DOCTOR OF PHILOSOPHY

Investigation of Machine Vision and Path Planning Methods for use in an Autonomous Unmanned Air Vehicle

Williams, Matthew

Award date:
2000

Awarding institution:
Bangor University

Investigation of Machine Vision and Path Planning Methods for use in an Autonomous Unmanned Air Vehicle

Matthew Williams

Thesis Submitted in Candidature for the Degree of
Doctor of Philosophy

December 2000

School of Informatics
University of Wales, Bangor
United Kingdom

Summary

The thesis investigates obstacle detection and path planning methods appropriate for use in an unmanned helicopter, which are essential to establishing autonomous control of the vehicle.

The research described here is complementary to a feasibility study by EA Technology Ltd to consider the use of robotic vehicles in the inspection of overhead electricity power lines. The context of this project is the proposal that the inspection be performed by a small, remotely-piloted helicopter fitted with a video camera, returning video imagery to the ground-based observer.

A major obstacle to this concept is that, under current flight regulations, the helicopter would not be allowed to fly outside the visual range of the ground station. This work investigates whether autonomous obstacle detection based on machine vision coupled with 3D path planning has the potential to remove this constraint.

The thesis first discusses the requirements and current methods of overhead line inspection. A review of relevant machine vision and path planning methods is given and a method for placing these into a hierarchical control system architecture is described. A method for fast 3D path planning, based on the distance transform, is introduced and experimental results are presented to show its effectiveness. Implementation of the control system, integrating the vision system and path planner, is done by means of a scale model laboratory test rig. The construction of this test rig is described and results are presented to show obstacle avoidance in action. The thesis ends with an assessment of how far the research has advanced the prospect of autonomous aerial power line inspection.

Contents

List of Figures

Acknowledgements

I wish thank Dr Dewi Jones for his invaluable advice, guidance and motivation throughout the course of this project.

I must also thank EA Technology Ltd, especially Mr Graham Earp, for sponsoring this project and for all the support given.

This PhD studentship was supported by the EPSRC.

Thanks to everyone at the School of Informatics who have made the years fly by!

Finally, thanks to my family for being there, and everyone who helped me get this far.

Document Information

This thesis was prepared using XEmacs, with Corel Draw, PovRay, and XFig providing the pictures. The thesis was compiled using LaTeX 2_ε.

Chapter 1

Introduction

The world in which we now live is totally dependent on electricity, and in recent years we have come to expect a very high level of service from the electricity distributors. Gone are the days in which power cuts, or outages, were considered as a normal part of life. In this electronic age we demand reliable power supplies to power our factories, farms and homes.

The demand for reliable power supplies has pushed the distribution companies to place more emphasis on preventive maintenance. In towns and cities the delivery of power is dominated by underground lines. However, in the majority of rural areas power is supplied via lines suspended from poles. These overhead lines and poles are exposed to the elements and over time faults can develop.

Underground lines are obviously difficult to inspect visually, but if a fault is found it is relatively straight forward to fix. Usually the people who do the work live locally, and the position of the lines are normally accessible. Over-head lines are easier to inspect, but to find a fault may mean that many miles of lines would need to be checked. Also once a fault is found it may be very difficult to get equipment and people to the area.

The types of power line that interest us are the 11KV and 33KV wooden pole mounted power-lines. These are most common in rural areas of the country distributing power to villages and isolated farms.

The power-lines need to be inspected by law. But also it makes sense for the power distribution companies to ensure that their lines are as reliable as possible.

1.1 Inspection Processes

The current methods of inspection can be split into two types:

- Ground Based Inspection
- Air Based Inspection

Ground based inspection means that an inspector walks from pole to pole inspecting the lines, poles and pole-top equipment. For a closer check the inspector may climb the pole to get a better view. However, for safety the power needs to be switched off during this type of inspection so it is only used once a fault has been located. This method is a slow and dirty job especially during the winter months. However, it is reliable and the inspection teams do find faults which are repaired before they become a problem. In the past few years the use of quad bikes to speed up the process has been tried, but even with these the process is slow.

Air based inspection relies predominantly on helicopters and human inspectors. As one might imagine this method is not cheap. Either the distribution companies, individually or in a consortium, run their own helicopter, or the helicopter plus pilot is chartered. A high level of competence and training is required by a pilot to fly these inspection missions as they are not without their danger. The inspection is performed by eye, with the inspector sometimes using binoculars for a closer look. The inspection process is fast, but can never be as reliable as the ground based method.

So how can the inspection process be improved so that the advantage of the speed of the helicopter and the reliability of the ground based inspection can be combined?

1.2 The RIPL Concept

The Robot Inspection of Power Lines (RIPL) [1] principle was developed jointly by EA Technology and the Control and Instrumentation group at the University of Wales, Bangor. The initial project aimed to investigate the use of an air vehicle as a platform for an automatic inspection system. A feasibility study [2] and a bridging study [3] both came to the same decision that an autonomous helicopter with a vision based inspection system mounted on a stabilised platform would be the best solution to the problem.

The SPRITE [4] airframe was chosen as the vehicle to model the system on. Although the vehicle's development has ceased due to the loss of a military contact it was decided that it was the most suitable concept on which to base this research programme. The Sprite in action can be seen in figure 1.1.

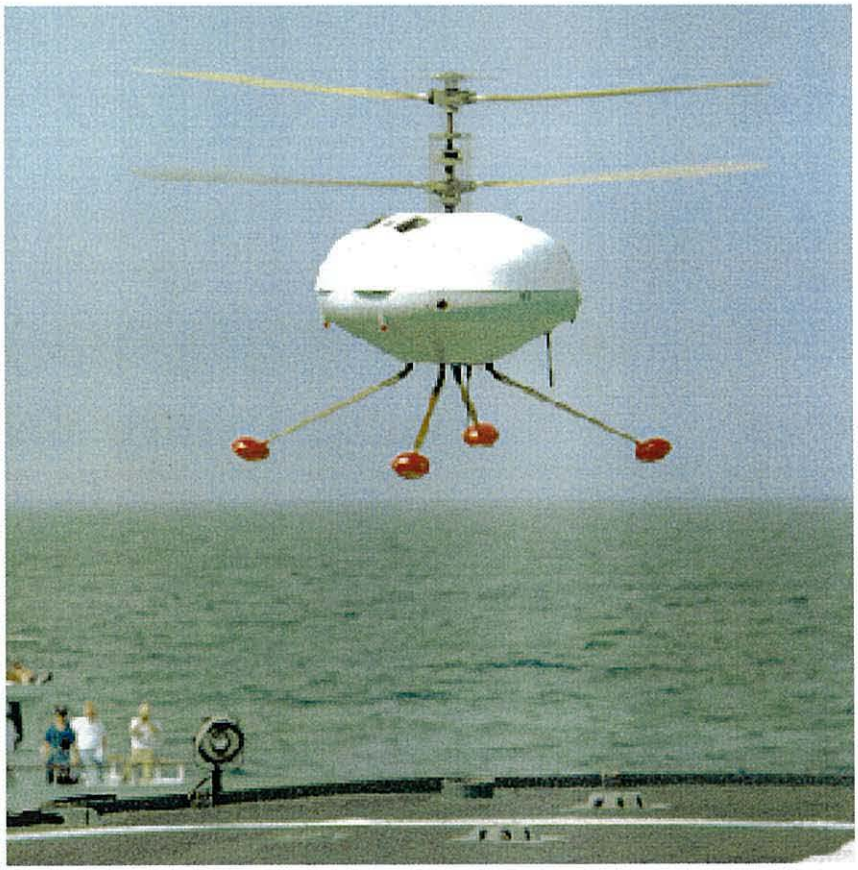


Figure 1.1: A flying Sprite

In figure 1.1 the unusual design of the Sprite vehicle can be seen. It has two contra-rotating rotor blades which removes the need for a stabilising tail rotor. This means that the helicopter has no concept of "front" or "back", so is equally happy flying in any direction. This means that it does not need to rotate in azimuth to travel in another direction, so a stabilised camera platform does not have to compensate.

The RIPL concept is based on a helicopter able to fly along a stretch of power lines, sending back high quality video images of power lines, poles and pole-top equipment so that either a human, or eventually a computer based inspector can check for faults. To make the inspection process cost effective about 30 miles of line would need to be inspected during each mission.

So the problem is easily solved! A remote control helicopter with a video camera to send back both information to the pilot and the inspector. However, life can never be that simple. What happens if the radio link between the pilot and the helicopter is lost? Most likely the helicopter would crash. With this in mind the reader should be glad that it is against the law to fly a remote control vehicle out of the visual range of the pilot. The air is policed by the Civil Aviation Authority (CAA), it is their role to maintain standards and improve safety.

The answer is to put the pilot back in the helicopter. But not a human pilot this time, a computer based pilot. The idea of autonomously controlling a vehicle is not new. Several autonomous aircraft exist, some have flown across the Atlantic but no autonomous vehicle has been approved by the CAA for civilian use.

The aim of the work presented here is to try to produce a system that could be used within the Sprite helicopter and would also satisfy the CAA as to its suitability for solo flight.

The RIPL concept encompasses many different engineering and scientific disciplines, other projects have been running in parallel with the work reported here. The interested reader can find more details about these from the following [1, 5, 2, 3, 6].

1.3 Obstacle detection, location and avoidance

So how can we put the *Pilot* back in the helicopter? There are two main issues here. Firstly the "pilot" needs to be able to "see" any obstacles that may pose a threat to the vehicle. Secondly, once an obstacle has been detected it must be avoided.

However, a controlling framework is required to incorporate all the different technologies that are needed for autonomous flight.

1.3.1 Controller Development

Before we put the "pilot" into the aircraft, what can be done to simplify the process? A controller hierarchy [7] has been investigated. This adds the concept of a *Planner* function to develop mission goals such as deciding which poles need inspecting, and a *Navigator* function to do off line planning using information from the *Planner* function and data gathered from previous inspection runs. Chapter 3 examines the controller.

The controller is used to gather all the individual elements that would be required to enable an autonomous aircraft to fly. For example the *Planner*, *Navigator* and *Pilot* functions all need access to a map. It makes sense to only have one map, which is used at different resolutions dependent on the controller level. This is just one advantage of a controller system.

The majority of this thesis is based in the *Pilot* level of the controller. This would be the part of the controller that would "fly" the aircraft based on the *Planner's* mission plan and the *Navigator's* flight plan. The *Pilot* level is fundamentally two separate functions, a machine vision system and a path planner. The next sections introduce these concepts.

1.3.2 Obstacle detection and location

In order to avoid something its position must be known. There are several methods for measuring position of objects, including:

- Touch
- SONAR
- RADAR
- LIDAR
- Machine Vision

Early robots used tactile sensors to find obstacles, in a similar way that a cat might. Obviously, tactile sensors are not an option on a helicopter!

The next three items in the list above all work in a similar manner. They are active sensors that send out a pulse of energy and, in their most simple form, measure the time it takes for reflections to return. In this way objects can be detected and located within an environment.

SONAR uses sound to locate obstacles, just like bats. However, it is mostly used in under water robots as sound travels further in that medium.

RADAR and LIDAR equipment is expensive and bulky. In fact, very few light aircraft are fitted with RADAR. However, much more than just location can be measured using it, and it is very accurate.

The last item, vision, is what all human pilots use. Machine vision means using cameras and computers to make sense of an environment. Much research has been done in trying develop machine vision techniques that enable a robot to be able to see. However, the day when a robot can make sense of its environment as well as we can is still some way off.

The research presented here aims to show that machine vision can give as much detail as other types of sensors. Machine vision systems are generally smaller and cheaper than RADAR systems which is a requirement for commercial operations.

The task of detecting obstacles has been split into the detection of high risk fast moving obstacles such as low flying jets, and the detection of lower risk slow moving obstacles such as hot air balloons. This distinction has been made as the manner in which the two types are dealt with is different. Slow moving obstacles can be manoeuvred around, fast moving ones need to be avoided.

Fast moving obstacles present an immediate threat to the helicopter. They must be detected quickly and as much information as possible about their motion should be measured. The exact size or shape is not important, the speed and direction are. Methods such as looming and time-to-contact have been investigated. These together with a reflex motion planner have been proposed as a possible solution for avoiding fast moving obstacles.

The slow moving obstacles that could be encountered by the helicopter during the flight do not pose an immediate threat, but need detecting all the same. The standard machine vision detection method presented here uses optical flow to get a measure of location.

Optical flow is a measure of motion between a series of images captured by a video camera. It is not a simple subtraction to see what has changed, but a measure of image plane motion. Motion in the image plane is either due to the motion of the camera or to the motion of obstacle, or a combination of them. The obstacles that have been classified as slow moving, can be regarded as stationary for the purpose of detection. Using this assumption, any motion within the image frame is due to the helicopter moving. The velocity of the helicopter can be measured in several ways including inertial navigation systems or even by using the Global Positioning System (GPS). Knowing the amount of helicopter motion during the series of images used to calculate the optical flow, and the camera parameters, it is possible to estimate the distance to objects and thus calculate their position in space.

1.3.3 Avoiding Obstacles

Once an obstacle has been detected then avoiding action may be needed. However, if at all possible the inspection process should continue. The aim of the inspection process is to examine a stretch of line, if some part is missed then it will have to be re-inspected which adds to the cost.

Slow moving obstacles can be avoided by path planning to allow the inspection process to continue.

When a fast moving obstacle is detected, the inspection process is of no consequence. The primary objective is to get out of danger. There may not be time even to plan a path. In this case a prepared escape path is activated. These escape paths would be continuously calculated during the flight ready to be put into action at any time. The advantage of using these prepared escape paths is that they will not put the vehicle into any further danger as they are based on all the known data about the environment.

Path planning systems have been developed for robots for many years, and there are many types. In chapter 2 a summary of the different methods is presented. The method chosen to implement the path planner in this project is based on the Distance Transform [8, 9, 10, 11]. This can be thought of as a combination of cell decomposition methods which are efficient for high dimensional planning, and potential field methods which are rapid.

The use of spatial decomposition has been incorporated into the scheme as it can reduce the computer memory requirements for an algorithm and more importantly, according to the Quadtree/Octree Complexity theory [12], improve the efficiency of the algorithm.

Both the standard avoidance method and the reflex method use a rapid three dimensional path planner using the distance transform and octree spatial decomposition [13, 14, 15].

1.4 Overview of the thesis

In chapter 2 a more detailed background of the RIPL concept is given together with a literature review of suitable technologies. A description of the different types of helicopter based guidance systems is given together with a description of a controller hierarchy that breaks the tasks of navigation and obstacle avoidance into distinct layers. The different types of path planning methods are presented. Obstacle detection systems are presented and the decision to use a machine vision system is justified.

Chapter 3 presents a controller hierarchy. The division of responsibilities into the three levels, the *Planner*, the *Navigator* and the *pilot* is shown. The Preferred Flight Space (PFS) is introduced to provide a method for guaranteeing the safety of any planned path.

In chapter 4 details about the obstacle detection and location systems are given. Obstacles are divided into two main types: Stationary or slow moving obstacles and fast moving obstacles. Methods to detect and locate both types are given and discussed. The use of the optical flow field for the location of objects is examined, several methods are considered and the one chosen is justified.

The Distance Transform Planner is described in detail in chapter 5. This is a new rapid path planner for three dimensional environments. The use of spatial decomposition to reduce the processing time and memory requirement is shown. This new method has been presented by Williams et al. in [12], Williams and Jones in [14] and Williams et al. in [15].

Chapter 6 presents the results of the various experiments performed during the development of the path planner. This chapter shows the development of the Distance Transform algorithm from a feasibility study based on a MatLab algorithm to a rapid 3D path planner.

The implementation of the Controller hierarchy is presented in chapter 7. This chapter concentrates on the design and construction of a new test rig that was used to investigate the combination of the distance transform path planner and

of an optical flow method for obstacle detection.

Chapter 8 discusses the results from experiments performed using the test rig. This chapter gives details about the configuration of the optical flow software using a "ground-truth" experiment. Results from several runs of the test rig are presented showing obstacle avoidance in action.

The final chapter discusses the work presented in this thesis. A judgement is made about the suitability of the system to answer the complex problems posed in trying to develop a safe control system for an autonomous system. Conclusions of the project in general are given together with recommendations for further work.

1.5 Contributions of this research work

The research contributions that are presented within this thesis are:

- The use of a controller architecture to combine all the different technologies needed for autonomous flight and the division of responsibilities between a *Planner*, *Navigator* and *Pilot* is introduced. The controller's benefit is that it can be tested and analysed by any certificating authority.
- The Preferred Flight Strip (PFS) and the Modified Preferred Flight Strip (MPFS) are introduced in order to improve safety and to prepare the flight path for the *Pilot* reducing the complexity of this "real-time" function.
- The development of a rapid three dimensional path planner using the distance transform and spatial decomposition has been the main contribution of this research work.
- A working three dimensional planner which has been incorporated with machine vision in a laboratory test rig.

Chapter 2

Background and Literature Review

The purpose of this chapter is twofold; first it discusses the background of the project and second it reports on the initial literature research that was performed. The aims of this project have been highlighted in the previous chapter. In this chapter a more detailed description of the power line inspection problem is given, together with details of related methods reported in the literature that will promote the aims of the project. The idea of using an autonomous air vehicle will be discussed and the reason for basing all subsequent work around the Sprite helicopter is explained.

The literature survey consisted of a search for information on what has been done before in the area of autonomous flight and the associated subject areas. The following sections give details about the varied mechanisms discovered in the literature with comments on their suitability, or otherwise, for the current project. The literature review is ordered in subject areas and each at the end of each section there is a discussion about the choices made for the project based on the information gained.

2.1 Background

This section describes the initial concept behind using an AUV to perform power line inspection. The reasons for performing the inspection are given, together with a description of current practice. The advantages and disadvantages are summarised, and a justification for using a UAV is presented.

2.1.1 Power Line inspection - Why?

The electricity companies need to reliably supply power to their customers. In urban areas electricity is distributed via underground cables. This method of distribution protects the cables from most forms of damage. However, to supply power to rural areas pole mounted cables are used. This method of distribution exposes the poles and cables to many forms of failure.

The electricity companies are bound by law to inspect their power distribution equipment at regular intervals. Failure to perform these checks can result in interruption of the supply but may have more serious consequences. In a recent case [16] Scottish Power was fined £100,000 after the death of a nine-year-old boy who was electrocuted. The boy was killed as he climbed a tree. The court heard that the power lines passed through the tree, but that it should have been trimmed by Scottish Power to at least three meters below the level of the lines. The electricity company admitted a charge under the Health and Safety Act. The court heard the trees had not been inspected for four years due to a fault in the company's computer system, although thousands of pounds had since been spent updating safety procedures.

This case highlights the dangers of not performing regular inspections. Less serious problems can also be prevented by performing regular inspection; these include:

- Cracks in insulators
- Signs of corrosion on cables

- Traces of arcing
- Worn insulators
- Structural damage to poles
- Broken or slack stay wires
- Leaking transformers
- Missing anti-climb guards and notices

Detecting these problems before they become too severe can reduce the likelihood of power cuts, or outages as they are known in the industry. To maintain a high standard of service power companies need to reduce the frequency of outages to a minimum. It is also beneficial in so much as the companies do not need to pay out compensation to their customers.

Inspection by whatever means is typically performed on a 4 year cycle. However, some lines that are considered "at risk" may be inspected more frequently.

2.1.2 Current Inspection Procedures

At present there are three main methods for power line inspection.

Pole climbing

The practice of pole climbing is now only used when looking for specific faults [2]. It has been discontinued for routine inspection because it is slow, and as the power is sometimes not removed from the lines, is a dangerous job for the inspection crews. This method does have the advantage of being thorough, but this comes at a cost.

Figure 2.1 shows a pole being repaired using a "cherry picker". This cannot be used in all cases as up to 50% of poles are situated at least 200m from the nearest road or track.

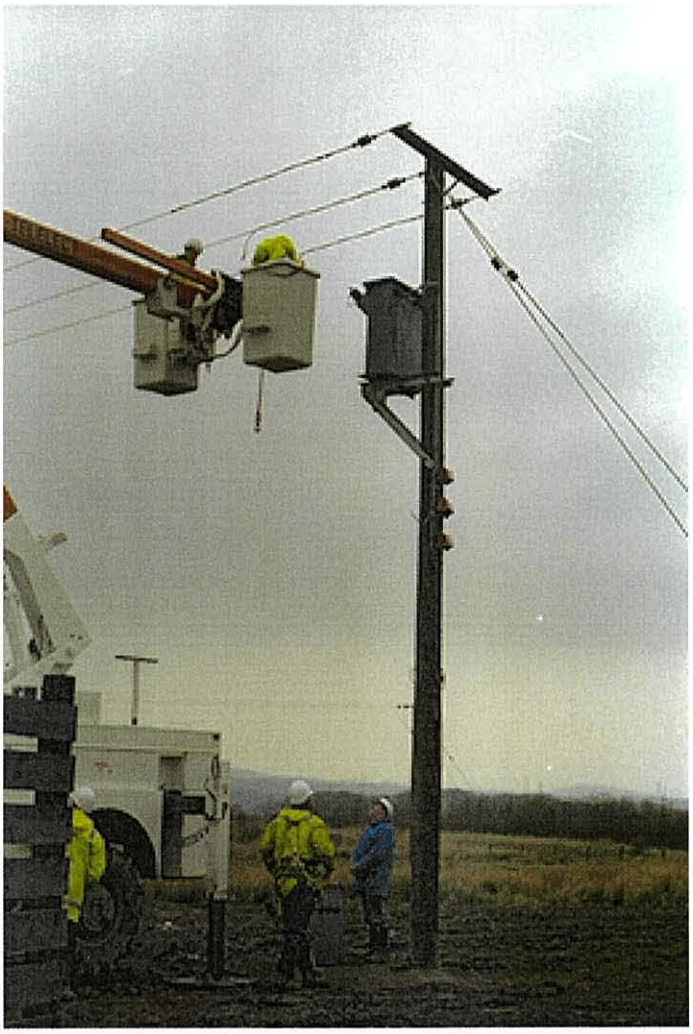


Figure 2.1: Line termination pole under repair

Foot Patrol

A team of two persons walk alongside the stretch of line noting any fault they may observe. The majority of the inspection process is performed at the poles. However, the majority of the time is spent walking between the poles. Since most of the poles are situated in rural areas the ground between the poles is generally rough with many obstacles such as walls, ditches and hedges impeding the lines persons. It is generally acknowledged that walking a line is very tedious, leading to lapses in concentration and occasionally a tendency to omit details [2]. Figures 2.2 and 2.3 shows a typical pole locations.

Figure 2.2: Power line situated in valley

Foot patrols can cover up to 150 poles in one day, but this could halve if the ground is bad or there are a number of spur lines requiring significant back tracking. In recent years quad-bikes have been used to inspect poles on open moorland which speeds up the operation.

Foot patrols give a good degree of accuracy, errors and omissions excepted. However, they can never provide the same amount of pole top detail as pole climbing. Also there is no record of the condition of the equipment except for the linesman's report and it is difficult to detect degradation from one inspection to another.

Manned Helicopter

The use of helicopters for inspection is not new. The national grid has been using this method for checking their transmission towers for a number of years. The electricity distribution companies have only started using helicopters to inspect their lines in recent years. This is because the cost benefit is not as great as that of the national grid as they have a much larger network to inspect.

Figure 2.3: Power line in upland area

Manned helicopter inspection is fast, and up to 250km of lines can be traversed in one day. However, they are far less accurate than foot patrols because it is difficult for the observer to know exactly which pole is in view and, with the unaided eye, only major faults can be detected although being able to view the poles from above does have advantages.

Another problem is that to obtain sufficient detail the helicopter needs to fly close to the poles and hence close to the ground. It is the pilot's duty to ensure that the helicopter does not pose a threat to those on the ground, and so he/she may need to veer away from the line to avoid a hazardous condition. This deviation from the line can result in a number of poles being missed which then have to be inspected by another method, which all adds to the cost. Figure 2.4 shows a number of poles in an area a pilot may not wish to enter due to the proximity of a road and trees.

Although there is no record of a power line inspection helicopter colliding with power lines there have been a number of cases where equally experienced pilots have done so [17, 18]. Flying so close to the lines and to the ground increases the

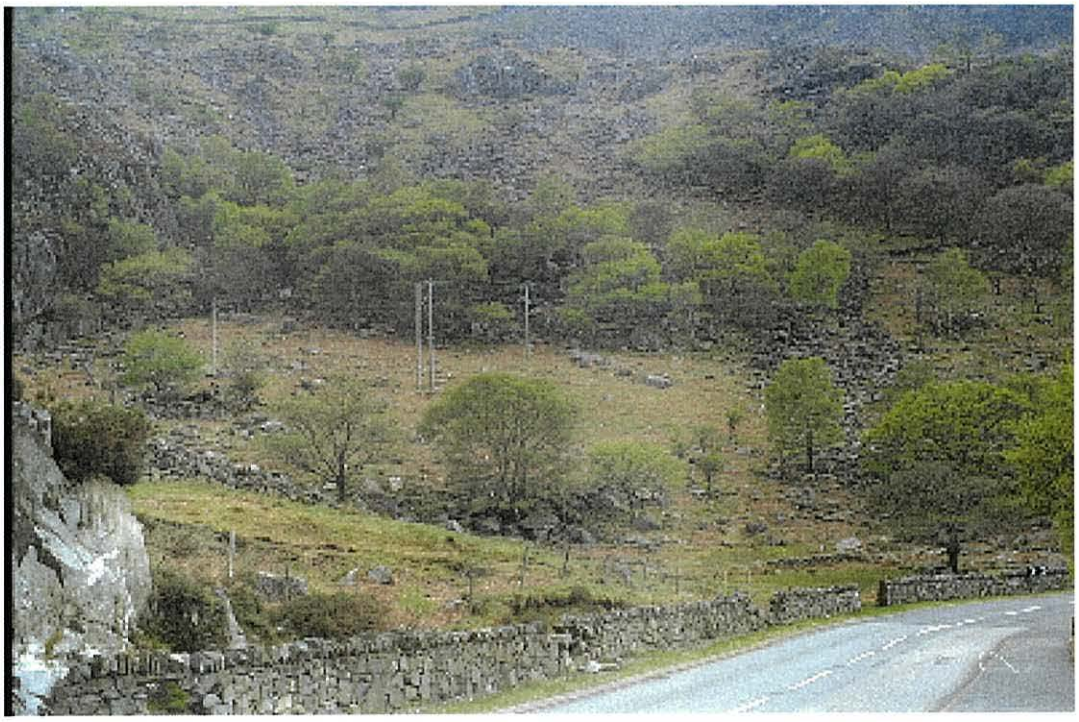


Figure 2.4: A difficult location for helicopter inspection due to the proximity of trees and a road

risk of both those in the helicopter and to those who may be on the ground.

To try and improve the inspection accuracy a stabilised mount with a video camera has been tested [19]. This allows the operator to fix onto a pole and zoom in to get a greater amount of detail.

In another project running at Bangor, an automatic pole tracker is being developed. This uses GPS information together with machine vision to lock a video camera onto a pole and allow an observer to zoom in and out to inspect it. The use of GPS will allow the operator to uniquely identify the pole removing the uncertainty that is currently the case. Another advantage of using a video camera is that the inspection results are recorded and can be inspected at a later date or during subsequent inspection. Figure 2.5 shows a picture of a pole top taken from a helicopter. This pole is in a dangerous condition as one of the conductors has come away from the insulator and is resting on the metal pole top.

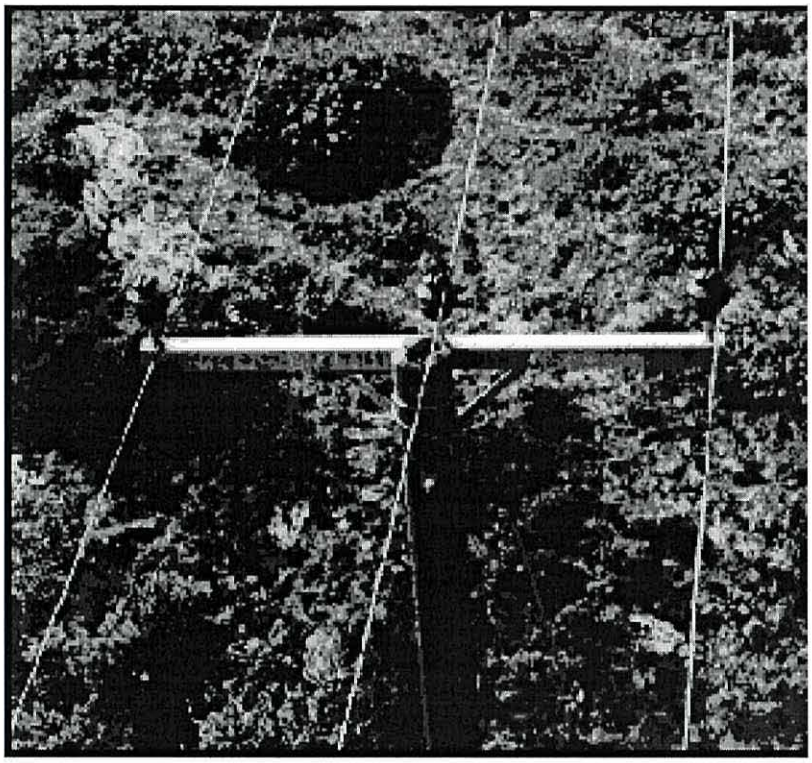


Figure 2.5: Faulty pole top

Summary

In general there is a trade off between speed of inspection and quality of inspection. Pole climbing is slow but thorough, helicopter inspections are fast but not accurate. The aim of inspection is to reduce the cost of maintenance and also the number of outages to the customer. The next section introduces a new method for power line inspection which will combine the accuracy of the pole climbing method with the speed of the helicopter inspection.

2.1.3 The RIPL Concept

The previous section has outlined the problems with the current methods for power line inspection. This section will introduce RIPL (Remote Inspection of Power Lines). A great deal of research has been performed by EA Technology as to which would be the best platform for remote power line inspection. A bridging study [2] suggested four different concepts:

- Semi-autonomous rotorcraft
- Semi-autonomous lighter-than-air vehicle
- Power line swinger
- Truck mounted long sight-line camera

Through the use of trade-off matrices to compare the advantages and disadvantages of the different platforms, the semi-autonomous rotorcraft was the concept that comes closes to matching the requirements.

System Description

The RIPL concept embraces several distinct engineering disciplines. In order to understand the complexity of the system a sub-system block diagram was constructed. Figure 2.6 on page 20 shows the complete system diagram.

The system is divided up into six separate sub-systems:

- Ground Station
- Ground Communications Link
- Vehicle
- Vehicle Navigation
- Line Inspection
- Safety

Each of these sub-systems is broken down further. The blocks that this thesis addresses are the Motion Planner and Flight Path Generator within the Vehicle sub-system, and the Obstacle Collision Detection within the Safety sub-system.

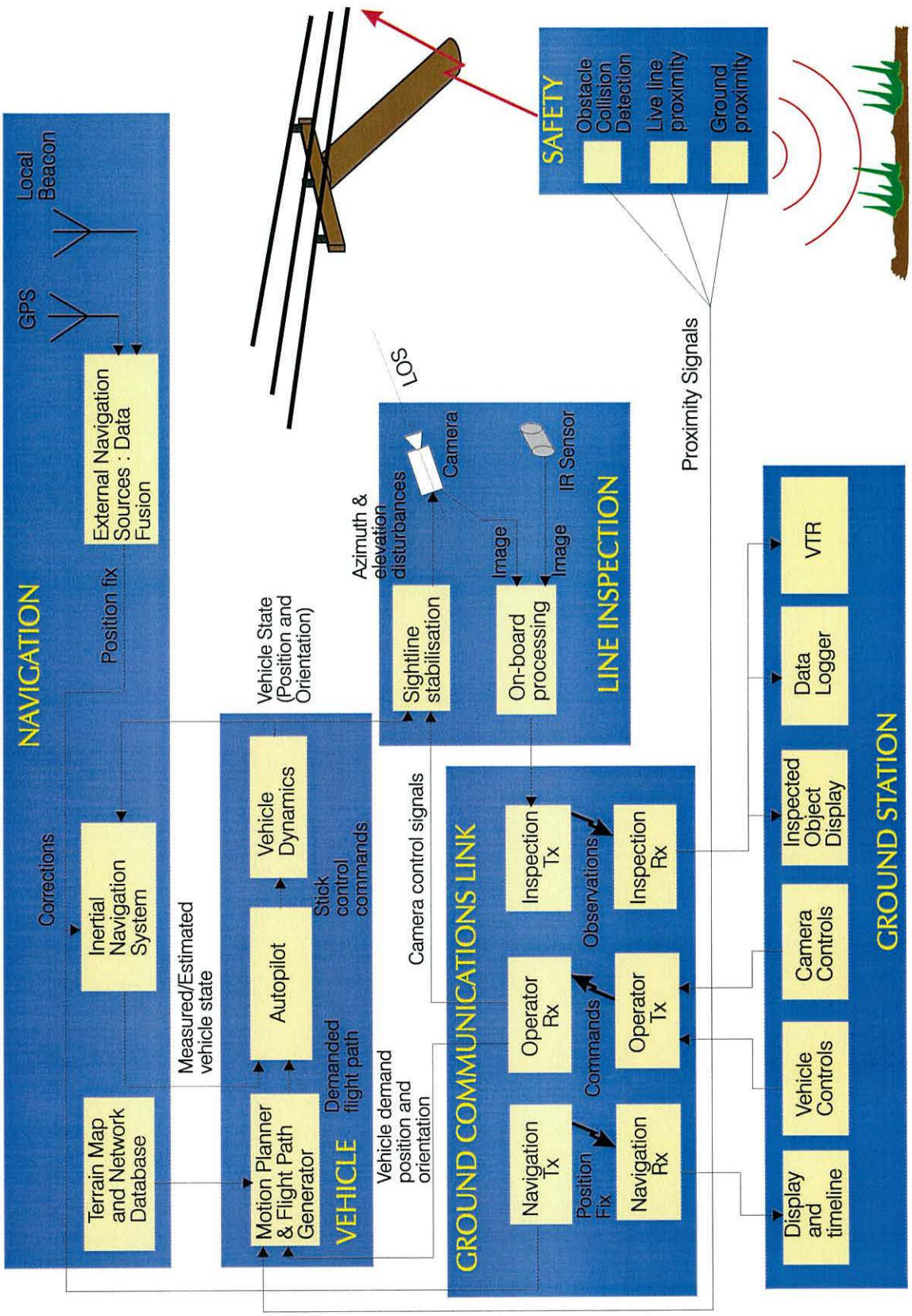


Figure 2.6: RIPL sub-system block diagram

The Sprite Helicopter

The concept based on a rotor craft was selected because it offers relatively high-speed inspection, good viewing angles and inspection detail which will improve the productivity and quality of line inspection.

There are a number of different rotor craft available ranging from small "hobbyist" vehicles up to military quality vehicles. A review of the then currently available airframes was conducted which resulted in the ML Aviation Sprite being selected [2].

Figure 2.7 shows the Sprite helicopter.

Figure 2.7: ML Aviation Sprite

The technical specification of Sprite is as follows:

- Airframe
  - Two two-blade coaxial counter-rotating rotors
  - Symmetrical planform body of glass fibre and carbon fibre designed to minimal radar, thermal, noise and optical signatures.
- Power plant
  - Two 4.5KW MLH 2/88 two-stroke flat-twin engines

  - Can fly on only one engine

- Launch and recovery
  - Vertical take-off and landing
- Guidance and control
  - Skyleader PCM radio command system
  - Automatic flight control system
  - Can be preprogrammed for autonomous flight or automatic recovery to base in the event of loss of control link
- Mission equipment
  - Two 500W alternators provide electrical power
  - Various sensor combinations including stabilised video or infra-red cameras, low light video, laser target designators, chemical sensors, electronic intelligence and ECM.

Dimensions

Rotor diameter (each)	1.60m
Body diameter (max)	0.65m
Height to top of rotor head	0.90m

Weights

Weight empty	28Kg
Max fuel	6Kg
Sensor pack (max)	6Kg
Max Take off and landing	40Kg

Performance

Max level speed (estimated)	130kmh^{-1}
Cruising speed	111kmh^{-1}
Climb rate at optimum speed of 83kmh^{-1}	366m per minute
Normal operating height range	250-500m
Operating ceiling	2440m
Maintainable height on one engine	2440m
Typical mission radius	32km
Endurance (mixed mission)	2 hours

The Sprite airframe and ground support element incorporates 70-80% of the technology required for RIPL. However, the three main problems with Sprite (as with any other systems) are:

- No certification for civilian work
- No autonomous capability
- Somewhat large and heavy to be exactly right for the RIPL concept

2.1.4 So what needs doing?

All civilian aircraft movements are subject to the regulations of the Civil Aviation Authority (CAA). The regulations cover aircraft from jumbo-jets down to hobbyists' radio-controlled models and therefore will include Sprite. Whether the RIPL concept is technically possible, or not, is of no consequence if the CAA is unwilling to certify its use.

The only UK legislation relating to unmanned air vehicles is the relevant parts of the UK Air Navigation Order (ANO) 1989 and the Rules of the Air, Regulations 1991. The overriding issue with both these is safety. Traditionally, the overall safety of an aircraft is achieved by the combination of the ANO and the Rules of the Air, the airworthiness of the aircraft, the licensing of the aircrew, air traffic control, operational control and maintenance and continued airworthiness control.

However, remote control hobbyist aircraft are exempt from the ANO, providing the aircraft is below 7kg in weight (excluding fuel) and that they are not used for commercial ventures. Sprite fails on both scores as its minimum weight without fuel or sensor payload is 28kg, and the intended use is commercial.

So Sprite will have to conform to the Air Navigation Order and satisfy the following sections:

Article 7 Certificate of Airworthiness: An aircraft shall not fly unless it holds a Certificate of Airworthiness.

Article 19(1) Composition of Crew of Aircraft: An aircraft shall not fly unless it carries a flight crew.

Article 20(1) Flight Crew Licences: Flight crew must hold appropriate licences.

Article 69(2) Rules of the Air: It is an offence to contravene of fail to comply with the Rules of the Air.

Articles 7 and 20(1) pose no difficulty in principle. Article 19(1) effectively prevents the flight of any unmanned vehicle which weighs more that 7kg. It would therefore be necessary to apply to the CAA for exemptions to conduct commercial operations. Article 69(2) is the main problem area as it requires every aircraft to obey the rules of the air.

Rules of the air

The Rules of the Air, Regulations 1991 outlines the best practice for flying which is backed up by law. The rules that are of particular interest for our application are:

Rule 5 Low Flying: Generally, an aircraft shall not fly over any congested area of a city, town or settlement essentially below 1500ft or below such height as would enable it to glide clear in the event of failure of a power unit etc whichever is the higher. An aircraft shall not fly closer than 500ft to any person, vessel, vehicle or structure.

Rule 17 Avoidance of collisions: "Rules of the Road" and "See and be Seen" principles for collision avoidance action.

The restriction of flying at an altitude of at least 1500ft over settlements is within the performance characteristics of the Sprite helicopter. However, the 500ft ($\sim$ 150m) proximity rule is restrictive. Current manned helicopters do have permission to fly within 30m of the electricity lines and the ground. The pilots employed on these missions have to be well trained and are usually ex-military.

In order to fly this close to the power lines the CAA would have to make an exception. This exemption could be justified from the point of view that it is the electricity company's property.

The "See and be Seen", or "sense and avoid" which is more commonly used for UAVs [20], requirement is the main driving force for the work presented in this thesis. The "be seen" part of the regulation is relatively easy to implement, by the use of lights and possibly a navigation transponder. However, the Sprite vehicle was intended for a military role and was deliberately designed *not* to be seen so modifications would be required to enhance its "visibility" both to the eye and to RADAR.

The main problem is the "Seeing" or "sensing" part of the regulations. The rules governing flying have been developed to cater for manned aircraft. At all times during a flight a responsible person is always in control of the aircraft, even when running on autopilot. For unmanned aircraft operated by remote control, it could be argued that the person on the ground is performing this function. However, the CAA regulations that apply to remote control aircraft state that the "pilot" should always be able to visually see the aircraft [21]. If the remote control aircraft carried a video camera that transmitted the view from the "cockpit" to the pilot, again it could be argued that the pilot is still in full control of the vehicle even when it is outside visual range. Unfortunately, the CAA would not allow this either, as they point out, correctly, that if the communications link failed then no one would be in control of the vehicle.

The only possible way to satisfy the CAA would be to put a "pilot" in the vehicle. In other words the vehicle would have to be autonomous. The autonomous

controller would need to perform all the crucial functions that a human pilot would. This thesis investigates the machine vision techniques and path planning methods that could be combined to produce the core functions of an autonomous controller.

The next sections presents a review of pertinent literature within the fields of machine vision and path planning.

2.2 Helicopter Guidance

Passive vision is used in military applications as it reduces the characteristic electromagnetic signature of a vehicle. This is important in military applications as the smaller the signature of a vehicle is, the harder it is to detect.

The main area of research is providing military helicopters, that use low flying techniques to avoid detection by the enemy, with intelligent guidance systems. The design of intelligent systems for helicopter guidance will require information about the objects in the vicinity of the flight path of the vehicle. The sensor system on the helicopter should be able to detect objects such as buildings, trees, poles and wires during flight. Sridhar and Chatterji [22] introduce a vision based obstacle detection system based on the optical flow/motion at different points in an image. The motion algorithms provide a sparse set of ranges to discrete features in an image sequence as a function of azimuth and elevation. They state that for obstacle avoidance guidance, and for display purposes, these discrete sets of ranges need to be grouped into sets that correspond to objects in the real world. They present tests on video footage from a helicopter moving slowly over a runway containing a number of objects. They state that the object segmentation algorithm together with a range estimation algorithm provides a basis for the object information required for helicopter guidance.

Cheng and Lam [23] describe an automatic guidance and control system for helicopter obstacle avoidance. Their paper describes the development of a guidance system that can reproduce low level nap-of-the-earth (NOE) flight. This flight mode is predominantly ground hugging and below tree tops, with mostly lateral

manoeuvres around obstacles. In this way the helicopter can use ground based obstacles such as ridges and banks of trees to provide cover from fire. Flying in the NOE mode places great demands on the pilot's flying skills, such a high degree of concentration is needed just to keep the vehicle safe. The system presented by Cheng and Lam is aimed at reducing the pilot's work load so that they can concentrate on deploying their weapons accurately rather than on flying the helicopter. They present simulation examples showing a helicopter altering course to avoid obstacles. They show that their method has the potential for producing trajectories that provide cover for the helicopter together with obstacle avoidance.

Sridhar and Phatak [24] also present a navigation system producing nap-of-the-earth flight, but unlike the previous paper their method is based solely on machine vision techniques. They split the guidance function into three levels - farfield, midfield and nearfield. The farfield function provides mission information incorporating global threat information and vehicle resources. The mission plan contains goals, and way points for directing the rotorcraft between goals. At this level, a coarse digital map of the terrain is used for planning. The midfield planning refines the path produced by the farfield level for a short duration ahead. The midfield planner used a more detailed map of the terrain. The flight path generated by the midfield planner is the path the rotorcraft would follow if all the a priori information is correct. The near field planner is a real-time function which uses the previously planned path and modifies it if and when unknown obstacles are detected. This method of off-line and on-line planning is similar to the method presented by Mystel [7] which will be discussed later. Their method for obstacle location is different to previous methods at it does not attempt to measure the rotorcraft's position using images. The rotorcraft's position is measured using an inertial navigation system. Kalman filters are used to recursively estimate object point coordinates. The filters perform two functions - Measurement update and time update. The first updates the filter when a new measurement is made, the second handles time between measurements. Three different filters are used based on different representations of the state vector, governed by the choice of coordinate systems. The first uses the earth coordinates of the object point as the state vector. The second uses the relative coordinates of the object point with respect to the rotorcraft sensor axis as the state vector. The final filter

assumes that image point motion can be represented by a polynomial model for each of the image coordinates. The results show that the first two Kalman filter implementations provide good estimates of object location, the third Kalman filter diverges because the actual image motion can be abrupt depending on the rotorcraft manoeuvre.

2.3 Controller Architecture Literature

The three papers on helicopter guidance above present a complete system description of a guidance system. The final paper by Sridhar and Phatak [24] describes the guidance function being broken down into three levels, the farfield, the midfield and the nearfield.

Mystel [7] introduces the principle of an Intelligent Mobile Autonomous System (IMAS). The controller is split both vertically and horizontally. The horizontal divisions give different levels in the controller, while the vertical divisions group associated functions at the different levels. A typical controller hierarchy is given in figure 2.8.

The different levels of the intelligent modules and their associated sensors and maps perform the following functions:

Planner The highest level of the intelligent module is the *Planner*. This level can be thought of as mission planning. This corresponds with Shridar and Phatak's farfield level. This level has the associated sensors, map and reporter. The sensors at this level could be as abstract as a database lookup to find which poles need inspecting. The map at this level could be a multidimensional Geographic Information System (GIS), which would provide much more than just terrain attitude. This GIS map would contain information about the approximate locations of distribution poles, together with information about likely static obstacles including transmission towers, communication structures, buildings. A GIS could also hold non-physical data such as areas where flying could be dangerous such as near airfields, or

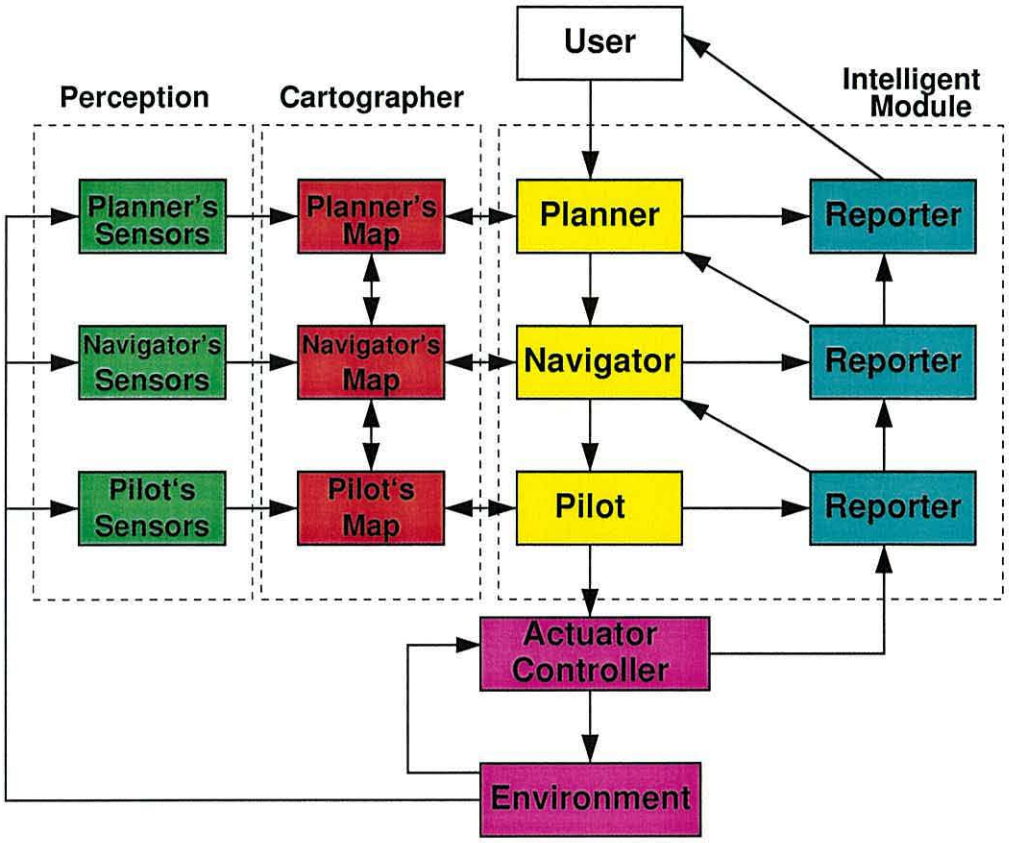


Figure 2.8: Controller Hierarchy

military test ranges, or even areas where manned helicopters have reported problems such as down-drafts. It could also hold information about the usage of the land. This could be important as manned helicopters always tend not to fly low over land that contains sensitive animals such as horses as they can be alarmed by the noise.

Navigator The second level of the intelligent module is the *Navigator*. This level is similar to the midfield function described previously. This level produces a flight path for the vehicle to follow, using the *Planner's* output as the basis for the path together with its own sensors and map. The path produced by this level would be the one the vehicle would follow in the absence of unknown obstacles.

Pilot This is the lowest level of the intelligent module. The *Pilot* performs path modification when an obstacle is detected. It uses details passed from the

Navigator and its own sensor suite and map to calculate these modifications.

Chapter 3 gives a detailed description of the implementation of this type of hierarchical controller for use in our particular application.

2.4 Path Planning Literature

Path planning is the process of generating a sequence of robot positions and orientations that moves a vehicle from a start location to a goal. It is different to trajectory planning as it does not take the dynamics of the vehicle into consideration, but only the kinematics. Motion planning is the term generally used to describe the combination of path and trajectory planning.

This section will outline the basics of path planning and will present some of the approaches that have been researched by others.

The task of finding a path from one position to another without causing damage to oneself or others is a process at which most animals, including humans, are exceptionally good. This ability has developed over countless years of evolution. The process uses information from a number of sensors including vision, tactile, aural and balance. The brain is naturally the key to the process analysing the inputs and using the knowledge base to produce stimuli to the limbs to generate motion. The current position of robot planning is nowhere near as advanced as even lowly organisms such as insects. These possess a complex planning capability which can plan paths between a nest and food sources and pass this information on to others.

There are a number of distinct methods for path planning, broadly classified as Roadmap, Cell Decomposition and Potential methods. Common to these methods is the idea of *configuration space*. The generation of configuration space reduces the problem of finding a path for a single point among a set of configuration space obstacles. Consider a rigid body robot which is free to translate but not rotate in a workspace consisting of two obstacles, as illustrated in Figure 2.9.

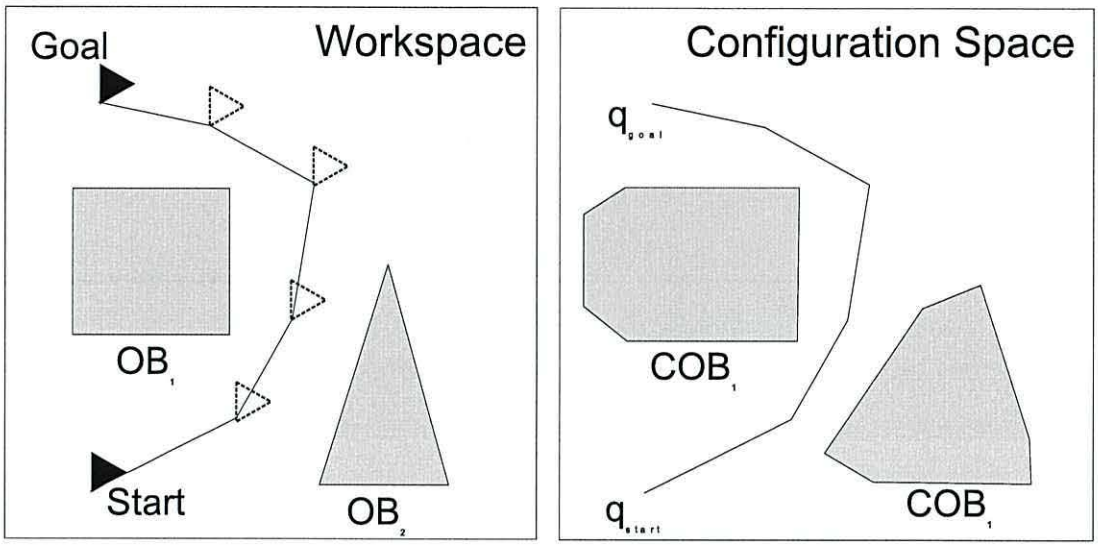


Figure 2.9: Configuration Space Example: The workspace diagram shows two obstacles and an example path for the triangular shaped robot. The configuration space diagram shows the path generated by reducing the robot to a point and *growing* the obstacles by an appropriate amount.

Consider the basic problem. Let the robot A (at a certain position and orientation) be described as a compact subset of $W = R^N$, N=2 or 3, and the obstacles $OB_1 \ldots OB_q$ be closed subsets of W. In addition, let F_A and F_W be Cartesian frames embedded in A and W, respectively, F_A is a moving frame, while F_W is a fixed one. By definition, since A is rigid, every point a of A has a fixed position with respect to F_A. But a's position in W depends on the position and orientation of F_A relative to F_W. Since the B_i's are both rigid and fixed in W, every point of B_i, for all $i \in [1, q]$, has a fixed position with respect to F_W.

A *configuration* of an arbitrary object is a specification of the position of every point in this object relative to a fixed reference frame. Therefore, a **configuration q** of A is a specification of the position T and orientation Θ of F_A with respect to F_W. The configuration space of A is the space C of all the configurations of A. The subset of W occupied by A at configuration $\mathbf{q}$ is denoted by $A(\mathbf{q})$. In the same fashion, the point a on A at configuration $\mathbf{q}$ is denoted by $a(\mathbf{q})$ in W.

One may describe a configuration by a list of real parameters. For example, the position T can be simply described by the vector of the N coordinates of F_A in

F_W. The orientation Θ can be described as the $N \times N$ matrix whose columns are the components, in F_W, of the unit vectors along the axes of F_A. Then $\mathbf{q} = (T, \Theta)$ is uniquely represented as a list of $N(N+1)$ parameters. Clearly, however, this representation is redundant, since the matrix describing Θ must have orthonormal columns and determinant of $+1$. Therefore, C is only a subset of $R^{N(N+1)}$.

There are many different methods for generating the configuration space. These are highlighted below [25]:

1. Point evaluation
2. Sweep volume
3. Minkowski set difference - This effectively grows an obstacle using the shape of a robot and a reference point on the robot. This type of workspace generation can be seen in figure 2.9.
4. Boundary equations
5. Slice projections
6. Templates
7. Jacobian based methods

Another method for generating the configuration space is presented by Kavraki [26]. This method uses the Fast Fourier Transform (FFT) to compute the convolution between the robot and the workspace. This method is useful when the robot is a complex shape and can benefit from specific hardware developed for the FFT algorithm.

Consider a robot A. A is a rigid body and moves in the workspace $W \subset \Re^N$, N is 2 or 3. β denotes one of the objects in W, and C the configuration space of A. The subset of W occupied by A at the configuration $\mathbf{q}$ is $A(\mathbf{q})$.

$$A(q) = A + q = \{x + q : X \in A\} \tag{2.1}$$

$$C\beta = \{q \in C \mid A(q) \cap \beta \neq \emptyset\} \tag{2.2}$$

The obstacle β maps to the region $C\beta$ within C - the c-obstacle. The configuration space can be computed in two dimensions as the convolution of the workspace and the translating robot. Consider a workspace W,

$$W = [a, b] \times [c, d] \subset \Re^2 \tag{2.3}$$

W is discretised into a $N \times N$ array,

$$Cell_{ij} = \left[a + \frac{i(b-a)}{N}, a + \frac{(i+1)(b-a)}{N}\right] \times \left[c + \frac{j(d-c)}{N}, c + \frac{(j+1)(d-c)}{N}\right] \tag{2.4}$$

where

$$i, j \in S = 0, \ldots, N-1$$

If there is an obstacle anywhere in the cell ij we let $W(i, j) = 1$, else $W(i, j) = 0$. The robot A can be approximated by a set of points $(i, j), i, j \in S$. For each fixed value of (x, y, θ) we consider the $N \times N$ bitmap array $A(x, y, \theta)$, where only the points that belong to the robot are marked with ones. A point (x, y, θ) in the (discrete) configuration space is free if and only if,

$$C(x, y, \theta) = \sum_{i,j=0}^{N-1} W(i, j) A_{(x,y,\theta)}(i, j) = 0 \tag{2.5}$$

We observe that whenever θ is fixed and x, y are varying, the various bitmaps $A(x, y, \theta)$ are all translations of each other and in particular of $A(0, 0, \theta)$. Then

$$C(x, y, \theta) = \sum_{i,j} W(i, j) A_{(0,0,\theta)}(i - x, j - j) \tag{2.6}$$

This is the convolution of W and A'_θ where

$$A'_\theta(i,j) = A(0,0,\theta)(-i,-j) \tag{2.7}$$

The convolution of two arrays Q and T is defined as ,

$$Q * T(x,y) = \sum_{ij} Q(i,j)T(x-i,y-j) \tag{2.8}$$

Hence,

$$C(.,.,\theta) = W * A'_\theta \tag{2.9}$$

The configuration space is the building block for many path planning algorithms, the next sub-sections describes different path planning procedures that can be applied to it.

2.4.1 Roadmap Methods

Roadmap approaches attempt to reduce the configuration space to a network of one dimensional curves, the roadmap. If the start and goal configurations are linked to this map then path planning becomes a graph search problem. The search can be performed using a variety of approaches, but among the most used is the A* algorithm. This searching method uses a heuristic measure to find minimum cost paths. It is shown that the A* algorithm is *admissible*, that it will always find the shortest path, and it is optimal [27].

The Sub-Goal Method The *Sub-Goal* Method generates a list of configurations that are reachable from the start point. A local operator is used to test the reachability between one configuration and the next. A simple local operator would be to join the configurations with a straight line if no collisions occur.

The initial use of the local operator is to test whether the goal is reachable. If it is not, a list of candidate intermediate configurations is generated, these are the subgoals. The local operator is used then to see which of these subgoals is reachable. This process is repeated until the goal is reached. The effectiveness of this method depends on the local operator and the generation of subgoals. The *viability graph* is one type of subgoal network commonly used.

The Visibility Graph The Vgraph is a roadmap that can be used in two-dimensional Cspace. It is constructed by using all the vertices of the obstacles and the start and goal configurations as the graph nodes, and the links are the line segments which connect two nodes without intersecting any obstacles.

An improvement to this method is the reduced visibility or the tangent graph. This graph is produced by removing all the line segments that are not tangents of the obstacles. This reduces the size of the graph which can make the structure quicker to search. The tangent graph will always contain the shortest path between the start and the goal, which will always be found when using the A* search.

For configurations of three dimensions or higher, the visibility graph cannot be applied as effectively. However several authors have proposed methods to improve the situation.

The Voronoi Diagram A Voronoi diagram contains the set of points that are equidistant from two or more object features. The space is partitioned into regions each containing one feature and any point in a region is closer to the feature in that region than to another feature. The resulting roadmap is the edge between regions. This edge is the safest possible path as it avoids obstacles by the maximum distance possible. The previous method produced paths that were optimal in distance and for some path planning applications this is preferred. However, for the application discussed here safety is the overriding concern and methods that produce optimally safe paths are preferred.

In higher dimensional workspaces, the boundaries between regions become sur-

faces. This graph is no longer a roadmap and in general is not suitable for path planning.

The Silhouette Method This method is suitable for constructing roadmaps in arbitrary dimensions [28]. Objects in higher dimensional space are projected into a lower dimension space and the boundary of the curves are traced, giving the silhouette. This reduction is performed recursively until they produce a set of one dimensional lines.

The final network of these lines produces the roadmap. The path is searched for by connecting the start and goal to this network. The method generates paths that trace around the edges of obstacles which may be desirable in some planning application but not to all.

2.4.2 Cell Decomposition Methods

The cell decomposition methods are characterised by the representation of a workspace as a series of smaller simple regions. These regions are called cells. A non-directed graph can be created depicting the adjacency relationship between cells which can then be searched. This graph is called a connectivity graph. The nodes of the graph are freespace cells, cells are only linked if they pass an adjacency rule. Searching this graph from a start node, to find a goal will only be successful if a channel can be found linking the start and goal. The method can be divided into two categories, exact and approximate.

Exact Cell Decomposition Methods Exact cell decomposition methods decompose the free space into cells whose union is exactly the free space. A function that decomposes the workspace should have the following attributes: [29]

1. The geometry of the cells should be simple, and
2. It should not be difficult to test the adjacency of any two cells.

It follows from these requirements that the boundary of a cell corresponds to a criticality, that is something significant changes when the boundary is crossed.

Approximate Cell Decomposition Methods Approximate decomposition methods use cells of predefined shape whose union is strictly included in the freespace. The boundary of a particular cell does not characterise a discontinuity of any sort and has no physical meaning.

These methods are realistically applicable only when the dimension of the workspace (n) is small, say $n < 4$. Specific tricks exploiting the structure of a particular task domain may possibly be used to develop planners working in higher dimensions [29].

Approximate cell decomposition has been investigated further than the exact methods since they are usually much easier to implement, thus requiring less computational power and are less sensitive to numerically approximate computations.

The general description of approximate cell decomposition is given in appendix B.

The basic cell decomposition method can be improved by using a *divide and label* algorithm. The most widely used technique is to compute a 2^m-tree decomposition, where m is the dimension of the configuration space.

A 2^m-tree decomposition of Ω is a tree of degree 2^m, each cell which is not a leaf cell has exactly 2^m children. Each cell of the tree is a rectangloid cell which is labelled as EMPTY, FULL, or GREY. The root of the tree is Ω. Only cells which are mixed may have children and they each have 2^m of them. All the children of a cell k have the same dimensions and are obtained by cutting each edge of k into two segments of equal length. If $m = 2$, the tree is called a quadtree, if $m = 3$, it is called an octree.

The root node of the tree represents Ω, the depth of a particular cell determines its size relative to Ω. The height of the tree h determines the resolutions of the

decomposition of Ω. The higher the tree the greater the resolution.

In the worst case the number of leaf cells in a 2^m-tree of height h is 2^{mh}. This increases exponentially with both the dimension of $\mathcal{C}$ and the depth of the decomposition. In practice, this number is significantly less, since the tree is pruned at every empty and full leaf.

2.4.3 Potential Field Methods

Potential field methods use the principles of electric field theory to model the workspace, obstacles and the robot. Imagine the robot, a, is a positively charged particle, and the goal, b, is a fixed negative charge. An attractive force exists between the two charges which can be expressed as:

$$F_{ab} = \frac{1}{4\pi\epsilon_0}\frac{q_a q_b}{r^2}\hat{r} \tag{2.10}$$

where ϵ_0 is the permittivity of freespace of freespace and r is the distance between the two particles. The field produced by the goal can be expressed as:

$$E = \frac{1}{4\pi\epsilon_0}\frac{q_b}{r^2}\hat{r} \tag{2.11}$$

The important features of this field are:

1. The field magnitude E is proportional to $|q|$.
2. E is proportional to $1/r^2$
3. The vector E points directly away from a positive charge, or directly towards a negative charge

So considering the robot to be initially released from rest in a uniform field, it will move with constant acceleration along the line parallel to E.

In this way we can produce a field that will attract a robot to a goal configuration. However, how do we represent obstacles in this configuration? Just as the goal attracts the robot the obstacles need to repel it. To achieve this the charge on obstacles are set to be the same as that of the robot. Using the principle of superposition we can model the forces on the robot a due to both the attractive force from the goal b and the obstacles $ob_1, \ldots, ob_n$, modelled as point charges as:

$$F = \frac{1}{4\pi\epsilon_0}\frac{q_a q_b}{r_{ab}^2}\hat{r}_{ab} + \frac{1}{4\pi\epsilon_0}\frac{q_a q_{ob_1}}{r_{aob_1}^2}\hat{r}_{aob_1} \tag{2.12}$$

$$F = \frac{q_a}{4\pi\epsilon_0}\sum\frac{q_i}{r_i^2}\hat{r}_i \tag{2.13}$$

Thus the resulting potential field can be represented as:

$$E = \frac{1}{4\pi\epsilon_0}\sum\frac{q_i}{r_i^2}\hat{r}_i \tag{2.14}$$

Figure 2.10a shows an example of an attractive field, while 2.10b show the repulsive field for a point obstacle. Consider the robot starting from rest, it would follow the contour of the field avoiding the obstacles and proceeding to the goal location. The combined attractive and repulsive fields are shown in figure 2.10c, a contour plot of the combined field is shown in figure 2.10d.

Unfortunately, the robot may not get as far as the goal. This is due to the generation of local minima in the field from the position of the obstacles. This is the major problem in simplistic implementations of the potential field methods for path planning.

Potential methods have the advantage of being fast and efficient planning methods. However, this speed is reduced when methods for avoiding local minima are introduced.

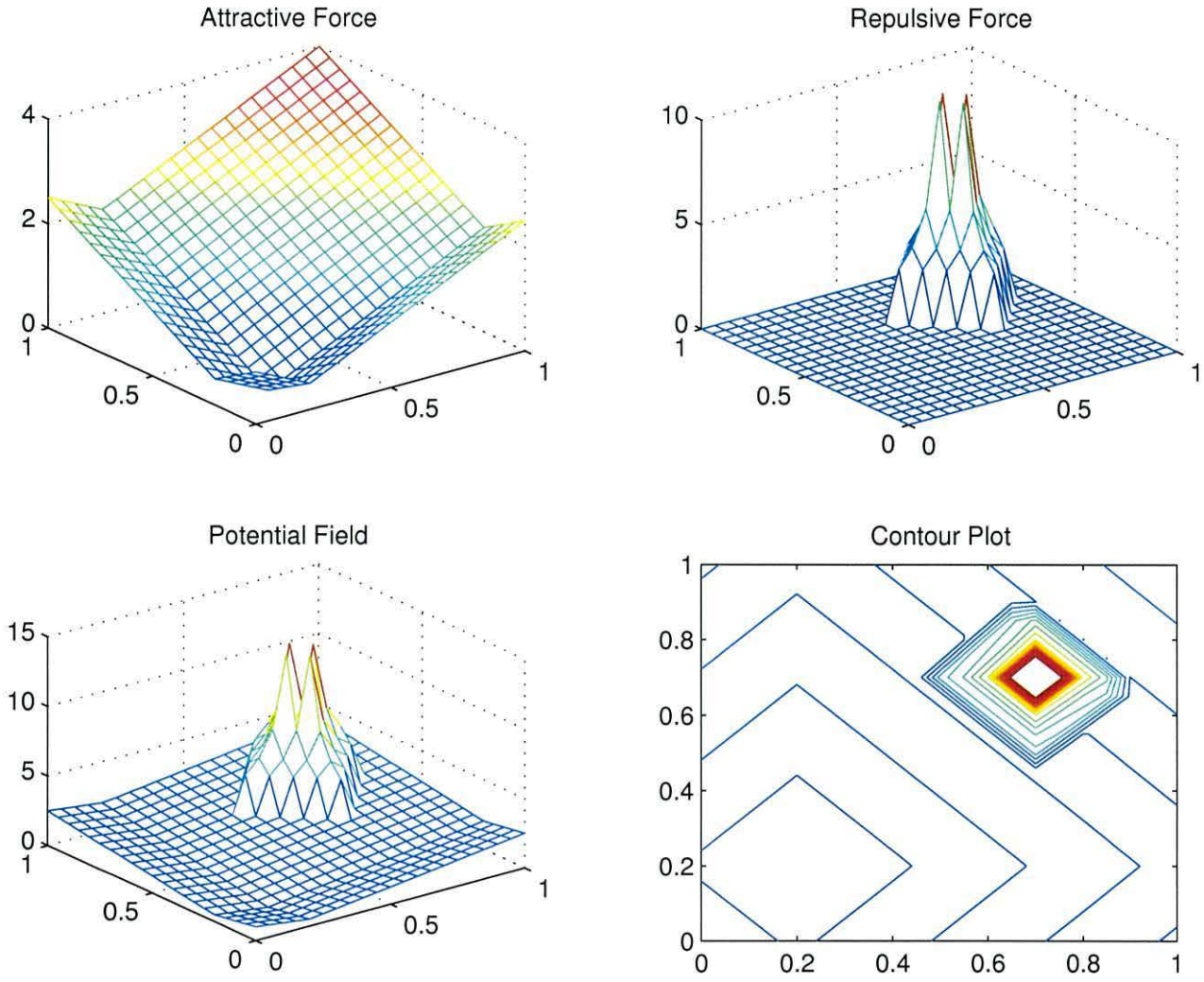


Figure 2.10: Potential Fields a) Attractive Field: Goal at lower left, b) Repulsive Field : Point obstacle at upper right, c) Combined Fields, d) Contour Plot of the Combined Fields

2.4.4 Path planning with the Distance Transform

The decision to use a distance transform path planning algorithm was made after studying the advantages and disadvantages of the planning methods given above. The key requirement for this application is safety. To be safe it is necessary to plan paths in three dimensions quickly and reliably. The ability to plan for multiple goals is also an advantage as it allows several routes to be planned in one operation. This is useful as it has been suggested that a reflex avoidance system would be required just to handle the case where a fast moving obstacle

enters the workspace of the helicopter. There would not be enough time to plan a standard path to continue the inspection process, but there should be enough time to avoid a collision.

Chapter 5 gives an in depth investigation of the distance transform.

The Distance Transform

The distance transform itself is not a new idea. The first reported use of the method for path planning was described by Jarvis and Byrne [10] in 1986. In this approach an attempt is made to find a route from goal location back to a start location in two dimensions. A uniform grid is placed over the workspace, and in Jarvis and Byrne's implementation each cell within the grid has 8 neighbours (they are 8 connected). The distance transform propagates from the goal location marking all freespace cells with an incrementing distance value. Once all cells that are not obstacles have been marked, a search from a selected start position can be made. If the start location has been marked with a distance transform value a path is possible, otherwise a path does not exist. The path is formed by walking downhill from the start via the steepest descent path.

The generation of the distance transform can be computationally expensive, especially as the resolution of the grid placed over the workspace increases. However, there are several advantages to this method. Firstly, the shortest path from any point to the goal is known for all freespace cells, thus multiple robots can be supported by the same planning algorithm. Secondly, the distance transform supports multiple goals. Also, the distance transform provides the feature to favour or avoid certain regions. All these features are attractive for use in our path planning system.

Kambhampati and Davis [30] present a multi resolution path planning method for mobile robots using a quadtree representation of the workspace. Given a binary representation of the robot's environment, the first step is to reduce the robot to a point by growing the obstacles by the radius of the robot's cross section. This process is widely used in path planning. They use a raster to quadtree algorithm

developed by Samet [31]. The complexity of this algorithm is $O(n)$ where n is the number of pixels in the raster being converted. The next step is to calculate the distance transform of the freespace nodes of the quadtree. This determines the minimal distance between the centre of a node and the boundary of an obstacle.

Samet [32] presents an algorithm for computing this distance transform which has complexity $O(n)$, where n is now the number of leaf nodes in the quadtree. The number of leaf nodes in a quadtree of an image map having polygonal obstacles is approximately $\frac{2}{3} \cdot O(p)$, where p is the sum of the perimeter of the (polygonal) obstacles in terms of the lowest resolution units [30]. The method then uses an A* search algorithm to attempt to find a path from start to goal. This A* search will only have to deal with about $O(p)$ nodes in the case of a quadtree, instead of the n^2 grid points in the case of a grid based search.

In Samet's method the distance transform is used in a different way to that presented by Jarvis and Byrne [10]. The measure of distance of a node is the distance from that node to an obstacle, rather than from that node to the goal. Shin [8] presents a similar method.

Subsequent authors have used the implementation of Jarvis and Byrne and improved the method by using different methods to model the workspace. Zelinski [9] presents an exploration algorithm using quadtrees and the distance transform. Initially the workspace is totally unknown. The robot knows its location and that of the start and the goal. The notional size of the workspace is $m \times n$ and the position reference of the robot is (x, y, q) relative to some reference point. A quadtree Q is generated of sufficient size to cover the workspace. The smallest quadtree leaf resolution size is of diameter d, a size that allows the robot to pass through. The size of quadtree is $Q = (2d)i$, where i is an integer such that $(2d)i > s$ and $s = max(m, n)$. The planner assumes that all unknown areas of the workspace are unoccupied. So initially a quadtree exists which represents complete freespace. Each node is marked with a distance transform value of ∞ (in practice it is the largest available integer). The distance transform is then calculated. Path planning is then performed to find a path from start to goal. Obviously the first path is just a direct route to the goal as no obstacles have been discovered.

The robot then starts moving on this path. One of two conditions exists during this motion: either the robot reaches the goal or it encounters an obstacle (in practice this may be detected by a tactile sensor). If an obstacle is discovered then the node corresponding to the obstacle is marked as such. The distance transform is then recalculated and a new path is generated. This sequence of detecting an obstacle, updating a quadtree and replaning the path continues until either the goal is reached or the path planner reports that no path is possible. The planner can definitively report that no path is possible when the distance transform value at the robot's current location is infinite as it has not been marked by a distance transform value.

2.4.5 Spatial Decomposition

The efficient storage and processing of workspace data for any path planning method is important. In applications operating in more than two dimensions it is critical.

An early driving force for the development of spatial decomposition was image processing. At the time computer memory was expensive and data transmission speeds were low. For example a 1024×1024 binary image requiring 1Mbit of memory to store, and transfered via facsimile at 9600 bits per second would take almost two minutes. For instance, an average length academic paper, say 10 pages, would take 20 minutes to transfer which was expensive if one was sending it from the UK to the USA. This stimulated many ideas for the compression of two dimensional images.

A number of different planar decomposition methods exist. In all the above examples, and the method we propose, square nodes are used as the building blocks. The reason squares are used in the majority of cases is that:

1. They yield a partition that is an infinitely repetitive pattern so that they can be used for workspaces of any size, and
2. The partition is infinitely decomposable into increasingly finer pattern.

Other shapes also satisfy the points above, one example is decomposition into four equilateral triangles. However, triangular decompositions do not have an uniform orientation. Decomposition into hexagons has a uniform orientation but do not satisfy point 2 above [33].

One motivation to use space decomposition is to save space. As mentioned above we predict that the environment local to the helicopter will not contain many obstacles, in fact in most cases it will be empty. It would be wasteful in both space and time to operate on a raster representation that was empty or in which only a small percentage of the workspace was used. This is particularly important as our requirement is for three dimensional planning. We have used quadtrees and octrees to represent the decomposition of space.

The first implementations of quadtrees used pointers to link nodes together. Each node has one parent pointer and four children pointers. The tree structure is built by linking these pointers to each other. The root node has no parent so its parent pointer is empty, also leaf nodes have no children so the children pointers are empty. This implementation obviously adds an overhead to any method using it.

Two further implementations have been proposed. The linear quadtree treats the image as a collection of leaf nodes where each node is encoded by a base 4 number termed a location code. This location code is a sequence of direction codes that locates a leaf along the path from the root. For example, node 123, means that the node is child three of node 12, which is in turn child 2 of node 1, which is child 1 of the root node. The second implementation, known as the DF-expression [34] (Depth First expression), represents an image as a traversal of the nodes of its quadtree. This method is very compact as it only requires two **bits** to represent a node. However, this method is not suitable when random access to nodes is required. Samet and Webster show that for a static collection of nodes, an efficient implementation of a pointer based representation will often be more economical than a location code representation. This is especially true for higher dimensional images.

The modern computer has an abundance of memory available, so saving space is not such an important factor. In many cased a binary array representation may

still be more economical than a quadtree [33]. However, efficient representation of data has an important effect on execution time of algorithms operating on them. Hunter and Steiglitz [12] present the Quadtree Complexity Theorem. This states that the number of nodes in a quadtree is proportional to the perimeter of the image. In other words in the resolution doubles the perimeter doubles thus the number of nodes will double. Compare this to an array representation, when the resolution doubles the number of elements in the array quadruples. The Quadtree Complexity theorem also holds for three dimensional data [35] were perimeter is replaced by surface area.

