DOCTOR OF PHILOSOPHY

Visual control of an unmanned aerial vehicle for power line inspection

Golightly, Ian

Award date:
2006

Awarding institution:
Bangor University

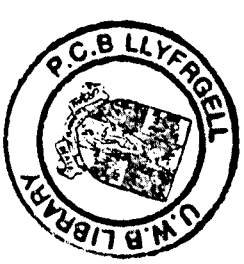
Visual control of an unmanned aerial vehicle for power line inspection

Ian Thomas Golightly

Thesis submitted in Candidature for the degree of Doctor of Philosophy

May 2006

School of Informatics
University of Wales, Bangor



Summary

This thesis describes the investigation into the use of visual servoing to keep an unmanned aerial vehicle (UAV) aligned with overhead electricity distribution lines, in order to use it to inspect them. The UAV would carry cameras in order to capture video footage showing the line's condition.

Firstly, the current methods of inspecting overhead electricity distribution lines, line-walking and manned helicopters, are described. A review of visual servoing and the relevant tracking methods is presented. Then a mathematical model of a ducted-fan UAV is developed. Analysis of the image geometry is performed to show how movements of the UAV affect the positions that the overhead lines appear in the images from the UAV's camera. This analysis shows that it should be possible to estimate the UAV's position relative to the lines if two cameras, one pointing forward and one pointing backwards, are used. The design and construction of a laboratory test rig to perform experiments is described. Then the image processing method, based on the Hough transform, used to extract the overhead lines from the image is described followed by the development of a tracker, which makes use of fuzzy logic and a Kalman filter, to track the overhead lines from frame to frame. Experiments are performed to see how well the UAV is able to follow the lines using the laboratory test rig. Finally, conclusions are drawn as to how well the system works as well as suggestions for the future direction of the project.

Contents

List of Figures

List of Tables

Declaration page not scanned
at the request of the
University

Acknowledgements

I would like to thank Dr Dewi Jones, for all the encouragement, advice and support with all areas of this PhD project and for checking this thesis.

I would also like to thank Mr Iwan Jones of the Mechanical Workshop for his advice and skills in the construction of the laboratory test rig.

Thanks also go to EPSRC for supporting the project.

Finally, thanks go to my family for their support and for being there and thanks also go to all my friends in the School of Informatics, the UWB Archery Club and elsewhere.

Chapter 1 Introduction

1.1 Introduction

In the modern world we are using more and more electrical equipment and are becoming increasingly reliant on computers in all aspects of life. In order to support this modern lifestyle, we have become increasingly reliant on a reliable electricity supply. In addition, more basic requirements for survival are now dependent on the electricity supply; for example, modern heating systems, while usually fired by gas or oil, require electricity to run the controllers and so during a power cut more and more people have no way of heating their homes. People are also left without lighting and, in many cases, cooking facilities. Power cuts, or outages, are no longer acceptable to consumers or businesses, and electricity companies have a duty to keep the electricity supply on.

Figure 1.1: Example Support Pole showing 3-phase Conductors on Pin Insulators and a Pole-mounted Transformer.

Electricity distribution in the U.K. is usually done by means of 3-phase overhead lines. These are open conductors supported on wooden poles, an example of which is shown in Figure 1.1, at a voltage of 11 or 33kV. There are approximately 150,000km of overhead line and 1.5 million wood support poles in the U.K., which are primarily in rural areas, as most distribution within towns and cities is done using underground cables. Putting cables underground means that they are

less susceptible to damage, although any repairs that are required necessitate the cables being dug up. The use of underground cables is far more expensive than overhead lines (around five to ten times the cost), hence the use of overhead lines for rural distribution, where the distances are longer. The lines and poles need regular inspection to detect faults, check for tree encroachment and ensure a reliable electricity supply. In addition the law requires the electricity companies to inspect the distribution lines, and electricity companies may have to pay compensation to customers if their supply is off for too long. The safety of members of the public may be at risk if safety measures such as warning notices and anti-climb guards become damaged or missing and are not replaced. The electricity companies also need to monitor the state of the lines so that replacement and upgrade strategies can be planned and estimates of required future investment can be made.

1.2 Overview of the Thesis

1.2.1 Motivation

In order to ensure a reliable supply of electricity, it is necessary to inspect the distribution lines regularly. Currently the lines are inspected by inspectors walking the line and observing its condition. This gives a good written record of the lines' condition, with any possible faults recorded, although degradation of the top of the insulators cannot be seen from below. Unfortunately it is a slow method of inspection and at times involves inspectors going into fairly rough terrain. On rare occasions, inspectors can also be attacked by livestock.

An alternative inspection method that is currently used is to inspect the lines from manned helicopters. This method is quite fast but expensive. Nevertheless, many companies use this method on a regular basis. Inspection using manned helicopters is hazardous because the helicopter flies within five to ten metres of the lines in order for the observer to be close enough to see the overhead line in sufficient detail for defects to be spotted. This requires great skill from the pilot. It is difficult to inspect power lines near roads or property due to the risk to people on the ground.

This project considers an alternative method of inspection. The idea is to use an unmanned aerial vehicle (UAV), carrying a camera, which would fly above the line to capture video, showing its condition. The UAV would be electrically insulated, so that if it landed on the line, it would not cause a short circuit. The UAV would also draw power from the line, which would limit its ability to fly away from the line. This "tethering" of the UAV to the vicinity of the lines is an important feature of the concept, with regard to preparing a safety case for the Civil Aviation Authority (CAA), and is discussed in more detail in Chapter 2. The video footage of the line could then be lodged in a database and analysed, and the data used to schedule maintenance. It is believed that this approach will be faster than walking the line, but without the cost and at reduced risk compared to manned helicopters.

1.2.2 Aims

The aim of this work is to investigate the possibility of using visual servoing to control a UAV for power line inspection. This requires the following objectives to be fulfilled:

- Develop a mathematical model of the UAV.
- Construct a test rig on which to perform experiments.
- Develop and test a method of locating the overhead lines in the images taken from a camera onboard the UAV.
- Develop and test a method of tracking the lines from frame to frame.
- Demonstrate visual control of the UAV.
- Demonstrate visual control of multiple axes of the UAV.

1.2.3 Structure

Chapter 2 describes the background of the project and includes a description of the concept of using a UAV for power-line inspection. A review of the literature for visual servoing is also presented. Based on the review, suitable methods are selected for guiding the UAV along the line and tracking of the lines from frame to frame.

Chapter 3 presents a mathematical model of the chosen type of UAV, along with simulation results of the model and the design of a feedback controller.

In Chapter 4 a geometric analysis of how the lines are transformed into the image is presented. The mathematical model is used to predict how movements of the UAV in lateral displacement, height, yaw, roll and pitch will affect the positions of the overhead lines in the image. It is shown that the effect of movements in different degrees of freedom on the line positions in the image is largely additive, meaning that it is possible to extract the position and pose of the UAV from image processing. In addition, it is shown that, in order to separate yaw, roll and lateral displacement a second camera, pointing rearward, is needed.

Chapter 5 describes the laboratory test-rig used for experiments. The extensive mechanical modification of an existing test-rig is described, along with the design and construction of a digital position controller, implemented in a microcontroller, to drive the rig. The structure of the control visual servoing software is also presented.

The image processing software is described in Chapter 6. This is based on the Hough transform, which transforms straight lines in the image into points in the transform space. The selection and testing of the methods of pre-processing of the image and post-processing of the transform are described.

