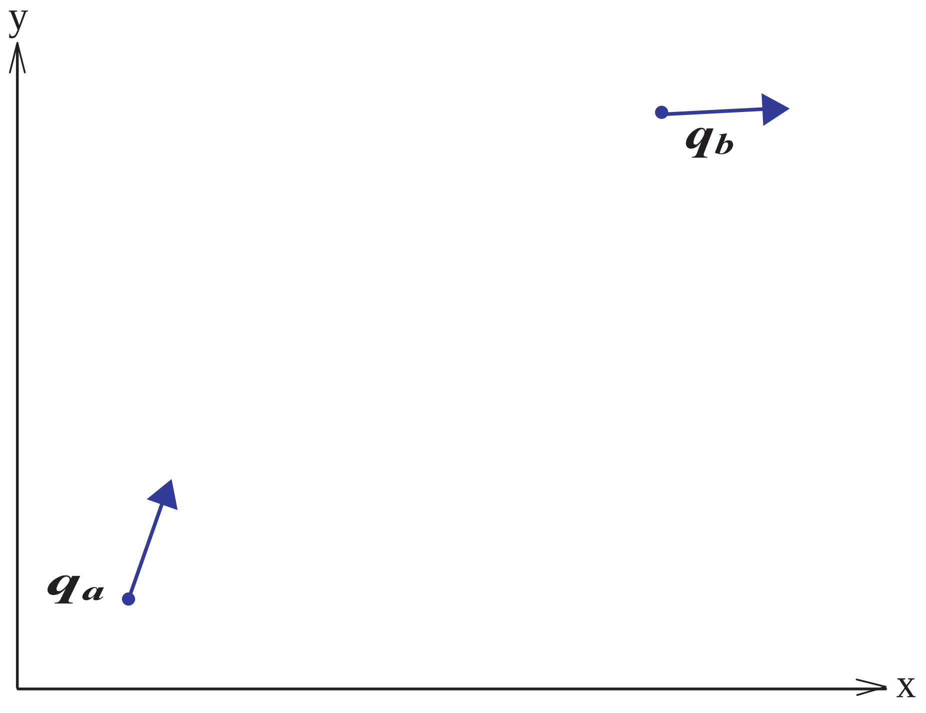


Figure 2.14: CCPP Technique 1: Step 1

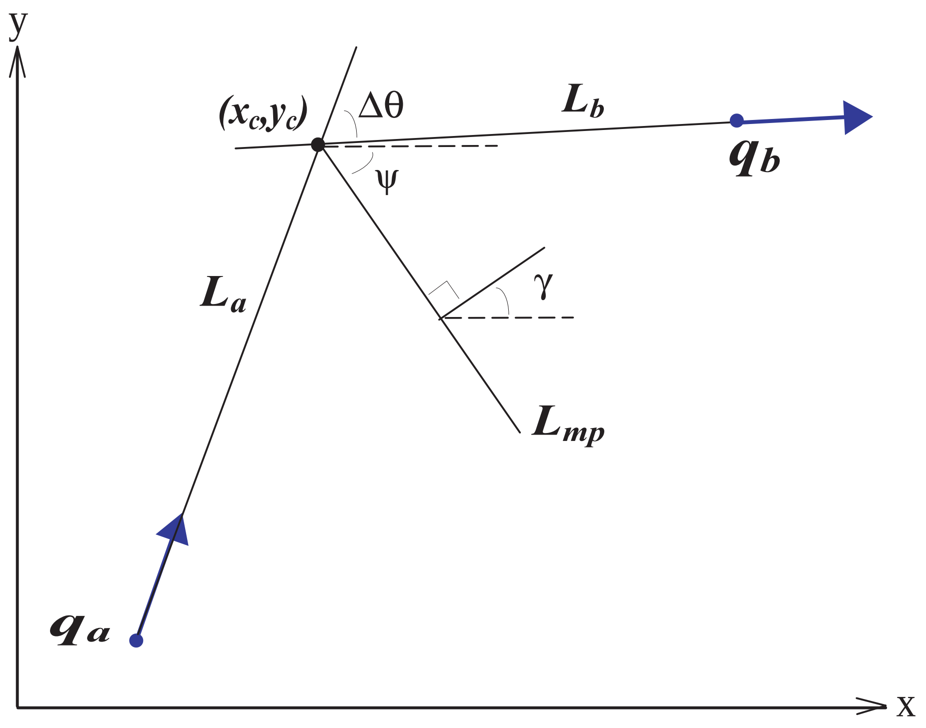


Figure 2.15: CCPP Technique 1: Step 2

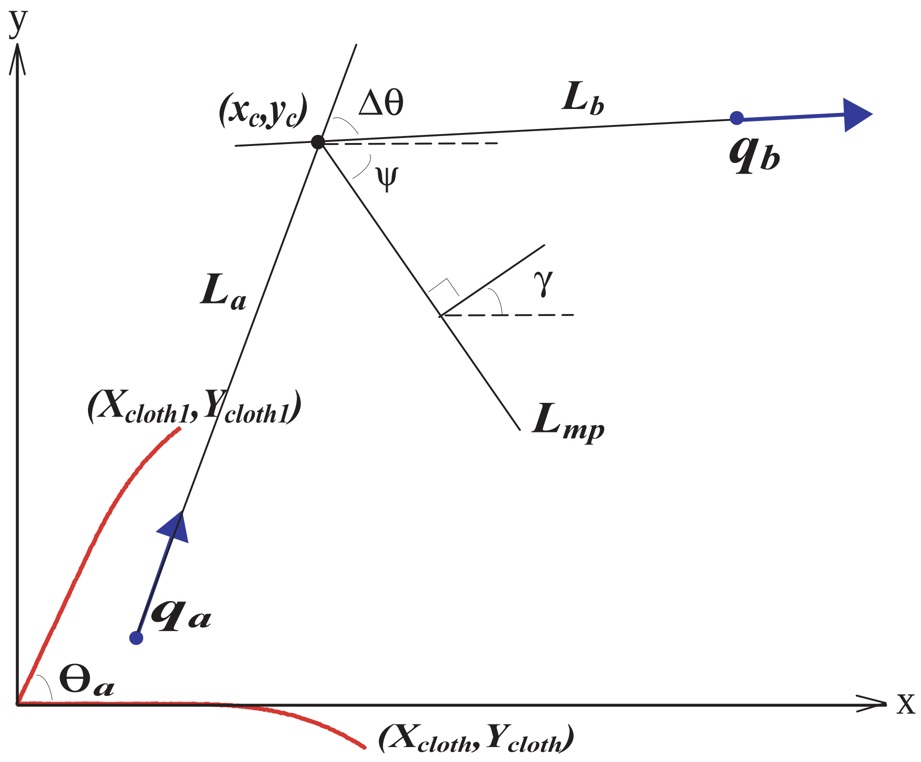


Figure 2.16: CCPP Technique 1: Step 3

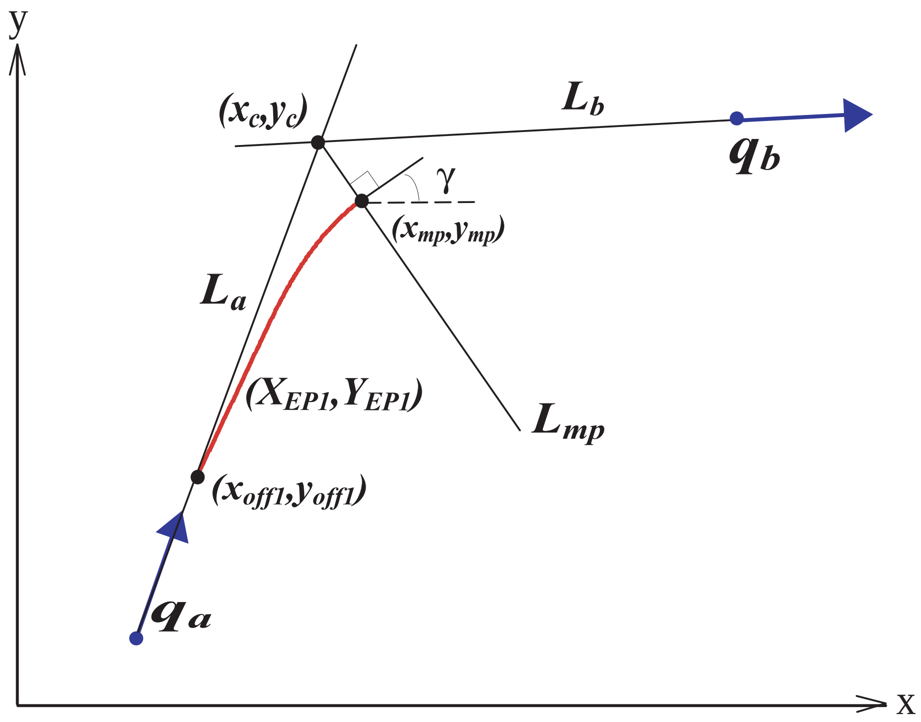


Figure 2.17: CCPP Technique 1: Step 4

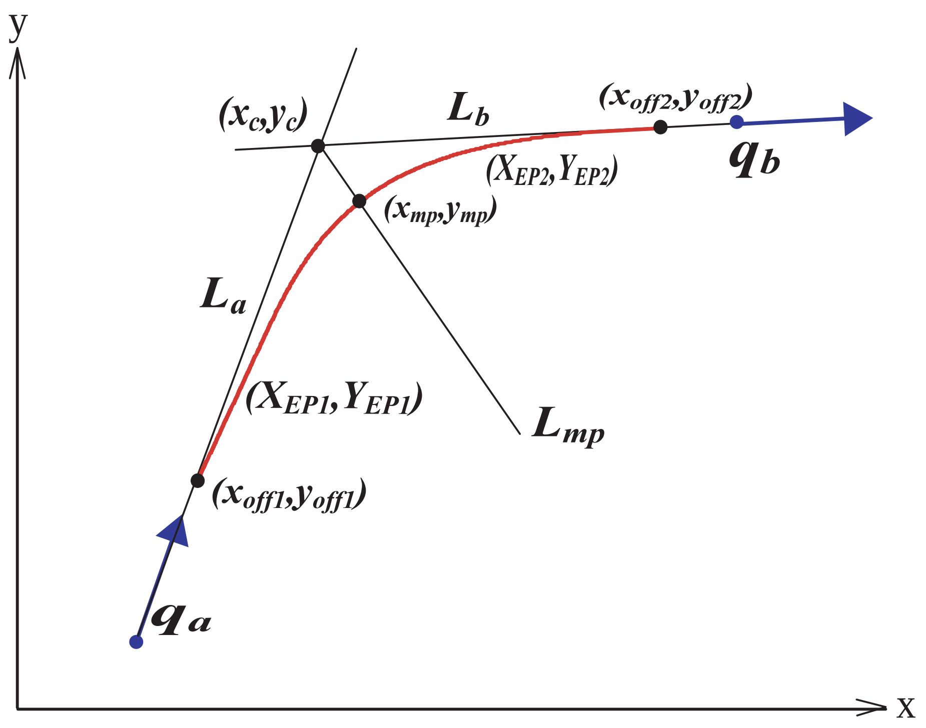


Figure 2.18: CCPP Technique 1: Step 5

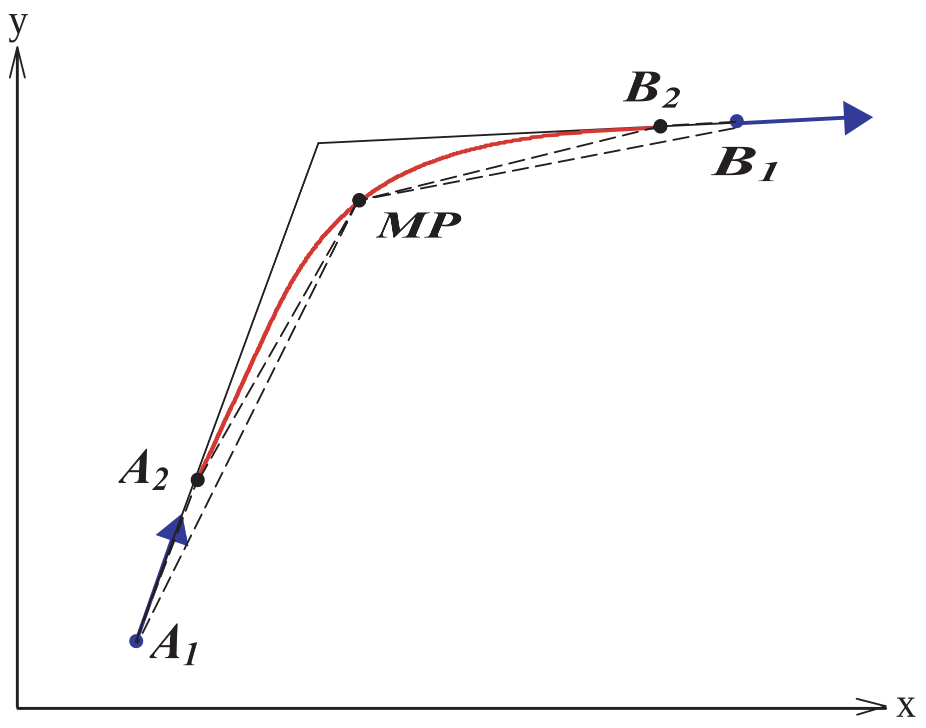


Figure 2.19: CCPP Technique 1: Step 6

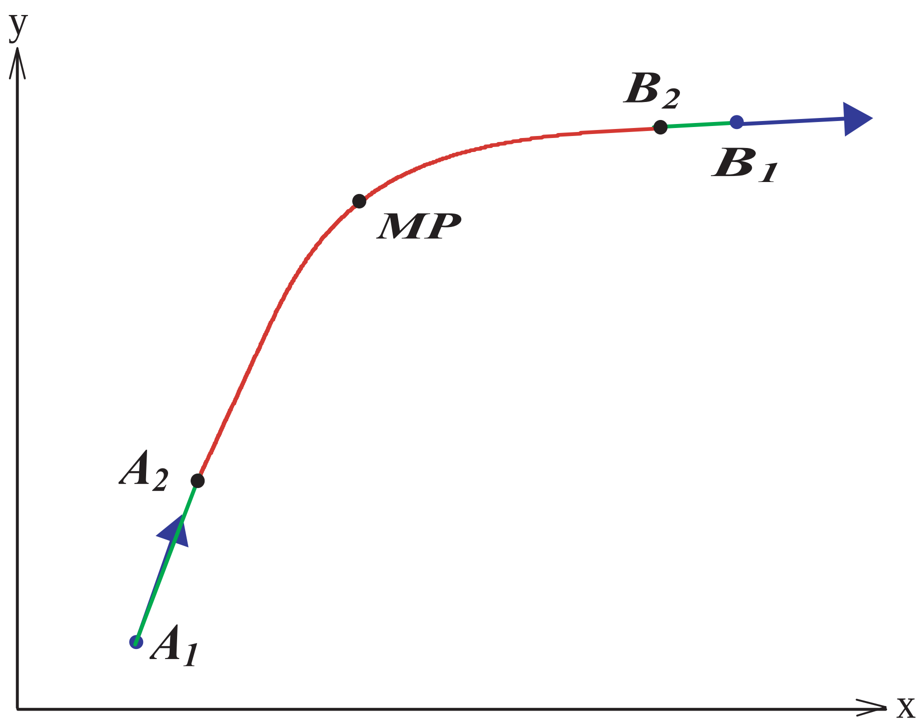


Figure 2.20: CCPP Technique 1: Step 7

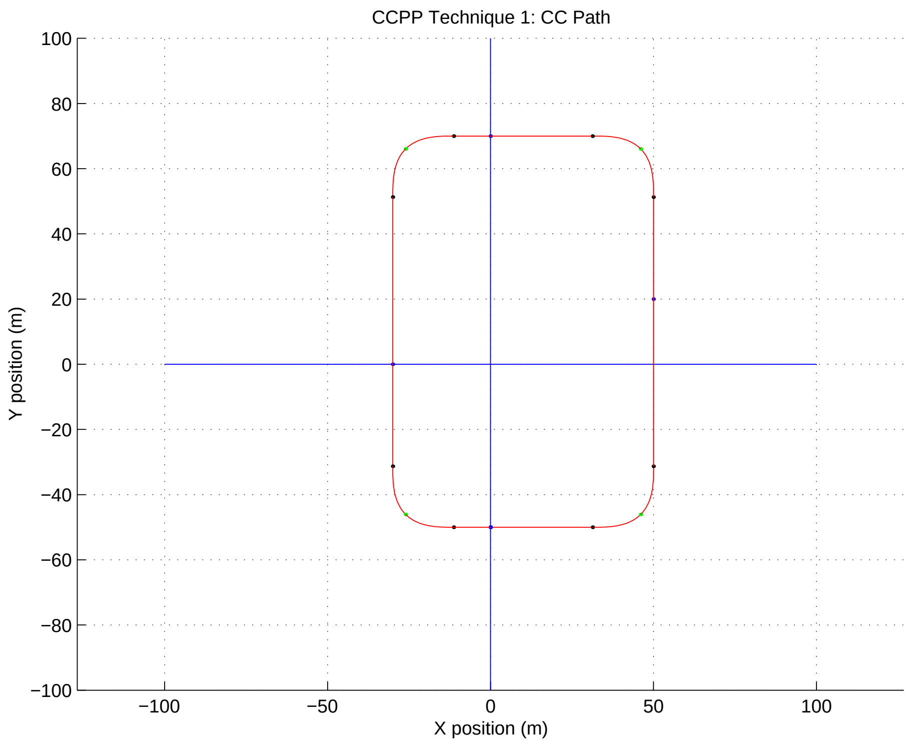


Figure 2.21: CCPP Technique 1: Sample Trajectory 1

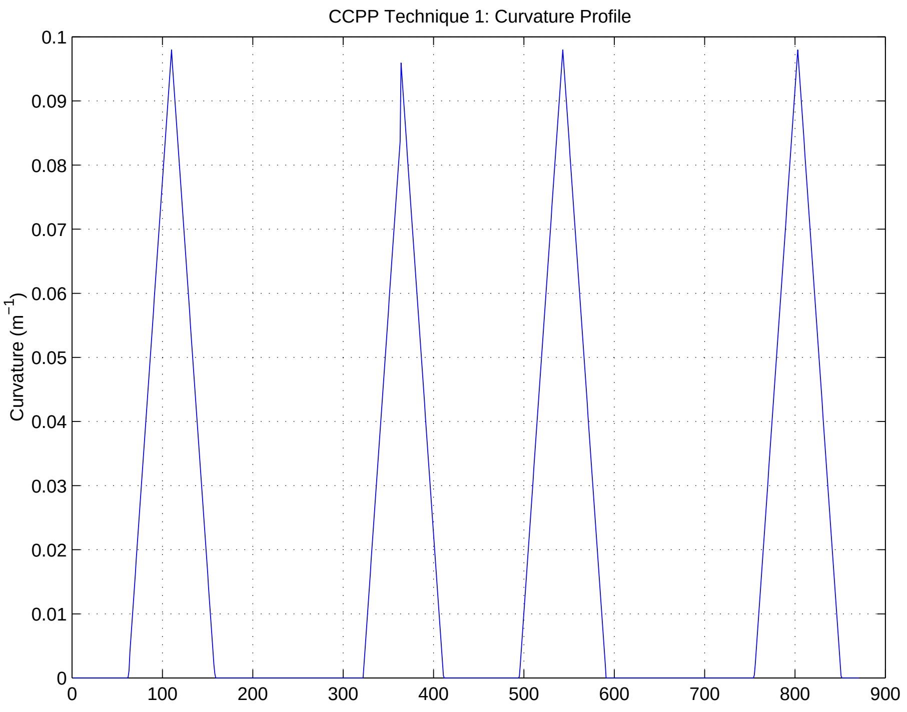


Figure 2.22: CCPP Technique 1: Sample Trajectory 1 - Curvature Profile

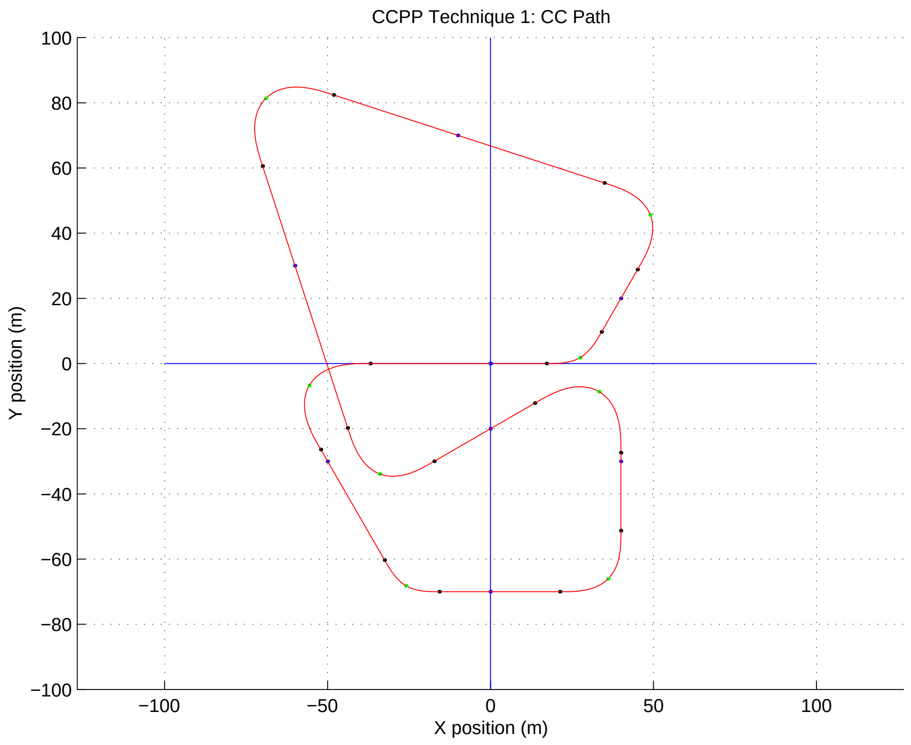


Figure 2.23: CCPP Technique 1: Sample Trajectory 2

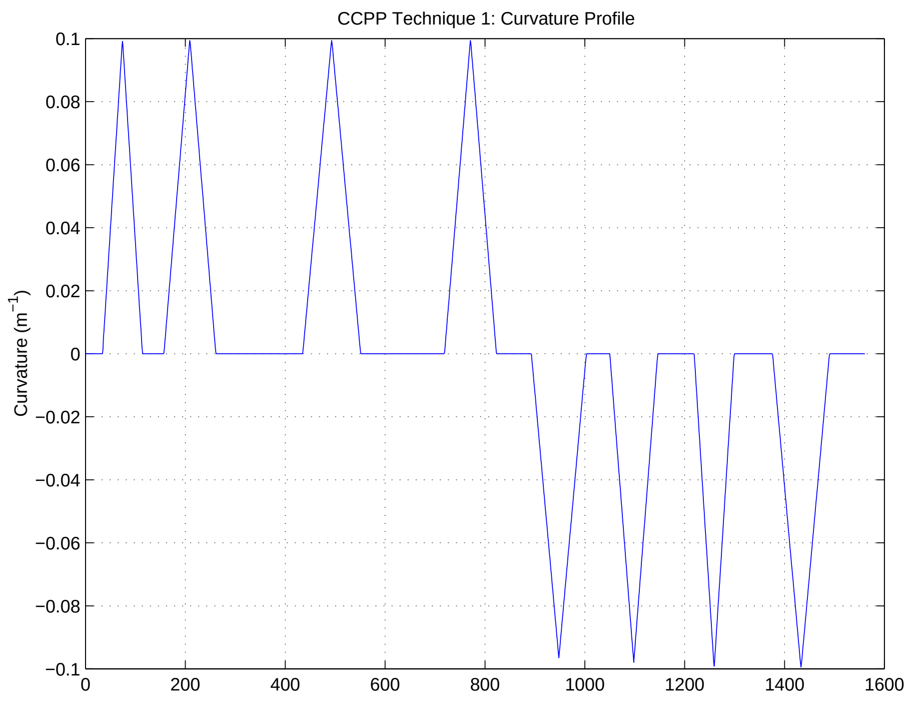


Figure 2.24: CCPP Technique 1: Sample Trajectory 2 - Curvature Profile

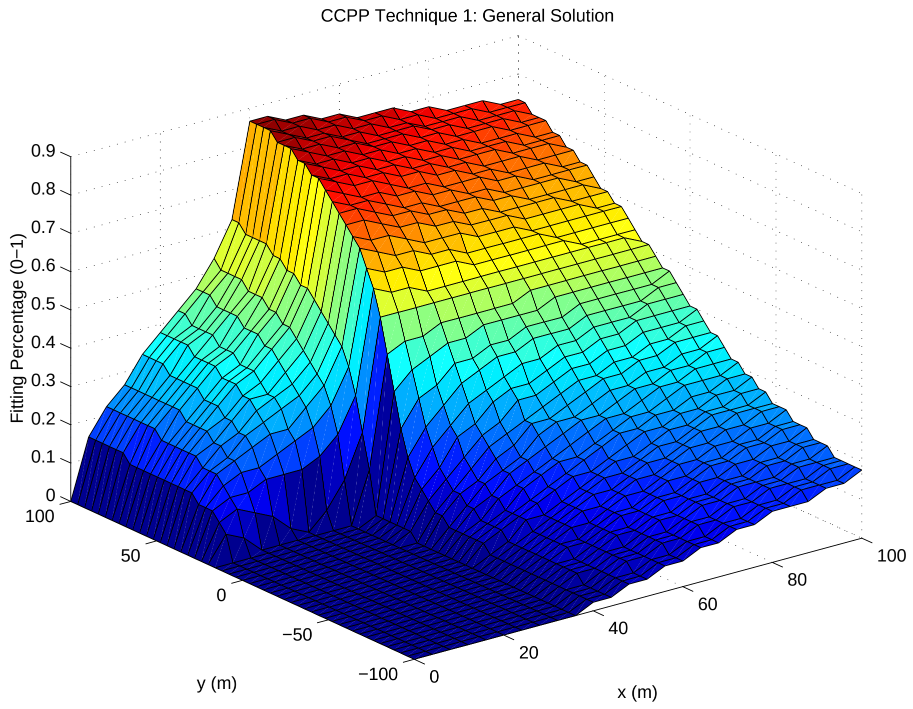


Figure 2.25: CCPP Technique 1: General Solution - Accessibility Surface

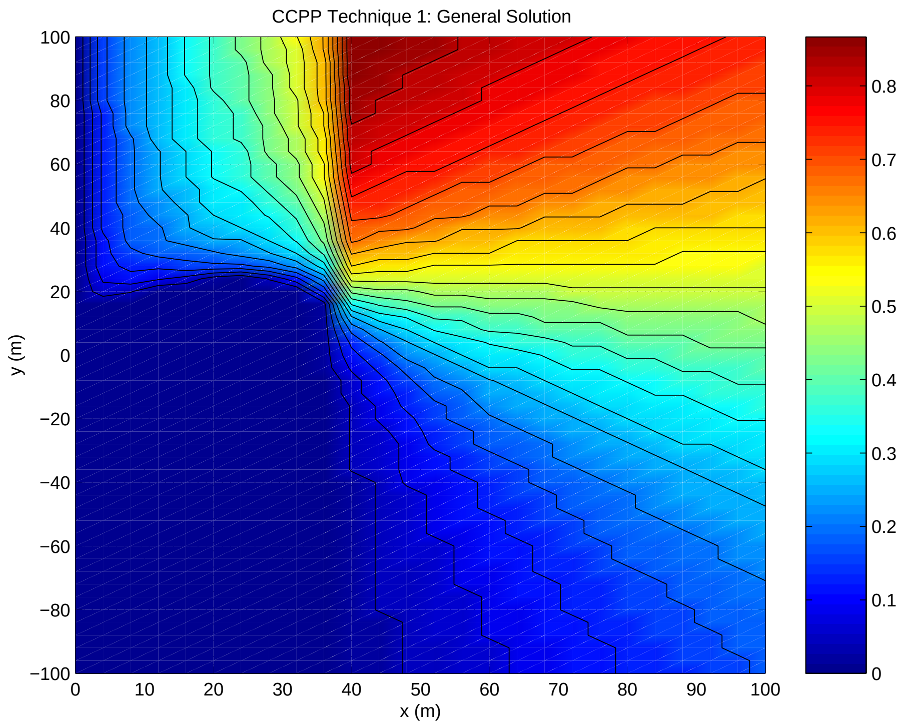


Figure 2.26: CCPP Technique 1: General Solution - Accessibility Contour Plot

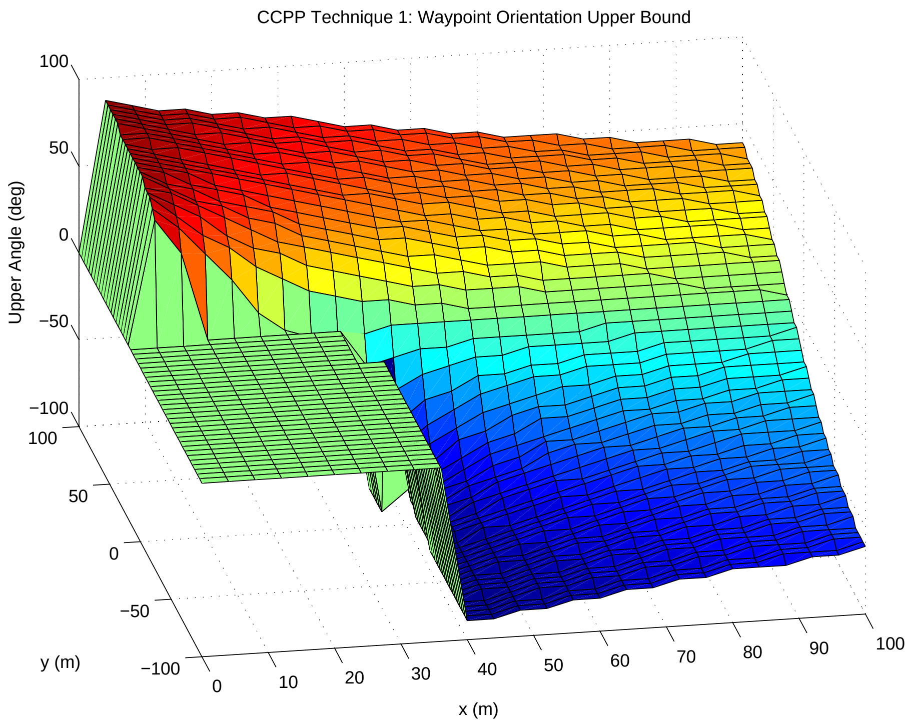


Figure 2.27: CCPP Technique 1: General Solution - Upper Orientation Bounds

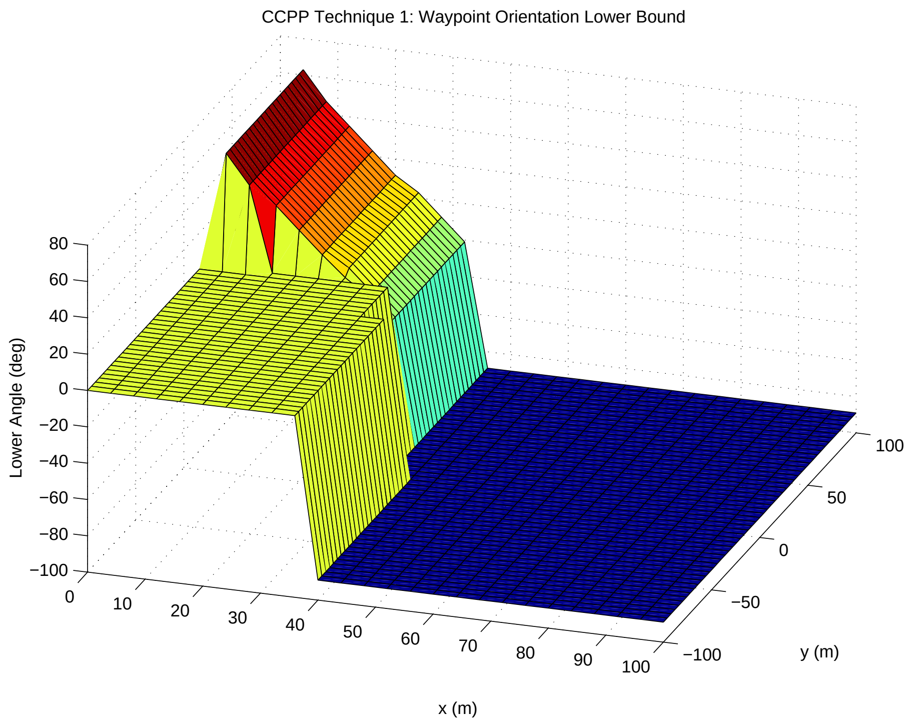


Figure 2.28: CCPP Technique 1: General Solution - Lower Orientation Bounds

for this planning technique, each elementary path reaches a curvature of k_{max}. The paths are constructed in such a way that clothoid segments tend to be short, with the line segments dominating the overall path. In section 2.5, a CCPP technique for creating paths with more flexible and gradual cornering is developed.

Finally, to obtain a clearer picture of the 'accessible region' for this CCPP technique the general solution[32] was generated about the following pose $(0, 0, \pi/2)$ for a 100x200m region. At each point in the region (at a 5m resolution), the path generation technique was performed for an orientation between -90 deg and 90 deg (at a 5 deg resolution) to determine if a feasible path existed for this particular configuration. The result is a 3D accessibility surface as shown in Figure 2.25[33] which shows the percentage of feasible orientations at each point in the region. The figure highlights the existence of inaccessible regions where a waypoint cannot be placed - this is used in the development of the GUI in Chapter 5. Figures 2.27 and 2.28 show the upper and lower orientation bounds for the general solution, indicating graphically that the relative positions/poses of consecutive waypoints strongly affect the path segment generated.

[32]Despite investigation by the author and a thorough review of the relevant literature, it would appear that no analytic general solution exists for either 'CCPP Technique 1 or 2'.

[33]Figure 2.26 presents the same information as a contour plot.