Consider the representation of a cube shaped space. For an octree decomposition representation based on the occupancy of the cube the number of octree nodes is proportional to the surface area, ie $6 \cdot l^2$. If this were to be stored as a simple matrix the number of elements would be proportional to the volume l^3. Now assuming that the simple matrix representation uses one byte of memory to store an element and, say that each octree node requires 50 bytes to store information about a node (a figure derived from software developed during this project), a comparison can be made. Figure 2.11 shows the comparison between memory requirements as a log-log graph.

In figure 2.11 the blue line shows the memory requirement of a simple matrix based storage of the cube space, and the green shows that required by an octree method. At low values of l the matrix representation is clearly more efficient, but as l increases the difference becomes smaller and eventually at values of l of more than 150 it is more space efficient to use the octree representation.

Chapter 5 presents a new method for path planning based on the distance transform and quadtree/octree decomposition.

Path planning is of no use without a sensor system to detect obstacles, unfortunately tactile sensors are out of the question! The next section will give a description of the sensor systems researched and will justify the decision to use vision.

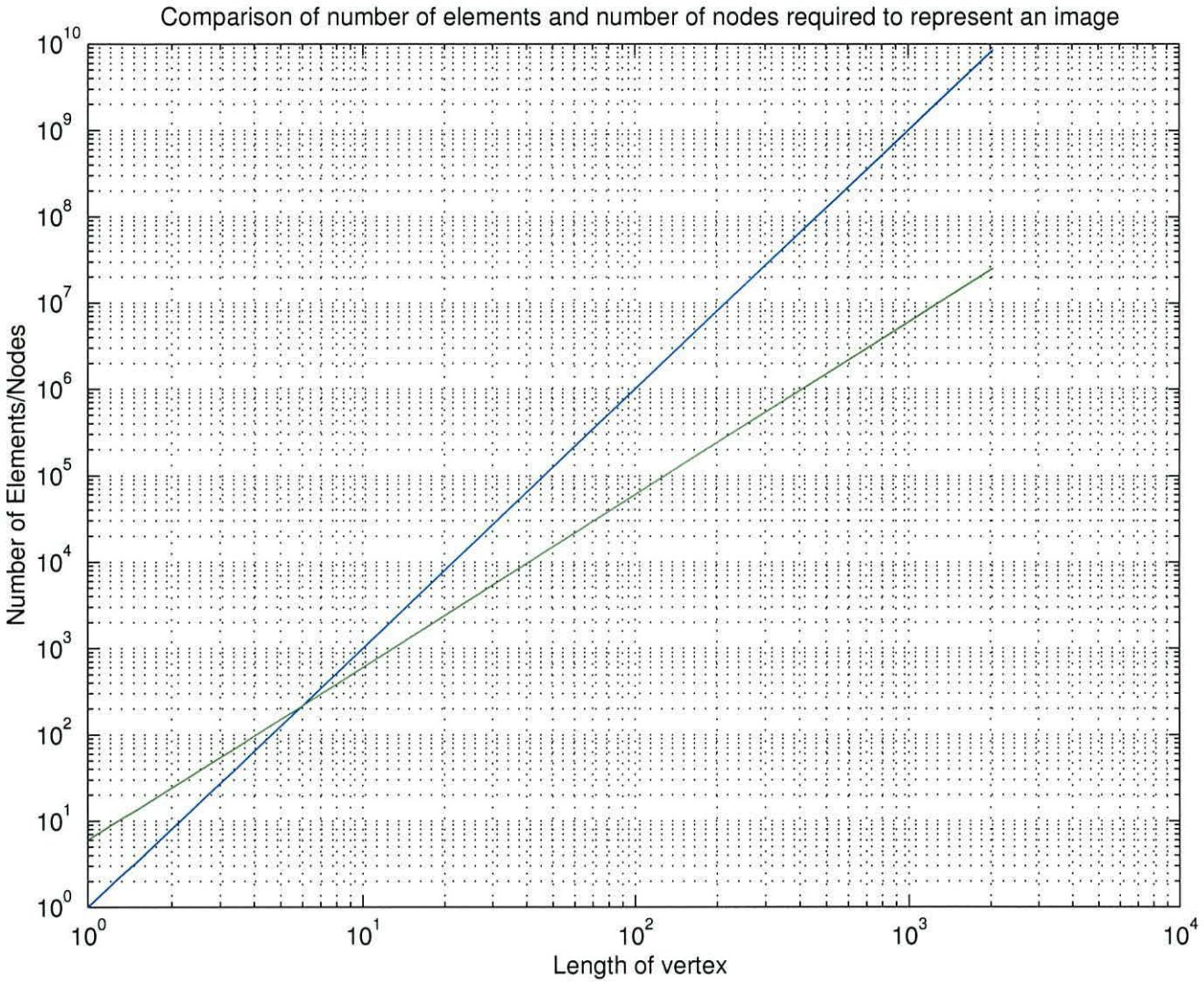


Figure 2.11: Memory requirement comparison of matrix (blue) and octree (green) based algorithms

2.5 Machine Vision Literature

Machine vision is the technology of using sensors and computers to make sense of the real world in a way that is comparable to humans. Machine vision systems are applied to tasks such as part placement and obstacle detection for mobile vehicles.

However, cameras are not the only sensor that can be used for this purpose. There are two distinct methods of sensing, passive and active. Video cameras and

thermal imaging cameras are passive as they do not emit any energy to acquire the information about a view. RADAR and SONAR are examples of the active type of sensor, these do emit energy and use the returned energy to acquire information. Another distinction between the two is that generally passive methods generate two dimensional information about an environment, while active sensors can build up a three dimensional picture without using computer processing.

The ability to build up a three-dimensional model of an environment is important for mobile robotics. Without knowledge of what is in the vicinity of the robot other functions, such as path planning, cannot work. This section will outline methods that have been found in the literature to produce three dimensional maps using a variety of different sensors, both passive and active. The results show that camera based vision together with computer processing prove no worse than other types of methods in terms of performance, and also show why, in certain categories, they are better for the intended application.

Typically, RADAR devices for microwave imaging designed for long distance ranging or radiometric purposes have poor range resolution and operate at a wavelength of several centimetres. Rožmann and Detlefsen [36] describe the use of a three-dimensional imaging RADAR for a ground based autonomous robot. The sensor they use is a monostatic pulse Doppler radar front end with a scanning deflection unit. A 20cm nearly parallel beam is directed by a reflector controlled mechanically by two servo motors which give an angular range of 360° in azimuth and a 25° in elevation. The carrier frequency used is 94GHz (λ=3.2mm), at a pulse peak power of 10mW. Their intended application is for a land based robot which operates in a human environment, so the power level is kept to a minimum for safety reasons. They claim a distance accuracy of less than 17mm, an angular accuracy of 0.125° (both over 50m) and a velocity accuracy of 1mms^{-1} (1×10^{-3}ms^{-1}) over a measurement domain of $\pm$8ms^{-1}. This sensor gives the robot a great deal of information such as distance, speed and bearing. A map of the environment can be readily built up from this information and then passed on to the robot system.

Three-dimensional laser range sensors operate in a similar way to RADAR. Horn and Schmidt [37] present a method for continuous localisation of a mobile robot

based on such as sensor. The sensor is designed for eye safe (emitted laser power less than 4.5mW) indoor operation. A collimated laser beam (λ=810nm) is directed onto the scene, and the back-scattered light is measured by a photodiode. The distance between the sensor and the object is determined by measuring the phase shift between the emitted and received beams of modulated (AMCW) laser light. A two frequency phase shift method provides high resolution, 0.45mm, over a 15m range. Three-dimensional range data within a field of view of 52°×57.6° (elevation×azimuth) is obtained by scanning the environment with two synchronised mirrors.

SONAR plays a vital role in the operation of Autonomous Underwater Vehicles (AUV). Wright et al. [38] shows the integration of several types of sensor, including SONAR, electro-optic and magnetic, in a vehicle designed to assist in the clearing of unexploded ordinance from shallow water areas. In this application two side-looking SONARs are used, one low frequency and the other high frequency. The low frequency sensor has a range limit of 37m and a range resolution of 7.5cm, and can penetrate soft ground up to a depth of 2m. The high frequency SONAR also has a range of 37m and a range resolution of 5cm but does not penetrate the ground significantly.

SONAR is governed by the speed of sound in the vehicle's environment. In air the speed of sound is about 300ms^{-1}. For a 100kHz signal (λ=3mm) there is a 5dBm^{-1} attenuation of the signal, the attenuation of the same frequency electromagnetic signal is approximately 1.3×10^{-3}dBm^{-1}. For underwater applications the speed of sound is about 1445ms^{-1} (at 0°C) and for a 100kHz signal the resulting attenuation is approximately 0.03dBm^{-1}. SONAR is the choice of AUVs as it can give measurements of distance over a large range from a relatively low power transducer, it is also unaffected by particles suspended in the water - just as infra-red can "see" through smoke, SONAR can "see" through turbid water which is useful in shallow water or near sea bed operations.

Passive sensors, such as video cameras, have been used for navigation and obstacle detection for manned rotor craft [39, 40, 24, 23, 22]. These researchers show how vision can be used, sometimes in conjunction with other types of sensors, to improve safety during low altitude flight. The systems are used to allow the pilot

to concentrate on mission tasks rather than flying the vehicle.

The area of machine vision of importance to this project is obstacle detection and location. Obstacle location (for both autonomous land and rotorcraft) using passive sensors relies on two fundamental techniques for range estimation: binocular stereo and motion stereo. Binocular stereo uses two laterally spaced cameras and feature matching between two images to calculate a range estimation to objects. Motion stereo uses one sensor from which images are collected as it moves. By observing the amount of image plane motion that a particular world point exhibits between frames and using information about the sensor motion, range to the world point can be computed. The amount of image motion can be computed either using the optical flow method [41] or by using feature matching between a temporal sequence of two or more images.

Bhanu et al. [39] give a description of a sensor suite that could be used in a military helicopter operating in Nap-of-the-Earth flight. They discuss what combinations of sensors give the best results and also outline the drawbacks. With purely *motion stereo* systems the obstacle must first be detected before an estimation of range can be calculated. Since the range from motion is not computable at the focus of expansion (FOE - the point in the image plane corresponding to the instantaneous velocity or trajectory direction) a purely motion stereo system will fail in exactly the direction which is most relevant. A *binocular stereo* system can provide range estimation anywhere in the stereo field of view (FOV). However, it is not a good choice for detecting small obstacles with little or no features. For example, a FOV of $\sim 4°$ would be needed to detect wires 3mm in diameter at a range of 40m. They propose using binocular stereo and LIDAR, to give a "maximally passive" system for obstacle detection and avoidance.

Sridhar and Chatterji [22] present a vision-based obstacle detection method for helicopter guidance. They use optical flow methods together with Kalman filtering to give improved accuracy and predict future obstacle locations. They present simulation results based on images recorded on actual low level flight over a runway containing a number of obstacles. The results show how obstacle points are detected and grouped, and then located in world coordinates.

Sridhar and Phatak [24] give an analysis of an image-based navigation system for

low level rotor craft flight. They divide the decision making process into three levels, the farfield, the midfield and the nearfield and use different sensors for each. For farfield use they suggest a course digital elevation map of the whole mission area, and for the mid field a high resolution map which can generate trajectories for a short duration ahead. The nearfield uses the midfield generated trajectory and replans this when obstacles that are not contained in the environmental maps are detected using either low-light-level cameras or forward-looking infrared detectors.

The majority of references found on the use of vision in rotorcraft are based on military applications were the main objective is to move away from active sensors to passive sensors. Active sensors emit a characteristic signature, which can be detected by an enemy, passive sensors do not, making them a safer system in battlefield situations. The following part of this section will investigate other uses of passive vision systems for guidance and obstacle avoidance which are mostly focused on ground-based vehicles.

Rožmann and Detlefsen [36] used a scanning RADAR to give a 360° azimuth and 25° elevation detection area for a mobile robot. A 360° coverage in azimuth can be achieved in a vision system either by using multiple cameras with overlapping fields of view or by using a single camera and a conic mirror as described by Yagi, Nishizawa and Yachida [42]. The conic mirror they use gives an elevation range of about 22° which allows the vehicle to see the ground around it and objects above it. Using a predefined environmental map together with information from the vision system a path is generated allowing the robot to move from a start to a goal without colliding with any obstacles.

Several methods for using vision to estimate object depth and position are given in the literature. The majority are based on binocular stereo or motion stereo principles. Dalmia and Trivedi [43] propose a method of depth extraction using a single moving camera by combining these two principles. They claim a mean error of less than 3% in depth estimation by using spatial and temporal gradient analysis. Mitiche et al. [44] use a method based on the conservation of distances in rigid obstacles and show how object location and camera motion can be recovered from a sequence of images.

Zhuang and Shieh [45] report an approach for computing depth maps from monocular image sequences with known camera motion. They use a combination of the direct depth estimation method with an optical flow based method to improve the overall estimation accuracy. They assume that the camera motion is known, the objects in the environment are rigid bodies, the image is changing slowly between two successive frames and that in a small interval of time the image brightness remains unchanged. They solve the depth estimation problem using a Kalman filter in three stages. In the prediction stage, the depth map of the current frame, together with information about the camera motion, is used to predict the depth and depth variance at each pixel in the next frame. The estimation stage refines the predicted depth for the next frame. The final stage, smoothing, is used to reduce measurement noise and to fill in untrustworthy areas. They produce simulation results on both synthetic and real images and show that using both gradient methods and optical flow methods can improve the operation of a depth estimation process.

Horn and Weldon [46] present direct methods for recovering motion of the observer in a static environment. Their methods work with cases of pure rotation, pure translation, and arbitrary motion when rotation is known. The methods do not establish point correspondence between frames, nor do they calculate the optical flow. They point out that the field of view should be large to accurately recover the components of motion in the direction towards the image region, and also discuss the problems resulting from very large depth ranges.

Observations

The discussion above gives details of different methods to accomplish the task of obstacle detection and location. The aim is to show that camera based vision can perform as well as active methods, and for the intended application have certain advantages over them. The active sensors that would be applicable for range and velocity estimation are RADAR and laser based systems, SONAR can be discounted as the attenuation characteristics of the atmosphere means that an acceptable range specification could not be achieved.

Rožmann and Detlefsen [36] state that a current drawback of RADAR is its high cost as integrated subsystems are not available. The same is true for Laser range finding as given in [37]. Both these papers report on the construction of their own sensor as nothing suitable could be purchased. This is one of the major drawbacks to this type of active sensors, in contrast passive vision equipment is readily available.

Another problem with the active sensors is their size, weight and power consumption. It would be difficult to incorporate a full RADAR system into the SPRITE helicopter, the sensor placement would be a problem due to shadowing from the helicopter's body resulting in blind spots in the sensory environment. A laser based system would have the same sort of problems. A vision system to give a full 360° field of view in azimuth is possible using a conic mirror assembly [42], but again this would have the problem of blind spots. The other way of giving 360° is to use multiple cameras mounted around the circumference of the vehicle, for example with each camera having a field of view (FOV) of 60° in azimuth, six cameras would be required. Using a 6mm lens mounted on a $\frac{1}{2}$" CCD camera gives an elevation range of 44° and a azimuth range of 57°. The choice of lens and the number of cameras depends on the resolution required from each camera - a large field of view means that the resolution detail of the images is low, while a high resolution image needs a small field of view resulting in more cameras.

The drawback of passive vision over active vision is the requirement to process the information gathered by the cameras to recover location and motion. It has been reported above that active sensors can do both tasks without the need for complex processing algorithms and independent of the viewing conditions. To recover depth either requires binocular stereo or motion stereo. The problem with binocular stereo is the need for a large base line for resolving the positions of distant objects. Motion stereo removes the need for this base line at the expense of requiring further processing of the images. However, the research presented above has shown that it is possible to recover obstacle locations and motion from monocular vision system in particular when camera motion is known. The majority of papers reviewed have limited their experiments to either computer simulations or live tests under strict laboratory conditions.

From studying the literature, judging the merits and drawbacks of the different types of sensors, it has been concluded that passive vision presents the best solution to the problem. The reasons for this conclusion is as follows:

- Passive vision offers a small volume and weight sensor system.
- Computer processing power is increasing, while the size and power consumption is decreasing.
- Vision equipment is readily available and, relative to RADAR, is cheap.
- A Multiple camera configuration can give full 360° coverage in azimuth and an elevation coverage as good as, or better than, active systems.
- There is no blind spot produced by the body of the helicopter as the cameras are mounted on its circumference.
- The passive system does not emit any harmful radiation.
- Machine vision techniques, such as motion stereo, looming, optical flow can give the same type of information at an accuracy equivalent to active methods.

The main drawback to relying on a machine vision system is that for very complex environments it would not be able to process all the information quickly enough to ensure safety. However, the machine vision system would never be alone, there are other sensors on the vehicle such as GPS, and the a priori information about the environment given by the GIS, which will all aid in the processing of the visual information. Also, at all times a human operator will be supervising the flight of the vehicle, ensuring that if there is a problem the vehicle can be safely recovered.

2.6 Summary

The need for visual inspection of overhead power lines is clear. The death of a young boy [16] due to the failure of an inspection system just goes to show

how important it is. The current methods of inspection are slow or do not give detailed results. Flying helicopters at low levels has its risks, several helicopters have crashed and their crew seriously injured or killed by flying into power lines [17, 18]. The aim of the RIPL project is to improve the inspection process, reduce the risks to the inspection personnel and to reduce the cost of inspection.

The Sprite helicopter was chosen as the airframe to provide the base for the inspection process. The CAA regulations require that a aircraft is always under the full control of a pilot. This means that a remotely controlled vehicles cannot venture outside the sight of the pilot. This, and other limiting factors, has driven the research into developing a system that can fly the helicopter autonomously, and therefore proving to the CAA that the vehicle is under the control of a "pilot".

There are two key areas of the CAA regulations. The first, *see and be seen* requires the pilot to be able to detect other aircraft. The second requires a pilot to fly safely avoiding obstacles such as other aircraft and keeping a safe distance from people, structures, vessels and vehicles.

The ability to "see" requires a sensor, the discussion above has reviewed the different type of sensor available and has shown that, for the RIPL project a machine vision has clear benefits over other types of sensors. In order to avoid obstacles a path planning system is needed. The information from the vision system plus information from other sources can be used to safely guide the helicopter through the inspection process while avoiding any obstacles detected during the flight. The different path planning systems described above all have the ability to produce flight paths for the helicopter, but the distance transform method has several advantages over the others such as quickly discovering when no path is possible and simultaneous multiple goal planning which add to the safety to the helicopter.

The combination of the machine vision system and path planning system within a controller hierarchy that splits the tasks into pre-flight operation, and real-time flight operations also improves the safety of the helicopter. The next chapter will detail the controller system leading on to subsequent chapters that detail the vision system and the distance transform path planner.

Chapter 3

Controller Architecture

This chapter will review the controller architecture of the autonomous unmanned air vehicle (AUAV). The controller architecture is a hierarchical structure containing three important levels: the *planner*, the *navigator* and the *pilot*. Each of these will be discussed in detail to show how they produce a coherent approach to the solution of the problems described in chapters 1 and 2.

3.1 System Architecture

There are many types of system architecture presented in the literature. The concept selected is based on Meystel's Intelligent Mobile Autonomous System (IMAS) architecture [7]. The architecture comprises of four levels, these being the *Planner*, the *Navigator*, the *Pilot* and the *Actuation Control System*. A pictorial representation of this structure is given in figure 3.1.

The division of responsibilities within this structure is as follows:

Planner This is the highest level controller. It is the main interface between the user and the vehicle. The operator enters the mission objectives and the *planner* then produces a mission plan detailing to the next level down the requirements of the mission.

Navigator Using the information from the level above the *navigator* produces a detailed path for the vehicle to follow using any a priori information that has been gathered. This path will satisfy the mission objectives as far as possible.

Pilot This is the lowest controller level investigated. The *pilot* is in charge of getting the vehicle from A to B, satisfying the mission objectives without colliding with obstacles. The *pilot* makes changes to the *navigator's* flight path to take into account any deficiencies in the *navigator's* information about the environment. The *pilot* will follow the *navigator's* flight path if no obstacles are detected.

Actuation Level This level is shown as the autopilot in figure 3.1. This controls the actuators of the vehicle. This level is machine dependent and as such will not be investigated further in this chapter as the low level controller for the Sprite helicopter is not known at present.

The IMAS architecture was chosen because it allows both off-line and on-line control of the vehicle. The *Planner* and the *Navigator* levels of the IMAS architecture can be preformed off-line as they operate with a priori information. The pre-flight planning is designed to produce the safest possible path given known information. The on-line control then only needs to react to unknown obstacles by modifying the path that was produced off-line.

Other controller architectures considered are all loosely based on this type of division of responsibilities. Meystel terms them *Nested Hierarchical Controllers.* They include intermediate levels between the three given above and some include details about the sensors system and low-level actuator control. It was decided to keep the controller architecture simple and if need be introduce complexities within each level. This is done in the *pilot* function where two types of obstacle detection system are employed, the first to deal with stationary or slow moving obstacles that can be avoided by path replanning, and the second to deal with fast moving obstacles that are handled by a reflex controller.

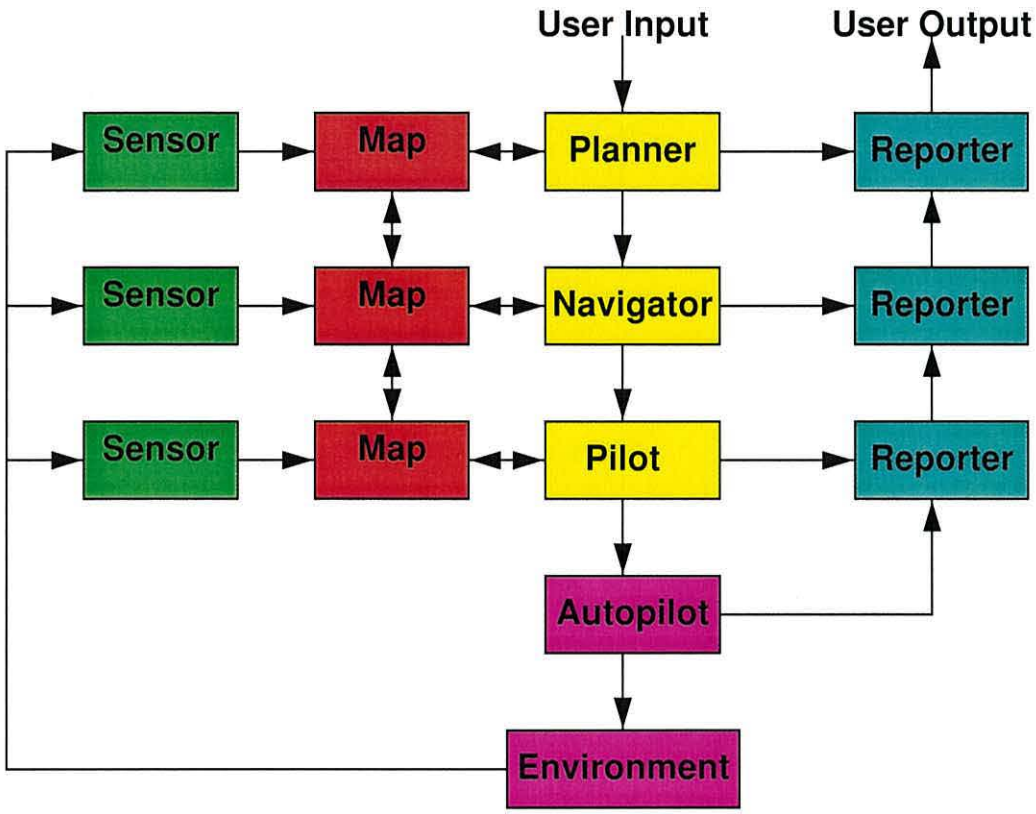


Figure 3.1: Hierarchical Controller Architecture

3.2 Planner

The *planner* is the top level of the architecture. This contains the user interface to the UAV, allowing an operator to select a series of poles that require inspection. It also contains a reporter which allows the operator to enquire about the state of the vehicle at any time.

In order to produce a maximally safe flight path the *planner* must have a priori information about the environment around the poles that are to be inspected. The inspection environment would be stored in a Geographical Information System (GIS). The GIS would contain information about the landscape, the use of the land, the type of land, the location of towers or masts, and the locations of the poles. The GIS would be updated after every flight to correct position errors and to introduce any new obstacles detected.

Using this a priori information would allow the majority of the planning stage to be performed off-line, prior to a mission.

3.2.1 Preferred Flight Space

The *planner* uses a GIS to produce a Preferred Flight Space (PFS). The PFS is a volume in which the helicopter flies during its mission. The PFS is **not** an exclusive airspace. The generation of the PFS is based on rules. The main rules are laws governing flying as laid out in the Air Navigation Order [29]. These will control the size of the PFS in height, minimum and maximum.

The maximum height for model aircraft as defined in CAP 658 [29] is 120m above ground level. There is no set minimum flight level, but in CAP 658 the recommendations are that any air vehicle should stay at least 50m from people, vehicles, vessels and structures. So a minium height of 50m would satisfy the best practice guide at all times. This would result in an height envelope of 70m.

The width of the PFS will be based on the dynamics of the helicopter allowing enough room for safe manoeuvring. The inspection equipment would also affect the width of the PFS, there will be a maximum distance away from the poles beyond which the inspection results would not be acceptable. The PFS is generated as a constant width and height volume of defined cross-section which follows the poles from start to end. This can be seen in figure 3.2.

The *planner* will use the concept of traversability. In ground based robotics traversability is used to describe the quality of the terrain. For example, on a map a lake would have a traversability value of 0 (for a purely land based vehicle) while a tarmac road would have a value of 1. The intermediate values would be assigned to intermediate ground conditions.

However, for an air vehicle, the condition of the ground is generally of little importance, other than at take off and landing or during an emergency so the meaning of traversability is different. Consider a wide area of flat land around the line of the poles. For this type of environment the traversability of the region

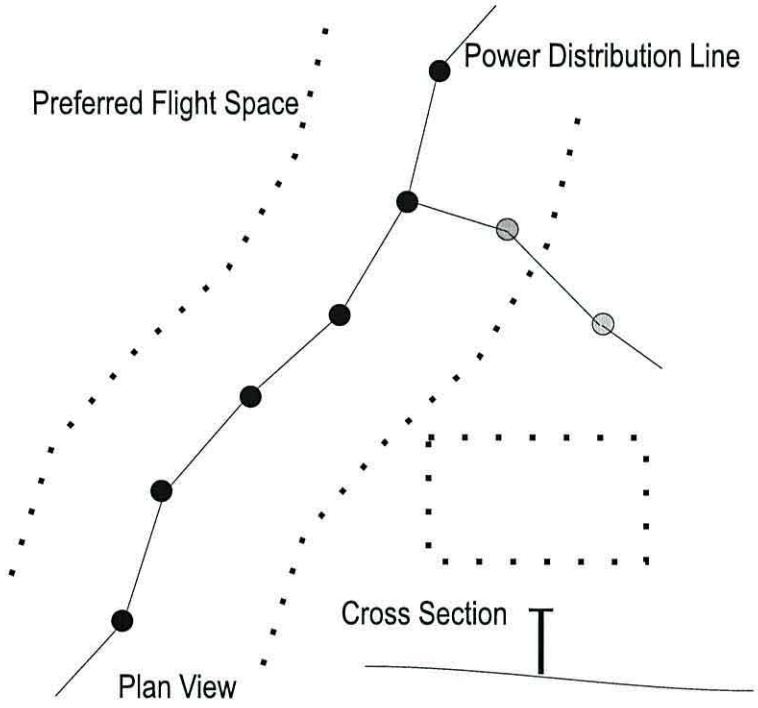


Figure 3.2: Preferred Flight Space: Conceptual visualisation

around the poles could be set to one. The reasoning behind this is that there are no known obstacles higher than our normal minimum flight altitude (50m plus safety margin) as defined in the CAP regulations.

In contrast, consider the case were the helicopter is required to fly up a valley to monitor the poles. As the poles are on the valley floor, there may be areas of the valley sides that could infringe on to the PFS, compromising safety. These areas would be allocated a low traversability value. These two examples are on either extremes of the environments that we envisage that our vehicle would encounter.

Another use for the traversability value is to identify areas where it would not be safe to fly for reasons other than the position of physical obstacles. Examples are near air fields, where other aircraft are likely to pass through the PFS, or target ranges which are specified as no fly areas. Areas where flying can be difficult, for example in mountainous areas where down drafts can affect the vehicle, could also be included but these factors would require local information from human pilots who know the area. Another use of the traversability factor would be to mark areas of farm land which contain animals that could be disturbed by the

noise from the helicopter. In manned flights, pilots are always on the look out for farm animals. In particular, horses seem to be easily distressed and can cause injuries to themselves. As stated previously, the goal is to minimise the risks both the the helicopter and others, before leaving the ground.

3.2.2 Modified Preferred Flight Space

So how would these traversability measures be used to improve the safety of the PFS? A number of methods could be used dependent on the rules adopted by the operator. A decision could be made to fly only in space of traversability values of one, ie where no known obstacles or other dangers will hinder the mission. Using this method if the PFS contains a hazard, then no path is possible. This is maximally safe, but not feasible. The MPFS can only be created within the bounds of the PFS, because as far as the *navigator* is concerned, the area outside is unknown.

The preferred option is to have a variable threshold. At the first pass test to see if a clear path is possible. If not, reduce the traversability threshold to (say) 0.9 and try again to produce a path. The reduction of the threshold value would be continued down to a predefined limit until a path is found. This is a better method than simply selecting an arbitrary lower limit as it will ensure that the safest possible MPFS is produced given the threshold degradation interval.

To illustrate this principle take, for example, the obstacle on a flat piece of land as shown in figure 3.3. The building could be a tall tower over 50m high, thus encroaching into the PFS.

How would the PFS be modified given an obstacle like that in figure 3.3? Figure 3.4a shows the obstacle near a stretch of power line together with the traversability values. The first test is to determine whether the PFS is an acceptable flight space. The safest path contains no obstacle regions, so the entire PFS has a traversability value of one. Figure 3.4a clearly shows that this is not possible.

Figure 3.4b shows that with a traversability threshold of 0.9 a path would be

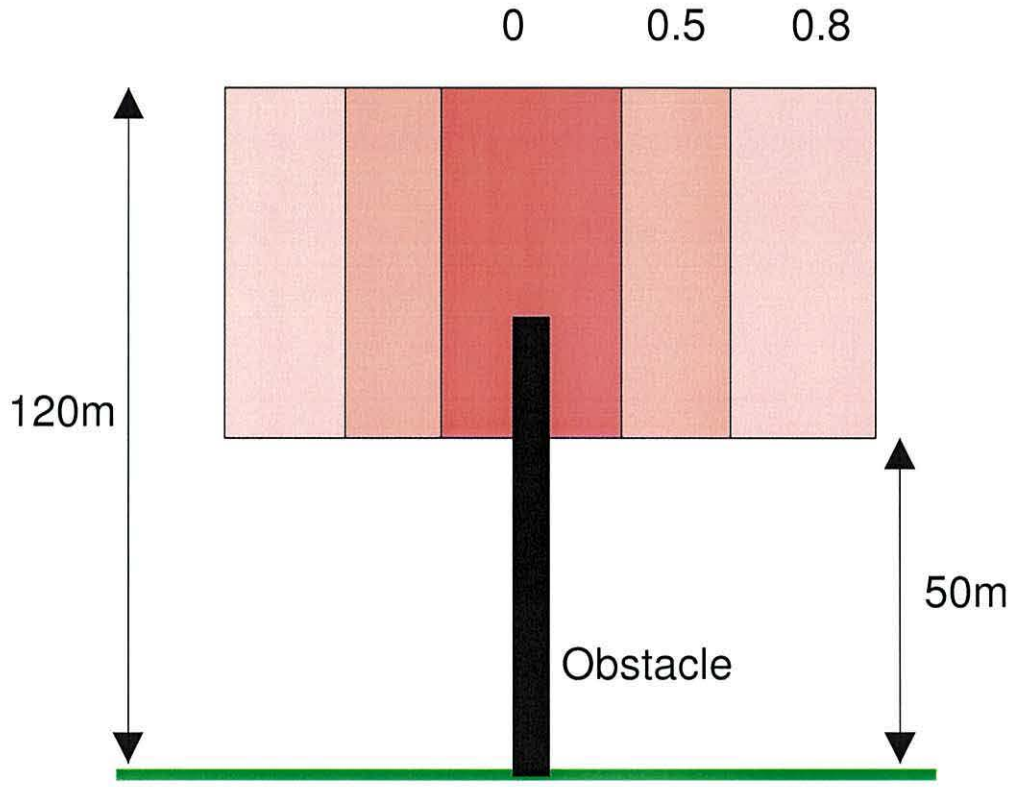


Figure 3.3: Example Obstacle highlighting how the PFS would look in cross section around an obstacle that encroaches into it

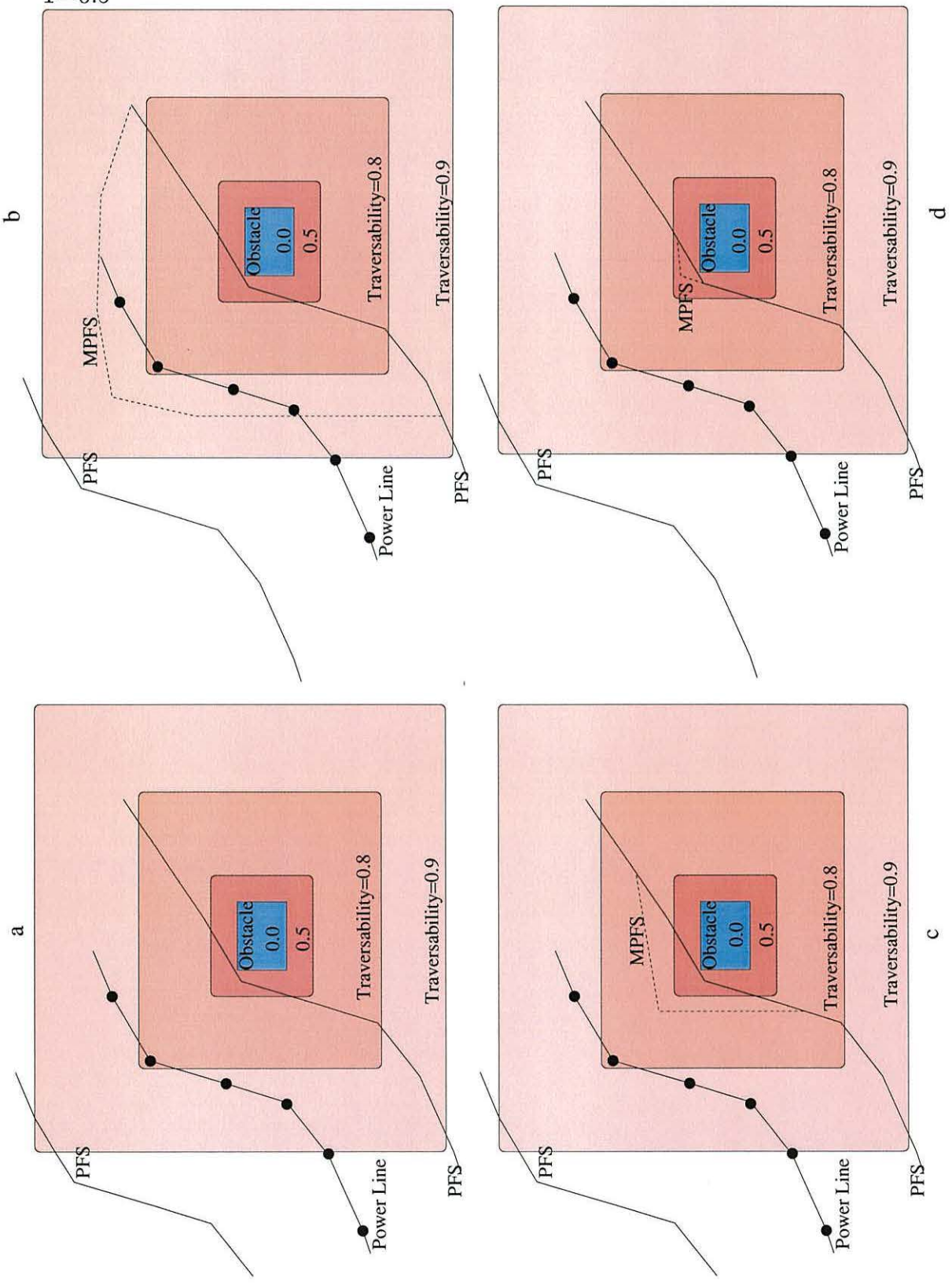
possible. However, the MPFS is not acceptable as the observer would not be able examine both sides of a number of poles.

Figure 3.4c shows the MPFS for a traversability threshold of 0.8 and figure 3.4d for a value of 0.5. In this case the traversability threshold of 0.8 would be acceptable giving the option of full inspection around the poles together with a higher level of safety. The lower the traversability value the higher the risk level of the MPFS.

The result of this process is a flight space that is maximally safe for the known environment. The *planner's* task is now complete and the MPFS is passed down to the *navigator* so that an actual path can be generated within the MPFS.

The process of selecting a MPFS need not be a completely computer operation. It would be advisable for the pilot and observer to have an input and be able to select the MPFS for all the obstacle that encroach into the PFS.

Figure 3.4: Modified Preferred Flight Space: a) T=1, b) T=0.9, c) T=0.8, d) T=0.5



3.3 Navigator

The *planner* has produced a MPFS which prioritised the volume of space that the helicopter should be confined to during the inspection process to remain safe given the a known environment. The *navigator's* task is now to produce a flight path for the helicopter to fly along to produce the best inspection results.

The measure of "best inspection path" is subjective, being based on the configuration of the inspection cameras used and possibly the operator's own preference. However, to produce a flight path some form of formalisation is required. It is suggested that the experience of pilots and observers who fly the manned inspection operations could be used as a model for the exact flight path configuration. This model could then be used to calculate the best distance and angle between the camera and power line. The best inspection path in not necessarily the safest path. There is then a trade-off between the best inspection path and the traversability values of the MPFS.

Manned inspection flights start at one end of the lines of poles and then move forward along the lines. The observer studies the "front" of the first pole as the helicopter is flying towards it together with the power lines, and once passed will then move onto the "front" of the next pole. During the transit to the second pole, the observer may look back to view the "rear" of the first pole to complete the inspection, before finishing with the "front" of the second pole.The process of looking back means that the helicopter can in general fly a reasonably straight path alongside the poles at a constant speed.

Once the *navigator* has produced a flight path the vehicle is ready to perform the inspection of the poles. In a perfect flight, ie. no obstacles are detected, the vehicle would just follow the *navigator's* path. However, once up in the air it is the *pilot's* responsibility to detect obstacles and avoid them.

3.4 Pilot

The *pilot* is the "real-time" element of the controller. The *pilot's* role is to follow the path generated by the *navigator* and to detect obstacles and replan the path to avoid them. The *pilot* function has been broken down into two main areas - obstacle detection and location, and path planning. This section will look in turn at these two functions.

3.4.1 Obstacle Detection and Location

The detection of obstacles in the environment is critical to the safe operation of the helicopter. Once an obstacle has been detected, its location in the environment is required so that a local map can be built up allowing a path planner to produce a flight path that avoids the obstacles.

This section will not give implementation details, but will give a high level view of how the controller at the *pilot* level would operate in a complete system. Chapter 4 will give a more detailed treatment of the obstacle detection and location methods that have been investigated and their implementation on the test rig. As stated previously in chapter 2 two types of detection mechanisms would be required for safe operation of the helicopter. The first; standard detection - would handle stationary or slow moving obstacles that do not pose an immediate threat to the helicopter. The second - reflex detection - would cater for obstacles that pose an immediate threat to the helicopter, such as fast moving jets.

Chapter 2 presented a discussion detailing why vision was chosen over other methods such as RADAR, and in this chapter the justification for choosing vision will be reinforced.

The proposed solution to the detection of obstacles is to use a number of video cameras mounted on the periphery of the helicopter. The optics would be chosen so that a continuous field of view in azimuth is achieved giving 360° vision around the vehicle. The field of view in elevation would typically be 50°. Any obstacles detected that are with a certain range of the helicopter would be entered into a

local map. A full discussion of the camera system is given in section 4.4.

Standard Detection

The aim of detection is to find areas in a two dimensional image which could be obstacles in the environment that may pose a danger to the vehicle. This would be a continuous process operation on a stream of video images. There would be a separate process for each stream of images, and a supervisor process to keep the process synchronised. The result would be areas in the cameras' image that have the characteristics of obstacles. The next step is to estimate the distance from the helicopter to the obstacles, and then determining their actual location in space.

The method chosen to estimate the distance from the helicopter to the obstacle is the optical flow method. This method uses a sequence of images and estimates the amount of motion in the sequence. Considering a fixed camera in a dynamic environment, only obstacles that move would be detected and very little information about stationary obstacles would be gathered. However, our helicopter is moving and this can be used to acquire more information about a static environment, provided the magnitude and direction of motion between acquired images is known. This process is known as motion stereo vision. The ability to deal with moving obstacles accurately depends on the precision of the helicopter position measurements. The result from this process would be an optical flow field, which shows the amount of motion in an image sequence. The larger the flow field the closer the obstacle is to the camera, and therefore the more dangerous it is.

Once candidate obstacles have been detected they need to be analysed. The information needed about obstacles is their location in space and their velocity. The distance away from the camera is estimated using the optical flow field and information about the camera calibration. The resulting distance can then be used in a simple mathematical relationship linking the position in space to the position in the image frame, provided the helicopter position is known. This results in an obstacle region which needs to inserted into the map the *pilot* uses to replan its path.

The initial acquisition of an obstacle only gives information about what the cameras can see - there is no information about what could be behind the obstacle. Using the overriding guideline of safety first the unseen region behind the obstacle must be marked as a "danger area". Once more information about the obstacle is know a better description of its shape can be progressively entered into the map.

Once a map, containing obstacles, is produced the next step is to use it to test if the *navigator's* flight path needs modifying.

Figure 3.5 shows an example of a the pilot's local map, the black regions contain obstacle regions and the white contains freespace. The red regions are marked as obstacle as the vision system has been unable to determine the full shape of the obstacle. The size of the map would be dependent on the vision system, there is a trade off between the maximum detection distance and the location accuracy. A guide would be that the resolution of the pilot's map should be no worse that the accuracy of the vehicle's positioning system. The maximum size of the map is dependent on the accuracy of the location system, the closer an obstacle is to the helicopter the more the apparent motion is between successive image frames which would improve accuracy. At greater distances the accuracy will deteriorate because of the reduction in apparent motion. At the limit, if the apparent motion falls below on pixel on the camera's CCD no motion can be detected and thus no position can be estimated.

Reflex Avoidance

The second mode of obstacle detection is the reflex avoidance system which is included to handle obstacles posing an immediate threat to the helicopter. The requirement is to detect a *looming* motion. An obstacle approaching the vehicle will appear progressively larger in the field of view, the faster it is coming towards the helicopter the larger the looming field. It is assumed that there is insufficient time to estimate distances, and that only the approximate direction of approach can be determined. There would also not be enough time to perform path planning, so some rapid response mechanism is needed to move the helicopter out of danger safely. To accomplish this a series of acceptable "escape" routes would be

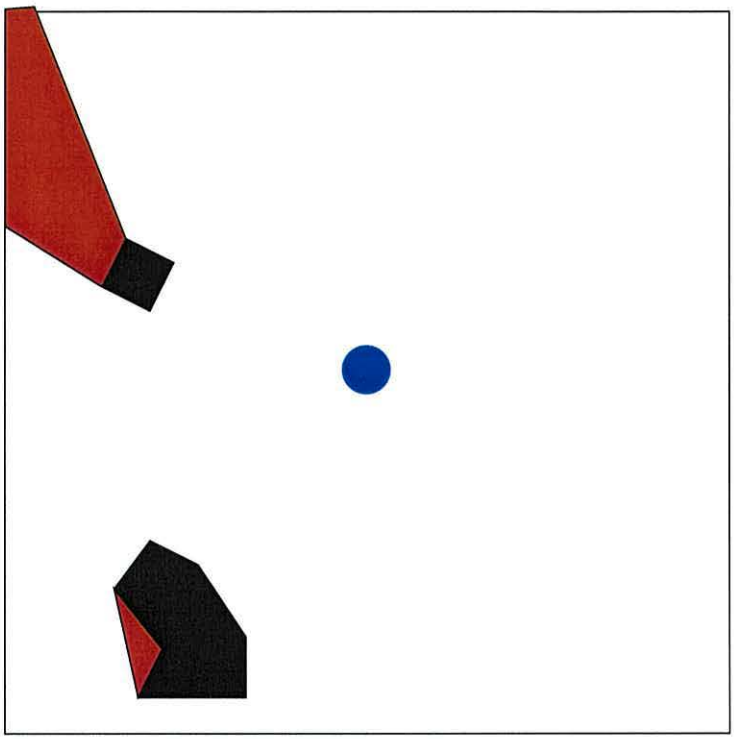


Figure 3.5: A visualisation of the pilot's map

planned as the helicopter proceeds along its flight path, so that at any location the vehicle can put one path into action dependent on the direction of approach of the threat. The method used to generate these paths is again based on the standard motion planner using the distance transform and the local map. A number of safe goal locations are selected in different directions from the helicopter's present location and then used to plan the escape route. Chapter 4 will describe the looming concept in more detail.

3.4.2 Motion Planning

If an unknown obstacle is detected in the vicinity of the helicopter then on-line path planning may be required to avoid it. The *pilot* is a "real-time" system which needs to produce safe paths very quickly when an obstacle is discovered. This places severe demands on the computational algorithms used for this purpose and

one of the main contributions of this thesis, path planning using the Distance Transform detailed in chapter 5, is to show how processing efficiency can be increased and the memory requirements minimised.

The map at the *navigator's* level contains information about the whole inspection run. At the *pilot* level, the data contained in the *pilot's* map only covers a subsection of the *navigator's* map. The map can be visualised as a cube with the helicopter at the centre. The size of the map is dependent on the dynamics of the helicopter and the dynamics of possible moving obstacles that could be encountered, and finally on the specifications of the visual system. The map will include known obstacles that would be passed down from the *navigator* level, which in turn was built based on the configuration of the landscape generated via the PFS and the MPFS at the *planner* stage.

In normal circumstances, when no unknown obstacles have been discovered, there would be no need for any path replanning. However, by including data from the higher levels and the motion of the helicopter, the map is always up to date should replanning be required.

This provision of a constantly updating map allows the generation of reflex avoidance escape routes throughout the mission. This process would be given priority over the standard path planning mechanisms as explained further in Chapter 5.

If standard replanning is required the *pilot* would initially select a goal point somewhere on the predefined *navigator's* flight path. Using this location and the *pilot's* map a first attempt would be made to generate a path. If this fails, more goal locations would be tried until a path is found. If no path is produced the *pilot* is then free to decide on any location as the goal. If, after trying different goal locations, a safe path is not found then the helicopter would slow down and the *navigator* and then the *planner* and finally the user would be asked for further instructions. It should be noted that even when the helicopter cannot follow the *navigator's* path, the reflex avoidance mode remains operational. The likelihood of the *pilot* failing to find any paths is very low so the referrals to the higher levels will be few and far between.

Once a path is generated it is passed down to the autopilot level which converts

the flight path into stick control commands. Refer back to page 57 for the diagram of the whole inspection system or figure 3.1 for the controller visualisation.

3.5 Summary

This chapter's purpose was to present a controller architecture which combines all the different technologies that would be required to produce a viable helicopter inspection platform. The controller architecture was chosen as it allowed the division of tasks into off-line, the *planner* and *navigator*, and the on-line, the *pilot*.

The *planner's* task is to define the poles that need inspecting and to use this to produce first a preferred flight space (PFS) and then using information about the environment to modify this as safely as possible to produce a modified preferred flight space (MPFS), which is a sub-set of the PFS.

The *navigator* used this MPFS to produce a flight path that is as safe as possible while giving the "best" inspection locations.

The *pilot* level has its hands on the controls and needs to be able to detect and locate obstacles and to then calculate a path modification that will avoid them. The subsequent chapters of this thesis introduce the technology needed to implement the *pilot* level.

Chapter 4

Obstacle Detection and Location using Machine Vision

The previous chapter gives details about the controller hierarchy which has been selected as the framework for producing a method for controlling an autonomous air vehicle. This chapter and chapter 5 concentrate on methods for solving two key issues regarding air safety. This chapter discusses methods for detecting and locating obstacles that may pose a threat to the vehicle. This chapter will show the work done in trying to answer the "seeing" aspect of the CAA's "see and be seen" regulations.

The two main areas of investigation are:

1. Obstacle detection;
    - Standard detection;
    - Reflex or panic detection.
2. Obstacle location.

The detection process will be discussed giving details about the operation of the two modes, standard and reflex. The problems associated with each will be given and applicable methods found in the literature will be reviewed as to their

suitability for our application. Reflex detection will be explained and the reasons for needing it will be discussed. The detection of looming motion using different methods will be explained.

The location of objects in an environment is key to satisfactory path planning. The methods considered will be expanded giving details about their suitability. Methods such as optical flow and stereo vision will be presented with an analysis of their performance and suitability.

The pilot function described in chapter 3 has two interconnected systems, a path re-planner which is used to avoid slow moving or stationary obstacles and a panic planner, or reflex planner, which is used when a fast moving or undetected obstacle poses an immediate risk to the safety of the vehicle. This is shown in figure 4.1.

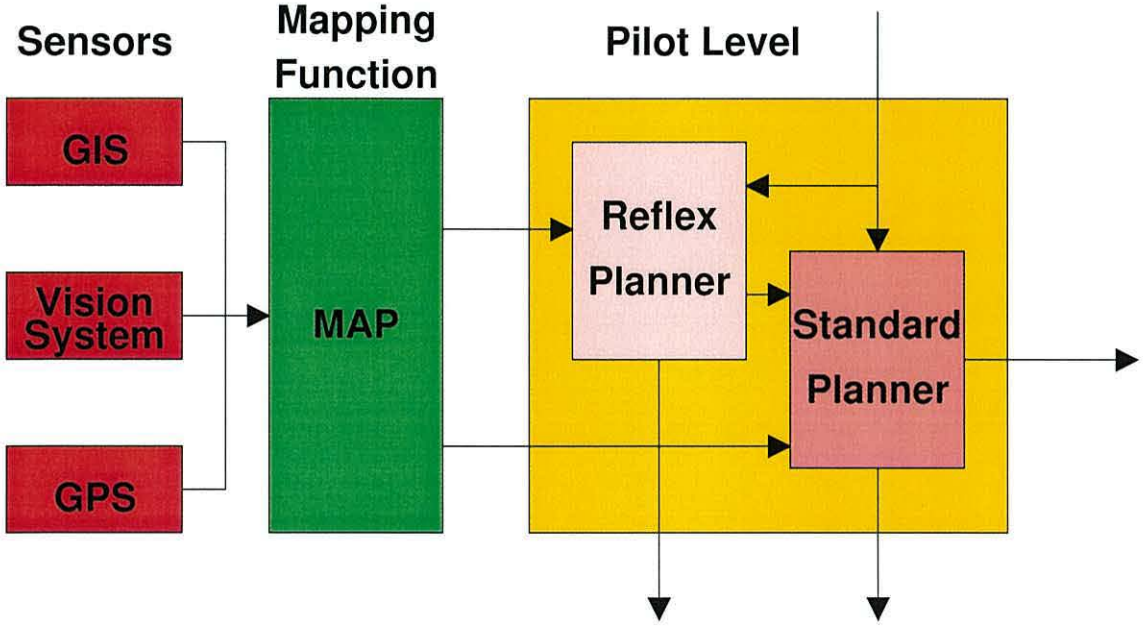


Figure 4.1: Pilot function showing the two types of avoidance systems

There are two types of obstacles:

- Slow moving or stationary objects (the norm);
- Fast moving objects (the exception).

The first type of obstacle can be avoided using the standard types of path planning techniques highlighted in chapter 2. These type of obstacles include balloons, hang gliders, parachutists and tall ground based structures such as radio towers.

The problem comes with fast moving obstacles as these may not be detected and analysed in time for the path planner to take avoiding action, this is where reflex motion planning is needed. The obvious obstacle in this group is a low flying fast jet. Although one may think that encountering such a threat would be a rare occurrence, training flights do take place in remote areas of this country, the very place that airborne inspection is most needed.

4.1 Reflex Collision Avoidance

This section will outline the principle of reflex collision avoidance and highlight the reasons for considering these methods for use in the hierarchical controller configuration.

Reflex motion planning means replying quickly to a sudden stimulus. It is analogous to the reflex action in humans, a knock on the knee results in a subconscious movement of the lower leg. It is suggested that vision will be used as the input to the panic planner but other sensory inputs have not been ruled out. Joarder and Raviv [47] outline a method for autonomous obstacle avoidance using looming. A description of looming as it is understood in nature is given in a paper by Regan and Vincent [48]. Murray et al. [47] show how looming can be integrated with other machine vision techniques on their Yorick camera platform.

4.1.1 Methods of Detection

Looming

The visual looming effect, a function related to the expansion of the projection of an object in the retina, is a prime cue in human reflex in avoiding obstacles

[49]. The same process can be used for machine vision using a video camera to give a measure of the location of approaching obstacles.

The looming value L of an infinitesimally small 3D object is defined as the negative value of the time derivative of the relative distance R between the observer and the centre of the object divided by the relative distance. The unit of looming is $time^{-1}$. It has been shown [50] that looming is independent of camera rotation and it can be conveniently measured using a logarithmic retina. Looming is related to, but different from, the so called Flow Field Divergence and Time-to-Collision.

Time-to-Collision

Time to collision is another method that could be used as a reflex avoidance system. Detection of a small time-to-collision triggers a defensive panic movement [47]. In [47] the looming function is used to protect the cameras should any object approach them, the time to collision is derived from the optical flow field by finding the point of zero motion in the flow field. When the scene is purely translational this is the focus of expansion, the point at which the scene translation vector, and thus the looming object, strikes the image plane.

As a simple demonstration of the time-to-collision principle, consider a planar version of the pinhole camera model, and a vertical bar perpendicular to the optical axis, travelling towards the camera with constant velocity. It is possible to compute the time, τ, taken by the bar to reach the camera only from image information; that is without knowing the real size of the bar or its velocity in 3-D space [51].

Figure 4.2 shows the bar approaching the camera, L denotes the real size of the bar, with V its constant velocity, and f is the focal length of the camera. The origin of the reference frame is the projection centre.

If $D(0) = D_0$ is the position of the bar on the optical axis at time $t = 0$, its position at time t will be $D = D_0 - Vt$. Then,

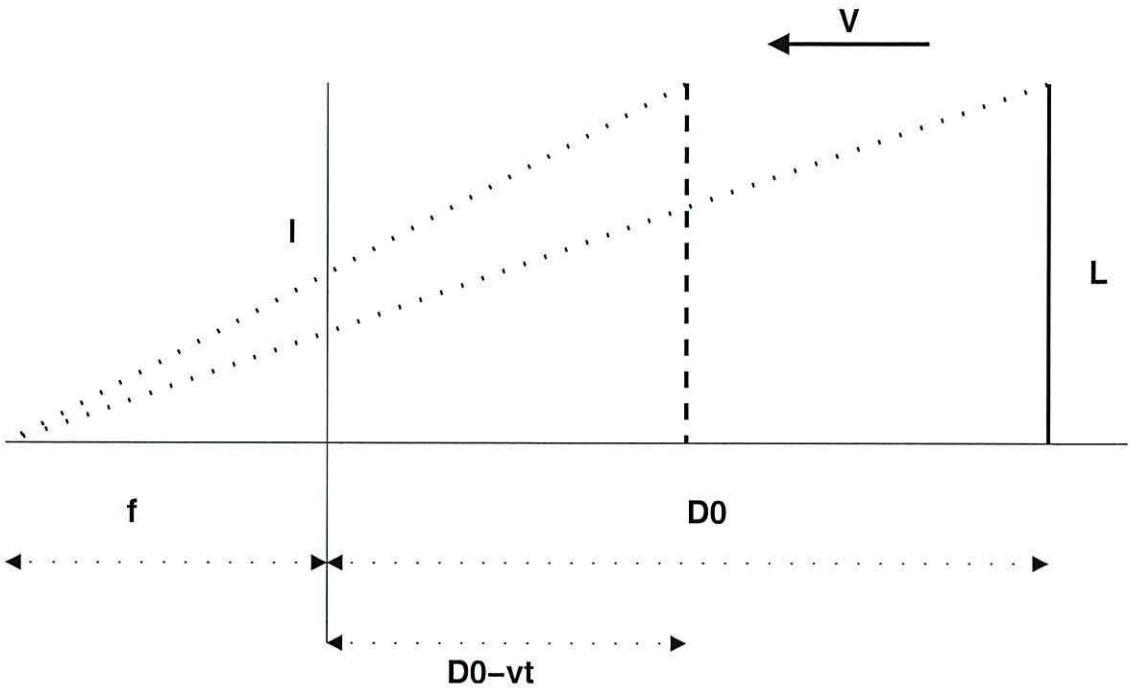


Figure 4.2: Time to impact

$$\tau = \frac{V}{D} \tag{4.1}$$

From figure 4.2 the apparent size of the bar on the image plane at time t is given by

$$l(t) = f\frac{L}{D}$$

If we compute the time derivative of $l(t)$,

$$l'(t) = \frac{dl(t)}{dt} = -f\frac{L}{D^2}\frac{dD}{dt} = f\frac{LV}{D^2}$$

now if we take the ratio between $l(t)$ and $l'(t)$ then

$$\frac{l(t)}{l'(t)} = \tau \tag{4.2}$$

Both the apparent size of the bar, $l(t)$, and its time derivative, $l'(t)$, are measured from the images which allows the computation of τ in the absence of **any** 3-D information.

4.1.2 Avoiding Action

Once a possible obstacle has been detected by either looming or time-to-contact how does the vehicle avoid a collision? It is relatively easy to detect a possible collision it requires little processing power and is robust against many variants such as camera rotation. This section will indicate some options that have been considered for avoiding a collision.

The easy answer to detecting a possible collision is to change the course so that it is perpendicular to the velocity vector of the obstacle, this would virtually guarantee that no collision with the obstacle occurs. The problem with this is that the new course may not be safe and in the urgency to avoid the moving object the helicopter could fly straight into a hill-side. What is needed is a flexible and dynamic avoidance plan which could be called upon at any point during the flight.

We have suggested previously in chapters 2 and 3 that a constantly updated avoidance plan that takes the current environment and the location of known obstacles into account should be built during the flight. If a hazard situation arises, such as a fast moving jet flying low and at speed in the general direction of the vehicle, the current avoidance plan is put into action. The plan does not have to be complex as long as it works and reduces the possibility of collision to a level acceptable to the CAA.

During the operation of the avoiding action new plans are generated so that at all points during the flight a valid and safe plan exists. The reflex planner is the last resort to avoid collision and does not guarantee avoiding every possible collision. For example an obstacle appearing from nowhere, out of a fog bank for instance, into the immediate vicinity of the helicopter may not be avoided. However, in all but the exceptional cases the reflex avoidance system must avoid collisions every time.

4.1.3 Conclusions

The aim is to produce a safe reliable system that performs correctly every time an obstacle, that is deemed to be a threat, comes within a set distance from the vehicle. The system must work alongside the other systems in the vehicle using information from them to best prepare for evasive action. It is the hope that the reflex motion planning system will never be needed and that the standard path planning system will cope with all environments configurations and obstacle positions and velocities.

Vision is our main sensory input to the system. Machine vision procedures must be implemented to detect obstacles that could be a threat. The level of threat must be measured so that if need be the reflex motion planning system is activated.

Other sensory inputs may be required to give the best protection. A recent article on the BBC Television programme "Tomorrow's World" showed a demonstration of a aircraft detection system using the characteristic signal produced by strobe navigation lights. The system operated in bright day light and was capable of detecting an obstacle which was invisible to the pilot as it was directly in line with the sun. Systems such as this would be an extra level of protection for our system, and may improve the case for the CAA granting a flying licence for the vehicle.

In conclusion, reflex motion planning is required because in certain, exceptional, circumstances the standard path planning system may not be able to cope with the obstacles in the environment either because of their speed or proximity. The reflex planner will maintain an up to date escape plan derived from information from other systems in the vehicle and will act on this plan when a sensor system detects a hazard situation. The reflex avoidance system has not been taken further than a paper study as effort was put in to develop the standard detection system first. It would be of little merit to avoid fast moving obstacles, only to collide with a stationary one. The next section will describe methods for detecting and locating the normal types of obstacles that the vehicle will encounter.

4.2 Standard Detection

The role of the machine vision system is obstacle detection and location. The objects detected are analysed and, if they are found to be either stationary or slow-moving and close to the helicopter, then they are added to the pilot's map. The pilot can then plan a path around the known and detected obstacles to get from start to goal. If the objects detected are found to be coming towards the helicopter at a measured speed greater than a certain threshold then a reflex avoidance system will be initiated and an escape path put into action based on the measured velocity.

This section will describe two methods that were investigated during the course of the project. The first uses the principle of conservation of distance, i.e. points on a rigid obstacle do not change their relative position. The second uses the optical flow method.

4.2.1 Conservation of Distance

Consider a video camera connected to a frame-grabber which acquires images for computer processing. The following assumptions are made about the scene and equipment:

- Obstacles are rigid,
- The image is changing slowly between successive frames,
- In a small interval of time the image brightness remains unchanged.

then, knowing the amount of camera motion and image plane motion the position of the obstacle can be found.

The following method is based on an idea given by Mitiche at al. [44]. They use the principle of conservation of distances in rigid obstacles to calculate position and displacement. This principle, which is the subject of a theorem in kinematics

of solids, simply states an obvious fact: distances in a rigid configuration of points do not change during motion. The parameters used in this method consist of the positions of image points and not optical flow.

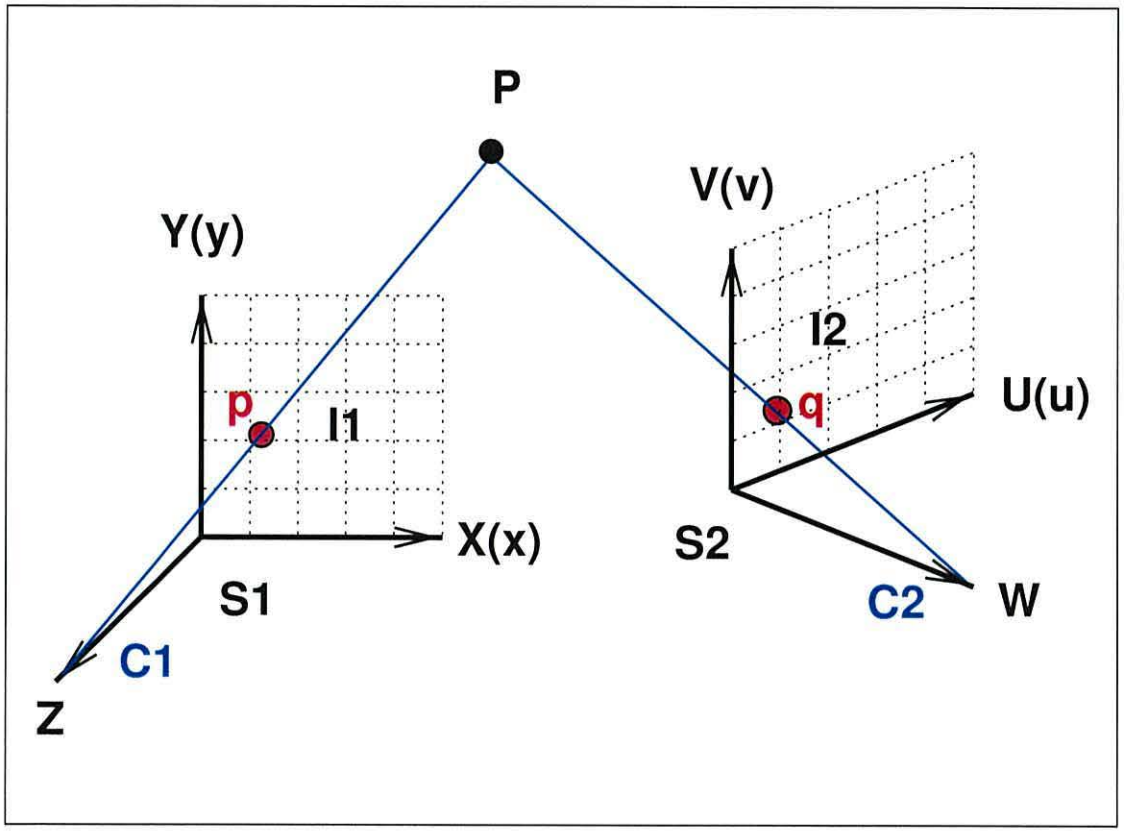


Figure 4.3: Viewing Configuration

Figure 4.3 shows a point being captured by two cameras. Both cameras are represented by a central projection model modified not to invert images. A point P_i in space, the coordinates of which are (X_i, Y_i, Z_i) in S_1 and (U_i, V_i, W_i) in S_2 is imaged on p_i in I_1 and q_i in I_2, where S_1 and S_2 are the coordinate systems of the camera at the different capture positions, and I_1 and I_2 are the image plane coordinate systems. Because P_i is on line $C_i p_i$ there exists a negative real number λ_i such that

$$X_i = \lambda_i x_i$$

$$Y_i = \lambda_i y_i$$

$$Z_i = (\lambda_i - 1) f_1$$

where (x_i, y_i) are the coordinates of p_i, and f_1 is the focal length. Similarly, P_i is on the line C_2q_i and if (u_i, v_i) are the coordinates of q_i then there exists $\gamma_i < 0$ such that

$$U_i = \gamma_i u_i$$

$$V_i = \gamma_i v_i$$

$$W_i = (\gamma_i - 1) f_2$$

λ and γ are negative because of the coordinate system used, C is the projection centre through which all projection lines pass through, this point is a distance f from the origin point.

The distance between points P_i and P_j expressed in S_1 is therefore

$$d_{ij}^{S_1} = (X_i - X_j)^2 + (Y_i - Y_j)^2 + (Z_i - Z_j)^2$$

or

$$d_{ij}^{S_1} = (\lambda_i x_i - \lambda_j x_j)^2 + (\lambda_i y_i - \lambda_j y_j)^2 + (\lambda_i - \lambda_j)^2 f_1^2$$

Similarly, the distance between P_i and P_j expressed in S_2 is

$$d_{ij}^{S_2} = (\gamma_i x_i - \gamma_j x_j)^2 + (\gamma_i y_i - \gamma_j y_j)^2 + (\gamma_i - \gamma_j)^2 f_2^2$$

Now the principle of conservation of distance, assuming identical units in S_1 and S_2, gives

$$d_{ij}^{S_1} = d_{ij}^{S_2}$$

or

$$(\lambda_i x_i - \lambda_j x_j)^2 + (\lambda_i y_i - \lambda_j y_j)^2 + (\lambda_i - \lambda_j)^2 f_1^2 = (\gamma_i x_i - \gamma_j x_j)^2 + (\gamma_i y_i - \gamma_j y_j)^2 + (\gamma_i - \gamma_j)^2 f_2^2 \tag{4.3}$$

Each point P_i contributes two unknowns λ_i and γ_i, and each pair of points P_i, P_j gives one second order equation in the form of (4.3) above. Therefore, 5 points yield 10 equations in 10 unknowns.

This method depends on point correspondence between two images. The assumption that the image is changing slowly over times allows the matching of image points between frames. In [44] they use a least-squares analysis using a FORTRAN subroutine. It should be noted that the position of the camera when the images were taken is not used in the equations given above, Mitiche discusses a method of recovering camera motion (both translation and rotation) using these equations by calculating the transformation matrix which takes S_1 to S_2. If the position of the camera is known then this matrix is known and this will simplify the solution to the second order equation given above. In the RIPL application the position of the vehicle would be known, thus this method could quickly determine the location of obstacles in space

This method was investigated but not taken any further due to the restrictions on rigid bodies. As described above, the slow moving obstacles that could be encountered include non-rigid bodies such as balloons. The next section will describe the method that was chosen to perform obstacle detection and location.

4.2.2 Optical Flow

Optical flow is the distribution of apparent velocities of movement of brightness patterns in an image [41]. It can give important information about the spatial arrangements of objects in an environment. Discontinuities within the optical flow field can aid segmentation of images into regions which correspond to different objects. The optical flow technique can also be used to estimate 3D scene properties and motion parameters from a moving visual sensor [52]. It can also be used to compute the focus of expansion and time-to-collision [53].

A detailed description of optical flow and methods for computing it are given in papers by Horn and Schunck, and Beauchemin and Barron [41, 54].

Mathematical Representation

Let the image brightness at the point (x, y) in the image plane at time t be denoted by $E(x, y, t)$. Now consider what happens when the pattern is displaced a distance δx in the x direction and δy in the y direction. The brightness of the patch is assumed to remain constant, so that

$$E(x, y, t) = E(x + \delta x, y + \delta y, t + \delta t)$$

Expanding the right-hand side about the point (x, y, t) we get,

$$E(x, y, t) = E(x, y, t) + \delta x \frac{\partial E}{\partial x} + \delta y \frac{\partial E}{\partial y} + \delta t \frac{\partial E}{\partial t} + \epsilon$$

Where ϵ contains second and higher order terms in δx, δy and δt. After subtracting $E(x, y, t)$ from both sides and dividing through by δt we have

$$\frac{\delta x}{\delta t}\frac{\partial E}{\partial x} + \frac{\delta y}{\delta t}\frac{\partial E}{\partial y} + \frac{\partial E}{\partial t} + \mathcal{O}(\delta t) = 0$$

where $\mathcal{O}(\delta t)$ is a term of order δt. In the limit as $\delta t \to 0$ this becomes

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

If we let

$$u = \frac{dx}{dt} \qquad \text{and} \qquad v = \frac{dy}{dt}$$

then is it easy to see that we have a single linear equation in the two unknowns u and v,

$$E_x u + E_y v + E_t = 0 \tag{4.4}$$

where we have also introduced the additional abbreviations E_x, E_y and E_t for the partial derivatives of the image brightness with respect to x, y and t, respectively.

The partial spatial derivatives of the image brightness are simply the components of the spatial image gradient, ∇E, and the temporal derivatives dx/dt and dy/dt, the components of the motion field, $\mathbf{v}$. Equation 4.4 can also be rewritten as the image brightness constancy equation:

$$(\nabla E)^\top \mathbf{v} + E_t = 0 \tag{4.5}$$

where $E = E(x, y, t)$ and $\mathbf{v}$ is the motion field and the subscript t denotes the partial differential with respect to time.

How well does (4.5) estimate the normal component of the motion field? To answer this we need to look at the difference, Δv, between the true value and the one estimated by the brightness constancy equation. In general, $|\Delta v|$, is unlikely to be identical to zero, and the apparent motion of the image brightness is almost always different from the motion field. The apparent motion is known as the optical flow.

The optical flow is a vector field subject to the image brightness constancy equation 4.5, and loosely defined as the apparent motion of the image brightness pattern [51]. The optical flow is only an approximation of the motion field which can be computed from time varying image sequences, the error of this approximation is small at points with high spatial gradient. The papers studied [55, 56, 48, 50, 49, 57, 58, 59, 60] give more information on the different methods for computing the optical flow.

There are other methods for computing object position and velocity, some use

object matching to a database of known obstacles and measure the relative sizes to estimate depth. Other use inter-frame matching of obstacle features, such as corner points, to estimate motion.

Optical Flow Methods

Barron et al. [61] and Galvin et al. [62] give comparisons of optical flow methods. The optical flow methods they have investigated include the following:

- Anandan
- Camus
- Fleet and Jepson
- Heeger
- Horn and Schunck
- Lucas and Kanade
- Nagel
- Proesmans et al.
- Singh
- Uras

The source code for the most of the above methods is available from:

- *ftp://ftp.csd.uwo.ca/pub/vision* [63]

The key requirement is to get a full picture of what is occurring in the environment. Many of the methods investigated by Barron et al. [61] and Galvin et al. [62] give very accurate results, but the measurement density is low. A measurement density of 100% means that every pixel in the original images has

been given an optical flow value. For example Barron et al. determine that the phase-based method of Fleet and Jepson and the first-order differential technique of Lucas and Kanade give the most accurate results. However, the measurement density of their output ranges from at best 76% to at worst 8.8%.

Some methods reported by Barron et al. always produced a measurement density of 100%. These are the methods of Anandan [64], Horn and Schunck [41] and Uras. There is a trade-off between accuracy of results and the density of the results.

In this application safety is of the paramount importance. This dictates that every obstacle must at least be detected. This requires the use of optical flow methods that produce a full set of results, even if these results are not as accurate as methods that do not give a full field of results. Anandan's method gave the best accuracy results of those methods that produced a full optical flow field.

4.2.3 Anandan's Method

Research on the interpretation of motion [52] indicates that a dense and reliable displacement field may be necessary for the successful determination of the structure of the environment. The most effective way for computing dense fields seems to be those methods that use either gradient-based or a matching approach in a hierarchical, multi-resolution scheme.

Anandan's method [64] uses such a hierarchical, multi-resolution scheme. The framework used to compute the optical flow uses spatial frequency decomposition to separate the intensity variations according to scale. Large scale (or low spatial-frequency) intensity variations can provide imprecise measurements over a large range of magnitude of motion. Small scale (or high spatial-frequency) variations can give more accurate measurements but over a smaller range. The matching-criterion used between the two images is the minimisation of Gaussian weighted sum-of-squared-differences (SSD) correlation template window. A confidence measure is used together with a smoothness constraint to specify the criterion for the propagation of reliable displacements within the framework.

The code used to implement the optical flow system was obtained from [63]. The code was reviewed and adapted to give the output in a form that could be integrated with the other software systems. Chapter 8 gives more results obtained during the testing and integration of the optical flow software. There are a number of parameters that can be varied in the software. The first is the size of the correlation template window (W). The supported values of W are 3×3, 5×5 and 7×7. The second value is the number of levels of spatial decomposition (L), the software supports up to 4 levels. The final parameter is the number of relaxation iterations (I), this has a default value of 10.

Figure 4.4 shows two of twenty images from a series of images showing a taxi turning. These images are used as standards to test optical flow systems. A fixed camera is pointing at a road junction, a number of vehicles are moving in the image. The optical flow shown in figure 4.5 was calculated using W=3,L=1 and I=10 as the parameters for the optical flow software.

Figure 4.4: Images 3 and 4 of the Taxi set of images used for testing optical flow systems

Figure 4.5 shows the optical flow field calculated by Anandan's method using the two images shown in figure 4.4. The white taxi seen in figure 4.4 is represented in this figure by a disturbance in the centre of the flow field. On further examination of the flow field (which may not be visible in the printed form), the direction of the motion of the taxi can be seen. A number of other features can be seen in the flow field, near the lower-left corner a disturbance due to the black car can be seen. Also, as important, where there is no motion the flow field is zero.