The development of a tracker to track the lines from frame to frame is described in Chapter 7. First, an early local search tracker is described. An acquisition routine that finds the lines to initialise tracking is developed. Refinements are introduced, including the addition of fuzzy logic rules to detect if the tracker has mistaken a sideline for the centre line or if the tracker has lost lock on the lines. Finally, a Kalman filter is added to smooth out noise in the line positions and reduce the chances of the tracker switching away from the lines. Results for tracking both lateral displacement and height are also presented.

Chapter 8 describes the use of two cameras, with the tracker algorithm applied to both video streams, in order to extract yaw and roll as well as lateral

displacement. Visual control of both the lateral displacement and yaw simultaneously is demonstrated, along with measuring the roll of the UAV from the image.

Finally, Chapter 9 discusses the work in this thesis and assesses the extent to which the objectives have been satisfied. Conclusions are drawn and a discussion of how the project could be developed in the future is given.

1.3 Contributions of this Research Work

The research described in this thesis makes the following contributions to knowledge, which, to the best of the author's knowledge, have not been previously reported.

- A mathematical model for a ducted fan rotorcraft UAV controlled by shifting its centre of gravity (CG) has been developed.
- An extension of the Hough Transform method, called the Aggregated Hough Transform, has been developed, which is particularly suited to locating overhead power lines in aerial images.
- A model-based visual tracking method, using fuzzy logic and a Kalman filter, has been developed to track the lines from frame to frame.
- Control of the UAV to keep it aligned with the power line, based on visual measurement, has been demonstrated. This includes controlling multiple degrees of freedoms including lateral displacement, yaw and height and estimating the roll of the UAV.

1.4 Contributions to Published Literature

Published:

[1] Golightly, I.T. and Jones, D.I., *Visual control of an unmanned aerial vehicle for power line inspection.* in *Proc. IEEE Int. Conf. Advanced Robotics (ICAR 2005)*. 2005. Seattle, USA: 228-295.

This paper was a finalist for the "Boeing Company Best Paper Award".

[2] Jones, D., Golightly, I., Roberts, J., Usher, K. and Earp, G., *Power line inspection - a UAV concept.* in *IEE Forum on: Autonomous Systems.* 2005. London, UK.

Accepted:

[3] Jones, D., Golightly, I., Roberts, J. and Usher, K., *Modeling and Control of a Robotic Power Line Inspection Vehicle.* to appear in *Proc. IEEE International Conference on Control Applications (CCA 2006).* 2006. Munich, Germany.

Chapter 2 Background and Literature Review

2.1 Overview

This chapter has three main functions. First, it describes the current methods used for inspecting power distribution lines, and discusses the possible use of a UAV for power line inspection. Secondly, a review of the current literature on visual servoing is presented. Finally the theory of Fuzzy Logic and Kalman filtering used in the tracking software is presented.

2.2 Power Line Inspection

The U.K.'s electricity distribution lines and support poles need regular inspection in order to comply with legal requirements and to ensure a reliable electricity supply. As the poles and lines are exposed to the elements, there are many ways that they can sustain damage. When the lines are inspected, the inspectors are looking for defects such as:

- Cracked, or degraded insulators.
- Signs of corrosion on the cables.
- Cables that have come off the insulators and are hanging or resting on the cross-arm.
- Damage to the pole or cross-arm.
- Broken, slack or missing stays.
- Damaged pole-mounted transformers.
- Encroachment of trees.
- Traces of arcing.
- Missing or damaged safety notices or anti-climb guards.

If these problems can be detected early, then they can be fixed before they cause a problem. For example if one conductor comes off its insulator and rests on the cross-arm, the line will still work, as the cross-arms are not earthed. However, if a

second conductor comes off, then there will be a short circuit, causing severe damage and presenting a danger to anyone near the affected pole, as lines could fall. If the problem is detected early, then the first conductor can be put back onto its insulator before there is a problem, and the other two conductors can be checked to make sure they are not about to come off. It is important to check for tree encroachment as most damage caused to power lines in storms is due to trees being blown over onto the lines. In addition, if trees grow too close to the line, then children can climb them and be electrocuted by the line. An example of tree encroachment is shown in Figure 2.1.

Figure 2.1: Tree Encroaching on a Power Line.

Currently the lines are inspected by inspectors walking the line and observing its condition. This gives a good written record of the lines condition, with any possible faults recorded, although degradation of the top of the insulators cannot be seen from below. This means that not all faults can be seen by this inspection method. Unfortunately it is a slow method of inspection, as inspectors have to walk between the poles. They often have to cross walls, fences and hedges, which slows the process further, and at times the inspectors have to go into fairly rough terrain, as shown in Figure 2.2. Many of the lines cross farmland and on rare occasions, inspectors can be attacked by livestock. It can also be quite tedious,

meaning that faults are occasionally not recorded or detail is poor. As there is no visual record, the inspector's report has to be relied upon, and it is difficult to tell if degradation has got worse between inspections.

Figure 2.2: Power line in an Upland Area.

An alternative method of inspection is to use manned helicopters. This method is also used by National Grid Transco to inspect transmission lines and pylons. This method is quite fast, but is very expensive. It is also hazardous, as the helicopter has to fly within five to ten metres of the line in order to allow the observer to see the lines with sufficient detail. This requires a highly skilled pilot in order to avoid crashing into the line or the ground and makes it difficult to inspect power lines near roads of property due to the risk to people on the ground. It can also be difficult for the observer to tell which pole appears in the image.

A project [4-9] was run at Bangor to improve the quality of inspection from helicopters by using a video camera on a stabilised mount. A visual tracker to keep the camera pointing at the poles was developed. This uses a combination of Differential Global Positioning System (DGPS) and machine vision to lock the camera onto the pole and keep it locked onto the pole as the helicopter flies past.

This system should allow the camera operator to zoom in on the pole to capture video showing its condition.

This thesis considers an alternative method of inspection. The idea is to use an unmanned aerial vehicle (UAV), which would fly above the line to capture video, showing its condition. An early artist's impression is shown in Figure 2.3.

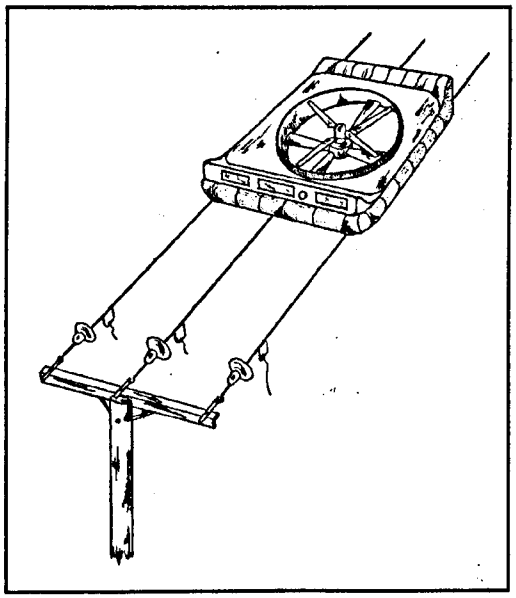


Figure 2.3: Early Artist's Impression of the UAV Flying Above the Lines.