2.5 CCPP: Planning Technique 2

In response to some of the limitations of the first path planning technique, a second (more flexible) planning technique was developed. This approach attempts to maximise the clothoid coefficient a, i.e. it decreases the trajectory sharpness σ, between consecutive waypoints so as to create a shorter, more 'natural' looking path profile, whilst still maintaining continuous path curvature. Where required, and if possible, this technique is able to generate smooth Reeds and Shepp paths to connect waypoints.

The planning problem to be solved remains unchanged. Given a series of non-symmetrical (i.e. equality 2.23 is not true), posture specified waypoints $\mathcal{WP}_{cc}$ as defined in section 2.3.2, a feasible path that joins these points and meets the three path planning criteria is required.

Firstly, let q_a and q_b be two consecutive non-symmetric waypoints between which a continuous curvature path must be placed. The initial steps of 'Technique 2' are identical to those previously described for 'Technique 1', so they are only briefly outlined here:

- The equation of the line passing through each waypoint, $\mathcal{L}_a$ and $\mathcal{L}_b$ is calculated as per equation set 2.24.
- The orientation change $\Delta\theta$ (equation 2.25) is required.
- Using equation 2.26, the intersection point (x_c, y_c) between $\mathcal{L}_a$ and $\mathcal{L}_b$ is calculated.
- The orientation ψ of the path segment midpoint/bisector is calculated using equation 2.27, then the bisector line equation $\mathcal{L}_{mp}$ is generated from equation 2.28.
- The angle γ, perpendicular to $\mathcal{L}_{mp}$ (orientation at elementary path midpoint) is calculated using equation 2.29.

Having performed the above calculations, the elementary path equality (eq. 2.23) is used to determine whether the elementary path segment of least admissible sharpness σ (for this particular configuration) starts at q_a or ends at q_b[34]. Since these are

[34]The elementary path of minimum allowable sharpness must have either q_a or q_b as one of its extremities.

(usually) non-symmetric points, an elementary path cannot be fitted directly between them, hence an intermediate (start or end) point is required. The location of intermediate point 1 (x_{int_1}, y_{int_1}) is found by first assuming that q_a is one extremity of an elementary path while the other extremity of the path lies at some point (x_{int_1}, y_{int_1}) on $\mathcal{L}_b$. This intermediate point will satisfy two equations: 1) the line equation $\mathcal{L}_b$ and 2) the symmetry equality for an elementary path (eq. 2.23). Hence, (x_{int_1}, y_{int_1}) can be found by the simultaneous solution of these two linear equations:

$$\begin{bmatrix} x_{int_1} \\ y_{int_1} \end{bmatrix} = \begin{bmatrix} tan\theta_b & -1 \\ \sin\left(\frac{\theta_b+\theta_a}{2}\right) & -\cos\left(\frac{\theta_b+\theta_a}{2}\right) \end{bmatrix}^{-1} \begin{bmatrix} y_b - (\tan\theta_b)x_b \\ x_a \sin\left(\frac{\theta_b+\theta_a}{2}\right) - y_a \cos\left(\frac{\theta_b+\theta_a}{2}\right) \end{bmatrix} \tag{2.39}$$

A geometrical check is now performed to determine if intermediate point 1 lies on the correct side of waypoint q_b to allow for the formation of an elementary path between q_a and q_b, where q_a is the starting point. For a fit to occur, intermediate point 1, IP_1 (x_{int_1}, y_{int_1}) must lie between the second waypoint WP_2 (x_b, y_b) and the intersection point IS (x_c, y_c)[35]. In other words, the following two conditions must be met:

$$Condition \quad 1: \quad \overrightarrow{WP_2IS} \geqslant \overrightarrow{IP_1IS} \tag{2.40}$$

$$Condition \quad 2: \quad \overrightarrow{WP_2IS} \geqslant \overrightarrow{IP_1WP_2} \tag{2.41}$$

If either of these conditions are not met, then the other possible elementary path configuration is selected, where q_b is its end point and its complimentary symmetric point, intermediate point 2 (x_{int_2}, y_{int_2}), lies on $\mathcal{L}_a$. Intermediate point 2 can be found using equation 2.39, by 1) replacing the $\mathcal{L}_b$ coefficients with those of $\mathcal{L}_a$ and 2) swapping (x_a, y_a) with (x_b, y_b). The start and end points of the curved path segment have now been determined.

Using equation 2.30, the clothoid parameter for the elementary path segments t_{ep} can be calculated (being a function of the orientation change $\Delta\theta$). The coordinates of the end points of the unit clothoid segment[36] $(a = 1)$ of length t_{ep}, x_{end} and y_{end} are

[35]Note: these three points all lie on the line $\mathcal{L}_b$.

[36]A unit clothoid segment is selected since the clothoid coefficient a is presently unknown - the fitting technique calculates a value for it later on.

calculated using equations 2.10 and 2.11[37] as part of the trajectory fitting process. The starting point, x_{start} and y_{start}, of the first clothoid segment on the elementary path is also required. The starting point is assigned in the following way:

$$\begin{aligned}
&if \quad intermdiate \quad point \quad one \quad met \quad conditions \quad 2.40 \quad and \quad 2.41 \\
&\qquad \begin{bmatrix} x_{start} \\ y_{start} \end{bmatrix} = \begin{bmatrix} x_a \\ y_a \end{bmatrix} \\
&otherwise \\
&\qquad \begin{bmatrix} x_{start} \\ y_{start} \end{bmatrix} = \begin{bmatrix} x_{int_2} \\ y_{int_2} \end{bmatrix}
\end{aligned} \tag{2.42}$$

Now, the value of the clothoid coefficient a that joins the unit clothoid segment originating at (x_{start}, y_{start}) (rotated to orientation θ_a) to the line $\mathcal{L}_{mp}$ is given by the following equation[38]:

$$a = \frac{-\tan\psi x_{start} + y_{start} - y_c + \tan\psi x_c}{\tan\psi(\cos\theta_a x_{end} - \sin\theta_a y_{end}) - (\sin\theta_a x_{end} + \cos\theta_a y_{end})} \tag{2.43}$$

Several checks may be performed to determine if an elementary path containing clothoid segments of sharpness $\sigma = \frac{1}{a^2}$ is feasible for $\mathcal{A}$. Firstly, one can check if $\sigma \leqslant \sigma_{max}$, i.e. is the sharpness of the path achievable by $\mathcal{A}$. If σ exceeds the allowable limit, then no feasible path exists between the waypoints q_a and q_b, since this technique generates a trajectory with the minimum allowable sharpness. A second check may also be made to establish whether the maximum allowable curvature κ_{max} is exceeded at the elementary path midpoint. This is performed (using eq. 4.18) by checking whether $\kappa = \frac{t_{ep}}{a} \leqslant \kappa_{max}$. If this curvature limit is also not exceeded, then it is possible to form a path between q_a and q_b using an elementary path.

However, if κ_{max} is exceeded, then it may still be possible to fit a smooth Reeds and Sheep path between the two symmetric points. Here, it is necessary to solve for the

[37] y_{end} must be multiplied by $sign(\Delta\theta)$

[38] Equation 2.43 uses the condition that the midpoint of an elementary path must lie on the bisecting line $\mathcal{L}_{mp}$. It was derived by solving for the clothoid coefficient a which means that the end points of the unit clothoid segment (x_{end}, y_{end}) (generated previously) when: 1) scaled up by a, 2) rotated to an orientation of θ_a and 3) offset by (x_{start}, y_{start}) lie on the line $\mathcal{L}_{mp}$.

parameters that define a clothoid arc which increases the curvature to κ_{max}, while a circular arc of maximum allowable curvature completes the path and the required orientation change on the midpoint line $\mathcal{L}_{mp}$. To define such a smooth Reeds and Shepp path between a point (x_{start}, y_{start}) and a line $\mathcal{L}_{mp}$ for a known orientation change, there are two variables which completely define the solution - the clothoid coefficient a and the clothoid parameter t_{cl}. These variables are found by solving two nonlinear equations. The first (eqn. 2.44) results from the condition that the clothoid segment must reach κ_{max}, while the second (eqn. 2.45) arises from the condition that the path segment endpoint $(x_{RS_{end}}, y_{RS_{end}})$ lies on $\mathcal{L}_{mp}$. The two non linear equations[39] to be solved simultaneously are given below:

$$k_{max} - \frac{t_{cl}}{a} = 0 \tag{2.44}$$

$$\tan\psi x_{RS_{end}} - y_{RS_{end}} + y_c - (\tan\psi)x_c = 0 \tag{2.45}$$

where (x_{inter}, y_{inter}) is the transition point between the clothoid and circular arcs, λ is the orientation of the path at the start of the circular segment, $(x_{circ_{end}}, y_{circ_{end}})$ is the endpoint of the unrotated circular arc segment and $(x_{cl_{end}}, y_{cl_{end}})$ is the endpoint of the unrotated clothoid segment of parameter length t_{cl}. Note that this last point is found using equations 2.10 and 2.11. The other points mentioned are defined as follows and are evaluated at each iteration of the equation solver:

$$x_{RS_{end}} = x_{inter} + (\cos\lambda)x_{circ_{end}} - (\sin\lambda)y_{circ_{end}} \tag{2.46}$$

$$y_{RS_{end}} = y_{inter} + (\cos\lambda)x_{circ_{end}} + (\sin\lambda)y_{circ_{end}} \tag{2.47}$$

$$x_{inter} = x_{start} + \cos\theta_a x_{cl_{end}} - \sin\theta_a y_{cl_{end}} \tag{2.48}$$

$$y_{inter} = y_{start} + \sin\theta_a x_{cl_{end}} + \cos\theta_a y_{cl_{end}} \tag{2.49}$$

$$x_{circ_{end}} = \kappa_{max}\cos((\gamma - \theta_a) - \frac{1}{2}(t_{cl}^2 - \pi)) \tag{2.50}$$

$$y_{circ_{end}} = sign(\Delta\theta)\kappa_{max}\sin((\gamma - \theta_a) - \frac{1}{2}(t_{cl}^2 - \pi) + 1) \tag{2.51}$$

$$\lambda = \theta_a - sign(\Delta\theta)\frac{t_{cl}^2}{2} \tag{2.52}$$

[39]Equations 2.44 and 2.45 were solved using MATLAB's non-linear equation solver 'fsolve'. An initial guess for a and t_{cl} must be provided to the solver.

If the value of a as determined from the solution of these equations is found to meet the condition $\sigma \geqslant \sigma_{max}$, then a feasible smooth Reeds and Shepp path exists between q_a and q_b. If not, then neither an elementary path nor a smooth Reeds and Shepp path can connect q_a and q_b, and hence a new waypoint configuration is required.

Once a feasible path has been identified, the path is generated in the same manner as outlined previously for 'Technique 1' Firstly, the basic (X, Y) path vectors are created using the curve parameters, then these are fitted using rotate and offset operations. The continuous curvature path segments are joined using line segments. Note that here only one line segment is required per pair of waypoints, unlike 'Technique 1' which required two line segments, since curved path segment begins at one the waypoints.

CCPP technique 2 is demonstrated for the same (two) waypoint sets used in section 2.4, the resulting paths and curvature profiles are displayed in Figures 2.29, 2.30, 2.31 and 2.32[40]. Visually, the trajectories generated appear shorter in length and more 'natural', i.e. the curved sections are longer and less sharp. This is verified by the curvature profiles which show elementary paths of varying sharpness and maximum curvature[41]. Additionally, a trajectory containing two smooth Reeds and Shepp path

[40]The values for σ_{max} and κ_{max} used in section 2.4 are also used here.
[41]Note: the length of straight segments is also reduced, as there is only one per curved path segment.

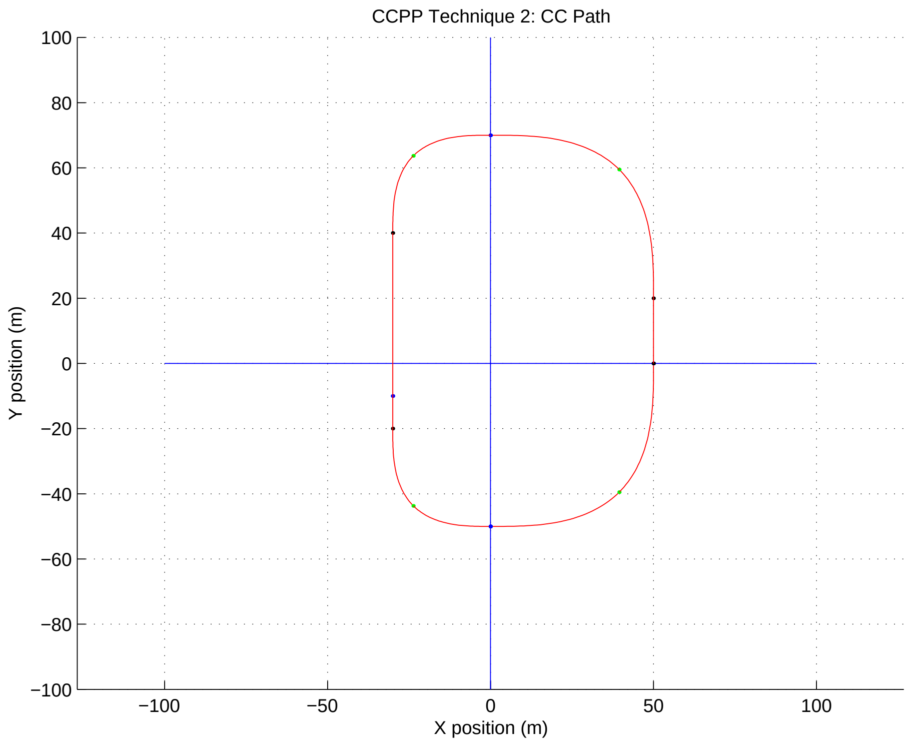


Figure 2.29: CCPP Technique 2: Sample Trajectory 1

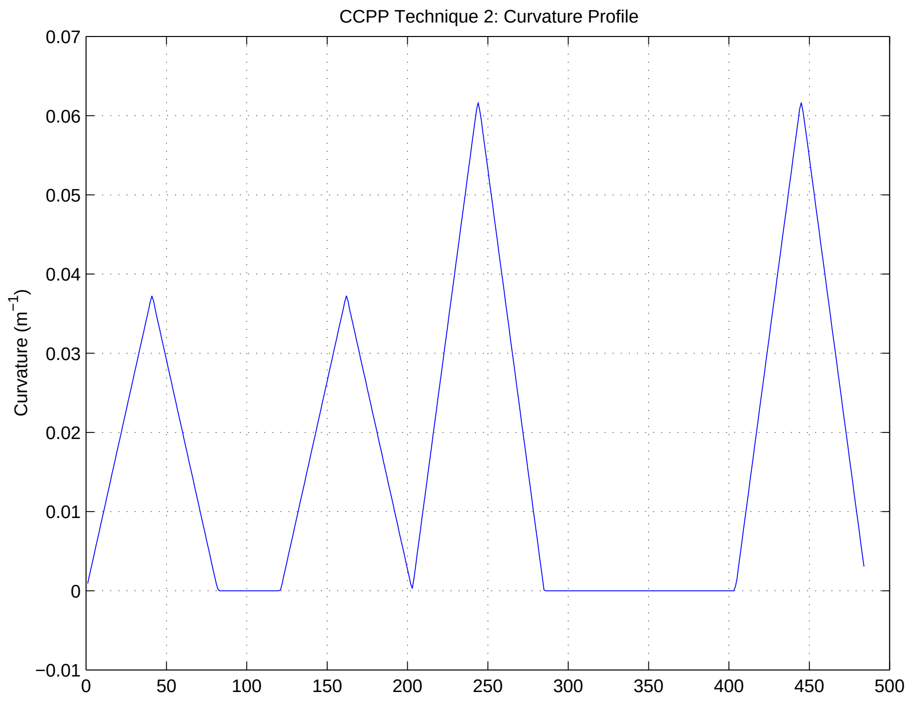


Figure 2.30: CCPP Technique 2: Sample Trajectory 1 - Curvature Profile

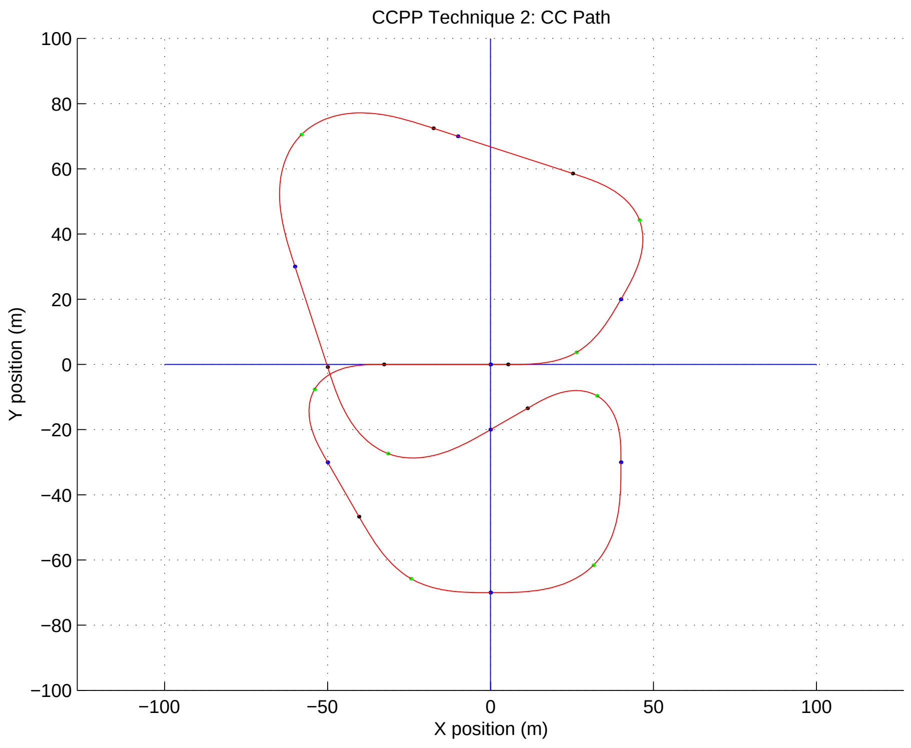


Figure 2.31: CCPP Technique 2: Sample Trajectory 2

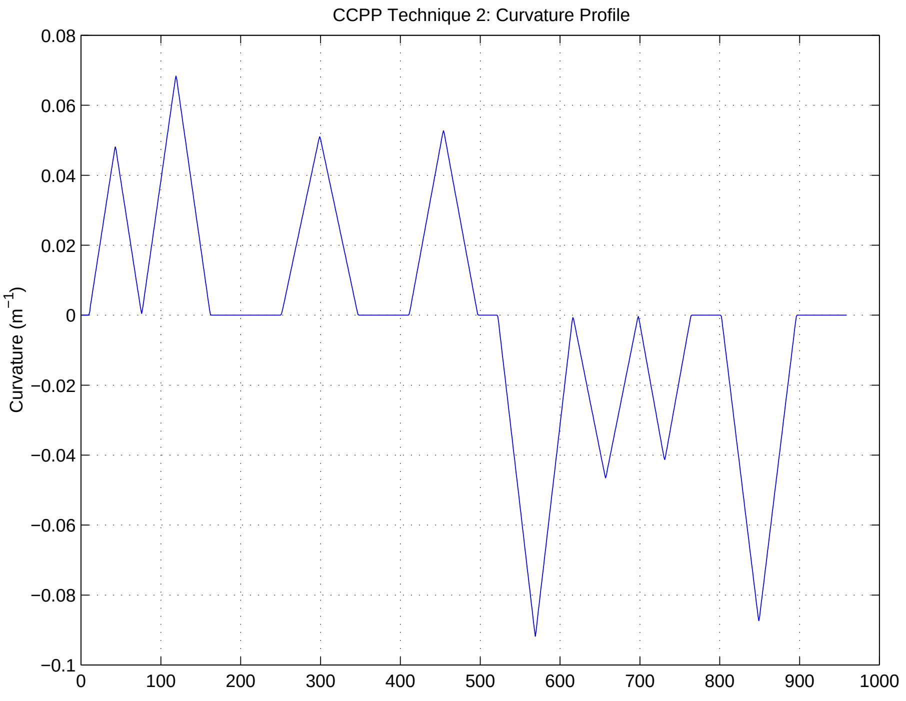


Figure 2.32: CCPP Technique 2: Sample Trajectory 2 - Curvature Profile

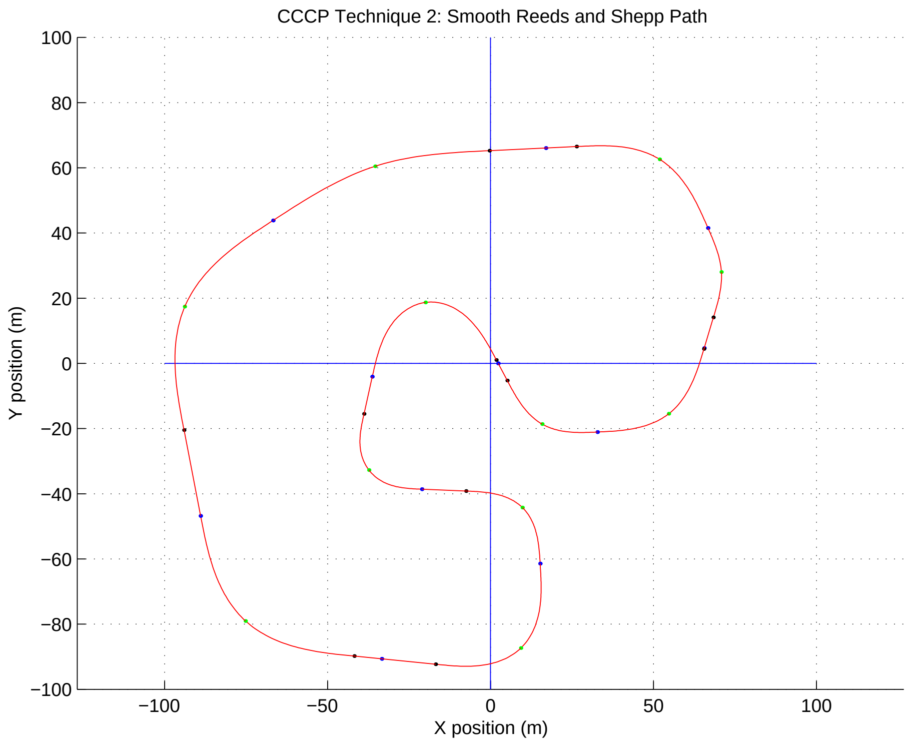


Figure 2.33: CCPP Technique 2: Smooth Reeds and Shepp Path

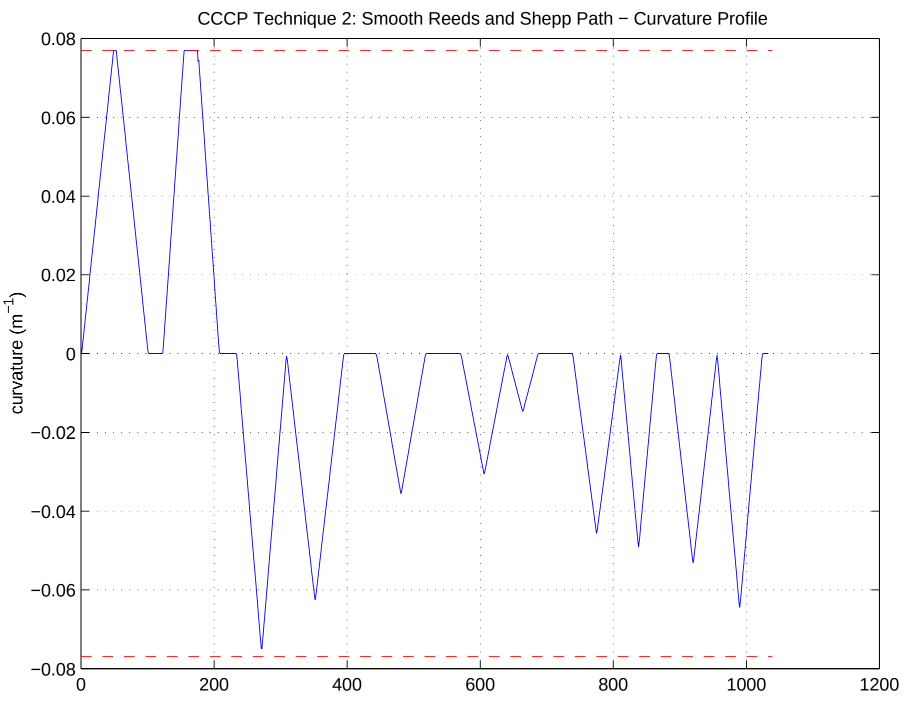


Figure 2.34: CCPP Technique 2: Smooth Reeds and Shepp Path - Curvature Profile

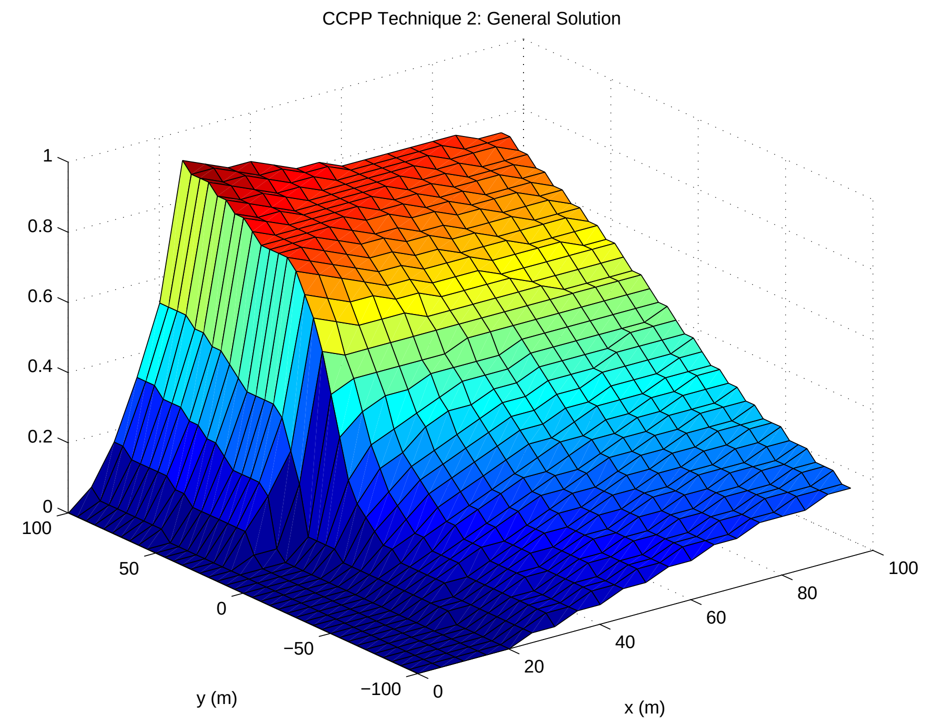


Figure 2.35: CCPP Technique 2: General Solution - Accessibility Surface

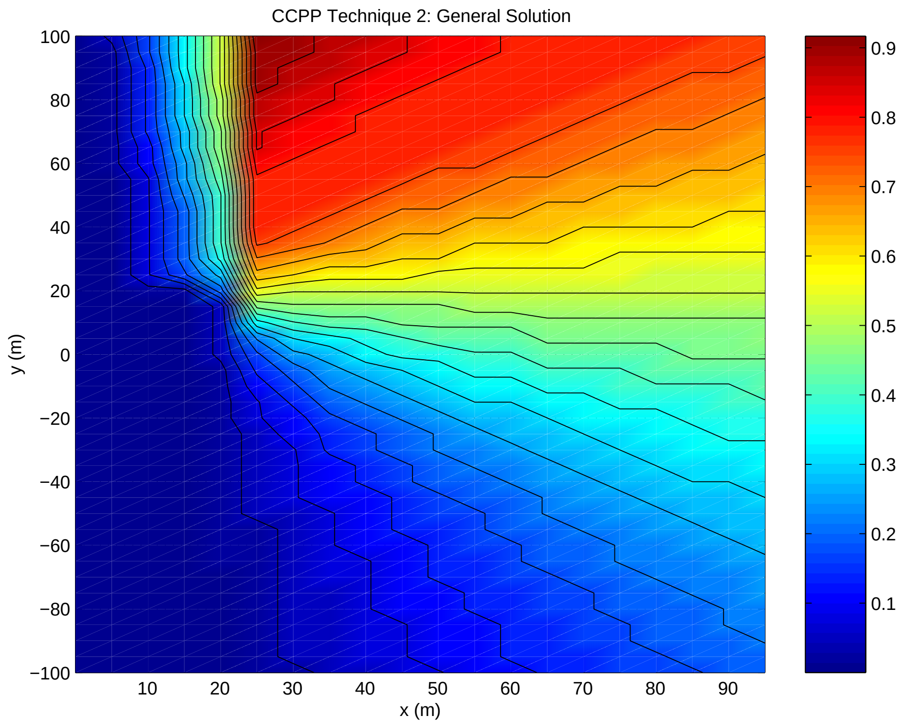


Figure 2.36: CCPP Technique 2: General Solution - Accessibility Contour Plot

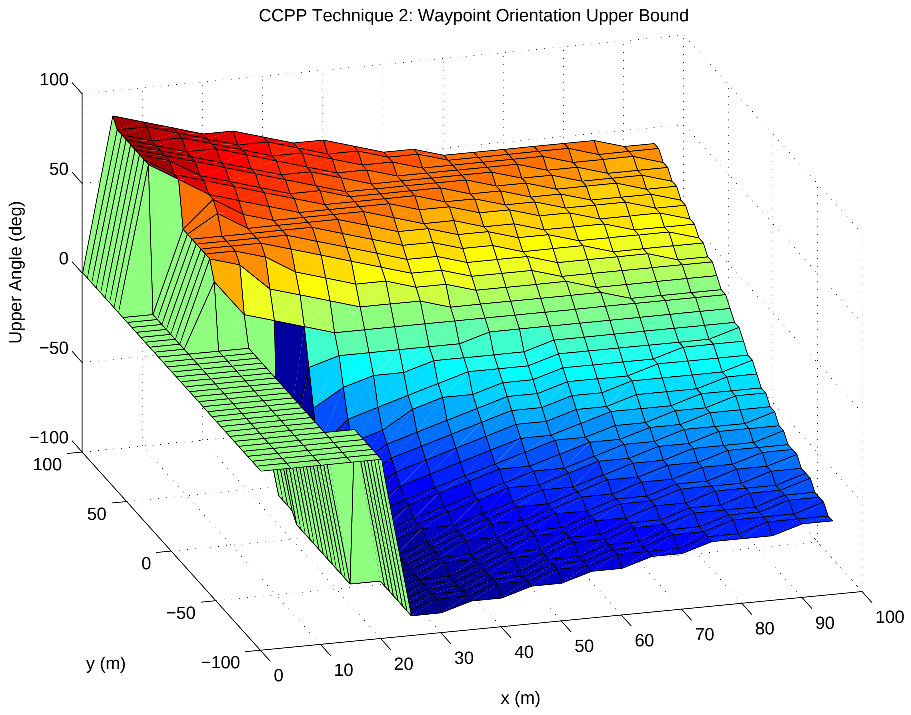


Figure 2.37: CCPP Technique 2: General Solution - Upper Orientation Bounds

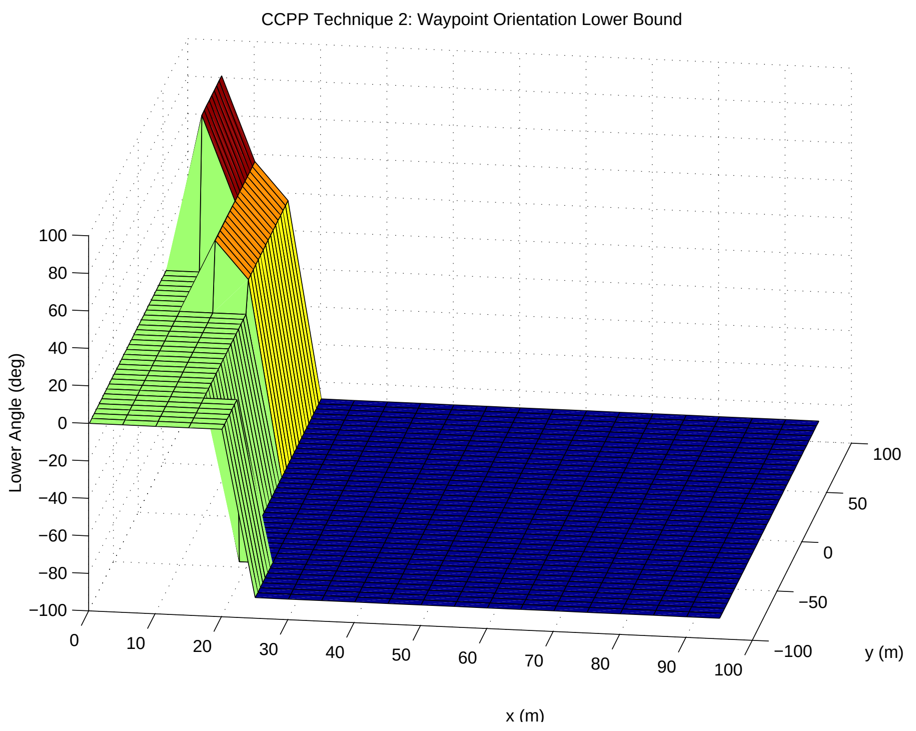


Figure 2.38: CCPP Technique 2: General Solution - Lower Orientation Bounds

segments (first two curves) has been generated, its path and curvature profiles being plotted in Figures 2.33 and 2.34, respectively.

For use in the user interface, the general solution was once again calculated for an initial pose $(0, 0, \pi/2)$ over the same 100x200m region with resolutions of 5m (area) and 5 deg (orientation). The resulting 'accessible' space is similar to that seen for 'Technique 1' - with the four general solution plots being shown in Figures 2.35, 2.36, 2.37 and 2.38.