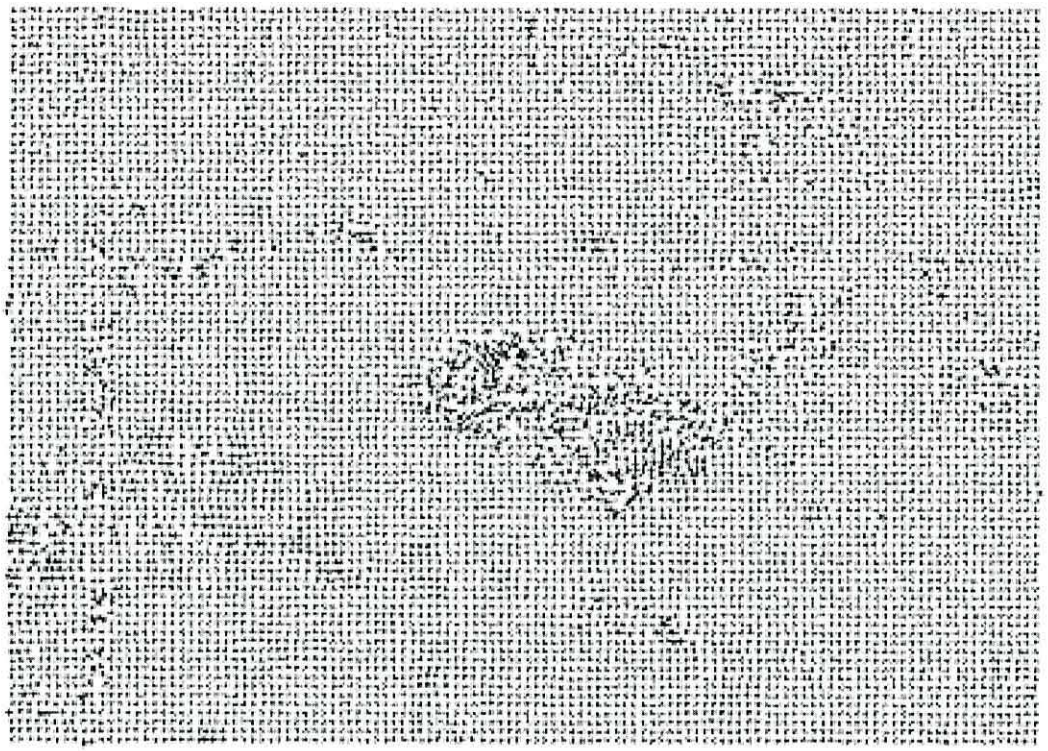


Figure 4.5: Flow field generated by Anandan's method using the images shown in figure 4.4

Anandan's method shows that a velocity field can be generated from just two images. The flow field is full so depth all over the image can be estimated. Chapter 8 gives details of the experiments used to configure the optical flow software. The next section shows how the flow field can be used to build a depth map.

4.3 Using optical flow to locate obstacles within the environment

The optical flow field gives information about the velocity of objects as they appear on the image plane of the camera. This motion is either due to the motion of the camera or the motion of the object or a combination of the two. This section will describe how the flow field can be used to estimate the position of an object in space. The coordinate system is centred in the middle of the

helicopter as shown in the figure 4.6.

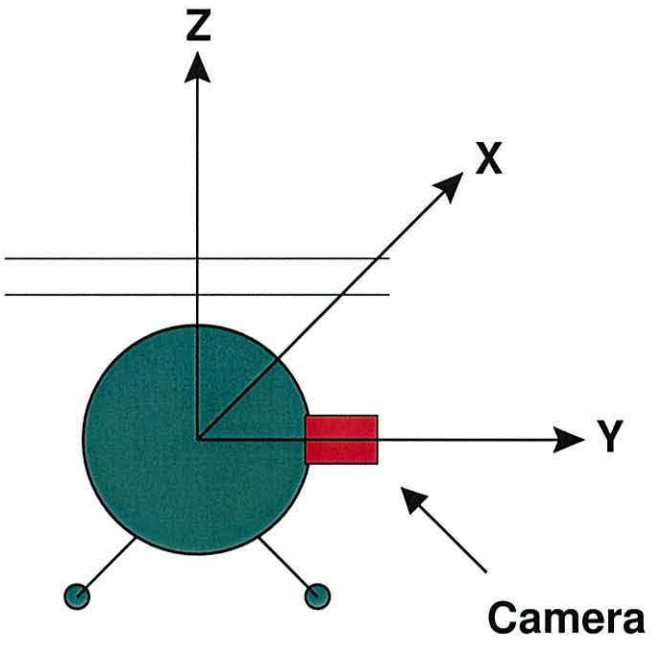


Figure 4.6: Coordinate System

The Y-axis is aligned with the focal axis of axis of the camera.

The magnitude and angle of each flow field vector are calculated. For stationary objects and a moving camera the distance from the camera to the obstacle can be estimated as the inverse of the magnitude of the flow field:

$$P_y \approx \frac{K}{1+f}$$

where K is a constant related to the configuration of the camera/lens and f is the magnitude of the flow field. This detection method is designed for slow moving obstacles, the amount of motion of the obstacles between images grabbed by the camera will be small compared to the distance moved by the helicopter so that obstacles can be considered to be stationary.

The next function is to estimate the position relative to the helicopter. This is achieved using the following:

$$
\begin{aligned}
i \times \alpha &= px \\
j \times \beta &= pz
\end{aligned}
$$

where $P_{x,z}$ is the estimated position is space, and i, j is the position of the flow vector in the flow field image and α and β are constants of the camera/lens combination. Chapter 8 describes this in more detail.

The final stage is to transform the camera based estimates of obstacle position into world based positions using the position of the helicopter. If the obstacle points are within the local environment they are entered into the pilot's map, if not they can be passed to the navigator's map.

If the map contains obstacles then the flight path that the navigator calculated may need changing, the next chapter will introduce a rapid path planner for use in three dimensional environments. The final section in this chapter gives details about the use of more than one camera to allow the detection of obstacles from all around the helicopter.

4.4 Camera Configuration

This section explains the decisions that were made about the configuration of the camera system that would be required for the helicopter. The vision system must be able to "see" as much of the environment as possible. There are two ways to achieve this using video cameras.

Yagi et al. [42] present a method using a conic mirror which gives a field of view of 360° in azimuth. The field of view in elevation is dependent on the specification of the mirror. Although this method give the full field of view in azimuth, there are still blind-spots.

4.4.1 Multiple cameras

The other way of giving a 360° field of view is to have multiple cameras with overlapping fields of view mounted on the circumference of the vehicle. Figure 4.7 shows a vehicle with four cameras from above.

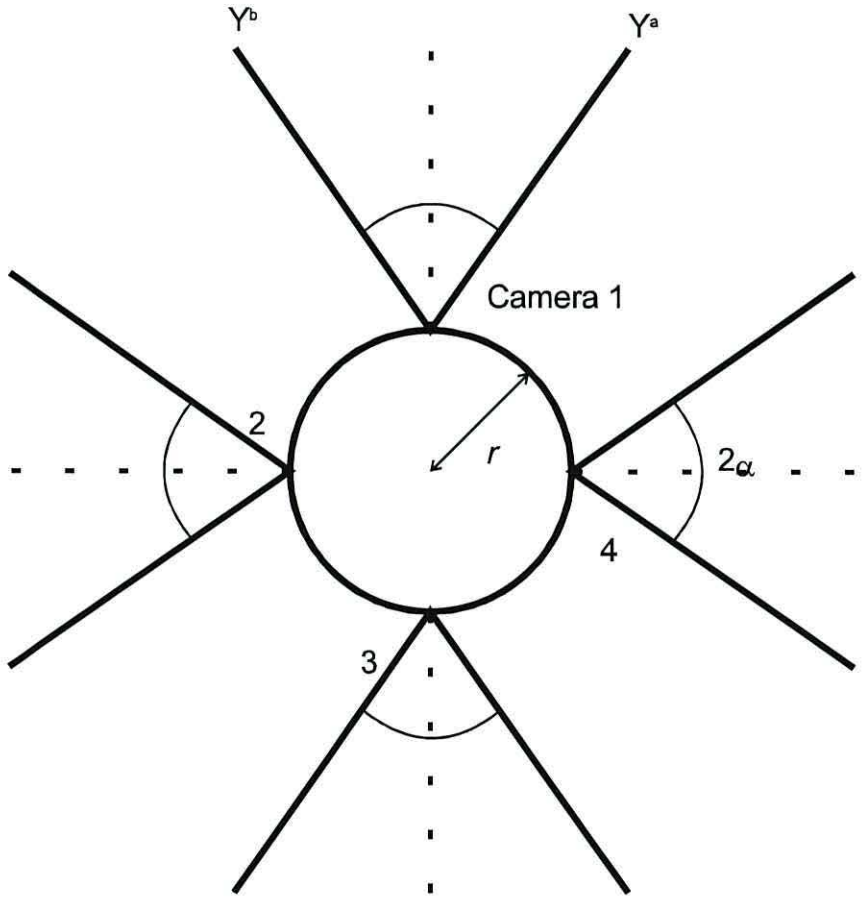


Figure 4.7: Field of view

Given that the origin of the coordinate system is placed at the centre of the circle, the equations of the field of view boundaries are given below. Equation 4.6 relates to camera 1, equation 4.7 relates to camera 2 on so on.

$$\left.\begin{aligned} y_a &= \tan(90-\alpha)x_a + r \\ y_b &= \tan(90-\alpha)x_b + r \end{aligned}\right\} y > r \tag{4.6}$$

$$\left.\begin{aligned} y_a &= \tan(\alpha)x_a - r\tan(\alpha) \\ y_b &= \tan(-\alpha)x_b + r\tan(\alpha) \end{aligned}\right\} x > r \tag{4.7}$$

$$\left.\begin{aligned} y_a &= \tan(-90-\alpha)x_a - r \\ y_b &= \tan(-90+\alpha)x_b - r \end{aligned}\right\} y < -r \tag{4.8}$$

$$\left.\begin{aligned} y_a &= \tan(\alpha)x_a + r\tan(\alpha) \\ y_b &= \tan(-\alpha)x_b - r\tan(\alpha) \end{aligned}\right\} x < -r \tag{4.9}$$

It can be seen that if α is less than 45° then the fields of view will never overlap. As α increases the more the fields overlap and the closer the boundary cross over points get to the vehicle. The trade off between number of cameras and field of view is determined by the space available on the vehicle.

Using a 6mm lens mounted on a $\frac{1}{2}$" CCD camera, both of which are commercially available for reasonable prices, gives a field of view of about 44° in elevation and 60° in azimuth. Six camera-lens combinations would be required to give a full field of view in azimuth.

There is no practical way to give 360° coverage in both azimuth and elevation. It should be noted that human pilots do not even come close to having 360° of field of view in azimuth in most small aircraft as there are always blind-spots caused by the vehicle.

4.4.2 Implications of using multiple cameras

The use of multiple cameras with overlapping fields of view presents some interesting questions about how to deal with the information. Either each camera should be treated individually or the images from all the cameras should be combined into one vista. The first case can be wasteful as two adjacent images will contain some identical information which will be operated on twice. The combination of images is difficult and requires more processing and could be hazardous should one of the cameras move or otherwise go out of working tolerance.

The aim of the machine vision system is to identify and locate obstacles in the environment so that they can be entered into a map. Therefore it is not a problem if the same obstacle is detected twice by different cameras as long as the positions in space determined from the two images are approximately the same. Therefore we favour using the information from the cameras individually. The machine vision system would then be several nearly independent software processes all running in parallel on separate image feeds. This process could either run on one computer with a multi-processing operating system, or on several different computers in a parallel processing environment. The second option is favourable, subject to space limitations, as it improves the safety of the whole system.

4.5 Summary

This chapter has described the different types of machine vision systems that would be required in the helicopter. Standard and reflex detection have been introduced and methods for implementing them have been described. The use of multiple cameras to give the maximum field of view possible has been discussed.

The reflex avoidance system needs to act quickly. Using methods such as looming and time-to-collision provide enough information to allow a panic motion to be put into action. However, these methods have not been take any further than a paper study and evaluation.

The standard detection system needs to detect stationary or slow moving obstacles and locate them in space. A number of methods exist to achieve this but they need to be quick. The conservation of distance method detailed above would have been used except for the fact that it only works when the obstacle is rigid. Such a restriction is not feasible in the airspace the helicopter would operate in, non-rigid obstacles such as balloons could be present. This method uses geometry to estimate position in space, and given that the position of the helicopter would be known should be a quick and reliable method.

The optical flow method described here depends on the obstacles being slow moving or stationary. The motion in the image plane is then only due to the

camera, thus the closer the obstacle is to the camera the larger that motion is. Therefore, image plane motion is inversely proportional to the distance from the camera. From this estimate of distance location in space can be determined, which allows any obstacles detected to be avoided. Anandan's method for calculating optical flow has been chosen to be used in subsequent work as it provides a full flow field and is accurate.

Once obstacles have been detected they may need avoiding. Chapter 2 describes some of the path planning methods that have been detailed in the literature. The next chapter describes a three dimensional path planner which uses the distance transform and space decomposition to effect rapid planning.

Chapter 5

Path Planning using the Distance Transform

This chapter discusses path planning using the distance transform [8]. A discussion of different methods for path planning is given in Chapter 2. The distance transform method gives two key advantages. Firstly, using a quadtree/octree decomposition reduces the complexity of the planning problem. Secondly, the distance transform allows for fast path planning in a similar manner to the potential method, but without the inherent disadvantage of generating false goals.

5.1 Basic Representation

This section will outline the principles of planning a path using the distance transform using a simple two-dimensional example. Figure 5.1 presents the workspace that will be used throughout the example.

Figure 5.2 shows the segmentation of the workspace into an eight-by-eight matrix structure where the cells coloured black represent obstacles in the workspace and the white coloured cells represent free space.

Figure 5.3 shows the distance transform values radiating from the goal, marked

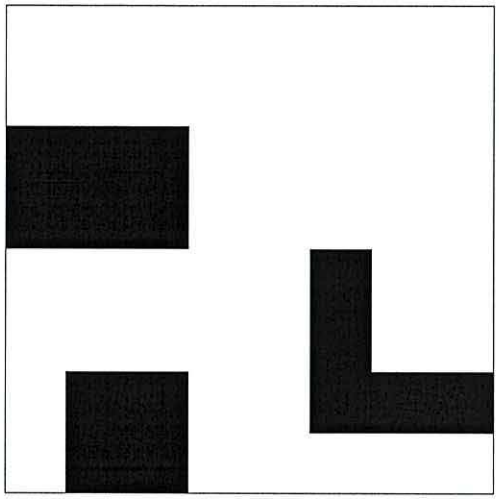


Figure 5.1: Example two-dimensional workspace

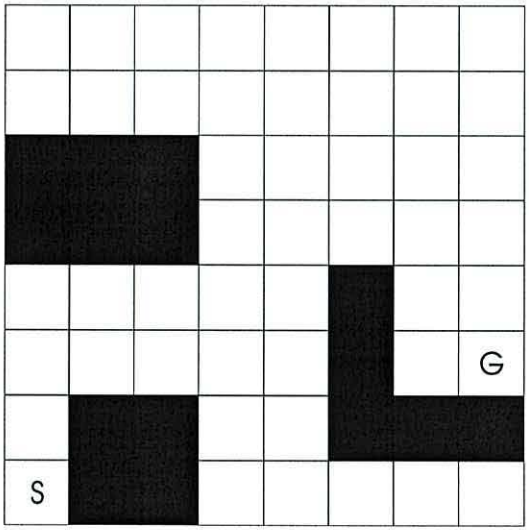


Figure 5.2: Segmentation of workspace into matrix structure

G, throughout the workspace.

In this example the four-connected principle is used. This defines that a cell only considers another cell to be a neighbour if it shares common edges, rather than just a common vertex connection. Figure 5.4a shows the four connected method, while 5.4b shows the eight or fully connected method.

Figure 5.4d shows the drawback of using eight-connected neighbours as the resulting path can contain elements that clip the corner of obstacles. Figure 5.4c shows how this can be avoided by using four-connected neighbours.

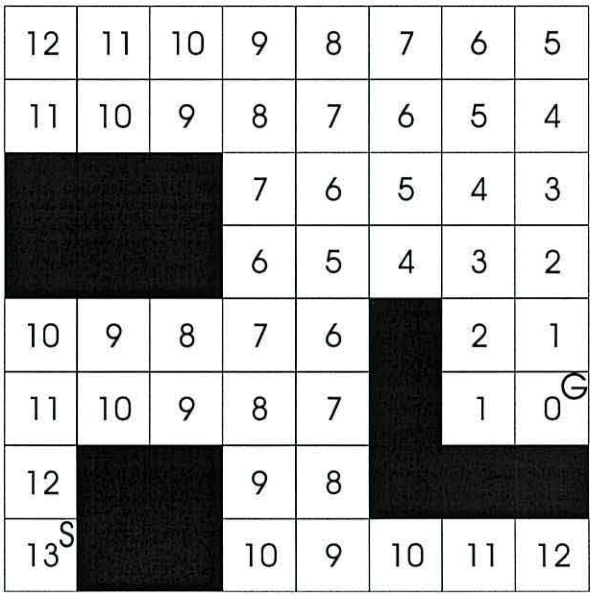


Figure 5.3: Application of the distance transform

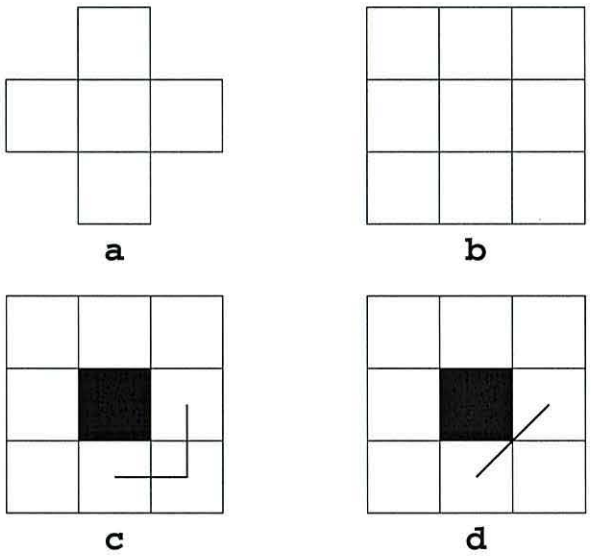


Figure 5.4: Neighbour configurations and their resulting path properties

Once the matrix structure has been labelled with the distance transform values a simple descent search can be used to find a path from the start to the goal. Figure 5.5 shows one possible path from the start location, marked S, to the goal location.

Although this method is simple it is very powerful and has several key advantages which make it an attractive solution to our path planning requirements. One of

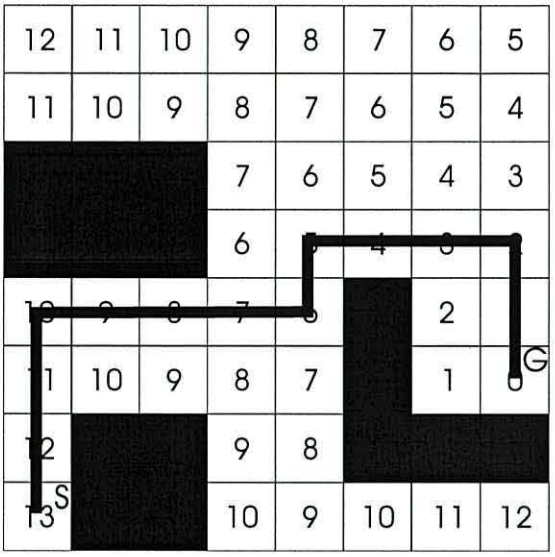


Figure 5.5: One possible path from the start to the goal

the principle advantages is the property that if a cell has not been labelled with a distance transform value then it is known that a path to the goal does not exist from that cell. Other search based methods would require time to determine that no path is possible, whereas in our application it is preferable to select another goal and apply the distance transform once more.

5.2 Quadtrees and Octrees

The previous section described the use of the distance transform in the simple matrix case. While this is adequate for small scale problems in two dimensions, for larger size workspace or planning in more than two dimensions the efficiency suffers due to the increased storage requirement and processing time. This section describes the decomposition of the workspace into a tree structure and shows how this improves the efficiency of the method.

5.2.1 What are quadtrees?

In section 5.1 a matrix was used to represent the occupancy of a workspace. However, this representation does not store the information efficiently. In this section the concept of quadtrees is introduced and, by reference to the Quadtree Complexity Theorem [33] it is shown that except for pathological cases, the representation of a workspace using a quadtree provides an improvement in both storage requirements and processing time over a matrix based representation.

Quadtrees are one example of hierarchical data structures. Hierarchical data structures are used to represent data in computer graphics, computer-aided design, robotics, computer vision and cartography [33]. The term *quadtree* is used to describe a class of hierarchical data structures whose common property is that they are all based on the principle of recursive decomposition of space. They can be differentiated on the following basis:

1. the type of data that they represent,
2. the principle guiding the decomposition process, and
3. the resolution (which may be variable or fixed).

In our application, a map is produced showing the location of obstacles and free space. The map is binary, i.e. there are only two possible values of the data, one value for obstacles and another for freespace. Regular decomposition is used in our example, this means that each decomposition results in equal sized parts. The resolution of the decomposition is variable.

The properties of the map can be used to accomplish the decomposition. We are interested in specific regions of the map, i.e. those containing obstacles and those containing free space and therefore make use of the most common type of quadtree, the *region quadtree.*

5.2.2 Node labelling scheme

The key to the subsequent planning operations is producing a fully connected tree structure. To aid in this process we need a method of knowing which nodes at any level are neighbours to other nodes. The way that this is achieved is by using a labelling scheme which uniquely marks a node giving both its position in the hierarchy of the tree and the position of the region within the map, so that the neighbourhood connection process can ascertain what neighbours a given node should have and assign them.

The numbering scheme operates as follows:

1. The root node is always labelled 0
2. On splitting the root node its children are labelled 1,2,3, and 4
3. Then, if node 1 is split, it is partitioned into nodes 11, 12, 13, and 14.
4. Node 11 is further split into nodes 111,112,113,114 and so on.

5.2.3 Building a quadtree

At this point the description of a *region quadtree* is best conducted using a series of figures. For the current example we will use the map in figure 5.1.

If this map contained one region only, that is if it were all black or all white, then no decomposition based on region would be necessary and only one node would be required to store all the information present in the workspace. However, if the map contains a mixture of obstacle and free space then the workspace is divided into four equal sized sub-workspaces as shown in figure 5.6.

This is not the only division possible; for instance it is possible to divide the workspace into four equal sized triangles by dividing the workspace along its diagonals. There are other shapes that can be used which have been discussed in section 2.4.5, but here, and in all subsequent references to quadtree or octree decomposition, square quadrants will be used.

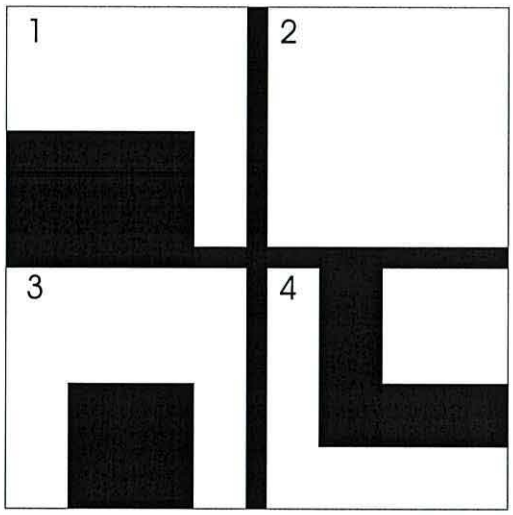


Figure 5.6: Workspace after first division

The next step is to perform the same occupancy test on the new sub-workspaces. If a mixture of black and white occurs then it is successively sub-divided until no further division is possible. Division stops either when no further partitioning is required or when the maximum resolution of the map is reached. Figures 5.7 and 5.8 show the results of the next and last division.

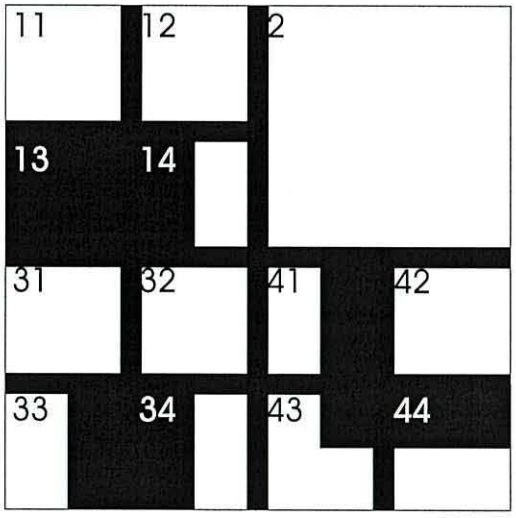


Figure 5.7: Intermediate decomposition of the data presented in figure 5.2

The resulting tree structure produced by this method is given in figure 5.9. Here the black leaf nodes represent an obstacle, the white represent free space and the grey nodes represent a region containing both obstacle and free space. Grey nodes are never leaf nodes: they must always be divided further.

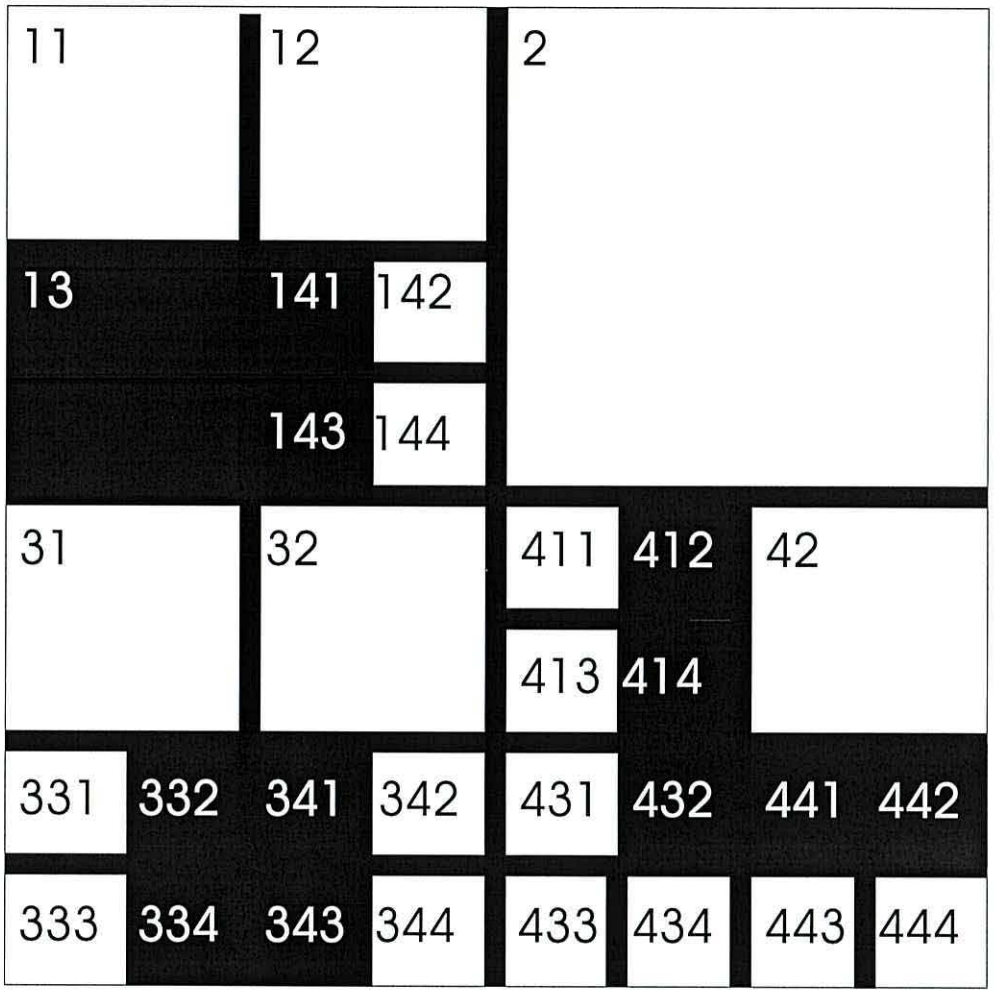


Figure 5.8: Final decomposition of the data presented in figure 5.2

The process can be described as a recursive algorithm which starts with the current quadrant set to the whole workspace:

```
Function generate tree
{
   Scan the current quadrant
   If quadrant contains both free and obstacle cells
   {
      Mark node as 'grey'
      Divide quadrant into sub-quadrants
      Label each new sub-quadrant as a child node
      For each sub-quadrant
```

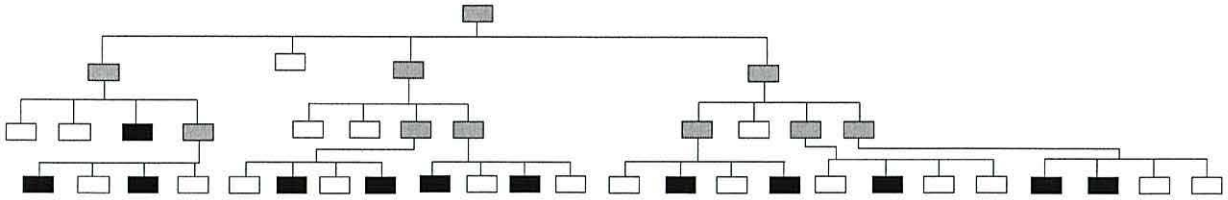


Figure 5.9: Quadtree resulting from the decomposition of the data presented in figure 5.2

```
        {
            Call generate tree
        }
    }
    Else
    {
        If quadrant is all free
        {
            Mark node as 'white'
        }
        Else
        {
            Mark node as 'black'
        }
    }
    Return
}
```

In this example the region information present in the map has been represented as forty-one nodes. At no point was the resolution of the map mentioned. To accurately describe the map a minimum resolution of 8×8 is required. However, this figure could be considerably greater for more complex maps.

The *region quadtree* used in this example partitions space into sets of squares whose sides are all a power of two long. This formulation is due to Klinger [65].

For path planning the neighbour relationships of leaf nodes are vital but these

have been lost during the translation from the matrix to the quadtree.

5.3 Tree connection method - the key to planning

The previous section describes how a tree structure can be derived from a map. However, for use in path planning a further step is required to enable the distance transform to be applied in a sensible manner. The *region quadtree* has the advantage of reducing storage requirements and increasing processing efficiency, but the decomposition removes the direct neighbourhood relationships of the regions. For example in figure 5.8 the leaf node marked 344 is a neighbour to leaf node 433. However, there is no direct way of determining this.

The tree structure must be transformed into an undirected connectivity graph containing information about the adjacency of freespace nodes before the distance transform can be applied. Establishing the connectivity requires three stages: initial neighbour assignments followed by two further node-linking passes through the structure.

5.3.1 Stage 1 : Initial neighbour assignments

Construction of the connectivity graph starts by forming links only between nodes that share a common parent. The children of every node in the quadtree (Figure 5.9) are examined in turn and a link made between all non-black nodes whose corresponding cells (see Figures 5.6 to 5.8) share a common boundary. The result is a number of intra-child-group connections as shown in Figure 5.10. Valid connections are stored as a pointer-driven list of Neighbour Information within each node's data structure; links to black nodes never appear in the list.

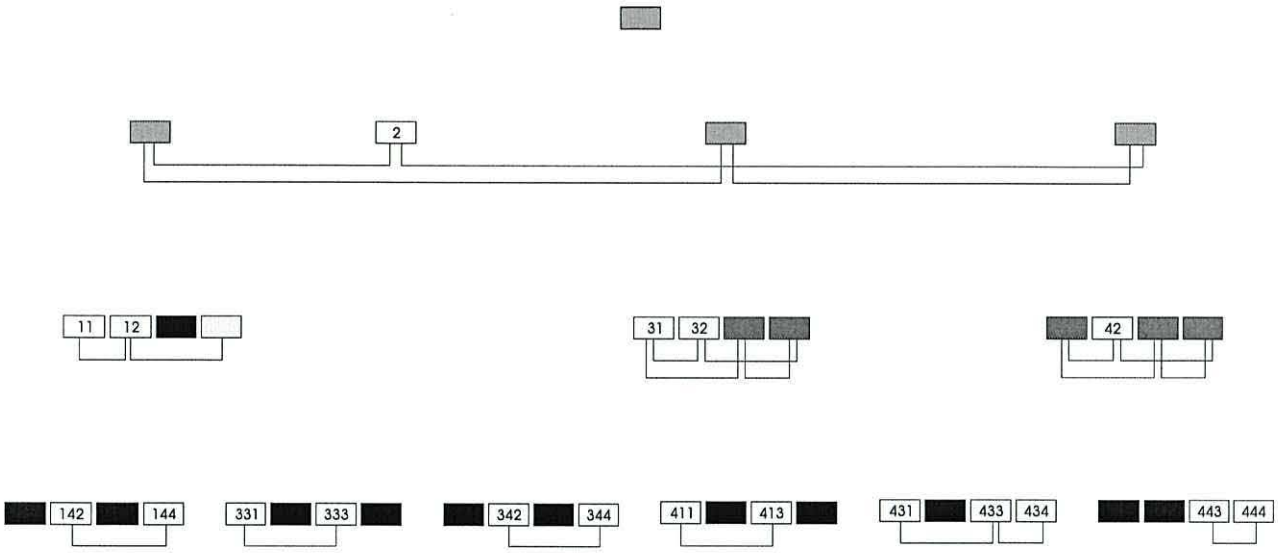


Figure 5.10: Connections after the initial connection stage

5.3.2 Stage 2 : Update connections

Two further types of connection are established at Stage 2. First, connections are made between the white children of a grey node and any neighbouring white node. An example of this in Figure 5.7 is the connection of cell 12 (a child of grey node 1) to white node 2. Secondly, all valid connections between nodes that lie at the same depth but do not share a common parent are added to the Neighbour Information list. An example of this in Figure 5.8 is the connection of cell 342 (a child of grey node 34) with cell 431 (a child of its neighbour grey node 43).

Traversal of the tree is accomplished in depth-first fashion by the recursive function *diver*, beginning at the root node, and applying the connection function *update1* when the conditions stated above are encountered.

```
Function diver(current_node)
{
   For n = 1:8
   {
      temp_node = nth child of the current node
      If temp_node is a grey node
      {
```

```
            diver(temp_node)
        }
        update1(temp_node)  /* call to the connection function */
    }
    Return
}
```

The *update1* function is called repeatedly during Stage 2 and the key to fast execution is the node labelling scheme which selects only those pairs of nodes which correspond to spatially adjacent cells to be tested for a freespace-to-freespace connection.

The adjacency relationships used by the connection function *update1* are described for the quadtree case - the rules for an octree are no different in principle but are considerably longer. Consider Figure 5.11 which shows the decomposition of four cells (1, 2, 3 and 4) into 16 sub-cells, whose corresponding nodes lie at the same depth in the quadtree.

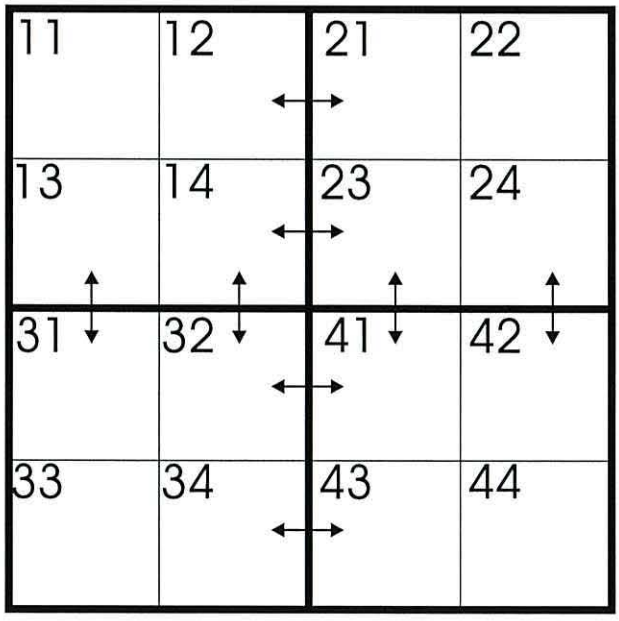


Figure 5.11: Adjacency Relationship

Evidently there are only 16 permutations of sub-cells (12-21, 21-12, 14-23, 23-14 .....etc) which share a common boundary with sub-cells of another parent, i.e. they are adjacent sub-cells. The node labelling scheme ensures that *update1*

Truth Table A

C/N	1	2	3	4
1		X2-Y X4-Y	X3-Y X4-Y	
2	X1-Y X3-Y			X2-Y X4-Y
3	X1-Y X2-Y			X2-Y X4-Y
4		X1-Y X2-Y	X1-Y X3-Y	

Truth Table B

C/N	1	2	3	4
1		X2-Y X4-Y	X3-Y X4-Y	
2	X1-Y X3-Y			X2-Y X4-Y
3	X1-Y X2-Y			X2-Y X4-Y
4		X1-Y X2-Y	X1-Y X3-Y	

Table 5.1: Truth tables used to determine adjacency relationships in the connection function *update1*

applies the tests for a freespace-freespace connection to adjacent cells only. In fact, as shown in truth tables 5.1A and 5.1B, any invocation of *update1* applies two tests at most. The truth tables are generic and applicable at any depth within the tree.

The tables 5.1A and 5.1B give the linking rules based on the final digits of the current node (C) and the neighbour node (N). For example if the current node is 34 (X=34) and the neighbour node is 43 (Y=43) then C=4 and N=3. Using table 5.1B, X2 (child 2 of the current node, in this case 342) would be linked to Y1 (child 1 of the neighbour node, in this case 431) and X4 (344) would be linked to Y3 (433).

```
Function update1(current_node)
{
   X = current node's label
   C = final digit of X
   For Each node in the current node's Neighbour Information List
   {
      Y = neighbour node's label
      N = final digit of Y
      If current node is grey AND neighbour node is white
```

```
      {
         Apply truth-table 1A to link white or grey children
         of the current node to the neighbour node.
      }
      Else If both current node AND neighbour node are grey
      {
         Apply truth-table 1B to link the white or grey children
         of the current node to the white or grey children of the
         neighbour node.
      }
      Else    /* current node is white AND neighbour node is grey
                 OR both current node AND neighbour node are white */
      {
         Do nothing /* the first case is dealt with in update2;
                       the second case does not occur because
                       such links will already have been made */
      }
   }
   Return
}
```

The dotted lines in Figure 5.12 show the connections for the example workspace after applying the *update1* algorithm.

5.3.3 Stage 3 : Update connections

Stage 3 connects nodes at different depths in the tree. The links that were made during the two previous stages are retained and the function *diver* is applied once more to traverse the tree but with the *update1* function replaced by the *update2* function, which again relies on the node labelling scheme for fast execution.

```
Function update2(current_node)
{
   X = current node's label
   C = final digit of X
   P = Parent node of X
   If current node has children
   {
     Switch (C)
     {
       Case 1:
        X2 linked P2 ; X3 linked P3 ; X4 linked P2 and P3
       Case 2:
        X1 linked P1 ; X3 linked P1 and P4 ; X4 linked P4
       Case 3:
        X1 linked P1 ; X2 linked P1 and P4 ; X4 linked P4
       Case 4:
        X1 linked P2 and P3 ; X2 linked P2 ;X3 linked P3
     }
   }
 Return
}
```

The dashed lines in Figure 5.12 show the Stage 3 connections for the example workspace. Note that Figure 5.12 now contains only the links between freespace nodes which are needed for path planning; the others are redundant once the connectivity graph is complete and are omitted. Whilst it would be possible to search this graph directly for a path, a much more efficient guided search can be performed using distance transform annotation.

5.4 Application of the distance transform

The previous sections have shown how a matrix based workspace can be converted into a quadtree and the resulting structure connected so that contiguous regions

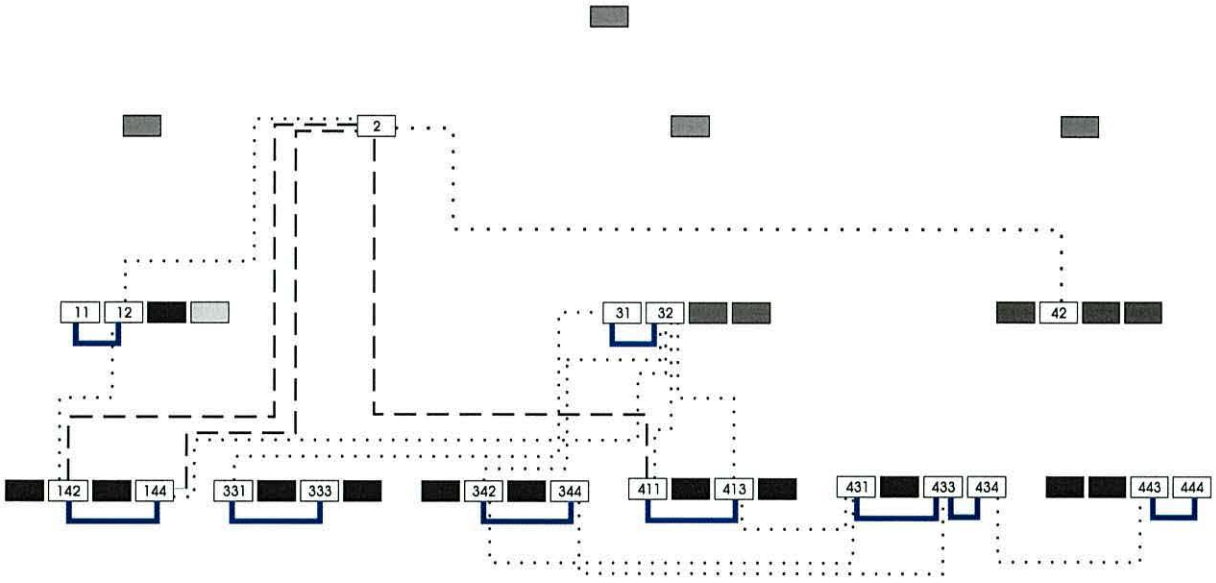


Figure 5.12: Fully connected tree after stages 2 and 3

are joined. The final step before planning a path is to apply the distance transform onto this structure.

In section 5.1 it was shown how the distance transform was applied to a matrix structure. The resulting distance transform for the example workspace was given in figure 5.2. The application of the distance transform onto the tree structure will produce a different result. The matrix structure is a uniform division of the workspace but the quadtree decomposition produces a non-uniform division. This can be seen by comparing figures 5.2 and 5.8. For example the node labelled 2 in 5.14 contains eight matrix cells in figure 5.2. In the matrix representation each cell is labelled with a distance transform value while, in the quadtree representation, every leaf node is given a distance transform value.

The distance transform is applied to the tree structure in the following way. Starting at the goal leaf node, mark it with a distance transform value of zero. The goal node contains a list of its neighbours. Each of these is labelled with a distance transform value of one, their neighbours are marked with a distance transform value of two and so on until all nodes that are accessible from the goal nodes have been labelled. Nodes that can not be reached from the goal node remain unmarked and show that from that node there is no feasible path to the goal. The distance transform applied to the example workspace of figure 5.13 is

shown as a grid representation in figure 5.14. In this figure the distance transform has radiated from the goal node through the structure until all the nodes are labelled. The above process can be described by the following algorithm:

```
Function distance transform
{
   Create empty queue
   Add goal node to queue
   Mark the goal node with a DT value of zero
   While the queue is occupied
   {
      Remove head of queue and make it the current node

      Copy all unlabelled neighbours of the current node
         onto the tail of the queue

      Mark all unlabelled neighbours of the current node
         with a DT value of one greater than the current node
   }
   Return
}
```

The final step in the path planning process is to find the path from start to goal.

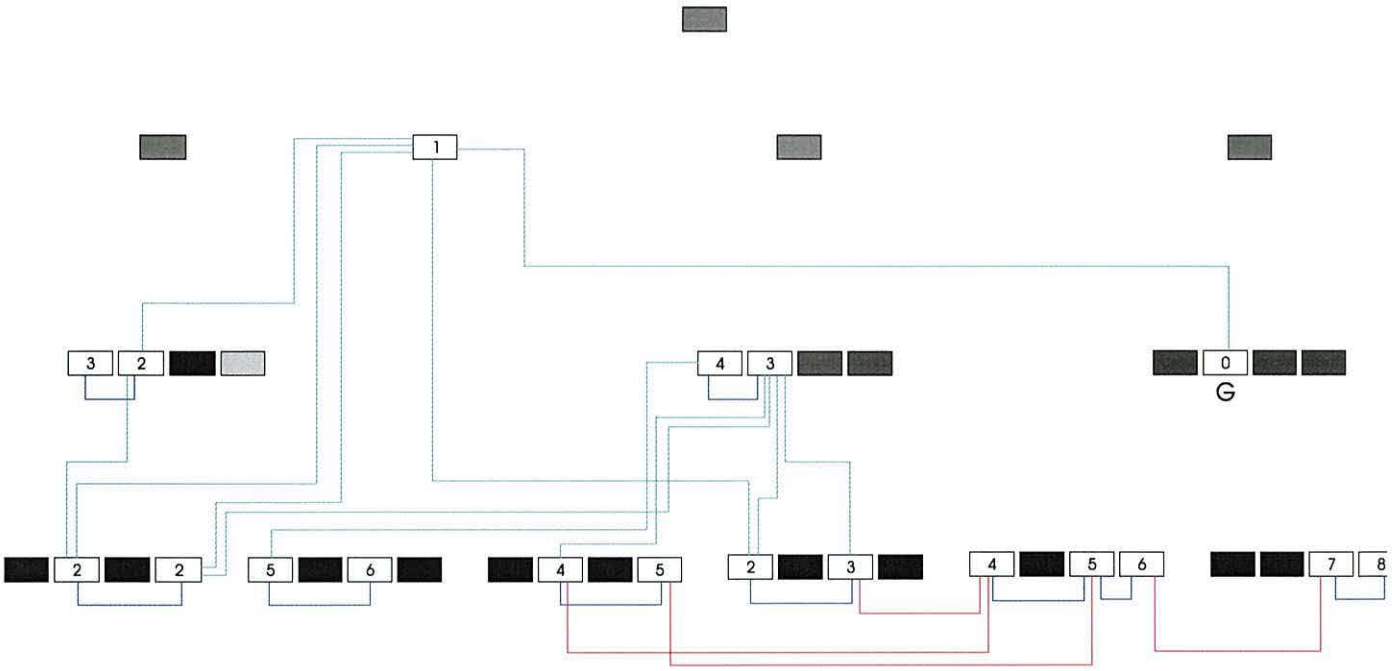


Figure 5.13: Distance transform applied to the example workspace

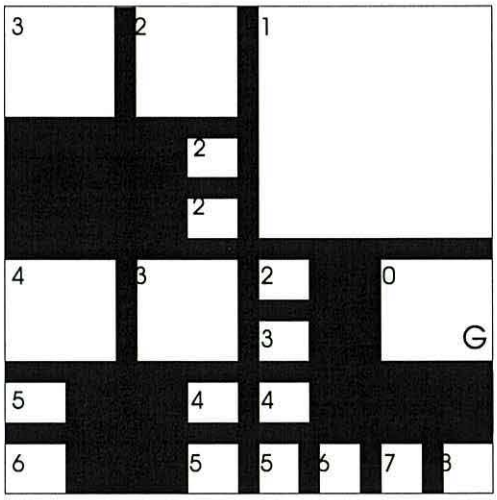


Figure 5.14: Distance transform applied to the example workspace

5.5 Path planning

The previous processes that have been applied to the local map have generated a structure that is ready for path planning. The path is produced by beginning at the start node finding its neighbour with the lowest distance transform value and making it the current node. The neighbours of this node are then analysed to find the one with the lowest distance transform value, which is then made the current node. This process repeats until the goal node is reached. If a start node is labelled then a path is **always** possible.

Figure 5.15 shows one path that could be constructed from the start node to the goal node. Figure 5.16 shows the same path in a grid representation of the quadtree.

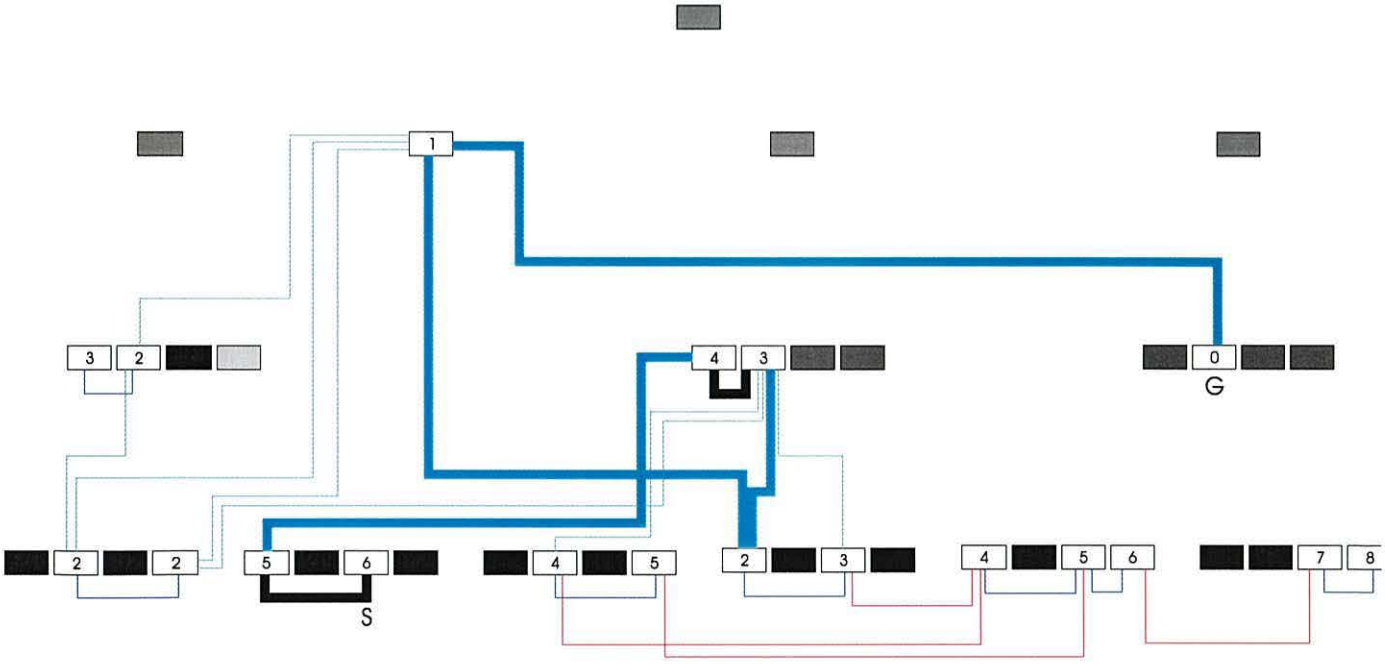


Figure 5.15: A path produced using the example workspace

If two or more neighbours of a node are found to have the same distance transform value, which is the best to choose? With this simple method of path planning it does not really matter. However, to improve safety it is always preferable to use nodes that cover the largest areas of free space. In this way the subsequent motion planner has the space to work in. If the nodes are the same size then the first one found is used.

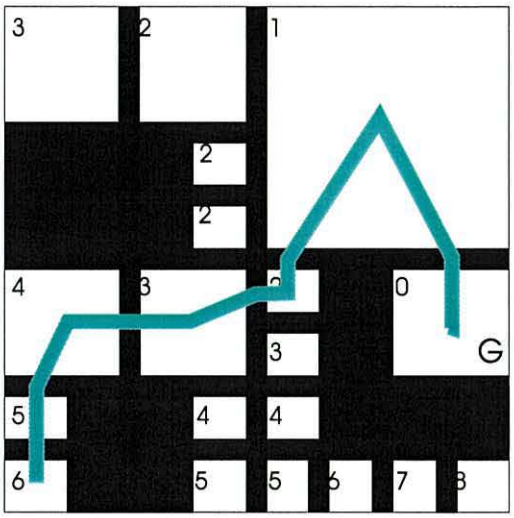


Figure 5.16: A path produced using the example workspace

The result from the path planning process is a number of nodes that have to be passed through to get from the start to the goal. Remember, that this is **not** the path that the vehicle would follow, but an area (or volume) that is safe for the vehicle to pass through. The path shown in figure 5.16 is constructed by connecting the node centres together, to get from the node path to an actual path requires robot specific motion planning based on the kinematics of the vehicle used. The path presented in figure 5.16 is only for illustration only.

5.6 Summary

This chapter introduces and describes a rapid method for path planning in three dimensions using the distance transform and spatial decomposition. The use of quadtree and octree decomposition has been explained and the concept of the distance transform has been introduced. The mechanics of node generation and neighbour connections has been shown by using a two-dimensional example.

The next chapter presents the results of the different experiments that were performed to test the three dimensional path planner.

Chapter 6

Path planning implementation and results

This chapter will review the results obtained from the different practical experiments that have been performed on the various implementations of the distance transform path planner.

6.1 Two dimensional planning

The section will give a brief review of the results from the two dimensional path planners that were developed to test the operation of the distance transform.

6.1.1 Matrix based implementation

The initial investigation of the distance transform used a standard matrix representation. This simple approach was used to firstly provide a straightforward way to investigate the properties and also to highlight any deficiencies in using the distance transform for path planning, and also to provide a reference for comparing any subsequent developments.

In order to keep the implementation as simple as possible, MatLab was used to model the path planning process. MatLab allows, through the execution of a series of function via a script file (called a m-file), a complex program to be developed. MatLab is particularly good at handling matrices, so it was the obvious first choice.

The *main* function sets up the required variables and matrices. The map is produced using a graphics package which results in a binary GIF file. This allows for different maps to be produced quickly and easily using a graphics application. There is the provision to include multiple goals but this is not used in the standard version.

The second function performs the distance transform calculation. This function scans the matrix until all the elements are labelled with a distance transform value. It performs this task by passing a 3×3 mask over the image. At the start only the goal location is marked with a distance transform value, in this case 1. When the centre of the mask reaches this goal location the elements on the periphery of the mask are set to a distance transform of one more than the value under the centre of the mask in this case 2. The mask is then applied to the map repeatedly until all the elements have been set. There are special cases of the mask to handle the corners and edges of the map. This has the effect of propagating the distance transform through the matrix.

This process of of passing a mask over the matrix takes a maximum of:

$$Pass_{max} = \left[\left(m \times Ceil\left(\frac{n}{2}\right)\right) + Ceil\left(\frac{n-1}{2}\right)\right] \tag{6.1}$$

loops of the algorithm to pass over the matrix. Equation 6.1 gives the worst case value for a workspace of size $n \times m$.

The criteria for stopping the generation of the distance transform is that all the elements have been set to a distance transform of 1 or more. However, if a region is not connected to the region with the goal in it, its distance transform values can never be set. To detect when to finish the application of the distance transform a "hole detection" algorithm is used. This marks all areas that cannot be reached

from a goal location. This method was used to allow the provision for multiple goals.

The holes are found by scanning the distance transform matrix after a maximum of $Pass_{max}$ loops of the above masking process have been performed. Any freespace areas that have not been marked with a distance transform value are more than likely to be contained in a hole. Each remaining freespace element contiguous to an obstacle is analysed by producing a vector that passes through the point vertically and horizontally. These vectors are searched for the sequence $\ldots, -1, 0, \ldots, 0, -1, \ldots$. If this sequence is found in both vectors it means that the point is enclosed.

The path planning function is a simple descent based search of the distance transform values from the start location to the goal location. If the distance transform value is set then a path is always possible, if it is not then no path is possible.

Figure 6.1 shows a simple example workspace. The goal location is marked with a red $\times$ and the start with a green $\times$, the path is shown as a series of $*$.

The MatLab implementation allowed many different configurations to be generated and tested easily. The algorithm was mouse based, with the start and goal nodes being entered on the display. However, the generation of the distance transform in two dimensions was slow, and would have been considerably worse if the algorithm had been extended into three dimensions.

However, it was a useful tool to investigate the distance transform and showed how the distance transform can be likened to the potential-method for path planning. This can be seen by looking at the surface plot of the distance transform given in in figure 6.2.

This surface plot shows the values of the distance transform. The goal locations is at the lower right hand side of the figure. It can be seen that as one moves away from the goal the distance transform values increase. The plot looks like a potential function with the goal at the bottom of the well. However, in the distance transform approach the obstacles do not have any influence over the generated

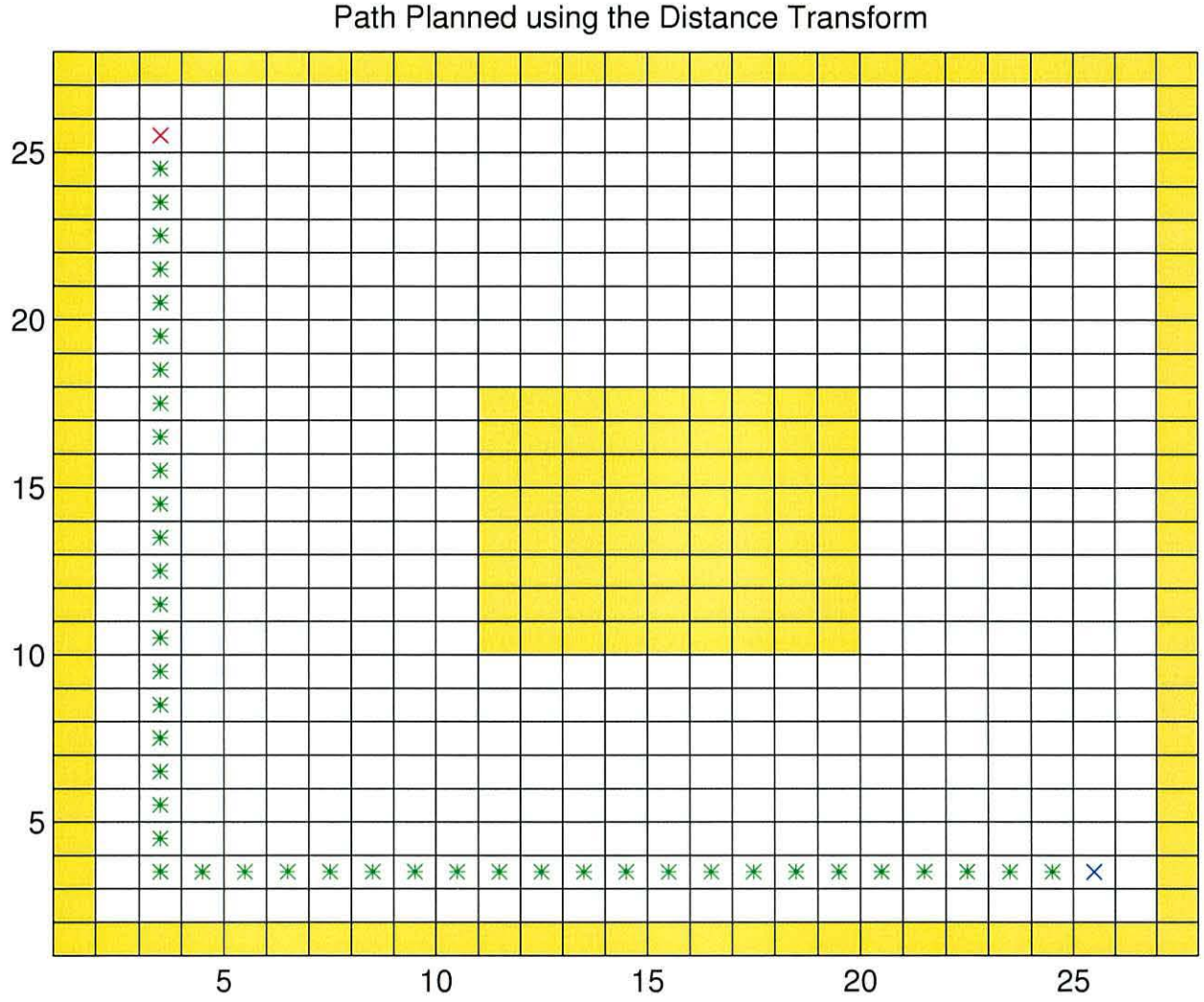


Figure 6.1: The path resulting from a simple workspace

surface. The potential method uses the obstacles to create a repulsive force which prevents obstacles from coming too close to them. This has advantages but, for some obstacle configurations, can produce local minima in the force field which will result in the path planner becoming trapped in a well other than that of the goal. The distance transform approach does not introduce these false minima. Without these false minima a simple descent based search can be used to plan the path, whereas a much more complex search method is required in the potential method.

A more complex map is given in figure 6.3. This shows a complicated environment together with an obstacle with holes. The figure shows how the holes are marked

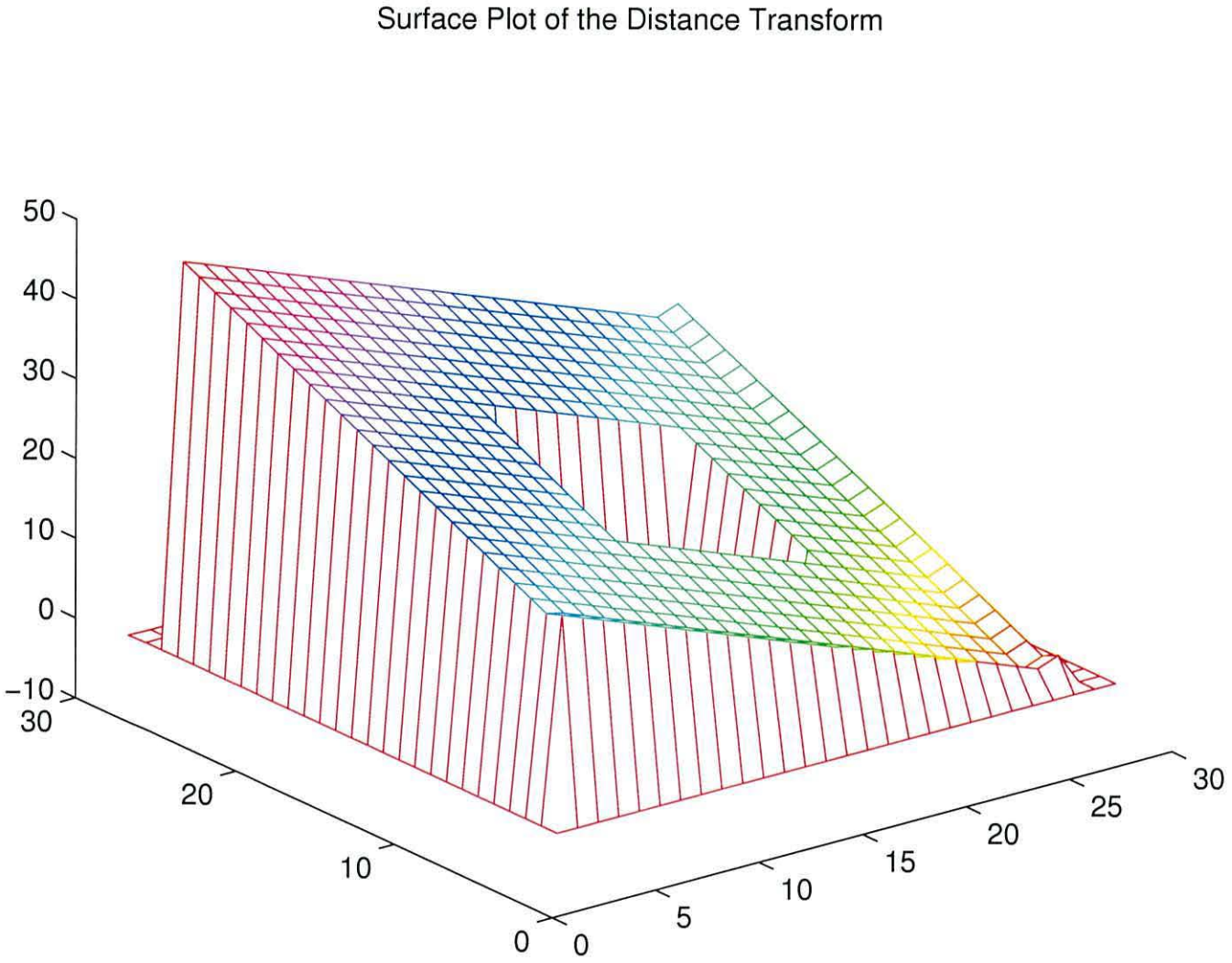


Figure 6.2: Distance transform surface produced from the example in figure 6.1

and also shows one path from the start location to the goal.

The surface plot of the complex example's distance transform is given in figure 6.4.

These investigations showed that a more advanced algorithm would be needed to allow for planning in "real time". The next section gives a description of the implementation of a quadtree based two dimensional distance transform planning algorithm.

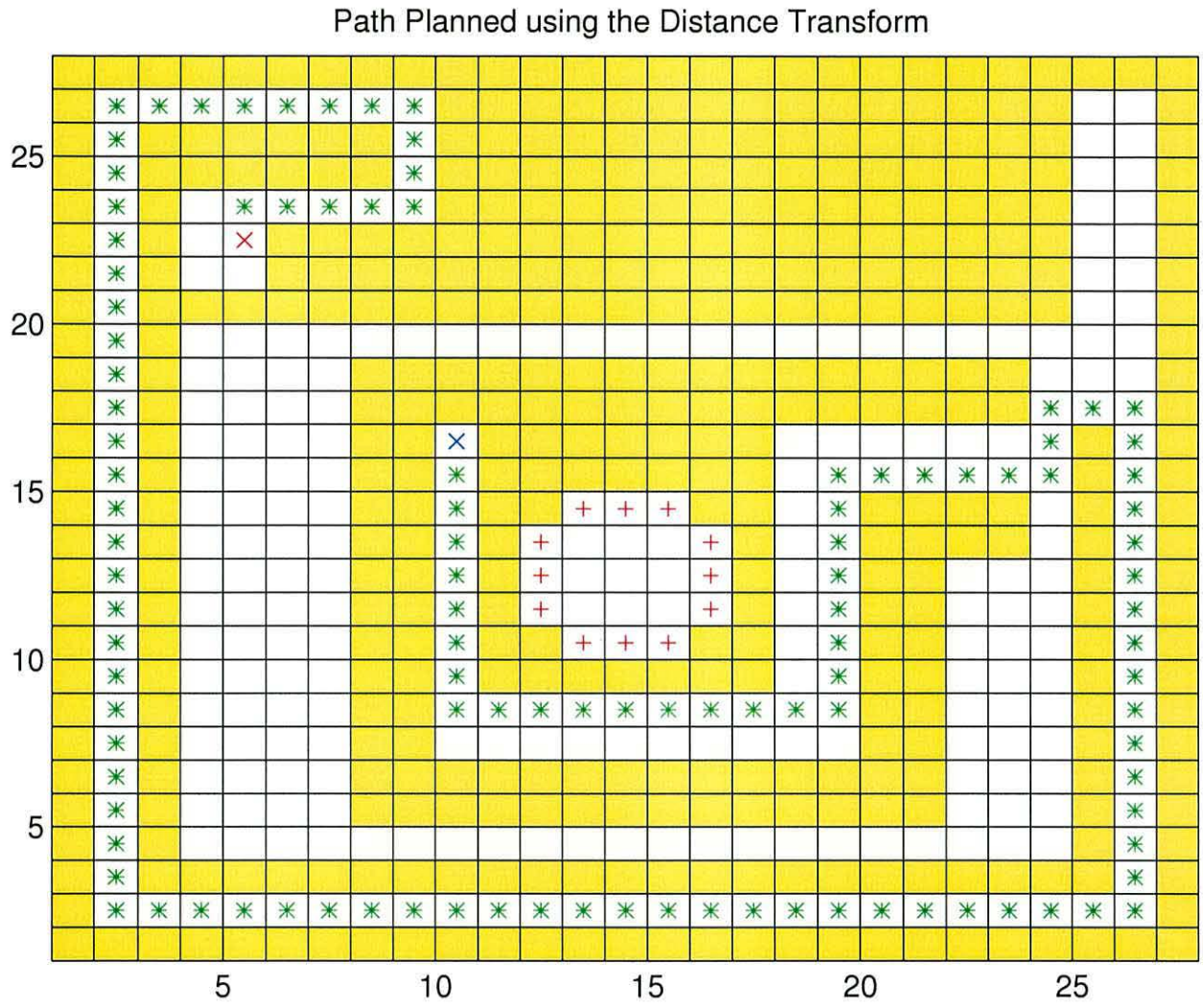


Figure 6.3: A complex workspace with the resulting path

6.1.2 Implementation of a Quadtree based Distance Transform Path Planning Algorithm in two dimensions

The matrix based software presented above showed that planning using the distance transform had clear and distinct advantages over other planning methods investigated for our intended application. However, the simple matrix implementation would not be satisfactory for use in a real-time system, especially if it were to operate in three dimensions. The main drawback of the matrix system is that it is wasteful of space in computing terms. This space inefficiency results in processing inefficiency due to the fact that all the operations had to operate on

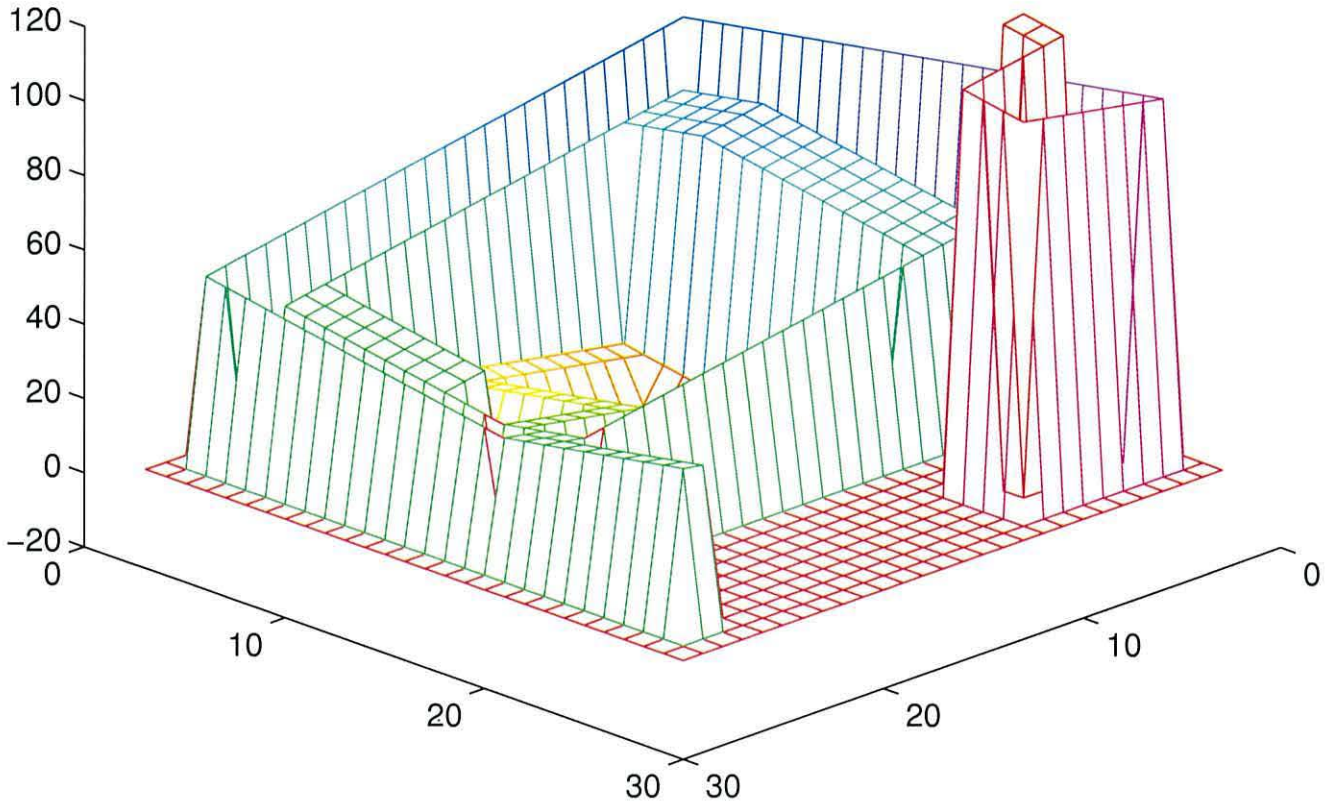


Figure 6.4: The surface produced using the workspace given in figure 6.3

a potentially large matrix. It was decided to research different methods for representing the data, which reduce the storage requirement and more importantly improve the processing time.

The method chosen was to use space decompositions based on the ideas of trees. In chapter 5, the quadtree and the octree have been introduced, in this section the processes used to implement these tree structures will be presented.

Implementation details

When the software starts execution, a workspace is loaded and stored in a matrix structure. The first process is to generate the quadtree. This is performed by successive division of the workspace as described in chapter 5. The rules used to generate the sub divisions are as follows:

$$a = (l, m) \tag{6.2}$$
$$b = \left(l + \left(\frac{(p+1)-1}{2}\right) - 1, m + \left(\frac{(q+1)-m}{2}\right) - 1\right) \tag{6.3}$$
$$c = \left(l + \left(\frac{(p+1)-1}{2}\right), m\right) \tag{6.4}$$
$$d = \left(p, m + \left(\frac{(q+1)-m}{2}\right) - 1\right) \tag{6.5}$$
$$e = \left(l, m + \left(\frac{(q+1)-m}{2}\right)\right) \tag{6.6}$$
$$f = \left(l + \left(\frac{(p+1)-1}{2}\right) - 1, q\right) \tag{6.7}$$
$$g = \left(l + \left(\frac{(p+1)-1}{2}\right), m + \left(\frac{(q+1)-m}{2}\right)\right) \tag{6.8}$$
$$h = (p, q) \tag{6.9}$$

Figure 6.5 shows the points $a, b, \ldots, h$ and the corner points of the area that is to be split, $(l, m) \rightarrow (p, q)$. Using this split method requires that the workspace is of size (s×s) where,

$$s = 2^x, x \in N \tag{6.10}$$

The fact that the workspaces are of power two sizes means that the division of the workspace can be implemented using pure integer division, which is significantly quicker than division using floating point numbers. A workspace can always be increased to the next power of two size.

After each split a new set of nodes are created. The following information is set in each of the nodes:

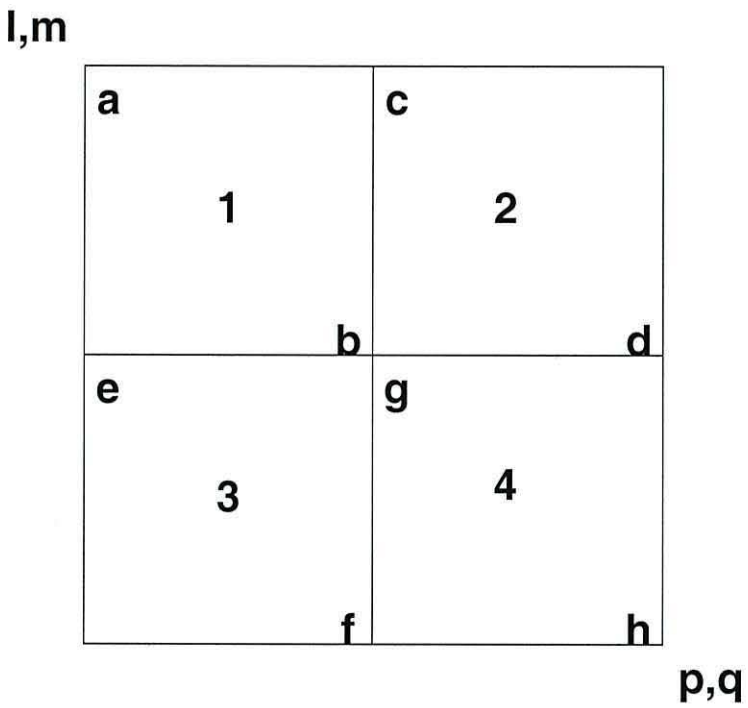


Figure 6.5: The split of a square workspace into four sub-workspaces

Node Name This records the position of the node within the tree

Parent This is the a link to the parent of the current node. All node, except the root node, have parents.