The UAV proposed for the project is a ducted fan rotorcraft. Many rotorcraft of type have been constructed, ranging from the "flying platform" [10] of the 1950s to current examples such as [11]. For this work, we have assumed a ducted fan with contra-rotating propellers (to equalise yaw moment on the airframe) and a payload above the duct so that the centre of gravity is above the aircraft centre. Prouty [12] shows that this configuration should be dynamically stable, but only marginally statically stable with a tendency to drift into translational flight. Ando [13] indicates that using a duct with a lip, or bellmouth, and placing the centre of gravity within a very small height range above the aerodynamic centre of the duct, then the craft can be made asymptotically stable. Height and yaw control of the craft is provided by adjusting the speed of the rotor blades either together for lift or differentially for yaw. Control of the pitch and roll axes is provided by moving the payload mass in the relevant direction. The craft moves in the horizontal plane by pitching and/or rolling in the desired direction of travel. This means that the

craft is under actuated: the same actuator controls both roll and lateral displacement, while another actuator controls forward movement and pitch. Some of the problems associated with this kind of craft are discussed by Hamel [14, 15]. The advantages of using a ducted fan are that the duct improves the hover efficiency and means that, in the event of a crash, the rotor blades are shielded and so won't hit anything. The airframe would be designed to be electrically insulating, such that if it comes to rest on the lines, it doesn't cause a short circuit.

In this concept, it is planned that the UAV will pick up its power from the line itself. The type of missions that the UAV will be expected to carry out vary in length and can be quite long, which means that the use of a battery powered vehicle would not be practical. A UAV powered by an internal combustion engine or a turbine could be used, although these are very noisy and would require the carrying of fuel. If the UAV is powered from the line then it is possible to reduce its weight because there is no need to carry fuel or large numbers of batteries. There will be the weight of the power pick-up and conversion equipment, however. An additional advantage of using an electric UAV is that, in the event of a crash, there is no fuel on board to cause an explosion or a fire. Even though the UAV will be powered from the line, the UAV will still need some battery power as it will need to disconnect from the lines in order to fly over the poles.

One of the problems facing the project is compliance with the CAA regulations on UAVs [16]. This primarily involves complying with the 'Sense and Avoid' requirement to avoid collision with other aircraft. In order to comply with this requirement, this would require an "intelligent electronic pilot" to be put into the UAV. The aim of this project is to design a component of this "pilot" that uses visual servoing to keep the UAV aligned with the lines. The UAV will also need to be able to avoid obstacles in its path and work has been done in Bangor by Matthew Williams [17] on a similar project looking at using vision to achieve this. Compliance with CAA regulations is another advantage of powering the UAV from the line. As the UAV is effectively tethered to the line, by its requirement for power, it cannot fly far from the line and so endanger other aircraft.

For small UAVs (<20kg) that are operated below 400ft and within sight of an operator, it is possible to operate under the rules for model aircraft [18], although permission is required for commercial use of UAVs between 7 and 20kg. It would be possible to operate the power line inspection UAV under the rules in [18] if it was kept in sight of the operator. To operate a UAV that is over 20kg would require airworthiness certification. Operating out-of-sight of the operator is currently not normally permitted and so the aim would be to gain approval for the use of an on-board electronic pilot with remote monitoring and supervision. The effective tethering of the UAV to the line will help with the safety case for such a vehicle.

The control system would guide the UAV along the lines under visual control, but with the ability for the operator to take control if necessary. The UAV would capture video showing the condition of the line. This would be viewed off-line to check for faults, as well as providing a visual record of the condition of the line. This would be the most likely way of operating early commercial versions of an inspection UAV. For the UAV to become economically attractive, longer-range missions with operation out-of-sight of the operator would be necessary. A business case for using a UAV for power line inspection is given in [19]. The use of a UAV should give higher quality video footage of the line and be cheaper and safer than using manned helicopters, as well as being quicker than line walking. A longer-term aim would be to have the UAV more autonomous. It may also be possible to include some fault detection software, to automatically spot faults on the lines.

2.3 Visual Servoing

Visual Servoing [20, 21] refers to the control of robots or vehicles using vision to provide the control feedback signals in real-time using a closed vision loop. In older systems, referred to as "look then move", an image would be captured and processed to locate the feature of interest. This robot would then move to the calculated position with no more input from the vision system. The term visual servoing is believed to have been used first in 1979 by Hill and Park [22]. Visual servoing can be used for both factory robots in a structured environment and for

robots operating in real-world environments. There is an excellent tutorial on visual servoing by Hutchinson et al. [23]. Visual Servoing is a very wide field and is addressed by many researchers.

Visual servoing systems all use one or more cameras for input. These can either be fixed in the workspace, looking at the robot and target, or mounted to the robot end-effector, referred to as "eye-in-hand". Factory robots can use either system, but when visual servoing is used to control a vehicle, the camera is mounted onto the vehicle, and so they are eye-in-hand systems. The images from the camera are processed in real-time in order to locate the required target in the image. As well as processing the current frame, the vision system needs to match features found in the current frame with those found in the previous frame and thus track the target object from each frame to the next. These trackers usually use a predictive filter, in order to aid finding the target object in each new frame and also to reduce the effect of noise on the measurements. Once this has been done, the current position of the robot or vehicle relative to the target is calculated and the difference between the demanded position and the current position is calculated and this error signal is used to drive the robot. This is updated with each frame from the camera.

The primary problems associated with visual servoing are associated with the image processing. In unstructured environments, features in the background can affect the location of the target in the image, by providing alternative possibilities as to what the target is. Changing lighting can also affect the processing of images and so affect the ability of the vision system to find the target. As the vision system has to run in real time, there is a compromise between accuracy and speed: it is often necessary to use less accurate image processing methods in order to allow the vision system to process frames sufficiently fast to provide control input to the robot or vehicle.

A lot of the current research is into the use of visual servoing to control robots that operate in the real world. These include using visual servoing to manoeuvre small robots around buildings [24, 25], visual control of cars or other motor vehicles [26-29] and visual control of aerial vehicles.

As this project involves guiding a UAV along electricity distribution lines, it was necessary to look at the literature to find what techniques are in use for controlling UAVs. There are a number of UAV projects using visual servoing happening around the world and there are summaries of some of these by Ollero [30] and Kontitsis [31]. The UAVs used are primarily helicopters or rotorcraft, but there are projects that use airships. A variety of techniques are used to process the images to estimate the position of the UAV. These include optical flow, stereo vision, pattern matching, edge detection and the Hough transform. The techniques chosen are suited to the particular application, rather than there being universal techniques.

Hrabar el al. [32] used a combination of optic flow and stereo vision for guiding a UAV through urban canyons. This uses optical flow to estimate the speed and rotation of the UAV and also locating objects to be avoided. Stereo vision is used to locate objects in the field of view for obstacle avoidance. The outputs from both of these are combined, giving priority to any objects found by the stereo vision, as this is good at finding objects that are in front of the vehicle.

Work has also been done by Mejias et al. [33] into guiding UAVs through urban canyons. In this case the UAV uses visual servoing, in addition to GPS, to guide itself towards features of interest, in this case, windows on buildings. It uses colour segmentation to highlight them in the image. This image is then thresholded to produce a binary image. A square finding algorithm is used to locate the window in the binary image. A Kalman filter is used in tracking the location of the window in the image from frame to frame.

Amidi [34, 35] developed a visual odometer for autonomous helicopters. This works by detecting arbitrary objects on the ground and then using template matching to track objects on the ground and estimating the helicopter's position by the movement of these objects within the images from the stereo cameras. The system is also able to measure changes in yaw and height of the helicopter by changes in the appearance of the objects in the images.

The ELEVA project [36, 37] is a project to develop a UAV for the inspection of electricity pylons. This project intends to use a small helicopter to inspect electricity transmission lines. The system uses a Hough Transform to extract the lines from images of the lines; these are then tracked from frame to frame. Stereo vision is also used to estimate the distance from the lines. The helicopter sends images to a ground-based computer for processing via radio link and control signals are sent back up to the helicopter. The control computer allows a user to manually override control of the helicopter in the event of a problem. The visual servoing is augmented with inputs from Differential Global Positioning System (DGPS), an inertial measurement unit and a laser altimeter.