2.6 Technique Evaluation

The two techniques described in this chapter both provide a solution to the path planning problem (where a path is specified by a system user as a series of pose waypoints) for a car-like robot. Given two consecutive waypoints both can generate a path that meets the three path planning criteria. As highlighted by the general solution results, not all positions/poses are reachable from an initial pose. The orientation of any waypoint is limited to $\pm(0 - 180)$ deg around the previous waypoint's orientation[42]

[42]The side of the orientation line (of the previous waypoint) on which the current waypoint lies determines the sign of the orientation bounds.

Sample Trajectory	**Technique 1 Path Length (m)**	**Technique 2 Path Length (m)**	**Length Difference (%)**
1	375.3	348.2	-7.2
2	655.1	605.0	-7.6

Table 2.1: Comparison of CCPP techniques

otherwise an incorrect path solution could result. If a waypoint's orientation is within these bounds, the checks performed inside the path generation techniques will identify any infeasible waypoint configurations.

Based upon the results presented in sections 2.4 and 2.5 it appears that 'Technique 2' is indeed the CCPP technique of choice. It offers greater path generation flexibility, a smoother path, is computationally more expedient, in addition to producing a significantly shorter trajectory as the results in Table 2.6 demonstrate. Hence 'Technique 2' is used for path generation by the system path entry interface developed in Chapter 5.

2.7 Conclusion

In this chapter, car-like robots and the motion constraints (nonholonomic) imposed upon them were introduced. The methods for specifying physically trackable, i.e. continuous curvature, paths were also explained. Two techniques were developed (and demonstrated) for the implementation of continuous curvature path planning (CCPP), the latter of these ('Technique 2') being identified as the superior. Since no general analytical solution appears to exist for determining an 'accessible region' for a new waypoint, areas of accessibility were, therefore, determined numerically (over a realistically fine grid) to provide the inaccessible region used in the system GUI developed in Chapter 5.

Chapter 3

Obstacle Avoidance

3.1 Introduction

In the previous chapter, the issue of path planning for a car-like robot was investigated. A continuous curvature path generation technique was developed for constructing a physically trackable path from a series of user specified waypoints. However, the path is generated without any prior knowledge of the environment and any obstacles which may be present. Therefore, this system, like any practical autonomous vehicle implementation, requires a means of perceiving its local environment - this is achieved through two core abilities:

- obstacle detection and characterisation
- obstacle avoidance and path replanning

Obstacle detection and characterisation is the focus of the work presented in the thesis of Lee (2001). The software module developed by Lee provides the path planner and tracker (see Figure 3.1) with the location and size information of obstacles infringing upon a desired trajectory.

This chapter is primarily concerned with using such obstacle detection to replan the path (i.e. to take evasive action), so as to ensure that static obstacles are successfully avoided[1]. Firstly, an overview of obstacle avoidance strategies are presented, followed

[1]An obstacle avoidance scheme where avoidance maneuvers are taken as obstacles are detected is known as reactionary navigation.

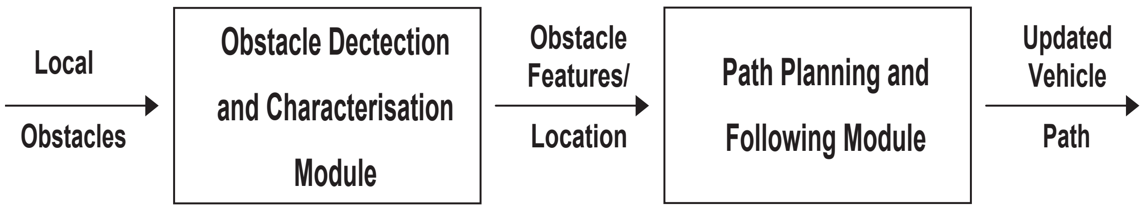

Local
Obstacles
Obstacle Dectection and Characterisation Module
Obstacle Features/
Location
Path Planning and Following Module
Updated Vehicle
Path

Figure 3.1: Structure of the obstacle avoidance system

by a brief outline of Lee's obstacle detection module. Next, two 'shortest path' algorithms are detailed prior to the development of a technique for replanning a trajectory while maintaining its continuous curvature profile.

3.2 Background and Critical Literature Review

In the literature, there exists a large number of published techniques for solving the mobile robot path planning problem in an environment where the obstacles are static and their configurations known. In many situations, these methods are readily applicable to performing obstacle avoidance in previously unknown environments. Three broad obstacle avoidance approaches are described in Latombe (1991), and are briefly outlined in the following subsections[2].

3.2.1 Roadmap Methods

The roadmap approach provides a robot with a collection of feasible path segments leading it around static obstacles. The roadmap network, i.e. the set of feasible paths segments, can be created using a number of techniques, including:

- Visibility graphs - the initial and goal configurations, as well as the vertices of all obstacles, are joined to all other visible vertices to build a roadmap. The construction of a visibility graph is shown in Figure 3.2.
- Voronoi diagrams - these create roadmaps that connect the initial and goal configurations by forming paths consisting of line segments and parabolic arcs

[2]As already detailed in Chapters 1 and 2 more complex CCPP/obstacle avoidance methods, using genetic algorithms (Fraichard and Ahuactzin 2001) and probabilistic techniques (Scheuer and Fraichard 1997) have been developed by researchers at various robotics institutes, such as INRIA. However, these are considered as being beyond the scope of the HSV project at the present time.

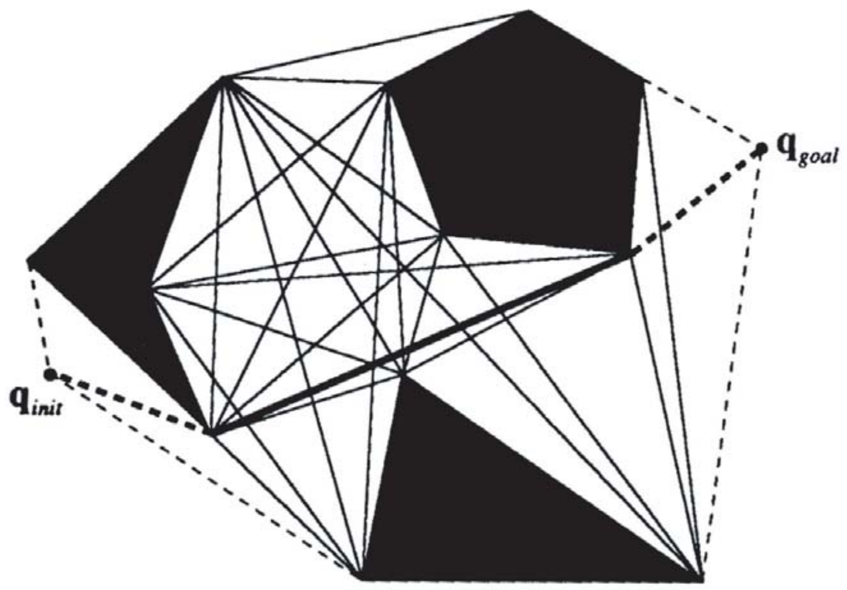


Figure 3.2: A Visibility Graph

(for polygonal obstacles) that maximise the clearance between the robot and the obstacles (see Figure 3.3).

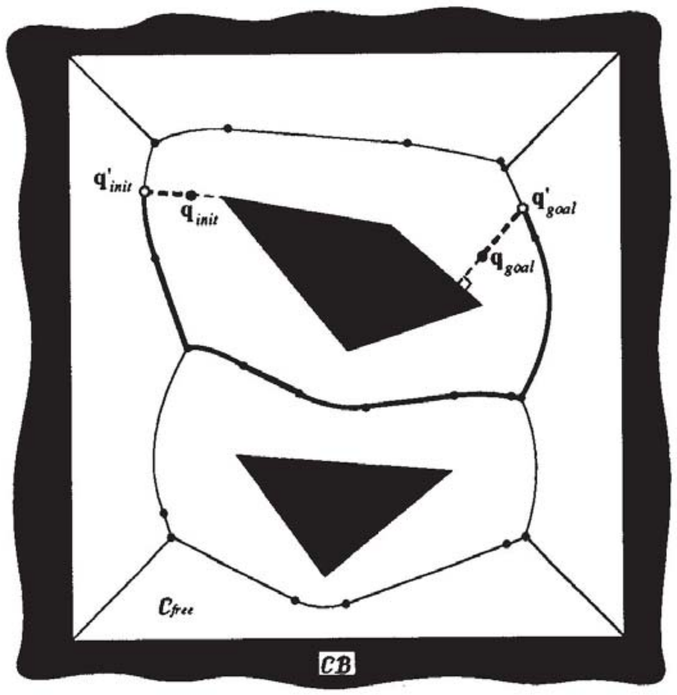


Figure 3.3: A Voronoi Diagram

3.2.2 Cell Decomposition

Cell decomposition methods divide free space into a number of regions known as cells. The connectivity graph, and hence the feasible paths, are built by joining adjacent cells, for example by connecting their midpoints. Two often used implementations of cell decomposition are:

- Exact Cell Decomposition - the free space is divided into trapezoidal and triangular segments using the external and internal obstacle vertices, as demonstrated in Figure 3.4.

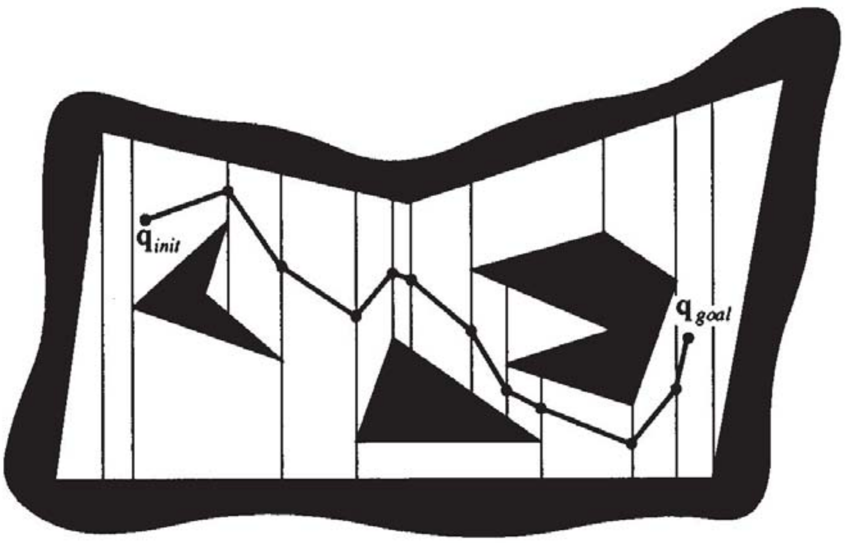


Figure 3.4: Exact Cell Decomposition

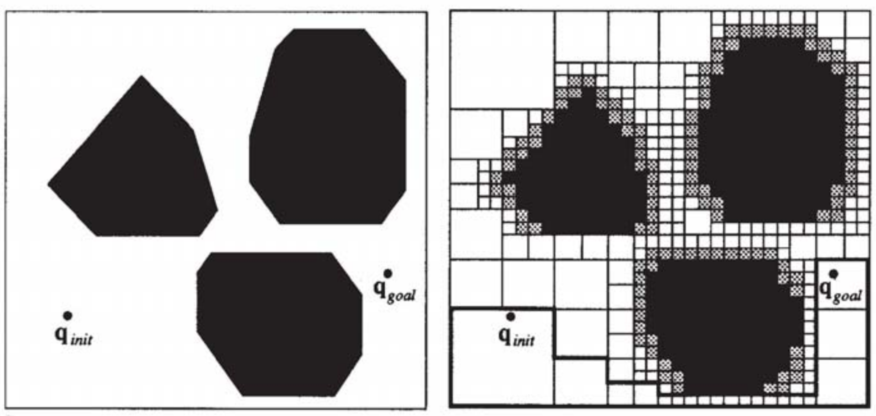


Figure 3.5: Approximate Cell Decomposition

- Approximate Cell Decomposition - also known as a ‘quadtree’ decomposition, breaks down the free space into progressively smaller rectangles (if required[3]) until a predefined resolution is reached (see Figure 3.5).

If a set of viable paths (or ‘channels’) exists, then the shortest one is computed to provide the channel joining the initial and goal configurations from which a path is generated.

3.2.3 Potential Field

A third technique is that of ‘potential fields’, an approach that breaks up the free space into a fine rectangular grid which is then searched for a free path. Each grid element (or cell) is assigned a ‘potential’, where the goal and neighbouring elements are assigned an ‘attractive’ potential and obstacles possess a ‘repulsive’ potential. This ensures that the path created moves towards the goal while steering clear of any obstacles. A simple potential field implementation is shown in Figure 3.6, subfigure (b) shows the equipotential contours and the trajectory taken by the robot which follows the path of least resistance.

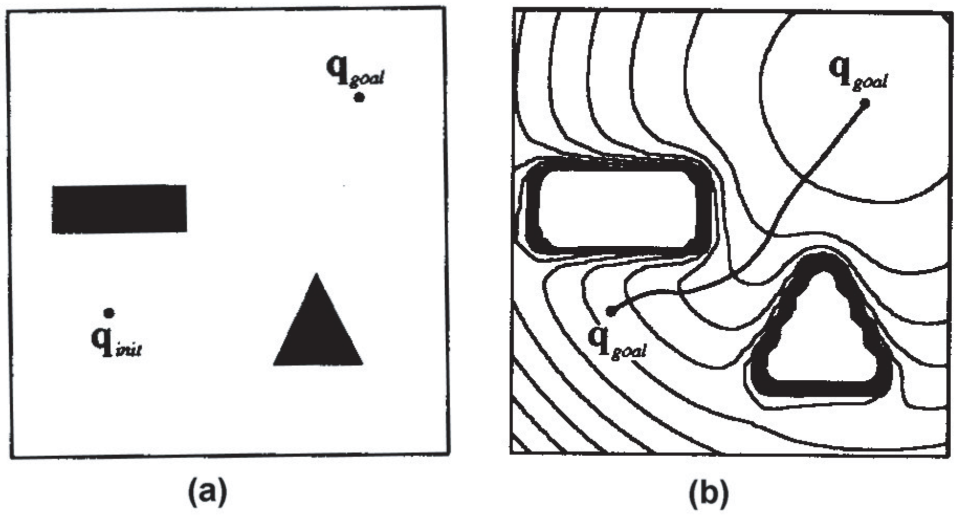
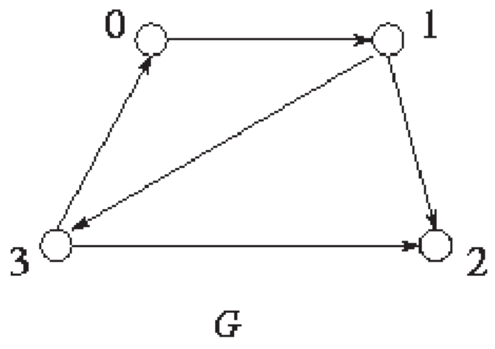


Figure 3.6: Potential Field: (a) Obstacle configuration (b) Path through equipotential contours

[3]Conditions for cell decomposition are detailed in Latombe (1991), page 18.



a) Directed Graph

$$A = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 \end{array}$$

b) Adjacency Matrix

Figure 3.7: A directed graph

3.3 Shortest Path Algorithms

The aforementioned path planning/obstacle avoidance techniques transform a continuous problem (i.e. path planning in an obstacle ridden environment) into a discrete problem that involves searching a directed graph (for example, a visibility graph or Voronoi diagram) for feasible solution paths - where a directed graph $G = (V, E)$ is essentially a collection of nodes/vertices V, interconnected by edges E (often straight lines) specified in a particular direction. The connections between nodes are often stored in an adjacency matrix A which is given below for the example graph.

The resultant directed graph typically possesses a number of feasible paths that join the start and goal configurations. From these feasible paths, the next challenge is to identify the optimal one. In robot planning situations, the cost function (typically based on time and cost considerations) that is usually minimised is the path length. Hence, algorithms for extracting this shortest path are required to allow efficient robot navigation (Rankin and Crane III 1996). The next two sections introduce two commonly used shortest path graph searching algorithms. These are used in the path replanning technique developed in section 3.13.

3.3.1 Dijkstra's Algorithm

Dijkstra's algorithm is a widely used shortest path graph searching technique. It finds the shortest path between two nodes in a graph, and in the process also extracts the minimum cost path from all nodes to the source node (Dijkstra 1959). To perform the search, the algorithm is applied once to each vertex and operates in the following way:

Theorem 2 *Given a directed graph $G = (V,E)$ with N vertices and a source (starting) node V_s, the algorithm maintains a set of S vertices for which the shortest path has been found, and a set of T vertices for which the shortest path has yet to be found. Two other data structures are also required, D, an array of the best estimates of the shortest path to each vertex and P, an array of the predecessors*[4] *for each node*[5]*.*

Initially, $T = V$, $S = 0$, $D_s = 0$, $D_i = \infty$ and $P = 0$. At each step of the algorithm (until T is empty), the vertex V_m in T with the smallest D value is removed from T and placed in S. Each neighbour (i.e. the node connected to V_m via an edge) of V_m is interrogated to see whether a path through V_m would be shorter than the currently best known path. If a shorter path estimate is found at an adjoining node, then the appropriate D_i is updated and V_m is placed in the predecessor element P_i. This operation is performed N times until T is empty. Once all nodes have been expanded, the shortest path between the start V_s and goal V_g nodes is then found by searching through the predecessors vector P from P_g until the initial vertex V_s is reached.

For a pseudocode description/listing of how to implement Dijkstra's algorithm the reader is referred to (Foster 1995). Additionally, a simple shortest path test program, 'GeneratePath.m' (written in MATLAB and included on the attached software CD) which demonstrates how to implement Dijkstra's algorithm could be perused. This program's results are displayed in section 3.3.3.

However, the 'greedy' nature[6] of Dijkstra's algorithm makes it inefficient (note that it must check all nodes), and hence inappropriate for searching large graphs or when limited computation time (for e.g. in real-time applications) is available. Therefore, when such constraints exist, an algorithm that uses knowledge of the graph domain (for e.g. by employing a heuristic 'look-ahead' approach) to improve computational efficiency is to be recommended.

[4]A vertex's 'predecessor' is defined as the previous node it is connected to.

[5]Note: the 's' subscript refers to the starting node, 'm' refers to the current node and 'i' is a general vertex for referencing all nodes.

[6]This is a classic example of a 'brute force' algorithm, since it uses no knowledge of the graph itself and finds the shortest path from all nodes to the source node.

3.3.2 The A* Algorithm

A widely used heuristic graph searching approach is the A* (pronounced 'A star') algorithm, proposed by Hart, Nilsson and Raphael (1968). Unlike Dijkstra's algorithm, this uses an 'intelligent' (and somewhat more complicated) approach to directing the graph search, i.e the process of deciding which node is opened at the next iteration[7]. This is achieved using what is termed an evaluation function (Hart et al. 1968). It is used when searching adjacent nodes, and estimates the shortest path from that node to the goal to approximate it's likelihood of being on the shortest path. The evaluation function $\hat{f}(i)$ is defined as follows[8]:

$$\hat{f}(i) = \hat{g}(i) + \hat{h}(i) \tag{3.1}$$

where $\hat{g}(i)$ is the actual cost of an optimal path from V_s to V_i and $\hat{h}(i)$ is an estimate of the optimal path from V_i to V_g. For the present robot path planning application, the heuristic estimate $\hat{h}(i)$ is obtained by simply computing the straight line distance between V_i and V_g[9]. Using the evaluation function, the A* algorithm is executed through the following simplified procedure[10], for a more detailed explanation the reader is referred to either Hart et al. (1968) or Latombe (1991):

Theorem 3 *Given a directed graph $G = (V,E)$ with N vertices, a source node V_s and a goal node V_g between which the shortest path is required, three key data structures O_v, C_v and P are firstly defined. O_v is a list of all 'visited', but as yet unexpanded*[11] *nodes which are sorted in ascending order by the current evaluation of $\hat{f}$ (eqn. 3.1) at each vertex*[12]*. C_v is the list of all 'closed' nodes, i.e. vertices that have already been expanded, this also contains the current value of $\hat{f}(i)$ at each 'closed' node. P also contains the vertices 'visited' so far (both opened and closed), in addition to their*

[7]To find the optimal (shortest) path between two nodes a limited, rather than the complete, set of vertices must be operated upon. A* attempts to expand the search to include the fewest nodes possible so as to find the optimal path (Hart et al. 1968).

[8]Note: where possible the nomenclature remains unchanged for the description of the A* algorithm.

[9]This represents the absolute minimum distance between these two nodes if connected on a straight line directed graph.

[10]Like Dijkstra's algorithm, only a single iteration of the A* algorithm is performed at each vertex in this application (Latombe 1991).

[11]'Expanded' refers to the interrogation of the nodes adjacent to the current node V_m. Once expanded, a vertex is termed to be 'closed'.

[12]O_v stores the nodes that have a connection to another node, but at this stage have not been expanded.

predecessor (pointer to previous vertex). Note that the P structure is also known as a 'spanning tree'.

To execute the A algorithm requires the following steps:*

1. *Initially, O_v and P are empty, except for the source node V_s. Therefore, this node is expanded first.*
2. *The node V_m in O_v that possesses the lowest value of $\hat{f}(i)$ is next selected for expansion.*
3. *If the current node V_m is the goal node V_g, then the shortest path between V_s and V_g has been extracted and the algorithm is terminated.*
4. *If V_g has not been reached, then V_m is 'closed' (removed from O_v and placed in C_v) and expanded. Calculate $\hat{f}(i)$ at each adjacent node, and add to O_v any node that is not a member of C_v, while updating the appropriate element of P so as to point to V_m. If an adjacent node is in C_v, but the latest evaluation of $\hat{f}(i)$ is smaller than its current one, then return this node to O_v while its predecessor in P is updated with V_m.*
5. *Return to Step 2, and continue iterating until the shortest path is determined or O_v becomes empty, i.e. no solution exists, since there are no nodes left to expand.*

The shortest path between the start V_s and goal V_g nodes, as for Dijkstra's algorithm, is found by searching through the predecessors vector P from P_g until the initial vertex V_s is reached.

For a pseudocode listing demonstrating how to implement the A algorithm, the reader is referred to page 606 of Latombe (1991). Also an implementation of the A* star algorithm can be found in the aforementioned (section 3.3.1) test program 'GeneratePath.m'.*

3.3.3 Implementation: Dijkstra vs A*

To demonstrate the reduced computation time offered by the A* star algorithm, compared to Dijkstra's algorithm, a test graph search was performed using a small directed graph consisting of 50 vertices and 67 edges (see Figure 3.8)[13]. The results are tabulated below, and as expected the performance advantage of A* is quite noticeable[14], reducing computation time by over a factor of four.

Algorithm	**Nodes Expanded**	**Search Time (ms)**
Dijkstra	67	75
A*	14	18

Table 3.1: Comparison of Dijkstra and A* algorithms

Clearly, this test is merely indicative of the computational savings offered by the A* algorithm. In any practical application, the time savings will depend upon the nature of the graph and the particular start and goal configurations selected. However, the shorter path computation time provided by the A* algorithm (a key concern in real-time applications) meant that this was selected for searching the area to be crossed in the obstacle avoidance technique developed in section 3.13.

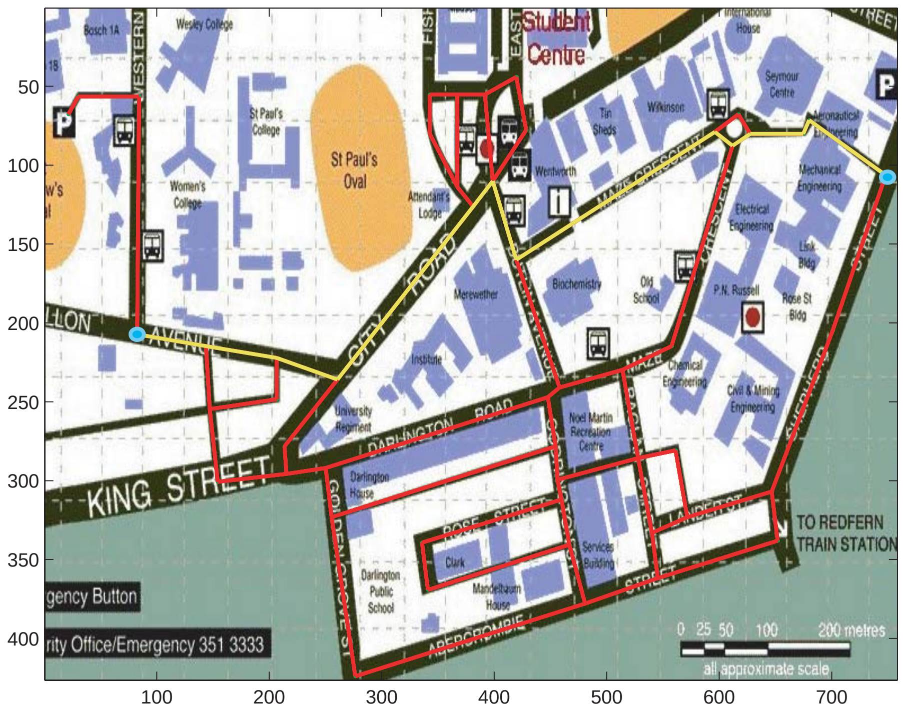


Figure 3.8: Test graph for shortest path algorithms

[13]Tests were carried out using a PC running a Pentium III 600MHz with 128Mb of RAM.
[14]This is despite requiring somewhat more complex software to implement the A* algorithm.

3.4 Obstacles

Path replanning and graph searching algorithms aside, an obviously key aspect in any obstacle avoidance technique is the nature of the obstacles themselves and how they are represented. In this thesis application, information regarding environmental obstacles is provided by the obstacle detection and characterisation software module presented in Lee (2001)[15]. This module serves a dual role, not only detecting obstacles but determining whether an obstacle poses a potential 'collision threat' based upon the current trajectory that it receives from the path planning and tracking module.

3.4.1 Detection

Obstacles are detected using a 180 deg bearing and range laser scanner (see Chapter 1). Based upon consecutive laser scans (and using the obstacle's relative motion), the existence and dimensions of local obstacles are identified, and then classified as being either static or moving with a certain velocity. A typical pair of consecutive laser scans is shown in Figure 3.9. This system is capable of detecting, characterising and tracking obstacles reliably at ranges of up to some 50m. One limitation of this module is that obstacles 'exist' for a limited period only, as they are detected but not stored (i.e. the obstacle detection module does not build a map of the robot's local environment). Rather, this module was designed purely to provide the input to a reactionary navigation scheme.

3.4.2 Representation

The obstacle detection module communicates any 'obstacles of interest'[16] to the path replanning module. Using data association techniques, the obstacles detected by the laser scanner are characterised (i.e. reduced to a 'primitive' form) as either lines (for 'wall like' structures) or circles (for clustered laser returns, such as those encountered from trees), as shown in Figures 3.9 and 3.10. The path replanning technique presented next must use this obstacle representation in recalculating the vehicle's trajectory (if deemed necessary).

[15]See Chapter 6 for implementation details.

[16]'Obstacles of interest' are considered to be those that lie on the current trajectory plus those in the immediate vicinity of this trajectory.

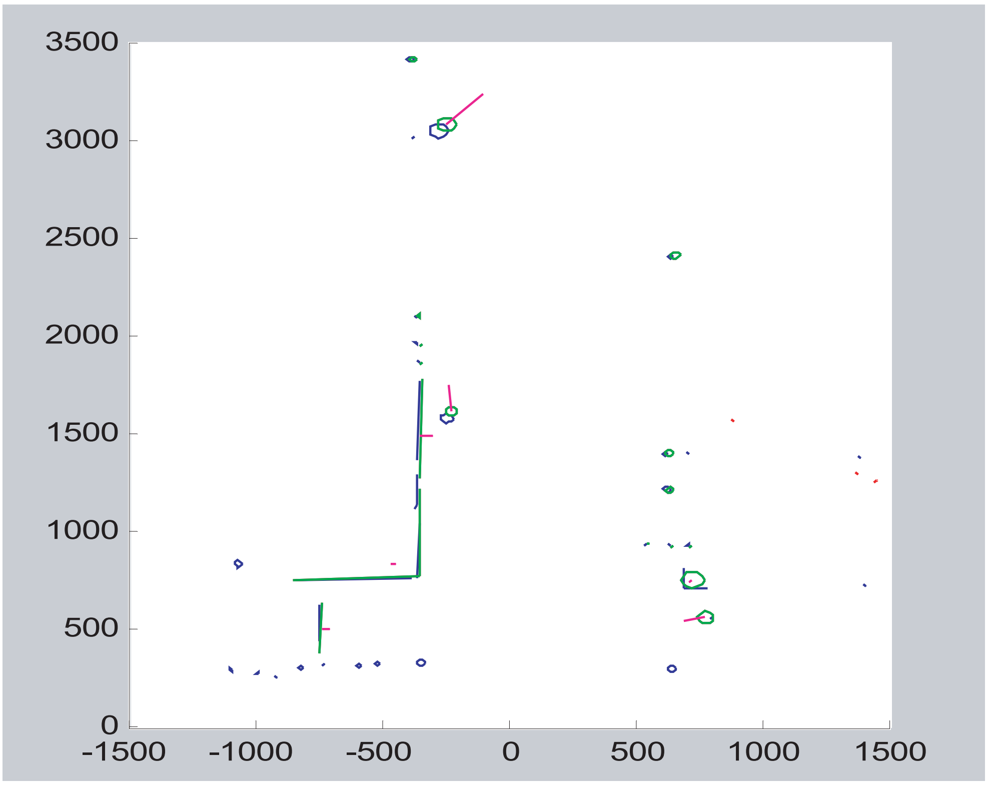


Figure 3.9: Consecutive scans from obstacle detection module

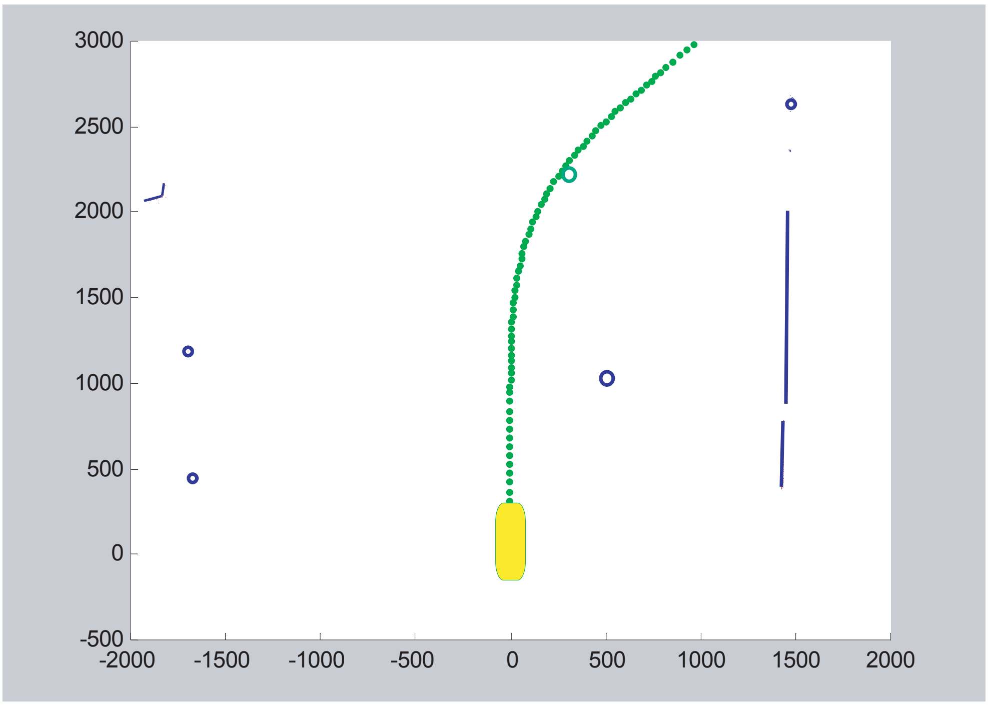


Figure 3.10: Obstacle within vicinity of the current trajectory (green line)

3.5 Path Replanning Technique

In this section, obstacle avoidance (i.e. path replanning), shortest path determination and CCPP techniques are combined to produce a path replanning method that is applicable for real-time implementation on the ute. The approach developed is based upon the path planning technique presented by Graf, Wandosell and Schaeffer (2001) for planning the motion of a wheeled home care robot operating in a 'cluttered environment'. Upon detection of a potential collision the following steps are carried out:

1. A predefined local roadmap is initialised. This will 'surround' the obstacle of interest plus any other nearby obstacles which might affect the path replanning process. The roadmap contains a number of potential paths for leaving (i.e. deviating from) the current trajectory, avoiding any obstacles and then returning to the user specified path.

2. The obstacles identified by the detection module are superimposed onto the trajectory and local roadmap stored by the path planner and tracker. A search of all path segments within the roadmap is then performed to determine if any are blocked by obstacles reported in the local area. Such blocked path segments are removed from the roadmap.

3. A shortest path search algorithm (either A* or Dijkstra) is then performed on the 'obstacle modified' roadmap which extracts the optimal path that avoids all potential obstacles.

4. Using the shortest path (which is specified as a series of straight lines), a smooth trajectory is generated as a series of clothoid or smooth Reeds and Shepp paths.

Obviously, such a basic path replanning technique has limitations[17], mainly arising from the assumptions made about the obstacle configurations the vehicle will encounter. Although the obstacle detection module developed by Lee (2001) is able to track obstacles, this path replanning algorithm will only avoid stationary obstacles[18].

[17]This path replanning method represents an initial technique that could readily be expanded and improved upon in 2002.

[18]Note: this is consistent with the fact that the techniques for obstacle avoidance outlined in section 4.2 typically deal only with static maps of the robot's environment obtained prior to path planning (Graf et al. 2001)

In addition, it is assumed that the environment will be one of 'low obstacle density' with obstacles (or groups of obstacles) well separated from each other, thus ensuring that multiple avoidance procedures are not required (i.e. as a result of new obstacles repeatedly coming into view). The final assumption is that obstacles are detected early enough to allow a reasonable avoidance maneuver to be performed.

Based on the above assumptions, the roadmap containing the possible paths will only be modest in size, looking ahead and replanning within the next 40-60m, or so. Once again, it must be emphasised that this is an initial (conceptual) reactive navigation implementation designed simply to avoid isolated, static obstacles. It was implemented late in this thesis primarily to show how third-party modules could be readily integrated within the path planning and following environment that was the central focus of this work[19].