Node Classification The type of node, either obstacle, freespace or grey

Workspace coordinates This records the coordinates of the current node in the workspace. The root node covers all the workspace

Distance Transform This is set to a negative number during the node creation stage

In the first implementation the neighbourhood relationships were not held on the the nodes, but were stored in a node map matrix. As each node was created the area in the node map that the node covered was labelled with the node name. So, for example, in an 8×8 workspace, the first division would generate 4 new sub-workspaces each 4×4. The top right node, node 1, would cover the $(0,0), (0,1), (0,2), \ldots, (3,2), (3,3)$ elements in the workspace matrix, so the corresponding cells in the node map matrix would be labelled 1. This node map

allowed a simple representation of the neighbourhood relations, as just by looking at the periphery of a node in the node map would give a list of its neighbours. Refer to figure 6.6 for further details.

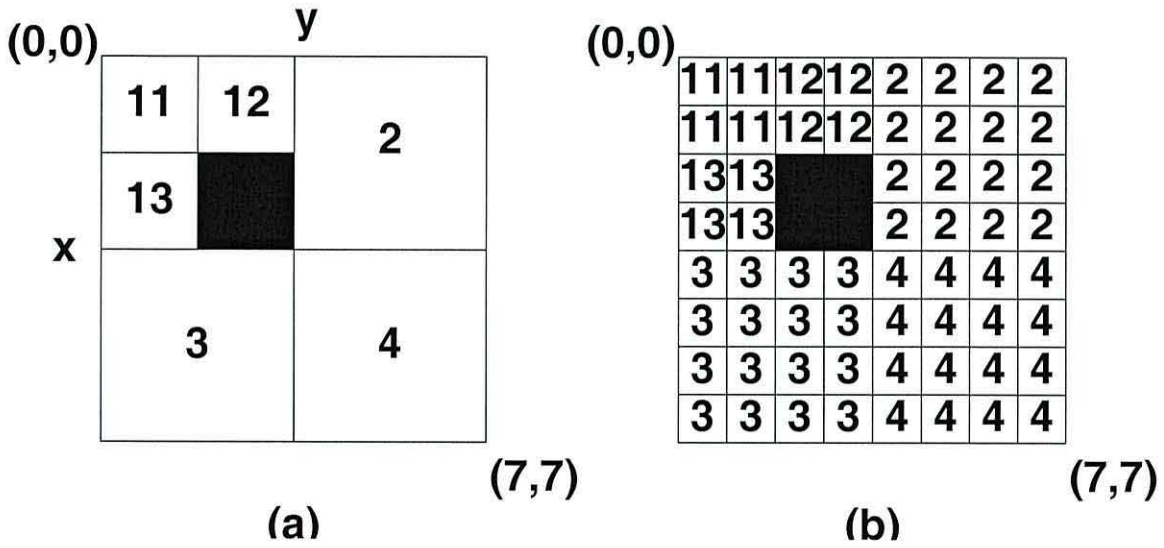


Figure 6.6: (a) Node representation, (b) Neighbour node Map

Taking node 13 for example, this covers workspace elements $(2,0), (3,0), (2,1), (3,1)$. The corresponding elements in the neighbour map are set with the value 13. So looking at the periphery of node 13 at all the elements we get a list of the form 2,2,11,11. Processing this list for duplicates we find that the neighbours of node 12 are 11, and 2. This method provides a straightforward way of storing neighbourhood relationships. However, this is not efficient as corners and edges of the map need to be catered for, and there is still a matrix to handle.

Once the workspace has been converted and the neighbourhood map has been constructed then the distance transform can be applied. This is achieved by starting at the goal node, where the distance transform value is set at zero, and searching the neighbour matrix for a list of the current node's neighbours. These neighbours are marked with a DT value of 1 and placed onto the end of a queue. The next step is to take the first node from the queue and mark its neighbours with a DT value of 2, unless they have already been assigned a value. These nodes are then placed on the end of the queue. This process of finding the neighbours, marking them with a DT value of one more than the current node and placing them on the queue continues until the queue becomes empty. This

process searches the entire workspace for paths that can be reached to the goal node. If a node cannot be reached from the goal, then it is not a neighbour to any nodes that are connected to the goal so the node never appears on the queue nor is it assigned a DT value.

The final step is to locate the start node and check if it has been marked with a DT value. If it has not then no path is possible from that start point. If the DT value is set then a simple descent based search is used to find a path from the start to the goal. Starting at the start node, find the neighbour with the lowest DT value and move to it. Then look at the neighbours of the current node and again find the one with the lowest DT value. This process repeats until the goal is reached.

This implementation proved that the division of workspace saved space, but since it still used a grid structure to store the adjacency relationships saving was lost. The next revision was to store neighbourhood relationships within the tree structure.

Results

The following figures give some examples of the workspaces that were used to test the path planner. In all cases the start location is at the top right and the goal is at the bottom left of the workspace. The workspace size was 32×32, except for the second which was 256×256.

The data produced by most of the path planning experiments is very detailed and is not presented.

The paths are given as a series of nodes represented by a pair of coordinates, $(i, j) - (k, l)$ the x direction is aligned with the vertical and the y with the horizontal in all the workspaces below.

Simple Workspace Figure 6.7 shows a simple workspace with 4 regular shaped obstacles. The node path can be seen in blue, the start point is at the top left

marked in red and the goal in the lower right marked green.

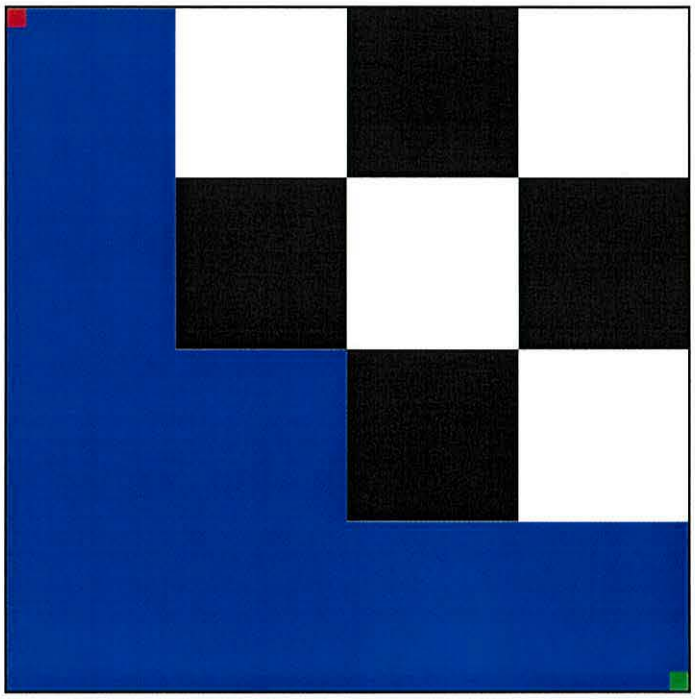


Figure 6.7: Simple workspace and node path

The configuration of the obstacles results in a small tree structure. In this case the full output from the planner is given bellow.

```
Dist1.08 - Distance transform and path planning
Output info
0 - No info
1 - Info
Enter now:
Enter the map file name:
Finding size
Size is 32
The workspace is 32x32
Do you want to see the matrix? (y|n)
Now enter Start and Goal coordinates
Enter Start Point
Enter start x:
Enter start y:
```

```
Enter Goal point
Enter goal x:
Enter goal y:
Generate quadtree
Finding Start and Goal
Distance Transform
Queue is empty
Print tree
Node: 0   Class=2 Start=0 Goal=0 Distance=-2
Node: 1   Class=2 Start=0 Goal=0 Distance=-2
Node: 11   Class=0 Start=1 Goal=0 Distance=5
Node: 12   Class=0 Start=0 Goal=0 Distance=4
Node: 13   Class=1 Start=0 Goal=0 Distance=-1
Node: 14   Class=0 Start=0 Goal=0 Distance=6
Node: 2   Class=0 Start=0 Goal=0 Distance=2
Node: 3   Class=2 Start=0 Goal=0 Distance=-2
Node: 31   Class=1 Start=0 Goal=0 Distance=-1
Node: 32   Class=0 Start=0 Goal=0 Distance=1
Node: 33   Class=0 Start=0 Goal=1 Distance=0
Node: 34   Class=0 Start=0 Goal=0 Distance=1
Node: 4   Class=2 Start=0 Goal=0 Distance=-2
Node: 41   Class=1 Start=0 Goal=0 Distance=-1
Node: 42   Class=0 Start=0 Goal=0 Distance=-3
Node: 43   Class=1 Start=0 Goal=0 Distance=-1
Node: 44   Class=0 Start=0 Goal=0 Distance=-3
Plan the path
Do you want to save the path to file? (y|n):
From (0,0) to (7,0)
From (8,0) to (15,7)
From (16,0) to (31,15)
From (24,16) to (31,23)
From (24,24) to (31,31)
```

This output shows the node tree printed out in text form. This gives the node names, a flag for the start and goal, and the distance transform value. Notice

that the obstacle points such as node 13 have a negative distance transform value.

Large Obstacle Figure 6.8 shows a workspace with one large obstacle.

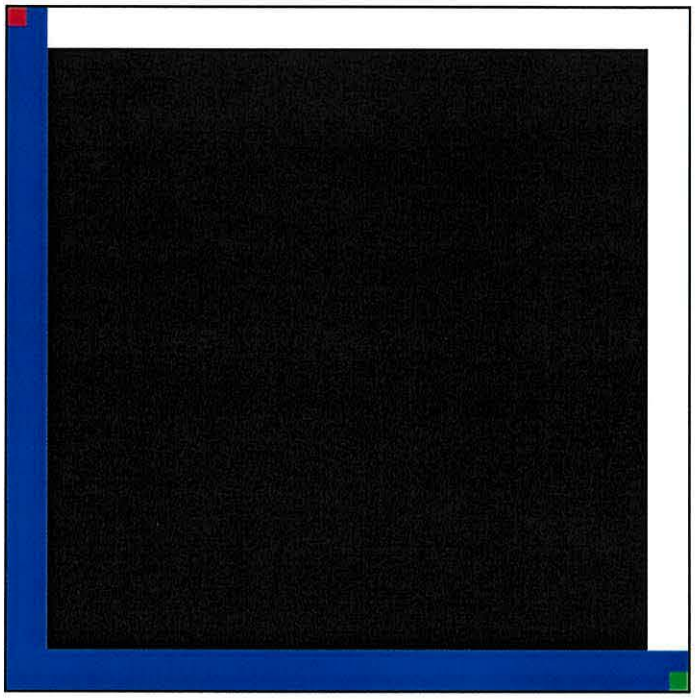


Figure 6.8: Workspace and path for one single large obstacle

The output from this run would take over 30 sides of A4 to reproduce. The workspace used for this example is 256×256 and thus the tree has a maximum of 8 levels. The path is quite simple, flowing around the obstacle to the goal.

Complex workspace with many small obstacles Figure 6.9 shows a complex workspace with many small obstacles.

This is a very complex workspace which results in a very "wide" tree, ie. there are many leaf nodes at lower levels. The blue path is shows the route from start to goal. The robot can move around in this blue region safe in the knowledge that it will not hit anything.

No path The next experiment shows the output from a workspace where no path is possible. This is included as one of the advantages of the distance trans-

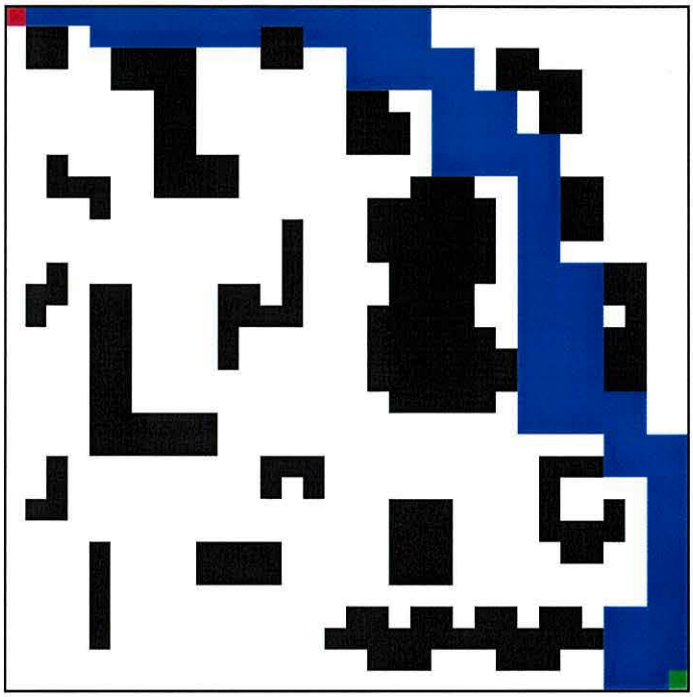


Figure 6.9: Workspace with many small obstacles showing node path

form method is the speed at which the absence of a path is determined.

The only test that needs performing on the tree is whether the start location has a distance transform value greater than one. If not no path is possible.

Complex workspaces Figures 6.11 and 6.12 show two further complex examples.

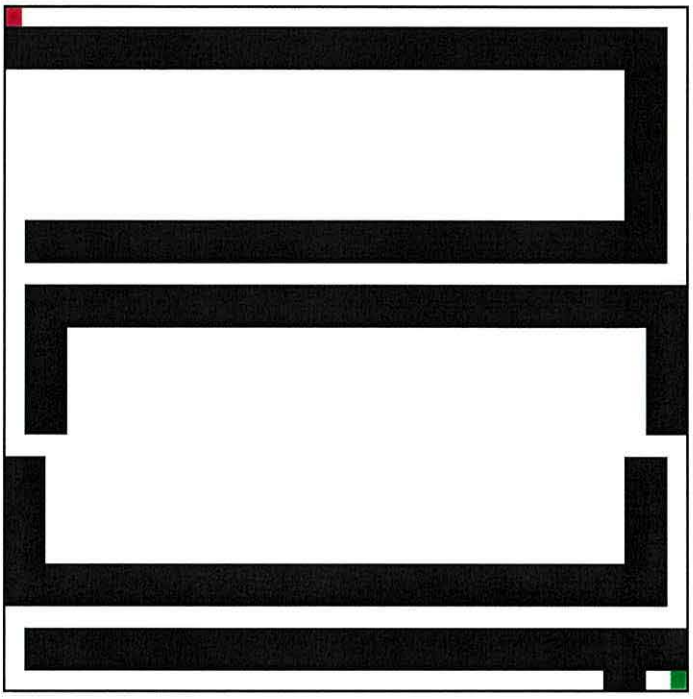


Figure 6.10: Workspace where no path is possible

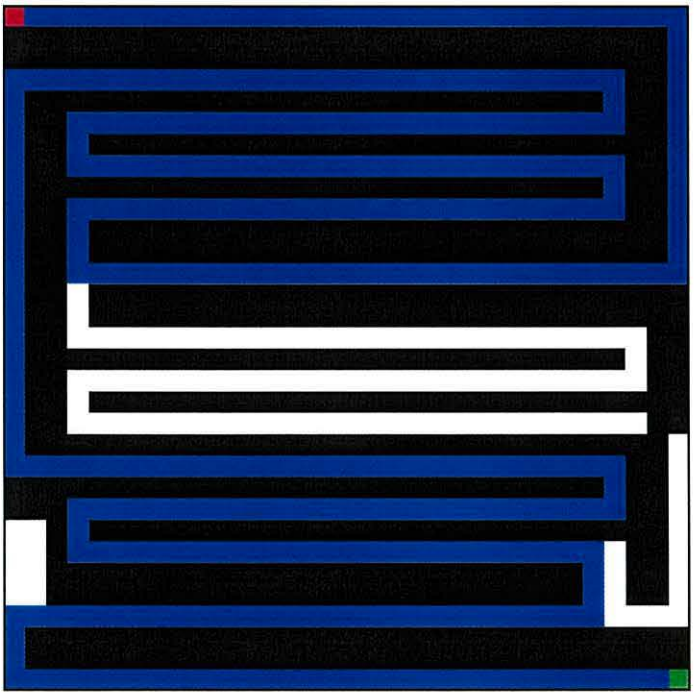


Figure 6.11: Complex workspace and node path

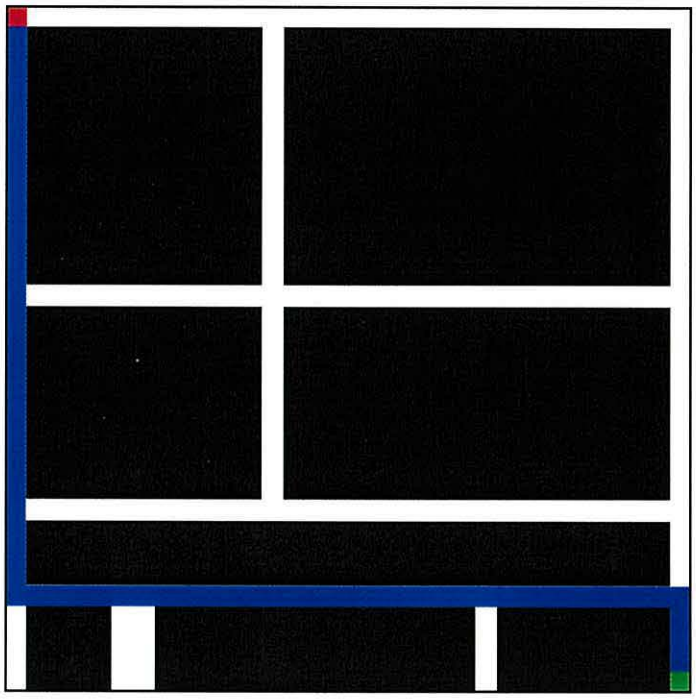


Figure 6.12: Another complex workspace and node path

6.2 Three dimensional planning

This section will reproduce the results obtained from the distance transform three dimensional planner. The results presented here show the planner operation under different workspace configurations. The output from the path planner is series of nodes that have to be passed through in order to get from the start to the goal. In order to present these results graphically the freely available *PovRay* ray tracing software was used. In all the results, obstacles are coloured blue, the goal is a green ball, and the start is a red ball. The path is shown as a black line which passes through the centre of all the nodes in the planned path.

6.2.1 Test 1: Empty workspace

This experiment shows the operation of the path planner when there are no obstacles in the environment. No spatial decomposition, tree building or path planning is performed in this case. The whole workspace is treated as one node, thus the motion path between the goal and node can just be a straight line. The workspace is shown in figure 6.13.

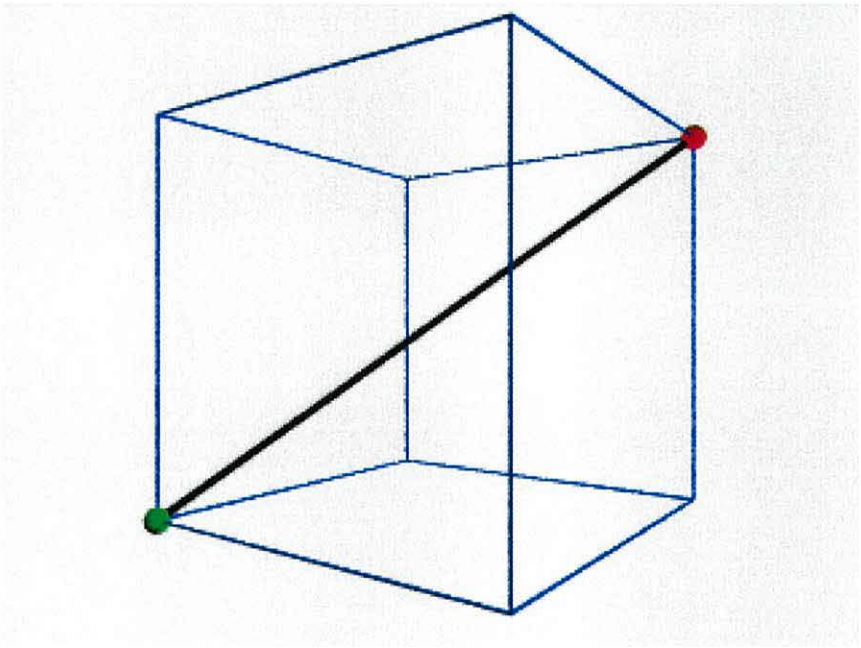


Figure 6.13: Empty workspace

6.2.2 Test 2: Large Obstacles

This experiment shows the operation of the path planner when there are only a few large obstacles in the environment. Figures 6.14 to 6.19 show a path from several different angles. Large objects are easier to deal with as they fill more of the tree up as obstacle reducing the possible width of the tree. Also, dependent on the location of the obstacles, they can limit the depth of the tree.

The path is constructed by linking the start point, through the centres of the nodes in the node path, and then to the goal point. As with the two dimensional examples above, it is impossible to include even one sample output within the thesis because of the length.

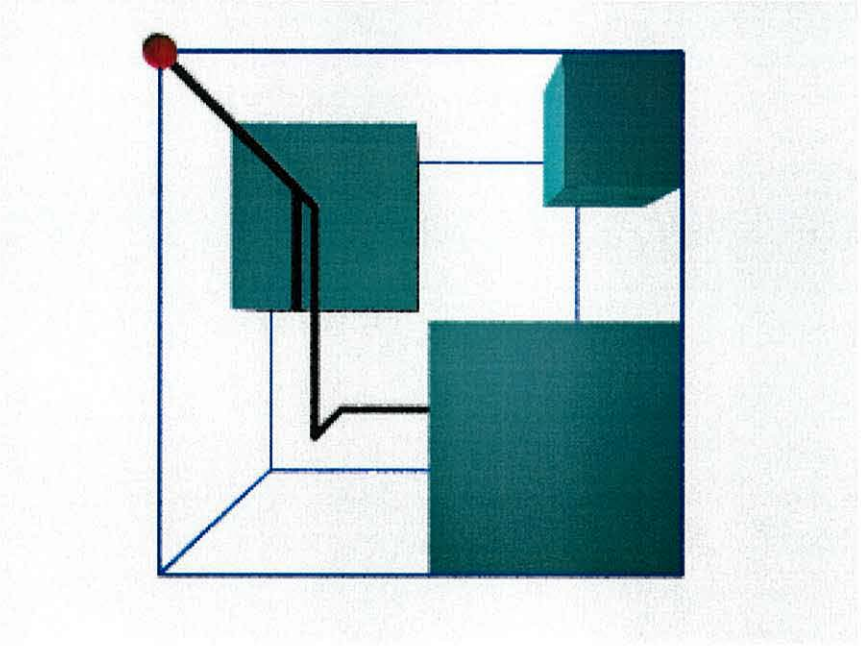


Figure 6.14: Workspace with a few large obstacles showing the path from start to goal: View 1

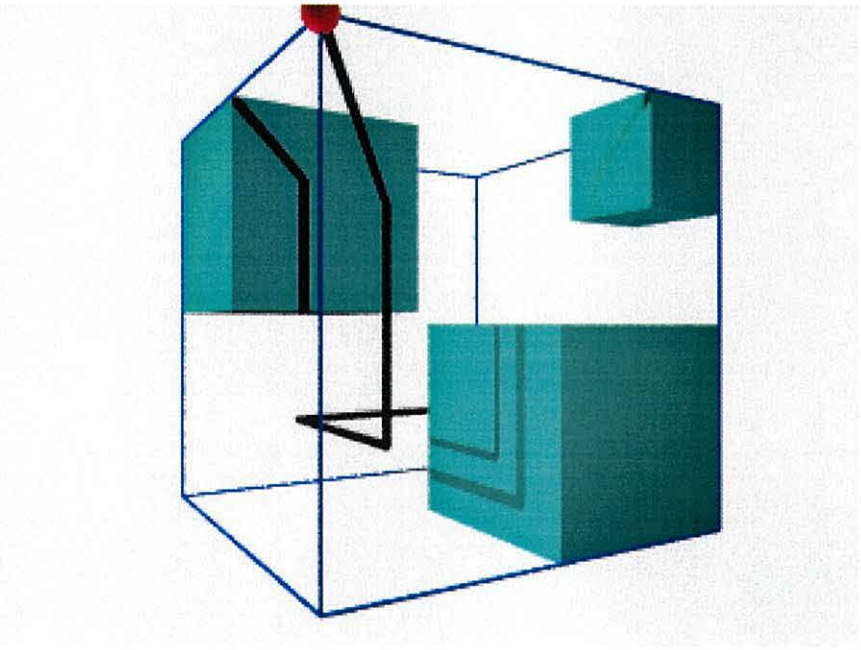


Figure 6.15: Workspace with a few large obstacles showing the path from start to goal: View 2

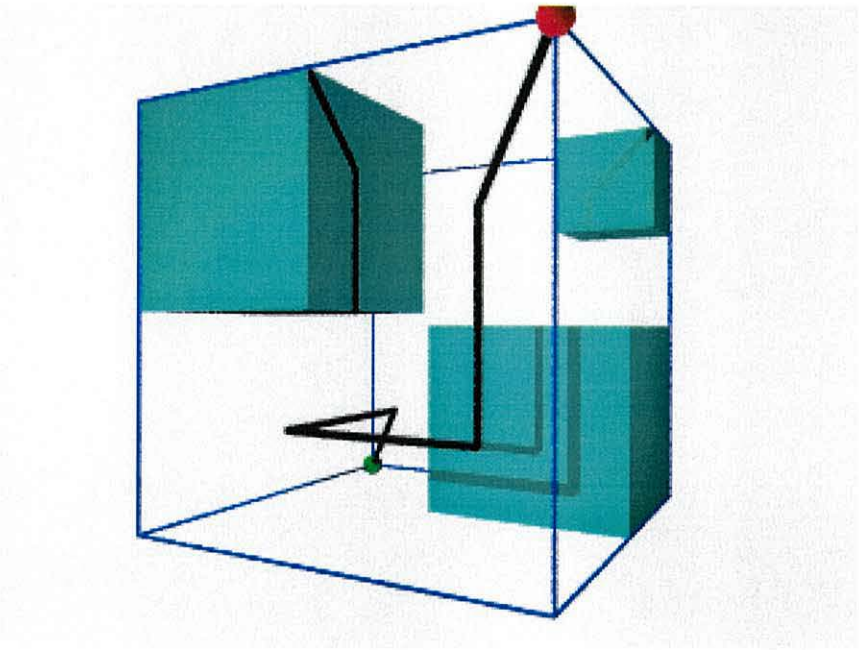


Figure 6.16: Workspace with a few large obstacles showing the path from start to goal: View 3

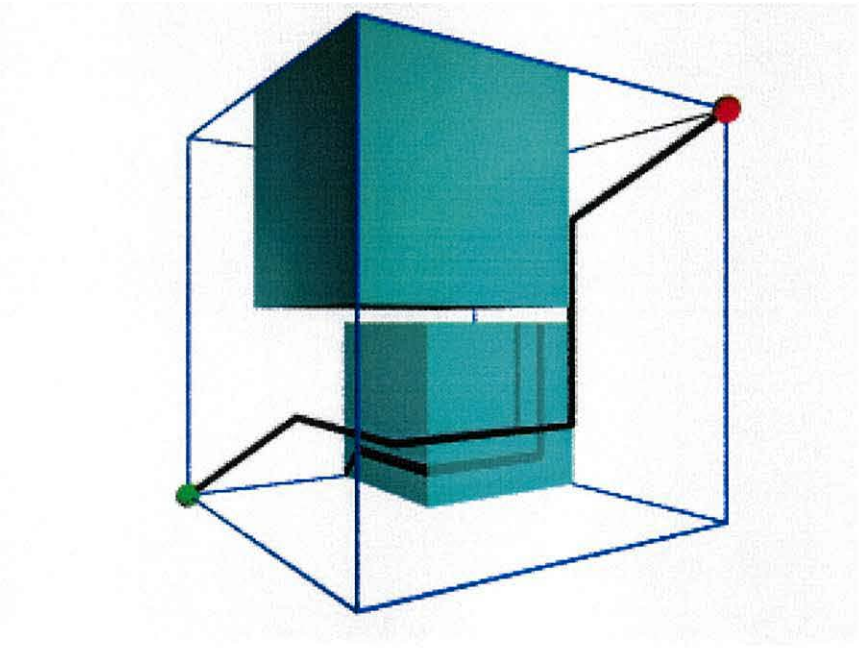


Figure 6.17: Workspace with a few large obstacles showing the path from start to goal: View 4

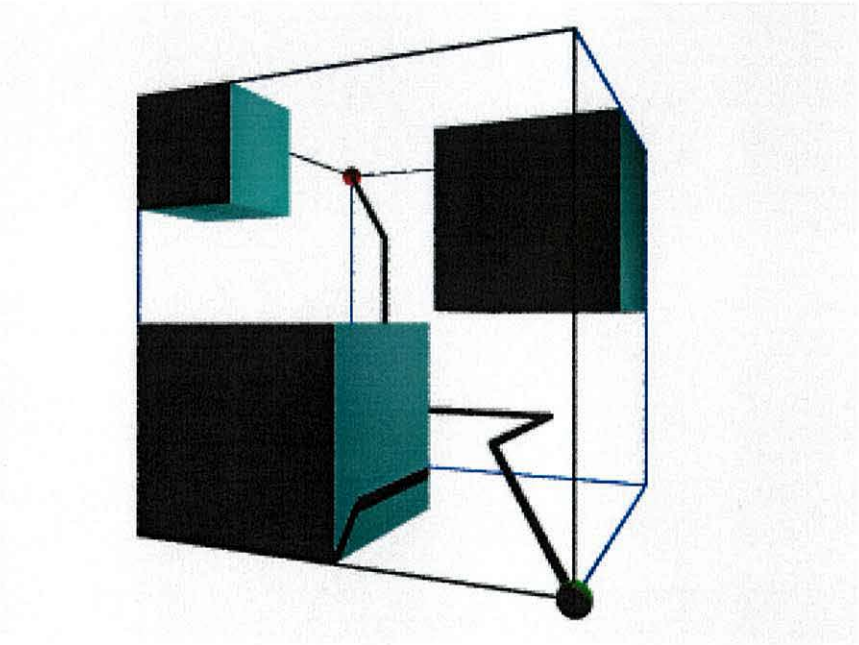


Figure 6.18: Workspace with a few large obstacles showing the path from start to goal: View 5

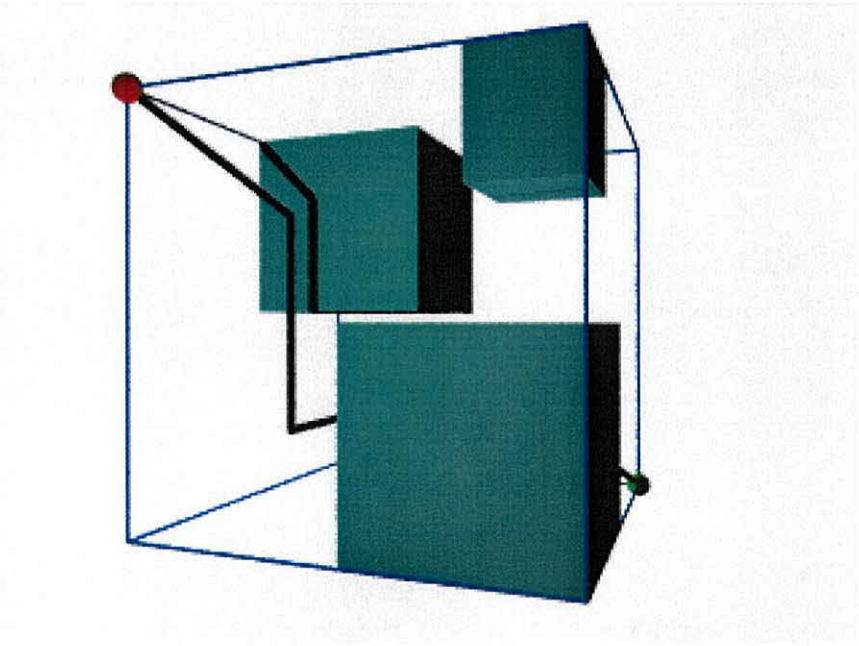


Figure 6.19: Workspace with a few large obstacles showing the path from start to goal: View 6

6.2.3 Test 3: Small obstacles

This experiment shows the operation of the path planner when there are only a few small obstacles in the environment. Figures 6.20 to 6.25 show a path from several different angles.

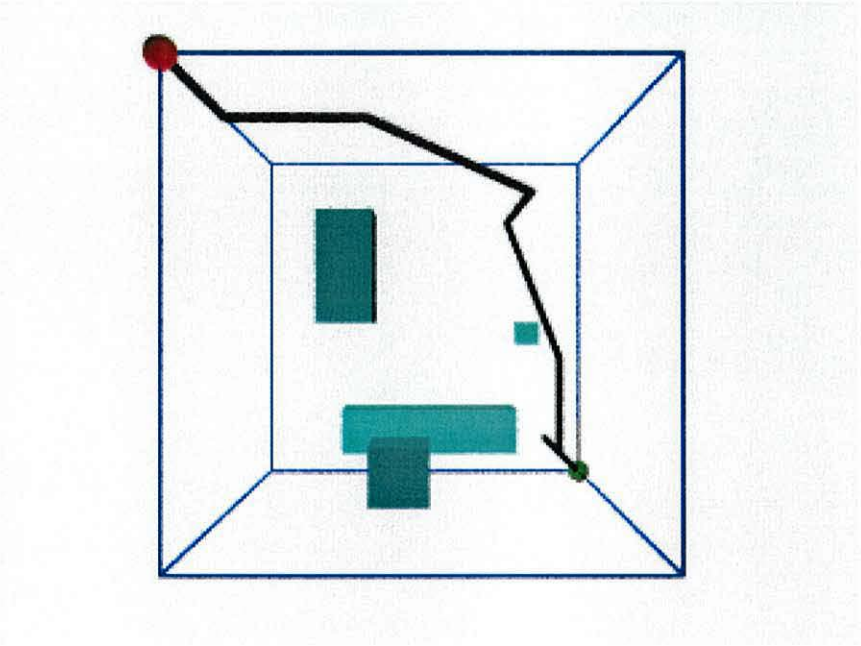


Figure 6.20: Workspace with a few small obstacles showing the path from start to goal: View 1

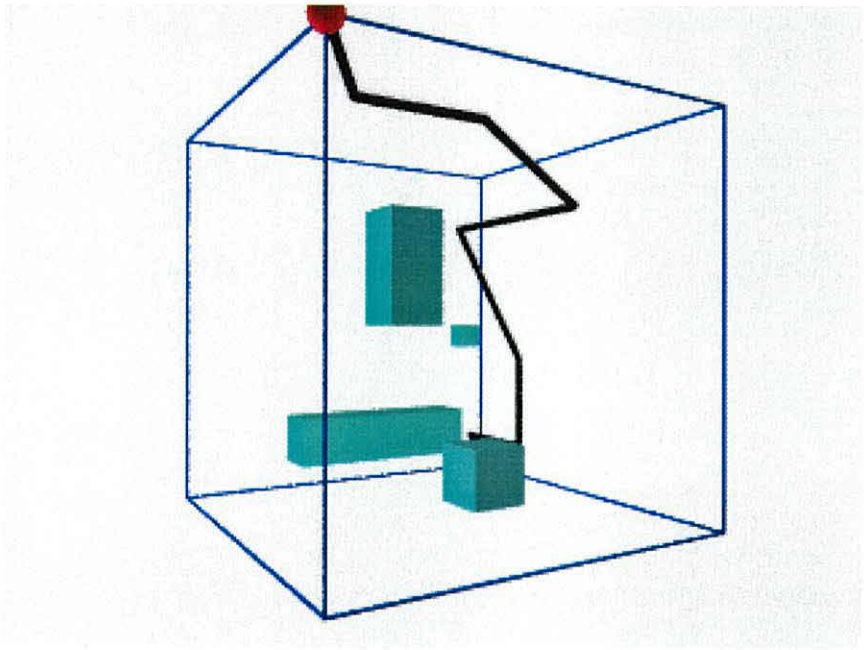


Figure 6.21: Workspace with a few small obstacles showing the path from start to goal: View 2

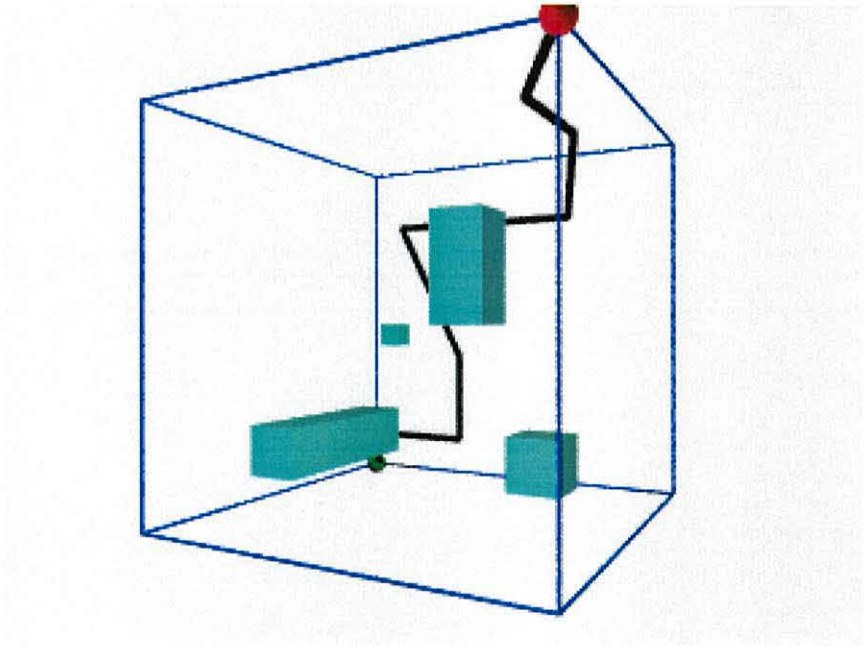


Figure 6.22: Workspace with a few small obstacles showing the path from start to goal: View 3

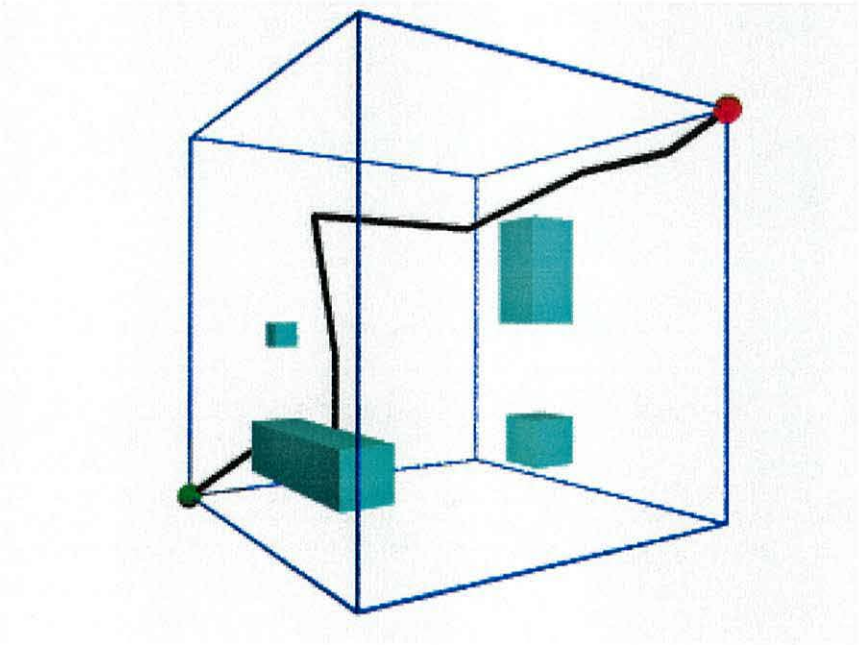


Figure 6.23: Workspace with a few small obstacles showing the path from start to goal: View 4

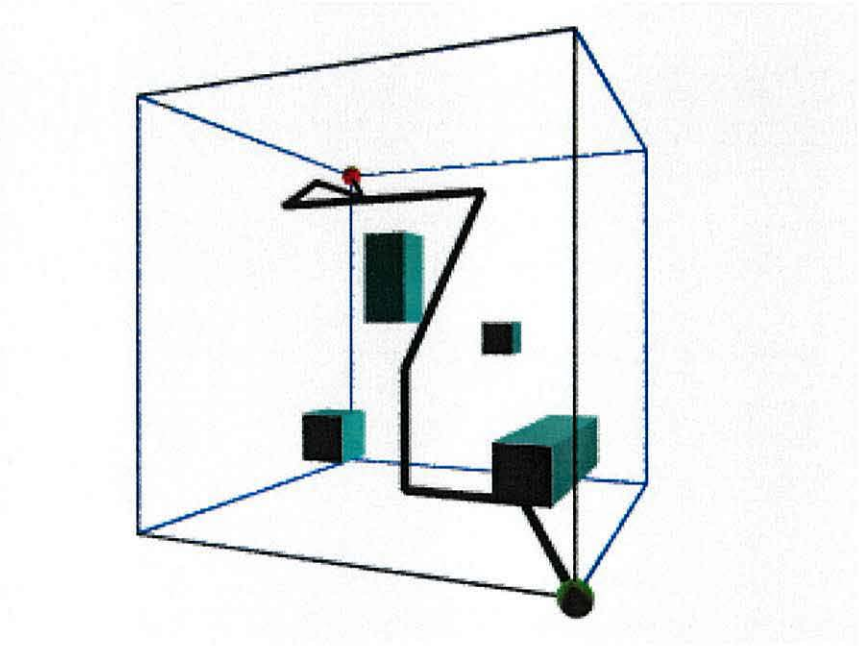


Figure 6.24: Workspace with a few small obstacles showing the path from start to goal: View 5

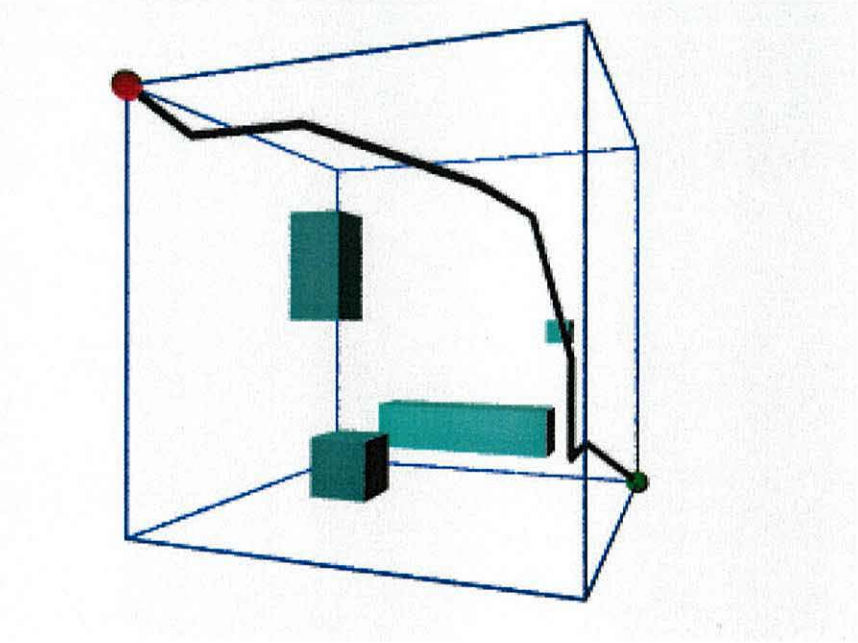


Figure 6.25: Workspace with a few small obstacles showing the path from start to goal: View 6

6.2.4 Test 4: No path

This experiment shows the operation of the path planner when there is no path between the start and goal. Figure 6.26 is included for completeness!

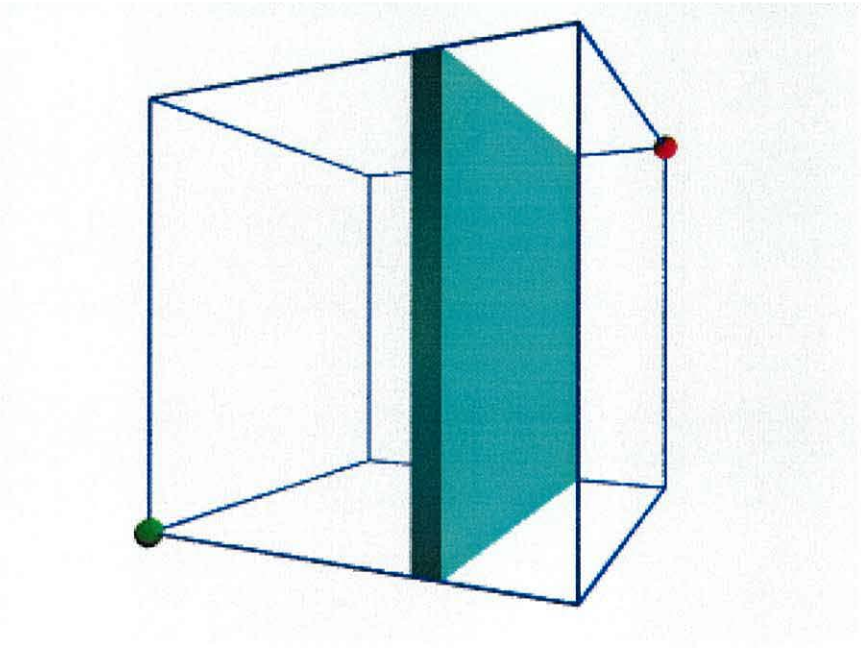


Figure 6.26: Workspace where no path is possible

6.2.5 Test 5: Complex workspace

This experiment shows the operation of the path planner when there are only a few small obstacles in the environment. Figures 6.27 to 6.32 show a path from several different angles.

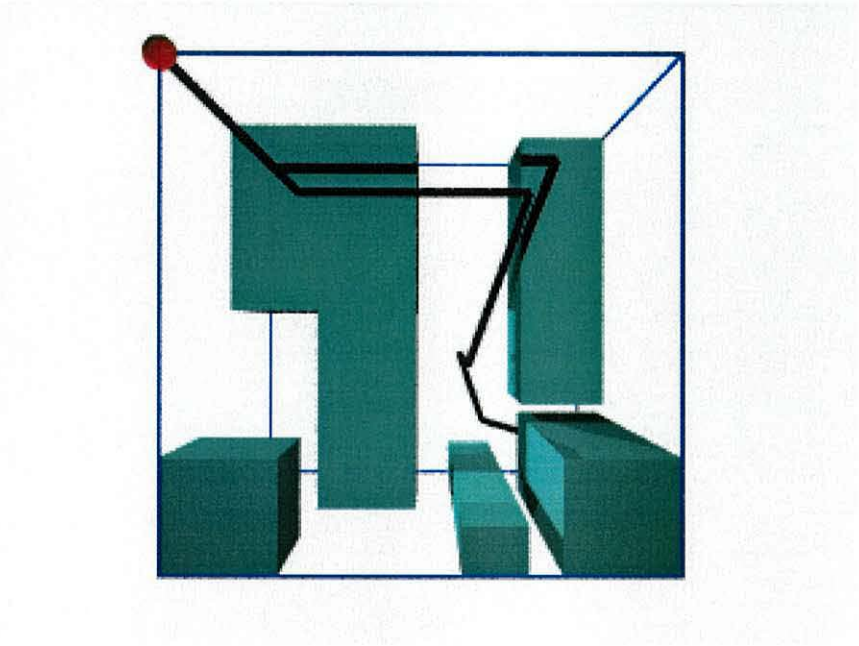


Figure 6.27: Complex workspace showing the path from start to goal: View 1

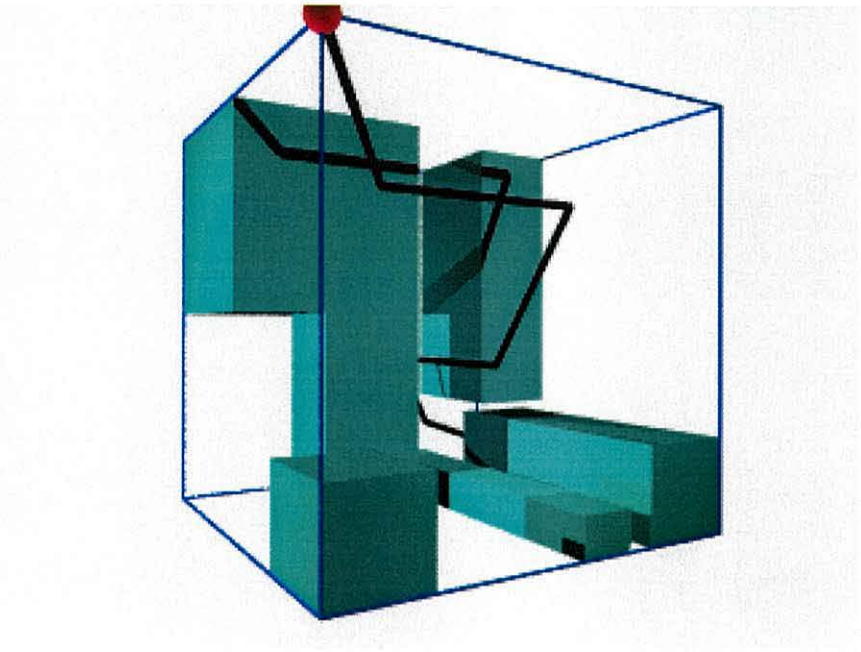


Figure 6.28: Complex workspace showing the path from start to goal: View 2

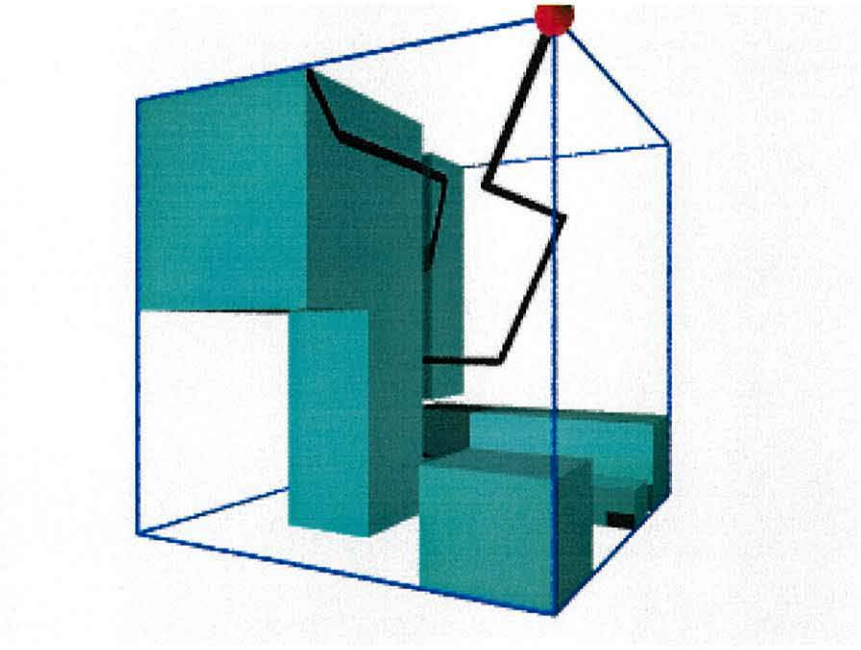


Figure 6.29: Complex workspace showing the path from start to goal: View 3

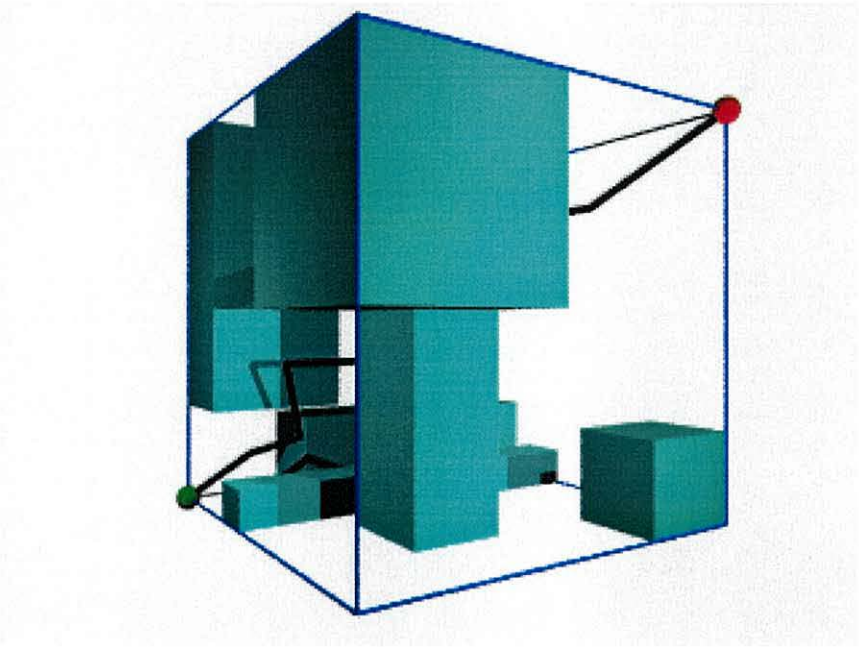


Figure 6.30: Complex workspace showing the path from start to goal: View 4

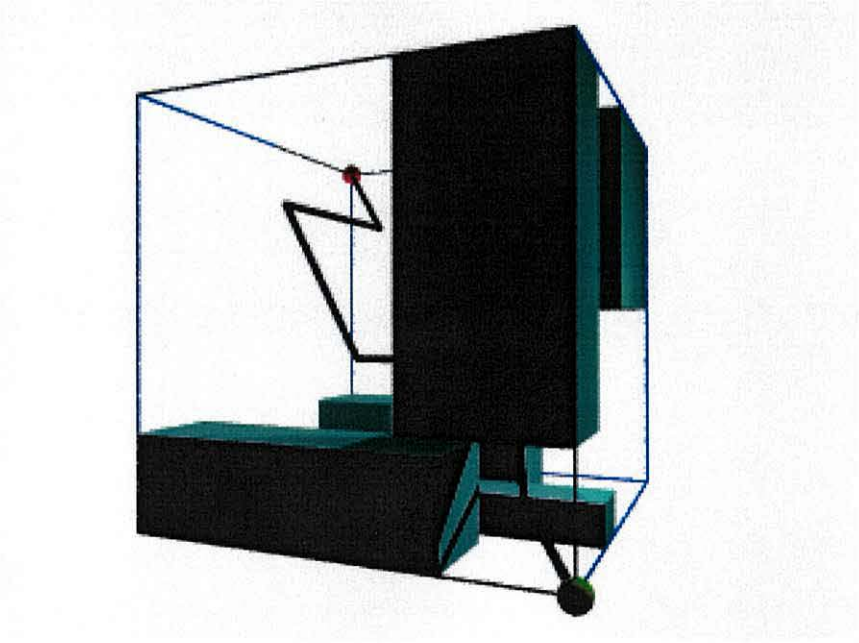


Figure 6.31: Complex workspace showing the path from start to goal: View 5

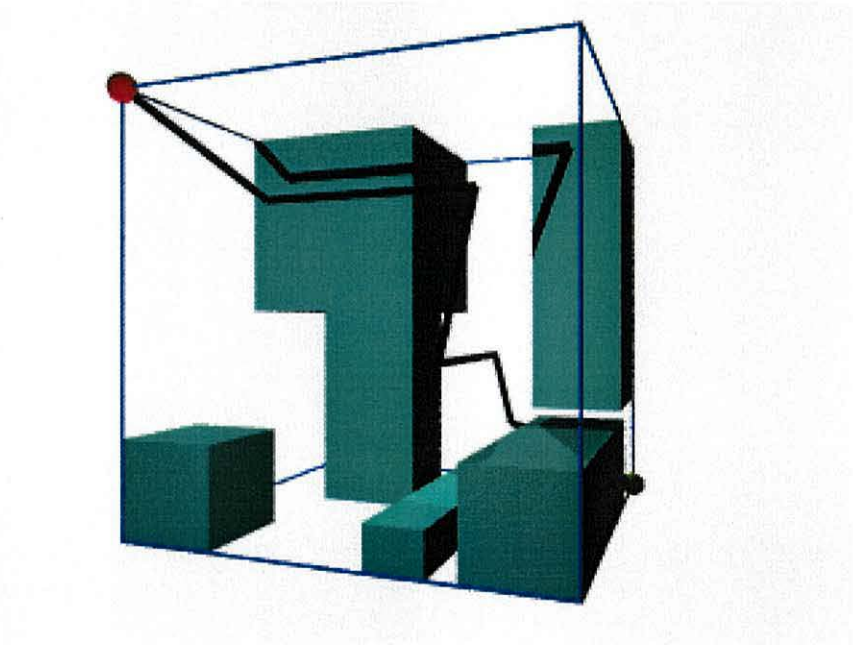


Figure 6.32: Complex workspace showing the path from start to goal: View 6

	Test 1	Test 2	Test 3	Test 4	Test 5
Execution Time	30ms	50ms	60ms	50ms	100ms

Table 6.1: Execution time of the three dimensional path planner

Performance The performance of the developed method was tested by measuring the time of execution. Because the simulation was performed using a multitasking operating system, it was necessary to run the experiment a number of times and then to record the minimum run time so that the overheads due to the operating system were minimised.

The results are shown in table 6.1. The first test is performed on an empty workspace in order to determine a reference processing time. The main component of the processing time is testing the occupancy of the workspace. In this case the whole workspace has to be scanned. The execution time is 30ms.

Test 2 uses the workspace presented in figures 6.14. In this test there are a few large obstacles, this tests the performance of the basic tree generation algorithms. The processing time of 50ms is less than twice that of the empty workspace.

Test 3 shows the effect of a number of small obstacles has on the processing time. In this test more time is needed to construct the tree structure and to link the neighbours was there are more nodes in the tree. This test workspace is shown in figure 6.20. This has only increased the processing time by 10ms to 60ms, twice that of the empty workspace case.

Test 4 uses the workspace given figure 6.26. In this case no path is possible. The execution time of 50ms shows that the path planner can quickly report when no path is possible to the goal. Remember no actual path planning is performed in this case. The application of the distance transform determines that no path is possible, therefore no execution time is wasted.

Test 5 shows the execution time for a complex workspace with a set of large and small obstacles. The execution time is 100ms. This is just over three times that of the empty workspace.

The execution times should only be used as a guide as they are PC and operating system dependent, but they do show that the developed method can quickly produce paths and quickly determine when no path is possible.

6.3 Summary

This chapter has described the results from the various experiments performed using the distance transform. A two dimensional feasibility study of the distance transform path planner implemented using MATLAB has been developed into the rapid three dimensional path planner which incorporates spatial decomposition and octrees.

Overall the experiments have shown that the three dimensional path planner developed [14] has contributed to the field of path planning, especially path planning for a mobile robot operating in three dimensions.

The next chapter describes the development of a hierarchical controller, defined in chapter 3, to integrate the path planning system described in chapter 5 with the machine vision system described in chapter 4.

Chapter 7

Hierarchical Controller Implementation

This chapter describes the design and construction of the controller defined in chapter 3. It has been implemented as a computer controlled test rig whose purpose is to allow the machine vision and path planning software described previously to be tested as a complete system. The controller hierarchy described in chapter 3 includes three levels but in the controller described here only the *pilot* level is implemented fully, the other two levels being implemented only to the point that allows the third to function.

In order to test the combination of the machine vision and path planning systems the decision was made to construct a test rig. The alternative option was to produce a test environment in software. This option was not chosen as it was thought to be too complex and would end up dominating the project.

7.1 Test Rig

The test rig is a large X-Y table with position and velocity control provided by custom built electronics. The Y axis is aligned with the parallel tracks, the X axis is perpendicular to this. Figure 7.1 shows the test rig.

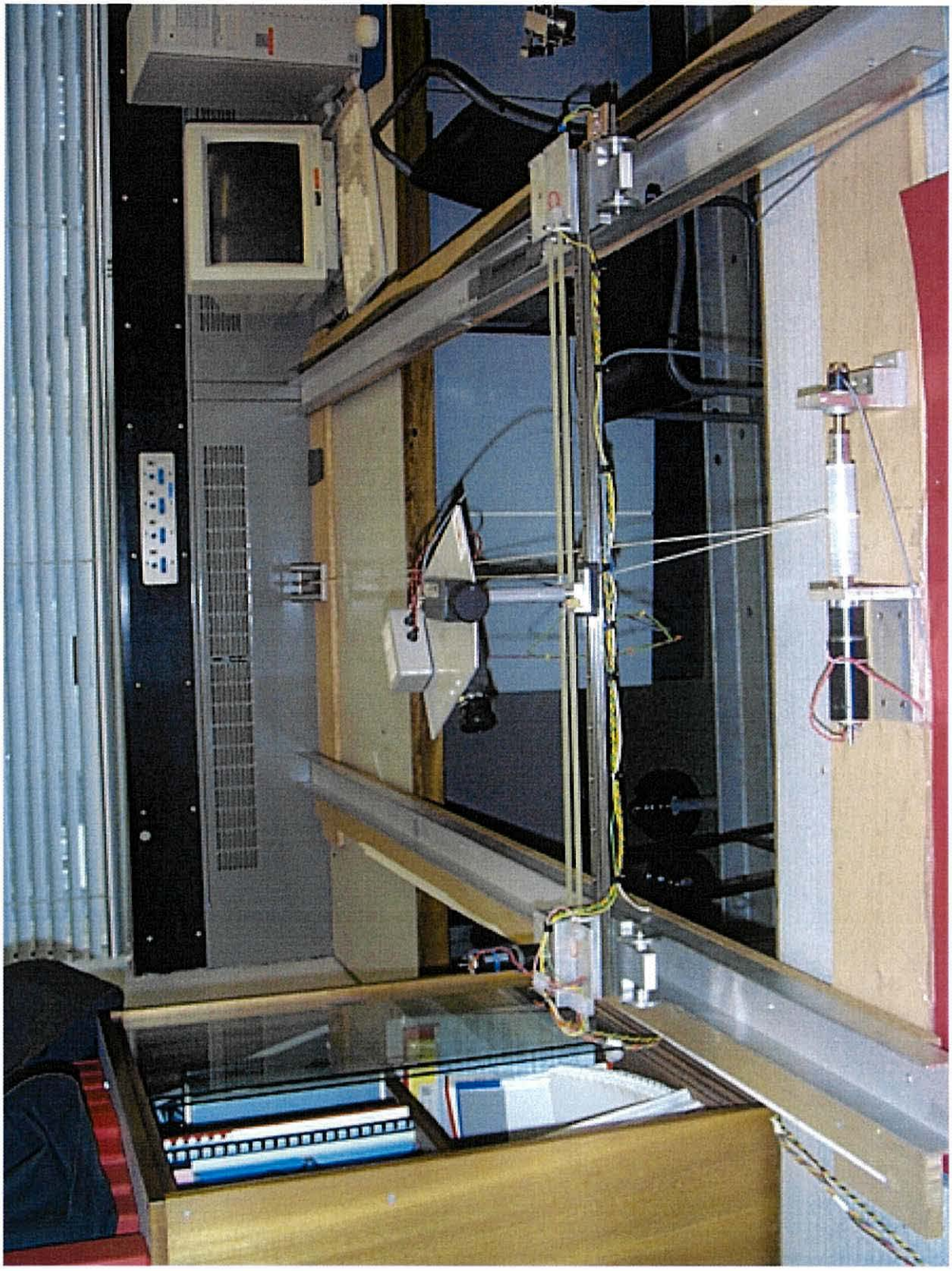


Figure 7.1: The test rig

In the figure two aluminium channels can be seen which are screwed down onto a wooden frame to keep them parallel. On each channel a carriage with two pairs of wheels is positioned. A linear rail is fixed between these two carriages. The linear rail has another carriage fixed to it which supports the camera mount fixings. Two pulley housings are fixed onto the two carriages which support a toothed belt connected to the carriage on the linear rail. A motor fixed to one pulley housing allows the linear rail carriage to be pulled in the X direction (positive X is right to left in figure 7.1).

In order to achieve motion in the Y direction, positive Y is bottom to top in figure 7.1, a capstan and pulley mechanism is attached to the wooden frame between the two channels. At the bottom of figure 7.1 a motor-capstan-tachometer can be seen. A piece of string is connected to the centre of the linear rail around the capstan, then to the pulley at the back of the test rig and back to the linear rail.

The test rig is controlled using custom built electronics as shown in figure 7.2. There are four main circuits, two of which drive and control the x and y motors. The x control is achieved using a potentiometer wire for position control, velocity is controlled in the Y direction using the feedback signal from a tachometer connected to the drive shaft of the motor/capstan assembly. The bespoke optical encoder circuit allows the position of the test rig in the Y direction to be measured. The final circuit interfaces the test rig electronics with the PC.

Figure 7.3 shows a block diagram of the controller electronics.

The rest of this section will describe in more detail the design and construction of the test rig both mechanically and electronically. The final section will then describe the software design and implementation.

7.1.1 Mechanical Design

The test rig was designed by the author. A large proportion of the construction was also completed by the author with assistance from School's Mechanical Workshop staff. Figure 7.4 shows a simplistic representation of the test rig. The

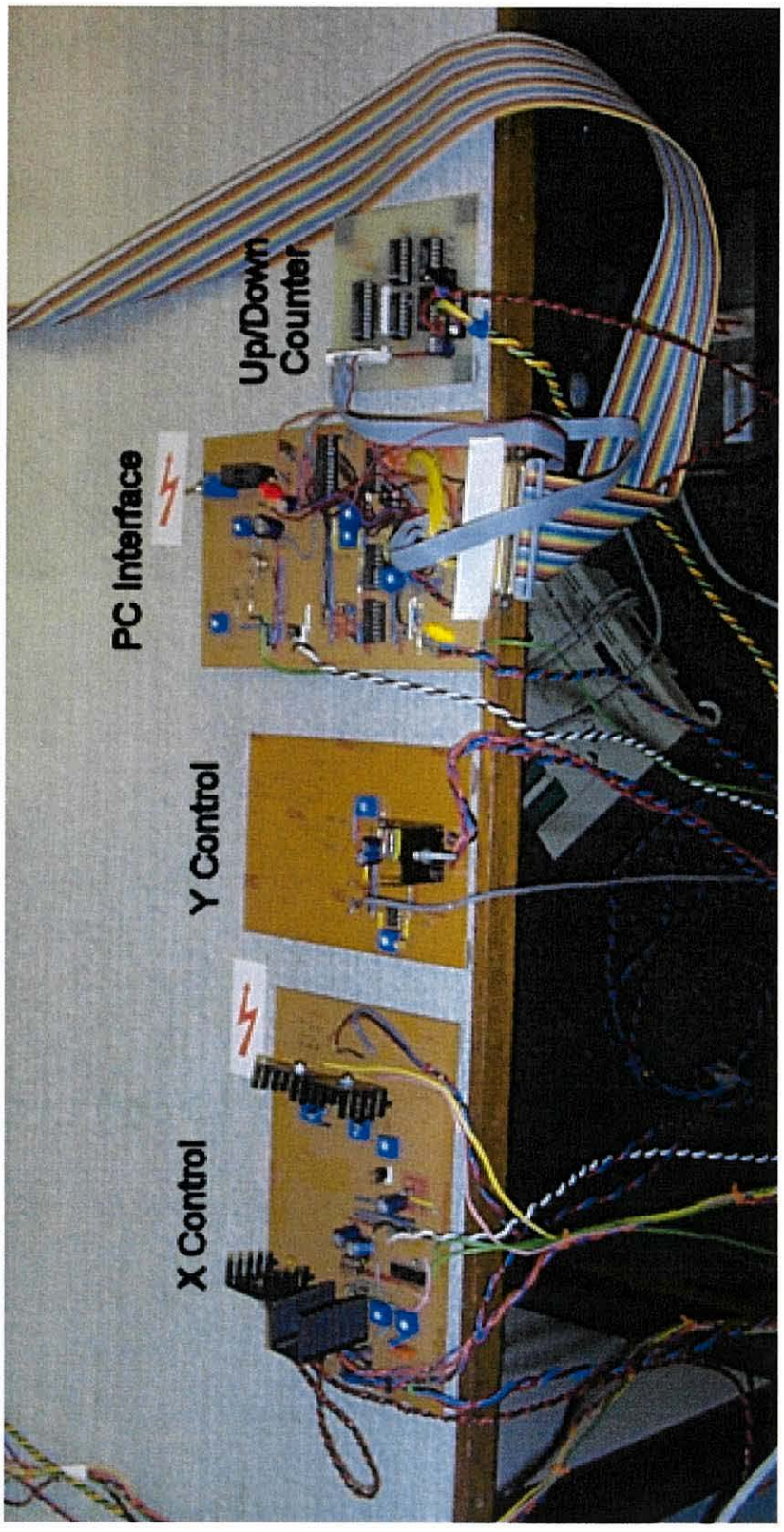


Figure 7.2: The Electronics

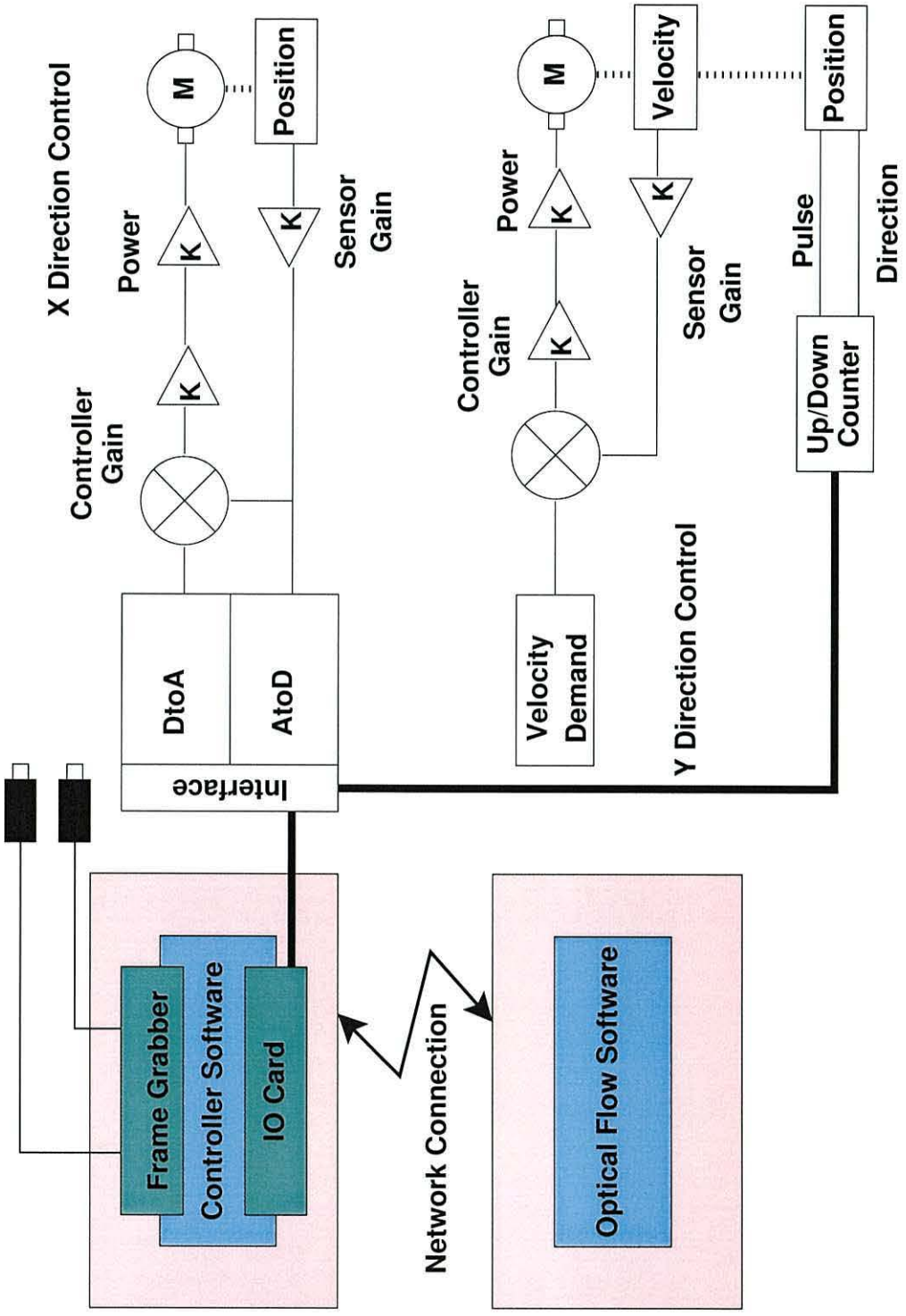


Figure 7.3: Controller block diagram

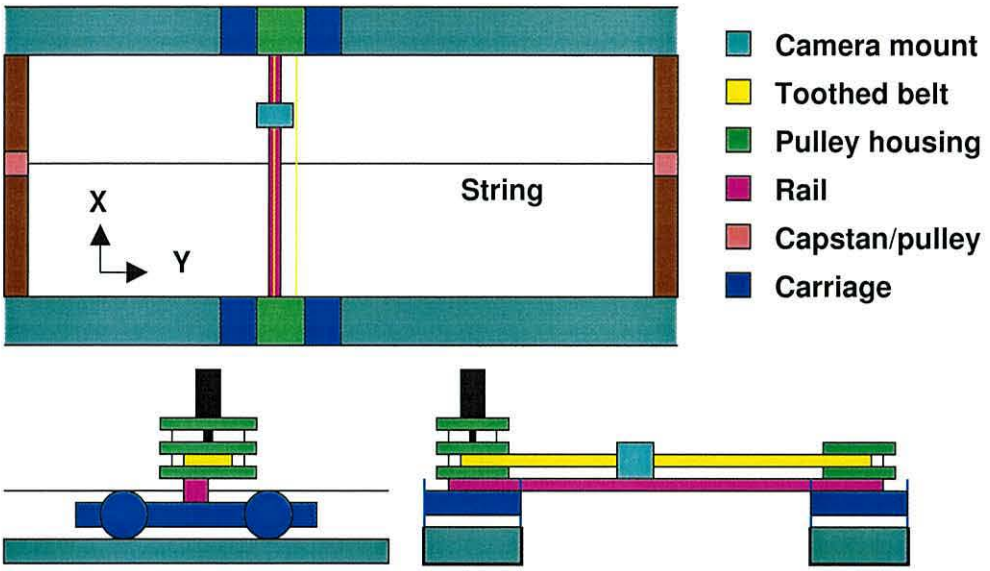


Figure 7.4: Simplified Rig Design

two aluminium rails are coloured cyan, the two carriages are coloured blue, the linear rail is magenta, the pulley housing green and the capstan/pulley system is pink. The yellow toothed belt is mounted on the pulley held in the pulley housing.

The camera mount is fixed onto a trolley that is moved along a linear rail by a toothed belt connected to a motor and gearbox. Figure 7.5 shows two cameras on a mounting plate which is connected onto a carriage fixed onto a linear rail. Also on the plate is a connection box to power the cameras. In the figure one of the cameras is mounted upside-down due to space limitations. Any images captured have to be inverted before they can be used.

The carriage on the linear rail traverses by means of a toothed belt connected to each side of the carriage. Figure 7.6 shows the toothed belt passing through a pulley housing.

Running parallel with the linear rail is a length of eureka wire, shown in figure 7.7, which is used in the positioning control system. A brass wiper moves along the wire allowing the voltage to be detected. The problem found with this method was oxidisation of the wire which affected the detected voltage. It was necessary to change the wire regularly to keep the positioning accuracy consistent.