Mejias et al. [38] have been working on using machine vision to locate a safe landing area for landing a UAV in the event of a forced landing being required. The paper considers a power line inspection vehicle inspecting a line that is forced to land within a very short period of time. In order to do this, a forward pointing camera is used to detect the lines. When the system is no longer able to detect the lines, focus is switched to a downward pointing camera in order to look for a landing site. In order to search for a landing site, a contrast threshold is applied to the image in order to pick out obstacles on the ground. An edge detector is then applied to give the edges of the obstacles. The UAV then heads towards the largest area free of obstacles. This work was tested on the Air Vehicle Simulator (AVS) [39, 40] at the Autonomous Systems Laboratory, CSIRO, Australia.

The optical flow and stereo vision techniques could be useful for avoiding objects while inspecting power lines. These would primarily involve debris resting on the line, tree encroachment and occasions where insulators are above the level of the lines and work has already been done into using optical flow for this purpose by Matthew Williams [17]. Optical flow may also be able to detect when the UAV is approaching a support pole. Template matching could also be useful for locating poles while inspecting power lines. The aims of the ELEVA project are quite similar to those of this project and also uses image processing based on the Hough transform in order to locate the lines. Carelli et al [25] use an edge detector and a similar line classification to extract lines from the scene, although this is being used to guide a wheeled robot along corridors. The ability for the UAV to land

itself in case of an emergency will be needed for a final inspection UAV. This ability may also be useful for landing the UAV at the end of a mission.

2.4 Tracking

Tracking involves taking features found in an image frame and locating the same features in subsequent frames. Davison [41] describes four main tracking methods:

- Exhaustive search, where the entire image, or the transform space in this case, is searched for a match between the object being tracked in the previous frame and the current frame.
- Local search: this is similar to exhaustive search except that only the local area around the point at which the object would be expected to be found is searched.
- Kalman Filter [42]: this attempts to find a best estimate of the object's position by combining a prediction from the previous frames with the measurement of the object's position in the current frame. All the errors are assumed to have a Gaussian distribution and are used form a weighting factor to combine the prediction and measurement.
- Particle Filter [43]: this also uses errors to produce position estimates but unlike the Kalman filter, the errors are not assumed to be Gaussian. Instead the error function is represented by a number of particles, which allows more complex error functions to be represented, including multi-modal functions, allowing multiple-hypothesis testing.

In this project, the image is processed using a Hough Transform, which transforms lines in the image to points in the transform space. A tracker was developed to track the points in the transform space. This tracker started as a Local Search tracker and was later adapted to include a Kalman filter. In addition, fuzzy logic is used in the tracker for hypothesis testing. The Hough Transform, Fuzzy logic and the Kalman filter are briefly described in the following sections.

2.4.1 Hough Transform

The Hough Transform is a well-known method for extracting lines that match parameterized functions from an image. The most common case is classifying straight lines in normal form according to their angle (θ) and distance from the image centre (ρ).

In order to create the transform, an edge detector is used to pick out lines in the image, and then each line is classified by its angle and distance from the image centre.

The transform consists of a 2D array of accumulators addressed by ρ and θ. For each edge pixel, the relevant accumulator is incremented. After this process is completed, the transform is normalised such that all the accumulators have a value between zero and one. A threshold, called the Hough Transform threshold (H threshold), is then applied. This picks out the features of interest and suppresses background noise. For this application, clusters of points are aggregated to a single point. An example image and the resulting transform are shown in Figure 2.4.

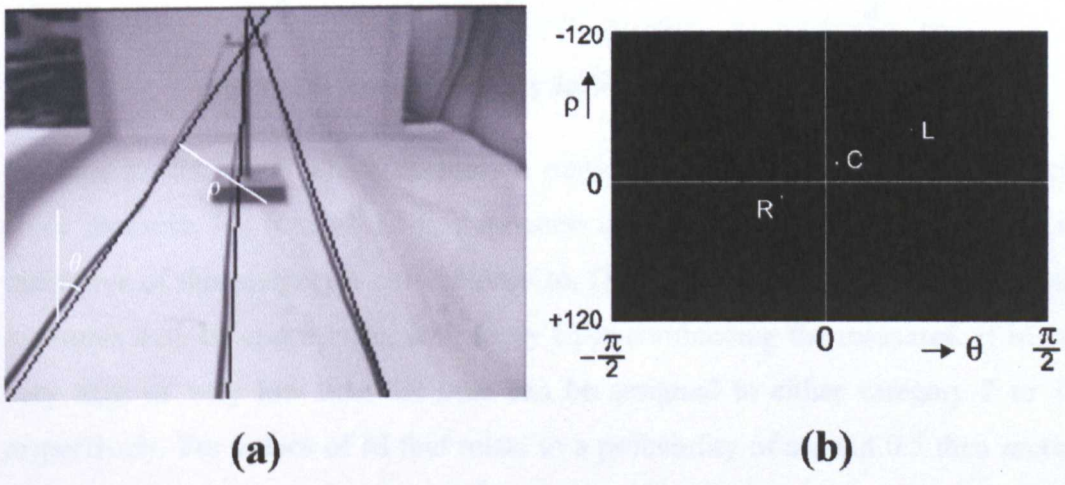


Figure 2.4: Typical Result showing: (a) the Three Overhead Lines Overlaid with the Straight Lines Generated by the Hough Transform and (b) the Corresponding Points in the Hough Transform Space.

2.4.2 Fuzzy Logic

Boolean logic is a useful method of decision making in situations where it is clear which category a given case belongs to. In many cases, however there is a grey area between the two categories. In these cases the possible categories form a fuzzy set and fuzzy logic [44, 45] can be used to differentiate between them. Figure 2.5 shows the different membership functions (μ) of a Boolean set (dotted) and a fuzzy set (solid). The value of μ indicates the probability of a given case belonging to category 2.

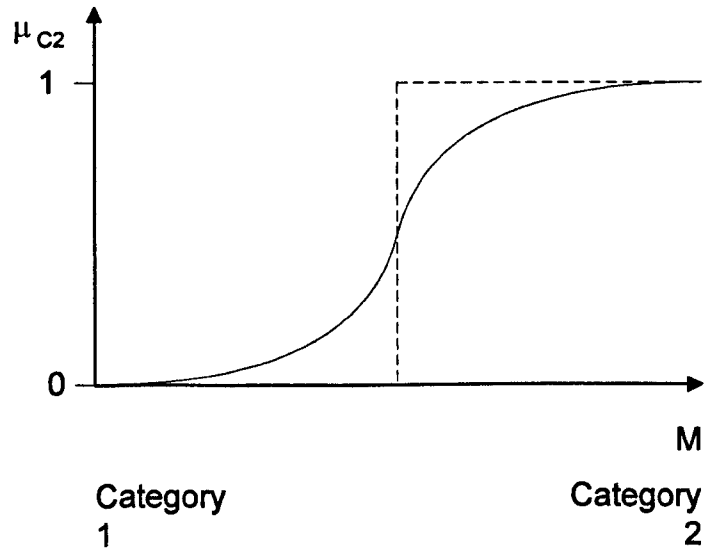


Figure 2.5: Classic and Fuzzy Set Membership Functions.