The following subsections present a more detailed discussion of three key aspects of the path replanning technique, in addition to preliminary simulation results using user specified obstacles.

3.5.1 Workspace Discretisation

Most obstacle avoidance techniques attempt to break up the robot's workspace into discrete points or cells. In this path replanning approach, a roadmap implementation is used (see section 3.2.1). This was chosen as it uses discrete points for representing path nodes, an approach that is similar to that employed in waypoint based CCPP techniques (from Chapter 2), hence the two approaches can be readily 'fused'.

Once an obstacle is detected, a square-shaped roadmap[20] is set up between the vehicle's current path position and a point on the path 40-50m further along the trajectory (i.e. two of the roadmap vertices lie on the path). The roadmap consists of 13 nodes with an orthogonal spacing of 8-10m[21] with the vertices connected by a constrained visibility graph. Note that the visibility graph is constrained by the fact that the path

[19]The results of the simplified obstacle avoidance system can be found in Chapter 6.

[20]A square-shaped roadmap, positioned as shown in Figure 3.11, was chosen as it provides potential paths for leading the vehicle around an obstacle and back to the original trajectory, which is the most obvious response for reactive navigation.

[21]This spacing was selected as it provides enough room to fit physically trackable clothoid segments between them.

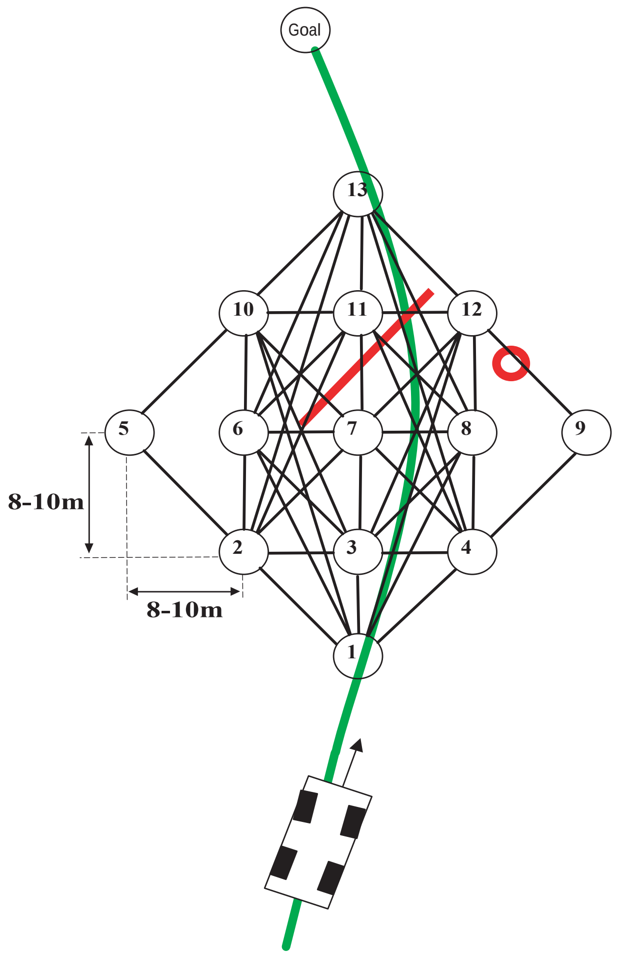


Figure 3.11: Obstacle avoidance visibility graph

segments must not travel backwards[22] in the predefined (unrotated) roadmap. An additional constraint is imposed which ensures any path segments that can only be traversed via an orientation change of greater than 90 deg from the previous segment are removed from the visibility graph. This arrangement is shown in Figure 3.11. The roadmap's adjacency matrix A is given below, this initial visibility graph is modified by the obstacle avoidance/path replanning process.

[22]This condition is imposed because such paths would require very tight cornering (i.e. large orientation changes) in a restricted space. These would be impractical for a car-like robot such as the 'ute'.

$$A = \left[\begin{array}{c|ccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
\hline
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
4 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
9 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]$$

3.5.2 Obstacle Handling

Having initialised the roadmap and its set of path segments, the edges blocked by obstacles must be removed from the adjacency matrix A. This task is performed by checking each edge and determining if it intersects with any of the obstacles (represented as circles or lines) or the region immediately surrounding a particular obstacle (for example, a 2m 'exclusion zone' may be set up around each obstacle). The resultant modification of the roadmap is performed by the procedure given below:

```
for each path segment in $A$:

    COLLISION_FLAG = 0

    while COLLISION_FLAG == 0, search through obstacle set

        if obstacle == line

            * generate 'safe' rectangle corners around line obstacle
            * find straight line equation of the path segment and two boundary
              segments
            * calculate intersection points between segment and obstacle
            ∗ determine if collision will occur
                **if collision is detected**
                    * COLLISION_FLAG = 1
                    * remove path segment from $A$
                **end**
            **end**

        **if obstacle == circle**
            ∗ increase circle radius by safe region
            ∗ calculate line equation for path segment and find the perpendicular
              distance between the circle and path segment
            ∗ calculate intersection conditions between segment and obstacle
            ∗ determine if collision will occur
                **if collision is detected**
                    * COLLISION_FLAG = 1
                    * remove path segment from $A$
                **end**
            **end**

        **end**
```

Figure 3.12 demonstrates how the path segment removal process is carried out. An exclusion zone is set up around each obstacle and all edges in the roadmap which pass inside these zones are removed from the visibility graph, these are the light dashed lines in Figure 3.12. The adjacency matrix A must also be updated by removing now obsolete connections.

3.5.3 Shortest Path Extraction and Smoothing

The adjacency matrix now contains only path segments that do not collide with, or pass near to, any of the obstacles detected in the trajectory's local environment. Figure 3.13 shows the roadmap which now contains only path segments deemed to be free of obstacles. A check must next be run through the obstacle avoidance graph

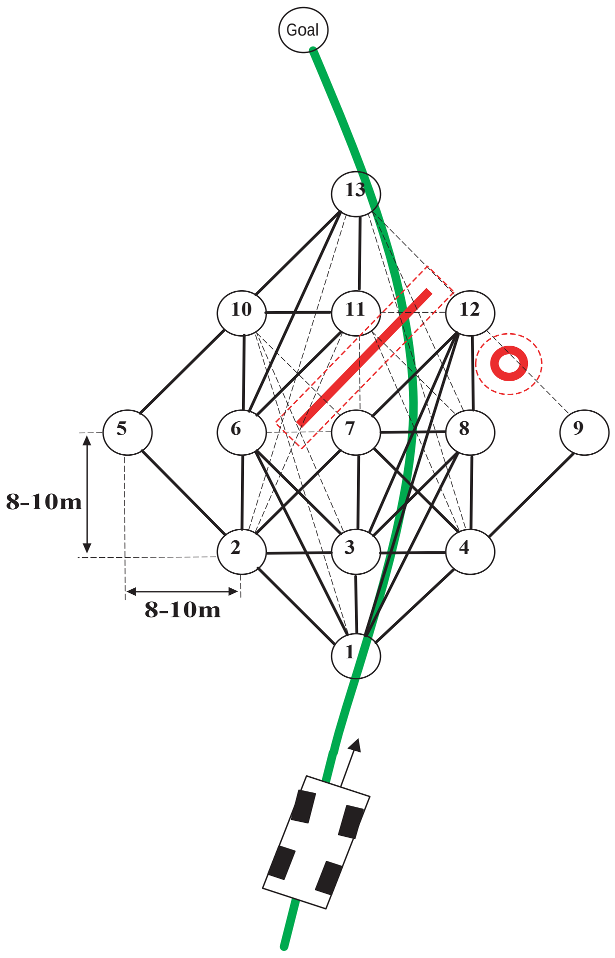


Figure 3.12: Removal of blocked edges from visibility graph

to remove any nodes which are now either completely isolated (i.e. with no path connections) or are unreachable (i.e. have no predecessors) as a result of the obstacle collision check. This is done to ensure that the shortest path algorithm (either Dijkstra or A*) functions correctly[23].

The running of the shortest path algorithm must also account for constraints imposed by the continuous curvature path planning technique used to produce a physically trackable path once the optimal path has been extracted. Within the confines of the predefined roadmap, the orientation change between any two consecutive path segments (given by their predecessors) is not allowed to exceed 90 deg. If this were not required, then the continuous curvature path that is fitted would require a turning manoeuver that is simply too tight for the vehicle. This check is performed during the vertex expansion phase of the shortest path algorithm. An adjacent node will, thus, only be connected to the one currently being expanded if this latter condition

[23]These two graph search algorithms require the ability to expand every node in the graph.

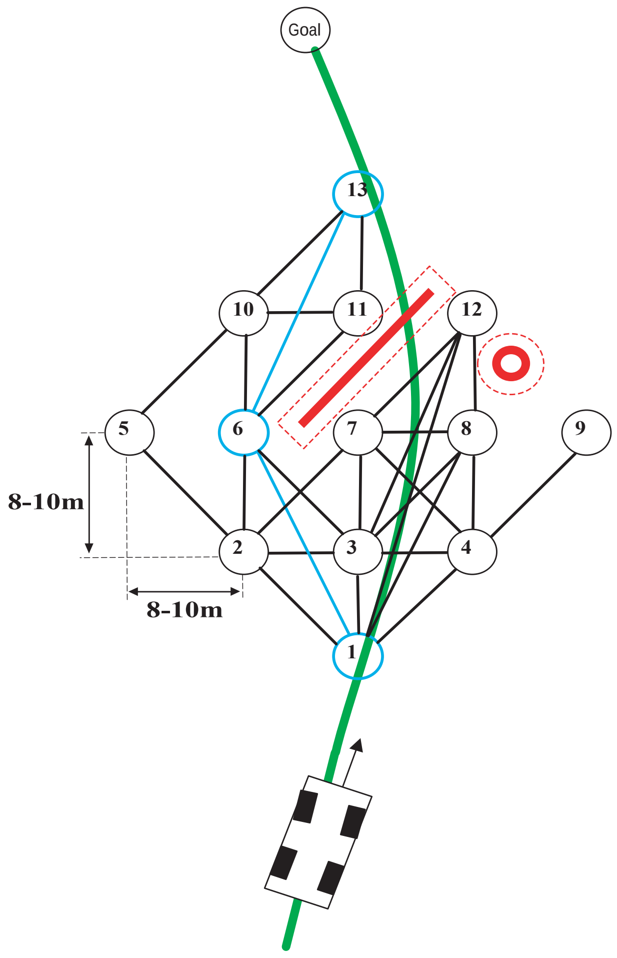


Figure 3.13: Visibility graph showing only ‘safe’ paths

is met in addition to the conditions intrinsic to the shortest path algorithm itself.

Once the search is completed, a ‘safe’ (i.e. avoiding all identified obstacles) shortest path is available as a series of straight line path segments between adjacent nodes (designated by the blue path segments in Figure 3.13). This is converted to a series of posture $(x, y, \theta, 0)$ specified waypoints calculated at the midpoint of each path segment. This series of waypoints is placed between the initial (i.e. current position of the vehicle on the path) and final postures located on the original trajectory (see Figure 3.14)[24]. Using the CCPP technique, a continuous curvature trajectory is generated between the waypoints[25]. This new path segment can now be tracked (by the path following algorithm) to enable the vehicle to avoid the obstacle(s).

[24]Note: the curvature of the path at the initial and final waypoints is assumed to be zero, despite the fact that this is most probably incorrect. A curvature discontinuity will (most likely) exist here, but is accepted to conform with the waypoint specification used by the CCPP techniques developed in Chapter 2.

[25]The continuous curvature path will deviate slightly from the straight path segments, but this is accounted for by the exclusion zone placed around each obstacle.

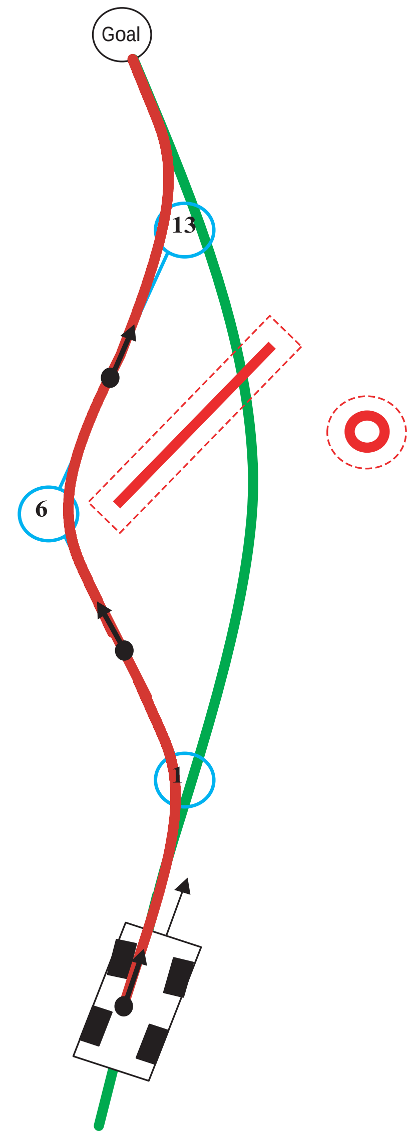


Figure 3.14: Replanned Path

3.5.4 Implementation and Simulation

The path replanning technique just described has been implemented in MATLAB for ‘offline’ operation with simulated obstacles specified by the user. Figure 3.15 shows an example of the trajectory being modified so as to avoid a straight line obstacle. The modified path itself is continuous in curvature, but at the boundaries between the two path segments there exists a small curvature discontinuity because the planning techniques developed in Chapter 2 assume waypoint curvature to be 0 [26]. These CCPP methods could be developed further at some later time, so as to maintain continuous curvature between waypoints possessing non-zero curvature. Another boundary condition imposed by the CCPP techniques used is that the transitional waypoint pairs

[26]Since the boundary waypoints are arbitrary points on the path, curvature is not necessarily 0.

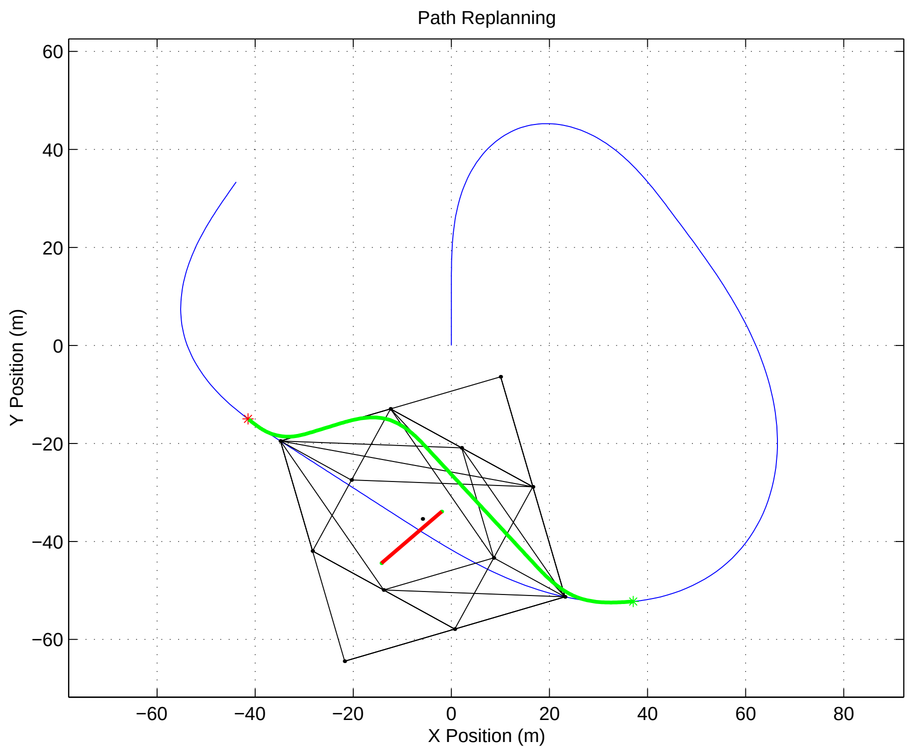


Figure 3.15: Path Replanning Simulation

(one on each of the original and modified paths) must intersect correctly, i.e when the CCPP software is executed, it must be able to generate a legal connection (note that this is detected by the path generator already).

In MATLAB[27] a complete obstacle avoidance iteration takes approximately 500ms [28] to perform. Given that, 1) the computer available on the 'ute' is considerably slower than the one used for testing, 2) multiple software modules would be running at any one time (for example, control, reactive navigation and path replanning), and 3) obstacles detected by the laser scanner were often 'seen' quite late, a simpler obstacle avoidance scheme was implemented on the 'ute' for testing purposes. As described in Chapter 5, the primary objective of performing such simple obstacle avoidance was to demonstrate how additional software modules can be readily integrated into the path planning and tracking system - and this objective was certainly achieved.

[27]Time was not available to convert this module into C/C++ where execution time would no longer be problematic.

[28]Using a Pentium III 600MHz with 128Mb of RAM.

3.6 Conclusion

In this chapter, the replanning of a predefined continuous curvature path for the avoidance of local obstacles (that is, reactive navigation), as detected by the obstacle detection and characterisation module, was investigated. Firstly, commonly used obstacle avoidance methods, and the associated graph searching algorithms, were researched to identify approaches potentially applicable to the path generation techniques developed in Chapter 2. Based upon the path specification and obstacle representation, a roadmap technique was formulated using a constrained visibility graph and the A* search algorithm. Once a shortest path was extracted, a CCPP technique (the second of the two investigated) was used to smooth the new path. Unfortunately, the execution time for this method (in MATLAB) proved too lengthy, hence a simpler obstacle avoidance technique was developed and used in the final system implementation.

Chapter 4

Path Following

4.1 Introduction

The previous two chapters have introduced, discussed and developed techniques for path planning and obstacle avoidance. These two topics are concerned with determining where an autonomous land vehicle will travel based upon two inputs: (1) a desired trajectory entered by the user, and (2) 'environmental' information obtained by the detection of local obstacles.

In this chapter, the focus shifts towards control i.e. to the development of a controller that allows a car-like robot (here the 'ute') to track a predefined, but modifiable (on the run) path, specified as a series of (x, y) coordinates. For this particular implementation, as outlined previously in the thesis specifications (see Chapter 1), only the vehicle's steering is to be controlled[1].

Firstly, car-like robot control is introduced. Next, previously implemented path tracking schemes for ALVs are discussed, prior to an outline being presented of the controller requirements and architecture for our path tracking system. What follows then is development of both the low level (steering) and high level (path tracking) controllers. Finally, the performance of the various control strategies are demonstrated through a series of simulation results[2].

[1]Recommendations relating to the investigation of velocity and transmission control are made in Chapter 6.

[2]The actual implementation (i.e. the 'field test' component of the work) results are provided in Chapter 5.

4.2 Background and Critical Literature Review

Any autonomous land vehicle requires a robust, reliable and accurate path following (control) system which should ensure that the motion objectives specified by the path planning and obstacle avoidance modules are successfully achieved. To perform this functionality, a wide range of control strategies can be implemented. These range from the very complex, where all aspects of vehicle performance (for example, brake, throttle and steering response, vehicle roll and tyre slippage) is both modelled and controlled[3] (Kelly 1994*b*), to the relatively simple. Simpler control strategies are often used in low speed applications, where the vehicle dynamics are such that simplifying assumptions can be made about the vehicle's dynamics and/or certain aspects of vehicle motion can be ignored.

However, in all path following controller realisations, the fundamental control principles remain the same. A process, in this case the motion of a car-like vehicle (through a combination of throttle, brake, steering), must be adjusted in a continuous fashion to achieve a set of specified objectives. A control adjustment 'decision' is made by assimilating information from a number of onboard motion sensing systems (for example, GPS, INS and dead reckoning sensors)[4]. From the sensors, variables such as position, heading and velocity are obtained, which along with the trajectory to be followed provide the inputs to the controller. The controller then proceeds to manipulate this information to generate the appropriate control signals to be sent to the actuators. This process is performed repeatedly so as to provide closed-loop path tracking. The development of ALV control systems, along with other aspects of land vehicle automation, is an area of intense research due to the many potential military, mining, agricultural (and other) applications.

In the following subsections, the path following requirements for a car-like robot are introduced and several control techniques (to achieve path following) are described.

[3]Vehicle modelling is particularly important in high speed applications ($>$ 25km/h) where phenomena such as tyre slippage can compromise controller integrity.

[4]The assimilation and subsequent fusion of data from a variety of sensing systems, using stochastic techniques such as the Kalman filter, is currently a major area of research at the ACFR and other robotics centres around the world - but is outside the scope of this thesis.

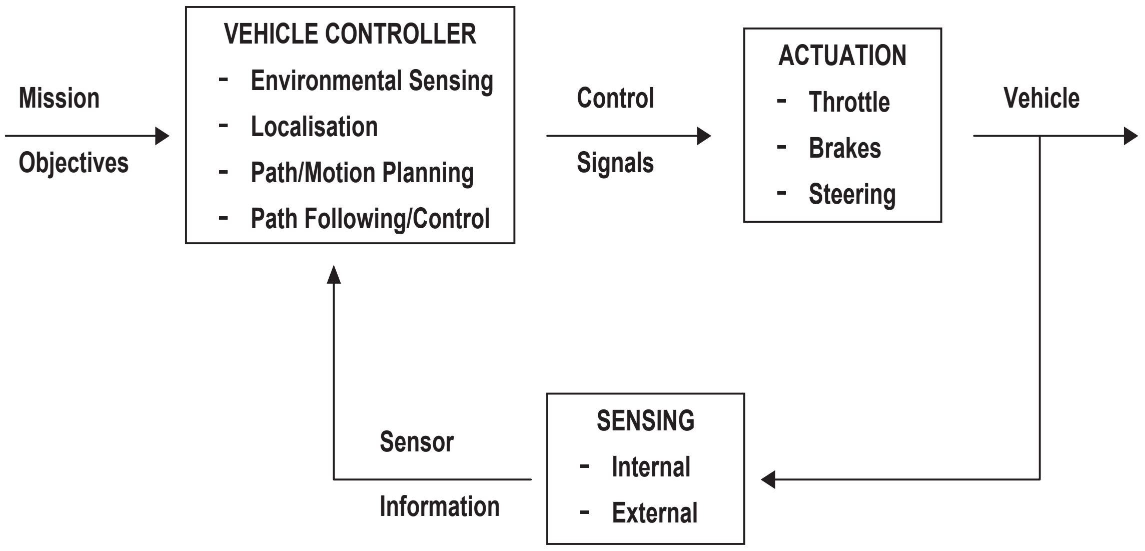


Figure 4.1: Structure of a generic car-like robot control system

4.2.1 Path Following for Car-Like Robots

In Chapter 2, a model for a car-like robot $\mathcal{A}$ was introduced, whose configuration at any moment in time could be defined by its instantaneous posture (x, y, θ, κ). To enable a vehicle to follow a continuous curvature path accurately[5] a control technique must be developed that allows the control of the three variables that comprise vehicle posture:

1. Position, (x, y) - the vehicle must be on the path
2. Orientation, θ - the vehicle must be facing in the correct direction
3. Curvature, κ - the vehicle's steering angle must match the curvature of the path

For the control system in this thesis, the desired posture must be attained by controlling only the vehicle's steering angle ϕ.

4.2.2 Pursuit Techniques

A class of relatively simple (and varied effectiveness), path following strategies are those known as 'pursuit techniques'[6] (Rankin n.d.). There are two key features com-

[5]Continuous curvature paths are the 'path profile of choice' for trajectory tracking, since there are no orientation or curvature discontinuities. Note that it has been proven that feedback controllers for car-like vehicles require continuous curvature for exact tracking of a specified path (De Luca, Orioli and Samson 1998).

[6]Pursuit techniques are able to provide both steering and velocity settings (Wit 2001).

mon to all such techniques, as listed below:

- The path is specified as a series of (x, y) points joined by line segments. This is effectively the same as the path specification output by the path planning module described in Chapter 2.
- A 'look ahead distance' is defined. The controller makes control decisions based upon the vehicle's current configuration[7] relative to some goal point further along the path (i.e. these are 'error based' techniques).

Three pursuit techniques (of increasing tracking accuracy and complexity) are outlined in the remainder of this section:

1. 'Follow-the-carrot' approach

   A carrot point, that is a 'look ahead distance' along the specified path, is defined, with this technique attempting to aim the vehicle directly towards the carrot point. This is achieved by minimising the orientation error between the vehicle and the carrot point. The steering angle ϕ_{con} setpoint is determined by the following simple proportional control law[8], where k_p is the proportional gain and e_{or} is the orientation error:

   $$\phi_{con} = k_p e_{or} \tag{4.1}$$

   However, the 'follow-the-carrot' approach has two major drawbacks. Firstly, the vehicle tends to oscillate about the path, particularly for short look ahead distances[9]. Secondly, the vehicle naturally 'cuts corners' (rather than going around them). Performance can be improved by combining the orientation and perpendicular distance errors into a double-input-single-output (DISO) steering control law.

[7]Configuration can mean position, pose or posture, depending upon the pursuit technique being used.

[8]The performance of this technique could be readily improved if this simple control law were replaced by a PID controller.

[9]A similar phenomenon is observed if the perpendicular distance between the path and the vehicle is the only input to the controller. Performance of this technique can be improved by making the look ahead distance a function of velocity.

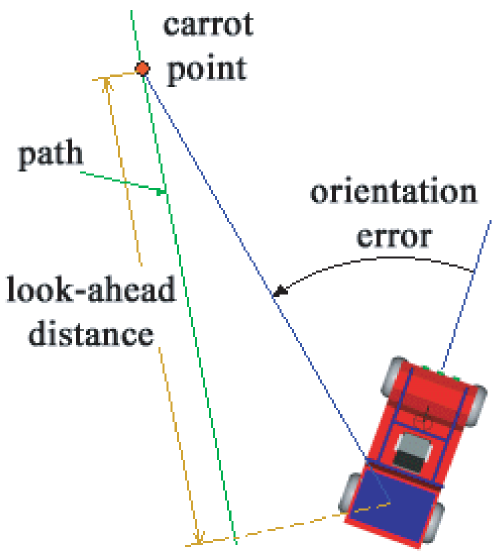


Figure 4.2: ‘Follow the carrot’ path tracking

2. ‘Pure pursuit’ approach

   An improvement on the ‘follow-the-carrot’ approach is the ‘pure pursuit’ algorithm. Once again, a carrot point at a known distance along the path is defined. A circle is then fitted between the vehicle’s current pose and the carrot point’s position. This technique effectively approximates the path as a continually updated series of circular arcs. The on-board steering controller is then required to track this circular arc[10]. The ‘pure pursuit’ technique exhibits a number of improvements over the ‘follow-the-carrot’ approach, such as improved curved path tracking and better handling of large-scale position and heading errors (in both cases, the vehicle did not oscillate wildly about the path).

3. ‘Screw theory’ based control

   The two previous geometric path tracking techniques only generate steering commands based upon the carrot point’s position. Hence, the requirement of vehicle posture control for accurate trajectory following (as introduced earlier) remains unsatisfied. However, the path orientation and curvature are known at the carrot point, hence an improved path tracking strategy would ensure that the vehicle not only arrives at the carrot point but does so with the correct orientation and curvature (i.e. at the correct steering angle).

[10]As highlighted in Chapter 2, paths comprising circles on their own present curvature discontinuities that cannot be tracked without the vehicle stopping. Hence their introduction into a path tracking algorithm is undesirable.

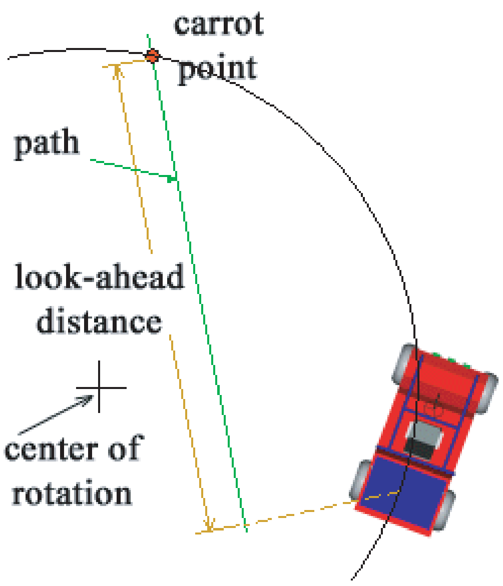


Figure 4.3: 'Pure Pursuit' path tracking

To realise such a trajectory tracking outcome, the position and orientation errors must be minimised simultaneously. Often, this is done by minimising an objective function that is a weighted sum of these two error terms[11]. This is known as proportional path tracking (Wit 2001)). However, despite their success, a central criticism of such (weighted input) algorithms is that all unitary and geometrical information is lost[12]. A recently developed (and quite novel) solution to this problem uses the theory of screws - first introduced by Sir Robert Ball over a century ago (Ball 1900) - to implement a geometric path tracker that acknowledges the nonholonomic motion constraints intrinsic to a car-like vehicle.

Screw theory is used to describe the instantaneous motion of a moving rigid body (such as a car-like robot) relative to a given coordinate system[13] (Wit 2001). Hence, screw theory can be used to describe the motion of a vehicle from its current posture to a future goal (i.e. a carrot point) posture. In the method developed by Wit (2001), two screws are used to specify the desired vehicle motion - a translation screw and a rotation screw. Figure They are selected so as to minimise the position and heading errors (relative to the goal

[11]This is similar to the feedback path tracking algorithm developed later and implemented for controlling the ute (see sections 4.4 and 4.5).

[12]As position and orientation have different units.

[13]Screws consist of a centreline and a pitch - thus, the instantaneous motion of a rigid body can be represented as if it were attached to a screw rotating at a particular angular velocity, hence the name. This is shown in Figure 4.4.

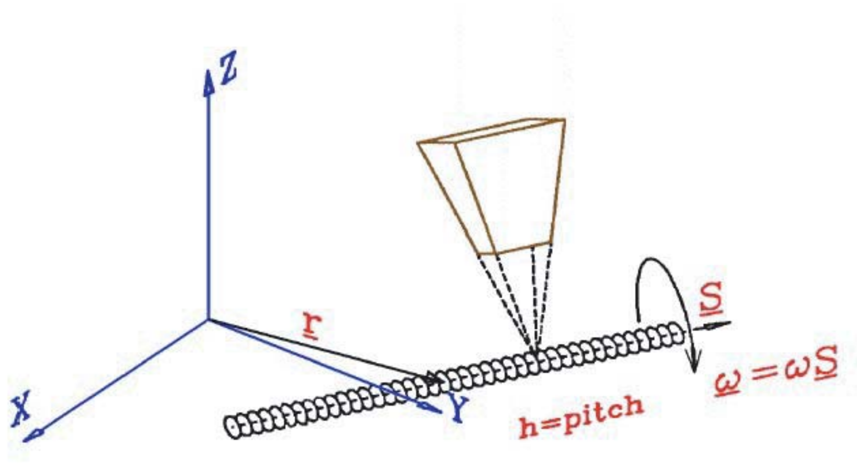


Figure 4.4: Instantaneous motion about a screw

point), respectively. The three key advantages of using screws are: 1) they possess the same units, hence they are additive, 2) control decisions that respect the vehicle's nonholonomic motion constraints can be made, and 3) they allow vehicle posture to be controlled. As a result, exceptionally accurate path tracking is possible, as detailed by Wit (2001).

Finally, techniques similar to those using look-ahead points (as just described) can be implemented through a 'virtual vehicle' approach, where the closest path point (i.e. a reference point) to the vehicle at a given instant is located (Egerstedt, Hu and Stotsky 1998). This point 'follows' the vehicle along the path, and all control decisions are made relative to this point[14]. Figure 4.5 demonstrates the principle of the 'virtual vehicle' for referencing a path. As the vehicle follows the trajectory the closest point changes (denoted by vehicle changing colour) hence the controller operates on a continually updated reference point.

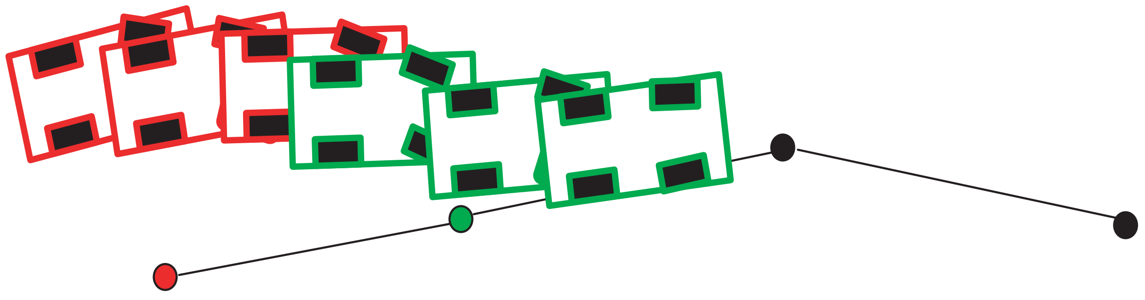


Figure 4.5: Virtual vehicle map referencing

[14]It is termed a 'virtual vehicle' because the mapping of the vehicle's position onto the path gives the appearance of there being a second vehicle tracking the path.

4.2.3 State-Space Techniques

The behaviour of complex systems, such as the motion of vehicles (particularly at high speeds), is highly nonlinear. State-space techniques lie at the heart of much modern control theory, and are built up on the assumption that even complex nonlinear systems behave approximately linearly in the vicinity of the point of linearisation. Once the system has been linearised (to provide a state-space description), multivariable linear control theory can be employed to design a wide range of controllers.

For example, a state-space representation of a vehicle's kinematics may be used in feedforward or (model-based) predictive controllers, as they can provide accurate estimates of the vehicle's future states based upon its current configuration and its known dynamic behaviour. As a result, control decisions/actions can be made based upon (predicted) future states of the vehicle. Hence, control action is not solely reliant upon error based techniques (such as for feedback control), and can be initiated 'ahead of time'. For example, as will be shown in section 4.5.5, a feedforward controller greatly improves the ability of a vehicle to follow curved paths[15].