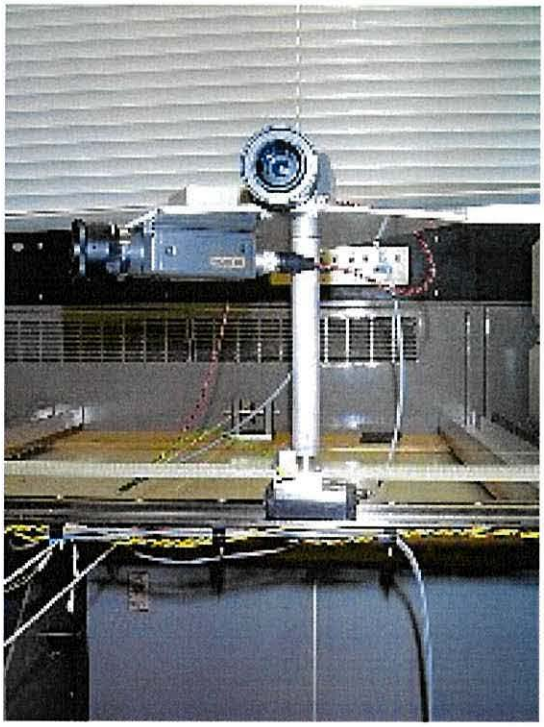


Figure 7.5: Camera mount

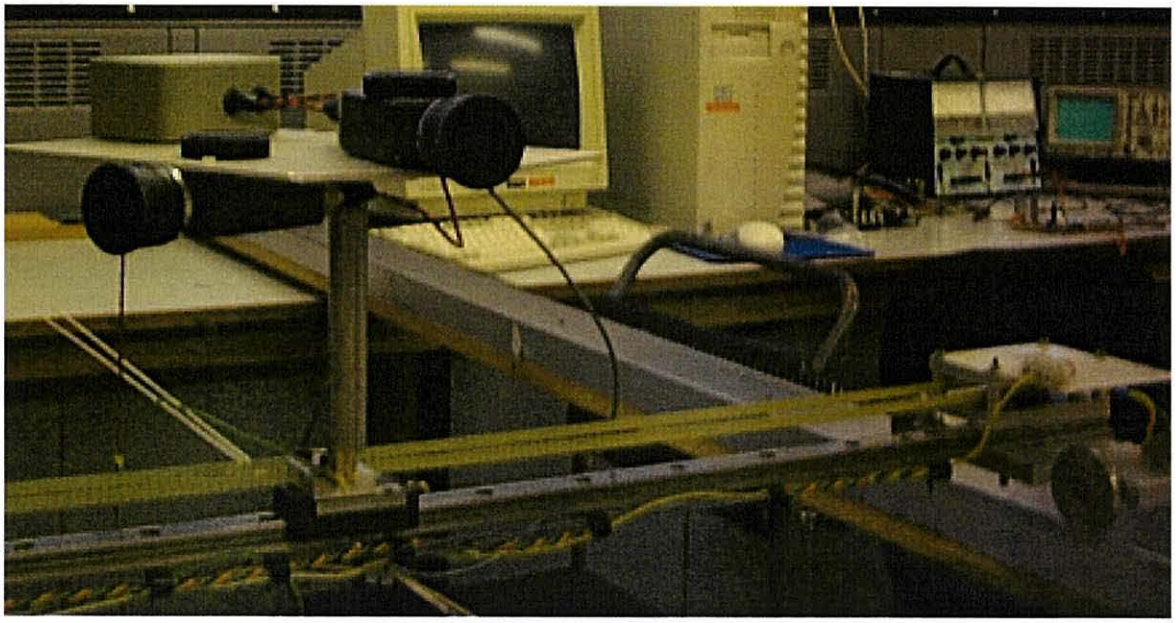


Figure 7.6: Toothed belt

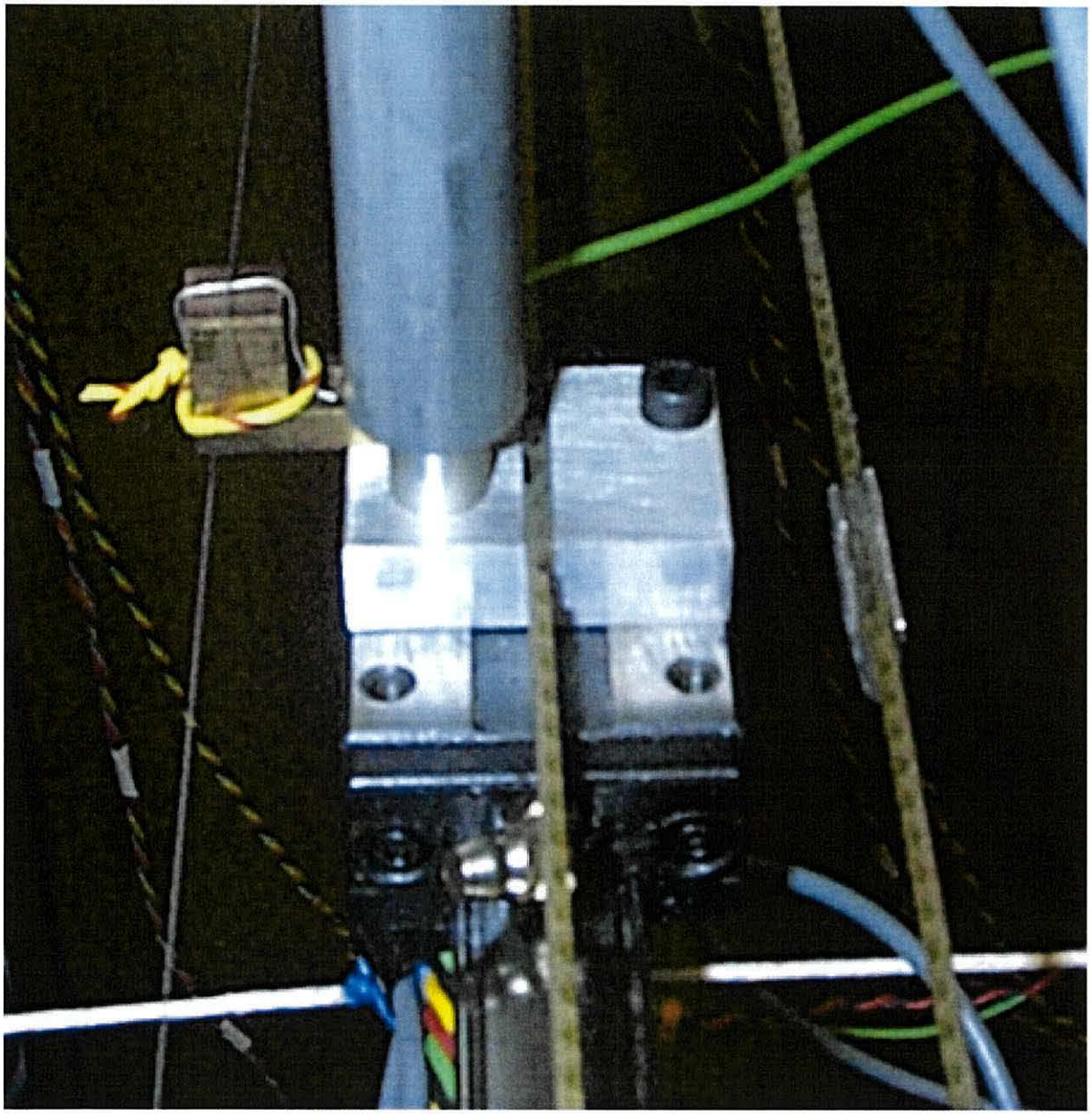


Figure 7.7: Potentiometer position sensor

The toothed belt is held on two pulleys; figure 7.8 shows one pulley housing.

Figure 7.9 shows the top and bottom view of the two plates that make up the housing. The cut-out in the plate holds a bearing for the shaft of the pulley to be mounted into. The plates are separated by 4 pillars which are hollow to allow the fixing bolts to pass trough the structure. An extra plate is used to mount the motor. The motor shaft is connected to the pulley shaft by a flexible coupling which reduces the problem of shaft alignment and can act as a "fuse" should there be a problem with the controller.

The pulley housings are mounted on two carriages. Figure 7.10 shows the pulley housing mounted on the carriage which is in turn mounted on an aluminium channel. Figure 7.11 shows the design of the carriage.

Also visible in figure 7.10 is the Y positioning sensor. This is made from a strip of acetate which has an alternating opaque/transparent pattern printed on it. The acetate strip runs between two optical sensors which are positioned 90° out of phase relative to the pattern. This allows the direction of motion to be determined.

A motor and gearbox plus capstan is mounted between the channels at one end. A length of string is connected to the middle of the linear rail, via a single turn on the capstan, then to a pulley at the other end of the rig and back to the linear rail. Figure 7.12 shows the capstan drive system.

7.1.2 Electronic Design

The position of the camera mount in the X direction is measured by a length of resistance wire which has a constant current flowing through it. The circuit used to control the voltage on this wire consists of two adjustable voltage references, the voltage being fine tuned with potentiometers. Figure 7.13 shows the circuit. The potential difference across the wire is ±2V and the centre of the wire should then be 0V.

The voltage measured at a point on the wire is therefore directly proportional to

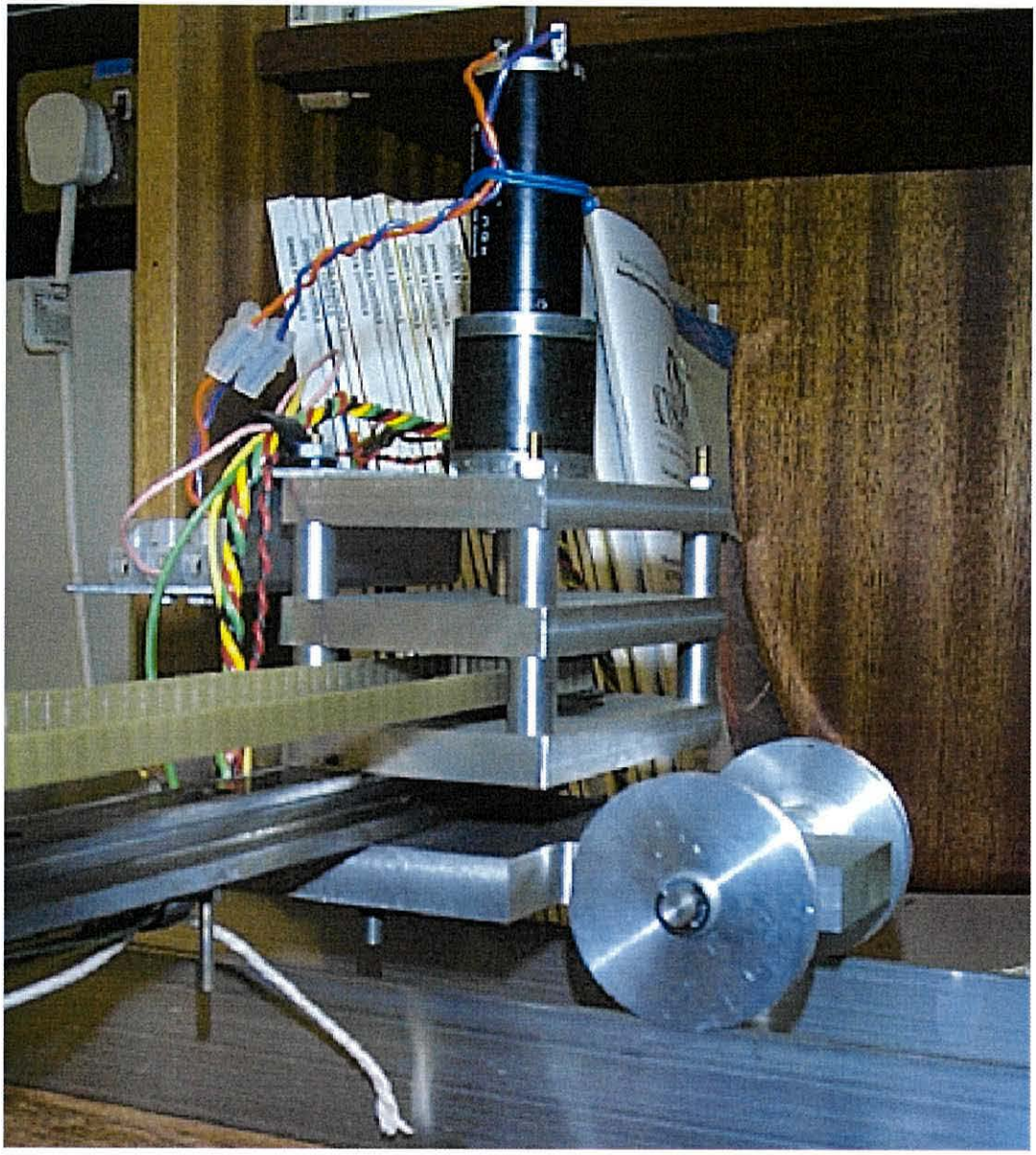


Figure 7.8: A view showing the toothed belt pulley arrangement

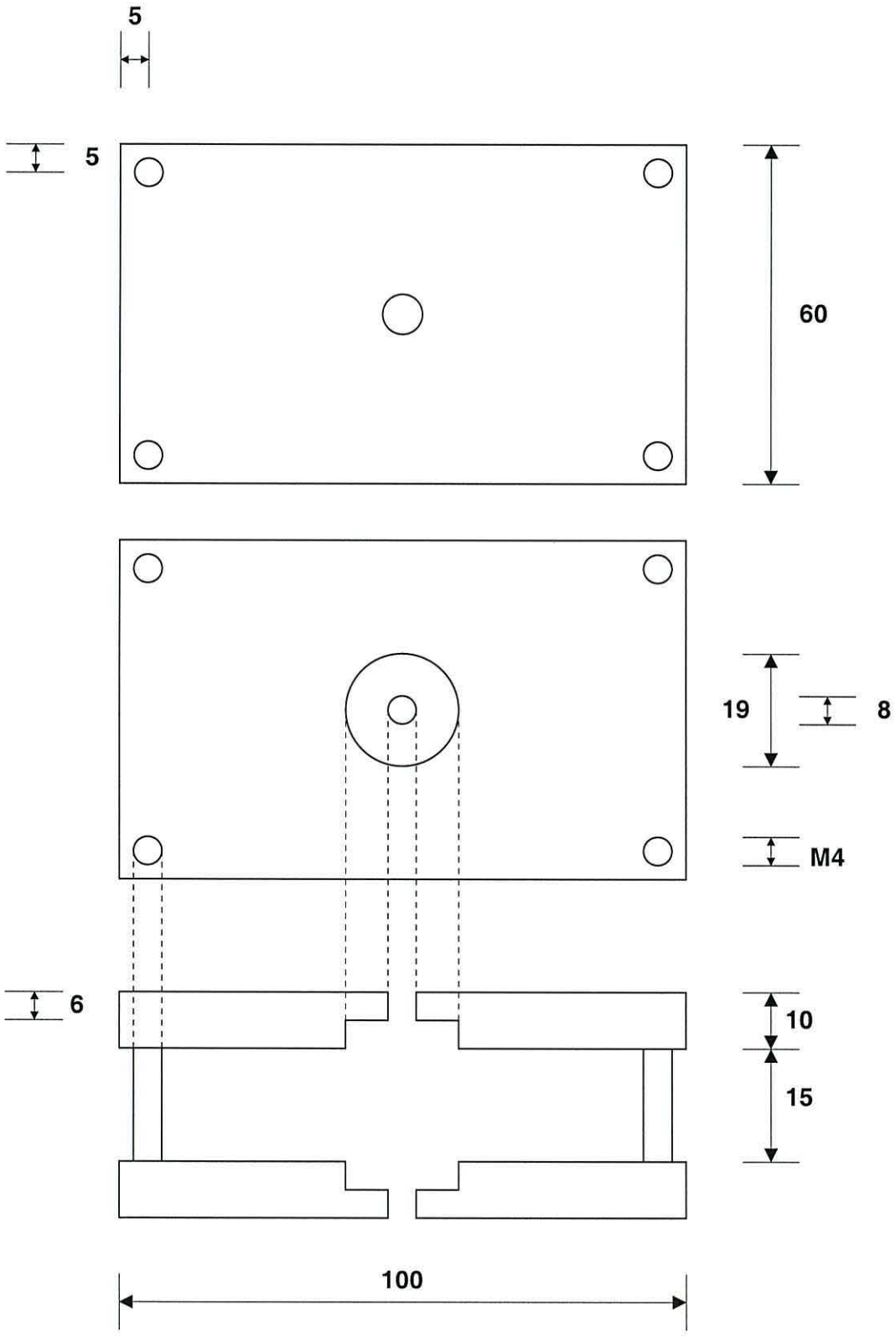


Figure 7.9: Mechanical design of the pulley housing

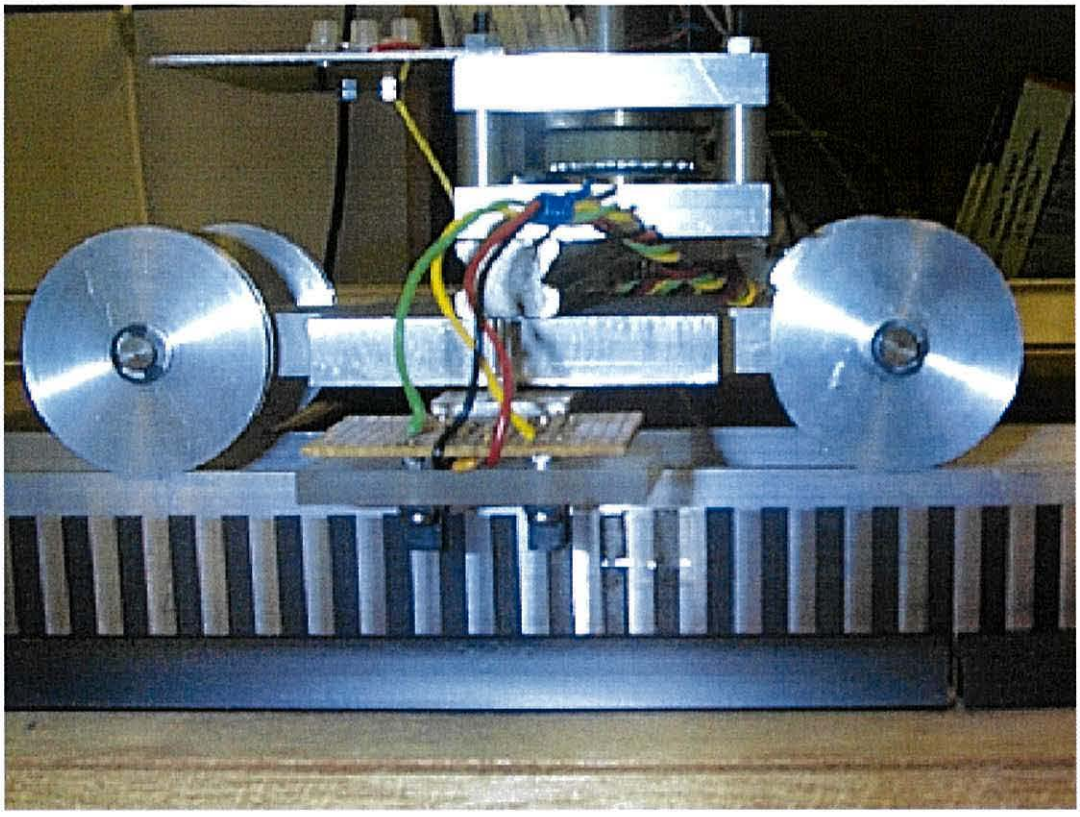


Figure 7.10: The position encoder sensor, the two detectors can be seen mounted over the acetate strips

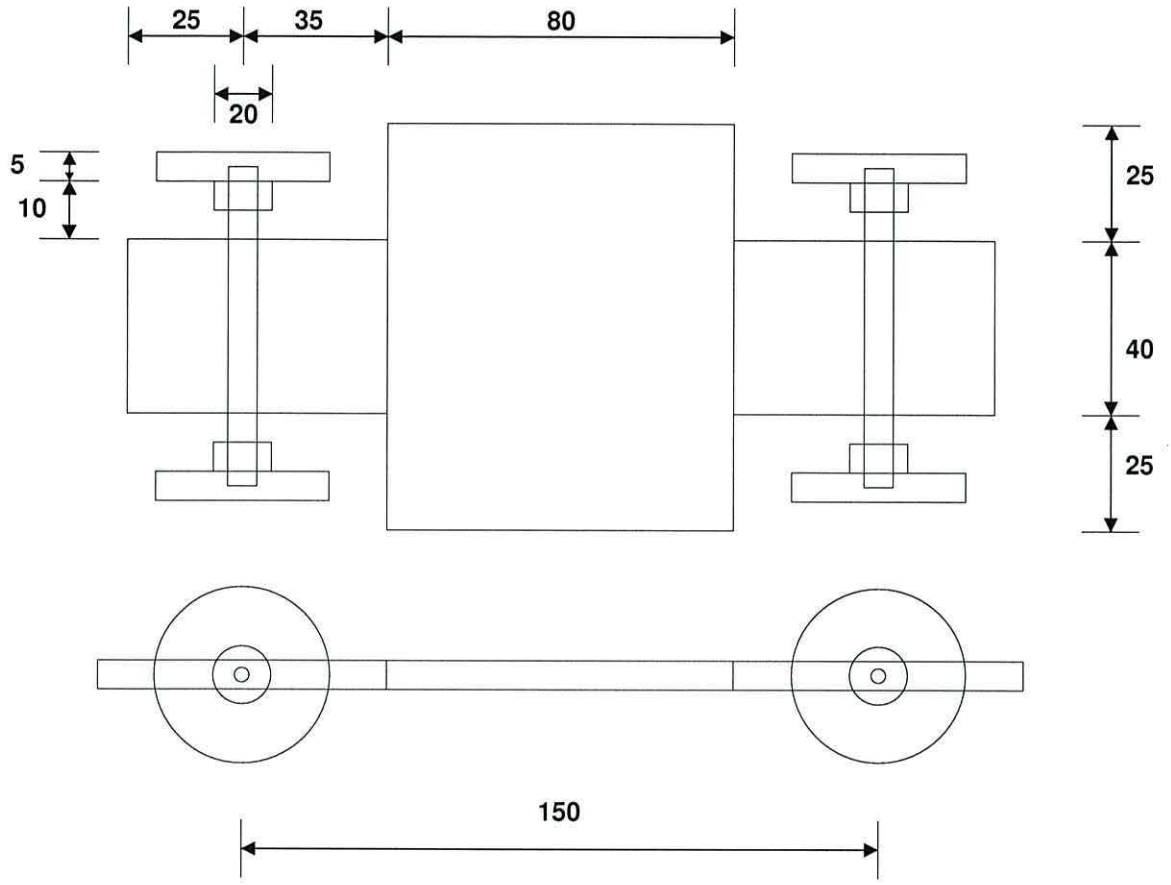


Figure 7.11: Mechanical design of the carriage

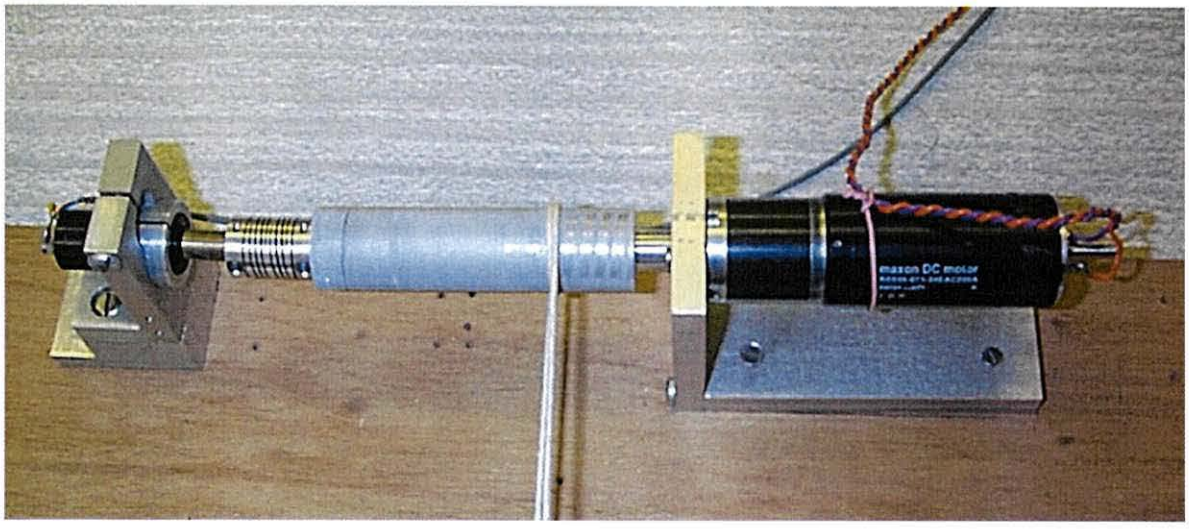


Figure 7.12: Drive motor, capstan and tachometer

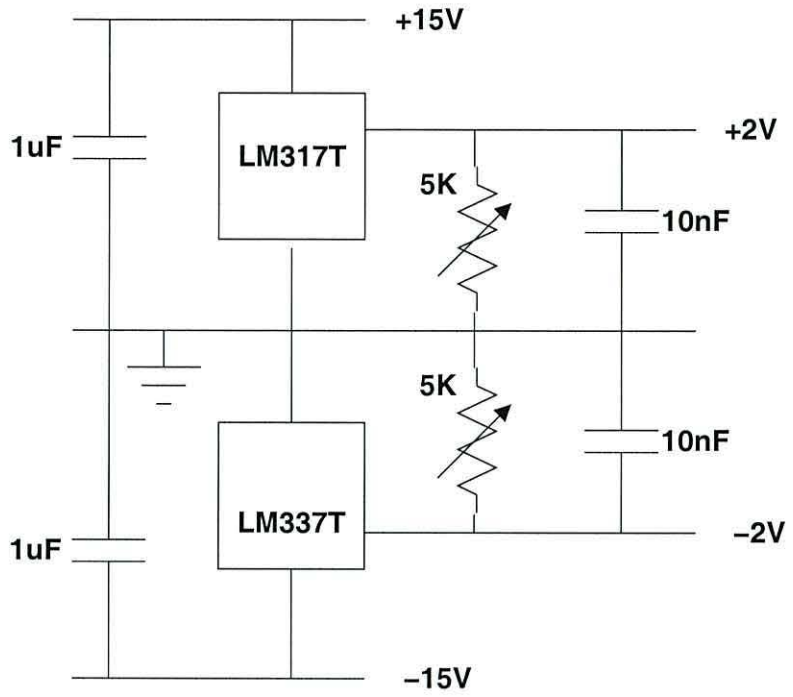


Figure 7.13: Potentiometer Wire Voltage Control Circuit

the position of the camera mount. In order not to load the potentiometer wire a high impedance amplifier is used to buffer the voltage and figure 7.14 shows the AD544 circuit used to do this.

The output from this circuit is fed into an analogue to digital converter circuit which encodes the voltage into an eight bit digital value which is input to the computer via a digital IO card.

The X position is set by the controller software running on the PC. The 8 bit value representing the required position is converted into an analogue voltage using the digital to analogue converter shown in figure 7.15. This circuit produces a voltage range between 0 and 5 volts, this must be re-biased to vary about zero volts so that the motor may be driven in both directions. A voltage reference circuit, also shown in figure 7.15, was built to achieve this. The two signals in this circuit are added together using a 741 operational amplifier.

Position feedback from the potentiometer wire is also used to locate the trolley accurately using a control circuit. Figure 7.16 shows the X direction controller circuit.

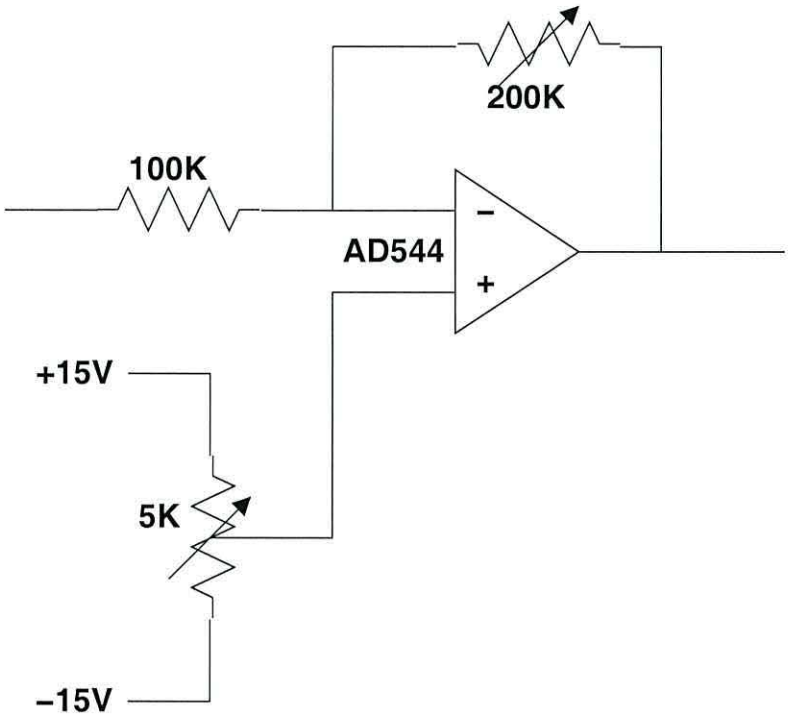


Figure 7.14: Buffer Amplifier

A counter is used to count the pulses from one detector, the quadrature detector controls the direction of count. The value stored in the counter is proportional to the position. Figure 7.17 shows a schematic of the Y direction position measurement circuit.

The trolley is pulled along in the Y direction at a fixed speed with feedback provided by a precision tacho generator. The computer can not control the position of the trolley in the Y direction, it is set manually using a potentiometer. Figure 7.18 shows the controller circuit.

These positioning circuits give a resolution of about 4mm in the X (over 1m) direction and 10mm in the Y direction (over 2.5m).

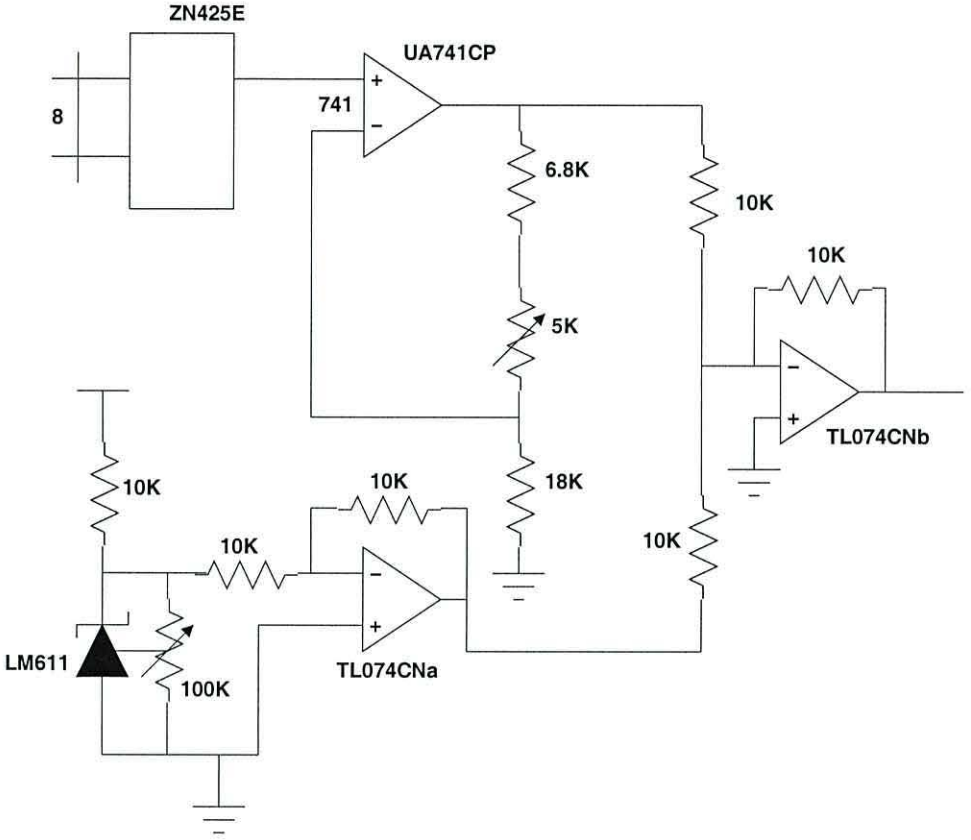


Figure 7.15: Digital to Analogue Conversion

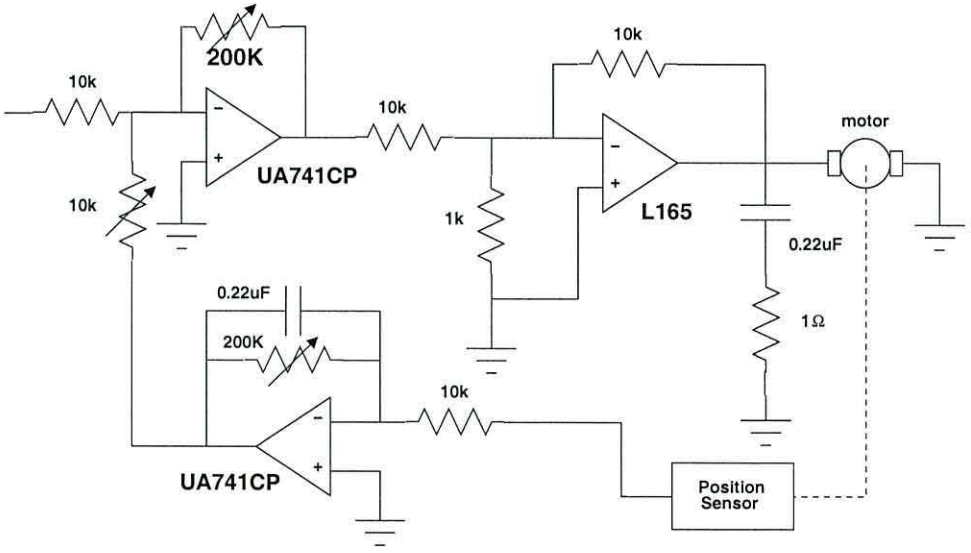


Figure 7.16: X Drive circuit diagram

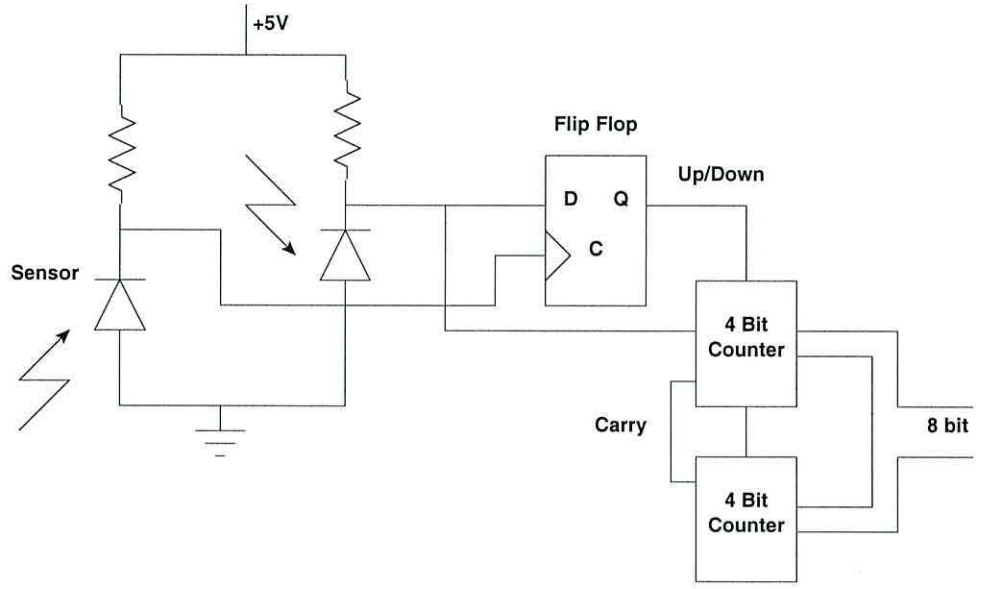


Figure 7.17: Y position measurement circuit

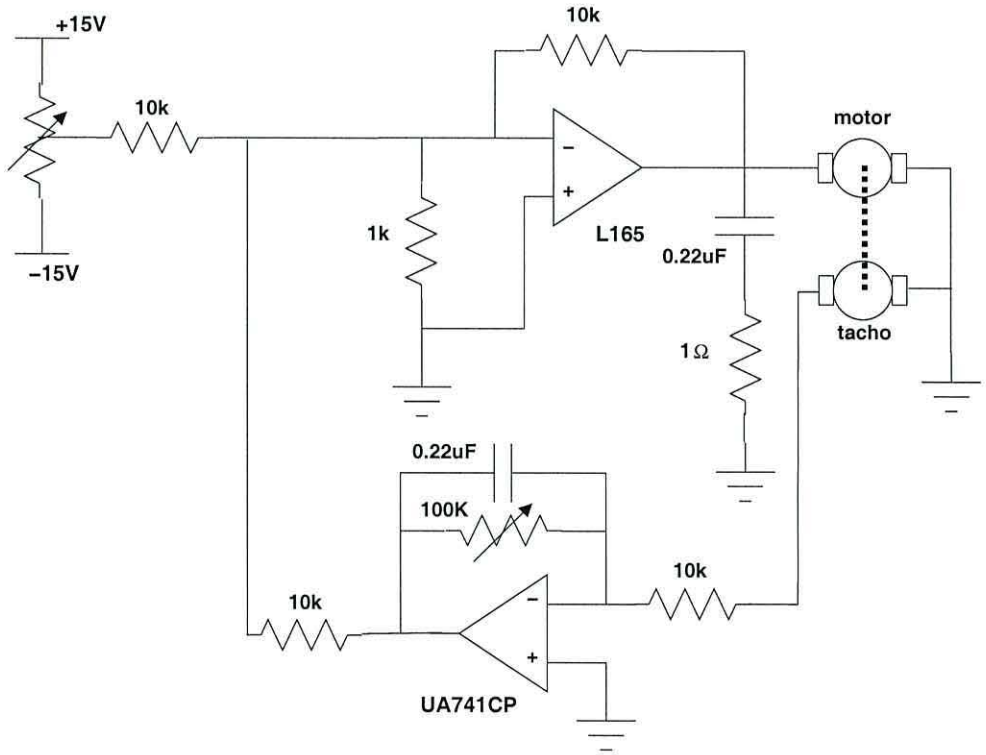


Figure 7.18: Y Drive circuit diagram

7.2 Controller Software

This section will outline the software design and implementation.

The main controller software was developed progressively during the course of the project. The major components of the software were developed separately from the main controller software. The software was implemented using an object-oriented aware programming language, but the actual design and the subsequent code uses both procedural and object-oriented methods.

7.2.1 Design

The organisation of the software can be seen in figure 7.19 which shows the main building blocks of the controller software. The *Planner* and *Navigator* have not been implemented fully but have been included in the software to allow for future development. The *Planner* function is called but only sets up some parameters that are used throughout the rest of the software. The *Navigator* also sets up parameters, but in addition produces the nominal Navigator's Path.

The *Pilot* is the main function. This coordinates the operation of the image acquisition, map building and path planning systems. The flow chart in figure 7.20 shows how the software operates.

The pilot function uses *objects* derived from *classes*. This is object-oriented terminology, a class is a container for data and functions that operate on that data. An object is simply one instance of a class. For example, in the class "road vehicle", one instance of that class could be a car another could be a bus.

The classes used in the pilot function include:

image This class is a container for the acquired image, but it also holds other information about the image.

queue This is a simple ring queue of limited length.

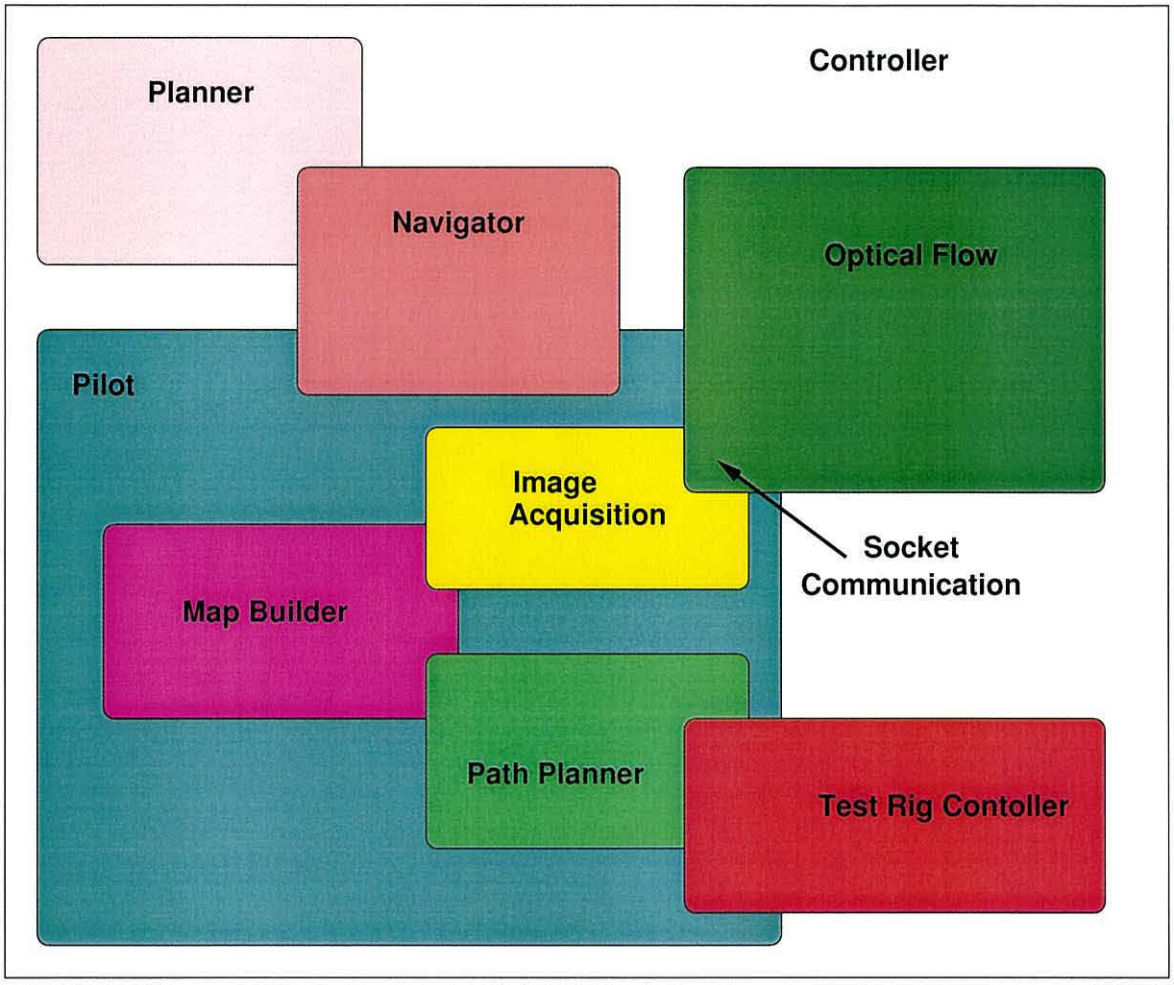


Figure 7.19: Design Layout

coordqueue This is derived from the queue class, but is suitable for storing coordinates.

nodequeue Again this is derived from the queue class, but this stores octree nodes.

imagequeue This a separate queue class for storing images

node A node is the building block of the octree structure

helicopter This class provides an interface to the test rig controller

Every time an image is acquired a new image object is instantiated to contain the image. This encapsulates the data with the functions that can operate on that data.

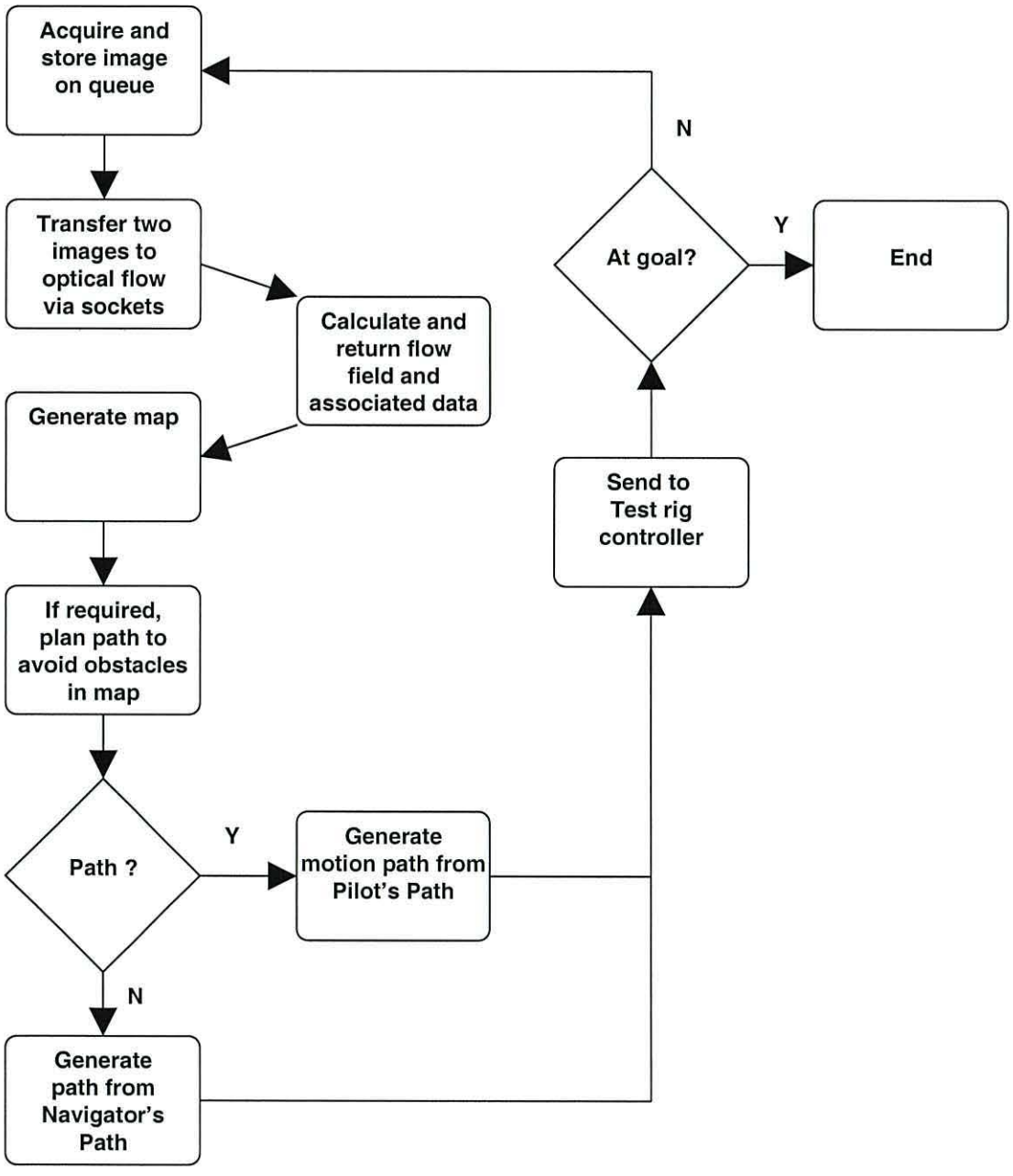


Figure 7.20: Flow chart of the pilot function

7.2.2 Implementation

Software was required to control both a video frame grabber and a digital IO card. This limited the operating system choice to Windows 95 as this was the only operating system, at the time, to have software drivers for both peripherals. Drivers could have been written for other operating systems, but this was seen as a distraction. The computer used to develop and run the software is a Pentium 166 with 64M of memory.

The software was written in C++ using the Borland C++ 5.01 design environment.

The optical flow operations were not performed on the same computer. This software ran on another PC running the FreeBSD Unix operating system. The decision to use two computers to control the test rig was borne out during experimentation. The test rig controller PC was not able to perform all the tasks within a acceptable time. A socket communication method was implemented which allowed the calculation of the optical flow and path planning to run in parallel. After each image was captured it was sent to the other machine for processing. While the optical flow was being calculated, the controller PC was free to continue to monitor the position of the test rig. This also allowed for the future implementation of the reflex avoidance system. When the optical flow had been calculated the flow fields were transfered to the controller PC for further processing.

The *planner*, *navigator* and *pilot* levels have been described in chapter 3. The software was developed from the bottom up, that is starting with the *pilot* level. This section will describe the implementation of the *pilot* level breaking down the procedure into defined tasks.

The flow chart given in figure 7.20 has been implemented in the following manner:

Acquisition The first task is to acquire an image. This is stored as an image object. The image object also contains the time the image was taken, the position of the camera when the image was taken and information about the camera that

took the picture. Storing data about the image with the image is useful as it allows further processing to be performed. Appendix D.1.4 gives the header file for the image code showing the different variables and functions. For example, one of the cameras on the test rig is mounted upside-down. This means that any image captured using it requires flipping before it can be used, the software knows about the position of the camera so this happens automatically. There is an option to then perform more image processing functions on the captured image. Functions to perform high-pass and low pass filtering, edge detection, normalisation and thresholding are all available in the software.

Optical flow generation Once two images have been acquired they are sent to the optical flow software and the flow field calculated. The computation time depends on the image size used, the parameters used within the optical flow software and of course the speed of the computer. Once the flow field has been calculated it is transferred back to the controller PC. There is an overhead in transferring two images and returning the flow field but this is justifiable because it allows the main controller software to continue performing other actions such as planning a path and controlling the test rig.

Map generation The returned flow field contains vectors representing the magnitude and direction of image motion. This is then used as described in chapter 4 to locate obstacles points that are close to the camera mount which are then added to the map. The map is stored as a matrix so that it is compatible with the distance transform software. It also allows the map to be saved to a file for subsequent processing or visualisation.

Path Planning Once a map exists, path planning may be needed. The operation of the path planner has been described in chapter 5. Appendix section D.2 gives highlights from the source code of the path planning software. The function *octree* given in section D.2.1 produces the unconnected graph. The functions *update1* and *update2* given in sections D.2.3 and D.2.4 generate the neighbourhood relationships. Section D.2.5 of Appendix D gives the source code for the distance

transform generation. The function *find path* in section D.2.6 then produces the node path.

7.3 Summary

During this project, much time and effort were expended on design and construction of the test rig but, once this was done, it has proved invaluable for testing both the machine vision system and the path planning system in a closed loop environment.

The rig was used to test individual sub-systems and the integrated system. The next chapter first presents an experiment to configure the optical flow software so that accurate distance measurements can be made. The chapter ends with results from the test rig showing obstacle detection and avoidance in action.

Chapter 8

Results

This chapter presents the results from the different experiments performed using the test rig. Results from the optical flow software are presented showing the effects of the variation of algorithm parameters. The optical flow software and the map building system are tested using the results from a "ground-truth" optical flow experiment, where the motion of the camera and obstacle positions are known. The aim of these experiments is to determine the best configuration of optical flow software and map building system. The chapter culminates with results from the test rig which show the obstacle avoidance system in operation.

8.1 Optical flow tests

This section describes the investigation of the optical flow software described in Chapter 4. In that chapter it will be recalled that Anandan's method was chosen to calculate the optical flow. This method has the key advantage that it always produces a full flow field, and that it works with two input images.

The results presented in here aim to show two things: firstly that Anandan's method does indeed compute the optical flow and secondly that, through the optimisation of the software parameters, the method is applicable to our problem.

8.1.1 Input Images

In order to test the optical flow system a set of "ground truth" images were obtained from the test rig. The camera was set in motion on a predefined path and a series of 72 raw images where captured. Figure 8.1 shows a selection of the captured images.

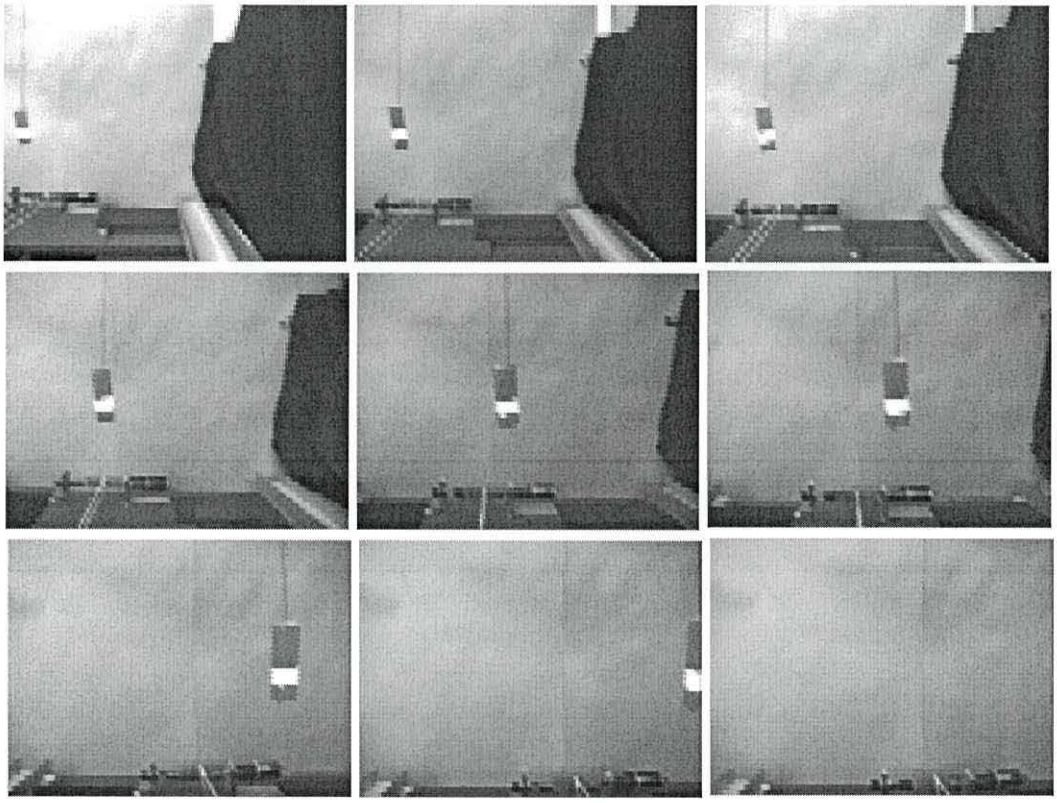


Figure 8.1: Images 0, 14, 20, 28, 37, 42, 50, 54 and 55

In figure 8.1, the object with the white band is an obstacle suspended in space (the string holding the obstacle can be seen) - this should be noted for future reference. In the first few images a black cloth can been seen on the right hand side which was used to hide a feature within the room. The "background" is made from two pieces of card; the join between these two can be seen in some of the images - again this should be noted for future reference.

The pulley mechanism can be seen near the bottom of all the images This is located at the same distance as the obstacle from the "background" in the y direction. This feature should be detected by the optical flow software.

Figure 8.2 shows the three pairs of images (27 and 28, 37 and 38, and 50 and 51) that where used in the optical flow tests described below. The images where captured at a resolution of 128×96 with 256 grey scales.

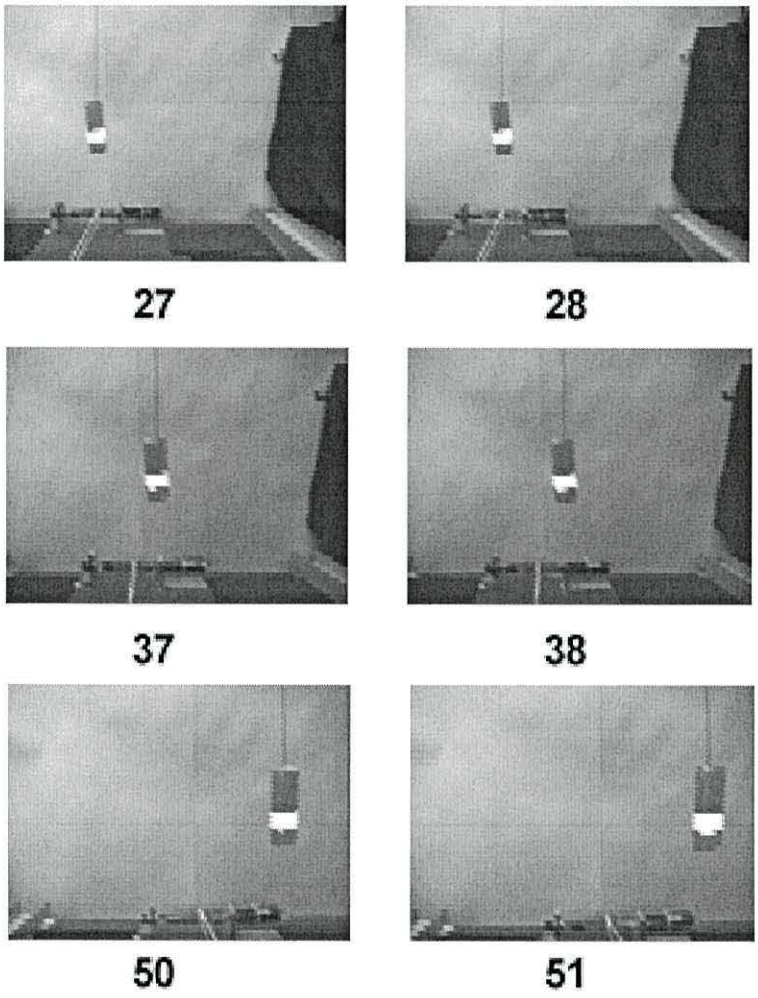


Figure 8.2: Test images

These pairs of images images were captured at three different camera location. The remainder of this section shows how the distance between the camera and obstacles can be estimated using optical flow.

8.1.2 Optical flow generation

There are a number of parameters that can be adjusted when using Anandan's method. These are:

- The number of hierarchical levels (L) which can range from 1 to 4, the default is 1. This gives the number of resolution levels that the flow field is generated from.
- The size of the correlation window (W) can be either 3,5 or 7, the default is 3. This gives the size of the mask used in pixel matching.
- The number of relaxation iterations (I) which can range between 1 and N, the default being 10. This controls how much smoothing is performed on the flow field.

Default values

In order to determine the optimum parameter values for this application, the flow field was calculated for the values of L =1-4, W=3,5, and 7 and I=1,10,20. As a benchmark the flow fields were calculated using the default values first. The flow field figures presented in this chapter have been calculated from the computed flow field values using a software utility. This generates a vector representation where the length of the arrow is proportional to the magnitude of the flow field and the arrow indicates the direction of motion on the image plane.

The images were captured at a resolution of 128×96. In all the figures given below the flow fields have been sub-sampled by a factor of two, and the magnitude has been doubled - this gives the best reproduction on paper.

Figure 8.3 shows the optical flow field generated from images 27 and 28 using the default values. There are three distinct areas in this flow field. At the bottom the motion relative to the pulley mechanism can be seen. On the right the interface between the background and the black cloth produces distinctive motion vectors. Finally, there is a flow associated with the obstacle in the upper left hand corner,

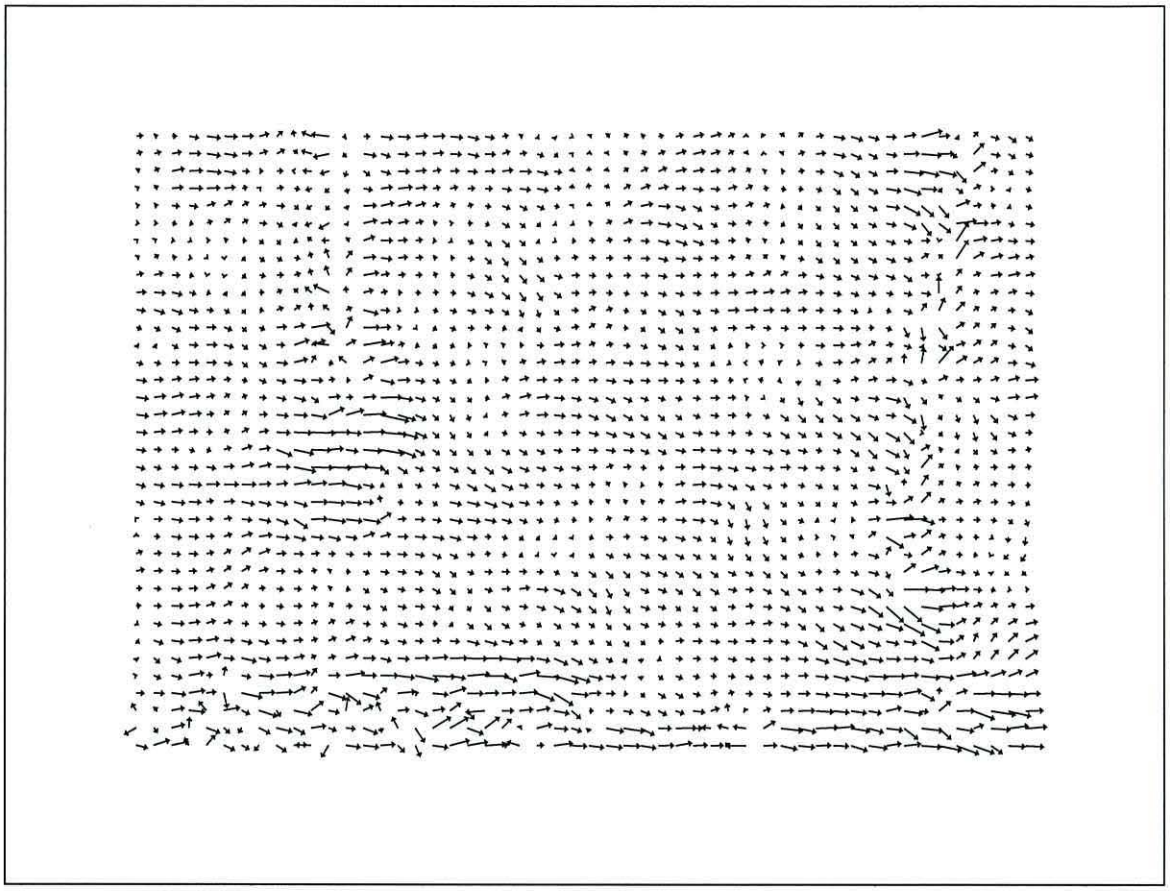


Figure 8.3: Flow field Images 27 28 (w=3 l=1 i=10)

demonstrating that the obstacle has been detected. On closer examination, the vectors can be seen to point in one general direction indicating flow to the right (meaning that the motion of the camera was to the left). Note that the string holding the obstacle has also been detected.

Figure 8.4 shows the optical flow field generated from images 37 and 38 using the default values. There are now only two main areas of flow. Again there is a flow at the bottom due to the pulley mechanism. The obstacle and string are now in the centre of the image which can be clearly seen in figure 8.4. The flow vectors are still showing motion towards the right.

Figure 8.5 shows the optical flow field generated from images 50 and 51 using the default values. There is now only one area of distinct flow, at the right hand corner of the flow field which is associated with the obstacle. The direction of flow is confused but the obstacle has been detected. Notice in figure 8.2 that the

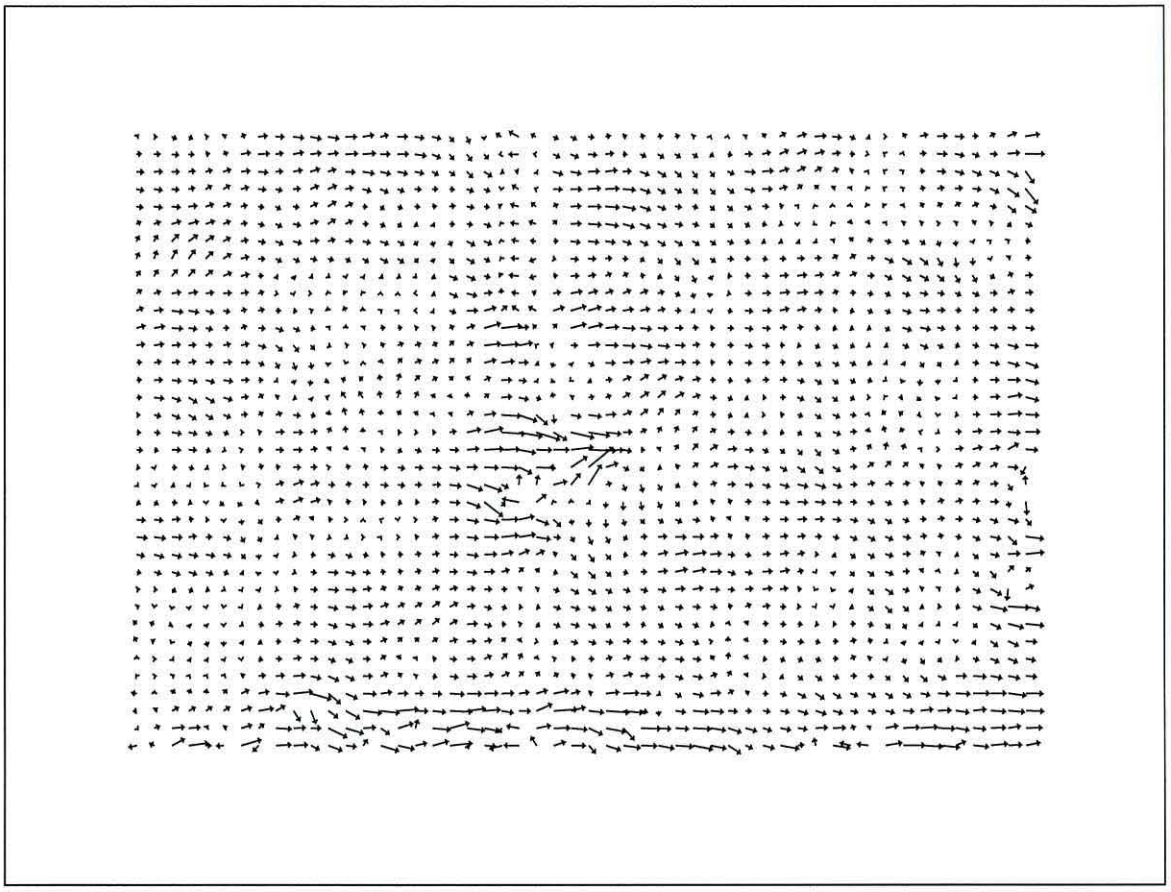


Figure 8.4: Flow field Images 37 38 (w=3 l=1 i=10)

pulley mechanism is present in images 50 and 51 but this feature is not present in the flow fields presented in figure 8.5. This is due to "edge-effects". The optical flow at the edge will always be difficult to compute as the matching function will not be able to find corresponding points between images.

Parameter Variation

The default values used to produce the optical flow fields presented in figures 8.3, 8.4 and 8.5 have shown that obstacles can be detected and that obstacles closer to the camera produce larger flow fields. However, figure 8.5 is disappointing because, although the obstacle is close to the camera, the flow field is small and indistinct. This section describes the effects of varying the software parameters, and shows that near-optimum values can be selected by inspection of the flow field. However, an exhaustive search is not possible because of the large number

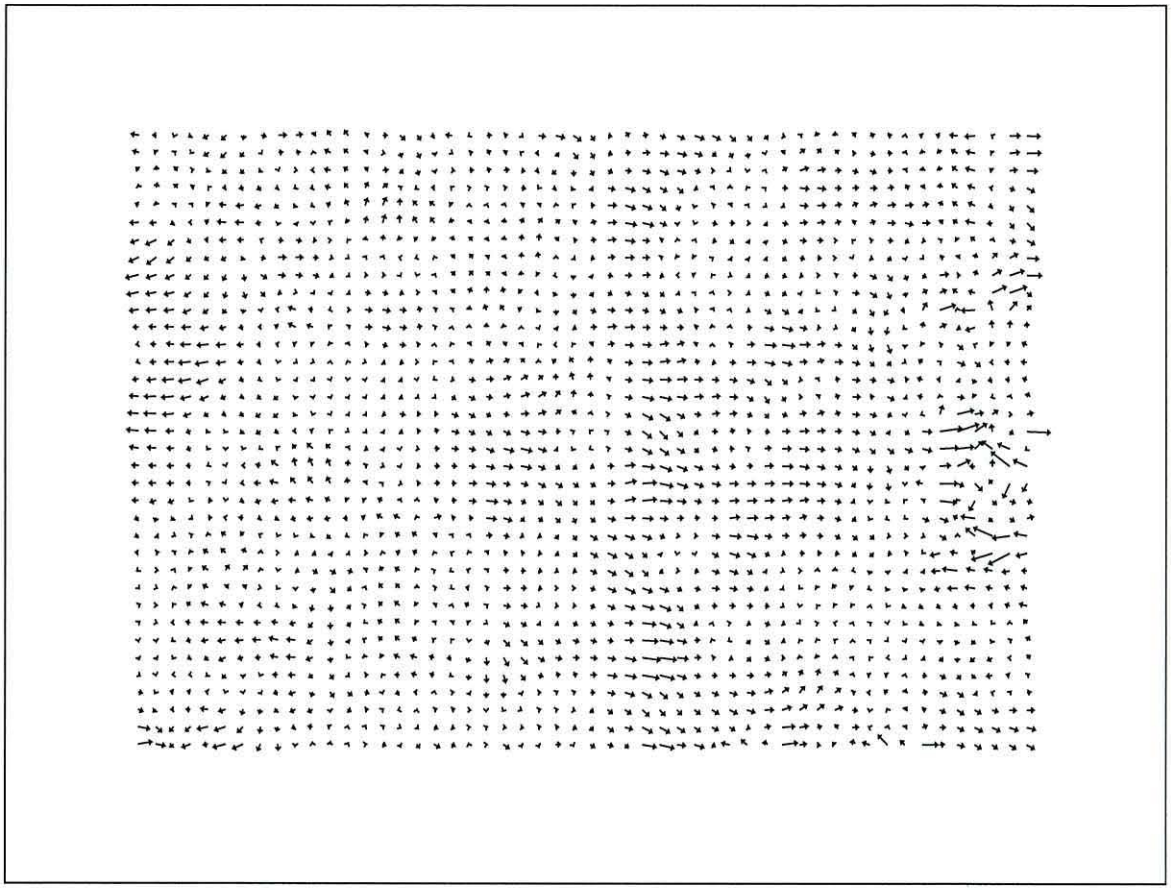


Figure 8.5: Flow field Images 50 51 (w=3 l=1 i=10)

of possible combinations of parameters. It should be noted that the effect of increasing the values of the parameters also increases the computation time, so there is a trade-off between flow field quality and processing time.

I=1 The effects of setting I=1 can be seen in figure 8.6. The figure show the flow fields for the "low" settings W=3 and L=1, and the "high" settings of W=7 and L=4. These show that, although some structure can be seen in the flow fields, even to the human eye (which is adept at pattern recognition), there is very little information contained within the flow fields. This result shows that a low relaxation interval produces unusable results. Further experiments confirmed that acceptable results could not be produced with a value of I of less than the default value of 10.

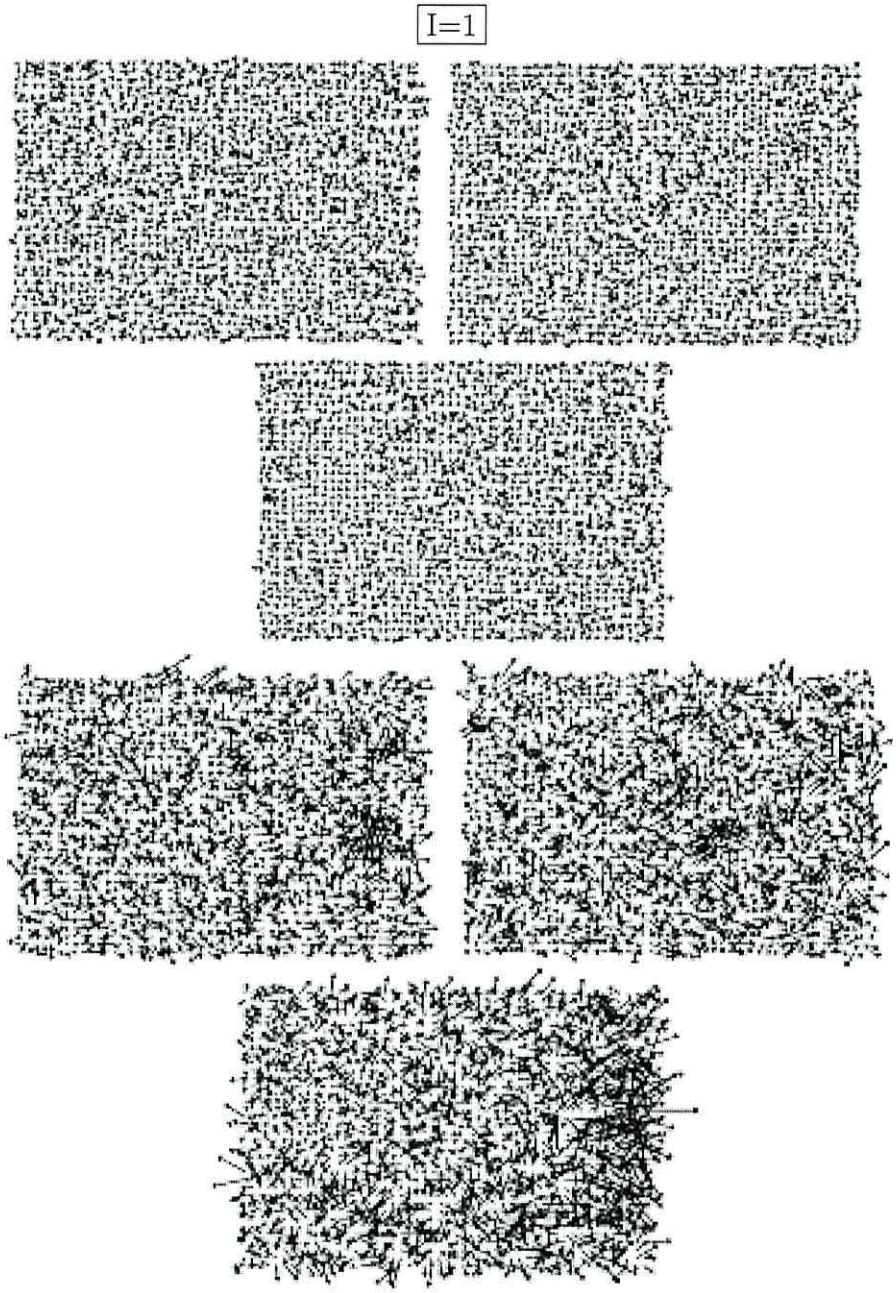


Figure 8.6: Flow field Images when I=1 - Top three images are number 27/28 37/38 50/51 for W=1 L=1 I=1 - Bottom three images are number 27/28 37/38 50/51 for W=7 L=4 I=1

I=10 Figures 8.7 to 8.12 show the results from the optical flow software when I=10.

Figures 8.7 and 8.8 shows the results produced using the first set of input images. The top four flow fields in figure 8.7 show how L=1,2,3 and 4 effect the flow field with W=3. It can be seen that as L increases the objects in the flow fields become more distinct. The lower four flow fields shows the same variation of L but here the correlation window size (W) is 5. The effect of increasing the correlation window size is to smooth the detail in the flow fields. In latter four flow fields L does not have much affect the resulting flow. This is because the smoothing effect of the correlation window has reduced the "higher" spatial frequencies during the matching between the two images. When W=7, figure 8.8 shows that the smoothing effect of the correlation window makes the result almost independent of the variation of L and very little detail is visible in the flow fields.

The same trends can also be seen in figures 8.9 and 8.10 for the second set of input images, and in figures 8.11 and 8.12 for the final set.