In order to determine which category a particular case belongs to, the value of some measure, M, is used. M is a measure associated with the situation and is indicative of the category a case belongs to. Depending on the situation, multiple measures may be appropriate, with fuzzy rules combineing the measures. If M is very high or very low then the case can be assigned to either category 2 or 1 respectively. For values of M that relate to a probability of around 0.5 then more information is needed to decide which category the case belongs to. This can be obtained from repeated measurements, which will either increase or decrease the probability value. As it is extremely unlikely to obtain perfect 0 or 1 probability values, it is necessary to defuzzify the set. This can be done by approximating the membership function to piece-wise linear. This creates a not sure category in between the two categories. When a case lies in this category, repeated

measurements must be made until the probability measure lies in one of the other two categories.

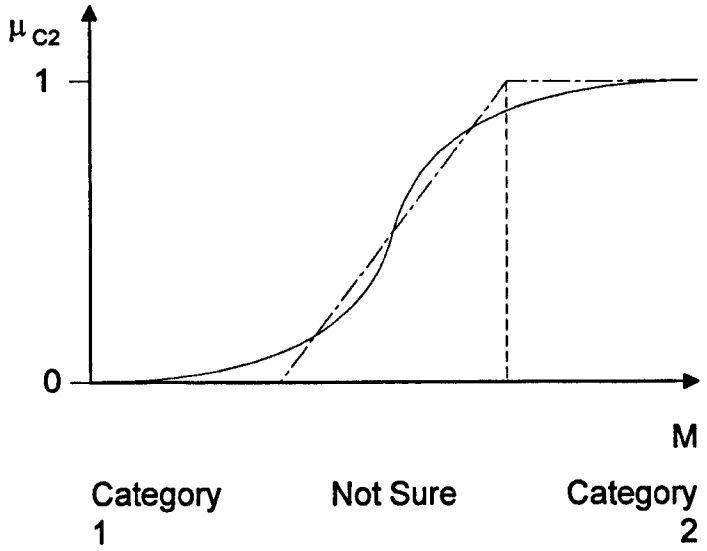


Figure 2.6: Defuzzifying a Fuzzy Set.

The tracking algorithm (Chapter 7) typically encounters a number of situations that require a decision to be made on the basis of uncertain data and this is where fuzzy logic is applied. For instance the tracker could mistake one of the sidelines for the centre line and cause a missing sideline in the Aggregated Hough transform (AHT). However, this could also be caused by a sideline disappearing from the frame. As it is not immediately apparent which of these situations has occurred, a fuzzy logic rule is used to distinguish between the two cases. A full account of the method is given in section 7.4.

2.4.3 Kalman Filter

When the lines are tracked from frame to frame, a measurement of their position in each frame is produced. Due to the nature of image processing, these measurements are quite noisy. In order to improve their accuracy, it is normal to filter the measurements. This smooths out the noise in order to give a more accurate estimate of the line's position. In 1960 Kalman published the Kalman filter [42, 46], which is a recursive filter. It gives an optimal solution if the errors in the measurements being filtered have a Gaussian distribution. With other error

probability density functions, the Kalman filter can still give good results; in order to use it in these situations, the errors are assumed to be Gaussian.

For the line tracking Kalman filters will be used to filter the ρ and θ values of each line. The Kalman filter has four stages:

- Calculate the Kalman gain: this determines the fraction of the estimate that is from the prediction and the fraction from the measurement.
- Update the estimate.
- Update the error (variance) associated with the estimate.
- Calculate the prediction for the next frame and its associated error (variance).

This is shown in Figure 2.7.

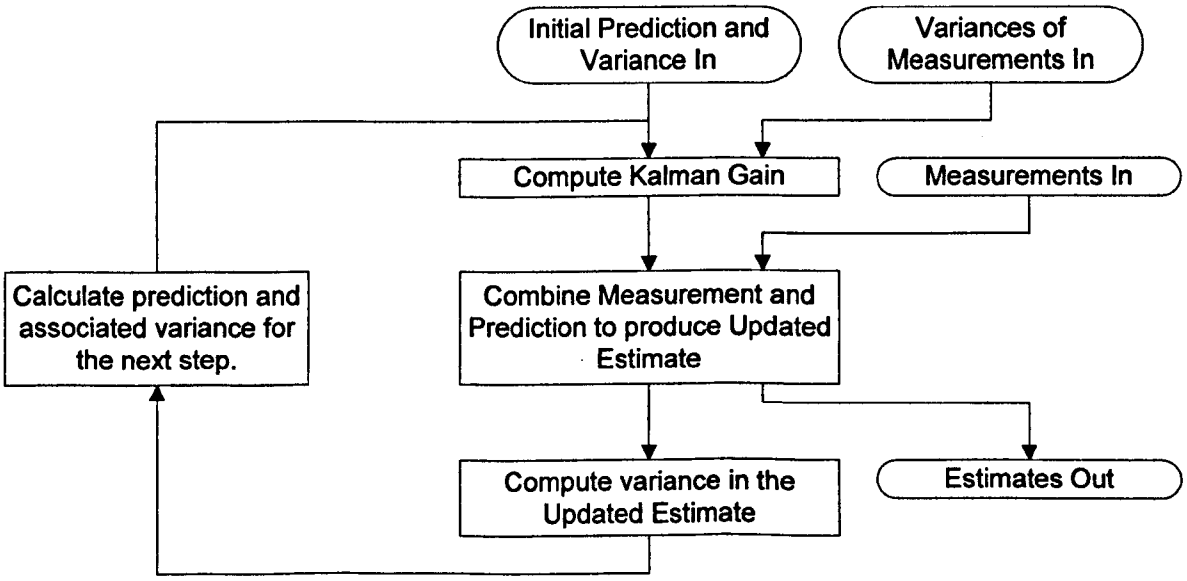


Figure 2.7: Operation of the Kalman Filter.

In Chapter 7, results for the performance of the tracking algorithm are presented with and without the Kalman filter.

2.5 Summary

In this chapter, a case has been made for using a UAV to improve the quality and speed of power line inspection. In order to navigate along the lines, the UAV must

be able to measure its position relative to the lines. The remainder of this thesis is an investigation of using machine vision for this purpose, allowing visual servoing of the UAV to be implemented. The following chapters describe the modelling of the UAV, how the overhead lines are located in the image and the closed-loop visual servoing of the UAV.

Chapter 3 UAV Model

3.1 Introduction

The UAV that is proposed for use in this application is a ducted fan rotorcraft, based on the 'flying platform' principle [10] discussed in section 2.2. This has the Centre of Gravity (CG) deliberately placed above the aircraft centre (AC) to give dynamic stability in hover [12]. Early construction of a laboratory demonstrator version of the craft is shown in Figure 3.1. This rotorcraft is approximately 35cm in diameter and 25cm high; the craft used for inspecting the lines would be larger. As the mathematical model developed in this chapter is for the small laboratory demonstrator, rather than a full sized inspection UAV it will be more easily affected by wind gusts.

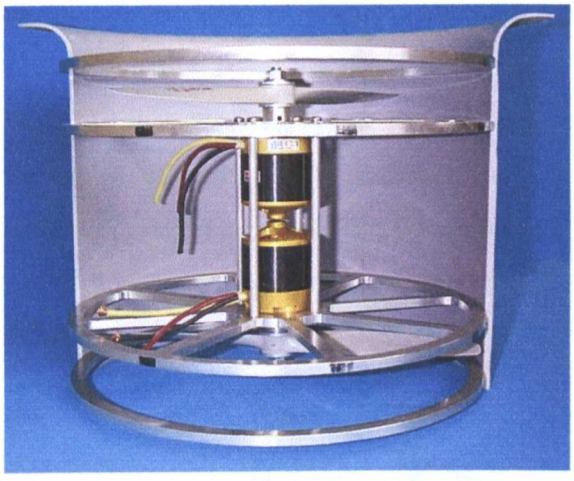


Figure 3.1: Ducted-fan Rotorcraft with Half of the Duct Removed to show the Twin Motors, Counter-rotating Propellers and Internal Construction.