State-space techniques are widely (and successfully) used for the task of path tracking in car-like vehicles, hence there are numerous examples available in the literature[16]. One well-known example is the state-space based feedback controller. Using the vehicle's kinematic equations, as described in Hemami, Mehrabi and Cheng (1994), De Luca et al. (1998) and Guldner, Sienel, Tan, Ackermann, Patwardhan and Bunte (1999), such controllers have been developed for trajectory following in car-like vehicles. Their effectiveness can be witnessed in the work of Guldner et al. (1999) who automated the steering of a passenger vehicle to meet tracking tolerances of 0.15m in dry conditions and 0.30 m the wet, while maintaining 'passenger comfort' at levels comparable to manual steering.

4.3 Control System Overview

In the context of this thesis, given a continuous curvature path (as developed in Chapter 2), a steering control system is required that enables the 'ute' to follow this

[15]Previous HSV theses, such as Cowlishaw (1997), have investigated the development of dynamic vehicle models, however this aspect is not a key focus of this present thesis.

[16]State-space techniques are often used to greatly improve the performance of simple path tracking schemes (such as a feedback controller), at the expense of a modest increase in computation.

path using steering control only. To meet the stated path tracking requirements, as introduced in section 4.2.1, a steering controller with the following architecture is proposed (Figure 4.6). This control system actually consists of two controllers - a 'high level' (path tracking) controller, and a 'low level' (steering) controller.

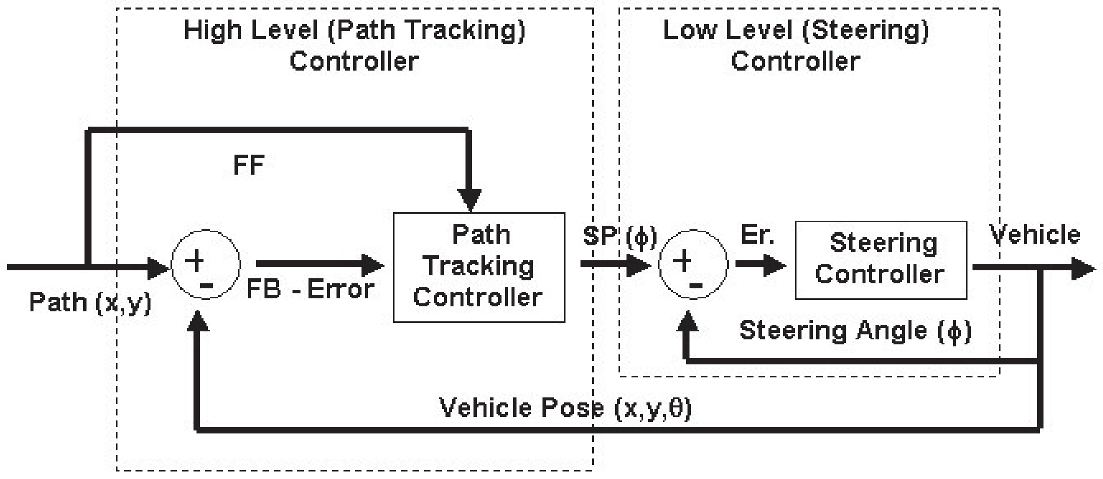


Figure 4.6: Steering controller structure

These two controllers interact as follows. The high level controller's (see section 4.5) responsibility is to assimilate current vehicle configuration and path information so as to generate a steering setpoint that is sent to the low level control block. It receives the vehicle's current pose (via the navigation loop[17]), in addition to the posture of the nearest path (x, y) element. The high level controller then uses a combination feedback/feedforward control law to calculate the new steering setpoint.

The low level controller (see section 4.4) takes the steering setpoint provided by the path tracking controller, and employs a PID (feedback) control law to generate the output signal to drive the steering motor, which turns the front wheels to the desired orientation. Steering angle feedback (for calculation of the steering error) is provided by the ute's LVDT (see Chapter 1). In the following sections, the two controllers are described, developed and the proposed control system evaluated (through a simulation study).

[17]The navigation loop can consist of any of a number of localisation sensors, including GPS, INS, compass and dead reckoning.

4.4 Low Level Control: Steering

Prior to this thesis, a low level steering controller was already in place on the ute (initial implementation by Gurce Isikyildiz, 1999). However, this required both substantial software alterations and controller tuning.

4.4.1 PID Algorithm

The steering controller employs the PID feedback control law. As such, it is an error based strategy, with the control action calculated from the behaviour of the error (i.e. the difference between the steering setpoint and the measured steering angle). This form of control may be represented by the following equation (Nebot 2001):

$$m = k_p\varepsilon + k_i \int \varepsilon dt + k_d \frac{d\varepsilon}{dt} \tag{4.2}$$

where m is the output (control) signal, ε is the error term, and k_p, k_i and k_d are the proportional, integral and derivative gains, respectively. Such a controller is tuned (i.e. its response is modified) by varying these three PID gains (Mutambara 1999).

However, to implement such a control law in terms of discrete time intervals, the 'velocity'(or 'difference') form of the PID equation is required (Nebot 2001):

$$\Delta m = k_p(\varepsilon - \varepsilon_{-1}) + k_i\varepsilon + k_d(\varepsilon - \varepsilon_{-1} + \varepsilon_{-2}) \tag{4.3}$$

$$m = m_{-1} + \Delta m \tag{4.4}$$

where Δm is the change in the control signal m, and the $_{-1}$ and $_{-2}$ subscripts indicate a term's value at one and two time steps ago, respectively. This is the implementation used for steering control on the ute. A typical steering response to a 10 deg step change is shown in Figure 4.9. Further discussion of the steering controller and it's performance can be found in Chapter 5 on system implementation.

4.4.2 Steering Modelling

Any path following system simulation requires that the steering response of the vehicle must also be simulated and implemented. To provide the latter, a steering model was developed based upon steering response data taken off the ute.

In many engineering situations, a system's dynamic behaviour can be well approximated as either a first or second-order linear system (defined by either a first or second-order differential equation, respectively). Consequently, both of these mathematical representations were investigated here. The transfer function[18] $G_1(s)$ (in the Laplace domain) for a first-order system is given by (Di Stefano, Stubberud and Williams 1990):

$$G_1(s) = \frac{1}{\tau s + 1} \tag{4.5}$$

where τ is the system's 'time constant'. The corresponding time domain differential equation for this system is:

$$\tau \frac{dS_{st}(t)}{dt} + S_{st}(t) = S_{sp}(t) \tag{4.6}$$

where S_{st} is the vehicle's actual steering angle while S_{sp} is the requested steering angle. As for the PID controller, this first-order model was rewritten in difference form, so that it can be solved repetitively at each new time interval. Making S_{st} the subject of the equation gives the following:

$$S_{st}(t) = \frac{S_{sp}(t) + \frac{\tau}{\Delta t} S_{st}(t-1)}{\frac{\tau}{\Delta t} + 1} \tag{4.7}$$

where Δt is the time interval between recalculations. Figure 4.7 shows a typical response of this first-order system to a 10 deg step change.

The transfer function for a second-order system $G_2(s)$ is given by the following (Di Stefano et al. 1990):

$$G_2(s) = \frac{{w_n}^2}{s^2 + 2\zeta\omega_n s + {\omega_n}^2} \tag{4.8}$$

[18]In this case, the transfer function relates the desired steering angle to the actual steering angle.

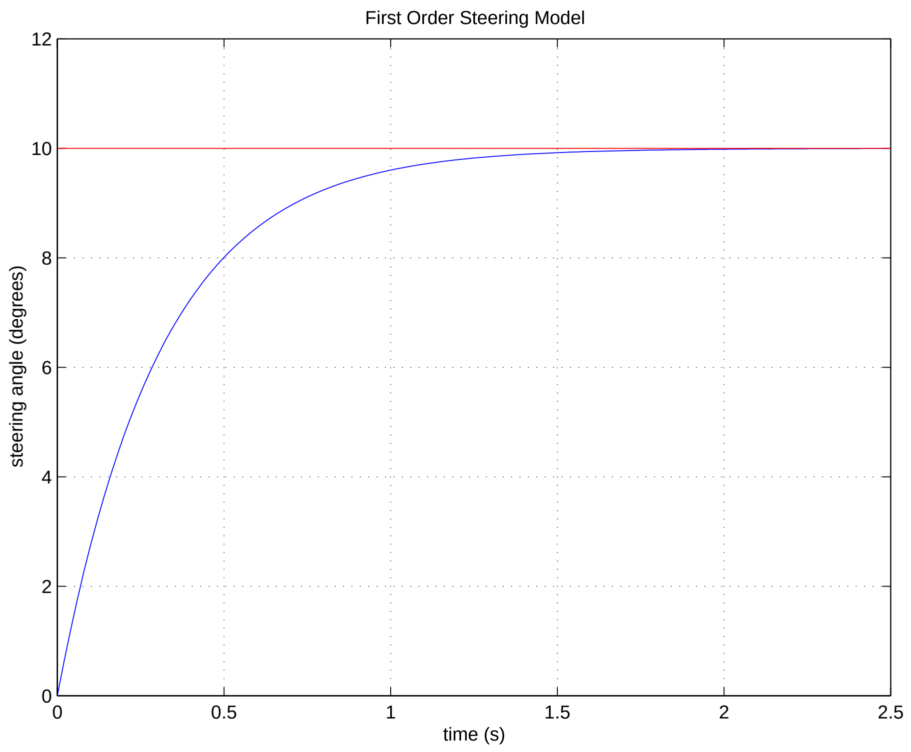


Figure 4.7: Typical response of a first order system to a step change

where ζ is the damping ratio and w_n is defined as the system's natural frequency. This transfer function equates to the following time domain differential equation:

$$\frac{d^2S_{st}(t)}{dt^2} + 2\zeta\omega_n \frac{dS_{st}(t)}{dt} + {\omega_n}^2 S_{st}(t) = S_{sp}(t){\omega_n}^2 \tag{4.9}$$

Replacing the derivative terms with their finite difference approximations gives the following difference equation for calculating the latest steering angle:

$$S_{st}(t) = \frac{\omega_n^2 \Delta t^2 S_{sp}(t) + (2 + 2\zeta\omega_n\Delta t)S_{st}(t-1) - S_{st}(t-2)}{1 + 2\zeta\omega_n\Delta t + \omega_n^2\Delta t^2} \tag{4.10}$$

Figure 4.8 shows a typical response of a second-order system to a 10 deg step change.

Based upon the measured response (to a 10 deg setpoint change) of the steering controller on the ute (discussed in Chapter 5), it was decided that a second-order system would provide the better approximation. Hence, a second-order system with the following parameters was employed in the simulation studies:

- Natural frequency, $\omega_n = 4.8$
- Damping Ratio, $\zeta = 0.33$

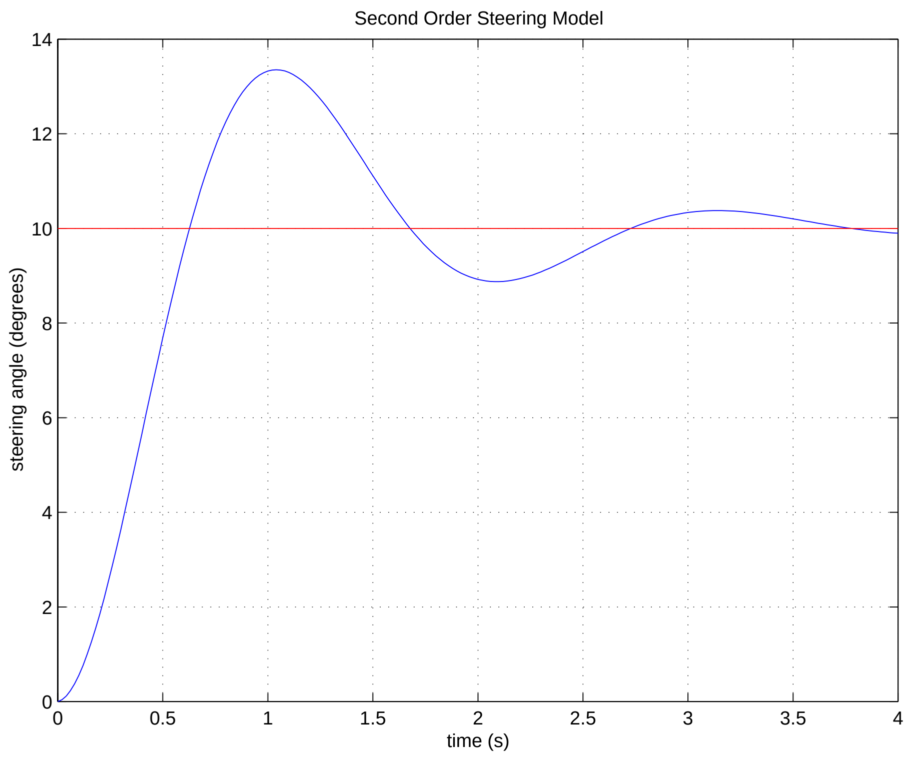


Figure 4.8: Typical response of a second order system to a step change

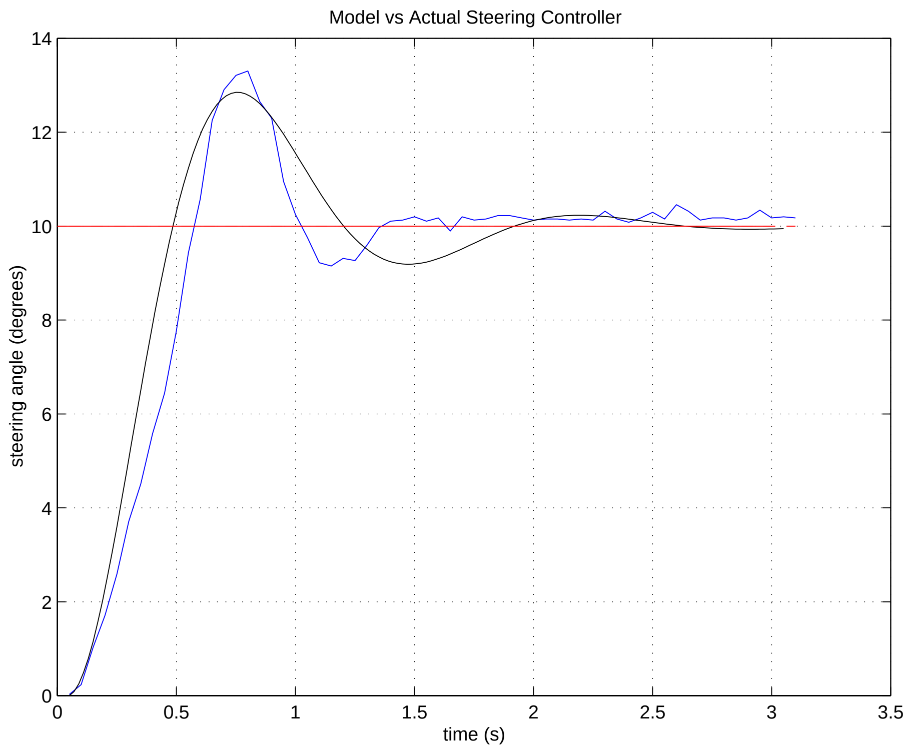


Figure 4.9: Second order model (black) for steering controller developed from actual steering response data (blue)

The result is a steering controller model with the following characteristics, 1) rise time of 0.5s, 2) overshoot of 3 deg and 3) a negligible steady state error. These values adequately match those of the 'ute's' actual steering response which is shown plotted against the model's steering output in Figure 4.9.

4.5 High Level Control: Path Tracker

In the previous section, the steering controller and a second-order steering model approximation were introduced. The steering setpoint the low level controller receives is generated by the second control block - the high level (or path tracker) controller. The role of this latter controller is to use path and vehicle pose information to generate an appropriate steering angle. In the following sections, the path tracking controller is developed and simulation results presented.

4.5.1 Path Linearisation

In section 4.2.1, the three parameters which require tight control for accurate path tracking were outlined. Prior to developing an algorithm to control vehicle posture, a means of extracting this information from the trajectory to be followed must be formulated. Given that the path is specified as a series of discrete cartesian coordinates at regular (approximately 0.5 m) intervals, a linearisation technique is proposed, i.e. the path is considered to consist of a series of 'short' line segments[19].

The high level controller uses a 'virtual vehicle' technique (see section 4.2.2) for referencing the trajectory, i.e. the control action is determined from calculations performed at the nearest path element to the vehicle. Having located this point $\mathcal{P}_{vv}$, given by (x_k, y_k), where $\mathcal{P}_{vv}$ is a point on the trajectory, the path linearisation may be performed (see Figure 4.10). The first step here is to calculate the line equation $\mathcal{L}_{vv_k}$ at this point - this is given below:

$$\mathcal{L}_{vv_k} : \quad \left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right) x - y + y_k - \left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right) x_k = 0 \tag{4.11}$$

The line coefficients a_k, b_k and c_k are defined as follows:

[19]This is somewhat similar to the path planning technique introduced in Chapter 2, except that the line segments are much shorter here.

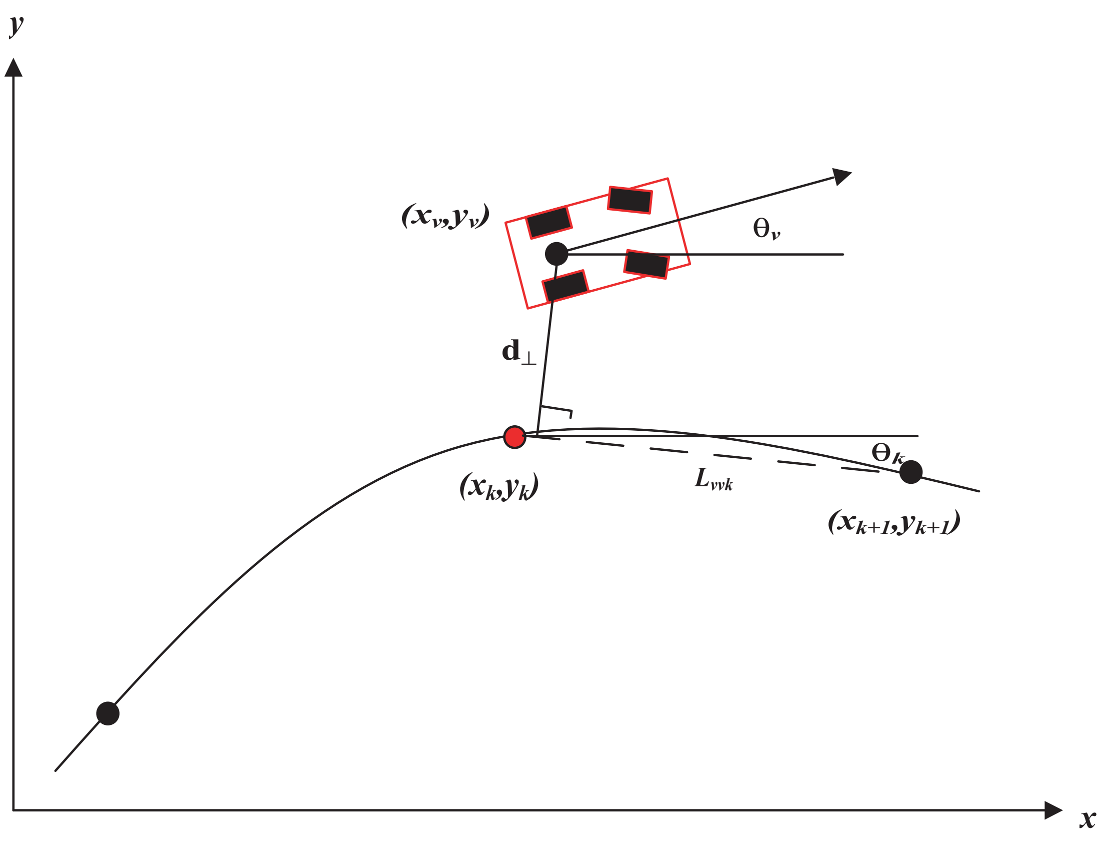


Figure 4.10: Controller path linearisation

$$a_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \tag{4.12}$$

$$b_k = -1 \tag{4.13}$$

$$c_k = y_k - \left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right) x_k \tag{4.14}$$

Using this equation, the perpendicular distance $d_\perp$ (used by the feedback controller as the vehicle position error in section 4.5.2) of the vehicle from the nearest linear portion of the path is measured relative to $\mathcal{L}_{vv_k}$, and is given by[20]:

$$d_\perp = \frac{|a_k x_v + b_k y_v + c_k|}{\sqrt{a_k^2 + b_k^2}} \tag{4.15}$$

where (x_v, y_v) is the vehicle's current position.

The current orientation θ_k of the path is also required and is calculated as follows:

[20]Note: the sign of the perpendicular distance specifies what side of the line the point (i.e. the 'virtual vehicle') lies on.

$$\theta_k = atan2(y_{k+1} - y_k, x_{k+1} - x_k) \tag{4.16}$$

The relative orientation e_θ between the path θ_k and the vehicle θ_v is also required (note that this is used by the feedback controller in section 4.5.2):

$$e_\theta = \theta_v - \theta_k \tag{4.17}$$

Finally, the current path curvature κ_k (i.e. the change in orientation per change in unit path length) is needed, and may be determined using the following difference formula:

$$\kappa_k = \frac{\theta_{k+1} - \theta_k}{\sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2}} \tag{4.18}$$

4.5.2 Feedback Control: Proportional Path Following

The initial implementation of the path tracking controller used a proportional path following (or PPF) algorithm for tracking trajectories consisting of only straight line segments[21]. This is an error based (feedback (FB)) technique that uses the weighted sum of the vehicle's position (calculated as a perpendicular distance) and orientation errors relative to the trajectory to calculate an appropriate steering setpoint. Mathematically, a steering only proportional path follower is realised by the following DISO control law:

$$\phi_{sp} = \pm k_\perp d_\perp \pm k_\theta e_\theta \tag{4.19}$$

where ϕ_{sp} is the steering setpoint sent to the steering controller, $d_\perp$ is the perpendicular distance from the path (given by eqn. 4.15), $k_\perp$ is the perpendicular distance gain, e_θ is the relative orientation between the path segment θ_k and the vehicle's orientation θ_v (eqn. 4.17) and k_θ is the orientation error gain. The sign of each term is important for ensuring that the algorithm drives the vehicle toward the path in the correct way.

[21]This path tracking technique is based upon concepts presented in the 'pursuit' techniques found in section 4.2.2, except that the path is here referenced using a virtual vehicle, rather than a carrot point approach.

A PPF algorithm was chosen as it allows both vehicle position and orientation to be controlled simultaneously[22]. This algorithm operates by attempting to minimise the path tracker's objective function (essentially a weighted deviation) by controlling only the steering angle[23]. This controller is quite adequate for paths comprising a number of straight lines, as curvature control is unnecessary, since the curvature of a straight line is, by definition, zero. As required, when both the position and orientation errors are zero, then this algorithm outputs the required steering angle setpoint of 0 deg. However, as demonstrated in the simulation results (section 4.5.4) a PPF algorithm, as all path following algorithms for car-like robots do, experience significant tracking errors when switching between path segments.

The gain terms are chosen to ensure that when the distance error is large, the perpendicular distance term dominates the objective function and drives the vehicle towards the path. As the vehicle approaches the path, so the orientation error term comes into play and the steered wheels are moved so as to correct the orientation error. Hence, the vehicle approaches the path asymptotically (as shown in Figure 4.13). Another advantage of the PPF algorithm is its ability to handle large scale errors, i.e. cases where the vehicle's initial error (position and/or orientation) are considerable[24]. Besides the magnitude of the gain terms, the behaviour of this control algorithm is also determined by the signs of these terms. These are set by the vehicle's position and orientation relative to the current path segment (implemented as a look up matrix in the control software) - they have been setup to handle large position and heading errors as demonstrated in Figure 4.13. For more details regarding the operation/implementation of both the feedback and feedforward/feedback (FF/FB) controllers the reader is referred to the software CD included with this thesis.

However, as the simulation results (in section 4.5.4) demonstrate, the tracking accuracy of the PPF algorithm diminishes drastically when continuous curvature paths are introduced. The vehicle now appears to consistently 'lag behind' the path (Figure 4.15). This is to be expected, as the PPF algorithm is a feedback controller, i.e. corrective action is only taken when an error is detected. Therefore, the vehicle must drift away from the desired path prior to the initiation of any corrective steering

[22]As noted in section 4.2.2, using only a single variable in the control algorithm leads to oscillatory, sometimes unstable, path tracking.

[23]The steering angle is mechanically limited to ϕ_{max}.

[24]However, in most practical situations, the initial errors will be small, as the vehicle will be located at/or near the start of the trajectory.

action. The control challenge with non-zero curvature paths (unlike straight lines) is that even when the position and heading errors are small, the required steering angle is not zero. Rather, it is a function of the current path curvature. Consequently, a modified PPF algorithm is developed in section 4.5.5 which takes into account the curvature of the path.

In the next two sections, the vehicle pose model (as used in the simulations studies) is outlined, followed by the simulation results for the feedback path tracking controller.

4.5.3 Vehicle Pose Update Model

When a path following simulation is performed, the vehicle's posture must be determined. In section 4.4.2, a steering model was developed to allow the steering response[25] to be calculated. Additionally, the pose of the vehicle requires updating with each iteration of the simulation. This is performed using a vehicle pose update model. For the simulation results presented in sections 4.5.4 and 4.5.6, the tricycle model for a car-like robot was used (Borenstein, Everett and Feng 1996), implemented by the following update equations[26]:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta t V_k \cos(\theta_k + \phi_k) \\ y_k + \Delta t V_k \sin(\theta_k + \phi_k) \\ \theta_k + \frac{\Delta t V_k}{L} \sin(\phi) \end{bmatrix} \tag{4.20}$$

4.5.4 FB Simulation Results

Simulations were widely used throughout the development of both the FB and FF/FB controllers, as they allow controller behaviour to be verified and the effects of different gain combinations to be investigated. Three indicative sets of simulation results[27] are presented for the FB controller. All simulations were run assuming, 1) a constant velocity of 20km/h, 2) proportional gains of $k_\perp = 0.04$ and $k_o = 0.35$ and 3) a maximum steering angle command of 15 deg.

[25]This updates the vehicle's curvature.

[26]The variable names used for the update model are consistent with those introduced in the vehicle model presented in Chapter 2. The reader is referred here for a diagram of the vehicle model.

[27]Each of the simulation result sets (both FB and FF/FB) contain a graph of the path tracking (desired and actual path) and a plot displaying steering angle, absolute positional error and heading errors.

In Figures 4.11 and 4.12 the results for tracking a ‘circular’ path consisting solely of straight lines is presented. The path tracker performed quite well, as a FB controller is able to track straight segments with minimal error. However, as outlined in Chapter 2, problems occur at connections between straight lines. The transition between line segments introduces a periodicity into both the heading and positional absolute errors, as shown in Figure 4.12, due to the controller switching from tracking one line to the next. Hence, the vehicle becomes instantly misaligned (meaning that the vehicle ‘cuts’ corners) relative to the new path segment (up to 6m and 60 deg). This introduces the ‘ECG-like’ spikes into the vehicle pose error, and results in strong steering action rather than the desired smooth turning of the wheel. Such tracking results clearly present a strong case for continuous curvature trajectories.

The second set of simulation results (Figures 4.13 and 4.14) demonstrate the successful negotiation of large scale vehicle pose errors by the FB controller (see software CD for implementation). Typically, the vehicle will be required to handle only small errors as it will start on (or very near) to the path to be tracked. However, this controller feature proved to be particularly valuable when the simplified obstacle avoidance technique was implemented (see Chapter 5) and avoidance left the vehicle 5-10m from the path.

Finally, the FB controller is used to track a continuous curvature (reference[28]) path[29] (Figures 4.15 and 4.16). The results are as expected from section 4.5.2 - the controller clearly ‘lags’ behind the corner, resulting in tracking errors of up to 3m and 25 deg. These errors are significantly reduced by the introduction of a feedforward term into the control law.

4.5.5 Feedforward Control

Once continuous curvature paths were introduced by the path planning module, the limitations of the PPF algorithm, as discussed in sections 4.5.2 and 4.5.4, were exposed. What is required to improve path tracking accuracy is a means of controlling vehicle curvature (i.e. the steering angle), so that when position and heading errors are small, control action is still provided to account for non-zero path curvature. This means that the three variables (defining posture) required for accurate path tracking

[28]For comparative purposes, this trajectory is used for testing the FF/FB controller developed in section 4.5.5. The tracking performance of the two controllers is also compared 4.5.6.

[29]This path was generated using CCPP Technique 2 with $a_{max} = 8m$ and $\rho_{min} = 10m$.

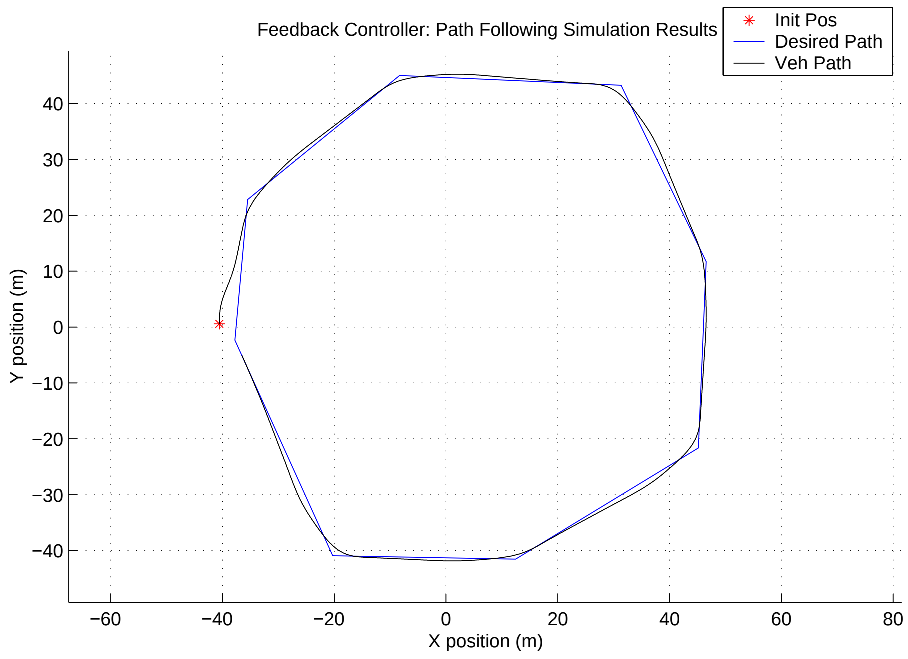


Figure 4.11: FB Controller: Path Following Results

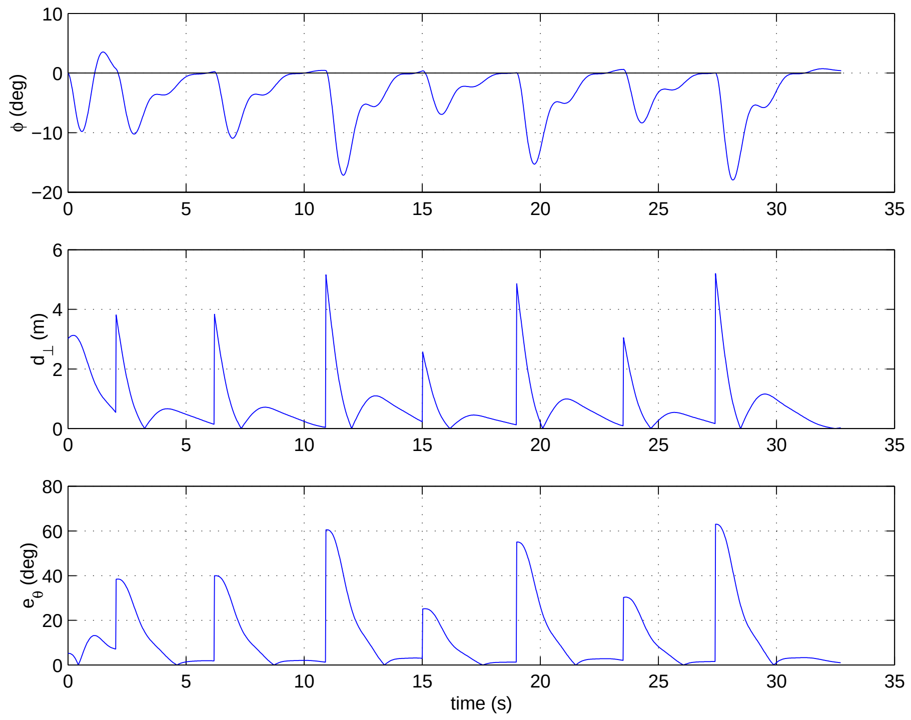


Figure 4.12: FB Controller: Path Following - Steering and Error Results

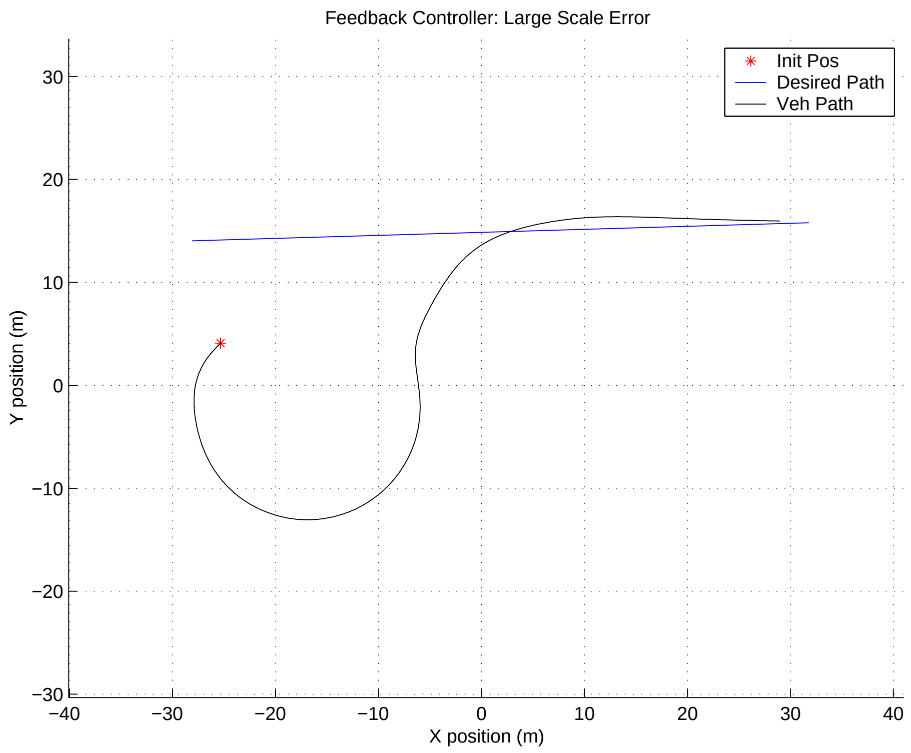


Figure 4.13: FB Controller: Path Following - Large Initial Error Results

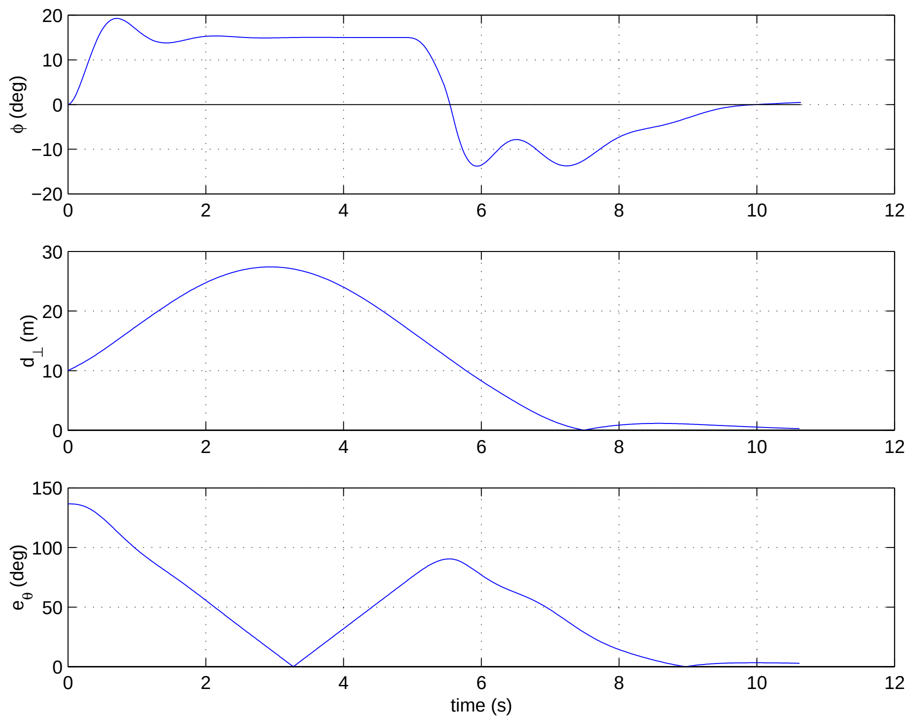


Figure 4.14: FB Controller: Path Following - Steering and Error Results for Large Initial Error