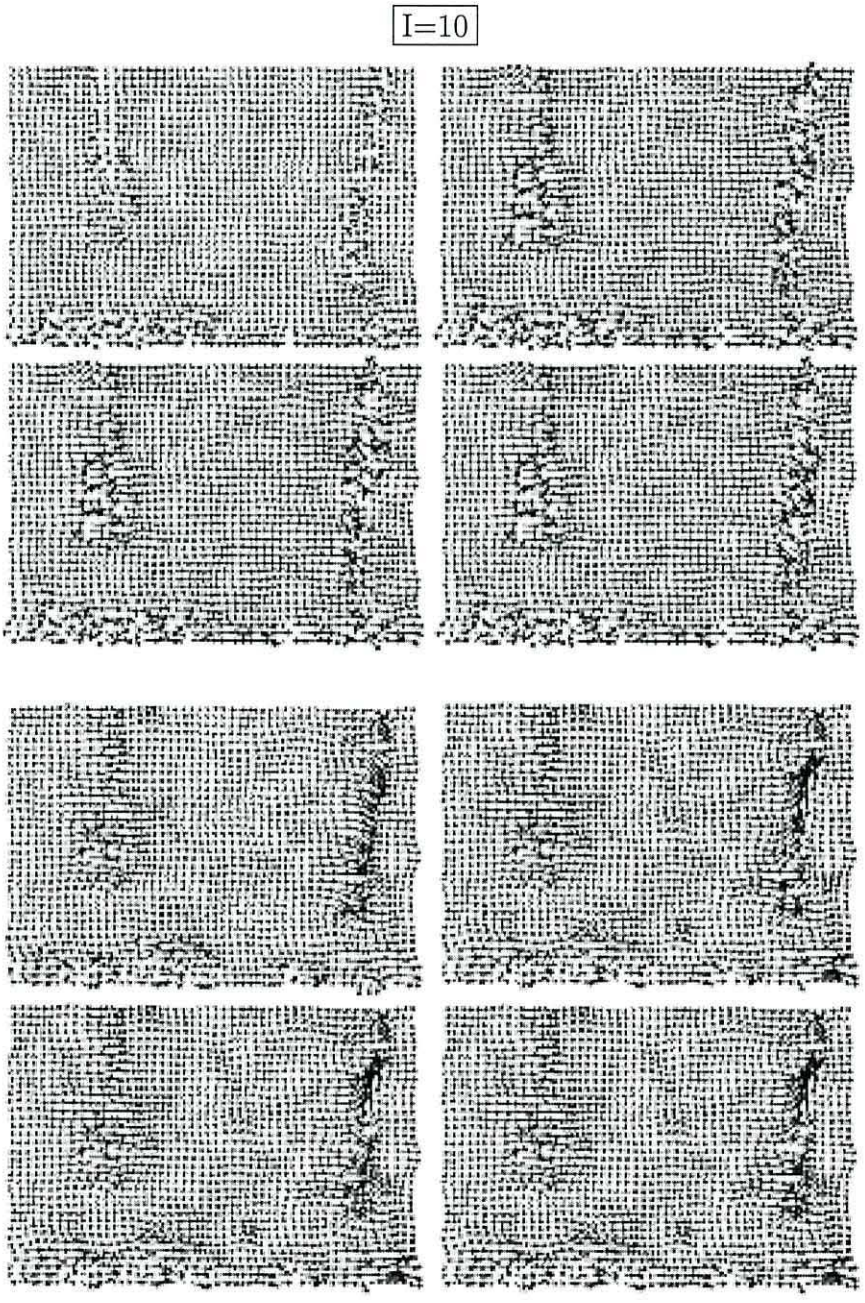


Figure 8.7: Flow field Images 27 28 - Top four w=3 l=1,2,3,4 - Bottom four w=5 l=1,2,3,4

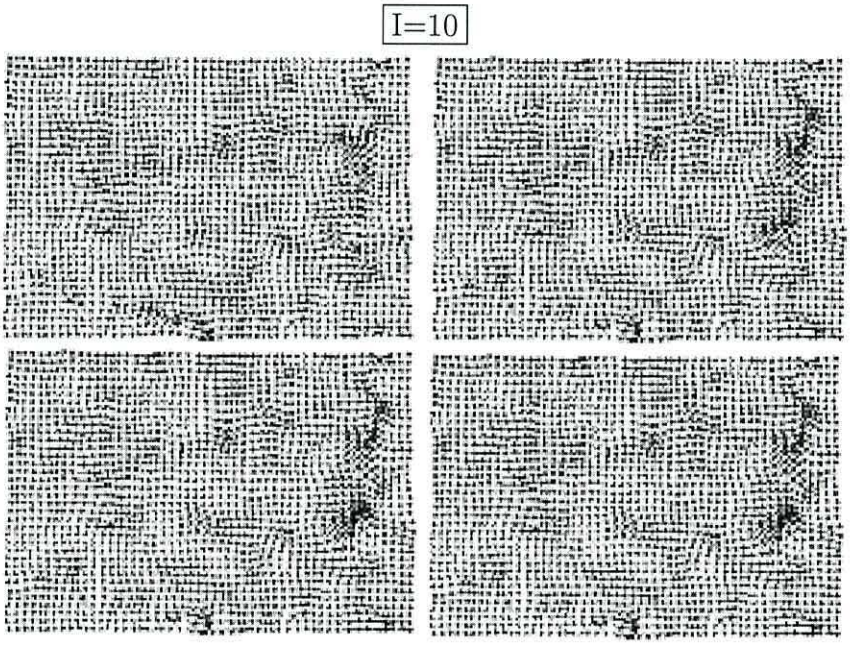


Figure 8.8: Flow field Images 27 28 - w=7 l=1,2,3,4

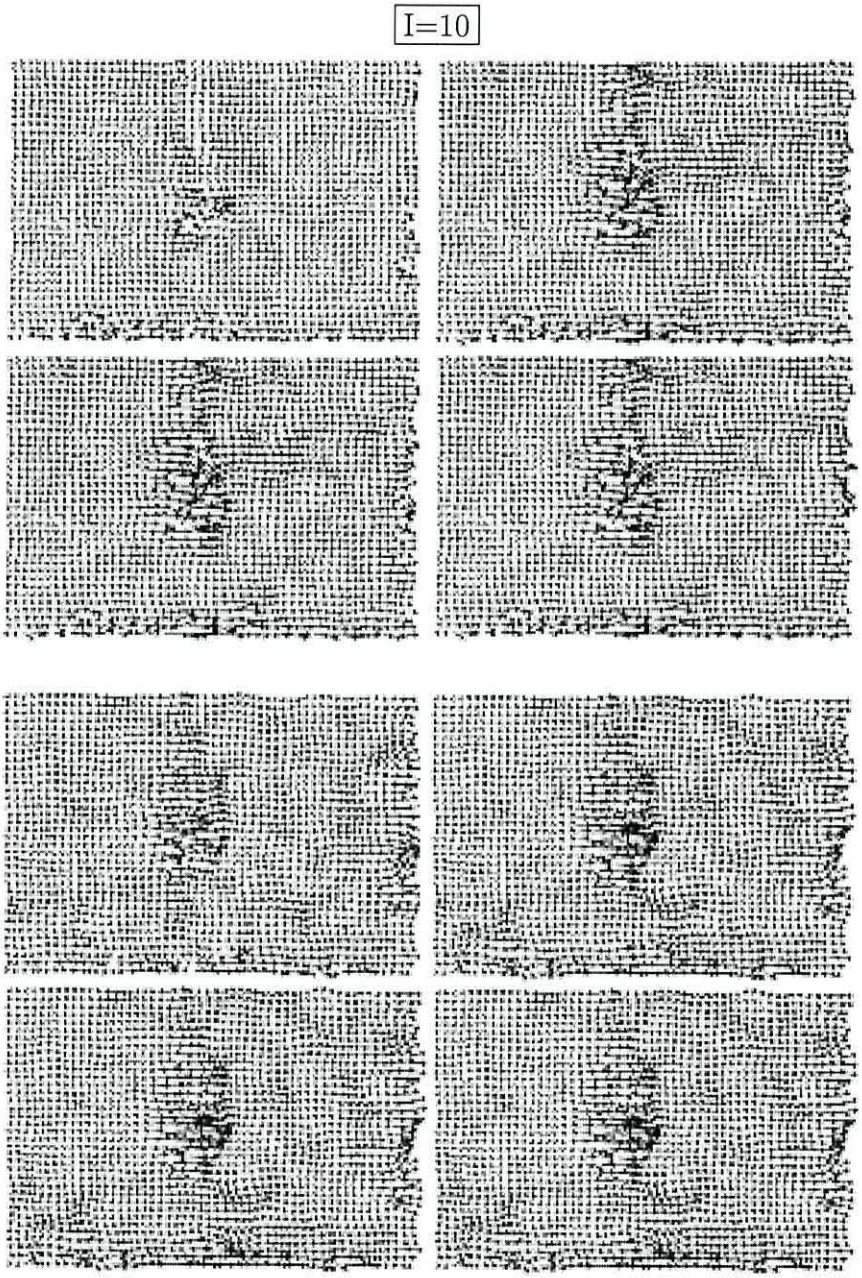


Figure 8.9: Flow field Images 37 38 - Top four w=3 l=1,2,3,4 - Bottom four w=5 l=1,2,3,4

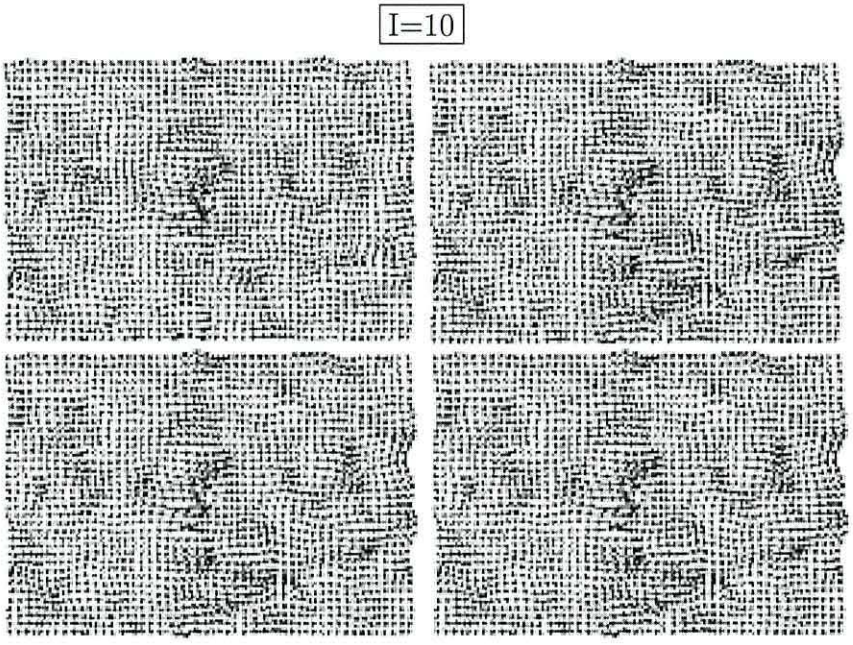


Figure 8.10: Flow field Images 37 38 - w=7 l=1,2,3,4

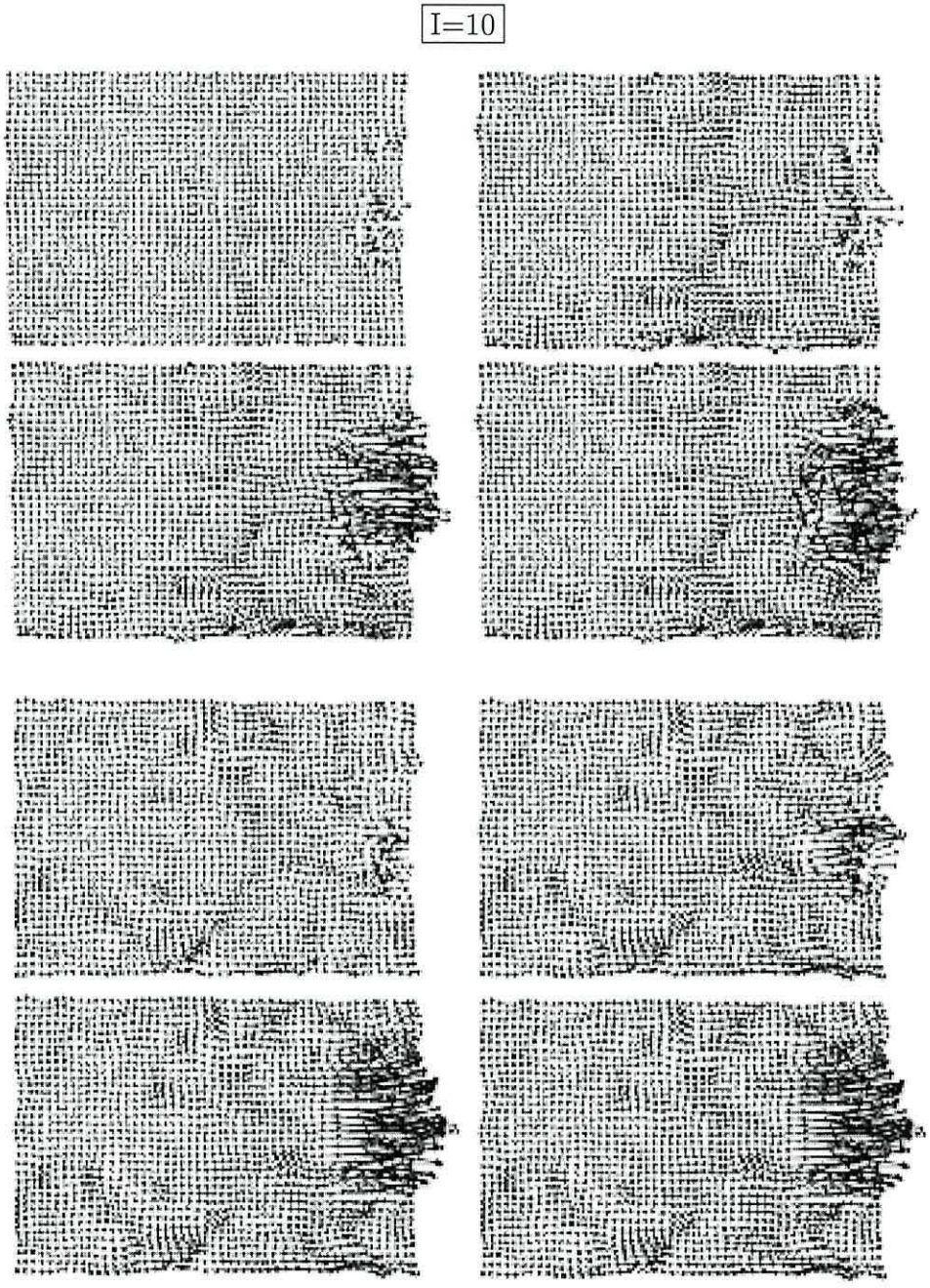


Figure 8.11: Flow field Images 50 51 - Top four w=3 l=1,2,3,4 - Bottom four w=5 l=1,2,3,4

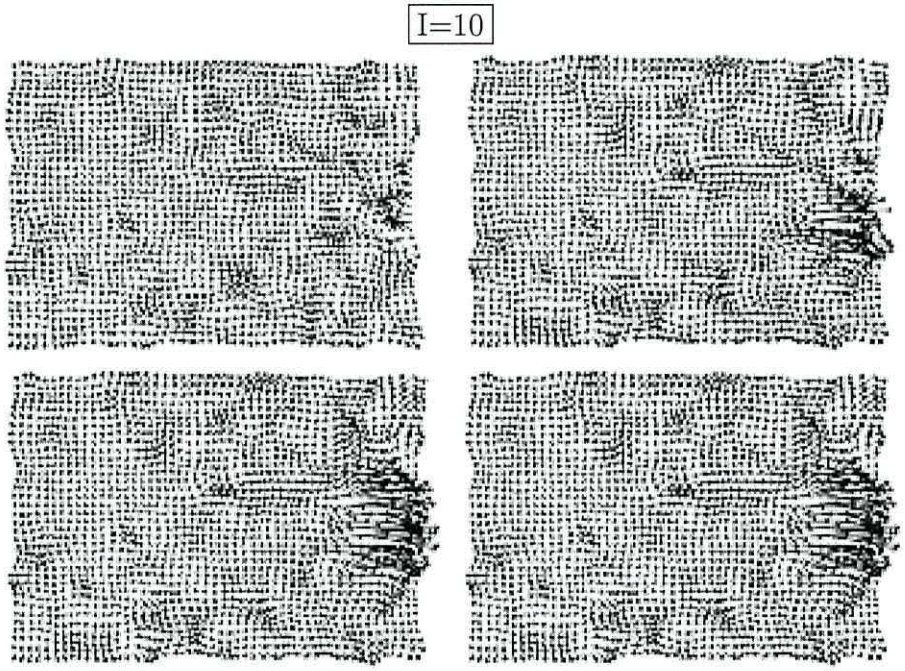


Figure 8.12: Flow field Images 50 51 w=7 l=1,2,3,4

W	L	I
3/4	3/5	10

Table 8.1: Table of selected parameters

I=20 Figure 8.13 shows the flow fields when I is set to 20. The flow fields here are similar to those were I=10, although more uniform as background "noise" has been reduced. However, the processing time is much longer. The effect of varying both L and W was investigated and the trend was found to be consistent. Only results from "low" and "high" values of L and W are presented in figure 8.13.

Selection of parameters to set in Anandan's Method

The investigation of the optical flow parameters have deduced that the parameters that should be selected for further investigation should be:

L Setting L to 3 or 4 produced a flow field in which the object stands out clearly from the background and are more distinct than when L=1 or 2. Values of L greater than 4 are prohibited by the current version of the software and in any case would produce very long computation times.

W W=7 has been discounted as it increases the processing time, and reduces the amount of information contained in the flow fields irrespective of the settings of the other values. Values other than 3,5 and 7 are not supported by the software

I Experiments have been performed for various values of I. Low values of I produced unusable results, although the calculation time was reduced. Values higher than the default of 10 produce good results but the processing time is greater. The default value process good results with an acceptable processing overhead.

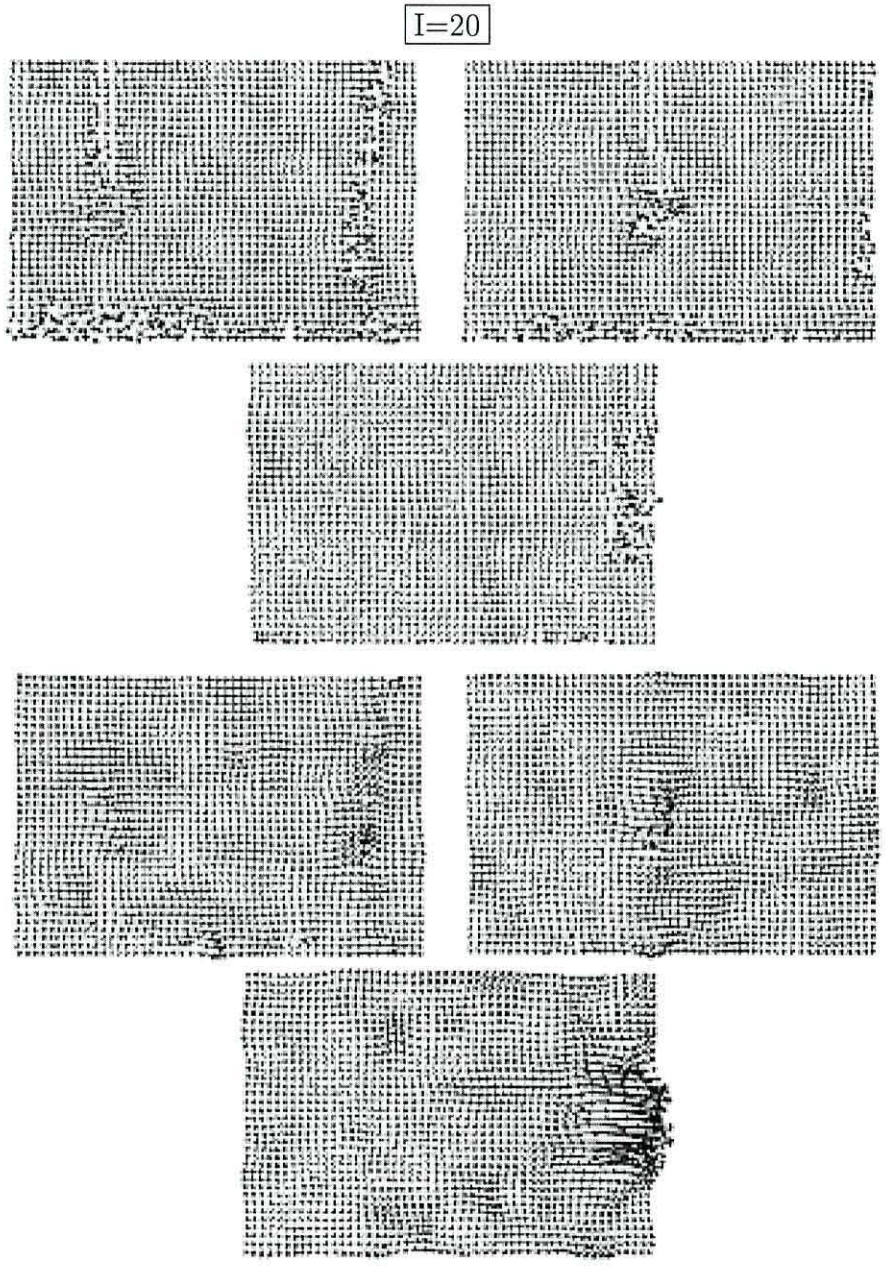


Figure 8.13: Flow field Images when I=20 - Top three images are number 27/28 37/38 50/51 for W=1 L=1 I=20 - Bottom three images are number 27/28 37/38 50/51 for W=7 L=4 I=20

8.1.3 Depth from flow vectors

Once the optical flow has been calculated a method is needed to convert from a measure of flow to a measure of distance. This section shows how this is achieved and selects the parameters to be used within the optical flow software of the test rig.

The range along the optical axis to the obstacles in the three sample images is known. In images 27 and 28, the obstacle is about 650mm away from the camera, in images 37 and 38 it is 550mm distant and in the final set of images it is 350mm away. The flow software has to estimate these ranges. This is achieved by tuning the optical flow software to minimise the difference between the estimated and known distances.

Flow field magnitudes

A visualisation of the depth information in the optical flow fields can be generated by calculating the magnitude of the flow vector at each point in the flow field:

$$f = \sqrt{u^2 + v^2} \tag{8.1}$$

where u and v are the components of the flow field in the vertical and horizontal directions with reference to the captured image. This can then be presented as a relief map. Figures 8.14 to 8.16 show the flow field magnitudes for the case where W=3, L=4 and I=10. The figures are colour coded, blue for low magnitudes to red for high magnitudes.

Figure 8.14 shows the flow field magnitude generated from the first pair of input images. There are three areas of flow, at the bottom of the figure due to the detection of the test rig, to the right caused by the black cloth. The final area of flow is due to the obstacle. Figures 8.15 and 8.16 show the flow field magnitudes for the second and third pairs of images, again the features seen in the flow fields presented above are present - especially the obstacle which can be seen moving

from left to right and coming closer to the camera.

These figures give the idea of depth, areas of higher values of flow are closer to the camera, and as the objects get closer to the camera the flow field magnitudes increase.

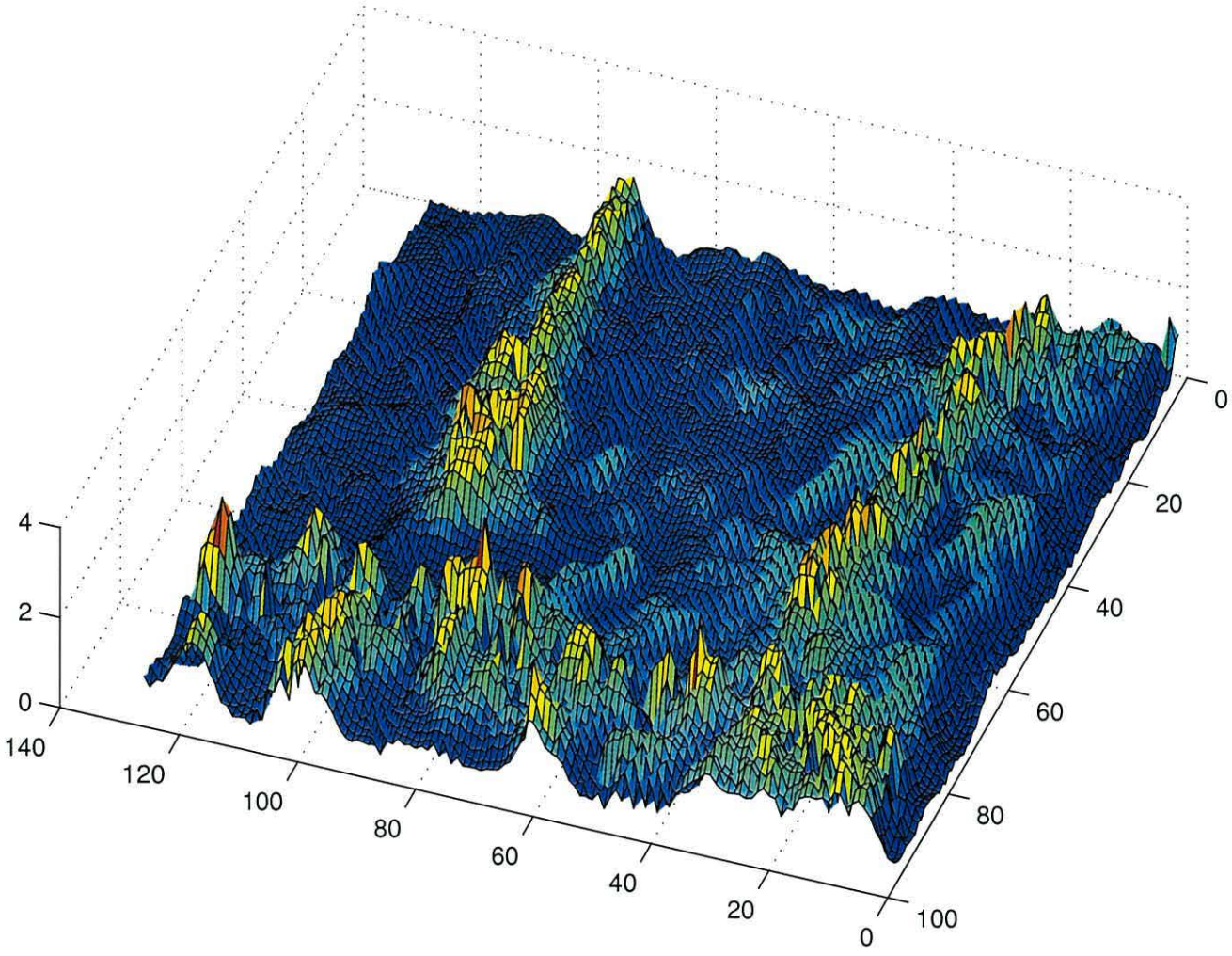


Figure 8.14: Optical flow magnitudes for images 27 28 where W=3 L=4 I=10

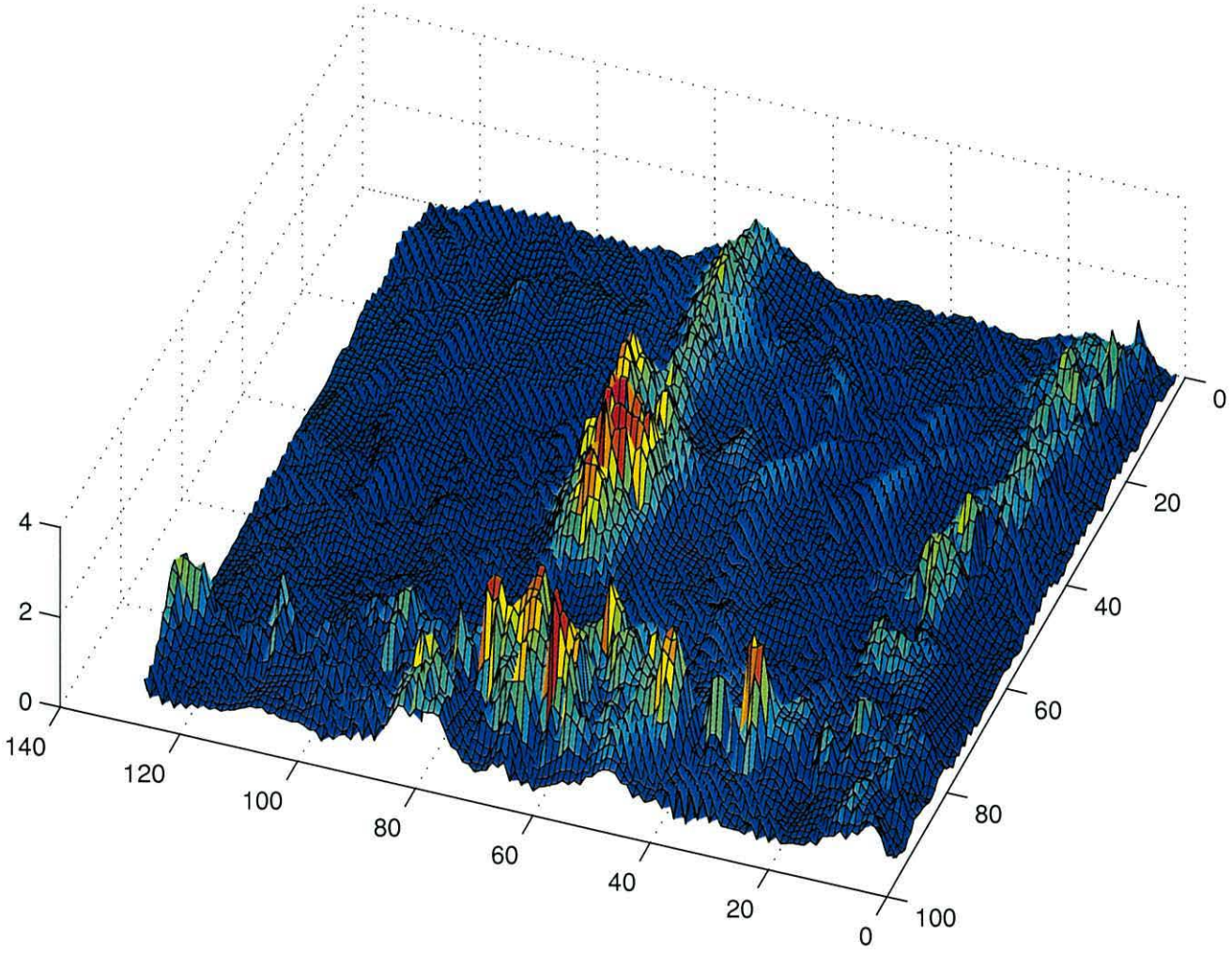


Figure 8.15: Optical flow magnitudes for images 37 38 where W=3 L=4 I=10

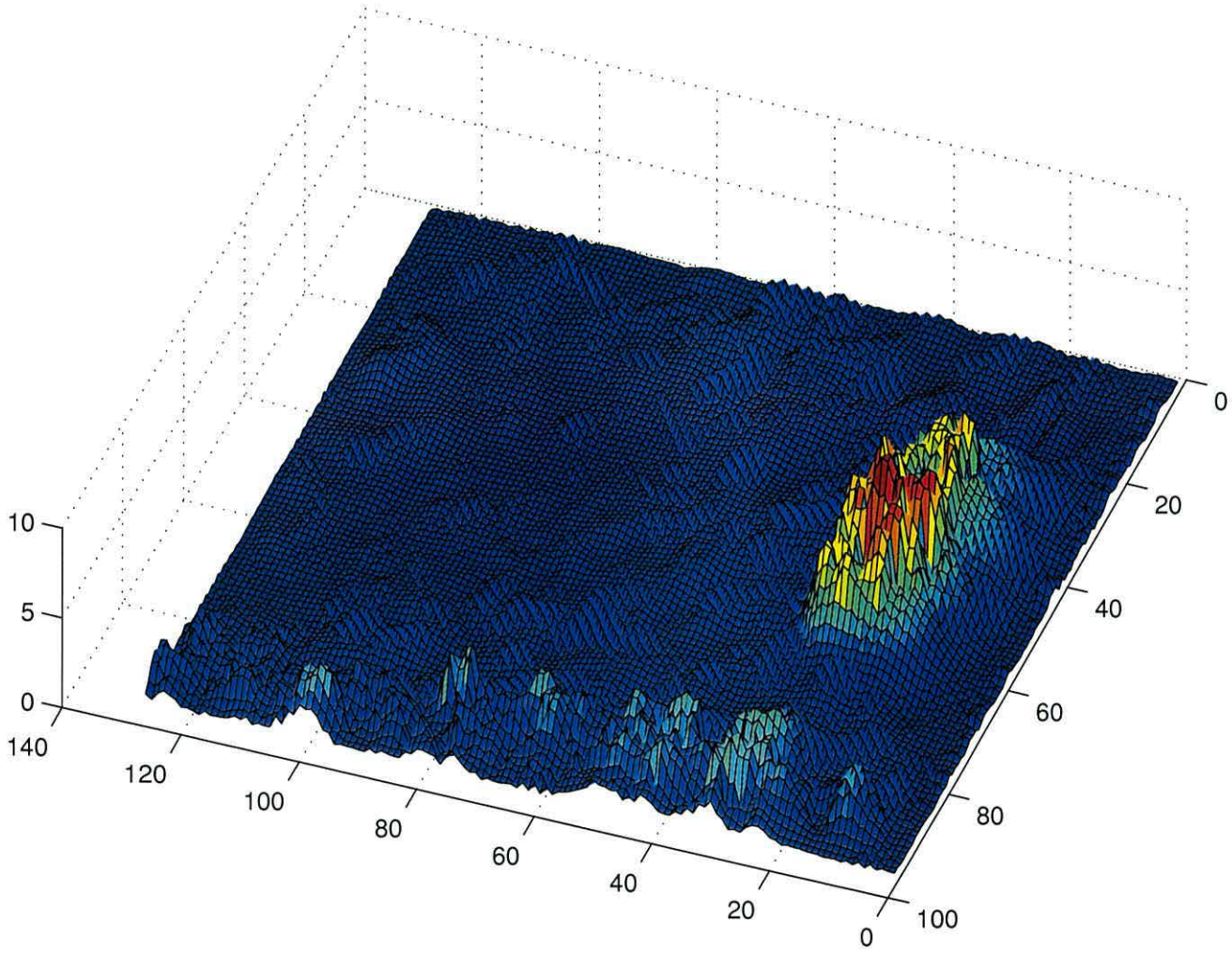


Figure 8.16: Optical flow magnitudes for images 50 51 where W=3 L=4 I=10

W	L	I	Images 27-28	Images 37-38	Images 50-51
3	3	10	-3758	-3528	-3377
3	4	10	-3529	-3526	-3540
5	3	10	-5126	-3769	-3573
5	4	10	-5126	-3769	-3573

Table 8.2: Calibration Values (k)

Calibration value determination

In order to convert from the magnitude of the flow field to a depth map, a calibration value (k) is required. This was calculated as follows:

$$k = D_n \times (1 + MAX(f)) \tag{8.2}$$

where D_n is the known distance from the camera to the obstacle in image n and f is the flow field magnitude. The function $MAX()$ determines the maximum magnitude of the flow field. The value of k was calculated for the flow fields with optical flow parameters W=[3,5], L=[3,4] and I=10. Table 8.2 details the results.

Note that equation 8.2 assumes that the maximum value of the flow field magnitude is associated with the obstacle.

Table 8.2 shows that the best estimate of distance occurs when W=3, L=4 and I=10, as the difference between the calibration values is the smallest.

The depth maps are calculated using the following expression:

$$d = \frac{k}{(1 + f)} \tag{8.3}$$

where d is the depth field and $k = -3500$. Figures 8.17, 8.18 and 8.19 show the depth maps for the optimal optical flow parameters.

Although the shape of the obstacle is clearly discernible in the figures, it is difficult

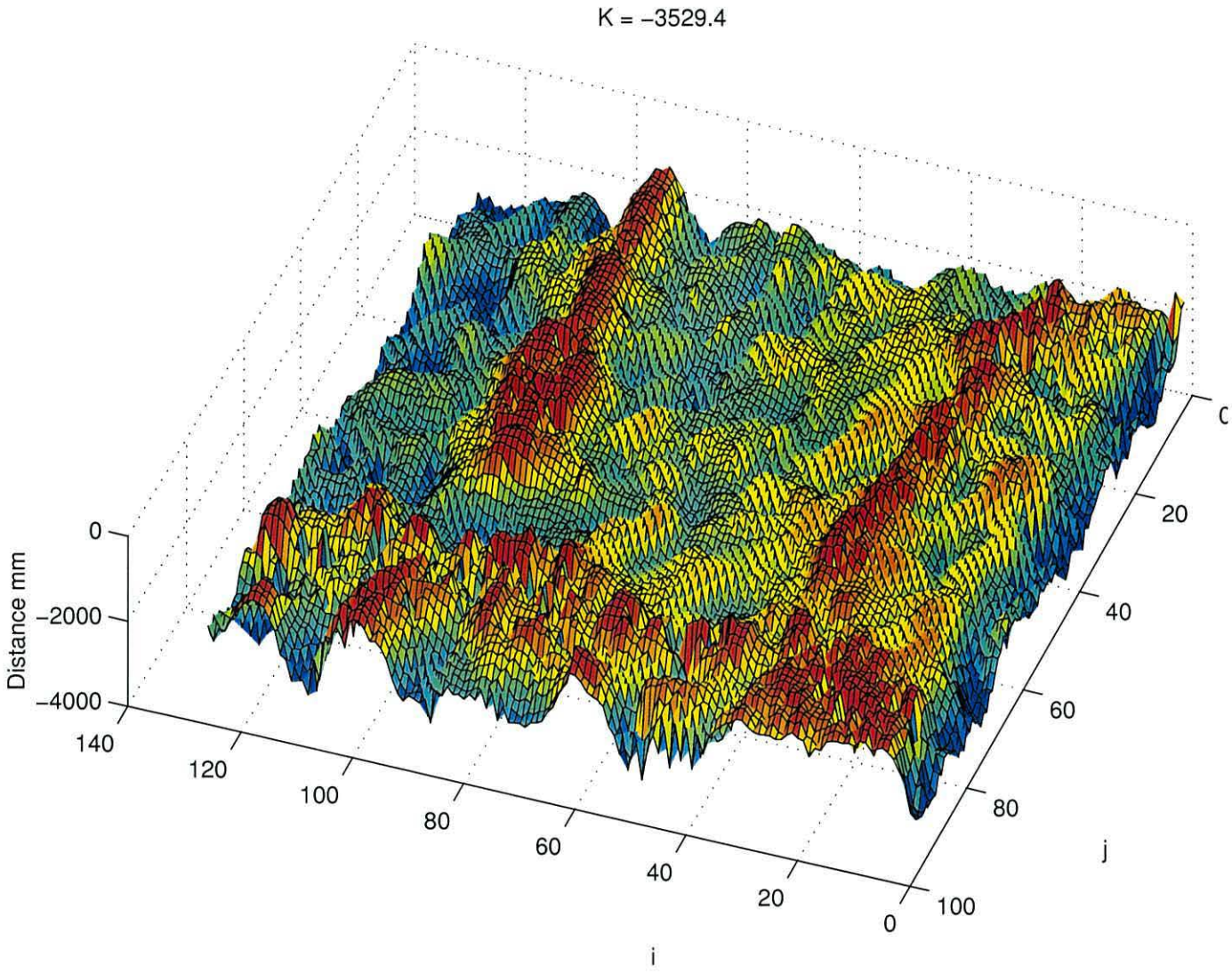


Figure 8.17: Range map for images 27 28 where W=3 L=4 I=10

to see the variation of depth. Figures 8.20, 8.21 and 8.22 show cross-sections through the depth maps of figures 8.17, 8.18 and 8.19 respectively. The cross-sections are made every 10 "pixels" (i=10,20...120) from right to left. The red line in the figures shows the 800mm range, ie. the distance at which the obstacle is entered into the local map. It can be seen that the obstacle is only within this range in figure 8.22.

The investigation of Anandan's method for determining optical flow has resulted in a set of parameters that give the best depth estimation from a set of "ground truth" images. This is only the first part of the obstacle detection and location,

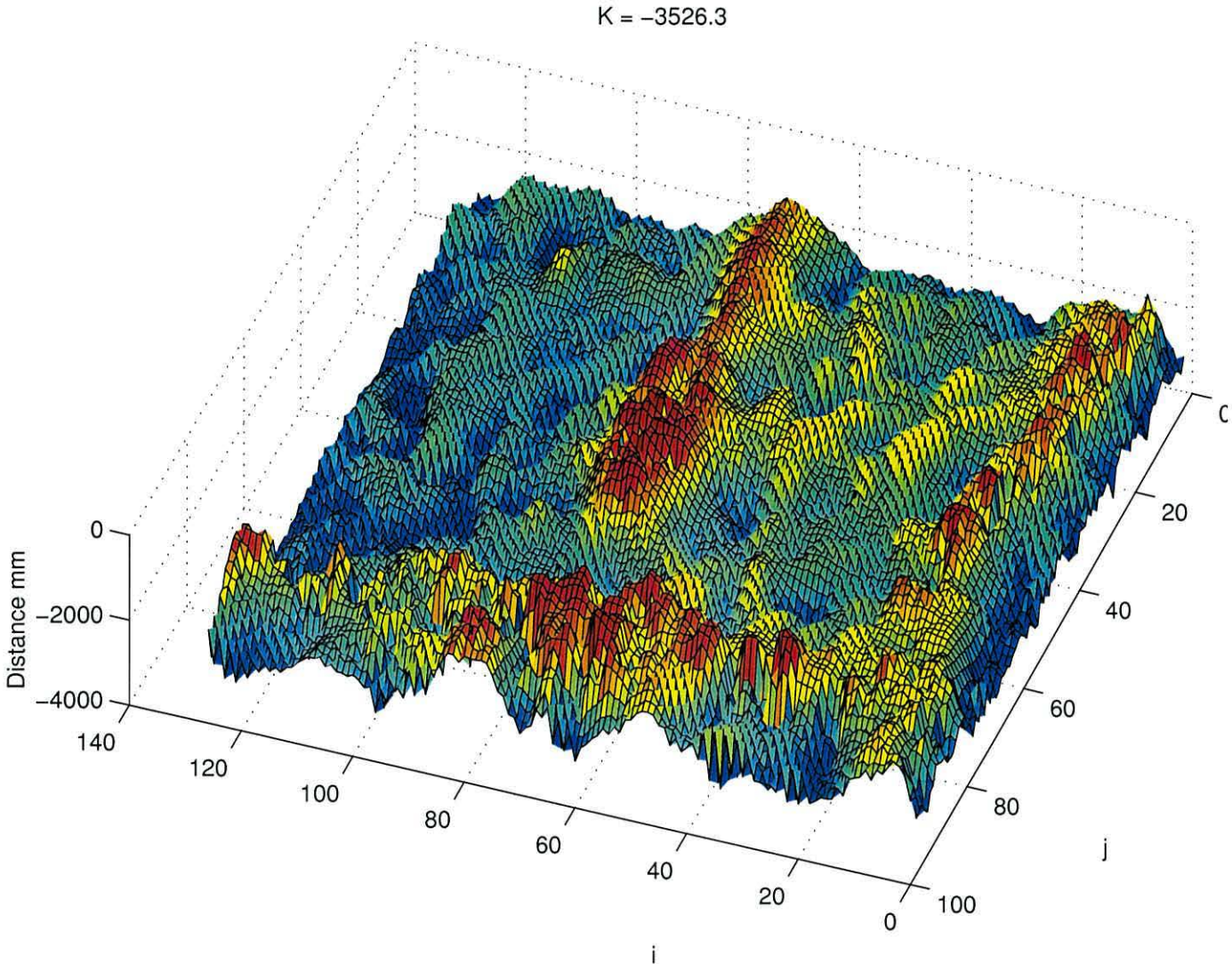


Figure 8.18: Range map for images 37 38 where W=3 L=4 I=10

the next step being to take the depth maps generated using the optimal parameters and convert them into a map that gives the positions of the obstacles.

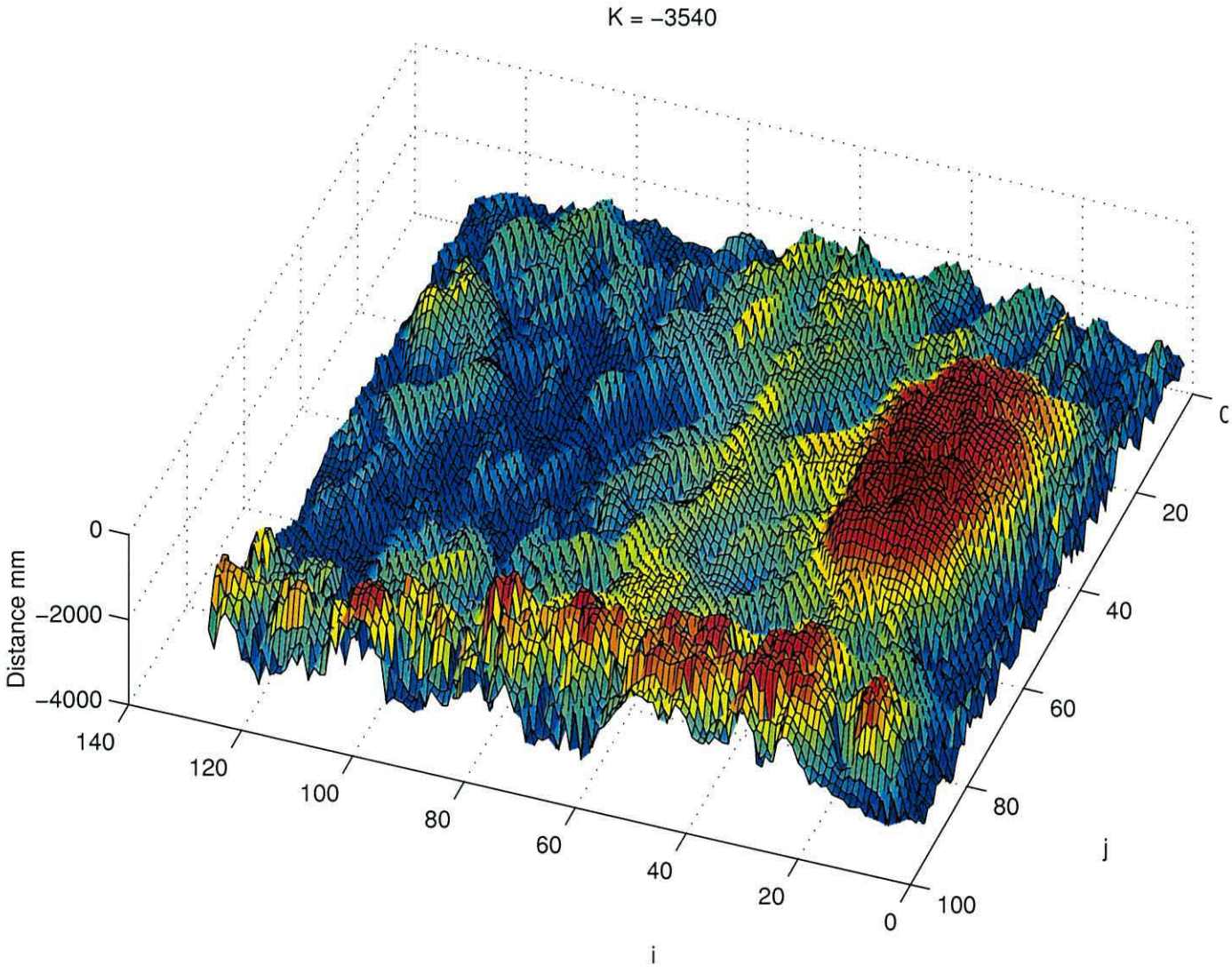


Figure 8.19: Range map for images 50 51 where W=3 L=4 I=10

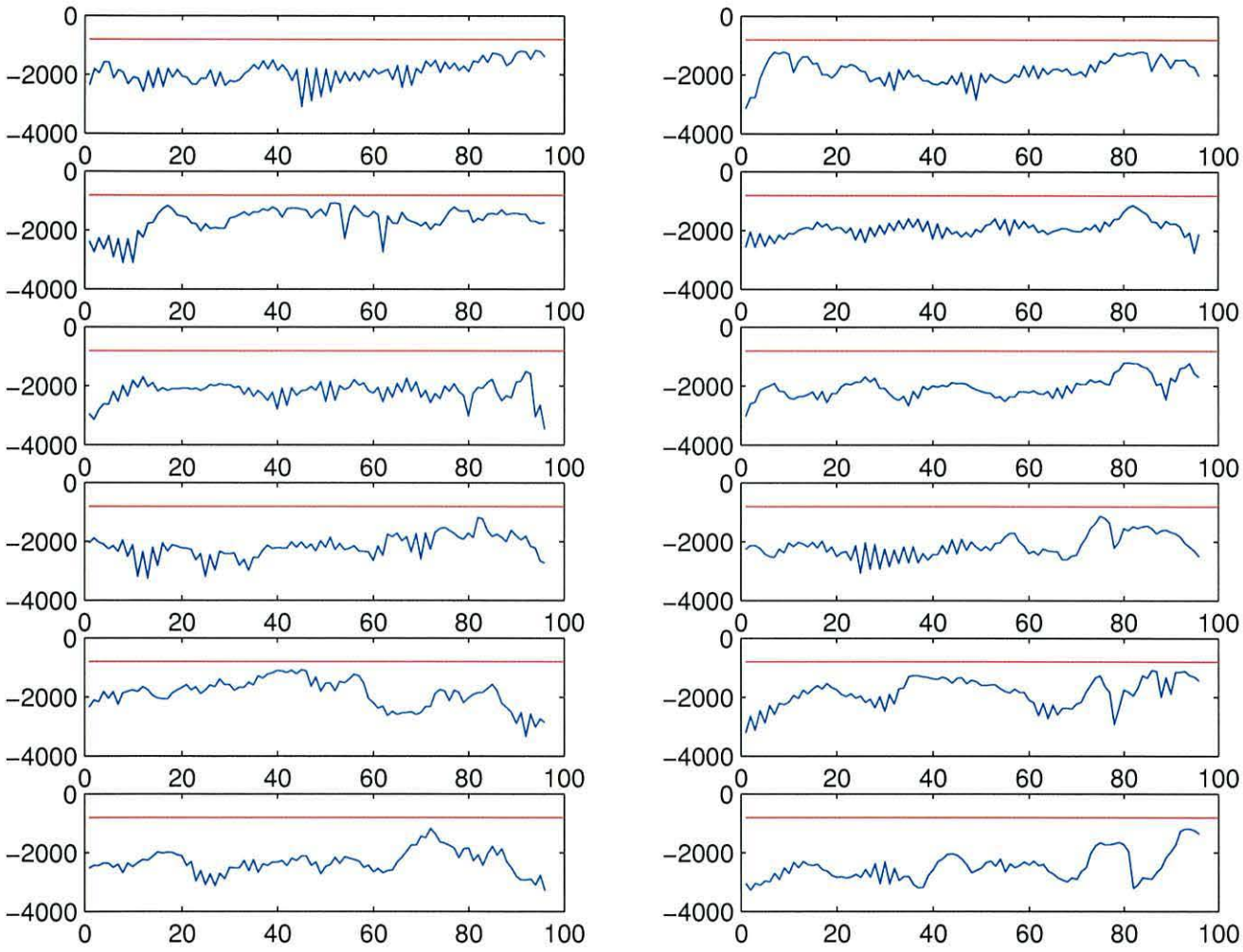


Figure 8.20: Depth cross-section for images 27 28 where W=3 L=4 I=10

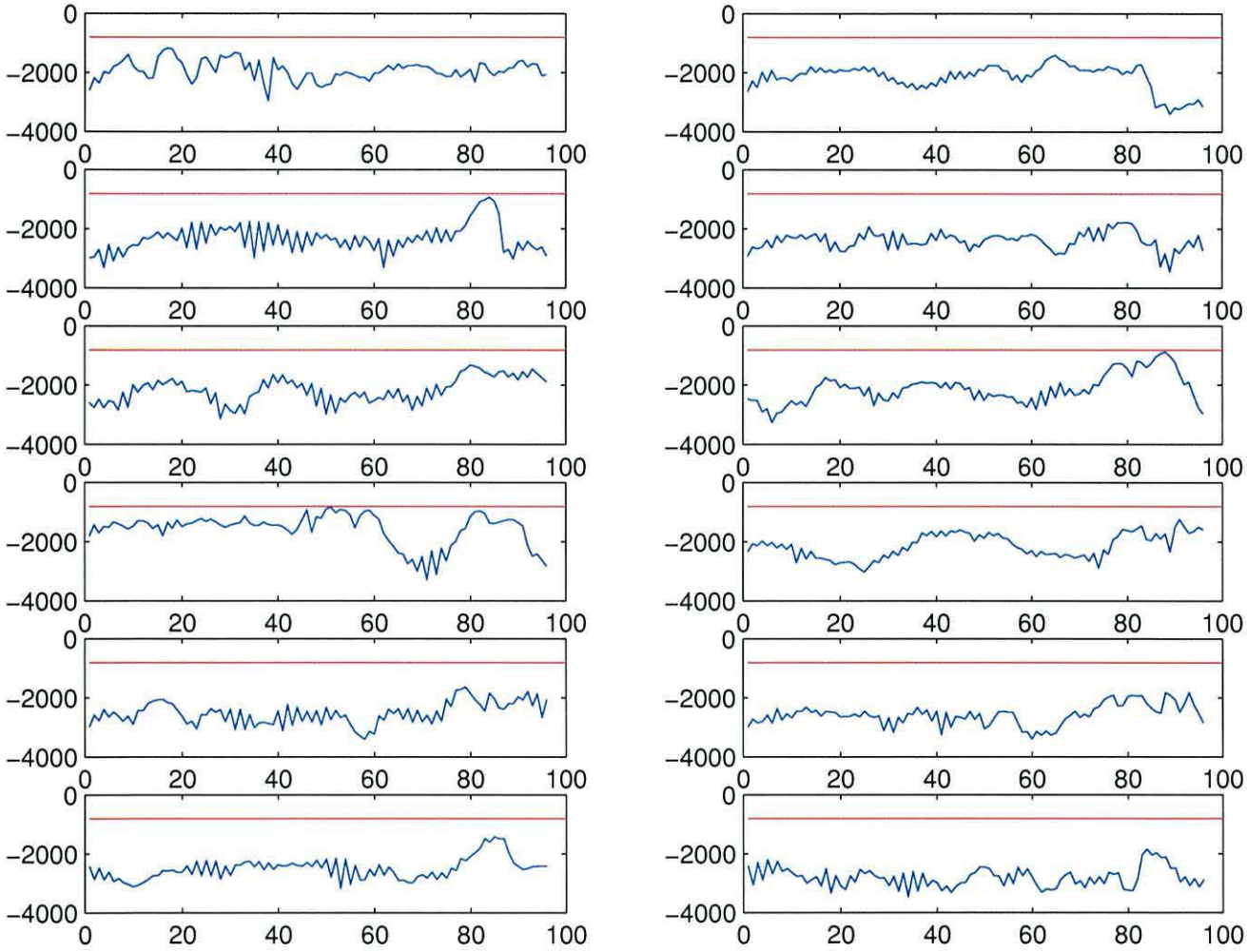


Figure 8.21: Depth cross-section for images 37 38 where W=3 L=4 I=10

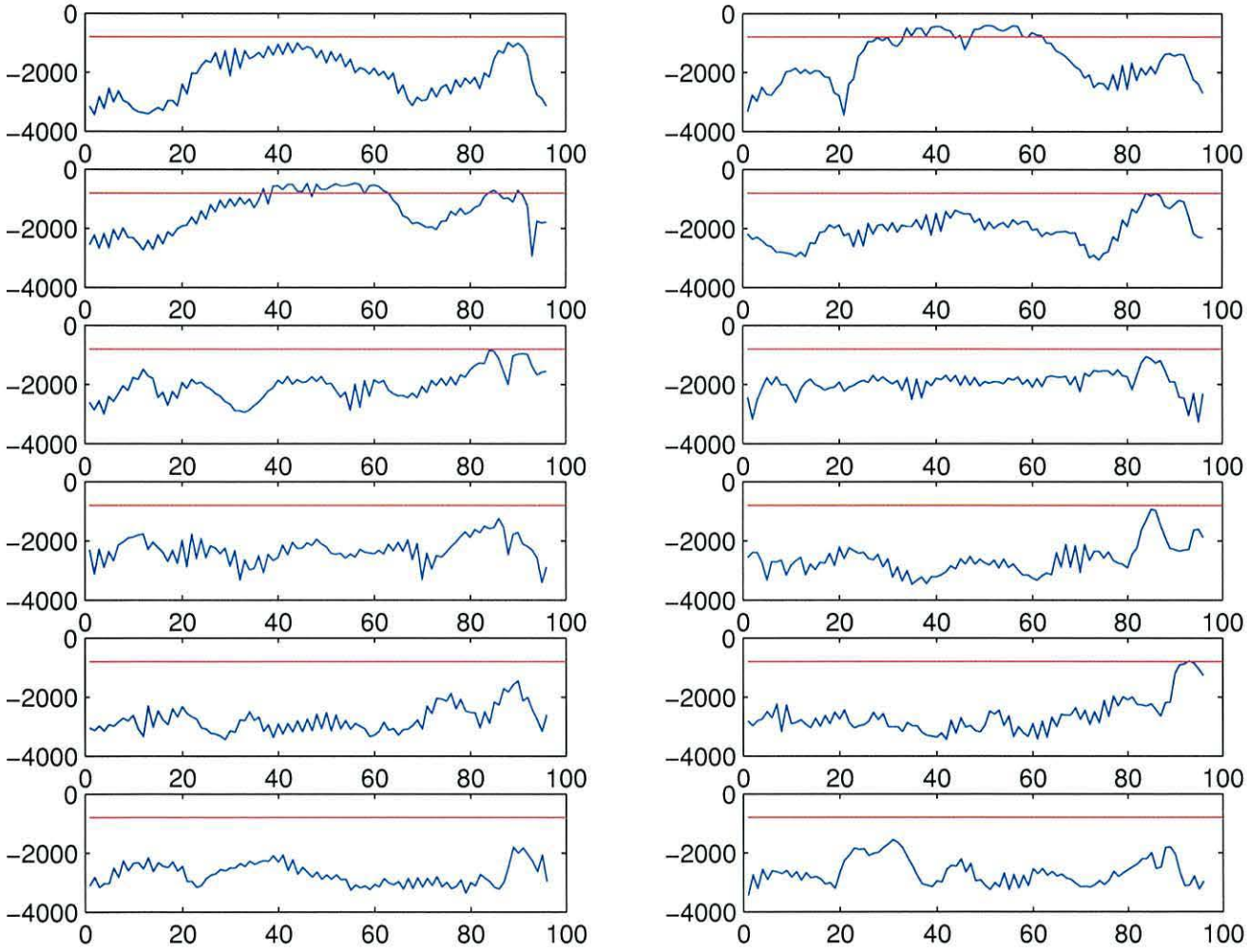


Figure 8.22: Depth cross-section for images 50 51 where W=3 L=4 I=10

8.1.4 Map building

Once the depths of each point in the image is known, a transformation between the image plane and the real-world coordinates referred to the cameras axes can be performed. The maps presented in this section have all been generated using the depth maps presented above. However, the depth estimation can only give details of the visible surfaces of the obstacles in the image, obviously there is no information about how far the obstacle extends away from the camera, nor is there any detail about what lies behind the obstacles. In the maps given below the volume "behind" the obstacle has been filled in to indicate that there may be an obstacle there.

A better method of map building would be to add information about obstacles to the map as they are discovered, ie. building the map over time. This would reduce the amount of space wrongly marked as obstacle, and could allow a path to be produced where otherwise one would not be possible.

The transformation between image plane and world coordinates is achieved using the following:

$$py = d \tag{8.4}$$

$$px = (i - \frac{I}{2}) \times \alpha \tag{8.5}$$

$$pz = (j - \frac{J}{2}) \times \beta \tag{8.6}$$

where d is the distance obtained from the optical flow, (i, j) is the position in the image plane, and (I,J) is the size of the image (128×96).

The values α and β are calculated from the size of the image, the lens configuration and the size of the map. In this case the map was set to be a cube size 1600mm with the camera at the centre. The CCD chip measures $\frac{1}{2}$" or 10.16mm by 7.62mm and we are using a resolution of 128×96. The field of view of the camera is 60° in the azimuth and 44° in elevation. Figure 8.23 shows the setup.

So α can be generated as follows:

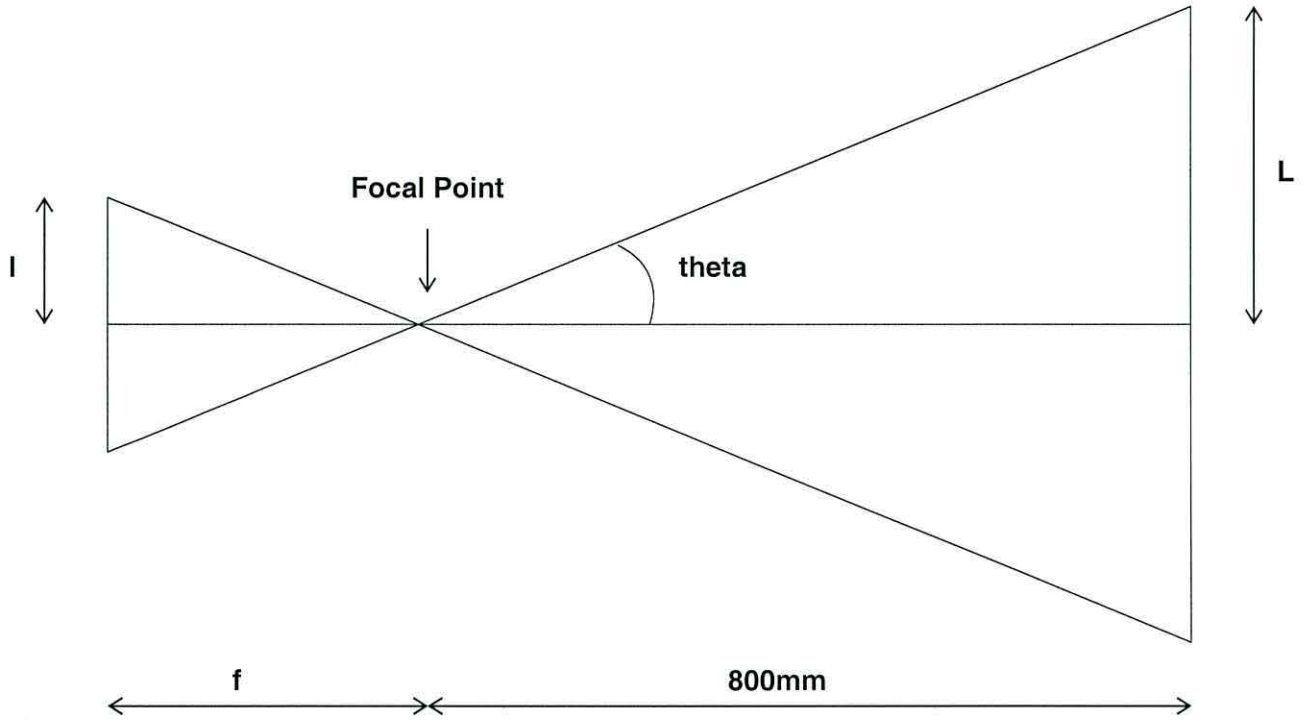


Figure 8.23: Map generation parameters

$$L = 800 \times \tan\theta$$
$$L = 800 \times \tan 30 = 461$$
$$\alpha = L/l = 461/64 = 7.20$$

β can be calculated in a similar way and is found to be 6.73. So,

$$\begin{aligned} 7.20 \times i &= px \\ 6.73 \times j &= py \end{aligned}$$

gives the transform from pixel (i,j) to world coordinates (px,py,pz).

Applying the transform to the depth values yields the three dimensional maps given in figures 8.24 to 8.31. The red sphere in the centre represents the camera pointing down at the bottom inside surface of the cube. This surface is the plane

orthogonal to the optical axis. The obstacles are coloured blue, and in order for depth to be represented shadows are present - these can be confusing but are necessary to present depths. Also, as the map builder only detects the surface of an obstacle and cannot determine what may be occluded by it, the map building algorithm fills in the volume behind the detected surface.

Figure 8.24 shows the first map generated and, correctly, it contains no obstacles. Figure 8.25 shows the map generated from images 27 and 28 and again the map is empty except for a small point near the centre; this is noise. From the cross-sections presented in figure 8.20 is is clear that all the obstacles are outside the limit of the map. This is true for the map shown in figure 8.26 where the cross-section given in figure 8.21 again shows that all the obstacles lie outside the map.

A sample of the sequence showing the progress of the map builder are shown in figures 8.27 to 8.30. Figure 8.27 start shows the obstacle breaking through the bottom surface and entering the map for the first time. Later figures in the sequence show the obstacle range reducing and the size increasing as the camera approaches it. The sequence also shows the lateral motion of the obstacle, although an animation of the complete sequence shows this much more clearly.

Figure 8.31 is an empty map (apart from noise) because the obstacle is no longer in the camera's field of view.

From these maps paths can be produced using the method presented in chapter 5.

In this section, it has been shown that Anandan's optical flow method, used with parameters chosen by experimentation from a set of "ground truth" images, can produce a flow field which reliably estimates distances to objects within images. This depth estimation allows a three-dimensional map to be generated. The map can then be used by a path planner to produce a path to avoid any obstacles within the map.

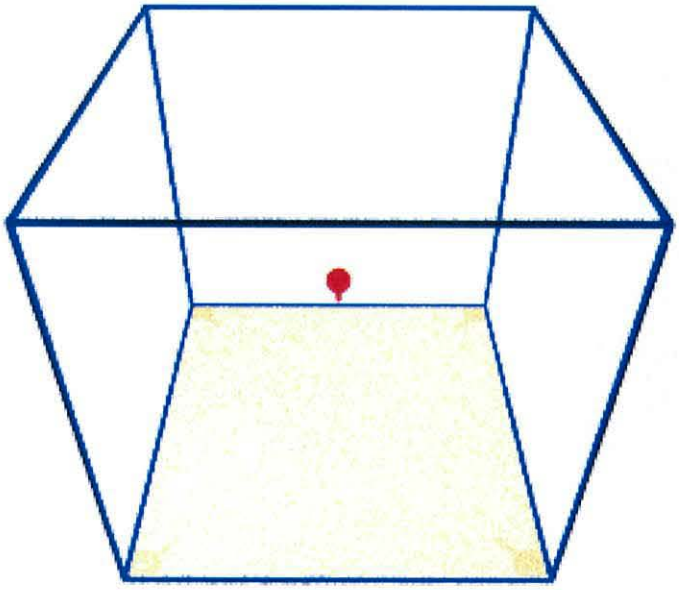


Figure 8.24: Map Images 0 and 1

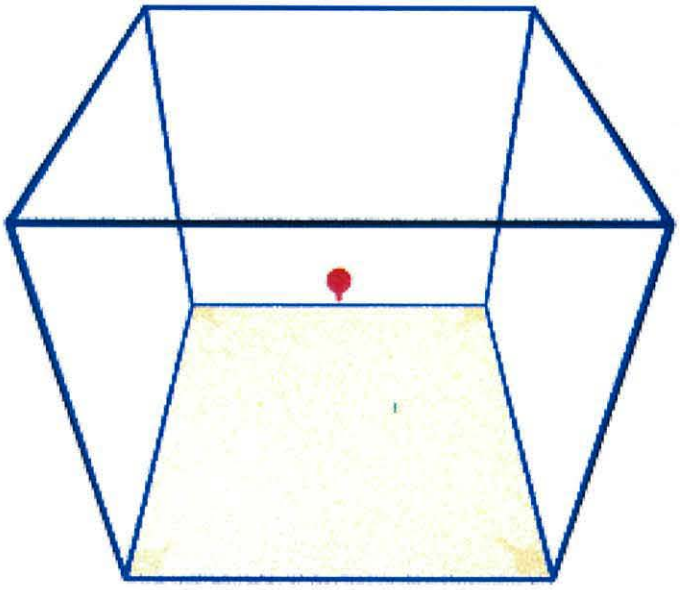


Figure 8.25: Map Images 27 and 28

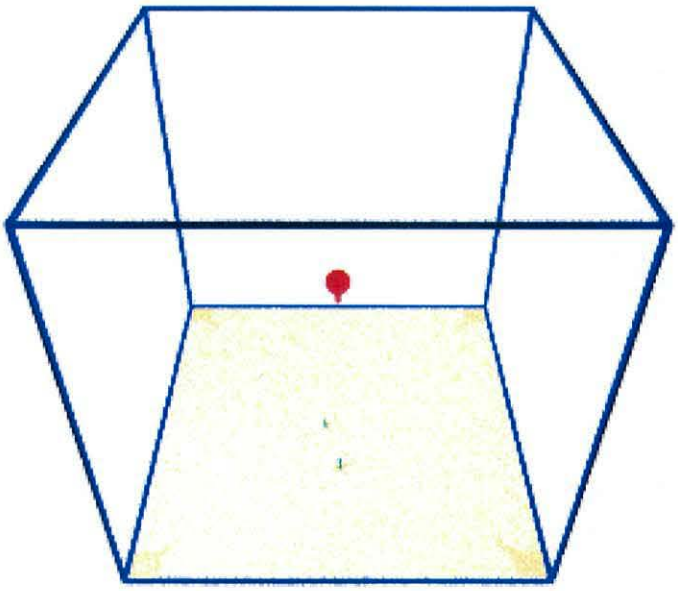


Figure 8.26: Map Images 37 and 38

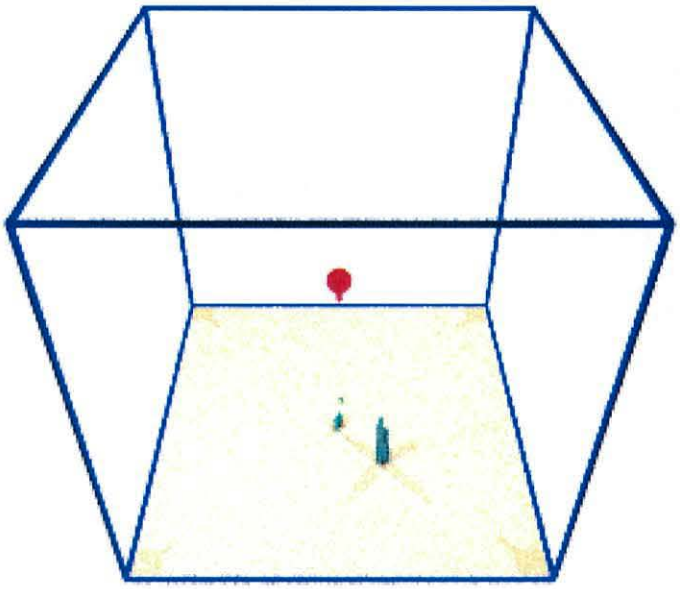


Figure 8.27: Map Images 40 and 41

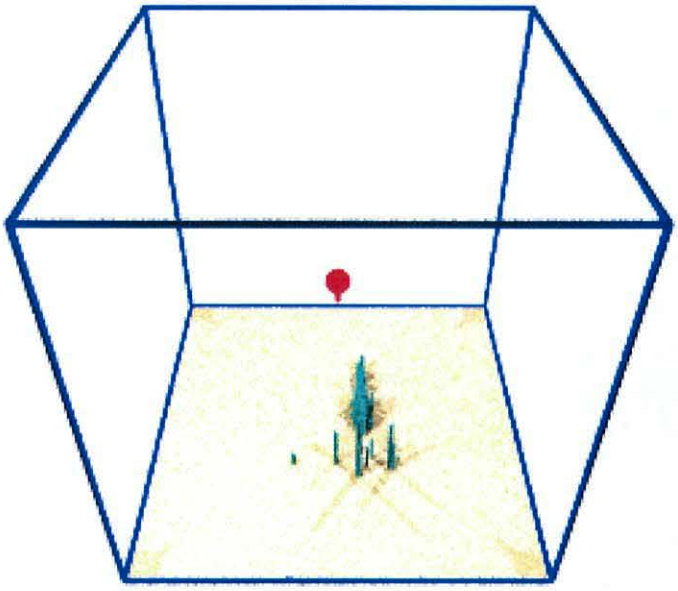


Figure 8.28: Map Images 45 and 46

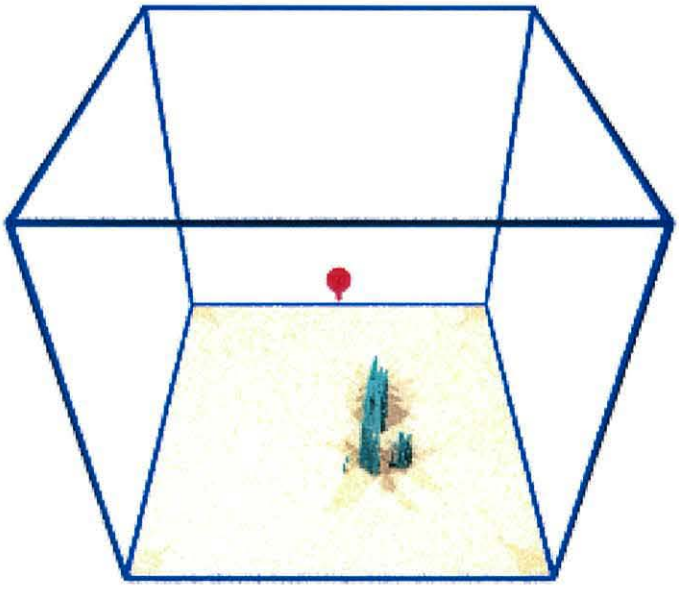


Figure 8.29: Map Images 47 and 48

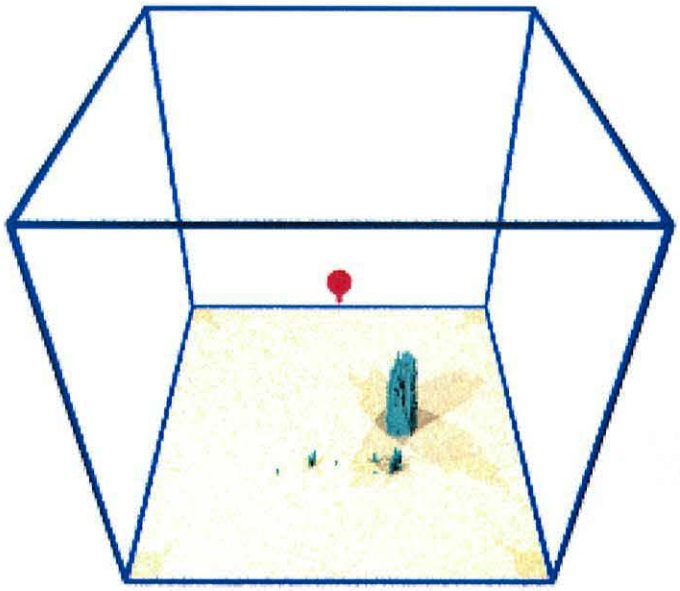


Figure 8.30: Map Images 50 and 51

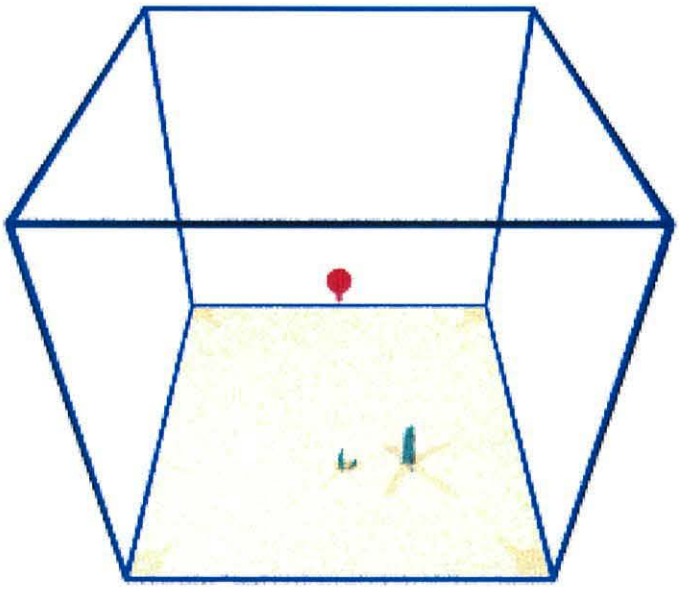


Figure 8.31: Map Images 54 and 55

8.2 Test Rig Experiments

This section gives details of experiments performed on the test rig. Up to now, all the results presented in this chapter and from the investigation of the path planner detailed in chapter 5 have used "hand-made" data. This was necessary in order to test the operation of the different sub-systems. The aim of this section is to show how the controller architecture has combined these sub-systems in order to perform real-time collision avoidance. A number of typical runs will be presented and a discussion of the results will be given.