The payload (not shown) is to be mounted above the duct and attitude control is accomplished by moving a mass to change the position of the CG. Lift and yaw control are achieved by changing the propeller speeds collectively or differentially. Overall, the design is quite similar to that described by Sherman et al [11].

3.2 UAV Model

Ando [13] has derived a simple 3 degree of freedom dynamic model for this configuration which suggests that placing the CG within a very small range of locations above the AC gives both static and dynamic stability. In practice, it is anticipated that active (gyro) stabilization will be necessary but starting with a system that has near passive stability is attractive. Ando's model is extended, as shown in Figure 3.2, to include a servo-controlled payload mass (m_p) on a prismatic joint with origin at a distance ℓ above AC; this places the CG a distance h above AC. Moving this mass to the right causes a moment about AC, M_{AC}, causing the duct to 'topple' clockwise and the horizontal thrust component thus produced accelerates it to the right. As the duct moves to the right, a force, H_A, acts against the duct; due to the presence of a lip, or bellmouth, at the top of the duct, H_A produces a moment that tries to return the duct to the upright position. It should also be noted that the velocity, U, the duct velocity, U_D, and H_A in Figure 3.2 are defined in the opposite direction to the expected motion; this is a convention taken from Ando's model. When testing the visual servoing, this model will be applied to the left/right displacement and roll axes, although in this chapter only pitch will be referred to; it should be noted that this model applies equally to the left/right/roll and forward/backward/pitch axes.

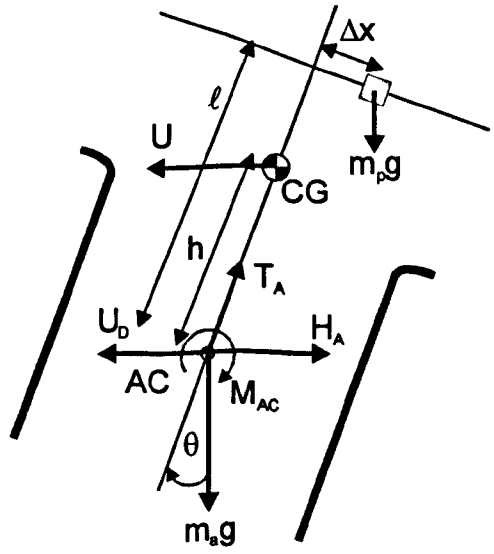


Figure 3.2: Forces, Moments and Velocities for the Ducted-fan Model.

If we assume that the thrust, T_A, is adjusted such that its vertical component balances mg, where $m=m_a+m_p$ and m_a is the aircraft mass, giving a hover condition and that θ remains small, the acceleration of the UAV and the pitch angle are given by (3.1) and (3.2).

$$\dot{U} = -g\theta - \frac{g}{V}U - \frac{hg}{V}\dot{\theta} \quad (3.1)$$

where:
g is the acceleration due to gravity
V is the velocity of air through the duct

$$\ddot{\theta} = \frac{1}{I}M_{AC} + \left(\frac{1}{I}\frac{\partial M}{\partial \dot{\theta}} - \frac{h^2 mg}{IV} + \frac{h}{I}\frac{\partial M}{\partial U}\right)\dot{\theta} + \left(\frac{1}{I}\frac{\partial M}{\partial U} - \frac{hmg}{IV}\right)U \quad (3.2)$$

where:
I is the moment of inertia of the UAV

$\frac{\partial M}{\partial U}$ and $\frac{\partial M}{\partial \dot{\theta}}$ give the contributions to the moment due to the velocity an angular acceleration and are assumed to be constant, as done in [13]. The moment about AC due to moving m_p is given by:

$$M_{AC} = m_p g \Delta x \quad (3.3)$$

If (3.2) and (3.3) are combined and the coefficients of $\dot{\theta}$ and U in (3.2) are replaced by K_1 and K_2 this gives:

$$\ddot{\theta} = \frac{m_p g}{I}\Delta x + K_1\dot{\theta} + K_2 U \quad (3.4)$$

By taking parameters from our lab demonstrator or using Ando's non-dimensionalised values and calculating estimated values for our lab demonstrator. The value of h is chosen in order to give a stable response. The values used are:

m_p=0.7kg

m_a=1.4kg

h=0.125m

V=16.59ms-1

I=0.0656 kgm2

$$\frac{\partial M}{\partial U} = 0.1559Ns$$

$$\frac{\partial M}{\partial \dot{\theta}} = -0.0580Nms$$

If these are substituted into (3.1) and (3.4) this gives:

$$\dot{U} = -9.81\theta - 0.591U - 0.0739\dot{\theta} \quad (3.5)$$

$$\ddot{\theta} = 104.7\Delta x - 0.883\dot{\theta} + 0.0103U \quad (3.6)$$

These equations can be used to simulate the UAV to see how it will perform.

3.3 UAV Simulation

3.3.1 Simulation of the UAV

The equations for the UAV were modelled in Simulink. Initially this was tested with a step input into the deltaX input, giving a ramp position output and a constant speed after the transient response. This showed that the payload needed to move only a tiny amount: 1mm movement of the payload caused a speed of $10ms^{-1}$. This would require the movement of a large mass with great precision. To solve this problem it was decided that the payload should be fixed and instead carry a moveable mass to tip the UAV. This moveable mass forms an actuator to tip the UAV. The mass is modelled as 1% of the payload mass. This changes (3.6) to:

$$\ddot{\theta} = 1.047\Delta x - 0.883\dot{\theta} + 0.0103U \quad (3.7)$$

Equations (3.5) and (3.7) were modelled in Simulink. Figure 3.3 shows the Simulink model of the UAV; the direction of U, shown in Figure 3.2 is opposite to the expected direction of travel; in order to correct this the Simulink model incorporates an inverting gain on U. The UAV model in Figure 3.3 was made into a Simulink subsystem and tested using the Simulink model shown in Figure 3.4.