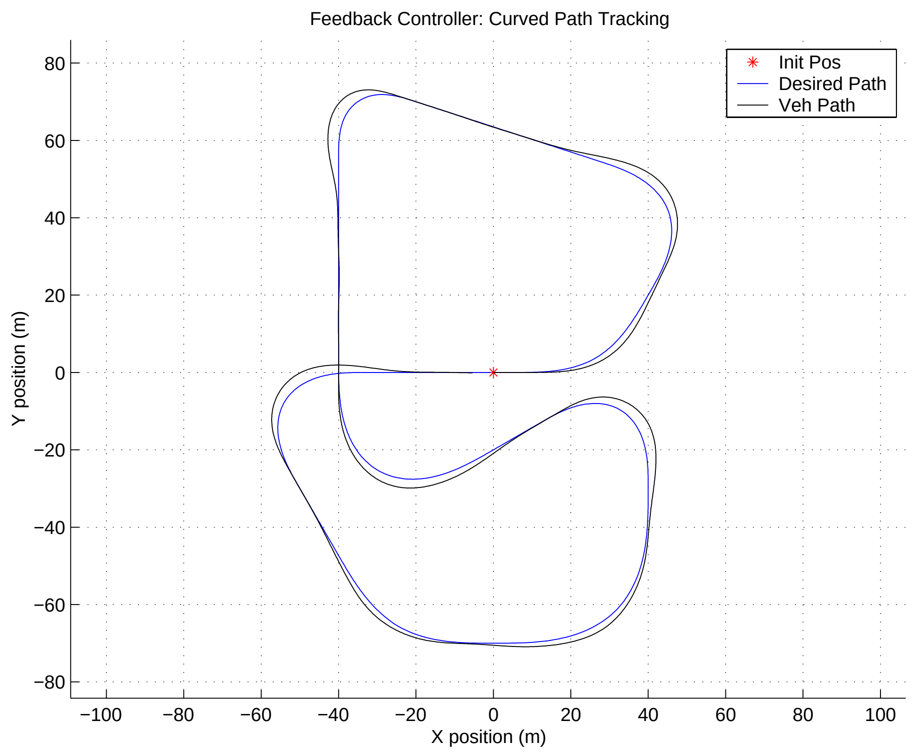


Figure 4.15: FB Controller: Curved Path Following Results

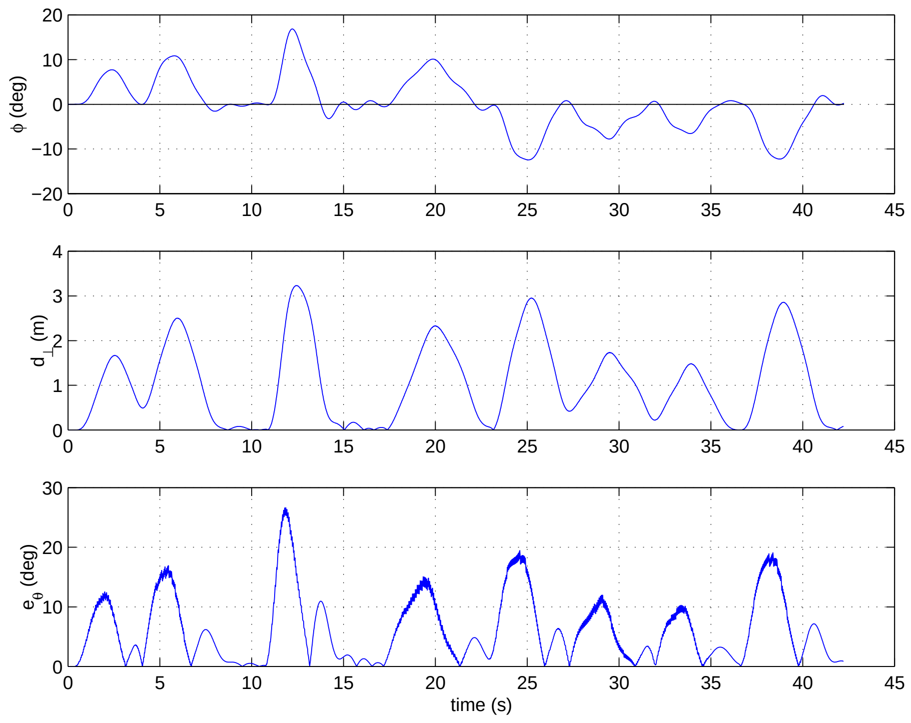


Figure 4.16: FB Controller: Curved Path Following - Steering and Error Results

(see section 4.2.1) would now be controlled. To provide this form of control, a feed-forward[30] component is introduced into the PPF equation (eqn. 4.19), to produce a feedback/feedforward controller as follows:

$$\phi_{sp} = \pm k_{\perp} d_{\perp} \pm k_{\theta} e_{\theta} + k_{\kappa} \phi_{ave_n} \tag{4.21}$$

where ϕ_{ave_n} is the average steering angle of the path over the next 'n' points, and k_{κ} is the feedforward gain. The feedforward component 'predicts' the curvature of the path into the future - therefore, it effectively acts as a dynamic steering angle offset/bias, being determined by the curvature of the path itself. To calculate the average steering angle ϕ_{ave_n}, the average[31] path curvature κ_{ave_n} over the next 'n' path elements is calculated using the following equation:

$$\kappa_{ave_n} = \frac{\sum_{i=k}^{k+n} \kappa_i}{n} \tag{4.22}$$

where k is the index of the current (i.e. closest) path element, and κ_i is the curvature at the ith element given by equation 4.18. All feedforward controllers require some form of process model to make their predictions - in this case, a means of relating path curvature to vehicle steering angle is needed. This is obtained by combining the three equations (eqn. 4.20) in the vehicle update model to obtain a difference approximation for steering angle ϕ as a function of curvature κ:

$$\phi = L \arcsin \kappa \tag{4.23}$$

This relationship is used to convert κ_{ave_n} to ϕ_{ave_n} which is then input into the new control algorithm. The greatly improved results from using this feedback/feedforward controller are shown in the following section.

[30]In the literature review, section 4.2, it was noted that feedforward controllers are widely used for path tracking in car-like robots.

[31]The average path curvature, rather than the instantaneous path curvature, is used as it allows 'future' curvature trends to be identified earlier. It must also be remembered that there is a lag associated with the steering response.

4.5.6 FF/FB Simulation Results

Using the same controller parameters as per the FB controller simulations presented in section 4.5.4, the FF/FB controller with a FF gain of $k_\kappa = 0.6$ was simulated - tracking the same continuous curvature path as shown in Figure 4.15. The results are displayed in Figures 4.17 and 4.18. From these graphs, it is clear that the introduction of a FF component into the control algorithm vastly improves curved path tracking, as was to be expected. The tracking error statistics (maximum, mean and standard deviation of the absolute errors), shown in Table 4.1 for both path tracking simulations, quantify the improvement achievable by adding a feedforward component to the controller.

Controller	**FB**	**FF/FB**
$e_\perp$ max (m)	3.2	0.6
$e_\perp$ mean (m)	1.1	0.2
$e_\perp$ s.d. (m)	0.9	0.1
e_θ max (deg)	26.6	16.1
e_θ mean (deg)	6.2	4.6
e_θ s.d. (deg)	5.6	4.3

Table 4.1: Comparison of simulation results for FB and FF/FB controllers

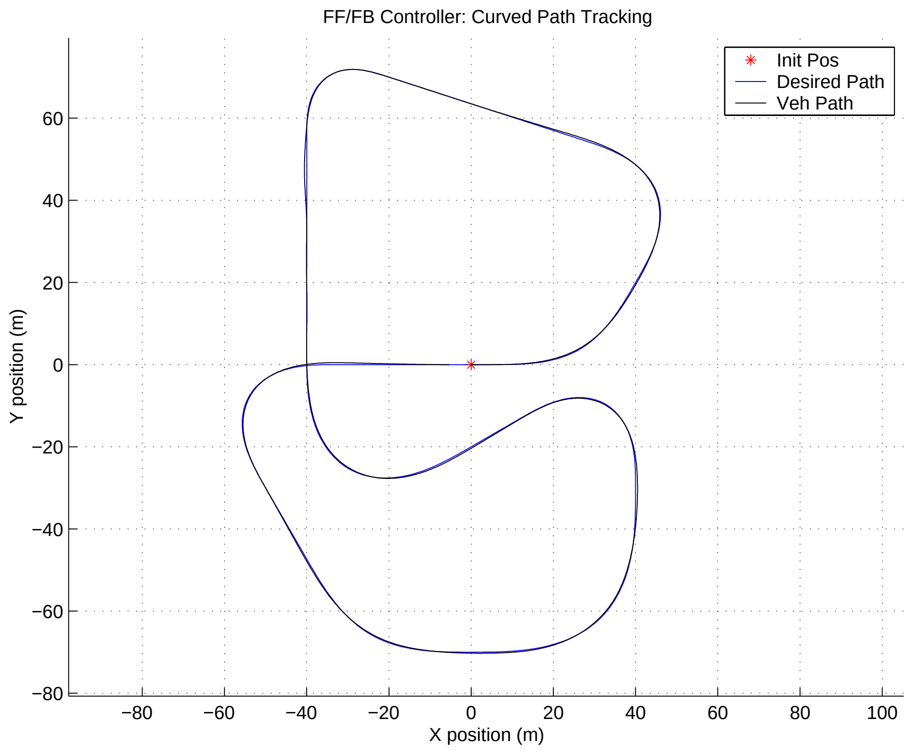


Figure 4.17: FF/FB Controller: Curved Path Following

All measures of tracking accuracy, for both position and heading errors, are improved using a FF/FB controller. The distance errors, in particular, are greatly improved

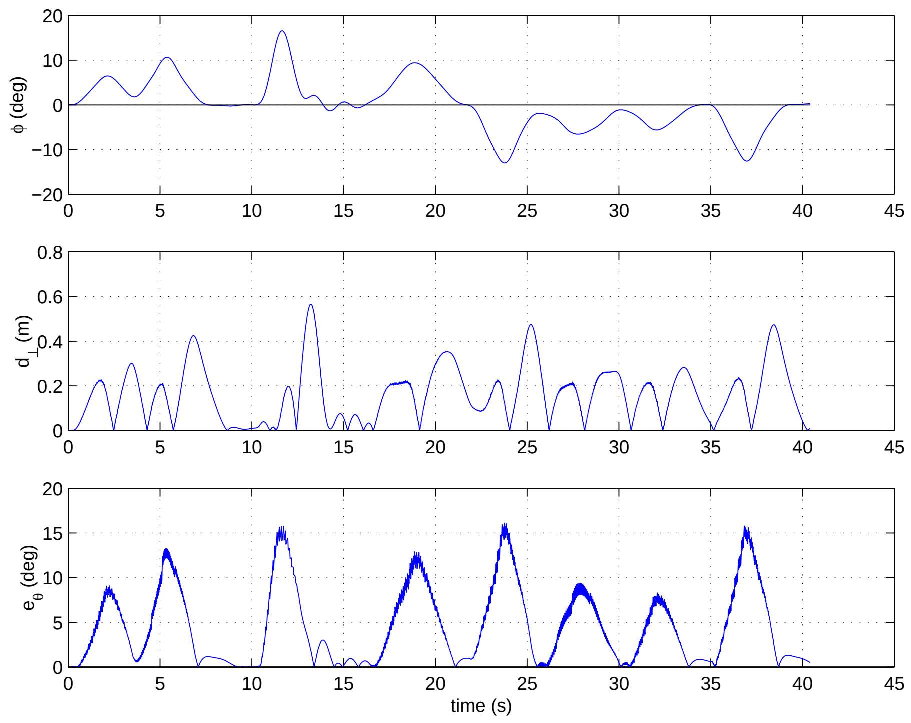


Figure 4.18: FF/FB Controller: Curved Path Following - Steering and Error Results

from a mean error of 1.1m to only 0.2m. Tracking improvements aside, the steering action is, as desired, very smooth, mirroring the continuous curvature profile of the path itself.

4.6 Conclusion

In this chapter, the developmental work performed for the path following (control) element of the system was outlined. The overall control system required was identified as possessing two components: 1) a low level (steering) controller, and 2) a high level path tracking controller. A number of approaches to vehicle control were reviewed and initially a simple, feedback based, proportional path following controller was implemented. Simulation results (using a steering model based upon ute steering data) found this control law unsuitable for the tracking of continuous curvature paths, hence a feedforward component was added to the original controller. This addition significantly improved path tracking in both the simulations carried out, and for the actual system implementation.

Chapter 5

System Implementation and Performance

5.1 Introduction

In the previous three chapters the theoretical basis for the path planning and following system was introduced. Methods for path planning, obstacle avoidance and path tracking for a car-like robot have been developed and their operation verified through the use of computer based simulations. This chapter describes how these three key aspects of an autonomous vehicle implementation are combined to produce a fully functional system that meets the thesis objectives.

Firstly, this chapter presents how the system is realised in software, included is an explanation of how the path planning and tracking module communicates with the obstacle detection and characterisation module (Lee 2001). This is followed by a description/demonstration of the user interface for the path planning and tracking system. Finally, experimental results obtained during field trials of the system are both presented and analysed.

5.2 Software Implementation and Structure

The prototypic nature of the path planning and following system developed in this thesis (prior to this thesis none of these capabilities were readily available within the

HSV project), meant that a complete real-time implementation, in a language such as C/C++ using proper real-time programming constructs, was not possible within the project's time frame[1]. Hence for programming expediency and ease of debugging, software development and implementation (for non-time critical tasks) was performed using the MATLAB programming language[2].

However, tasks which are time critical or require deterministic/synchronised execution, such as recording data from the sensors (for e.g. the laser scanner and the GPS returns) and controlling the actuators (for e.g. the ute's steering angle) is managed by a front-end program already implemented in Hyperkernel[3], a real-time operating system for Microsoft Windows NT. Whenever a Hyperkernel application, such as the front end, is running a block of 'shared memory' is initialised which the user can write to, or read (in real time) from within Hyperkernel, this memory block can also be accessed/modified by non-Hyperkernel applications (Nebot 2001). The front-end publishes all the updated sensor data at specific (predefined) locations within this shared memory, while the actuators read their control inputs from another set of shared memory locations. A screen shot of the Hyperkernel front-end interface, showing the outputs of the various onboard sensors and actautors is displayed in Figure 5.1.

To allow our MATLAB development/simulation software to access and control the ute's sensing and actuation capabilities a means of accessing the Hyperkernel front end from MATLAB was required. This took the shape of a MATLAB MEX[4] function called 'jo6'[5]. The premise behind this function was simple, basically it allowed a program written in MATLAB to access and modify the data published in the shared memory by the Hyperkernel front end. Hence any MATLAB program could now interact with the sensing and actuation systems aboard the ute.

Such a function meant that simulation software could now be modified for testing on the vehicle with relative ease. The software implementation issues/changes regarding each of the three system aspects are examined in turn:

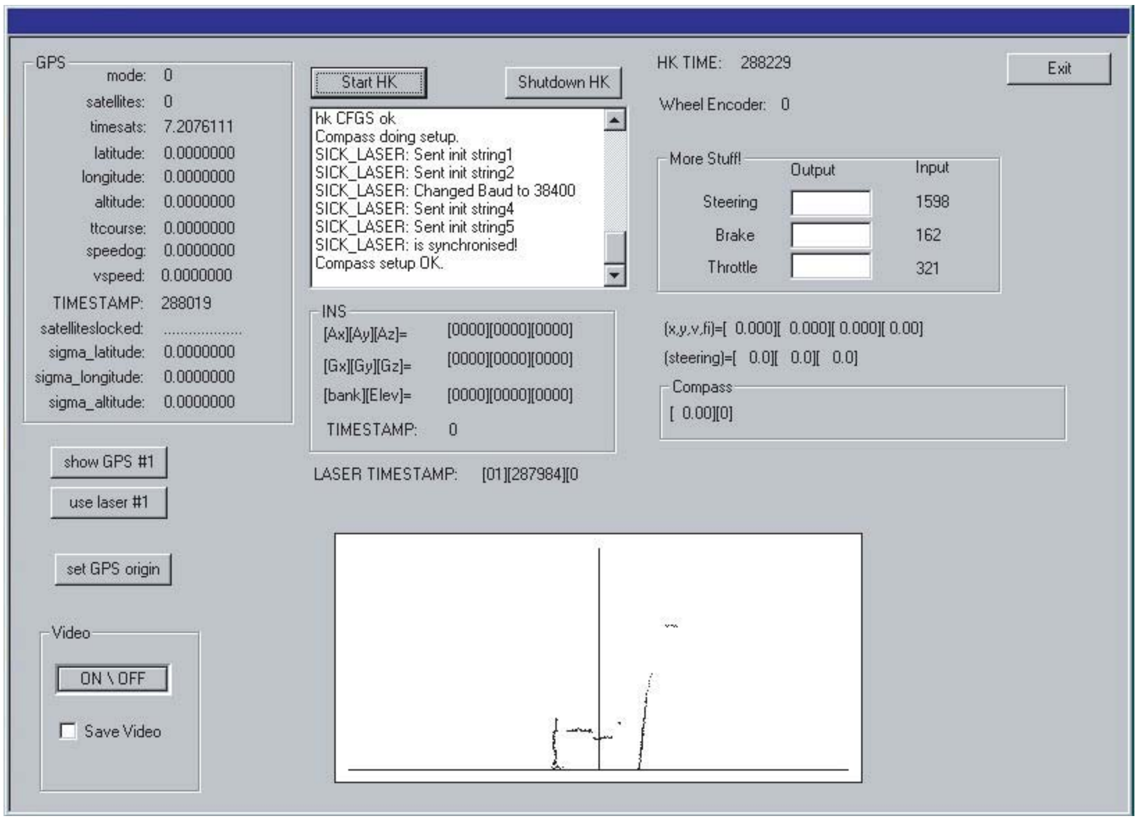
[1]A C/C++ implementation of the system is tendered as a potential task for 2002 (see Chapter 6).

[2]The author is aware that compared with software written purely in C/C++, MATLAB code executes considerably more slowly (typically a factor of 50-100 times).

[3]This program and its subsequent revisions have been written by HSV project Ph.D. student José Guivant.

[4]A MEX function is a function written in C/C++ which is compiled for use in the MATLAB programming environment.

[5]This function was written in early/mid 2001 by José Guivant.



```
GPS
mode:             0
satellites:       0
timesats:         7.2076111
latitude:         0.0000000
longitude:        0.0000000
altitude:         0.0000000
ttcourse:         0.0000000
speedog:          0.0000000
vspeed:           0.0000000
TIMESTAMP:        288019
satelliteslocked: ..................
sigma_latitude:   0.0000000
sigma_longitude:  0.0000000
sigma_altitude:   0.0000000

hk CFGS ok
Compass doing setup.
SICK_LASER: Sent init string1
SICK_LASER: Sent init string2
SICK_LASER: Changed Baud to 38400
SICK_LASER: Sent init string4
SICK_LASER: Sent init string5
SICK_LASER: is synchronised!
Compass setup OK.

INS
[Ax][Ay][Az]=    [0000][0000][0000]
[Gx][Gy][Gz]=    [0000][0000][0000]
[bank][Elev]=    [0000][0000][0000]
TIMESTAMP:       0

LASER TIMESTAMP:   [01][287984][0

HK TIME:   288229
Wheel Encoder:   0

More Stuff!
            Output   Input
Steering             1598
Brake                162
Throttle             321

(x,y,v,fi)=[ 0.000][ 0.000][ 0.000][ 0.00]
(steering)=[  0.0][  0.0][  0.0]
Compass
[ 0.00][0]
```

Figure 5.1: Hyperkernel Front End

- Path Planning - The path entry software remained largely unchanged, except that it is placed inside an interactive GUI (discussed in section 5.3). A trajectory is still specified as a series of (x, y, θ) coordinates, however, it now always starts at the centre of the local map with the following pose $(0, 0, \pi/2)$, this is the vehicle's initial pose.

- Path Following - Implementation of the path following controller required the removal of the pose update and steering models, introduced in Chapter 4, from the simulation software. Vehicle pose is now obtained using a navigation loop, this obtains pose information from one or a number of navigation sensors (for e.g. GPS, compass, dead reckoning). The output of the high level controller is now sent directly to the low level, PID steering controller which drives the front wheels. Controller gains must also be provided when the steering controller is initialised.

- Obstacle Avoidance - This requires knowledge of the obstacles present on or near the current trajectory to be passed from the obstacle avoidance module to the path planning and tracking system so that an appropriate avoidance decision can be made. To determine whether an obstacle represents a collision threat the obstacle detection module requires the vehicle's current pose and the trajectory over the next 40-50m. This data must be sent from the path following module.

Therefore an intermodule communications system was required to pass the relevant information between the two software modules (see Figure 5.2). The communications system was implemented by extending the functionality of 'jo6'. Since this function could already read from, and write to, Hyperkernel's shared memory it was decided that two additional read and write options would be added. The obstacle and vehicle pose/path information would be published in a region of the shared memory specifically allocated for the task. The data would be written to the appropriate location by one module and the other would check every so often if any new data had been published, if new data is available the module will access it and perform the necessary processing. The results of any processing would then be sent back to the shared memory.

The pose and trajectory information is sent by the path tracker as a 3x101 matrix, containing the vehicle's pose and the next 100 (x, y) points of the path, if available[6]. The obstacle detection module uses the vehicle's current pose to transform the position of any obstacles from the vehicle's frame of reference (f.o.r) to the local map's f.o.r. Then the obstacles (now in local map coordinates) are checked for collisions with the path. If any collisions are detected the obstacles of interest (up to 50) are sent to the appropriate location in the shared memory. The current obstacle memory block is interrogated during each iteration of the control loop within the path following module. If any obstacles are present, normal execution is suspended and the path replanning routine is performed[7]. The modified trajectory is now published in shared memory.

In summary, to run the path planning and following system with the obstacle avoidance capabilities enabled, three software modules are required[8]:

[6]A time stamp associated with each position is also provided, this would be used for detecting potential collisions with moving obstacles. However, the system implementation under its current guise does not support this.

[7]Note: like the path planning (entry) module the trajectory replanning for obstacle avoidance remains largely unchanged, from that developed in Chapter 3.

[8]The two MATLAB programs are run independently on separate MATLAB applications. Each module running in MATLAB must have access to the updated (inter MATLAB communications) version of the 'jo6' function.

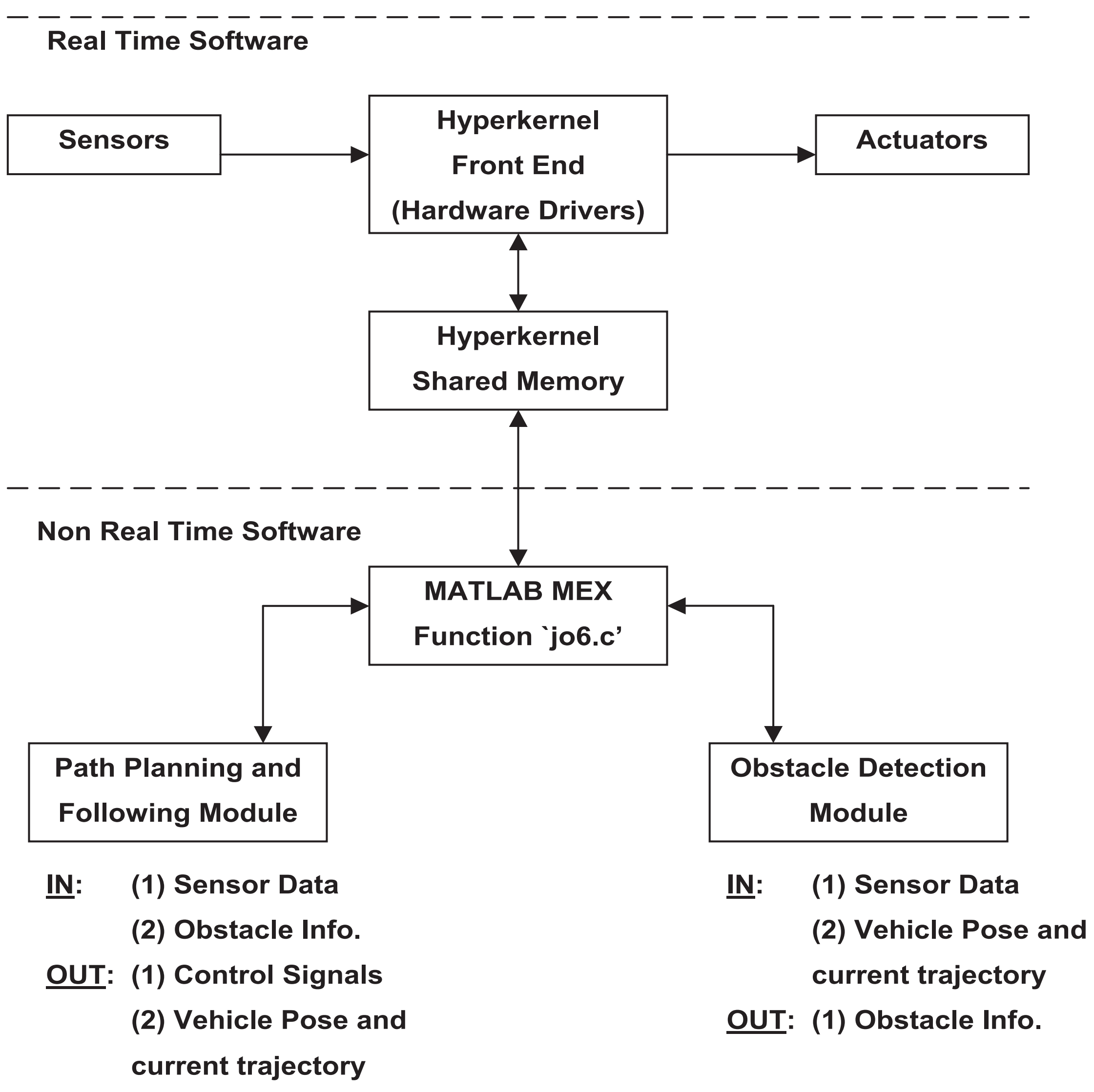

Real Time Software
Sensors
Hyperkernel Front End (Hardware Drivers)
Actuators
Hyperkernel Shared Memory
Non Real Time Software
MATLAB MEX Function `jo6.c'
Path Planning and Following Module
Obstacle Detection Module
IN: (1) Sensor Data (2) Obstacle Info.
OUT: (1) Control Signals (2) Vehicle Pose and current trajectory
IN: (1) Sensor Data (2) Vehicle Pose and current trajectory
OUT: (1) Obstacle Info.

Figure 5.2: Software configuration and intermodule communications

- Real Time Sensing and Actuation Front End (Hyperkernel)
- Path Planning and Tracking System (MATLAB)
- Obstacle Detection Module (MATLAB)

The complete software library for the system is included on the attached CD (see back inside cover of thesis). In the next section another aspect of the system implementation, the user interface, is discussed.

5.3 Graphical User Interface

In addition to successful software implementation and integration another requirement of the system (see Thesis Objectives, Chapter 1) was that the user be provided with an interface which offers a convenient and clear means of: 1) entering a path and 2) operating the path following system. The path planning and tracking GUI (see Figures 5.3-5.8) provides three main functions (activated by the three upper buttons in lower right corner) which are discussed in turn:

- Path Entry - When this function is run an interactive version of the path generation software developed in Chapter 2 is executed. The path's starting waypoint (and the vehicle's initial pose) is set to the centre of the local map $(0, 0, \pi/2)$[9]. Around this waypoint an inaccessible region (bounded by the green lines) is plotted. The edges were obtained from the numerical general solutions presented in Chapter 2 (for CCPP Technique 2) for typical maximum sharpness σ_{max} ($1/64\ m^{-2}$) and curvature κ_{max} ($0.1\ m^{-1}$) values[10]. Any point inside this region does not possess a feasible solution, or the feasible solution is over a very narrow orientation range, hence any mouse click here is ignored (see Figure 5.3).

  When a waypoint outside the inaccessible region is entered, a general solution function is run. This calculates the upper and lower orientation bounds for the waypoint for which a feasible clothoid/smooth Reeds and Shepp segment can be fitted between the new and previous waypoints. These bounds are displayed as two red lines extending to the map boundary, as shown in Figure 5.4.

  The next point entered by the user must lie in the region between the upper and lower orientation bounds (i.e. in the non reflex angle). When a point is clicked within this area, a straight line is formed between the waypoint and the site of the mouse click. The orientation of this line is assigned to be the orientation of the new waypoint. Once the new waypoint's pose has been specified, the path generating function is executed[11] so as to create the next path segment between the two waypoints, which is then appended to the current path (x, y) vectors. The inaccessible region is replotted based upon the pose of the updated current

[9]This initial orientation was selected as it means the local map presented by the GUI is always aligned with the vehicle itself. This makes path entry easier for the user.

[10]The green lines approximate the edges of the flat regions of the general solution shown in Figure 5.3

[11]For further details, see Chapter 2 or the path generation software included on the attached CD.

waypoint (Figure 5.5). To add further path segments, the process is simply repeated[12]. Six consecutive screen shots (5.3-5.8) of the path generation GUI, demonstrating the creation of a closed path, are included on the following pages.

The system user is able to specify an initial map size. However, if the path is detected to be within the vicinity of the map's edge, then the path entry screen is rescaled. If a closed path is entered, the number of laps over which to track the trajectory can be specified. While the system's GUI is running, the last entered path remains available, and hence can be reused, possibly for retracking from a new starting location. Finally, the characteristics of the continuous curvature paths can be altered by modifying the minimum allowable clothoid coefficient a_{min} (which determines the maximum path sharpness σ_{max} and the minimum radius of curvature ρ_{min} which are inputs into the CCPP software).

- Set Origin - As the name suggests, this function is run to initialise the pose of the vehicle, that is, to provide an origin for the local map. In the current implementation, the vehicle's pose is obtained using GPS 'fixes'. Upon execution, the driver is allowed 7 seconds to move the vehicle forwards (at a speed greater than 1 ms^{-1}) to obtain an accurate estimate of the vehicle's current pose[13]. After this, (i.e. while tracking the path) all GPS fixes are transformed into local map coordinates relative to this point (this places pose estimates into the same f.o.r as those of the path).