In all the results presented in the experiments below, the camera mount is set to move along the *navigator's* path which has been defined as a straight line from the starting point parallel with the Y axis towards the background. If the camera deviates from this path to avoid an obstacle, then once the obstacle is passed the camera should return to the original path.

The sequence of figures is the same for each experiment below. The first figure gives a sample of the images captured during the experiment, the next figure shows the flow fields for these images. Next the maps generated from the flow fields are given and finally the path followed during the experiment is shown, the dots on the path show the location where the images were captured.

Three dimensional paths are produced by the planner within the test rig's software, but as the rig only has two degrees of freedom only the X and Y components of the paths are used. Not having the third axis will have an effect on the subsequent paths which may make the resulting motion seem inefficient or unusual.

8.2.1 Experiment One

In this test one obstacle is present. Figure 8.32 shows a selection of the images captured during the run.

In this run, the camera moves both towards the obstacle and from right to left on the test rig. The obstacle therefore appears larger in the scene and moves from

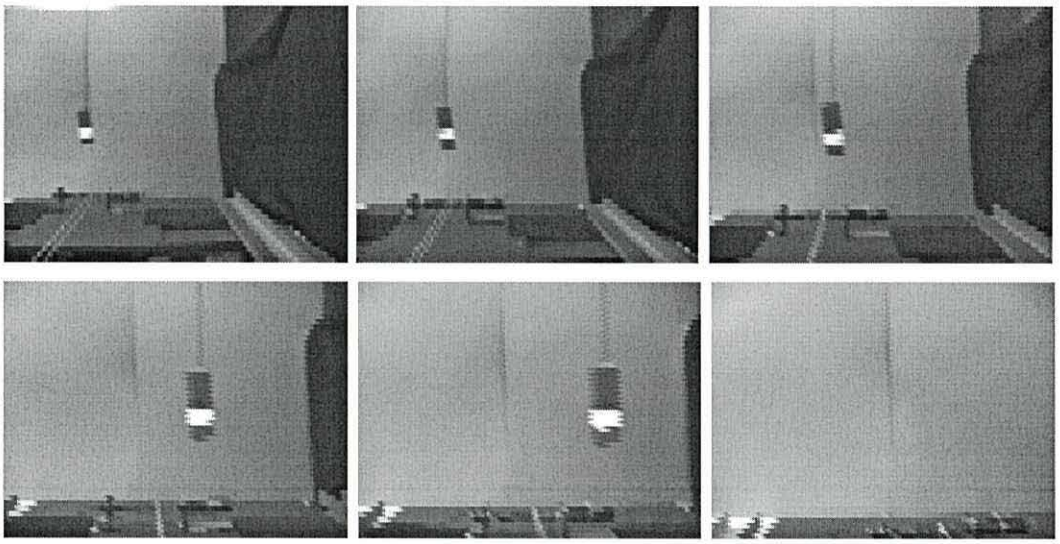


Figure 8.32: Images captured during experiment 1

left to right in the image. As in the experiments presented above, a number of other features are present in the images. These include the black cover to the right, a join in the back ground and the capstan assembly towards the bottom of the images. These all play an important part in the understanding of the flow fields presented in figure 8.33.

The flow fields in figure 8.33 correspond to the images presented in figure 8.32. In figure 8.32 at the top right, a number of features can be seen. The capstan assembly has been detected at the bottom of the image and this is also present in the next three images. Towards the right hand side of this flow field a distinct line from top-to-bottom can be seen, this corresponds to the join between the background and the black cloth. Just visible at the left of the image is the obstacle. In the subsequent images the features can be seen to move from left to right, also the "size" of the flow fields can be seen to be increasing, this is particularly true for the obstacle. In the penultimate case the flow field corresponding to the obstacle is large, indicating that it is close to the camera.

The next stage is crucial. This is the conversion from flow field to location. Figures 8.34 to 8.39 show the maps calculated from the flow fields above.

Figure 8.34 to 8.39 shows the maps generated during this experiment. In the case of the map presented in figure 8.34 it has been calculated that no obstacles are

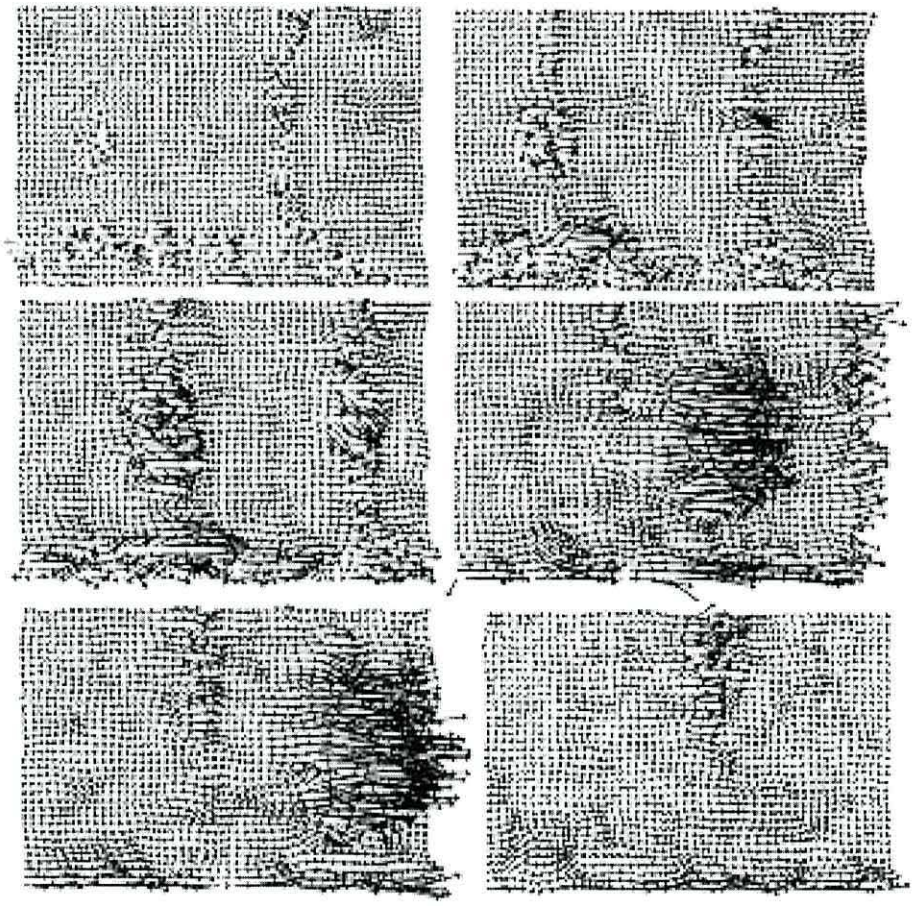


Figure 8.33: Flow fields generated during experiment 1

within range and the map is empty. As the camera moves a distinct obstacle area appears to map in figure 8.35, this is the capstan assembly. Figure 8.36 to 8.38 shows the maps produced as the camera approaches the obstacle. Figure 8.39 shows that once the images is out of view the map becomes less cluttered.

The path generated by this run is given in figure 8.40. This shows the test rig moving the camera assembly to avoid the obstacle that was detected. By looking at the position of the camera and the obstacle at the beginning of the sequence, path planning should not be needed as the *navigator's* path misses the obstacle. However, during the initial stages the obstacle location system has detected the black cloth to the right and has produced a path to avoid this. Once the camera begins to move to the left, it then detects the obstacle as continues towards the left to avoid it. Eventually, the obstacle moves out of the field of view and the rig begins to return to the pre-planned *Navigator's* path.

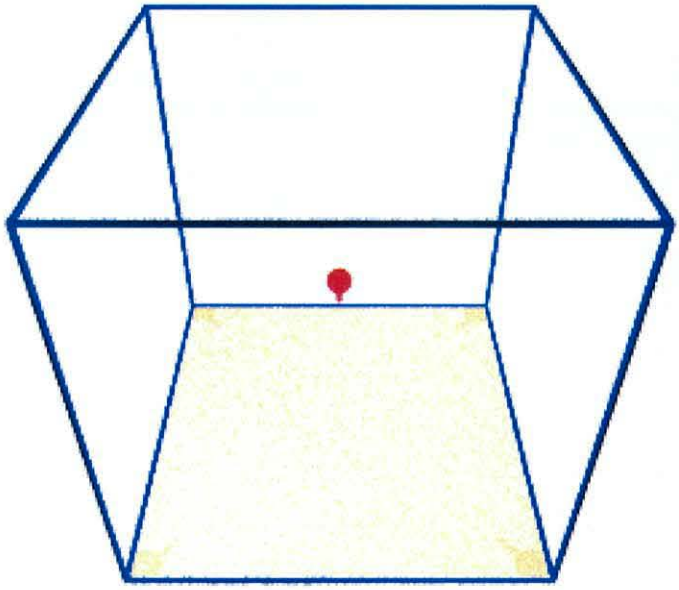


Figure 8.34: Map 0 calculated during experiment 1

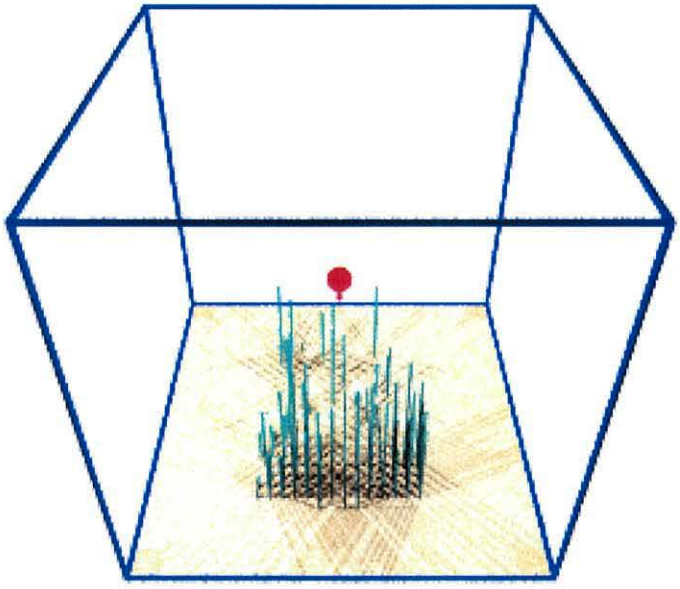


Figure 8.35: Map 5 calculated during experiment 1

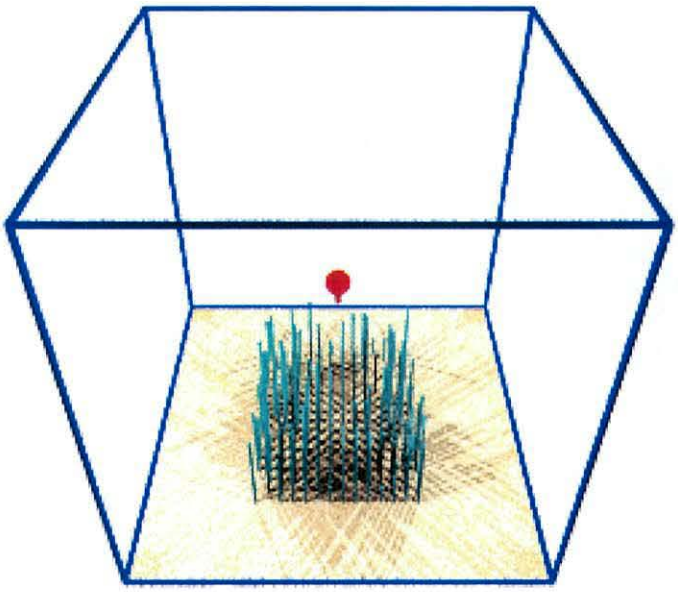


Figure 8.36: Map 10 calculated during experiment 1

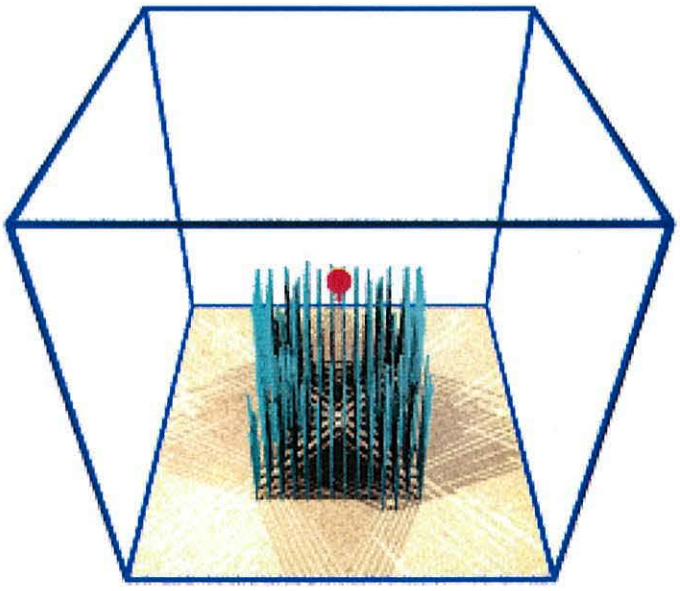


Figure 8.37: Map 15 calculated during experiment 1

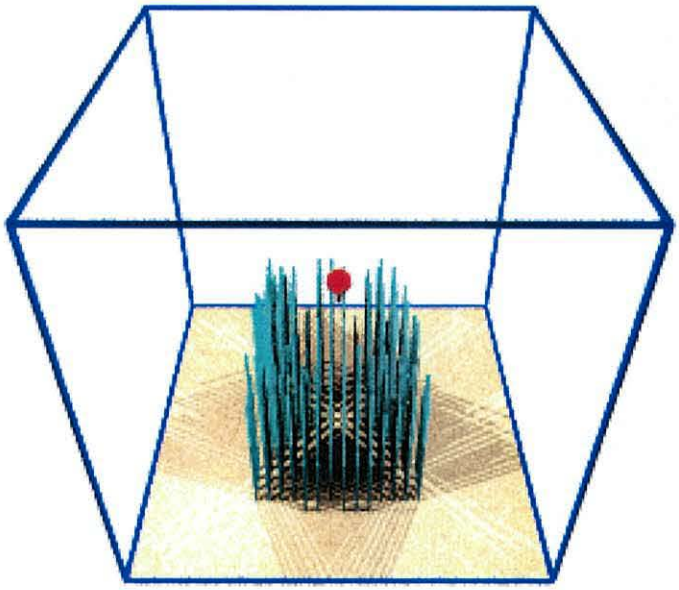


Figure 8.38: Map 17 calculated during experiment 1

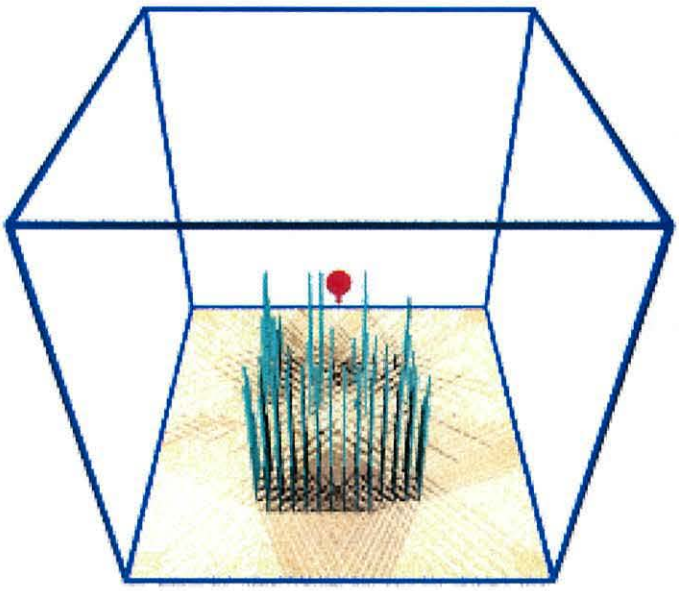


Figure 8.39: Map 20 calculated during experiment 1

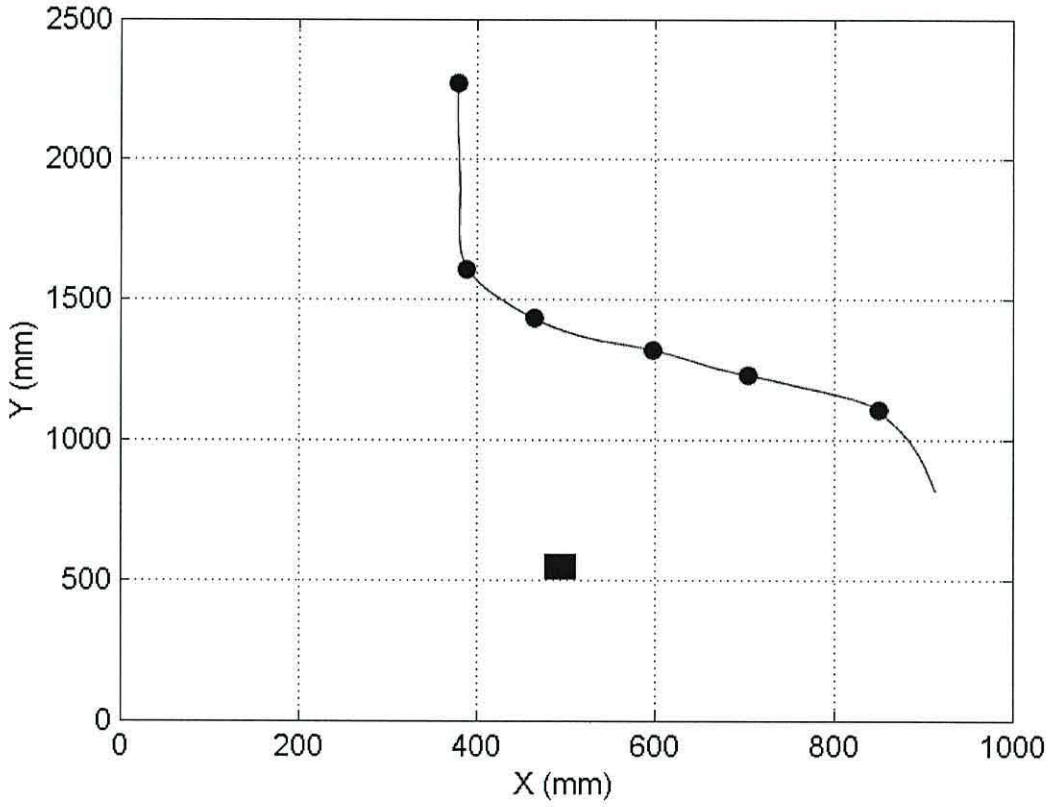


Figure 8.40: The path followed during experiment 1

8.2.2 Experiment Two

In this test no distinct obstacle was present. Figure 8.41 shows some of the images captured during this experiment.

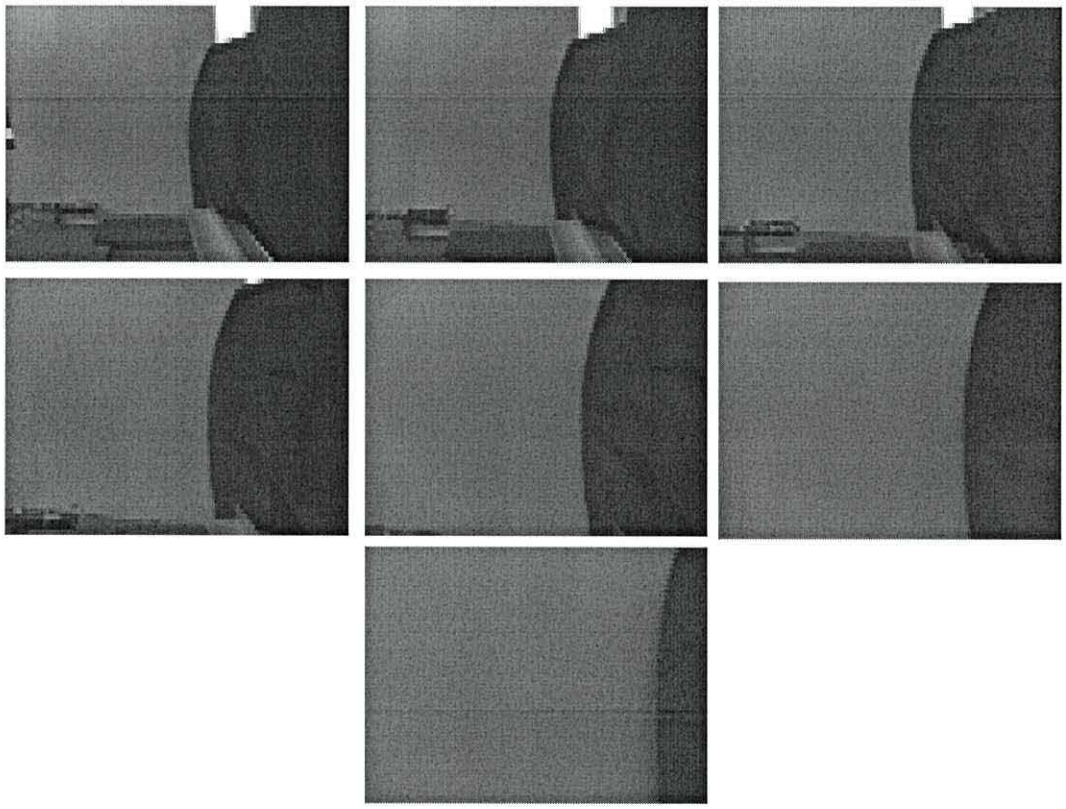


Figure 8.41: Images captured during experiment 2

In this series of images, the camera does not move in the X direction at all it simply moves slowly towards the background. The resulting flow fields are shown in figure 8.42. In this collection of flow fields, the join between the background and the black cloth is in the centre. As the camera moves towards it, the flow fields sizes get bigger. Also note on the right hand side a large flow field is measured which is due to the proximity of the black cloth. Also note the capstan assembly is still visible at the bottom of the initial images.

The resulting maps are shown in figures 8.43 to 8.49. The maps show that as the camera gets closer to the background the larger the flow fields detected. Figure 8.43 shows the first map produced, here only a few points are visible in the map. As the experiment progresses more obstacle points appear in the image and by

the map shown in figure 8.46 two distinct obstacle areas are visible. These are due to the gap in the background at the top and the capstan assembly at the bottom. As these move out of the field of view of the camera their effects reduce and by the map shown in figure 8.47 they have disappeared. The remaining maps show show the background being detected.

The path generated by this run is given in figure 8.50.

In this experiment the camera does not move in the X direction even though obstacles have been detected, and simply follows the *Navigator's* path. This is entirely possible as the path planner has detected that the obstacles are not on the path. However, the results from the map building system in this experiment are less than ideal. This experiment shows a weakness in the obstacle detection system. The optical flow software seems to be very sensitive to fluctuation in light level. The net effect of these fluctuations is that the obstacles appear closer than they are.

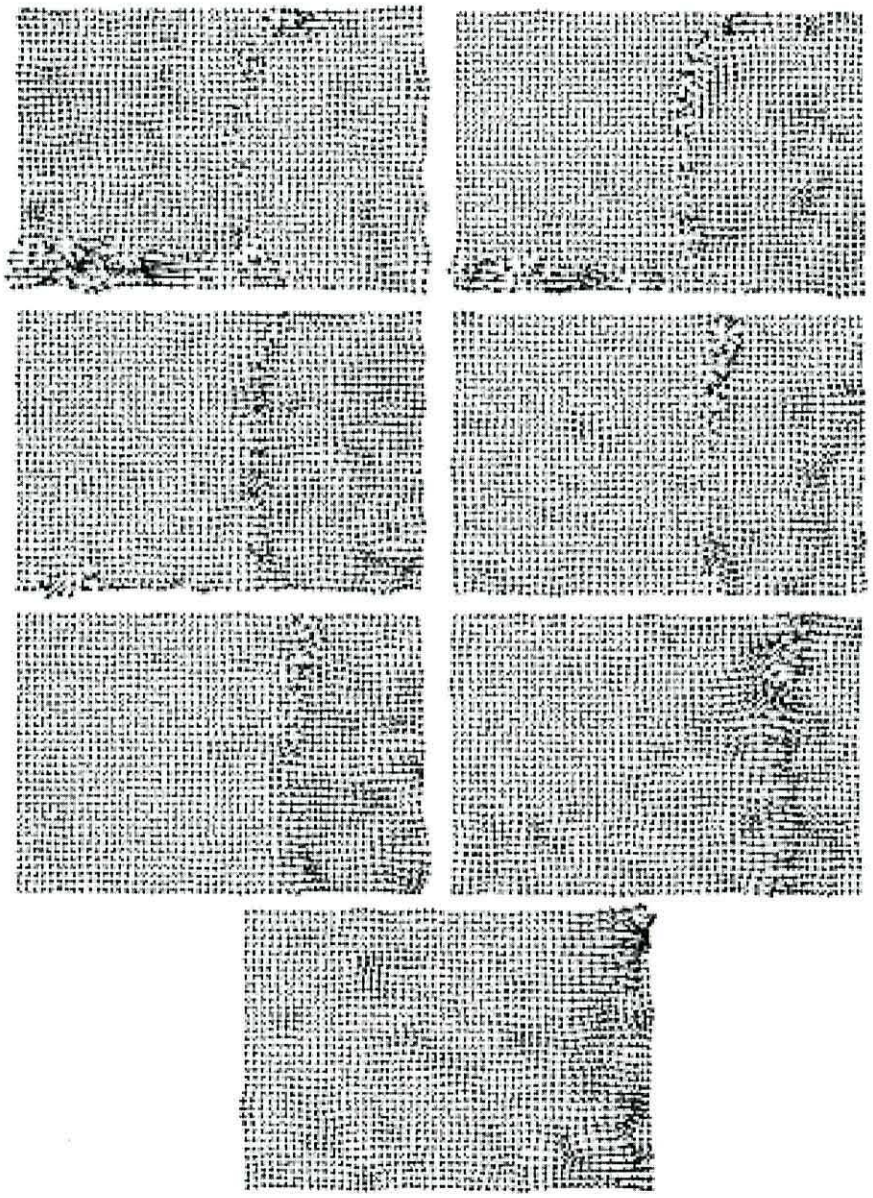


Figure 8.42: Flow fields generated during experiment 2

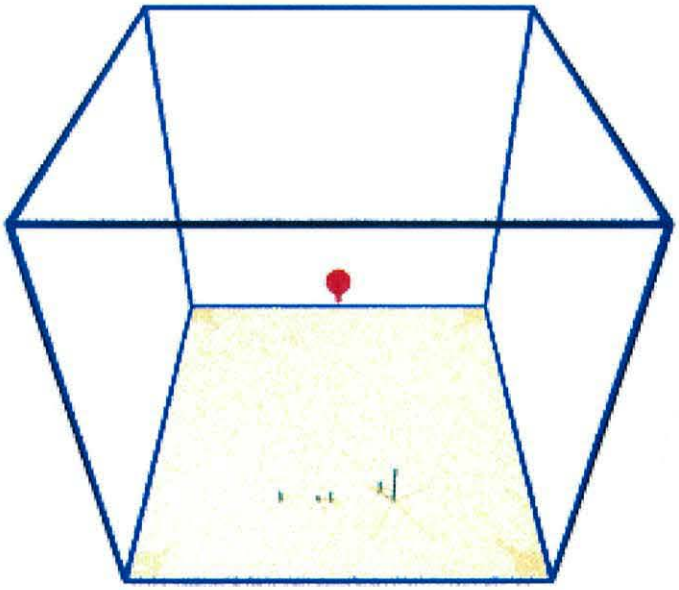


Figure 8.43: Map 1 calculated during experiment 2

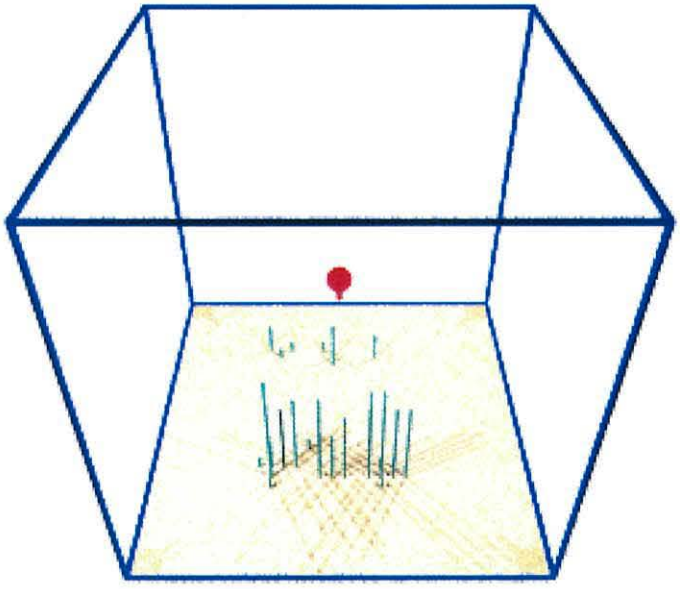


Figure 8.44: Map 5 calculated during experiment 2

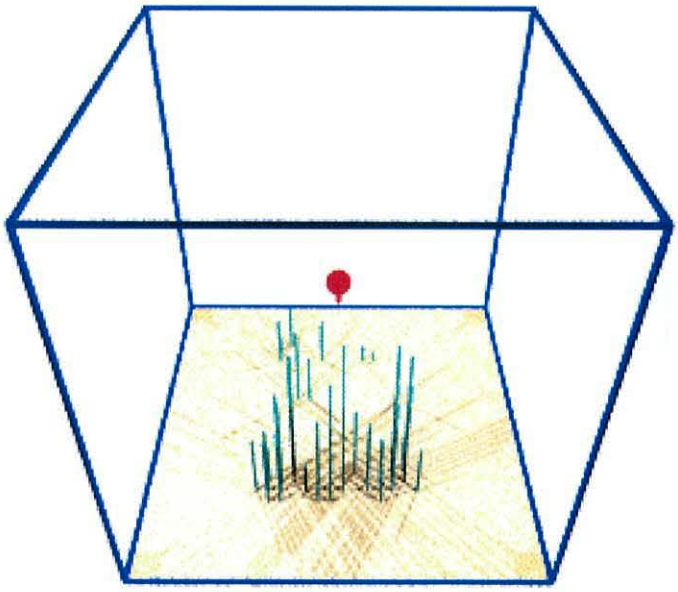


Figure 8.45: Map 10 calculated during experiment 2

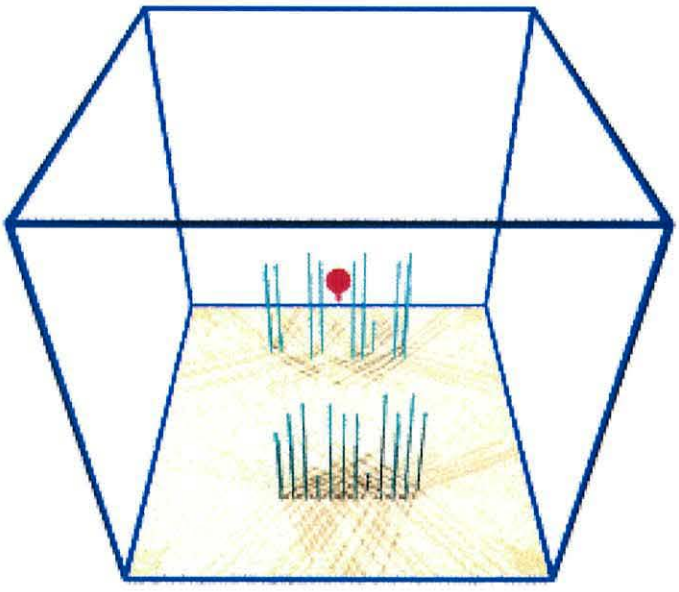


Figure 8.46: Map 15 calculated during experiment 2

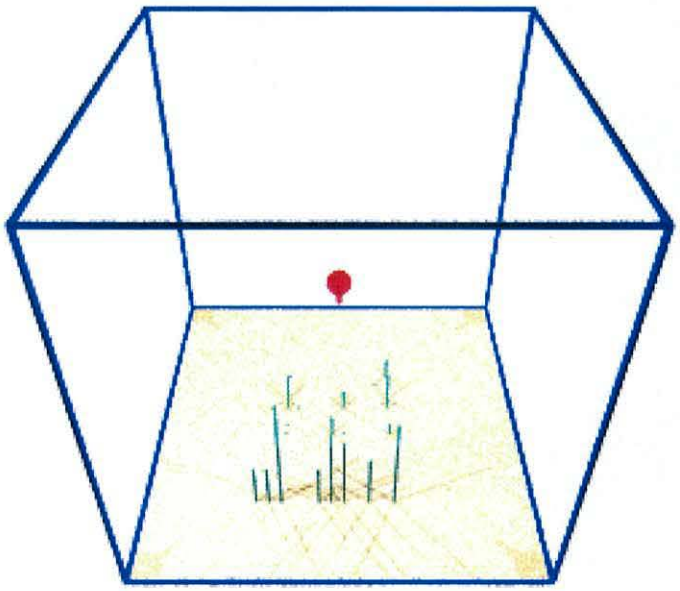


Figure 8.47: Map 20 calculated during experiment 2

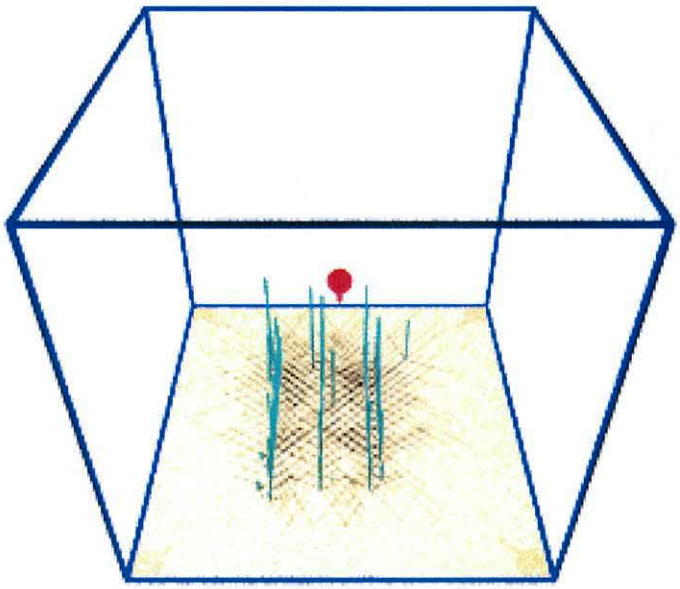


Figure 8.48: Map 25 calculated during experiment 2

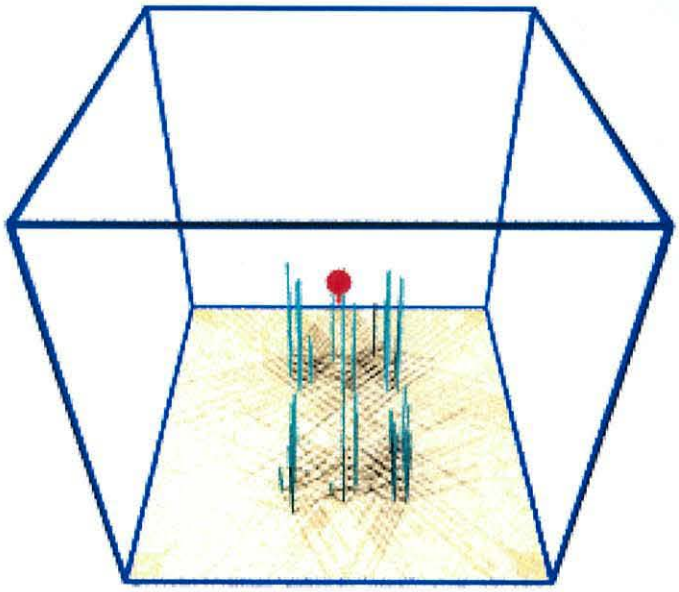


Figure 8.49: Map 30 calculated during experiment 2

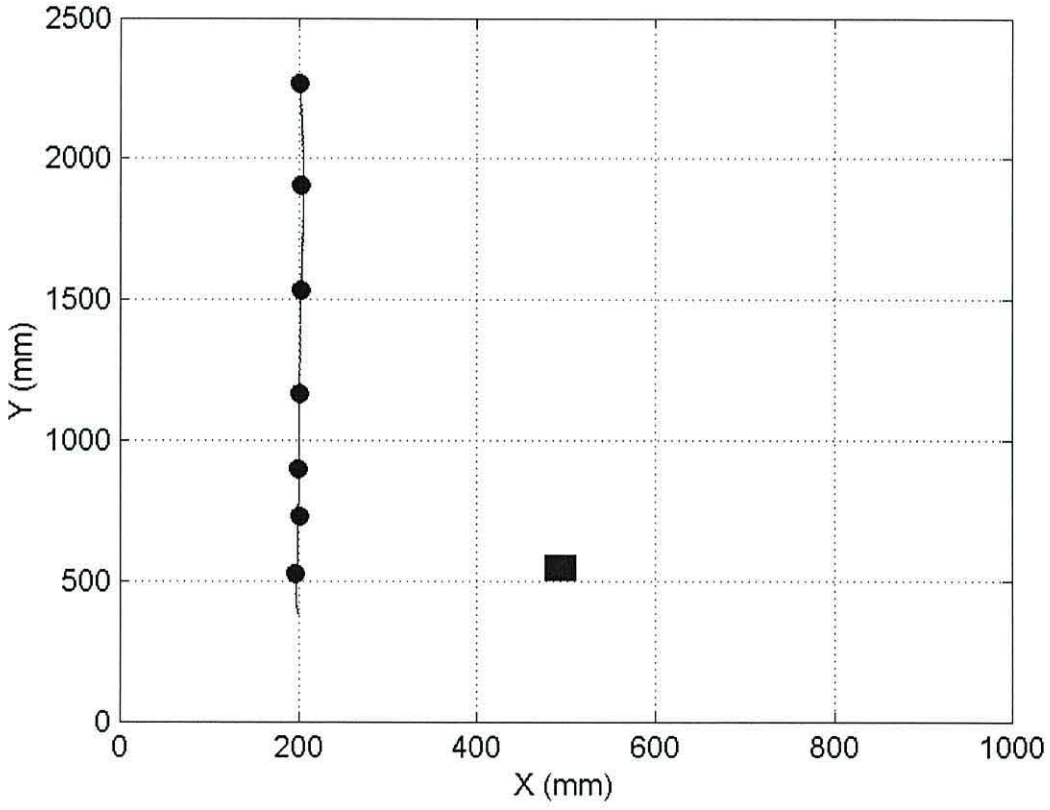


Figure 8.50: The path followed during experiment 2

8.2.3 Experiment 3

In this experiment two potential obstacles are present. Figure 8.51 shows some of the images captured during the experiment.

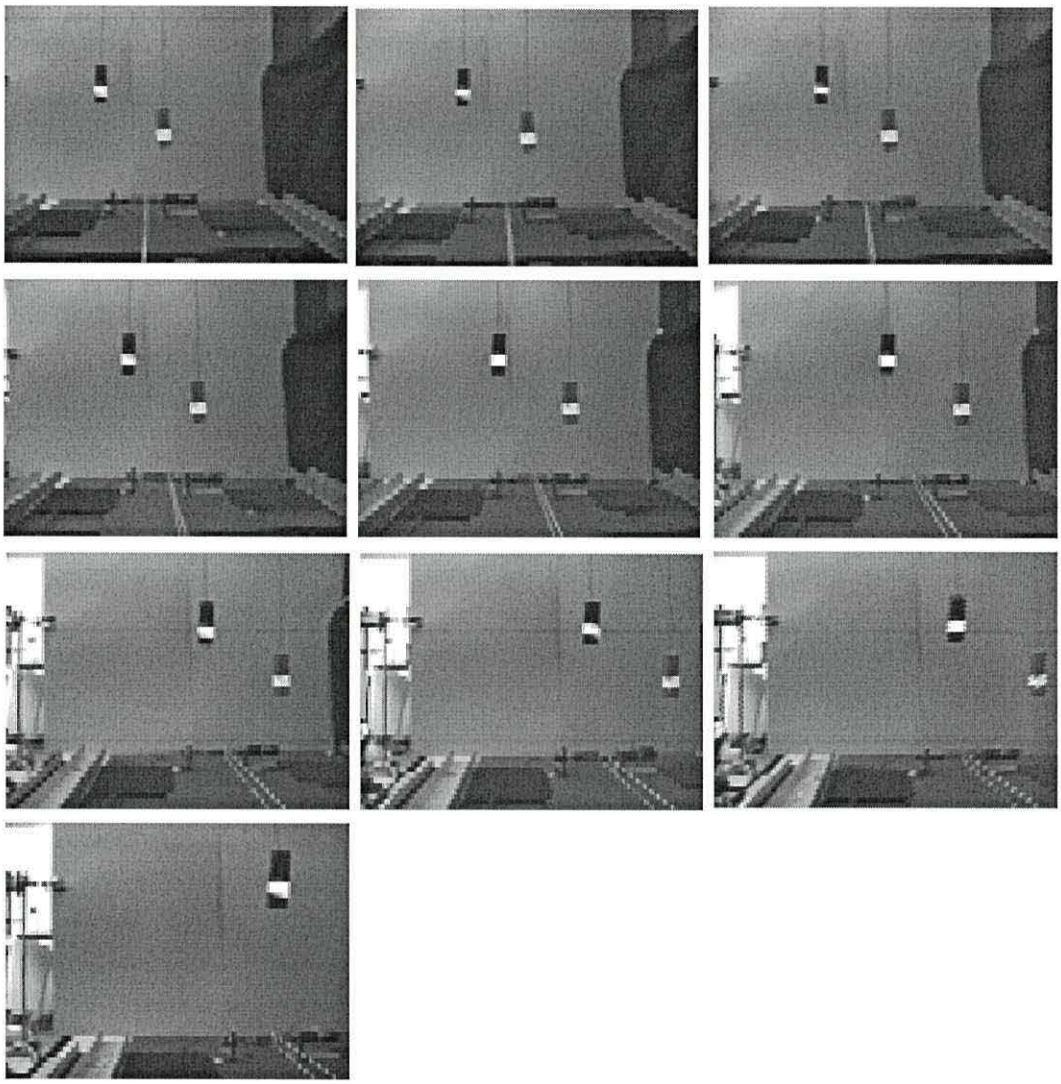


Figure 8.51: Images captured during experiment 3

The flows generated from the image sequences above are shown in figure 8.52. These clearly show the two obstacles. However, the join in the background card, the capstan assembly are also visible. In the last few flows the left hand edge is prominent because the lateral motion of the camera causes the edge of the background card to appear in the scene

The maps produced from these flow fields are shown in figures 8.53 to 8.63. The first few maps are empty as the obstacles are out of range. The next few maps then show the obstacles being detected and coming closer to the camera, although the rate of change of depths is small as the camera is moving laterally in the middle period of the sequence.

The figures above show the progression of the two obstacles through the image sequence. With reference to the results from experiment 2, the obstacle location system is working better in this experiment as the light level was constant during the experiment.

The path generated by this run is given in figure 8.64. The path shows that the obstacles were first detected when they were about 750mm away from the camera. As the camera moves towards them the path planning system begins to move the camera to the left and successfully avoids the obstacle.

The path in figure 8.64 was cut short by the software crashing. This is due to the camera moving too far to the left, beyond the limit of the "background". Figure 8.51 shows the bright room emerging at the left of the last couple of images. The software crash is caused by the map building function of the software, the exact reason for the crash is unknown.

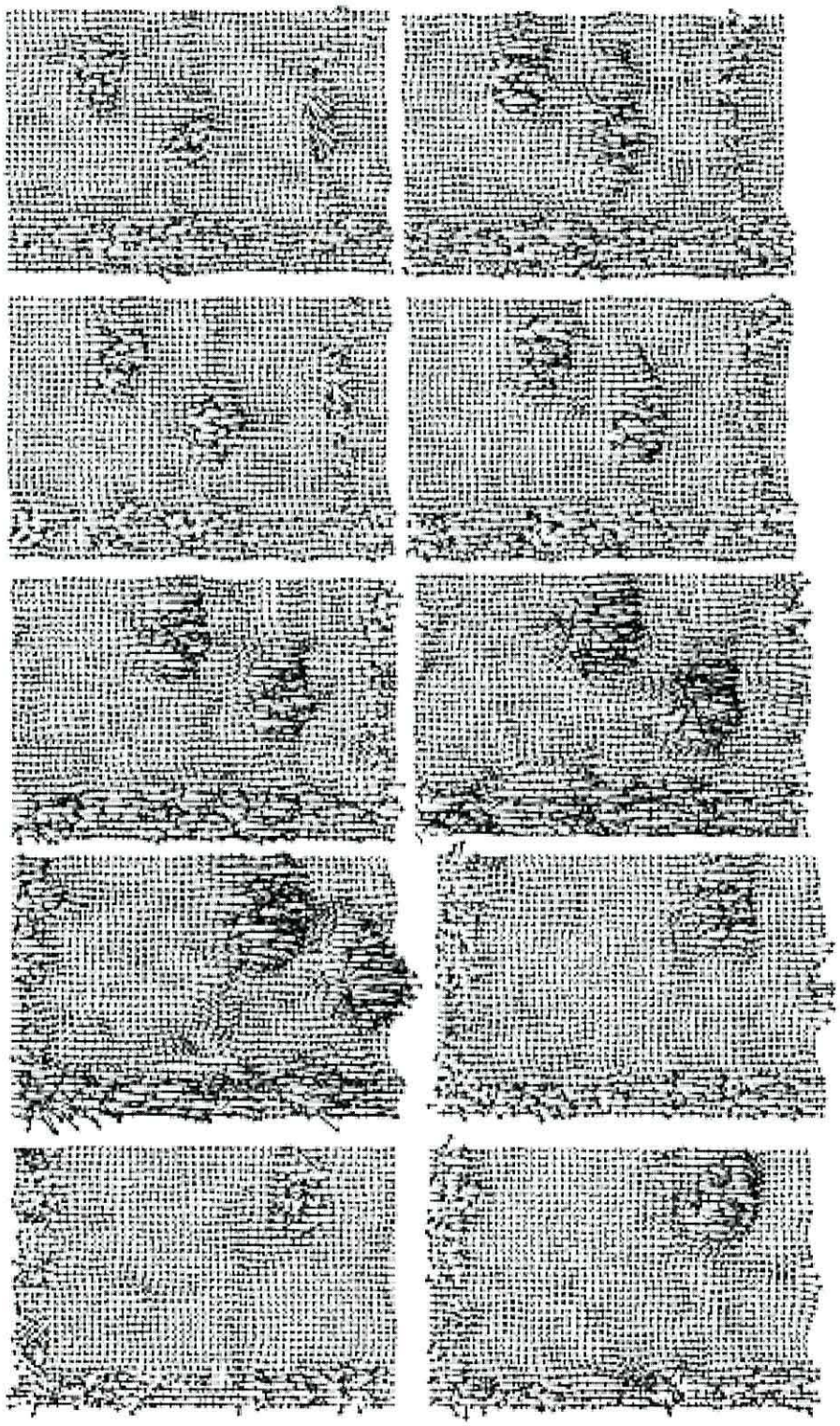


Figure 8.52: Flow fields generated during experiment 3

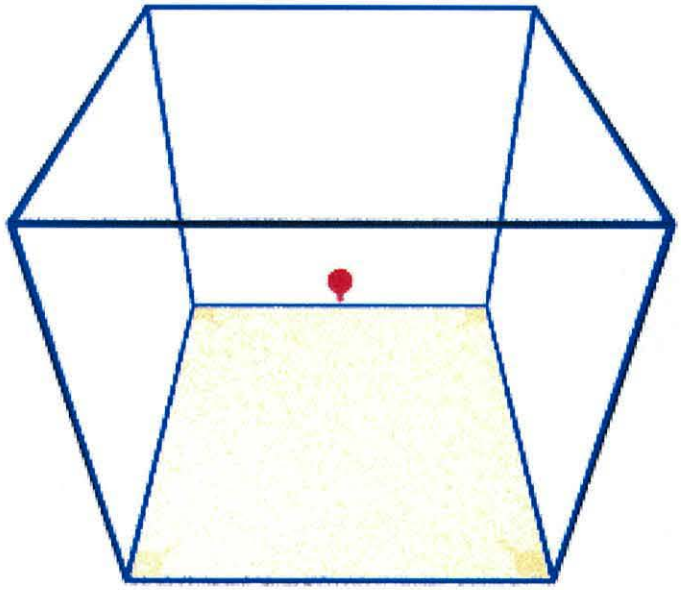


Figure 8.53: Map 1 calculated during experiment 3

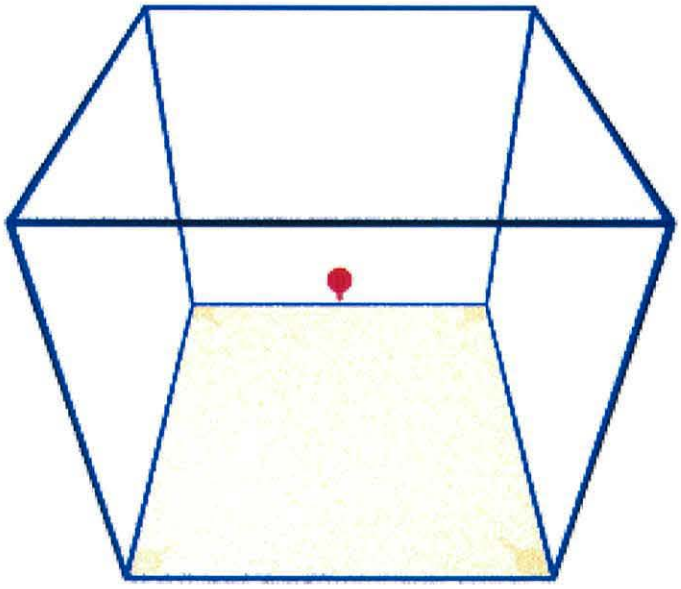


Figure 8.54: Map 5 calculated during experiment 3

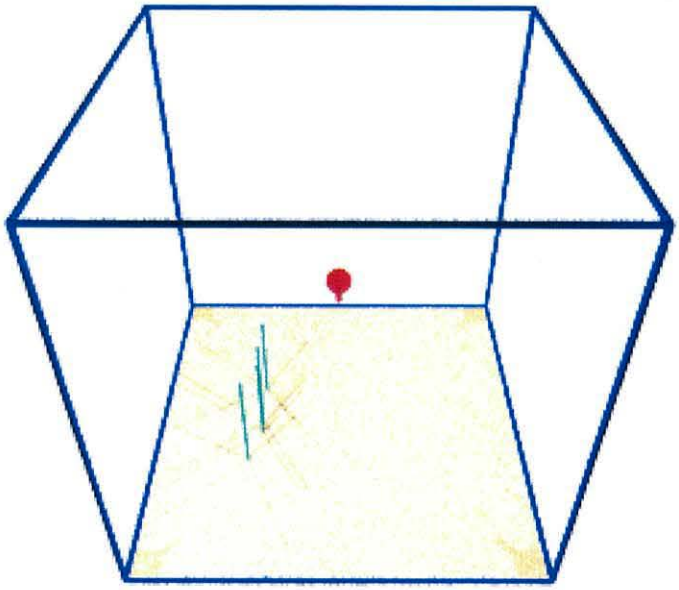


Figure 8.55: Map 10 calculated during experiment 3

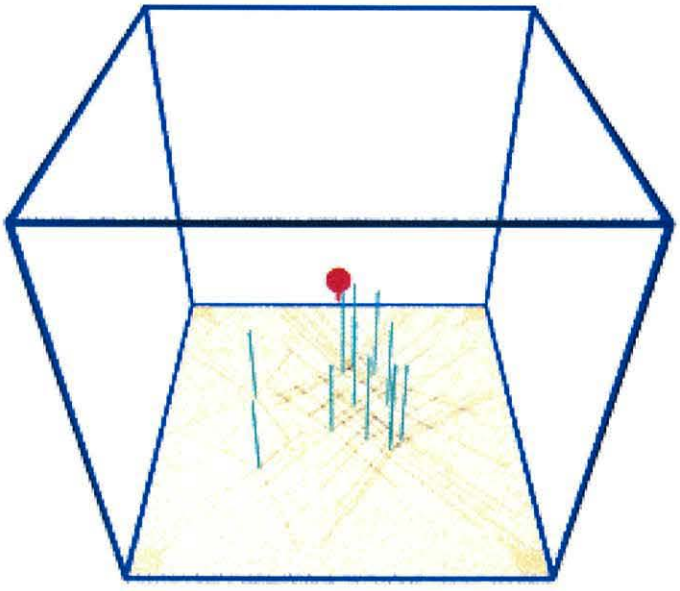


Figure 8.56: Map 14 calculated during experiment 3

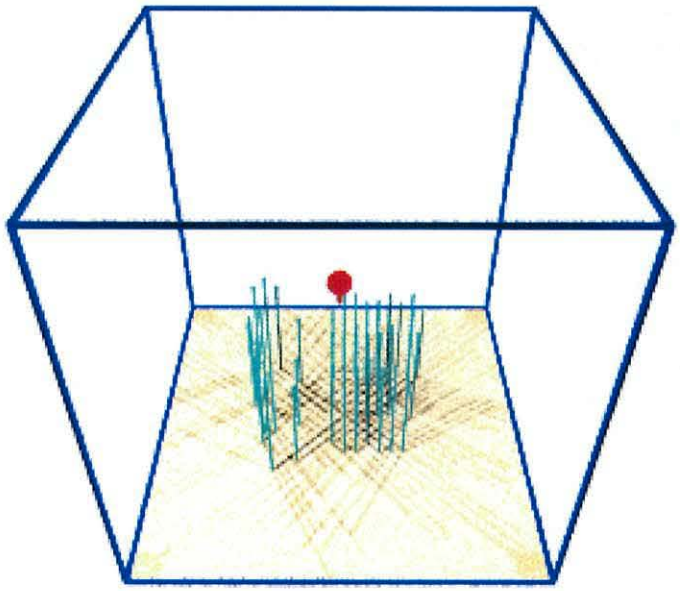


Figure 8.57: Map 17 calculated during experiment 3

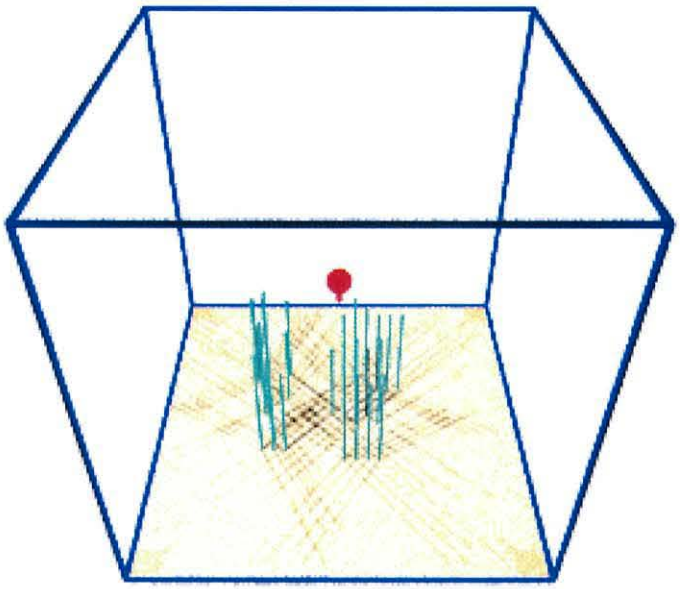


Figure 8.58: Map 24 calculated during experiment 3

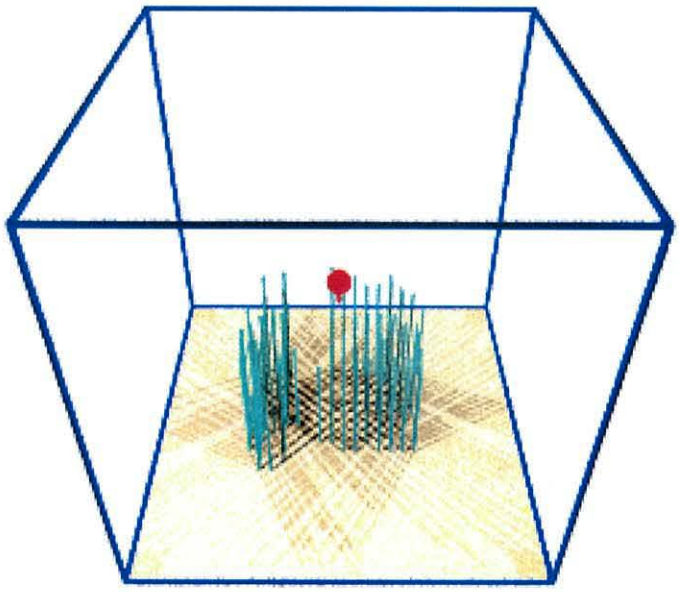


Figure 8.59: Map 31 calculated during experiment 3

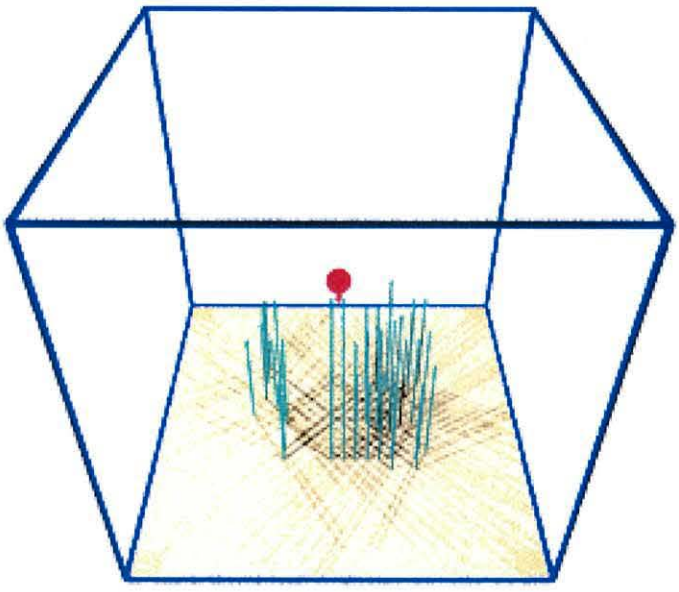


Figure 8.60: Map 37 calculated during experiment 3

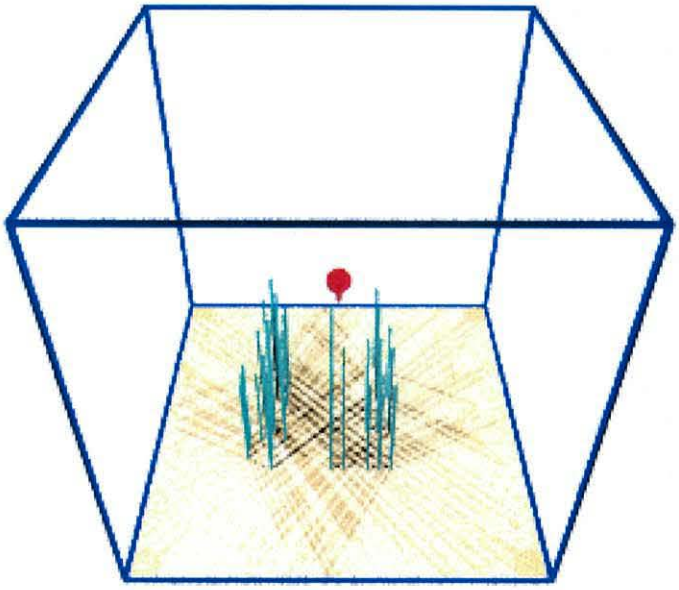


Figure 8.61: Map 40 calculated during experiment 3

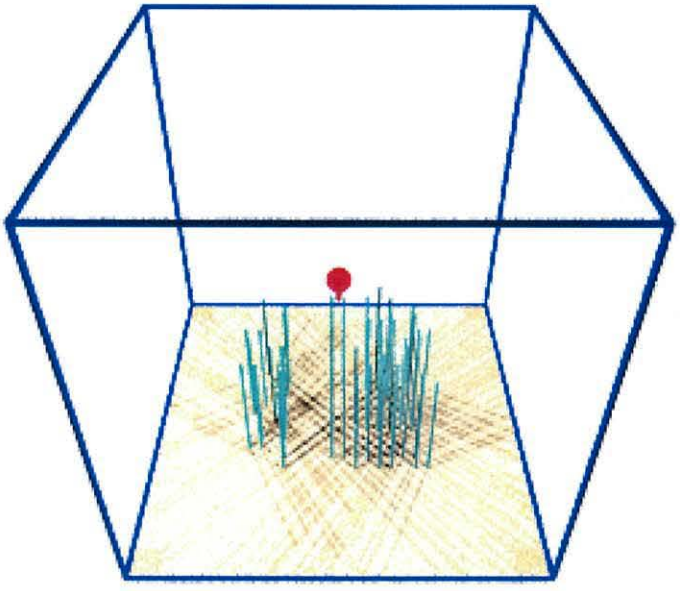


Figure 8.62: Map 46 calculated during experiment 3

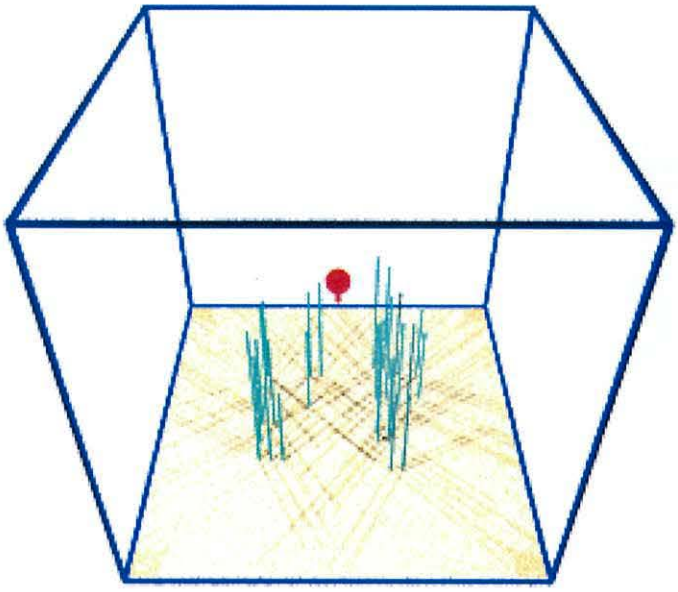


Figure 8.63: Map 50 calculated during experiment 3

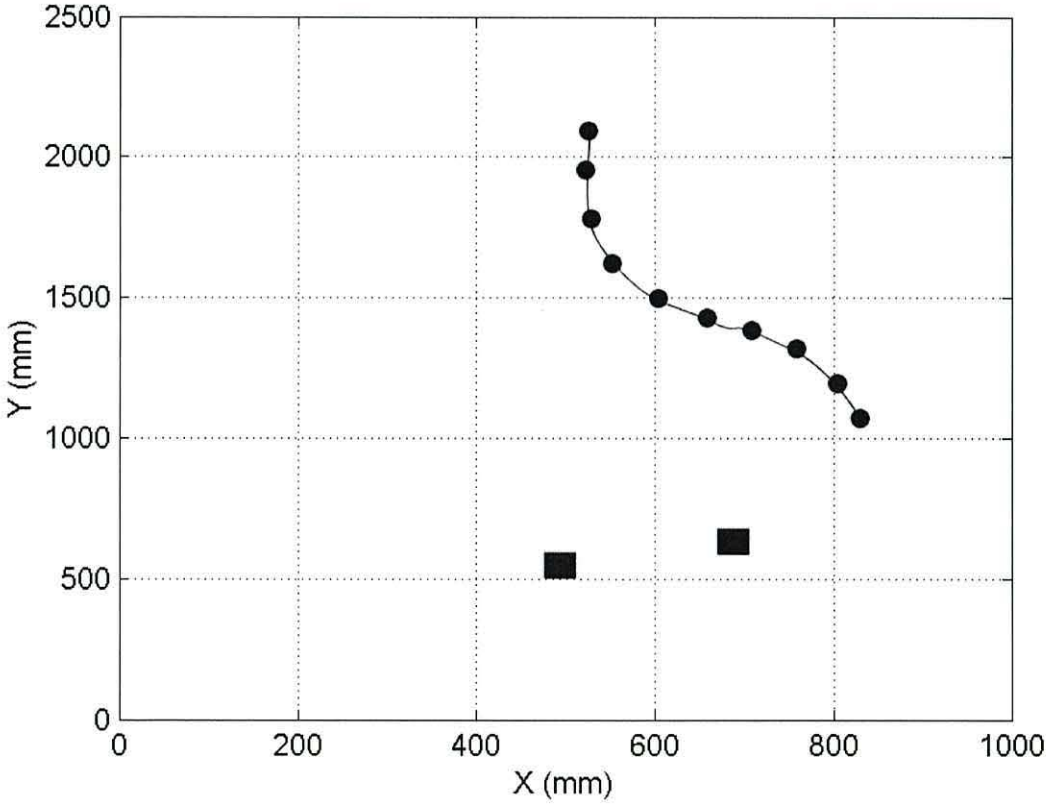


Figure 8.64: The path followed during experiment 3

8.2.4 Experiment 4

Figure 8.65 shows some of the images captured during the experiment. In this experiment, which has a starting point similar to the previous one, the path generated in completely different. The images below this time show a left to right movement.

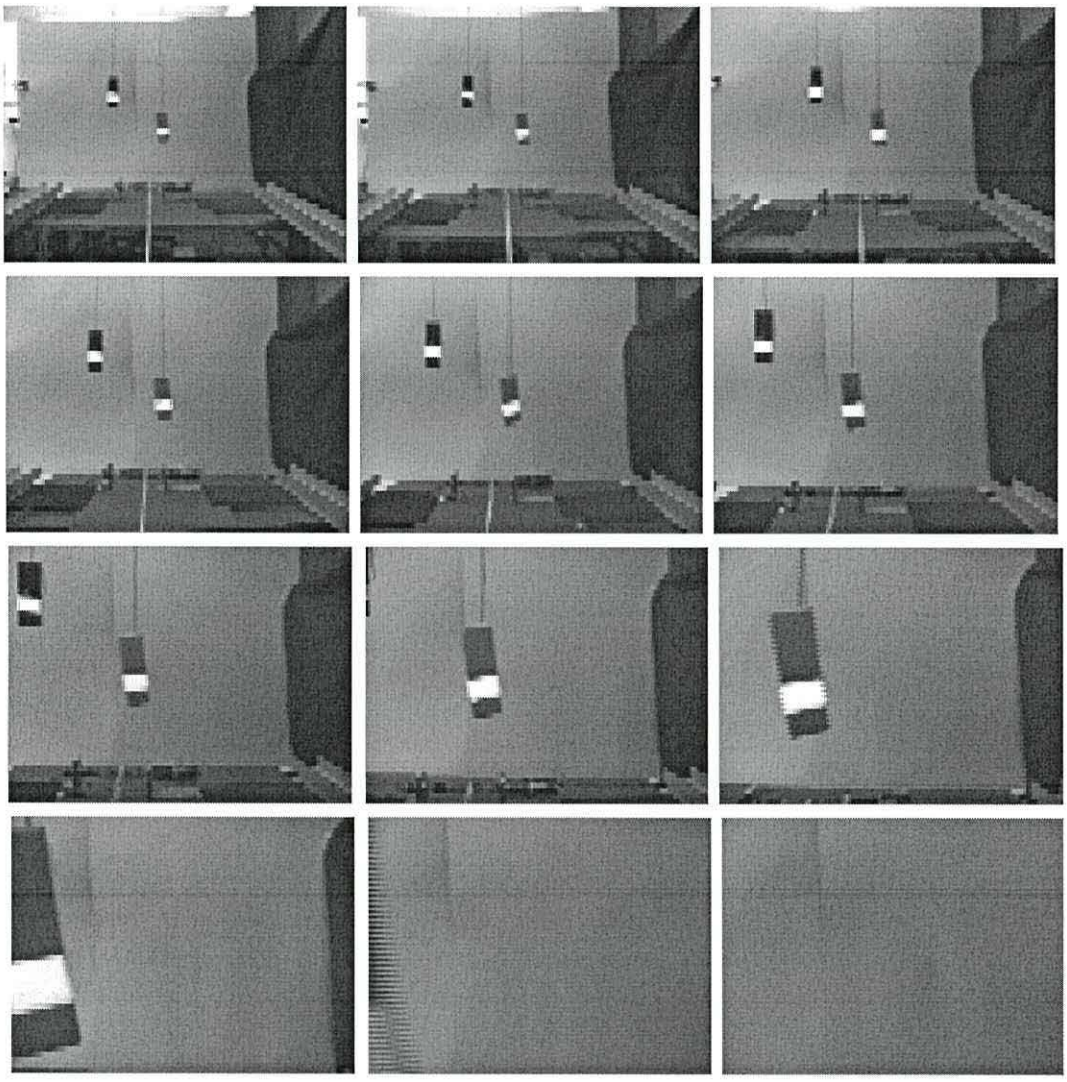


Figure 8.65: Images captured during experiment 4

The flows generated from the image sequences above are shown in figure 8.66. These show that the camera does come close to one of the obstacles, the flow vectors are very large in the final few flow fields. In this experiment the details noticed on the background in previous experiments are not as prominent,

compared with the obstacles.

The maps produced from these flow fields are shown in figures 8.67 to 8.77. These clearly show the obstacles coming close to the camera and then disappearing out of view.

The path generated by this run is given in figure 8.78. The camera mount did come close to the obstacles but it did not collide with it. This path shows the camera moving the opposite direction to that in experiment 3 although the starting point and the obstacle locations are similar. Also notice that the software does not crash in this case.