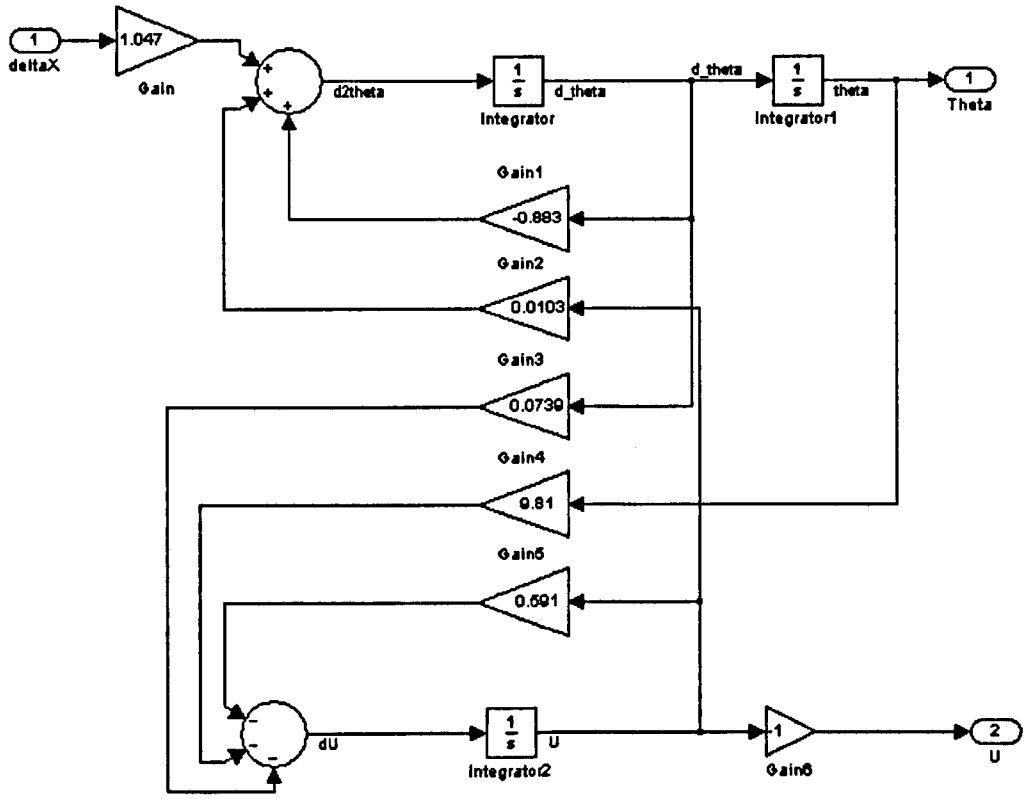


Figure 3.3: Simulink Model of UAV.

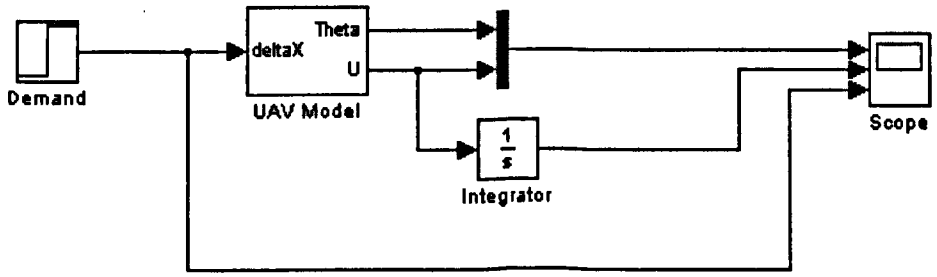


Figure 3.4: Simulink Model used to Test the UAV Model.

To test the UAV model, a step input was put into the actuator input (deltaX) to tip the duct. It would be expected that this would produce a constant speed response,

after the initial transient, and an increasing position response at a constant rate. This is because moving the actuator causes the duct to tip. However, as the duct tips, a back force, H_A, acts such as to stop the duct tipping. The tipping moment from the actuator is balanced by H_A meaning that, for a given actuator input, the duct settles at a constant pitch and so has a constant speed.

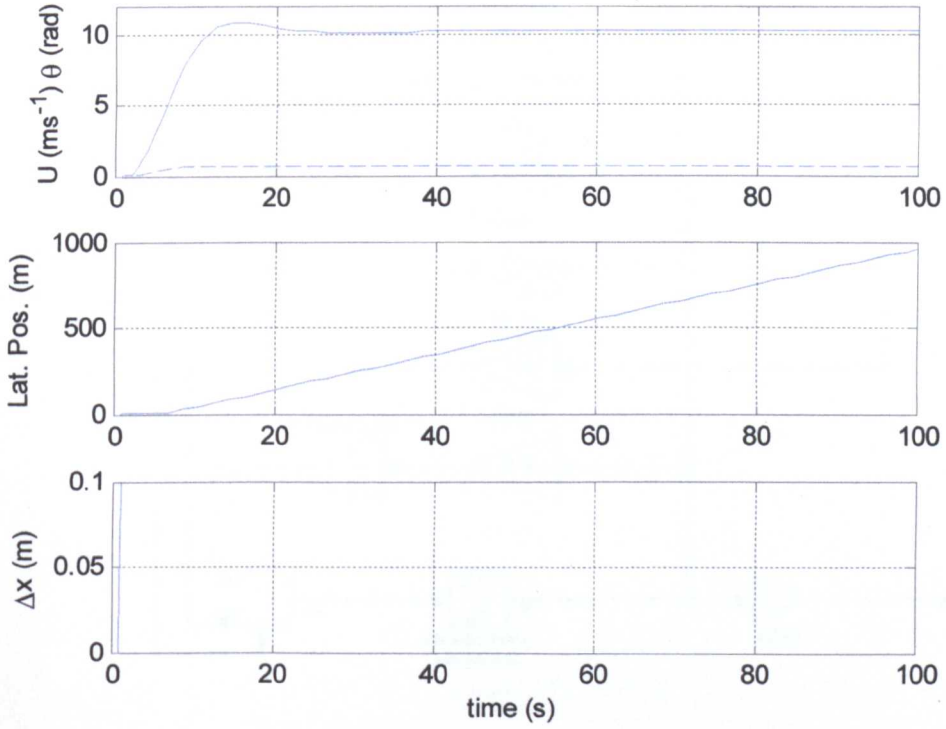


Figure 3.5: UAV Position, Speed and Pitch Response.

Figure 3.5 shows the result of putting a step input into the UAV's actuator. The upper graph shows the velocity, U, (solid) and pitch angle, θ, (dashed), while the middle graph shows the lateral position of the duct and the lower graph shows the step input to the actuator. The graphs show that the UAV has quite a slow response time. The response is as expected for the chosen CG position.

As the UAV will be simulated in real-time on a computer at a fixed sample rate, it is necessary for the model to work in discrete time. The model will be run at 25Hz, which is considerably faster than the response of the UAV; according to [47] a discrete approximation to integration should give a good match with the continuous time model. As backward Euler integrators are easy to code in software, the model was converted to discrete time using the backward Euler rule.

Figure 3.6 shows the discrete-time Simulink model of the UAV; the unit delay blocks represent the fact that the values used come from the previous time-step. This model was tested using the Simulink model shown in Figure 3.7.

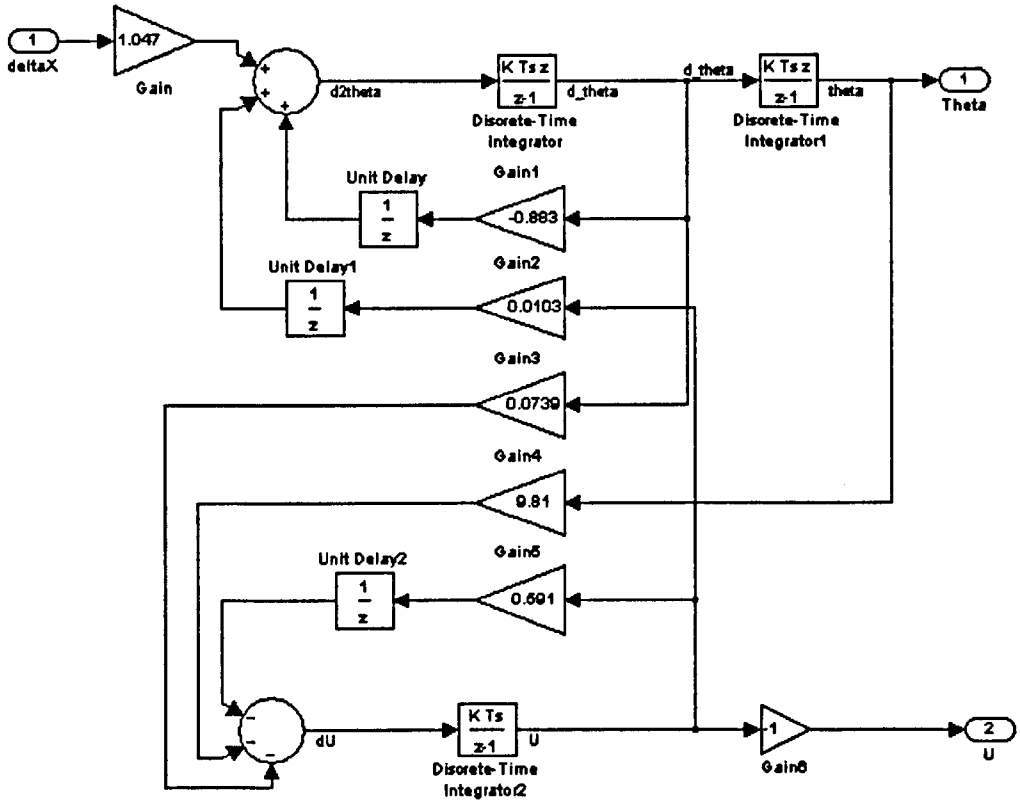


Figure 3.6: Discrete-time Version of UAV Model.