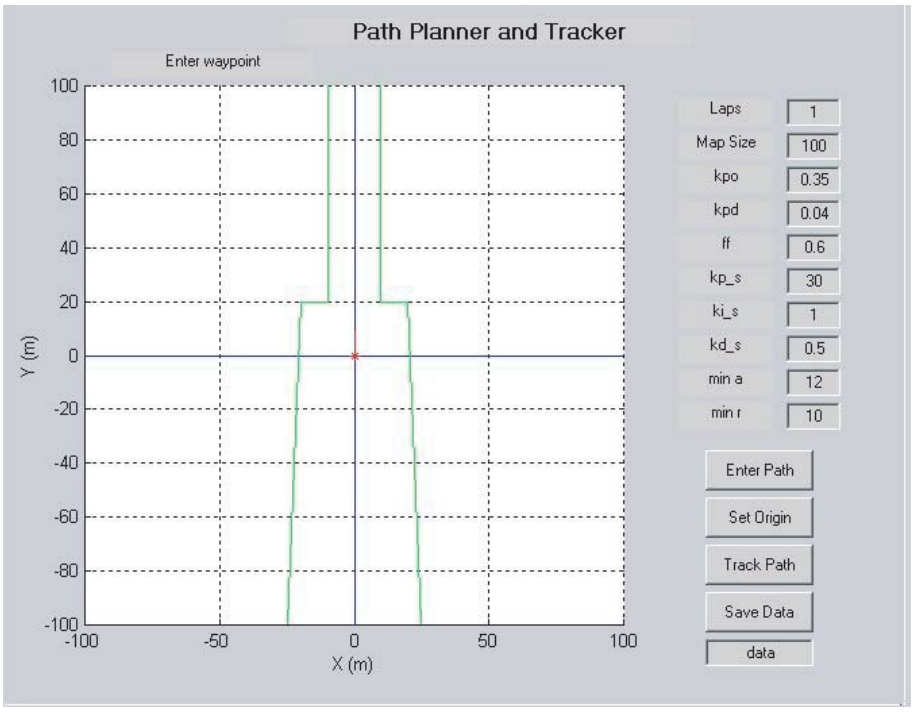
- Track Path - Once both a path and a vehicle origin have been specified, the proposed trajectory can be tracked. Edit boxes for setting both the high and low level controller gains are available. When activated, this program runs the steering control loop[14] for path following, the vehicle's pose is constantly updated, and the appropriate steering action is made. Tracking continues until the end of the path is reached.

The results of the extensive field tests carried out to implement and test this system are detailed in the following section.

[12]The user can opt to terminate a path, or if the path is closed, i.e. becomes a circuit, the path entry program automatically terminates.

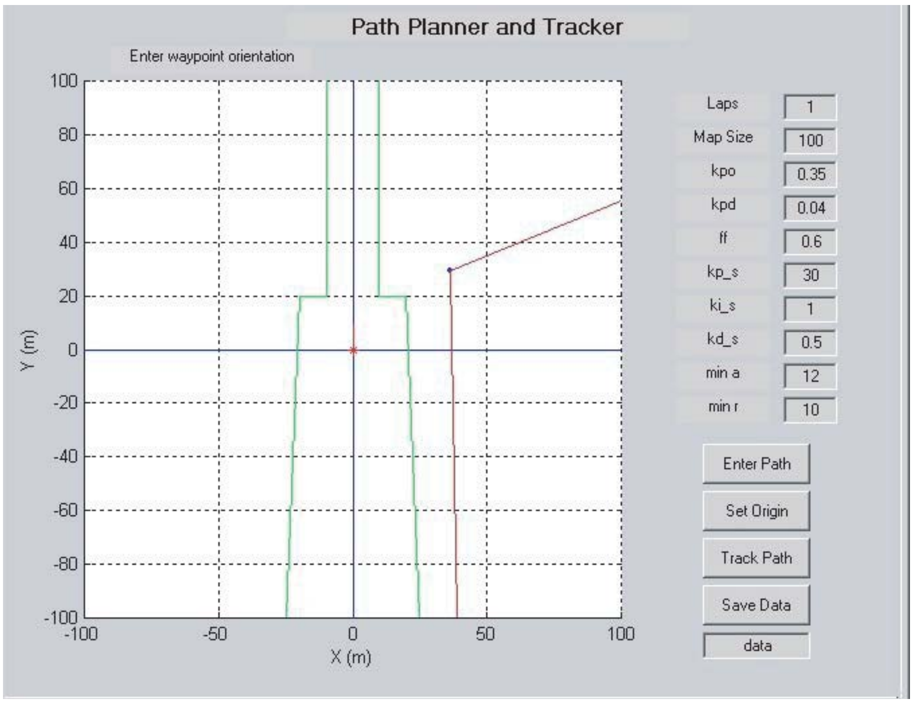
[13]The vehicle must be moved, since GPS heading estimates can only be considered reliable when the vehicle is moving above a certain speed.

[14]At this stage a driver provides the throttle and brake control.



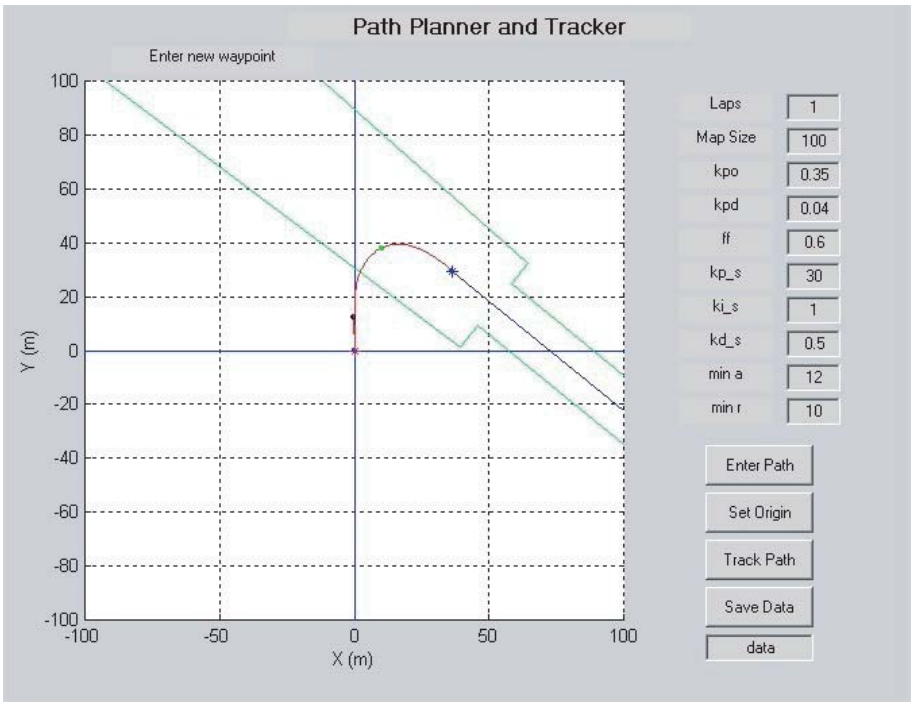
Laps	1
Map Size	100
kpo	0.35
kpd	0.04
ff	0.6
kp_s	30
ki_s	1
kd_s	0.5
min a	12
min r	10

Figure 5.3: GUI: Path Entry - Step 1



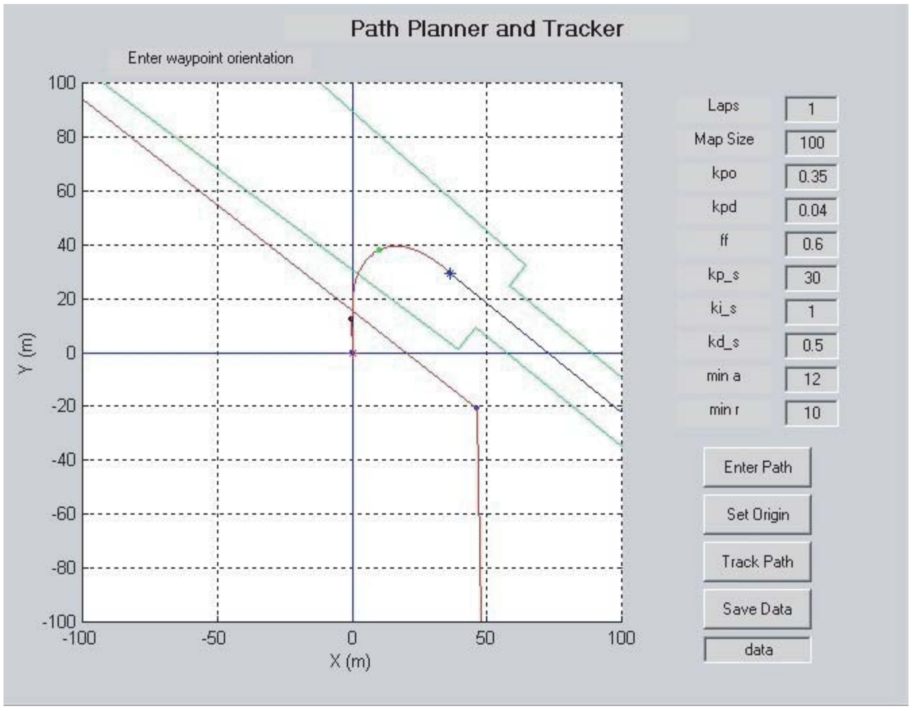
Laps	1
Map Size	100
kpo	0.35
kpd	0.04
ff	0.6
kp_s	30
ki_s	1
kd_s	0.5
min a	12
min r	10

Figure 5.4: GUI: Path Entry - Step 2



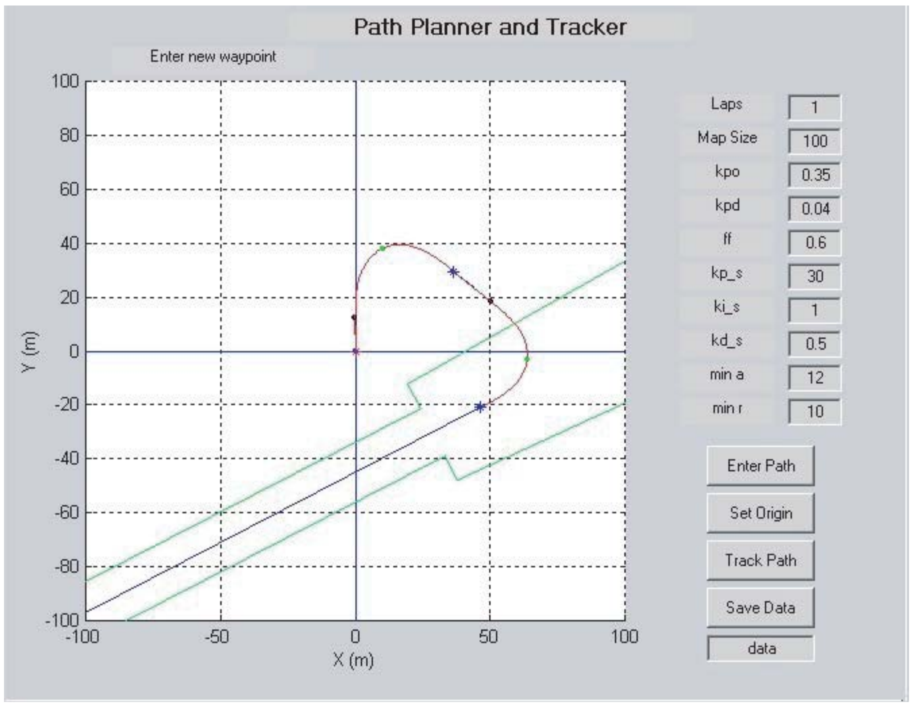
Laps	1
Map Size	100
kpo	0.35
kpd	0.04
ff	0.6
kp_s	30
ki_s	1
kd_s	0.5
min a	12
min r	10

Figure 5.5: GUI: Path Entry - Step 3



Laps	1
Map Size	100
kpo	0.35
kpd	0.04
ff	0.6
kp_s	30
ki_s	1
kd_s	0.5
min a	12
min r	10

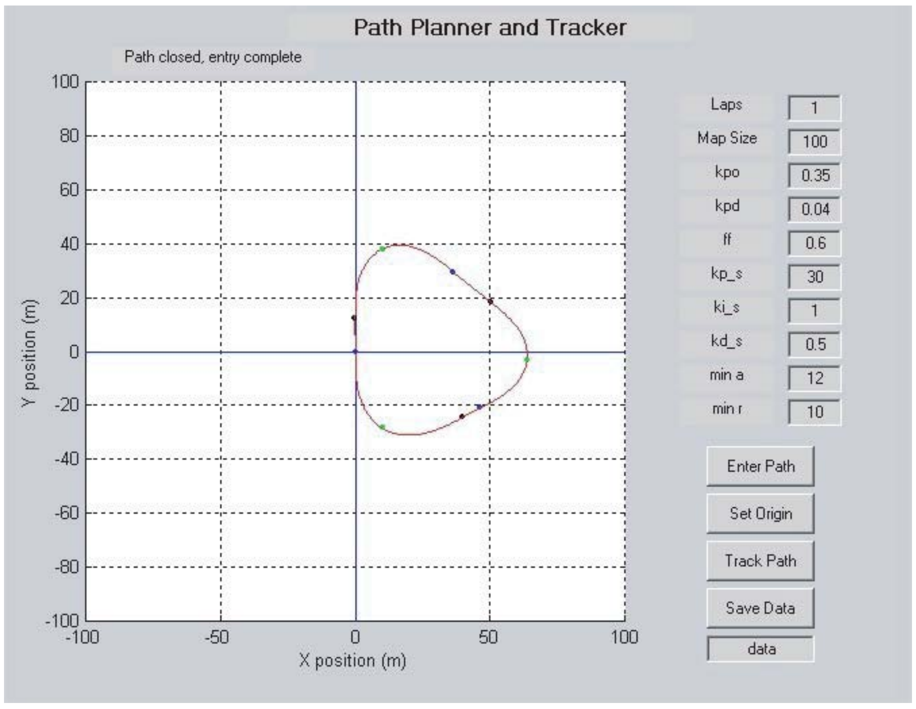
Figure 5.6: GUI: Path Entry - Step 4


Path Planner and Tracker

Enter new waypoint

Laps	1
Map Size	100
kpo	0.35
kpd	0.04
ff	0.6
kp_s	30
ki_s	1
kd_s	0.5
min a	12
min r	10

Figure 5.7: GUI: Path Entry - Step 5


Path Planner and Tracker

Path closed, entry complete

Laps	1
Map Size	100
kpo	0.35
kpd	0.04
ff	0.6
kp_s	30
ki_s	1
kd_s	0.5
min a	12
min r	10

Figure 5.8: GUI: Path Entry - Step 6

5.4 System Testing

System implementation and field testing, as stressed in the thesis objectives, represents a critical aspect of the work undertaken. The technical work presented thus far is only part of the task, it must also be made to work on the vehicle to provide a reliable system for those already working within the HSV project, and to act as a foundation stone for future project development.

So, once the software modules had reached the desired level of robustness in simulated operation, they were converted into the 'on vehicle' implementation just described. Testing and demonstration[15] of the system took place at a number of venues around the university (much to the bemusement of onlookers) including, the college ovals of St. Andrew's and St. Paul's and 'The Square'. The results of these tests are now presented.

5.4.1 Steering Control

As described in Chapter 4, a PID steering controller was already in place on the ute at the start of this thesis but required both tuning and software modifications. Tuning was performed by varying the PID gains until an adequate steering response was obtained. The steering tracked paths more than accurately with the following gains:

- $k_p = 15$
- $k_i = 0.5$
- $k_d = 0.5$

This set of controller gains was used to give the steering response shown in Figure 4.10, upon which the steering model was based. Figure 5.9 shows the steering controller tracking a sine wave of amplitude 10 deg and period 20s. The steering controller tracked this setpoint profile quite accurately, but one feature is very noticeable - the controller tends to turn the steering wheel in discrete (approximately 2 deg) steps (this gives rise to the 'stair-like' pattern shown), rather than one smooth continuous

[15]The path planning and tracking system was demonstrated to visitors from the Singaporean Defence Department on 28th October 2001.

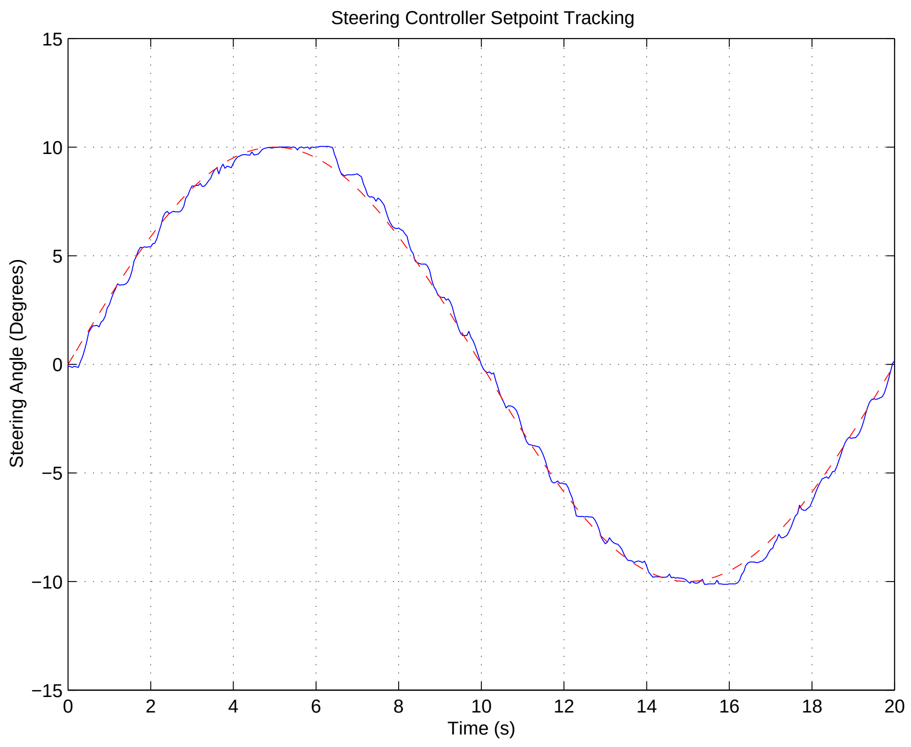


Figure 5.9: Steering Controller: Setpoint Tracking

motion. This manifests itself most noticeably in the path tracking results where the steering controller tends to provide only large steering angle changes, rather than a smooth turn of the wheel.

Hence, even for a 'good' set of controller gains, it was concluded that completely smooth steering control was not possible due to non-linearities in the steering mechanism itself (note that any steering response is not only a function of the PID gains but also the setpoint change required, the current position of the wheel, friction and inertia of the steering hardware etc). To remedy this problem in future, it is recommended that a model-based controller be implemented that accounts for both these non-linearities and the dynamics of the steering mechanism.

5.4.2 Localisation

An imperative for any path tracking system, indeed any ALV implementation, is the ability to localise itself. For this system, the navigation loop which provides the current vehicle pose used standard GPS[16] (frequency 5 Hz, nominal error 0.5-

[16]It was hoped that compass readings could be integrated into the system, so that initial orientation could be acquired without moving the vehicle, but despite numerous mounting locations and calibrations, it was found to have a significant non-linear offset with respect to the GPS heading.

1m). As the tracking results suggest, despite it's low frequency, GPS fixes provide reliable estimates of both position and heading[17]. However, GPS operates best in 'open environments' where inaccuracies due to multipathing effects (the result of GPS signals bouncing off the local environment, such as trees and buildings) are minimised. These effects appear as vehicle position 'jumps', examples of which can be seen on the right side of Figure 5.14, a path tracking test performed on St. Andrew's Oval, where one side of the trajectory was bordered by a row of trees.

In future, and particularly if increased vehicle speeds are involved, a complete high frequency, multisensor navigation loop will be required (for both control accuracy and safety reasons). However, this could be plugged directly into the current software and has been proposed as an area for development in 2002.

5.4.3 Path Following: Line Segment Paths and FB Control

With both the steering controller and a 'simple' navigation loop set up, the first implementation of the path planning and following system was tested to verify the FB controller's behaviour. Paths were constructed by the user as a series of straight line segments, and the results of two trajectory tracking tests are shown in Figures 5.10, 5.11, 5.12 and 5.13.

The FB controller worked well, but again the inability of a car-like robot to accurately track a path consisting solely of line segments is highlighted, as the vehicle tracks the path with a positional error of approximately 2-5m (Figures 5.10 and 5.12). Steering control (Figures 5.11 and 5.13) is also quite aggressive, requiring rapid, large angle ($\phi \geqslant 15$ deg) turns of the wheel, which is undesirable. Higher frequency oscillations are also present within the steering output due to the step-wise nature of the steering wheel movements (as discussed in section 5.4.1). Path tracking performance is vastly improved by the introduction of continuous curvature paths and FF/FB control, as demonstrated in the next section.

[17]Simulations were run where the vehicle's pose was only updated every 200ms, and showed, as witnessed in the field tests, that path following would remain stable under these conditions. Also, the orientation was only updated if the vehicle was travelling above 1m/s.

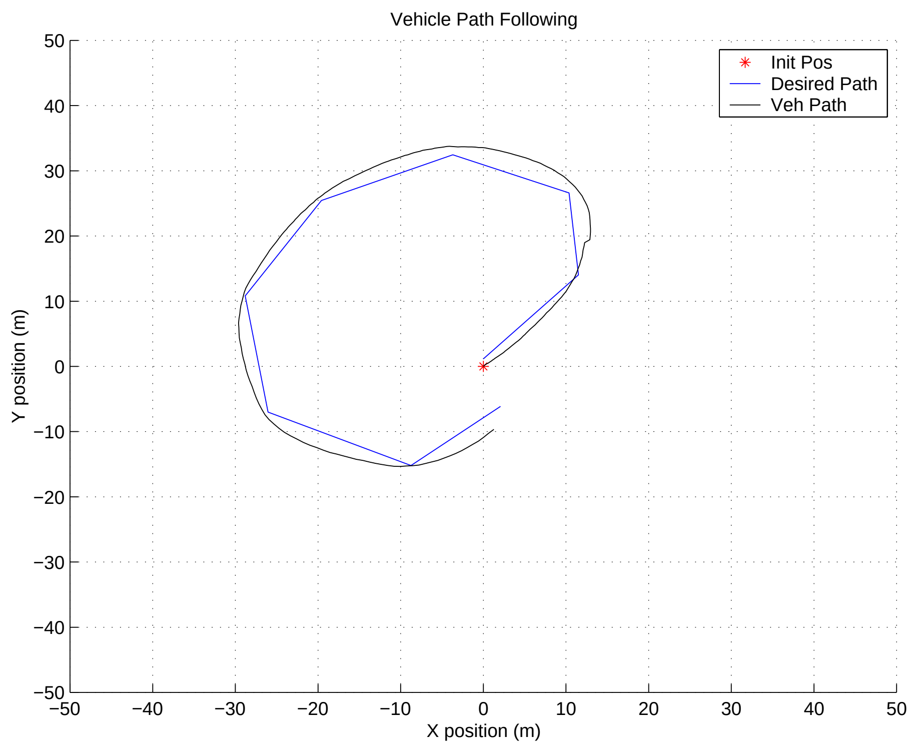


Figure 5.10: FB Controller: Straight Line Path Following

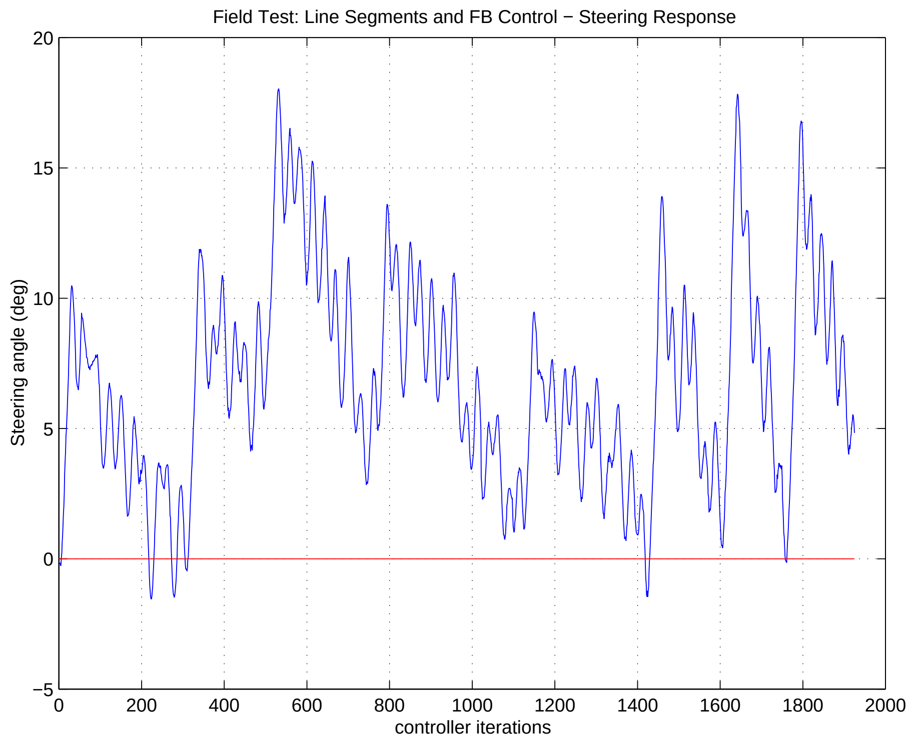


Figure 5.11: FB Controller: Straight Line Path Following - Steering Angle

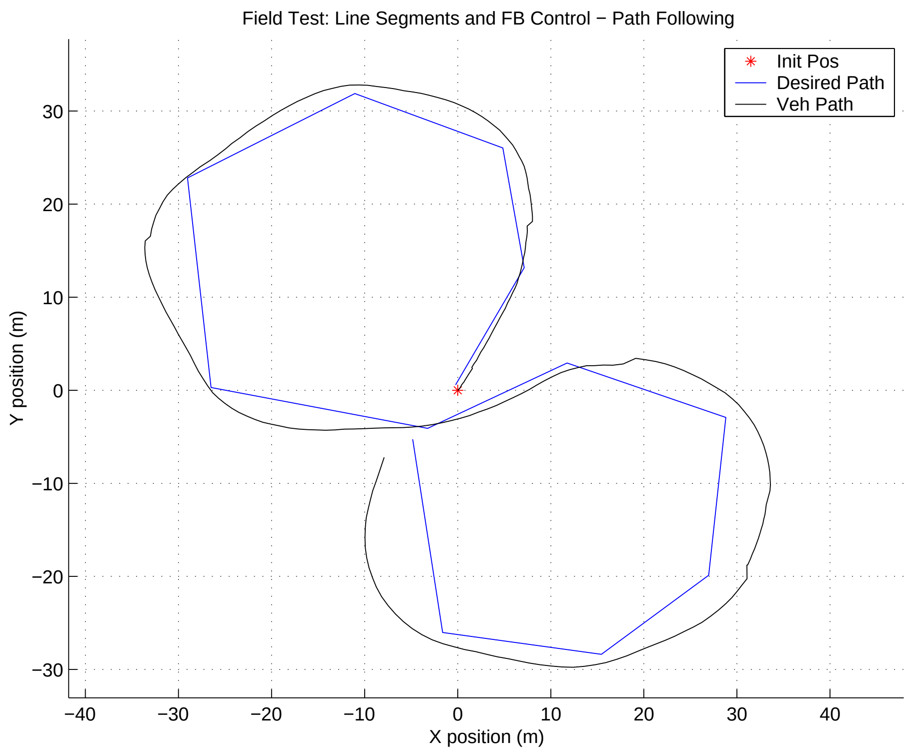


Figure 5.12: FB Controller: Straight Line Path Following

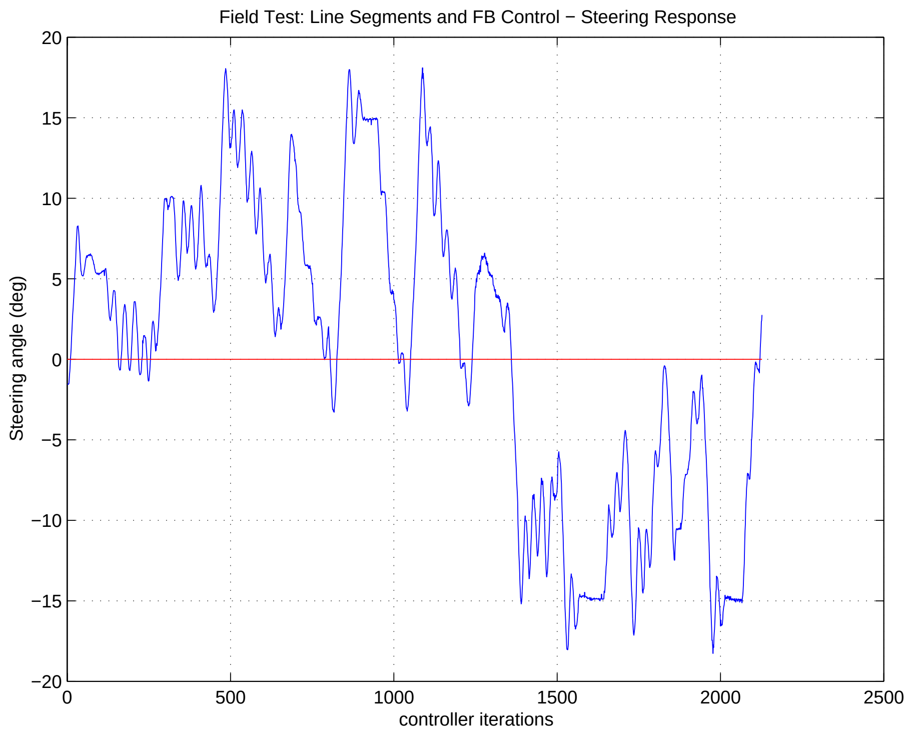


Figure 5.13: FB Controller: Straight Line Path Following - Steering Angle

5.4.4 Path Following: CCPP and FF/FB Control

This section presents the field test results for the tracking of two continuous curvature trajectories ('Test 1' and 'Test 2') generated by the path entry user interface[18] (see Figures 5.14, 5.15, 5.16 and 5.17)[19]. The error statistics (maximum, mean and standard deviation for both the absolute position and heading errors) have been calculated and are listed in Table 5.1. These statistics reinforce the observation from the tracking graphs (Figures 5.14 and 5.16) that a striking improvement over the previous tracking results has been achieved. The tracking errors for both test runs (position errors of 0.44m and 0.54m and heading errors of 2.9 deg and 3.6 deg) now compare very favourably with those predicted by the FF/FB simulations[20] in Chapter 4. Repeatability was also a key requirement for the path tracking controller. This was verified by repeatedly tracking a closed path as performed in Tests 1 and 2. The minimal variation between consecutive laps is displayed in Figures 5.14 and 5.16 (and by the overlaid tyre tracks displayed in the oval's grass surface).

Error	**Test 1**	**Test 2**
$e_\perp$ max (m)	2.69	2.63
$e_\perp$ mean (m)	0.44	0.54
$e_\perp$ s.d. (m)	0.41	0.52
e_θ max (deg)	20.7	18.0
e_θ mean (deg)	2.9	3.6
e_θ s.d. (deg)	3.0	3.2

Table 5.1: CCPP and FF/FB Control: Field Test Results

The ute's steering response (while still showing some 'step-wise' response), was much smoother, more gradual and required smaller turns of the steering wheel[21]. This was one of the primary reasons for implementing continuous curvature paths - so that steering control could be made smoother and more applicable to operation at higher speeds (as will be required in future implementations).

[18]These tests were performed at relatively low speeds, during Test 1 the vehicle had a mean velocity of 9.8km/h (maximum of 16.7km/h) while in Test 2 the mean velocity was 14.8km.h (maximum of 26.7km/h).

[19]The results presented are for the following path tracking controller gains $k_\perp = 0.04$, $k_o = 0.35$ and $k_\kappa = 0.8$, while the PID gains for the steering controller remain as listed in section 5.4.1.

[20]The tracking errors are now within the noise of the pose estimates provided by the GPS.

[21]The steering angle for accurate path tracking should resemble the curvature profile of the path being tracked. In Figures 5.15 and 5.17 for Tests 1 and 2, respectively, the curvature profile for the trajectory can clearly be seen (as the triangular spikes) in the steering angle plots.