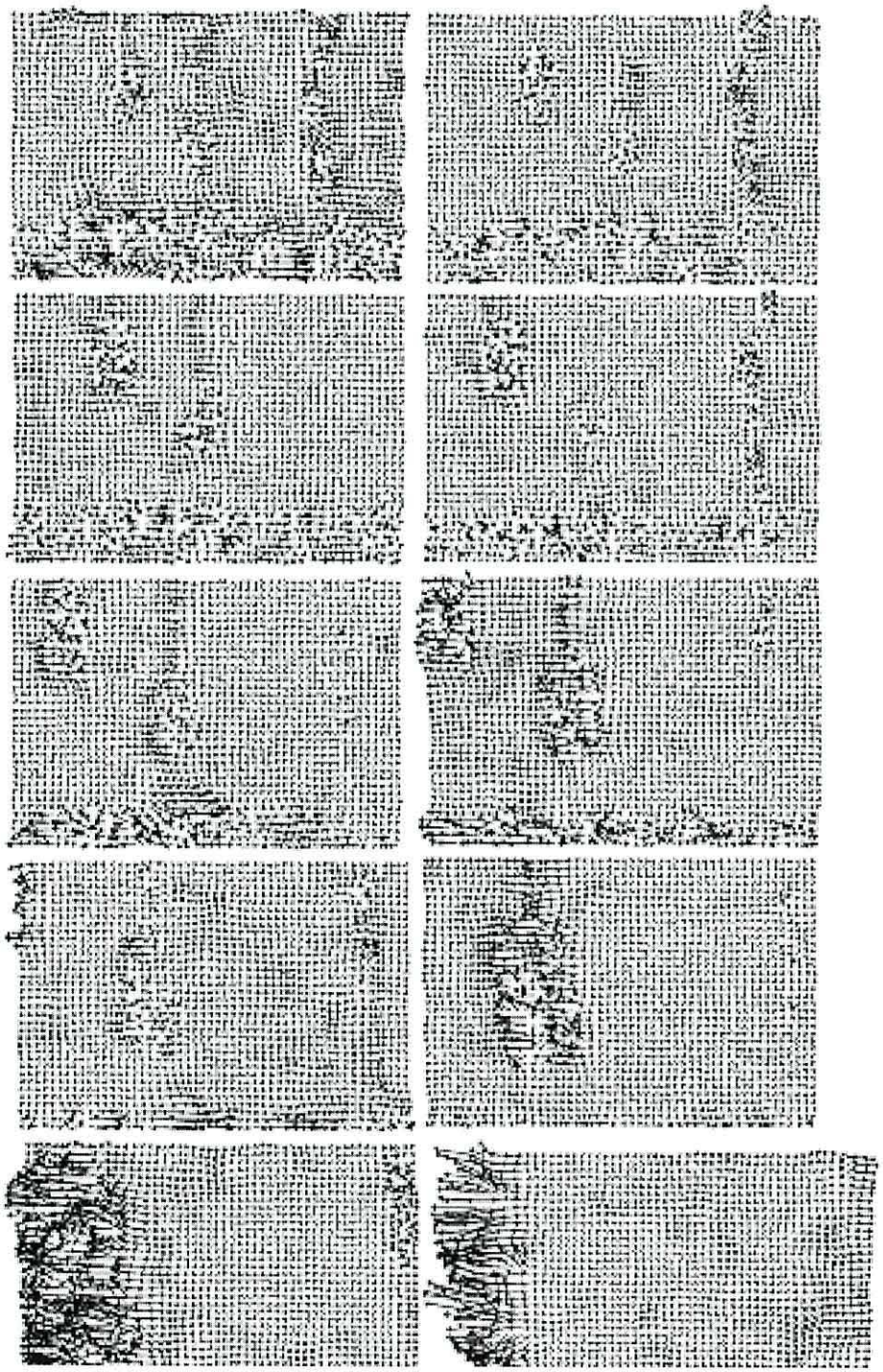


Figure 8.66: Flow fields generated during experiment 4

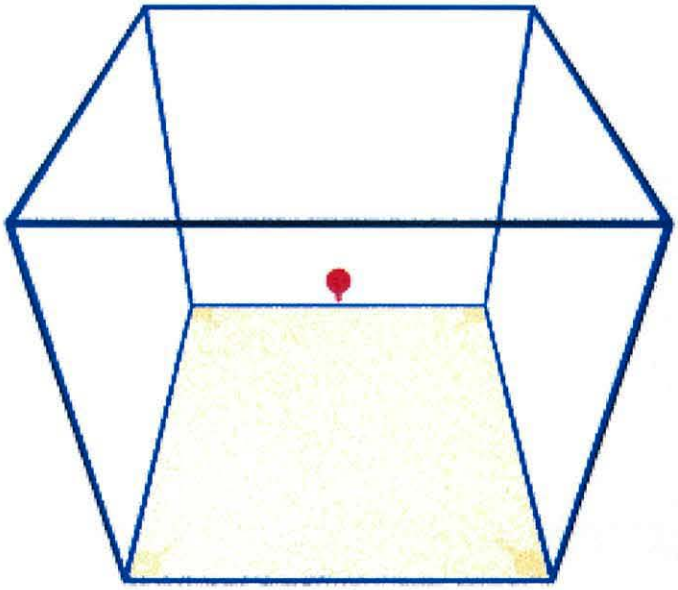


Figure 8.67: Map 1 calculated during experiment 4

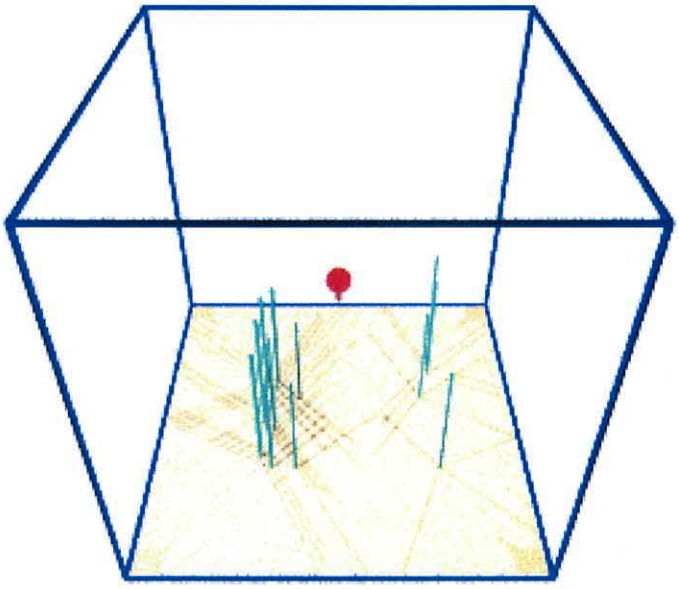


Figure 8.68: Map 5 calculated during experiment 4

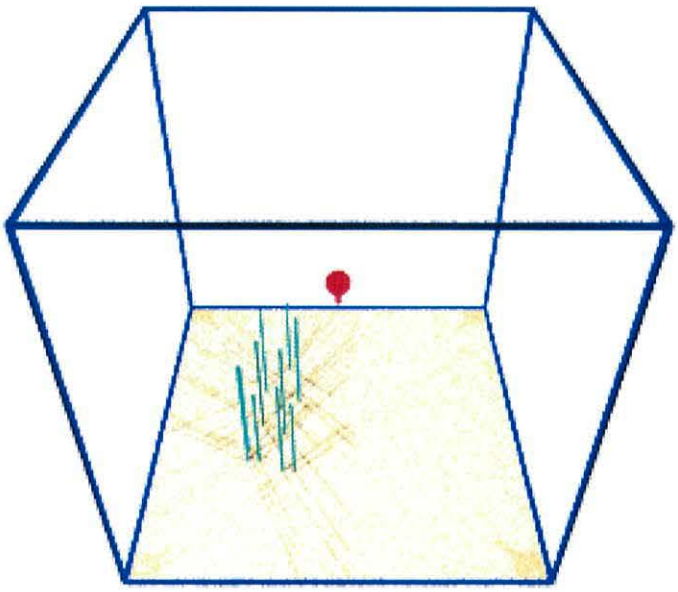


Figure 8.69: Map 10 calculated during experiment 4

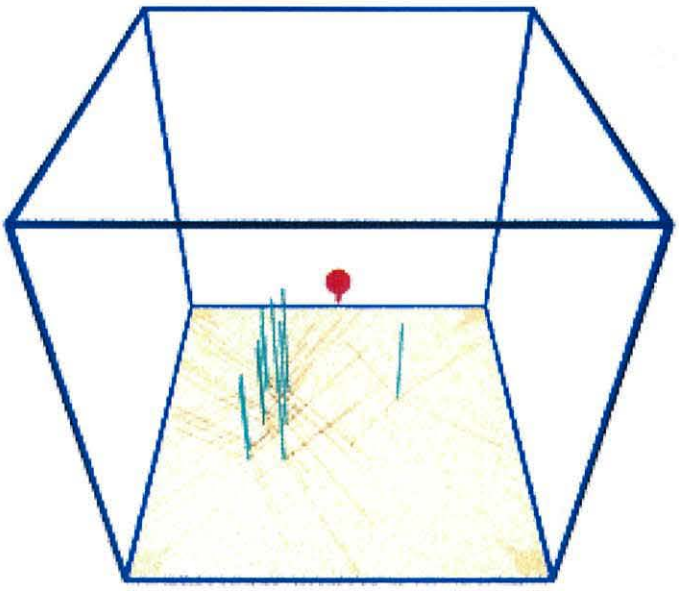


Figure 8.70: Map 15 calculated during experiment 4

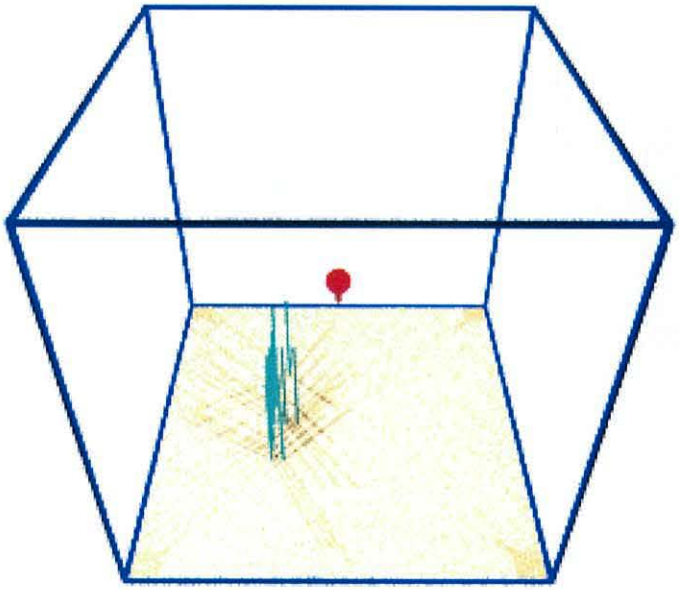


Figure 8.71: Map 20 calculated during experiment 4

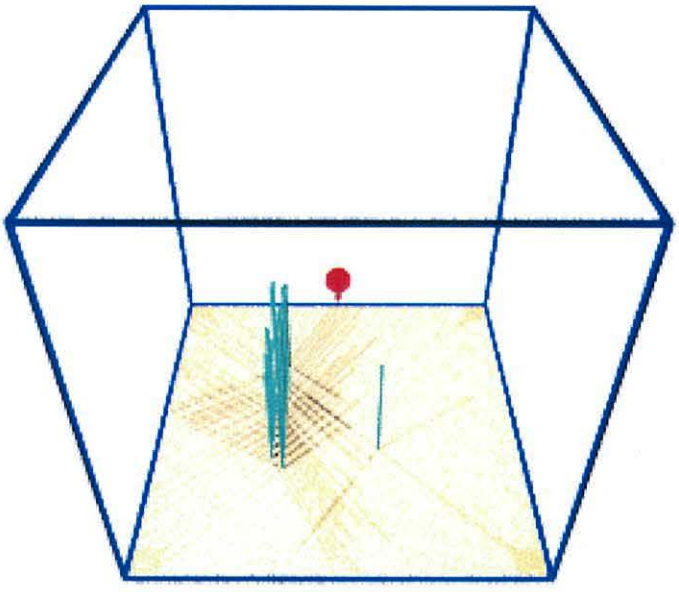


Figure 8.72: Map 25 calculated during experiment 4

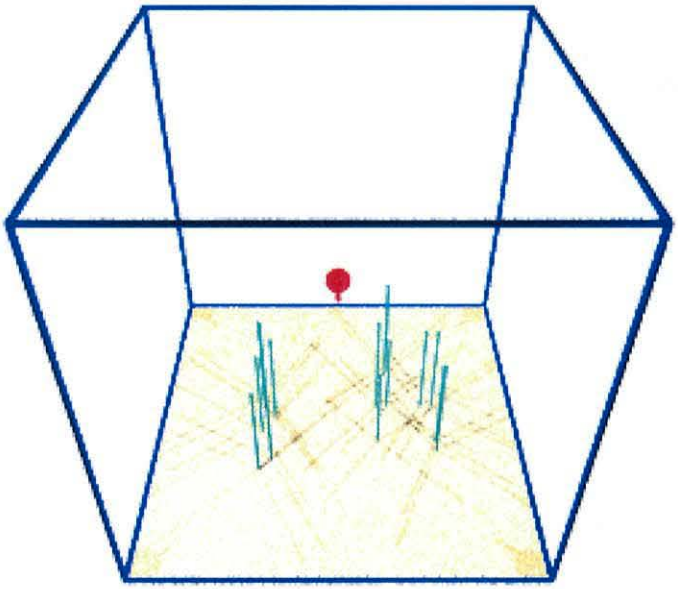


Figure 8.73: Map 30 calculated during experiment 4

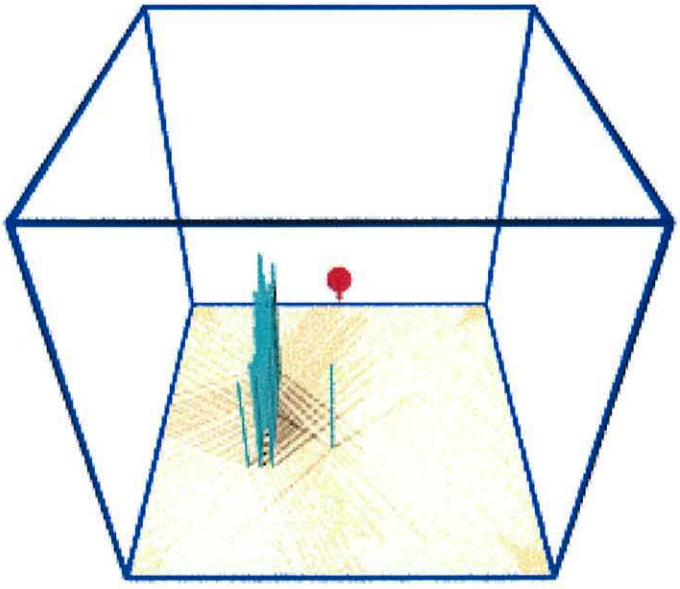


Figure 8.74: Map 35 calculated during experiment 4

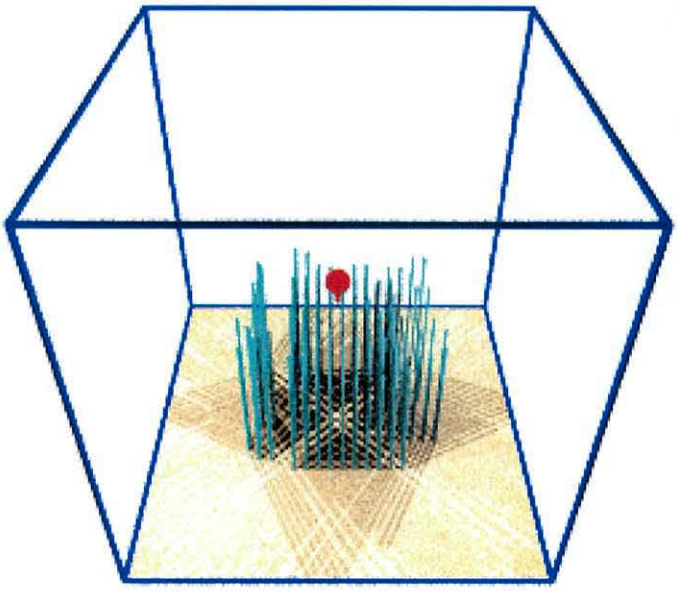


Figure 8.75: Map 40 calculated during experiment 4

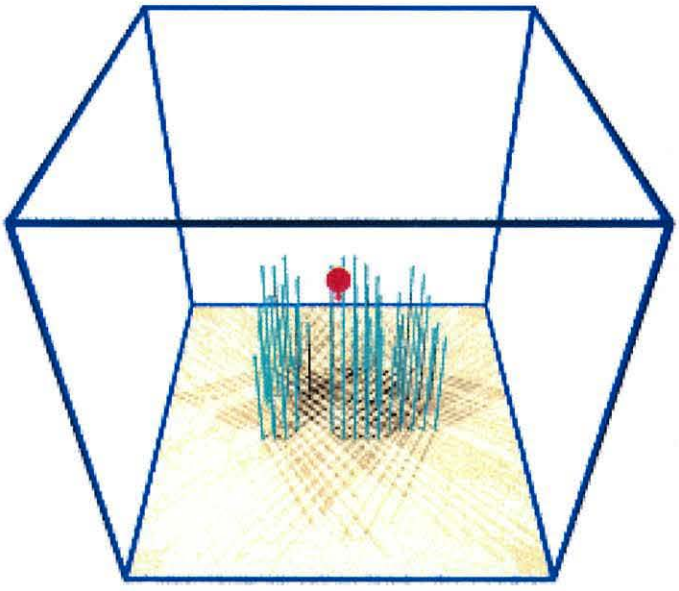


Figure 8.76: Map 50 calculated during experiment 4

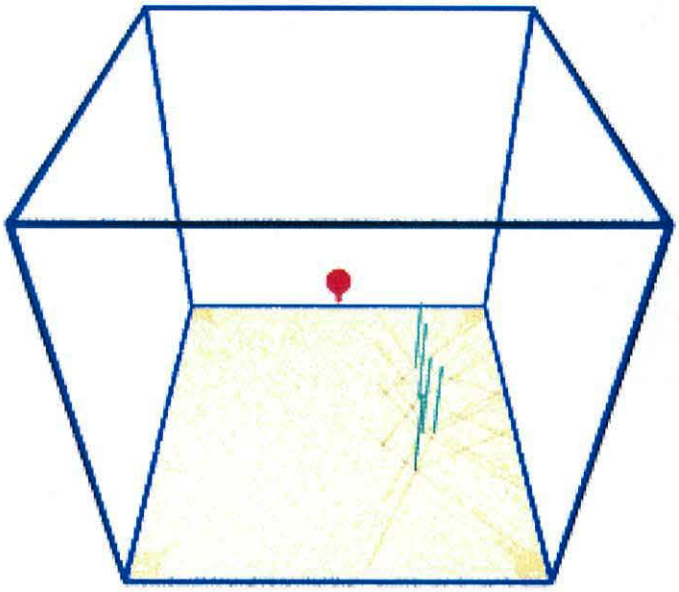


Figure 8.77: Map 53 calculated during experiment 4

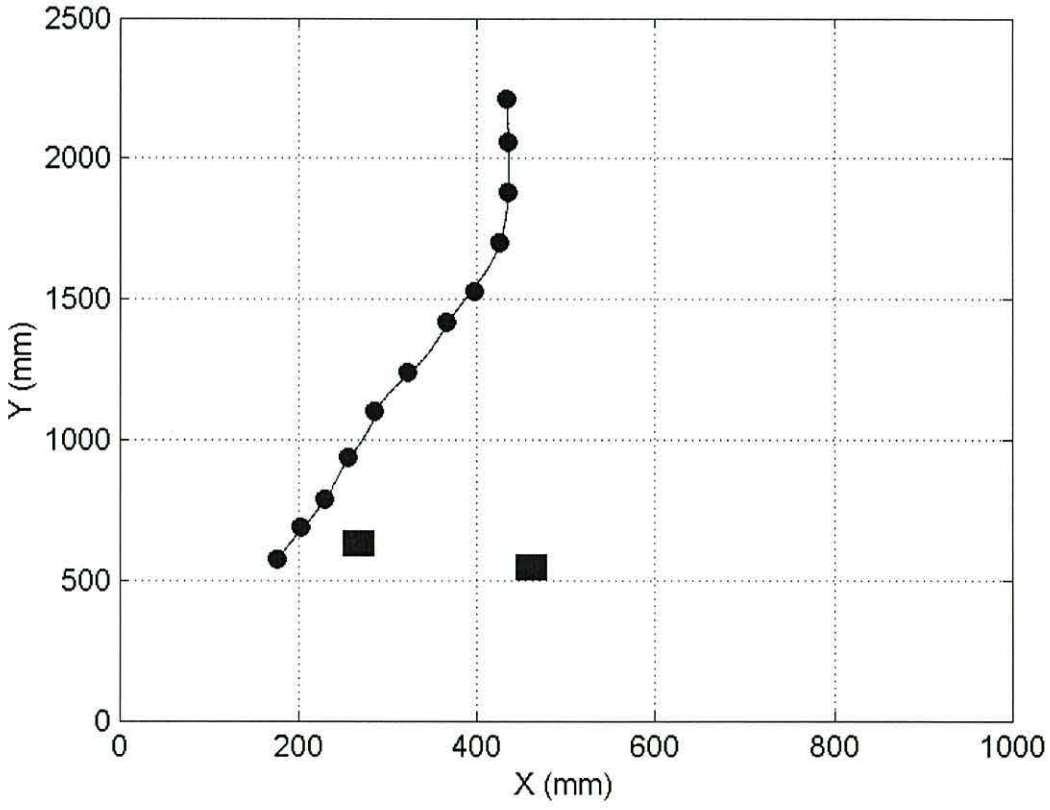


Figure 8.78: The path followed during experiment 4

8.3 Discussion and Conclusions

The aim of this project was to investigate whether collision avoidance could be achieved using a machine vision based obstacle detection and location system providing information for a path planner. The results from the test rig experiments presented above show that obstacle detection and location is possible using PC based software. Path planning using the distance transform was presented in chapter 5. The test rig experiments do show that combination of these two systems makes obstacle avoidance possible.

The "Achilles Heel" of the system is the map builder. The optical flow method assumes that light levels remain constant between images but this was not the case in many of the experimental runs that were performed because of varying daylight conditions in the laboratory. Once the optical flow results are compromised, the rest of the location system fails. It would have been possible to construct the test rig in a dark room with strictly controlled lighting, but the fact that the lighting levels affect performance would have to be faced at some point.

A more realistic remedy to the problem would be to use a more advanced camera and lens system. The camera currently used on the test rig has a fixed focal length lens with manual aperture and focus control. Adding an auto-iris and auto-focus would probably contribute greatly to the consistency of the results. However, at the time of purchase, this would have been beyond the project's budget. Also, the inclusion of an advanced lens could have introduced its own set of problems, focusing on the background and not the obstacles for example could result in the obstacle being missed initially leaving less time for the avoidance system to generate a path around it.

The results presented here do show that obstacle avoidance is possible. This implementation uses optical flow to estimate obstacle location in space that under controlled conditions will produce accurate results. However, the results from the test rig show that optical flow on its own would not be acceptable as a location system. Therefore it is necessary to continue research into methods of obstacle location, but always with the constraints that the resulting system should be "cheap", easily assembled using standard parts and be able to fit into a small air

frame.

The next chapter will conclude the thesis. A summary of the work preformed together with the achievements of the project will be given. Areas for improvement will be discussed and recommendations for further work will be outlined.

Chapter 9

Discussion and Conclusions

The aim of this project was to investigate machine vision and path planning methods for use in an autonomous unmanned air vehicle. This chapter begins with a summary of the work performed and its achievements. The chapter continues by discussing the outcomes of the project with reference to the original aims and objectives. A review of the key research areas is presented, highlighting the contributions of this work to progress in the field. Finally, recommendations for further research are made, showing how the project could be taken forward.

9.1 Hierarchical Controller

The machine vision and path planning methods which constitute the main technical investigation of this thesis must exist within a defined framework if they are to be effectively employed for controlling an autonomous air vehicle. Chapter 3 discusses a suitable controller hierarchy which divides responsibilities between three levels, the *planner*, the *Navigator* and the *Pilot*. The majority of the thesis has been aimed at implementing the *Pilot* level as this is the "real-time" component of the system that controls the helicopter.

The *Planner* and *Navigator* functions are employed before the helicopter leaves the ground. The *Planner* generates a Preferred Flight Space (PFS), which is

refined into a Modified Preferred Flight Space MPFS - a volume of space where the threat of collision with known obstacles is below a certain level.

The *Navigator* then uses the MPFS as its workspace and produces a flight path along which the best inspection results can be obtained. This ensures that the flight path is safe and can give the good quality inspection results.

The nominal flight path generated by the *Planner* and *Navigator* is followed by the *Pilot* unless an unknown obstacle is detected.

Finally, the *Pilot* level performs the key tasks of obstacle detection and location, and path planning.

While the generation of the nominal flight path from geographical data has not been considered in detail, it is nevertheless a contribution of this work that the principle of how this cam be done has been established. Further, its relationship to autonomous guidance functions is given by the hierarchical architecture and provides a defined route to implementation.

9.2 Machine Vision

Given that other sensors exist, such as RADAR which can give range and velocity estimates, it is not obvious at first sight why machine vision should be considered in this application. However, chapter 2 provides a rationale for doing so in economic and practical terms. By and large, the results of this work support this view because it is shown that optical flow based methods can yield the same type of results as RADAR (although perhaps not the same quality).

Chapter 4 proposes that two different types of detection systems are required. The first is used when a fast moving object such as a low flying jet comes into the locality of the helicopter. By the use of time to collision or looming, a measure of the velocity can be obtained allowing a reflex avoidance plan to be triggered. This function has not been considered during this project.

The second type of detection is used when the threat level is not so severe. In this mode, time can be spent determining the position of the obstacle so that a controlled manoeuvre can be executed which maintains the inspection process.

The machine vision system investigated here uses Anandan's optical flow method. This was chosen as it produces a dense flow field and in addition only operates on two frames at a time. This particular method was chosen over other methods after reviewing comparisons within the literature. In addition the software source code was freely available. The code was reviewed and tested using standard publicly available test images and images captured from the test rig. The software was then adapted for use within the controller allowing the machine vision system to run in parallel with the other software components.

9.3 Path Planning

In order to avoid the obstacles detected by the machine vision system path planning is required. Chapter 2 introduced a variety of existing path planning methods. The decision was made to use a path planner based on the Distance Transform, which can be thought of as a combination of a cell decomposition method and a potential field method. The advantages of both types (cell decomposition methods are useful for higher dimensional problems and potential methods tend to produce fast planners) have been incorporated into the method.

The resulting path planner operates in three dimensions, is rapid and does not suffer from the problems of false minima which are inherent in the potential method. Chapter 5 gives a detailed account of the Distance Transform based path planner.

The use of spatial decomposition of a three dimensional workspace into an octree was key to the success of the path planner. This is because decomposition of space acts like a dimension reducing device. In other words, it is as though the path planner is operating on a two dimensional workspace. However, there is an overhead in processing to convert the workspace into a tree structure and it has been shown that for the process to be efficient, the workspace needs to be greater

than 150 units in size or the obstacles need to be few and large.

As mentioned in the previous section there are two types of detection and consequently there are two modes of avoidance. It is true that path planning method described here is rapid; nevertheless, in the case of a reflex response to the detection of a fast-moving object, it is preferable to follow a pre-defined path. It has therefore been proposed that, as the vehicle is moving along, a number of escape paths should be generated as part of the normal path planning process. This would only add a minimal amount of processing time to the overall path generation process because the path planner can easily handle multiple goals from one start point. Thus a number of escape paths can be continually generated and one of these put into action, dependent on the velocity of the approaching object.

9.4 Results

The experiments performed using the path planner are presented in chapter 6. The final three dimensional planner has been shown to be effective and rapid. The execution time ranges from 30ms for an empty workspace to 100ms for a complex workspace, and more importantly it takes only 60ms to determine that **no** path is possible for a complex workspace.

Chapter 7 has detailed the construction of a new computer controlled test rig. The mechanical design of the rig and the design and construction of the electronics was completed by the author. The controller software combines the machine vision software with the path planning system together with the necessary sub-systems to acquire images and control the position of the test rig.

The results of the optical flow investigation show that obstacle location is possible. The results presented at the beginning of chapter 8 show that the method can be used to determine the distance from a camera to an object, and from that to generate a obstacle map. This map can then be used for path planning.

The results from the test rig are also presented in chapter 8. The conclusion of this chapter is that obstacle avoidance using the system developed is possible, but

as the system stands the results are not reliable enough to proceed to a prototype stage.

9.5 Recommendations for further work

The recommendations for future work based on this project include:

- The Distance Transform path planner has worked well in three dimensions. It would be interesting to investigate how it operates in higher dimensions, for example in path planning for a multi-link robot arm in configuration space.
- The path planner is an ideal candidate for parallel or cluster computing with the tree structure being spread among the processors. Investigation of the potential speed-up to be obtained by parallelisation of the path planner, especially if using more than three dimensions would be an interesting research topic.
- Reflex avoidance systems have been introduced in this thesis, it would be interesting to develop these further and asses the possible contribution to improving the safety of the air vehicle.
- A specific recommendation for improving the test rig would be to add the third axis and incorporate position and velocity control on every axes. This would allow the helicopter to be modelled more closely and the possibility of implementing paths planned in three dimensions.
- Camera and lens technology has improved over the period of this project. An Automatic Gain Control (AGC) and automatic focus camera/lens combination would help in the acquisition of images by ensuring that lighting level fluctuations has less of an effect on the images. However, further work examining the effect of these improvements on the operation of the optical flow software is required to find out if there are any detrimental effects.

9.6 General Conclusions

This PhD project encompasses a large area of science and engineering. The work was therefore performed at several levels:

- At the application level, the requirements of power line inspection and the constraints of current aircraft regulations were analysed, leading to the idea of autonomous software to provide the "See and Be Seen" function.
- At the organisational level, the hierarchical architecture was introduced as a suitable framework into which the low level functions could be embedded.
- At a detailed technical level, methods for obstacle detection and path planning were considered.

It was impossible to explore every area in depth within the time frame of a PhD project. The machine vision system is the crucial technical problem. Selection of a suitable method was done by reviewing the literature, especially papers giving comparisons between the different types. Also, the software code was obtained from a publicly available Internet site. It would have been preferable to investigate machine vision in much more depth and to develop software from scratch so that it was customised for use within the project. However, this was never going to be feasible.

The results from the test rig were not of a consistent standard. The results presented in chapter 8 do show obstacle avoidance in action. However, map building using optical flow has demonstrated the sensitivity of the software to brightness fluctuations between images. More work is required to determine whether machine vision system can provide obstacle detection and location with the same degree of accuracy and reliability as other sensors.

As a whole the PhD experience has been a profitable one. In particular the development of a distance transform based path planner for three dimensional use had not been reported before and this work has therefore contributed to the literature on path planning methods.

Appendix

Appendix A

CAA Regulations

This appendix is provided to show the main guidelines set out for small (model) aircraft in CAP 658 [21]. The important items are highlighted in **bold face**.

1. Any model aircraft
    - Choose an unobstructed site and at all times keep a safe distance from
        - Persons
        - Vessels
        - Vehicles
        - Structures
    - Only fly
        - In suitable weather
        - With regard to any other conditions such as local bylaws
        - With due consideration for other people and property
2. Models weighing 7Kg to 20Kg
    - Should only be flown
        - When the weather is suitable
        - Clear of controlled airspace unless with Air Traffic Control (ATC) permission

  - Clear of any aerodrome traffic zones unless with ATC permission
  - Below 120m above ground level unless with ATC permission
  - **Within the sight of the operator at all times**
  - Well clear of any congested area of a city, town or settlement. Not closer than 150m is suggested
  - At least 50m clear of persons, vessels, vehicles or structures. This can be reduced to 30m for take off.
- and
  - A serviceable 'fail-safe' mechanism should be incorporated
  - Ensure that any load carried on the model is secure
  - Flights must comply with any other conditions or bylaws
  - **Authority permission is required for any commercial flights**

Appendix B

Approximate Cell Decomposition

B.1 General Description

In the following a rectangloid designates a closed region of the following form in a Cartesian space R^n:

$$\{(x_1, \ldots, x_n)/x_1 \in [x'_1, x''_1], \ldots, x_n \in [x'_n, x''_n]\} \tag{B.1}$$

Let A be a robot whose configuration space C is R^N, with N =2 or 3. A configuration q is represented by the coordinates of A's reference point O_A in the frame F_W attached to the workspace.

We assume that the set of possible positions of A is contained in a rectangloid $D \subset R^N$. We represent C_{free} as :

$$\mathcal{C}_{free} = R \backslash \mathcal{CB} \tag{B.2}$$

where CB is the C-obstacle region and:

$$R = int(D) \tag{B.3}$$

Let $\Omega = cl(R)$. It is a rectangloid of R^m, where m is the dimension of the configuration space $\mathcal{C}$.

A rectangloid decomposition $\mathcal{P}$ of Ω is a finite collection of rectangloids $k_{ii} = 1, \ldots, r$ such that:

- Ω is equal to the union of the k_i, ie:

$$\Omega = \bigcup_{i=1}^{r} k_i$$

- The interiors of the k_i's do not intersect, ie:

$$\forall i_1, i_2 \in [1, r], i_1 \neq i_2 : int(k_{i_1}) \cap int(k_{i_2}) = \emptyset$$

Each rectangloid k_i is called a cell of the decomposition $\mathcal{P}$ of Ω.

Two cells are adjacent if and only if their intersection is a set of non-zero measure in R^{m-1}.

A cell k_i is classified as:

- Empty, if and only if its interior does not intersect the C-obstacle region, ie $int(k_i) \cap \mathcal{CB} = \emptyset$
- Full, if and only if k_i is entirely contained in the C-obstacle region, ie $k_i \subseteq \mathcal{CB}$
- Grey, otherwise.

The connectivity graph associated with a decomposition $\mathcal{P}$ of Ω is the non-directed graph G as follows:

- The nodes of G are the empty and grey cells of $\mathcal{P}$.
- Two nodes of G are connected by a link if and only if the corresponding cells are adjacent

Given a rectangloid decomposition $\mathcal{P}$ of Ω, a channel is defined as a sequence $(k_{\alpha_j})_{j=1,\ldots,p}$ of empty and/or mixed cells such that any two consecutive cells k_{α_j}

and $k_{\alpha_{j+1}}$, $j \in [1, p-1]$, are adjacent. A channel that contains at least one grey cell is called a G-channel, a channel containing only empty cells is called an E-channel.

Given an initial configuration $q_{init} \in \mathcal{C}_{free}$ and goal configuration $q_{goal} \in \mathcal{C}_{free}$, the problem is to generate an E-channel $(k_{\alpha_j})_{j=1,\ldots,p}$, such that $q_{init} \in k_{\alpha_1}$ and $q_{goal} \in k_{\alpha_p}$. If such a channel is generated, let $\beta_j = \partial k_{\alpha_j} \cap \partial k_{\alpha_{j+1}}, j = 1, \ldots, p-1$, be the intersection of the boundaries of two successive cells. A free path joining the initial to the goal configuration can be extracted from the E-channel by linking q_{init} to q_{goal} by a polygonal line whose vertices are points $Q_j int(\beta_j)$.

Appendix C

Potential Field Methods

C.1 Description of potential function

Most potential methods use the idea that the robot should be attracted to a goal and repulsed by the obstacles. This section will introduce the basic method where the robot $\mathcal{A}$ translates freely at a fixed orientation in a workspace $\mathcal{W} = R^N$, where N=2 or 3, ie $\mathcal{C} = R^N$.

The field of force $\vec{F}(q)$ in $\mathcal{C}$ is produced by differentiable potential function U : $\mathcal{C}_{free} \rightarrow R$, with:

$$\vec{F}(q) = -\vec{\nabla}U(q) \tag{C.1}$$

where $\vec{\nabla}U(q)$ denotes the gradient vector U at q. In $\mathcal{C} = R^N$ (N=2 or 3), we can write $q = (x, y)$ or (x, y, z) and:

$$\vec{\nabla}U = \begin{pmatrix} \partial U/\partial x \\ \partial U/\partial y \end{pmatrix} \text{ or } \begin{pmatrix} \partial U/\partial x \\ \partial U/\partial y \\ \partial U/\partial z \end{pmatrix} \tag{C.2}$$

In order to make the robot attracted towards a goal and repulsed by obstacles,

U is constructed as the sum of two functions.

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{C.3}$$

U_{att} is the attractive potential associated with the goal configuration q_{goal}. U_{rep} is the repulsive potential associated with the C-obstacle region. The attractive potential is independent of the obstacles, and the repulsive potential is independent of the goal configuration.

C.1.1 The attractive potential

The simplest definition of U_{att} is a parabolic well:

$$U_{att}(q) = \frac{1}{2}\xi\rho^2_{goal}(q) \tag{C.4}$$

where ξ is a positive scaling function and $\rho_{goal}(q)$ denotes the Euclidean distance $\|q - q_{goal}\|$. The function U_{att} is positive, except at the goal, where $U_{att}(q_{goal}) = 0$.

C.1.2 The repulsive potential

The repulsive potential creates a barrier around the C-obstacle region, that should not be traversed by the robot's configuration. However, it is desirable that the obstacles potential field does not affect the motion of the robot when it is significantly distant from the C-obstacles. One method to achieve this is to create the repulsive potential function as follows:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q)} + \frac{1}{\rho_0}\right)^2 & \text{to} \quad \rho(q) \leq \rho_0, \\ 0 & \text{to} \quad \rho(q) \geq \rho_0 \end{cases} \tag{C.5}$$

where η is a positive scaling faction, $\rho(q)$ denotes the distance from q to the C-obstacle region $\mathcal{CB}$, ie:

$$\rho(q) = \min_{q' \in \mathcal{CB}} \|q - q'\| \tag{C.6}$$

and ρ_0 is a positive constant called the distance of influence of the C-obstacles. The function U_{rep} is positive or zero, tends to infinity as q approached the C-obstacle region, and is zero when outside the influence of the C-obstacle region.

C.2 Potential Guided Path Planning

This section will describe simple potential-guided path planning techniques, which operate on the above potential function and others that have been described in the literature.

In its original conception, the potential field approach to motion generation consists of regarding the robot in the configuration space as a unit mass particle moving under the influence of the force field $\vec{F} = -\vec{\nabla}U$. At every configuration q the artificial force $\vec{F}(q)$ determines the acceleration of the particle.

C.2.1 Depth-first planning

This technique constructs a path as the produce of successive path segments starting at the initial configuration q_{init}. Each segment is oriented along the negated gradient of the potential function computed at the configuration attained by the previous segment.

Let q_i and q_{i+1} be the origin and extremities of the i^{th} segment in the path. Let $x_j(q_i, j = 1, \ldots, m$, be the coordinates of q_i in some chart (U, ϕ). We define the inner product in the tangent space $T_q(\mathcal{C})$ so that the basis β induced by this chart in $T_q(\mathcal{C})$ is orthonormal. We than have:

$$\left[\vec{F}\right]_\beta = -\left[\vec{\nabla}U\right]_\beta = (-\partial U/\partial x_1 \ldots - \partial U/\partial x_m)^T \tag{C.7}$$

We denote the components of the unit vector $\vec{t}(q_i) = \vec{F}(q_i)/\|\vec{F}(q_i)\|$ in β by $t_j(q_i)$. The coordinates of the configuration q_{i+1} attained at the i^{th} iteration, (U, ϕ), are:

$$x_j(q_{i+1}) = x_j(q_i + \delta_i t_j(q_i), j = 1, \ldots, m \tag{C.8}$$

with δ_i denoting the length of the i^{th} increment.

For example, if $\mathcal{A}$ is a planar object moving in $\mathcal{W} = R^2$, we can parameterise any q by $(x_1, x_2, x_3) = (x, y, \theta) \in R^2 \times [0, 2\pi)$, with x and y being the coordinates of $\mathcal{A}$'s reference point $\mathcal{O}_\mathcal{A}$ at q, and θ being the angle (modulo 2π) between the x-axes of the frames $\mathcal{F}_\mathcal{W}$ and $\mathcal{F}_\mathcal{A}$ attached to $\mathcal{W}$ and $\mathcal{A}$, respectively. Then:

$$x(q_{i+1}) = x(q_i) + \delta_i \frac{\partial U}{\partial x}(x, y, \theta), \tag{C.9}$$

$$y(q_{i+1}) = y(q_i) + \delta_i \frac{\partial U}{\partial y}(x, y, \theta), \tag{C.10}$$

$$\theta(q_{i+1}) = \theta(q_i) + \delta_i \frac{\partial U}{\partial \theta}(x, y, \theta) \mod 2\pi \tag{C.11}$$

In order to "normalise" the displacements along the θ-axis relative to displacements along the x- and y-axis, on may parameterise q by $(x, y, \phi) \in R^2 \times [0, 2\pi R)$, by posing $\phi = \theta R$, with $R = \max_{a \in \partial\mathcal{A}} \|\mathcal{O}_\mathcal{A} - a\|$ being the maximal distance between the reference point $\mathcal{O}_\mathcal{A}$ and $\mathcal{A}$'s boundary. This yields:

$$\phi(q_{i+1}) = \phi(q_i) + \delta_i \frac{\partial U}{\partial \phi}(x, y, \phi) \mod 2\pi R \tag{C.12}$$

$$\theta(q_{i+1}) = \theta(q_i) + \frac{\delta_i}{R} \frac{\partial U}{\partial \phi}(x, y, \theta R) \mod 2\pi \tag{C.13}$$

This technique simply follows the steepest slope of the potential function until the goal is obtained. Unfortunately, local minima may be present in the potential

field which act like false goals, trapping the robot in a well which using the simple technique above it would never escape from. Escaping from local minima in a depth first search is not straight forward, The first problem is to detect that the robot is in a false minima, as motions is discretised, the planner does not stop exactly at the zero-force configuration. The second problem is escaping from the local minima, one approach would be to move a given distance at a certain orientation before resuming depth-first planning.

C.2.2 Best-first planning

Let us place a grid over the configuration space $\mathcal{C}$, we donate this grid by $\mathcal{GC}$. $\mathcal{GC}$ can be defined by considering a single chart over $\mathcal{C}$ and discretising each of the m corresponding coordinate axes. For example, if $\mathcal{A}$ is a free-flying object in $\mathcal{W} = R^2$, the grid consists of the configurations $(k_x\delta_x, k_y\delta_y, k_\theta\delta_\theta)$, with $k_x, k_y, k_\theta \in Z$ and modulo 2π arithmetic on θ.

Given a configuration q in the m-dimensional grid $\mathcal{GC}$, its p-neighbours ($1 \leq p \leq m$) are defined as all the configurations in $\mathcal{GC}$ having at most p coordinates differing from those of q, the amount of difference being exactly one increment in absolute value. There are $2m$ 1-neighbours, $2m^2$ 2-neighbours, ..., and $3^m - 1$ m-neighbours. Here we consider two configurations of $\mathcal{GC}$ to be neighbours if and only if they are p-neighbours for a predefined $p \in [1, m]$. To simplify the following algorithm:

- Both q_{init} and q_{goal} are configurations in $\mathcal{CG}$.
- If two neighbours in $\mathcal{GC}$ are in frees pace, the straight line segment connecting them in R^m also lies in free space.
- The grid $\mathcal{GC}$ is bounded and forms a rectangloid

The best-first techniques consists of iteratively constructing a tree T whose nodes are configurations in $\mathcal{GC}$. The root of T is q_{init}. At every iteration, the algorithm examines the neighbours of the leaf of T that has the lowest potential value,

retains the neighbours not already in T at which the potential function is less than some large threshold M, and installs the retained neighbours in T as successors of the current leaf. The algorithm terminates when the goal configuration is reached, or when the free subset of $\mathcal{GC}$ accessible from the initial configuration has been fully explored and no path to goal has been found.

This procedure follows a discrete approximation of the negated gradient of the potential function until it reaches a local minimum. When this happens, it operates in a way that corresponds to filling the well until a saddle point is attained.

This algorithm is guaranteed to return a free path when one exists, and to report when no path is possible. The implementation presented by Latombe [29] gives the basic time complexity as $O(mr^m \log r)$. In practice this method is only suitable when m is small. For the case of a free-flying robot in two-dimensional space ($m = 3$), it provides a means for very fast and reliable path planning with grid resolutions of the order 256^3.

Appendix D

C++ Source code

D.1 Header Files

D.1.1 Node.h

```
#ifndef _node_h
#define _node_h


#include <fstream.h>
#include <stdio.h>

// Class node manages the tree structure
// All variables are private to the class

class node
{
private:
    int node_name;
    node *parent;
    int goal,start;
```

```
    int node_classification;
    int distance_transform;
    int start_x, start_y, end_x, end_y;
    node *sw_child;
    node *nw_child;
    node *ne_child;
    node *se_child;
public:
    node(node *parent_to_this, int father_node, int node_offset);
    ~node();
    void classify_node(int useage);
    int analyse_node(void);
    void set_size(int sx, int ex,int sy, int ey);
    int *give_region_coords(void);
    void set_nw(node *nw);
    void set_ne(node *ne);
    void set_se(node *se);
    void set_sw(node *sw);
    int give_node_name(void);
    int give_class(void);
    node *give_node(int node_number);
    node *give_parent(void) { return parent;}
    int give_goal(void) { return goal;}
    int give_start(void) {return start;}
    void set_goal(void);
    void set_start(void);
    void reset_goal(void);
    void reset_goal_above(void);
    void set_distance(int value);
    int give_distance(void);
    void print_coords(void);
};

#endif
```

D.1.2 Matrix.h

```
#ifndef _matrix_h
#define _matrix_h

#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include "node.h"

class matrix
{
private:
    int nr,nc;
    int *data;
public:
    matrix(int w,int h);
    ~matrix();
    void set_element(int i,int j,int value);
    int get_element(int i,int j);
    void fill_matrix(void);
    int give_matrix_size_x(void);
    int give_matrix_size_y(void);
    int analyse_region(int *coords,node *current);
    void show_matrix(void);
};

#endif
```

D.1.3 Queue.h

```
#ifndef _queue_h
#define _queue_h
```

```
#include <fstream.h>
#include <stdio.h>

#define MAXQUEUE 100

class queue
{
    private:
    int count;
    int front;
    int rear;
    int entry[MAXQUEUE];
    public:
    queue(void);
    void add(int data);
    int remove(void);
    int full(void);
    int empty(void);
    int size(void);
};

#endif
```

D.1.4 Image.h

```
// Image.h
// Header File

#ifndef _image_h
#define _image_h
#include <classlib\time.h>
#include <windows.h>
#include <mil.h>
```

```
#include "workspace.h"
#include <stdio.h>
#include <fstream.h>
#include <cstring.h>
#include <math.h>
#include "model.h"
#include "obstacle.h"
#include "flowfield.h"

#define IMAGEDIRECTORY "m:\\optical\\"

#define THRESHOLD_VALUE    200

//#define ACQUIRE_IMAGE_SIZE_X   128
//#define ACQUIRE_IMAGE_SIZE_Y   128

#define ACQUIRE_IMAGE_SIZE_X  128
#define ACQUIRE_IMAGE_SIZE_Y  96


#define PREWITT                 0
#define SOBEL       1

#define PGM         0
#define PBMP1       1
#define PBMP4       2
#define TIFF        3
#define FLIP        1
#define NOFLIP      0
#define CAM0        0
#define CAM1        1
#define CAM2        2
#define CAM3        3
#define PROC        4
#define GREY_LEVELS       256
```

```
#define MAXLINE             64

// Define focal length as 6mm
#define FOCAL_LENGTH 6.0
#define FLOWTHRESHOLD 0.1

// define the flip state of each camera
#define CAM0FLIPSTATE   NOFLIP
#define CAM1FLIPSTATE   FLIP
#define CAM2FLIPSTATE   NOFLIP
#define CAM3FLIPSTATE   FLIP


// The class definition of the image object
class image
{
 private:
  char *display_data;
  unsigned int *image_data;
  bool *binary_image_data;
  float *flow_field_u;
  float *flow_field_v;
  double focal_length;
  long x_size;
  long y_size;
  long length;
  int x_position;
  int y_position;
  int z_position;
  TTime *grab_time;
  int camera_number;
  bool flipstate;
  bool flowdone;
  int current_image_number;
  static int master_image_number=0;
```

```
  obstacle *obstacles[MAX_OBSTACLES];  // Array of type obstacle
  int obstacle_count;
  model *current_model;
  char rawfilename[30];
 public:
  image(model *helicopter);
  image(int image_size_x,int image_size_y,int camera_num,
   model *helicopter);
  ~image();
  long give_x_size(void) { return x_size; }
  long give_y_size(void) { return y_size; }
  long give_length(void) { return length; }
  char *give_raw_filename(void)
    { return rawfilename;}
  unsigned int give_element(long index)
    { return *(image_data+index); }
  void set_element(long index,int value)
    { *(image_data+index)=value;}
  unsigned int give_element(long x,long y)
    { return *(image_data+(y*ACQUIRE_IMAGE_SIZE_X)+x); }
  bool give_bin_element(long index)
    { return *(binary_image_data+index); }
  bool give_bin_element(long x,long y)
    { return *(binary_image_data+(y*ACQUIRE_IMAGE_SIZE_X)+x); }
  void acquire(MIL_ID VisionMilSystem,MIL_ID VisionMilDigitizer);
  void flipxy(void);
  void preprocess_image(void);
  void normalize(void);
  void lowpass(void);
  void highpass(void);
  void edgedetect(int mask_type);
  void threshold(int threshold_value);
  void obstacle_detection(void);
  void loadimage(void);
  void saveimage(int type);
```

```
  void saveraw(void);
  void loadflowfield(char *filename);
  bool doneflow() {return flowdone;}
  double give_focal_length(void) { return focal_length; }
  void locate_objects(void);
  int give_x_position(void) { return x_position; }
  int give_y_position(void) { return y_position; }
  int give_z_position(void) { return z_position; }
  long *give_obstacle_boundary(int obstacle_number)
    { return obstacles[obstacle_number]->give_boundary(); }
  int give_number_obstacles(void) { return obstacle_count;}
  int give_image_number(void) { return current_image_number; }
  unsigned int *give_image_data_pointer(void) { return image_data; }
  bool *give_bool_image_pointer(void) { return binary_image_data; }
  char *give_image_pointer(void) { return display_data; }
  void update_display_data(void);
  double *analyseflowfields(image *previousimage);
};

#endif
```

D.2 Selected Source Code

D.2.1 Function: Octree

```
// This functiom generates the tree structure
// from the workspace volume

void octree(node *current_node,volume *workspace)
{
  node *children[8];
  node *temp;
  node *store;
```

```
bool obstacle_flag,freespace_flag;
long sx,sy,sz,ex,ey,ez;
long x,y,z;
long *coord;

coord=current_node->give_coords();
sx=*(coord+0);
sy=*(coord+1);
sz=*(coord+2);
ex=*(coord+3);
ey=*(coord+4);
ez=*(coord+5);
obstacle_flag=false;
freespace_flag=false;

for(z=sz;z<ez+1;z++)
{
  for(y=sy;y<ey+1;y++)
  {
    for(x=sx;x<ex+1;x++)
    {
      if(workspace->get_element(x,y,z)==OBSTACLE)
   obstacle_flag=true;
      if(workspace->get_element(x,y,z)==FREESPACE)
   freespace_flag=true;
    }
  }
}

if(obstacle_flag==false && freespace_flag==false)
{
   exit(1);  // Image must include freespace or obstacle
   // if nothing found something went VERY wrong!
}
```

```
if(obstacle_flag==true && freespace_flag==false)
{
  // Current map is full of obstacles - we have hit the sea?????
  current_node->set_type(FULL);
  current_node->null_children();
}

if(obstacle_flag==false && freespace_flag==true)
{
  // Map is clear do nothing
  current_node->set_type(MT);
  current_node->null_children();
}

// Map contains obstacle and freespace so split using recursion
if(obstacle_flag==true && freespace_flag==true)
{
  // Map contains obstacles and freespace
  current_node->set_type(GREY);

  children[0]=new node((current_node->give_name()*10)+1,
        current_node,sx,sy,sz,sx+(((ex+1)-sx)/2)-1,
        sy+(((ey+1)-sy)/2)-1,sz+(((ez+1)-sz)/2)-1);
  current_node->set_child(1,children[0]);
  octree(children[0],workspace);

  children[1]=new node((current_node->give_name()*10)+2,
        current_node,sx+((ex+1)-sx)/2,sy,sz,ex,
        sy+(((ey+1)-sy)/2)-1,sz+(((ez+1)-sz)/2)-1);
  current_node->set_child(2,children[1]);
  octree(children[1],workspace);

  children[2]=new node((current_node->give_name()*10)+3,
        current_node,sx,sy+((ey+1)-sy)/2,sz,
        sx+(((ex+1)-sx)/2)-1,ey,sz+(((ez+1)-sz)/2)-1);
```

```
current_node->set_child(3,children[2]);
octree(children[2],workspace);

children[3]=new node((current_node->give_name()*10)+4,
      current_node,sx+((ex+1)-sx)/2,sy+((ey+1)-sy)/2,
      sz,ex,ey,sz+(((ez+1)-sz)/2)-1);
current_node->set_child(4,children[3]);
octree(children[3],workspace);

children[4]=new node((current_node->give_name()*10)+5,
      current_node,sx,sy,sz+((ez+1)-sz)/2,
      sx+(((ex+1)-sx)/2)-1,sy+(((ey+1)-sy)/2)-1,ez);
current_node->set_child(5,children[4]);
octree(children[4],workspace);

children[5]=new node((current_node->give_name()*10)+6,
      current_node,sx+((ex+1)-sx)/2,sy,
      sz+((ez+1)-sz)/2,ex,sy+(((ey+1)-sy)/2)-1,ez);
current_node->set_child(6,children[5]);
octree(children[5],workspace);

children[6]=new node((current_node->give_name()*10)+7,
      current_node,sx,sy+((ey+1)-sy)/2,
      sz+((ez+1)-sz)/2,sx+(((ex+1)-sx)/2)-1,ey,ez);
current_node->set_child(7,children[6]);
octree(children[6],workspace);

children[7]=new node((current_node->give_name()*10)+8,
      current_node,sx+((ex+1)-sx)/2,sy+((ey+1)-sy)/2,
      sz+((ez+1)-sz)/2,ex,ey,ez);
current_node->set_child(8,children[7]);
octree(children[7],workspace);

// Set local node neighbours
temp=current_node->give_child(1);
```

```
store=current_node->give_child(2);
temp->add_neighbours(store);
store=current_node->give_child(3);
temp->add_neighbours(store);
store=current_node->give_child(5);
temp->add_neighbours(store);

temp=current_node->give_child(2);
store=current_node->give_child(1);
temp->add_neighbours(store);
store=current_node->give_child(4);
temp->add_neighbours(store);
store=current_node->give_child(6);
temp->add_neighbours(store);

temp=current_node->give_child(3);
store=current_node->give_child(1);
temp->add_neighbours(store);
store=current_node->give_child(4);
temp->add_neighbours(store);
store=current_node->give_child(7);
temp->add_neighbours(store);

temp=current_node->give_child(4);
store=current_node->give_child(2);
temp->add_neighbours(store);
store=current_node->give_child(3);
temp->add_neighbours(store);
store=current_node->give_child(8);
temp->add_neighbours(store);

temp=current_node->give_child(5);
store=current_node->give_child(1);
temp->add_neighbours(store);
store=current_node->give_child(6);
```

```
    temp->add_neighbours(store);
    store=current_node->give_child(7);
    temp->add_neighbours(store);

    temp=current_node->give_child(6);
    store=current_node->give_child(2);
    temp->add_neighbours(store);
    store=current_node->give_child(5);
    temp->add_neighbours(store);
    store=current_node->give_child(8);
    temp->add_neighbours(store);

    temp=current_node->give_child(7);
    store=current_node->give_child(3);
    temp->add_neighbours(store);
    store=current_node->give_child(5);
    temp->add_neighbours(store);
    store=current_node->give_child(8);
    temp->add_neighbours(store);

    temp=current_node->give_child(8);
    store=current_node->give_child(4);
    temp->add_neighbours(store);
    store=current_node->give_child(6);
    temp->add_neighbours(store);
    store=current_node->give_child(7);
    temp->add_neighbours(store);
  }
  return;
}
```

D.2.2 Function: Diver

```
//Recursive search function
```

```
void diver(node *current_node,int type)
{
  node *temp;

  // Functions using the diver search
  void update1(node *current);
  void update2(node *current);

  // Start at root and pass through all nodes
  for(int i=1;i<9;i++)
  {
    // Test to see if current node has children
    temp=current_node->give_child(i);
    if(temp==NULL)
    {
      return;
    }
    else
    {
      // Recurse
      diver(temp,type);
    }

    // Call functions
    if(type==0) update1(temp);
    if(type==1) update2(temp);
  }
  return;
}
```

D.2.3 Function: Update1

This code has been edited for length.

```
void update1(node *current_node)
{
  node *temp;
  node *parent;
  node *store;
  int node_number;
  int node_id;

  // This will catch the root node only case
  if(current_node->give_child(1)==NULL) return;

  // Get the current name
  node_id=current_node->give_name();
  // Get the node location 1,,2,3,4,5,6,7,8
  node_number=node_id%10;

  // This statement connects the children of the current node
  // to the current node's sisters. Dependent on the location
  // of the current node
  switch(node_number)
    {
    case 0:
      break;
    case 1:  // Current node is top-left-front

      // Get the parent of the current node
      parent=current_node->give_parent();

      // Get the lower-left-front child of current node
      temp=current_node->give_child(2);
      // Get the parents lower-left-front child
      store=parent->give_child(2);
      // Add the parents llf child to the current
      // nodes llf child
      temp->add_neighbours(store);
```

```
// Add the current node llf child to the parent
// llf node child
store->add_neighbours(temp);

temp=current_node->give_child(3);
store=parent->give_child(3);
temp->add_neighbours(store);
store->add_neighbours(temp);

temp=current_node->give_child(4);
store=parent->give_child(2);
temp->add_neighbours(store);
store->add_neighbours(temp);
store=parent->give_child(3);
temp->add_neighbours(store);
store->add_neighbours(temp);

temp=current_node->give_child(5);
store=parent->give_child(5);
temp->add_neighbours(store);
store->add_neighbours(temp);

temp=current_node->give_child(6);
store=parent->give_child(2);
temp->add_neighbours(store);
store->add_neighbours(temp);
store=parent->give_child(5);
temp->add_neighbours(store);
store->add_neighbours(temp);

temp=current_node->give_child(7);
store=parent->give_child(3);
temp->add_neighbours(store);
store->add_neighbours(temp);
store=parent->give_child(5);
```

```
    temp->add_neighbours(store);
    store->add_neighbours(temp);

    temp=current_node->give_child(8);
    store=parent->give_child(2);
    temp->add_neighbours(store);
    store->add_neighbours(temp);
    store=parent->give_child(3);
    temp->add_neighbours(store);
    store->add_neighbours(temp);
    store=parent->give_child(5);
    temp->add_neighbours(store);
    store->add_neighbours(temp);
    break;
  case 2:
    parent=current_node->give_parent();

    temp=current_node->give_child(1);
    store=parent->give_child(1);
    temp->add_neighbours(store);
    store->add_neighbours(temp);

    temp=current_node->give_child(3);
    store=parent->give_child(1);
    temp->add_neighbours(store);
    store->add_neighbours(temp);
    store=parent->give_child(4);
    temp->add_neighbours(store);
    store->add_neighbours(temp);

    temp=current_node->give_child(4);
    store=parent->give_child(4);
    temp->add_neighbours(store);
    store->add_neighbours(temp);
```

```
      .........................
      .........................
      .........................

      temp=current_node->give_child(7);
      store=parent->give_child(7);
      temp->add_neighbours(store);
      store->add_neighbours(temp);
      break;
    default:
      cout << "ERROR: Default in Node Parse\n";
      break;
    }
  return;
}
```

D.2.4 Function: Update2

This code has been edited for length.

```
void update2(node *current_node)

{
  node *from_list;
  node *temp;
  int size,counter;

  // This will catch the root node only case
  if(current_node->give_child(1)==NULL) return;
  if(current_node->give_type()==FULL) return;

  counter=0;
  current_node->iter_restart();
  size=current_node->give_items();
```

```
while(counter<size)
  {
    from_list=current_node->next();
    if(from_list->give_child(1)==NULL)
 {
   switch(current_node->give_name()%10)
     {
     case 1:
       switch(from_list->give_name()%10)
    {
    case 1:
      break;
    case 2:
      from_list->add_neighbours(current_node->give_child(2));
      from_list->add_neighbours(current_node->give_child(4));
      from_list->add_neighbours(current_node->give_child(6));
      from_list->add_neighbours(current_node->give_child(8));
      break;
    case 3:
      from_list->add_neighbours(current_node->give_child(3));
      from_list->add_neighbours(current_node->give_child(4));
      from_list->add_neighbours(current_node->give_child(7));
      from_list->add_neighbours(current_node->give_child(8));
      break;
    case 4:
      break;
    case 5:
      from_list->add_neighbours(current_node->give_child(5));
      from_list->add_neighbours(current_node->give_child(6));
      from_list->add_neighbours(current_node->give_child(7));
      from_list->add_neighbours(current_node->give_child(8));
      break;
    case 6:
      break;
    case 7:
```

```
  break;
case 8:
  break;
}
   break;
 case 2:
   switch(from_list->give_name()%10)
{
case 1:
  from_list->add_neighbours(current_node->give_child(1));
  from_list->add_neighbours(current_node->give_child(3));
  from_list->add_neighbours(current_node->give_child(5));
  from_list->add_neighbours(current_node->give_child(7));
  break;
case 2:
  break;
case 3:
  break;
case 4:
  from_list->add_neighbours(current_node->give_child(2));
  from_list->add_neighbours(current_node->give_child(4));
  from_list->add_neighbours(current_node->give_child(6));
  from_list->add_neighbours(current_node->give_child(8));
  break;
case 5:
  break;
case 6:
  from_list->add_neighbours(current_node->give_child(5));
  from_list->add_neighbours(current_node->give_child(6));
  from_list->add_neighbours(current_node->give_child(7));
  from_list->add_neighbours(current_node->give_child(8));
  break;
case 7:
  break;
case 8:
```

```
  break;
}
   break;
 case 3:
   switch(from_list->give_name()%10)
{
case 1:
  from_list->add_neighbours(current_node->give_child(1));
  from_list->add_neighbours(current_node->give_child(2));
  from_list->add_neighbours(current_node->give_child(5));
  from_list->add_neighbours(current_node->give_child(6));
  break;

  ..................
  ..................
  ..................

 case 8:
   switch(from_list->give_name()%10)
{
case 1:
  break;
case 2:
  break;
case 3:
  break;
case 4:
  from_list->add_neighbours(current_node->give_child(1));
  from_list->add_neighbours(current_node->give_child(2));
  from_list->add_neighbours(current_node->give_child(3));
  from_list->add_neighbours(current_node->give_child(4));
  break;
case 5:
  break;
case 6:
```

```
      from_list->add_neighbours(current_node->give_child(1));
      from_list->add_neighbours(current_node->give_child(2));
      from_list->add_neighbours(current_node->give_child(5));
      from_list->add_neighbours(current_node->give_child(6));
      break;
    case 7:
      from_list->add_neighbours(current_node->give_child(1));
      from_list->add_neighbours(current_node->give_child(3));
      from_list->add_neighbours(current_node->give_child(5));
      from_list->add_neighbours(current_node->give_child(7));
      break;
    case 8:
      break;
    }
       break;
     }
}
    else
{
  switch(current_node->give_name()%10)
     {
     case 1:
       switch(from_list->give_name()%10)
    {
    case 1:
      break;
    case 2:
      temp=from_list->give_child(1);
      temp->add_neighbours(current_node->give_child(2));
      temp=from_list->give_child(3);
      temp->add_neighbours(current_node->give_child(4));
      temp=from_list->give_child(5);
      temp->add_neighbours(current_node->give_child(6));
      temp=from_list->give_child(7);
      temp->add_neighbours(current_node->give_child(8));
```

```
  break;
case 3:
  temp=from_list->give_child(1);
  temp->add_neighbours(current_node->give_child(3));
  temp=from_list->give_child(2);
  temp->add_neighbours(current_node->give_child(4));
  temp=from_list->give_child(5);
  temp->add_neighbours(current_node->give_child(7));
  temp=from_list->give_child(6);
  temp->add_neighbours(current_node->give_child(8));
  break;
case 4:
  break;
case 5:
  temp=from_list->give_child(1);
  temp->add_neighbours(current_node->give_child(5));
  temp=from_list->give_child(2);
  temp->add_neighbours(current_node->give_child(6));
  temp=from_list->give_child(3);
  temp->add_neighbours(current_node->give_child(7));
  temp=from_list->give_child(4);
  temp->add_neighbours(current_node->give_child(8));
  break;
case 6:
  break;
case 7:
  break;
case 8:
  break;
}
   break;
 case 2:
   switch(from_list->give_name()%10)
{
case 1:
```

```
        temp=from_list->give_child(2);
        temp->add_neighbours(current_node->give_child(1));
        temp=from_list->give_child(4);
        temp->add_neighbours(current_node->give_child(3));
        temp=from_list->give_child(6);
        temp->add_neighbours(current_node->give_child(5));
        temp=from_list->give_child(8);
        temp->add_neighbours(current_node->give_child(7));
        break;
      case 2:
        break;
      case 3:
        break;
      case 4:
        temp=from_list->give_child(1);
        temp->add_neighbours(current_node->give_child(3));
        temp=from_list->give_child(2);
        temp->add_neighbours(current_node->give_child(4));
        temp=from_list->give_child(5);
        temp->add_neighbours(current_node->give_child(7));
        temp=from_list->give_child(6);
        temp->add_neighbours(current_node->give_child(8));
        break;

        ....................
        ....................
        ....................

      }
         break;
       }
   }
      counter++;
    }
}
```

D.2.5 Function: Distance Transform

```
void distance_transform(node *root_node,node *current_node)
{
  nodequeue *myqueue;
  int queue_size;
  int counter;
  node *temp;
  node *store;
  int dt_value=1;
  long current_dt;

  // Set goal node DT to 0
  current_node->set_distance_transform(0);

  // root node only case
  if(current_node==root_node) return;

  // Create a queue to store neighbours
  myqueue=new nodequeue;

  do
  {
    queue_size=current_node->give_items();
    current_node->iter_restart();
    counter=0;
    while(counter<queue_size)
    {
      // Get nodes neighbours
      store=current_node->next();
      if(store->give_dt()<0 && store->give_type()==MT)
      {
   // Build the queue up with neighbours
   myqueue->add(store,dt_value);
```

```
      }
      counter++;
    }

    // Remove the next node from the queue
    temp=myqueue->remove();

    // Remove the current node's DT value
    current_dt=myqueue->remove_dt();

    temp->set_distance_transform(current_dt);

    // Assign the temp node to the current node
    current_node=temp;

    // Increment the DT value
    dt_value++;
  }
  while(myqueue->empty()==false);

  return;
}
```

D.2.6 Function: Path Planning

```
bool find_path(node *start,node *goal,nodequeue *path)
{
  node *temp;
  node *store;
  node *lowest;
  int counter;
  int queue_size;
  int path_test;
```

```
// Flag true if path exists, false if not
bool path_flag=false;

temp=start;
lowest=start;

path->add(temp,0);

// Start and goal are the same node
// Also caters for the case of only a root node
if(start==goal) return true;

// If start location has a negative dt value
// no path exists
if(start->give_dt()==-1)
{
  MessageBox(0,"No path possible","Find path",MB_OK);
  return false;
}

// Find a path
while(temp!=goal)
{
  queue_size=temp->give_items();
  temp->iter_restart();
  counter=0;
  path_test=0;
  while(counter<queue_size)
  {
    store=temp->next();
    if(store->give_dt()==-1)
    {
 path_test++;
    }
    else if(store->give_dt()<temp->give_dt()
```

```
          &&store->give_dt()<lowest->give_dt())
        {
    lowest=store;
        }
        path_flag=true;
        counter++;
      }
      temp=lowest;
      path->add(temp,0);
      if((queue_size-path_test)==1)
      {
        path_flag=false;
        break;
      }
    }

    if(path_flag==false)
    {
      // Something went wrong
      MessageBox(0,"No path","Find path",MB_OK);
      return false;
    }
    else
    {
      // Path found
      MessageBox(0,"Path Found","Find path",MB_OK);
      return true;
    }
}
```

References

[1] D.I.Jones and G.K.Earp, "Requirements for aerial inspection of overhead power lines," in *Proceedings of the 12th International Conference on Remotely Piloted Vehicles*, 1996.

[2] D.I.Jones, "A robotic device for aerial inspection of overhead power lines: Feasibility report," EA Technology Report 3510, EA Technology Ltd, December 1995.

[3] D.I.Jones, "A robotic device for aerial inspection of overhead power lines: Bridging study," EA Technology Report 3398, EA Technology Ltd, November 1995.

[4] R.G.Austin and P.A.Ryrie, "The sprite system - an update," in *Proceedings of the 8th International RPV Conference*, December 1995.

[5] D.I.Jones, "Case study and requirements for robotic inspection of power lines," Seminar on robotics for use in the electricity industry, EA Technology, January 1997.

[6] D.I.Jones and G.Earp, "Requirements for aerial inspection of overhead electrical power lines," Technology report, EA Technology Ltd, 1996.

[7] A.Meystel, *Autonomous Mobile Robots: Vehicles with Cognitive Control*, World Scientific, 1991.

[8] D.Shin, "A fast motion planning algorithm for a mobile robot using a distance transformation image," *Systems and Computers in Japan*, vol. 25, no. 5, pp. 88–99, 1994.

[9] A.Zelinsky, "A mobile robot exploration algorithm," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 6, pp. 707–717, 1992.

[10] R.A.Jarvis and J.C.Byrne, "Robot navigation: Touching, seeing and knowing," in *Proceedings of the 1st Australian Conference on Artificial Intelligence*, November 1986.

[11] R.Jarvis, "An all-terrain intelligent autonomous vehicle with sensor-fusion-based navigation capabilities," *Control Engineering Practice*, vol. 4, no. 4, pp. 481–486, 1996.

[12] G.M.Hunter and K.Steiglitz, "Operations on images using quad trees," *IEEE Transactions on Patern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 145–153, 1979.

[13] M.Williams, D.I.Jones, and G.K.Earp, "Obstacle avoidance during aerial inspection of power lines," in *2nd Workshop on European Scientific and Industrial Collaboration*, G.N.Roberts and C.A.J.Tubb, Eds., pp. 61–68. Mechatronics Research Center, University of Wales College, Newport, September 1999.

[14] M.Williams and DI.Jones, "A rapid method for path planning in three dimensions for a small aerial robot," *Robotica*, vol. 19, pp. 125–135, 2001.

[15] M.Williams D.I.Jones and G.K.Earp, "Obstacle avoidance during aerial inspection of power lines," *Aircraft Engineering and Aerospace Technology*, vol. 73, no. 5, pp. 472–479, 2001.

[16] BBC News, "Power firm hit by 100,000 fine," Web Document http://news2.thls.bbc.co.uk/hi/english/uk/newsid_392000/392268.stm, July 1999.

[17] BBC News, "Fatal police air crash investigation," Web Document http://news2.thls.bbc.co.uk/hi/english/uk/newsid_190000/190543.stm, October 1998.

[18] BBC News, "Helicopter crash kills three," Web Document http://news2.thls.bbc.co.uk/hi/english/uk/newsid_139000/139680.stm, July 1998.

[19] G.K.Earp and L.D.Malone, "Evaluation of a stabilised camera platform for power line inspection from a helicopter," EA Technology Report 4806, EA Technology Ltd, February 1999.

[20] A.Clot and D.J.Smith, "Making sense of 'see and avoid'," *Unmanned Vehicles*, pp. 24–27, October 2000.

[21] CAA, *CAP 658: Small (model) aircraft: A guide to safe flying*, Civil Aviation Authority, 1995.

[22] B.Sridhar and G.B.Chatterji, "Vision based obstacle detection and grouping for helicopter guidance," *Journal of Guidance and Control*, vol. 15, no. 5, pp. 908–915, 1994.

[23] V.H.L.Cheng and T.Lam, "Automatic guidance and control for helicopter obstacle avoidance," *Journal of Guidance and Control*, vol. 6, no. 1, pp. 1252–1259, 1994.

[24] B.Sridhar and A.V.Phatak, "Analysis of image-based navigation systems for rotorcraft low-altitude flight," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 290–299, 1992.

[25] Y.K.Hwang and N.Ahuja, "Gross motion planning - a survey," *ACM Computer Surveys*, vol. 24, no. 3, pp. 219–291, 1992.

[26] L.E.Kavraki, "Computation of configuration-space using the fast fourier transform," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 408–413, 1995.

[27] A.B.Doyle, *Algorithms and Computational Techniques for Robot Path Planning*, Ph.D. thesis, School of Electronic Engineering and Computer Systems. University of Wales, Bangor, September 1995.

[28] J.Canny, *Complexity of Robot Motion Planning*, MIT Press, August 1988, ISBN 0262031361.

[29] J-C.Latombe, *Robot Motion Planning*, Kluwer Academic, 1991.

[30] S.Kambhampati and L.S.Davis, "Multiresolution path planning for mobile robots," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 135–145, September 1986.

[31] H.Samet, "An algorithm for converting rasters to quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, no. 1, pp. 93–95, January 1981.

[32] H.Samet, "Distance transform for images prepresented by quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 4, no. 3, pp. 298–303, May 1982.

[33] H.Samet, "An overview of quadtrees, octrees, and related hierarchical data structures," in *Theoretical Foundations of Computer Graphics and CAD*, R.A.Earnshaw, Ed., vol. F40 of *NATO ASI*, pp. 51–68. Springer-Verlag, 1988.

[34] E.Hawaguchi and T.Edno, "On a method of binary-picture representation and its application to data compression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, no. 1, pp. 27–35, January 1980.

[35] D.Meagheer, "Octree encoding: a new technique for the representation,the manipulation, and display of arbitarry 3-d objects by computers," Technical report TR-80-111, Rensselaer Polytechnic Institute, October 1980.

[36] M.E.Rozmann and J.Detlefsen, "Environmental exploration based on a three-dimensional imaging radar sensor," in *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992, pp. 422–429.

[37] J.Horn and G.Schmidt, "Continuous localization of a mobile robot based on three-dimensional-laser-range-data, predicted sensor images and dead-reckoning," *Robotics and Autonomous Systems*, vol. 14, pp. 99–118, 1995.

[38] J.Wright, K.Scott, T-H.Chao, and B.Lau, "Multi-sensor data fusion for seafloor mapping and ordnance location," in *Proceedings of the 1996 Symposium on Autonomous Underwater Vehicle Technology*, June 1996, pp. 167–175.

[39] B.Bhanu, S.Das, B.Roberts, and D.Duncan, "A system for obstacle detection during rotorcraft low altitude flight," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, no. 3, pp. 875–897, 1996.

[40] Y-L.Tang and R.Kasturi, "Accurate estimation of object location in an image sequence using helicopter flight data," *Robotics and Computer-Integrated Manufacturing*, vol. 11, no. 2, pp. 65–72, 1994.

[41] B.K.P.Horn and B.G.Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[42] Y.Yagi, Y.Nishizawa, and M.Yahida, "Map-based navigation for a mobile robot with omnidirectional image sensors copis," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 5, pp. 634–478, 1995.

[43] A.K.Dalmia and M.Trivedi, "Depth extraction using a single moving camera: and integration of depth from motion and depth from stereo," *Machine Vision and Applications*, vol. 9, pp. 43–55, 1996.

[44] A.Mitiche, S.Seida, and J.K.Aggarwal, "Determining position and displacement in space from images," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1985, pp. 504–509.

[45] H.Zhuang, R.Sudhakar, and J.Shieh, "Depth estimation from a sequence of monocular images with known camera motion," *Robotics and Autonomous Systems*, vol. 13, pp. 87–95, 1994.

[46] B.K.P.Horn and E.J.Weldon, "Direct methods for recovering motion," *International Journal of computer vision*, vol. 2, no. 1, pp. 51–76, 1988.

[47] D.W.Murray and K.J.Bradshaw, "Driving saccade to pursuit using image motion," *International Journal of Computer Vision*, vol. 16, no. 3, pp. 205–228, 1995.

[48] D.Regan and A.Vincent, "Visual processing of looming and time to contact throughout the visual field," *Vision Research*, vol. 35, no. 11, pp. 1845–1857, 1995.

[49] K.Joarder and D.Raviv, "A new method to calculate looming for autonomous obstacle avoidance," in *Proceedings of the 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1994, pp. 777–780.

[50] R.C.Nelson and Y.Aloimonos, "Obstacle avoidance using flow field divergence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 10, pp. 1102–1106, 1989.

[51] E.Trucco and A.Verri, *Introductory Techniques for 3-D Computer Vision*, Prentice Hall, 1998.

[52] G.Adiv, "Determining three-dimensional motion and structure from optical flow generated by several moving objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, no. 4, pp. 384–401, 1985.

[53] P.Burlina and R.Chellappa, "Analyzing looming motion components from their spatiotemporal spectral signature," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 10, pp. 1029–1033, 1996.

[54] S.S.Beauchemin and J.L.Barron, "The computation of optical flow," *ACM Computer Surveys*, vol. 27, no. 3, pp. 433–467, 1995.

[55] R.Sharma and Y.Aloimonos, "Early detection of independent motion from active control of normal image flow patterns," *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 26, no. 1, pp. 42–52, 1996.

[56] M.Campani and A.Verri, "Computing optical flow from an over-constrained system of linear algebraic equations," in *Proceedings of the Third IEEE International Conference on Computer Vision*, 1990, pp. 22–26.

[57] M.Tistarelli and G.Sandini, "On the advantages of polar and log-polar mapping for direct estimation of time-to-impact from optical flow," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 4, pp. 401–410, 1993.

[58] M.Tistarelli, "Multiple constraints to compute optical flow," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 12, pp. 1243–1250, 1996.

[59] E.De.Micheli, V.Torre, and S.Uras, "The accuracy of the computation of optical flow and of the recovery of motion parameters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 5, pp. 434–447, 1993.

[60] Y.Shirai, Y.Mae, and S.Yamamoto, "Object tracking by using optical flows and edges," in *The 7^{th} Symposium on Robotics Research*, May 1996, Lecture Notes in Computer Science, pp. 440–447.

[61] J.L.Barron, D.J.Fleet, S.S.Beauchemin, and T.A.Burkitt, "Performance of optical flow techniques," in *Procedings of the IEEE on Computer Vision and Pattern Recognition*, pp. 236–242. 1992.

[62] B.Galvin, B.McCane, K.Novins, D.Mason, and S.Mills, "Recovering motion fields: An evaluation of eight optical flow algorithms," in *Proceedings of the ninth British machine vision conference*, P.H.Lewis and M.S.Nixon, Eds., vol. 1, pp. 195–205. 1998.

[63] University of Western Ontario, "Optical flow software," ftp://ftp.csd.uwo.ca/pub/vision, 1992.

[64] P.Anandan, "A unified perspective on computational techniques for the measurement of visual motion," in *Proceedings of the First International Conference on Computer Vision*, June 1987, pp. 219–230.

[65] A.Klinger, "Patterns and search statistics," in *Optimizing Methods in Statistics*, J.S.Rustagi, Ed., pp. 303–337. Academic Press, 1971.

The End