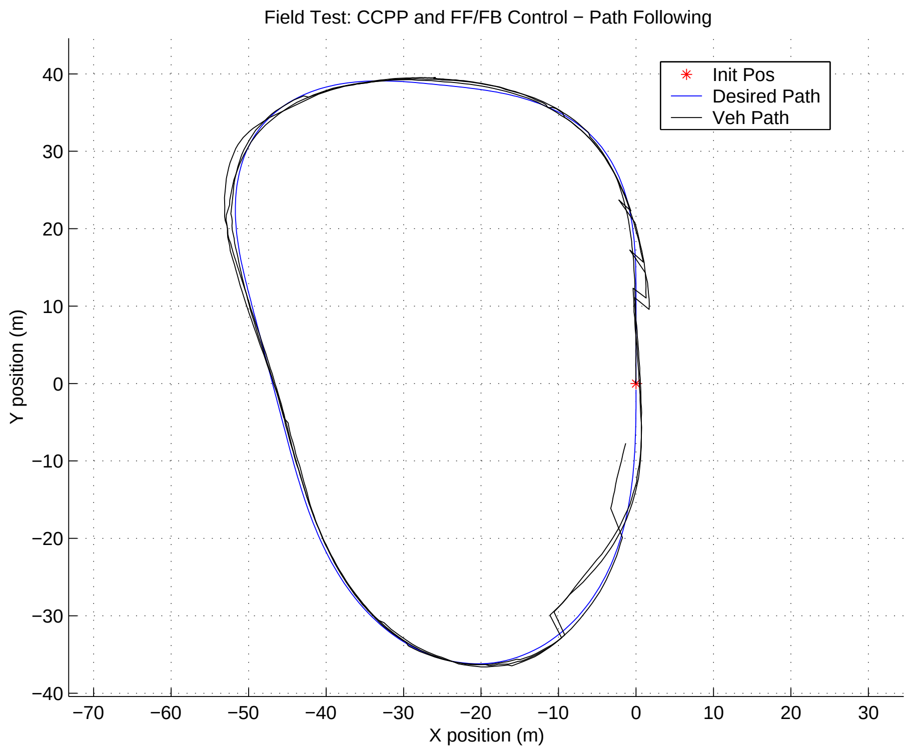


Figure 5.14: FF/FB Controller: ‘Test 1’

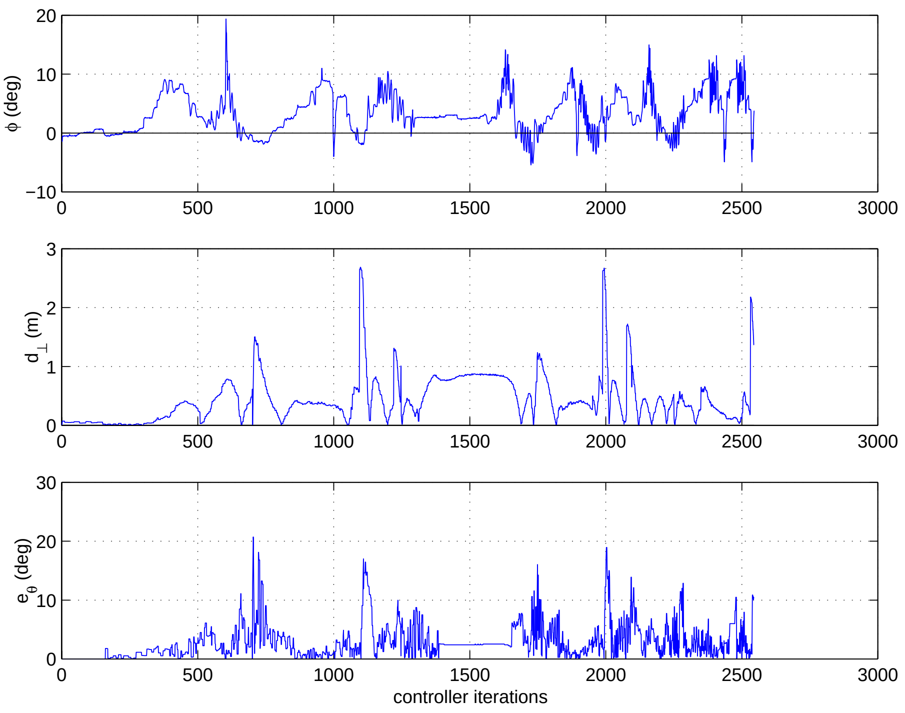


Figure 5.15: FF/FB Controller: ‘Test 1’ - Steering Angle and Tracking Errors

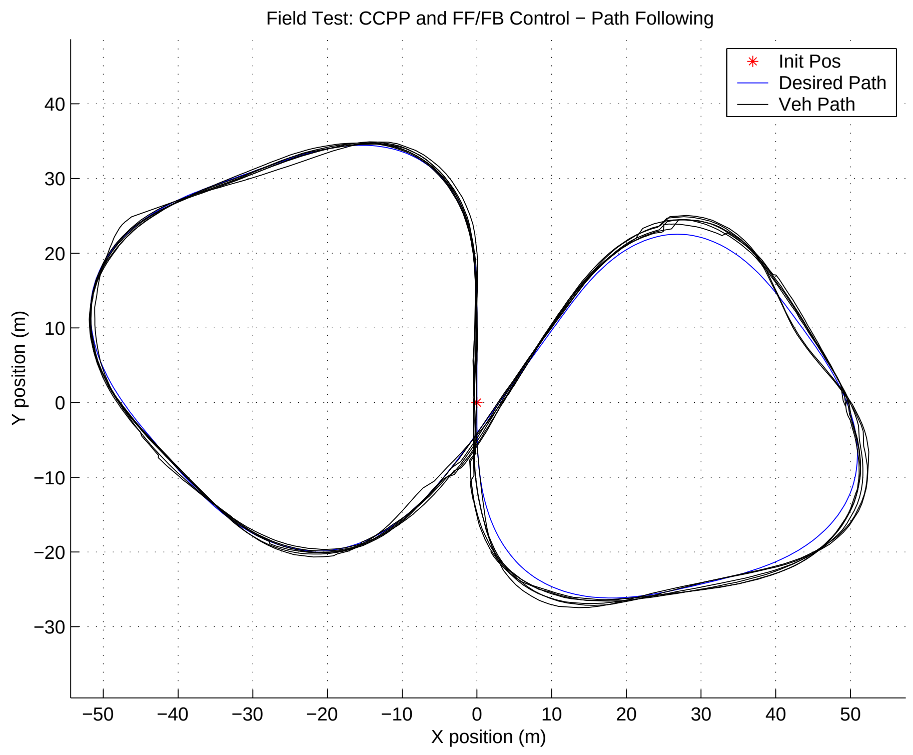


Figure 5.16: FF/FB Controller: ‘Test 2’

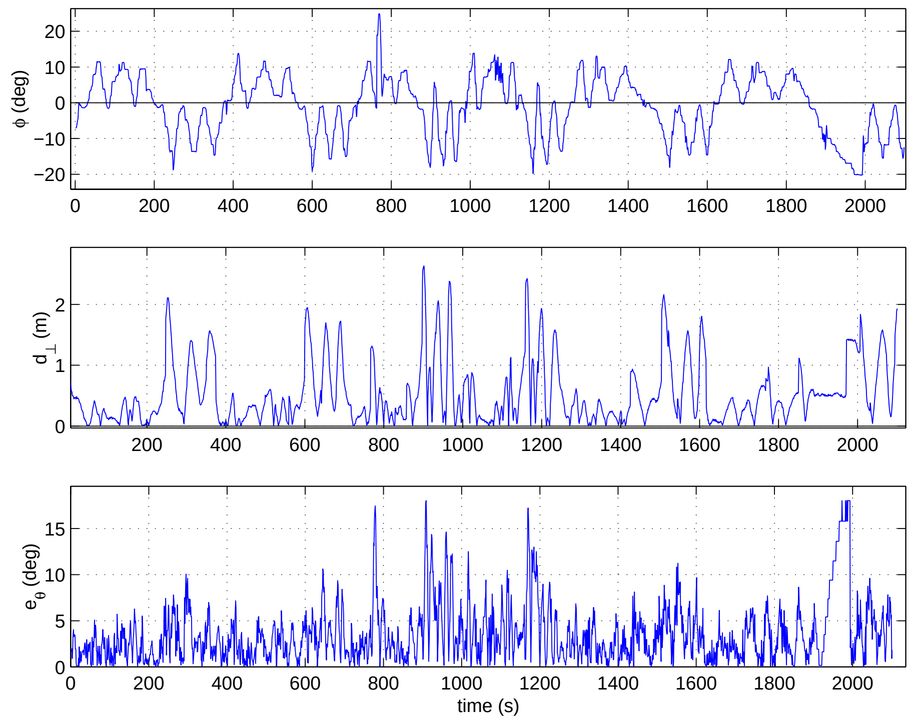


Figure 5.17: FF/FB Controller: ‘Test 2’ - Steering Angle and Tracking Errors

5.4.5 Path Following: Obstacle Avoidance

Once the path planning and tracking system had been successfully implemented on the ute, the final feature of the system's original specifications was introduced. This was the addition of obstacle avoidance/path replanning capabilities. To realise this objective, the path planning/tracking module and the obstacle detection module developed by Lee (2001) had to communicate with each other using the intermodule communications structure outlined in section 5.2.

In Chapter 3, a (near) continuous curvature path replanning technique was developed, but found to have an execution time too lengthy for real-time implementation from within the MATLAB environment. Hence, to demonstrate obstacle avoidance, a simplified approach was employed for this task. When an obstacle is detected on (or near) the planned path by the detection module[22], two waypoints, a safe distance either side of the obstacle are generated and sent to the path tracking module (see Figure 5.18).

This module now selects the avoidance waypoint that results in the least disruption to the original path, that is, the waypoint that causes the minimum deviation from the current trajectory[23]. A straight path is then generated from the vehicle's current position to the waypoint, and this 'temporary' path is followed instead of the user specified path. Once the waypoint is reached, the vehicle resumes tracking the original trajectory[24] A flow diagram showing the operation of the obstacle avoidance system has been included (see Figure 5.19).

The results of field tests for obstacle avoidance using this 'simple' technique have proved very promising, and three sets of successful avoidance manoeuvrers have been included (see Figures 5.20, 5.21 and 5.22) and are briefly outlined here. At a formal demonstration of this system, one of the tests was to have a person stand on the planned trajectory and have the vehicle detect and avoid that person. Figure 5.20 shows exactly this, the two obstacles being (thesis supervisor) Eduardo Nebot and (Ph.D. student) Trevor Fitzgibbons. The next avoidance manoeuvre shown was carried out at St. Andrew's oval, the four obstacles displayed being goal posts (see

[22]This system has been set up to handle single obstacles - for example, a person standing on the path.

[23]This was determined by calculating the shortest path between the current position, the waypoint and a point on the path 20m beyond the obstacle.

[24]Note that to avoid the vehicle side-swiping the obstacle when the original path is resumed, the avoidance path generated is extended past the obstacle.

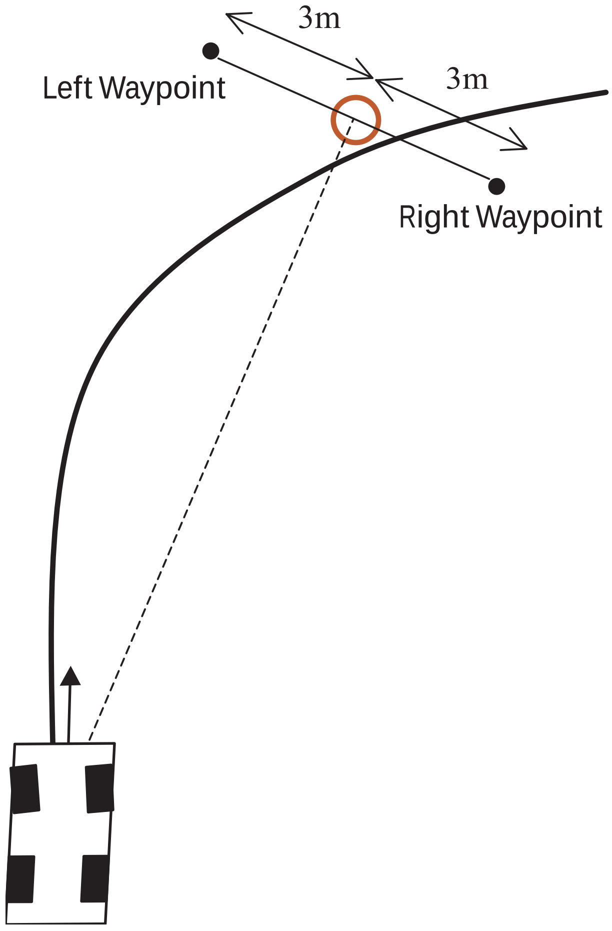


Figure 5.18: Obstacle Avoidance: Waypoint Specification

Figure 5.21). The system was able to generate and track a safe path between them and resume tracking the original closed path. The third and final example given (Figure 5.22) shows a 'wall-like' structure[25] being avoided.

5.5 Conclusion

Chapter 5 has detailed how the theoretical work of the previous three chapters may be combined to produce a complete path planning and tracking system that meets the original thesis objectives. Using Hyperkernel's shared memory, the path planning/tracking and obstacle detection modules (run through MATLAB) are able to communicate path and obstacle information in real-time, so that obstacle avoidance is possible. A robust GUI has also been developed for convenient operation of the system (for both path planning and following). Finally, a series of field trial results on

[25]Created using a number of laser beacons.

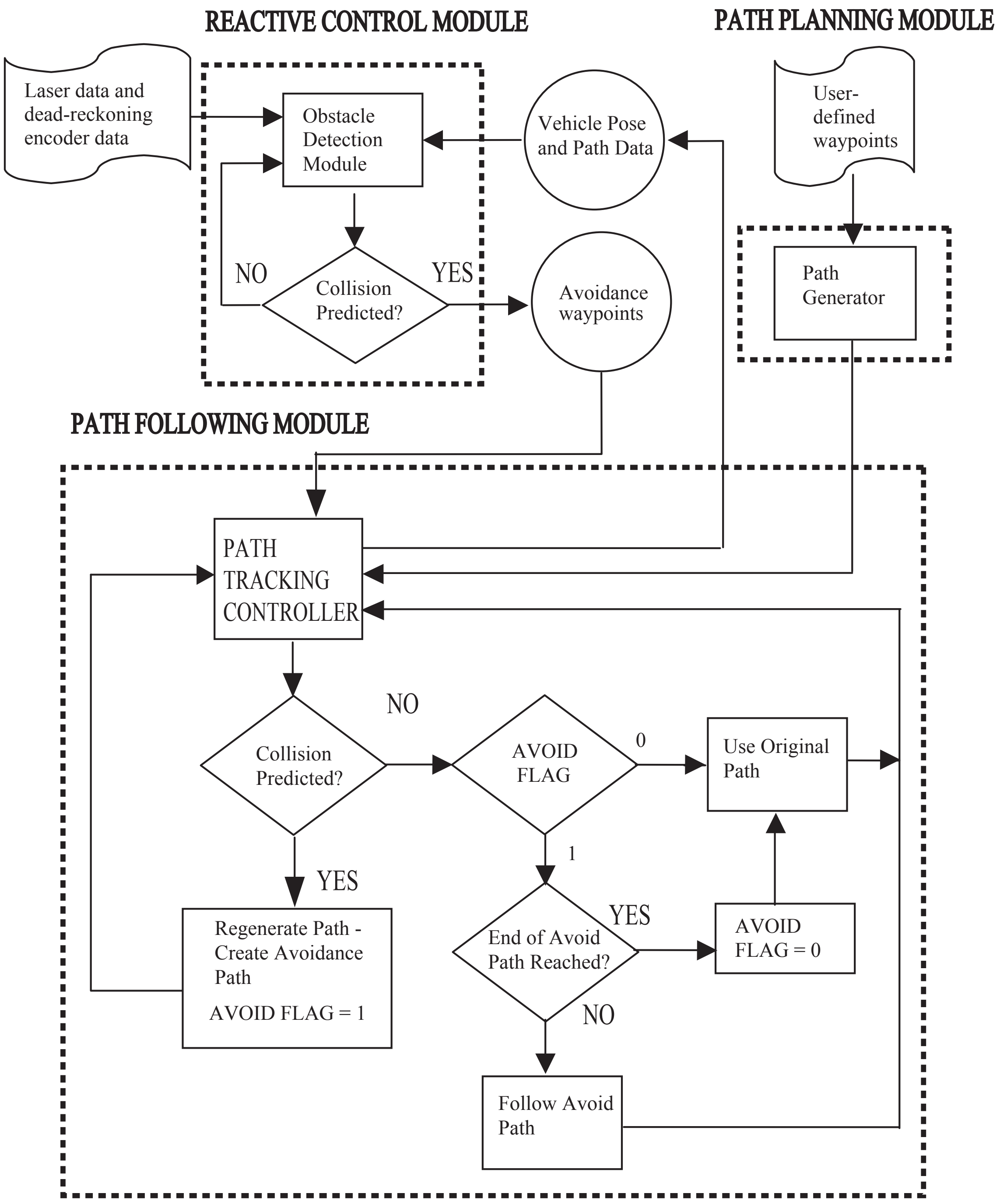


Figure 5.19: Path Planning/Tracking and Obstacle Detection System: Operational Flow Diagram

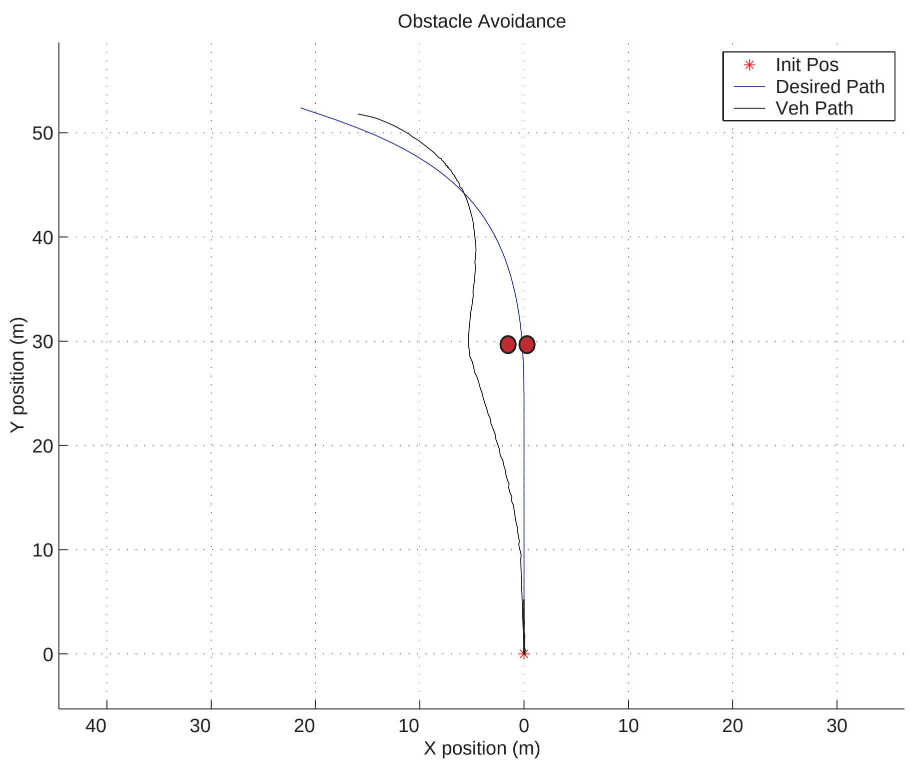


Figure 5.20: Obstacle Avoidance: Two people standing on path

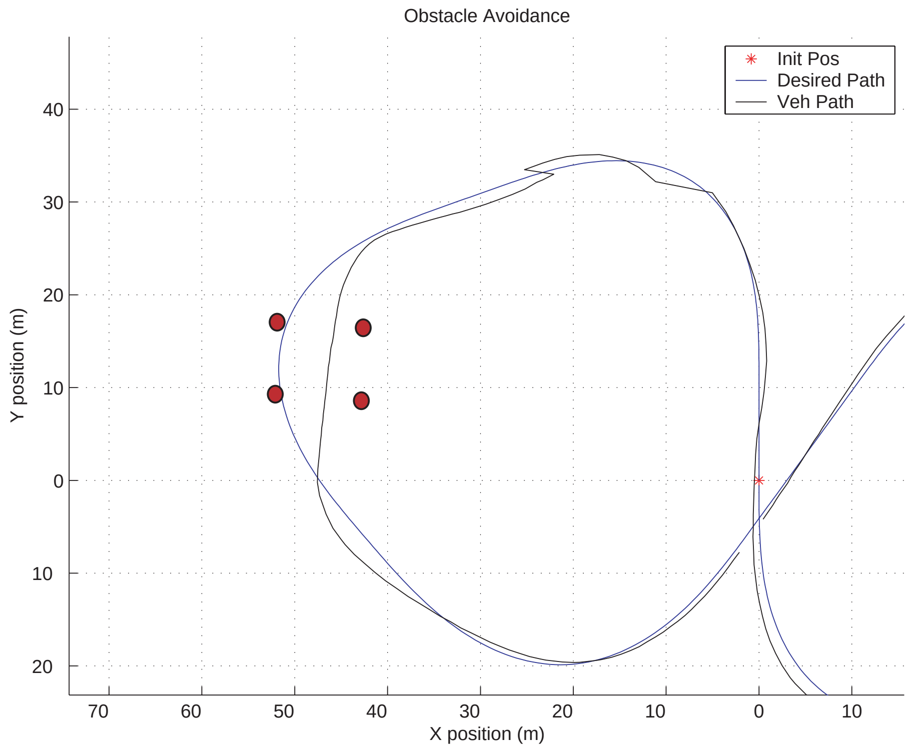


Figure 5.21: Obstacle Avoidance: Four goalposts on St. Andrew's oval

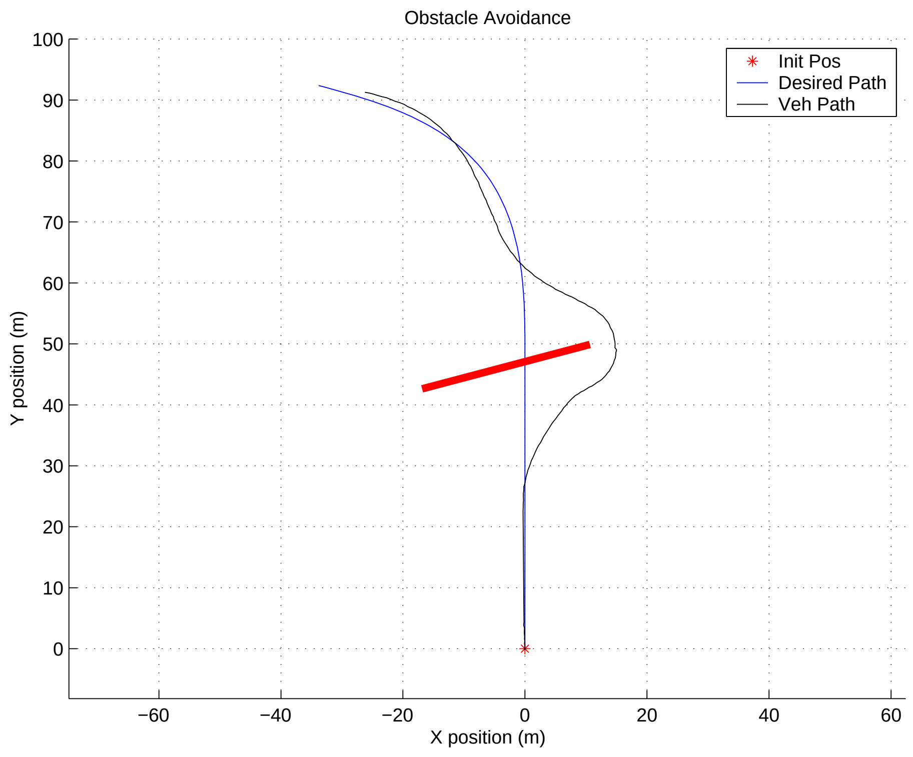


Figure 5.22: Obstacle Avoidance: A 'wall-like structure'

the ute clearly showed the system performing as specified, in terms of path generation, tracking and obstacle avoidance.

Chapter 6

Conclusions

As originally stated, the aim of this thesis was - 'given the computing, sensing and actuation capabilities available on the HSV project's test vehicle to develop, implement and test a system for an autonomous urban vehicle that facilitates the two key competencies of any ALV':

- Path Following, and
- Path Planning

To successfully achieve these goals, the thesis would, by definition, consist of two key components: 1) the necessary developmental/theoretical work (covered in Chapters 2, 3 and 4), where the techniques for CCPP, obstacle avoidance and path following were researched, developed and implemented through off-line simulations, and 2) the implementation of this theory together with interfacing to the obstacle detection module developed by Lee (2001) on the ute, so as to produce the complete system (presented in Chapter 5).

The research component of this thesis has necessitated the investigation of a number of important aspects of car-like robot systems. In Chapter 2, the motion (nonholonomic) constraints of such a vehicle were introduced, and techniques (notably CCPP) for generating trajectories that are physically trackable by car-like vehicles were derived in a manner which was applicable to this system's particular requirements. Chapter 3 looked at the approaches available for path replanning (as a consequence of obstacle avoidance). Several graph searching algorithms were implemented and their performance compared. These were combined with the obstacle representation from the obstacle detection module to produce a local path replanning technique. However, computational time constraints meant that a simplified (but successful) method had to be implemented on the ute. Once reliable path (re)planning methods were available, car-like robot control was investigated. A steering only feedback/feedforward controller, using a PPF control law,was developed that demonstrated accurate path tracking in both simulated and field tests.

Upon reflection, though, probably the key aspect of this thesis was the successful implementation of the theoretical developments into the path planning and tracking system described in Chapter 5. A number of recent undergraduate thesis within the HSV project have suffered from the lack of tangible outcomes. This has meant that old ground has often been retraced, and progress between consecutive years slowed. However, in 2001, significant progress has been made. The system currently implemented on the ute meets all of the specifications outlined in the thesis objectives, and the software is in a clear, well documented form (see attached CD), ready for continued further development in the coming years.

As is always the case with the 'bottomless pit' that is research and development work, particularly in a project as broad as this one, it was not possible to investigate all project avenues. So, throughout the course of the thesis, suggestions for future work were made. These are reiterated here along with some previously unmentioned potential research areas:

- The system in its current form consists of two MATLAB software modules accessing Hyperkernel's shared memory. These modules, along with the system interface, could be rewritten in C/C++ to provide a proper real-time system realisation.
- The behaviour of the steering controller (currently a simple PID controller) appears very non-linear. This fact should be investigated and a controller, possibly a model-based one, developed to provide smoother steering control.
- A robust, multisensor navigation loop is also required for providing high accuracy and high frequency vehicle pose estimates.
- The CCPP techniques presented in this thesis could also be researched further, particularly their role in real-time obstacle avoidance systems.

- The functionality of the path entry interface could be expanded to ensure that it is used in future (for example, to allow the user to enter waypoints by moving the vehicle around).
- Throttle, brake and transmission control remain largely undeveloped within the HSV project. Given the progress made this year, it is probably time for investigation of these aspects of vehicle control.
- The obstacle avoidance capabilities of the system could be expanded to handle more 'obstacle dense' environments.
- Finally, the dynamics of the vehicle could be characterised further, so that the continuous curvature paths generated can be matched to the motion characterstics of the ute.

In summary, then, it would seem that this thesis has helped smooth the path for future developments with respect to the HSV project.

Bibliography

ACFR (2001), 'Autonomous Open-Cut Mine Haul Trucks', April, 2001. http://www.acfr.usyd.edu.au/projects/development/mining/haul-truck September 2001.

Ball, R. (1900), *A treatise on the theory of screws*, Cambridge University Press, Cambridge.

Bedford, A. and Fowler, W. (1996), *Dynamics: Engineering Mechanics*, SI edn, Addison-Wesley.

Borenstein, J., Everett, H. and Feng, L. (1996), *'Where am I': Sensors and Methods for Mobile Robot Positioning*, University of Michigan.

Boyce, N. and Coghlan, A. (2000), 'The End of the Beginning', *New Scientist* **167**(2245), 4–5.

Cowlishaw, M. (1997), Dynamic modelling of a high speed vehicle for location estimation, Undergraduate thesis, University of Sydney.

Davis, T. (1999), 'Total Least-Squares Spiral Curve Fitting', *Journal of Surveying Engineering* **125**, 159–176.

De Luca, A., Orioli, G. and Samson, C. (1998), Feedback control of a nonholonomic car-like robot, *in* J. Laumond, ed., 'Robot motion planning and control, Lecture Notes in Control and Information Science', Vol. 229, Springer, pp. 171–253.

Di Stefano, J., Stubberud, A. and Williams, I. (1990), *Theory and Problems of Feedback and Control Systems*, second edn, McGraw-Hill.

Dijkstra, E. (1959), 'A Note on Two Problems in Connexion with Graphs', *Numerische Mathematik* **1**, 269–271.

Dubins, L. (1957), 'On curves of minimal length with a constraint on average curvature and prescribed initial and terminal position and tangents', *Amer. J. Mathematics* **79**, 497–516.

Egerstedt, M., Hu, X. and Stotsky, A. (1998), Control of a car-like robot using a dynamic model, *in* 'IEEE International Conference on Robotics and Automation', Vol. 4, pp. 3273–3278.

Foster, I. (1995), '3.9 Case Study: Shortest-Path Algorithms', Designing and Building Parallel Programs, http://swt.cs.tu-berlin.de/pa/dbpp/text/node35.html, July, 2001.

Fraichard, T. and Ahuactzin, J. (2001), Smooth Path Planning for Cars, *in* 'IEEE Int. Conf. on Robotics and Automation'.

Fraichard, T., Scheuer, A. and Desvigne, R. (1999), From Reeds and Shepp's to Continuous-Curvature Paths, *in* 'Int. Conf. on Advanced Robotics', pp. 585–590.

Graf, B., Wandosell, J. and Schaeffer, C. (2001), Flexible Path Planning for Nonholonomic Mobile Robots, *in* '2001 Fourth European Workshop on Advanced Mobile Robots (Eurobot'01)', pp. 199–205.

Gu, W. (1999), 'The Clothoid', Course Notes: Differential Geometry of Curves and Surfaces, Harvey Mudd College, January, 1999 http://www.math.hmc.edu/faculty/gu/ August, 2001.

Guivant, J. and Nebot, E. (2001), 'Optimization of the Simultaneous Localization and map Building Algorithm for Real Time Implementation', *IEEE Transactions on Robotics and Automation* **17**(3), 242–257.

Guivant, J., Nebot, E. and Baiker, S. (2000), 'Autonomous navigation and map building using laser range sensors in outdoor applications', *Journal of Robotic Systems* **17**(10), 565–583.

Guldner, J., Sienel, W., Tan, H., Ackermann, J., Patwardhan, S. and Bunte, T. (1999), 'Robust Automatic Steering Control for Look-down Reference Systems with Front and Rear Sensors', *IEEE Transactions on Control Systems Technology* **7**(1), 2–11.

Hart, P., Nilsson, N. and Raphael, B. (1968), 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths', *Robotica* **SSC-4, No. 2**, 100–107.

Hemami, A., Mehrabi, M. and Cheng, R. (1994), 'Optimal kinematic path tracking control of mobile robots with front steering', *Robotica* **12**, 563–568.

Kanayama, Y. and Hartman, B. (1989), Smooth local planning for autonomous vehicles, *in* 'IEEE Int. Conf. Robotics and Automation', Vol. 3, pp. 1265–1270.

Kanayama, Y. and Miyake, N. (1985), Trajectory generation for mobile robots, *in* 'Int. Symp. on Robotics Research', pp. 16–23.

Kelly, A. (1994*a*), A feedforward control approach to the local navigation problem for autonomous vehicles, Technical report, Carnegie Mellon University, The Robotics Institute.

Kelly, A. (1994*b*), A partial analysis of the high speed autonomous navgation problem, Technical report, Carnegie Mellon University, The Robotics Institute.

Kelly, A. (1995), An Intelligent, Predictive Control Approach to the High-Speed Cross-Country Autonomous Navigation Problem, PhD thesis, Carnegie Mellon University.

Konishi, Y. and Takahashi, H. (n.d.), Autonomous loading of rocks by use of intelligent loaders with a vision system: A concept of autonomous loading and path generation, Technical report, Tohoku University, Dept. of Geoscience and Technology.

Kreyszig, E. (1999), *Advanced Engineering Mathematics*, eighth edn, John Wiley and Sons.

Lai, J. (2000), Reacitve navigation, Undergraduate thesis, University of Sydney.

Latombe, J. (1991), *Robot Motion Planning*, Kluwer Academic Publishers.

Laumond, J. (1986), Feasible trajectories for mobile robots with kinematic and environmental constraints, *in* 'Int. Conf. on Intelligent Autonomous Systems', pp. 346–354.

Laumond, J., Sekhavat, S. and Lamiraux, F. (1998), Guidelines in nonholonomic motion planning for mobile robots, *in* J. Laumond, ed., 'Robot motion planning and control, Lecture Notes in Control and Information Science', Vol. 229, Springer, pp. 1–53.

Lee, K. (2001), Reactive navigation for an autonomous outdoor vehicle, Undergraduate thesis, University of Sydney.

Lim, E. (1998), Mechanical Hardware Interface and Path Planning for The High Speed Vehicle, Undergraduate thesis, University of Sydney.

Lin, J. (1996), Path Following For Wheeled Mobile Robots, Undergraduate thesis, University of Sydney.

Matheson, D. (2000), Low/High Level Control, Undergraduate thesis, University of Sydney.

Mazer, E., Ahuactzin, J. and Bessiere, P. (1998), 'The Ariadne's Clew Algorithm', *Journal of Artificial Intelligence Research* **9**, 295–316.

Mok, L. (1998), Low Level Control, Undergraduate thesis, University of Sydney.

Mutambara, A. (1999), *Design and analysis of control systems*, second edn, CRC Press.

Nagy, B. and Kelly, A. (2001), Trajectory Generation for Car-Like Robots Using Cubic Curvature Polynomials, *in* 'Field and Service Robots 2001'.

Nebot, E. (2001), Computers in Real Time Control and Instrumentation (MECH4730), University of Sydney.

Nelson, W. (1989), Continuous curvature paths for autonmous vehicles, *in* 'IEEE Int. Conf. on Robotics and Automation', Vol. 3, pp. 1260–1264.

Öztan, O., Baykal, O., Müftüoglu, O. and Sahin, M. (1995), 'Intersection of Spiral Curve with Circle', *Journal of Surveying Engineering* **121**(1), 3–7.

Pomerleau, D. (1995), RALPH: Rapidly Adapting Lateral Position Handler, *in* 'Intelligent Vehicles', pp. 54–59.

Rankin, A. (n.d.), 'Vehicle Control', Unmanned Vehicles, http://www.me.ufl.edu/ webber/web1/pages/research_areas/vehcile_control.htm, September, 2001.

Rankin, A. and Crane III, C. (1996), A multi-purpose off-line planner based on an A* search algorithm, *in* 'ASME Design Engineering Technical Conferences and Computers in Engineering Conference'.

Reeds, J. and Shepp, L. (1990), 'Optimal paths for a car that goes both forwards and backwards', *Pacific Journal of Mathematics* **145**, 367–393.

Robinson, A. (2001), 'Smart ICT Manufacturing for a Smarter Australia', *AGSEI Technology Business Review* pp. 7–11.

Scheuer, A. and Fraichard, T. (1996*a*), Global continuous curvature path planners for car-like robots, Technical report, Inst. Nat. de Recherche en Informatique et en Automatique (INRIA).

Scheuer, A. and Fraichard, T. (1996*b*), Planning Continuous-Curvature Paths for Car-Like Robots, *in* 'IEEE/RSJ Int. Conf. on Intelligent Robots and Systems', Vol. 3, pp. 1304–1311.

Scheuer, A. and Fraichard, T. (1997), Collision-Free and Continuous-Curvature Path Planning for Car-Like Robots, *in* 'IEEE Int. Conf. on Robotics and Automation'.

Spanier, J. and Oldham, K. (1987), *An Atlas of Functions*, Hemisphere Publishing Corporation.

Sukkarieh, S., Nebot, E. and Durrant Whyte, H. (1999), 'A high integrity IMU GPS navigation loop for autonomous land vehicle applications', *IEEE Transactions on Robotics and Automation* **15**(3), 572–578.

Takahashi, A., Hongo, T. and Ninomiya, Y. (1989), Local path planning and control for AGV in positioning, *in* 'IEEE Int. Conf. Intelligent Robots and Systems', pp. 392–395.

Wit, J. (2001), Vector Pursuit Path Tracking for Autonomous Ground Vehicles, PhD thesis, University of Florida.