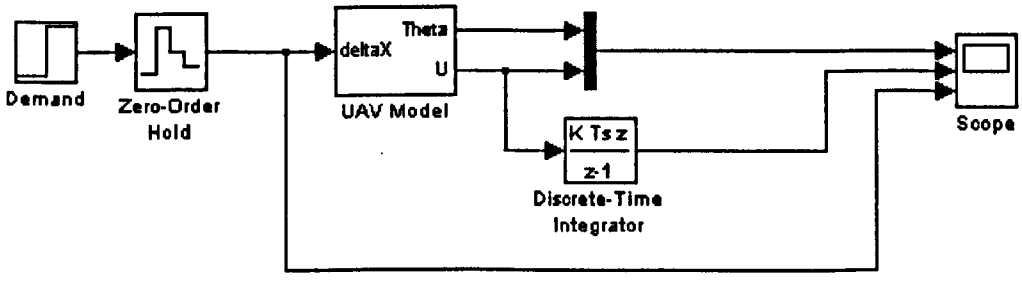


Figure 3.7: Simulink Model used to Test the Discrete-time UAV Model.

Figure 3.8 shows that the response of the discrete UAV model matches well with the continuous time version.

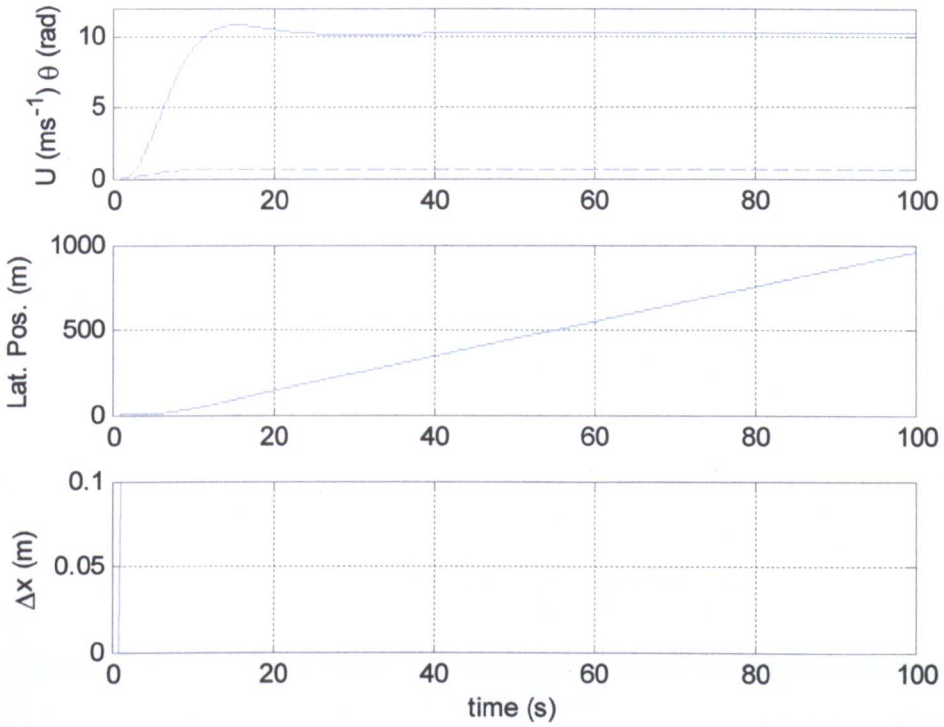


Figure 3.8: Discrete-time UAV Model Response.

3.3.2 Position Control Loop

This application requires position control of the UAV and so a position control loop was wrapped around the UAV model in Figure 3.6. The aim is to control the lateral position of the UAV with respect to the overhead lines. This requires a lateral position feedback loop. The system is shown in Figure 3.9. Normally, the lateral position of the UAV is estimated from image processing shown in the vision-processing loop. In order to bring the UAV into the vicinity of the lines and in case vision feedback fails, there is also a feedback loop using DGPS. The model switches to vision feedback when the lines are acquired and away to DGPS if the lines are lost or the UAV strays to far from the lines.

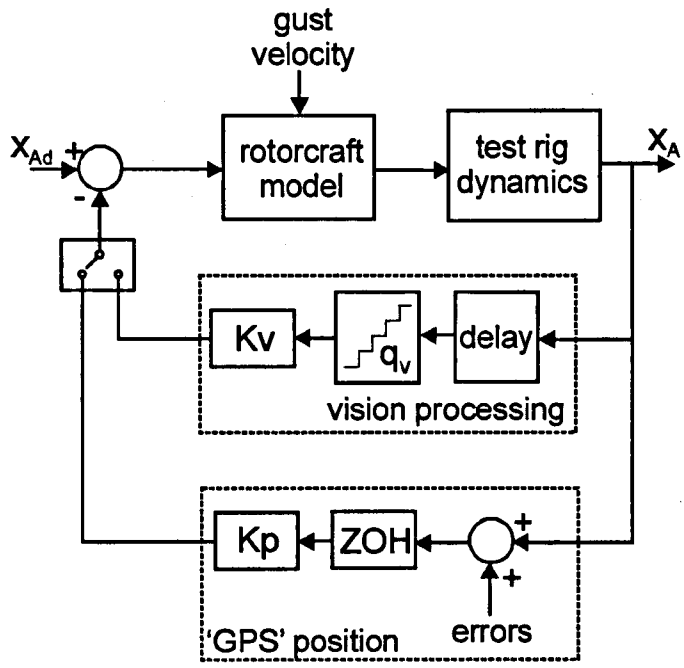


Figure 3.9: System Model, where X_A is the lateral position resulting from the demand position, X_{Ad}.

SISOtool was used to select an optimal position loop gain (K_P or K_V). Working from the block diagram in Figure 3.3, the transfer function of the UAV, from Δx to U is:

$$\frac{U(s)}{\Delta x(s)} = \frac{0.0774s + 10.27}{s^3 + 1.474s^2 + 0.523s + 0.101} \quad (3.8)$$

The speed, U, is integrated to give the position, X. The transfer function from Δx to X is:

$$\frac{X(s)}{\Delta x(s)} = \frac{0.0774s + 10.27}{s^4 + 1.474s^3 + 0.523s^2 + 0.101s} \quad (3.9)$$

The transfer function for the position of the UAV was entered into SISOtool and the resulting root locus is shown in Figure 3.10. In order to see the pole, it was necessary to zoom in on the region around the origin; the zoomed in version is shown in Figure 3.11.

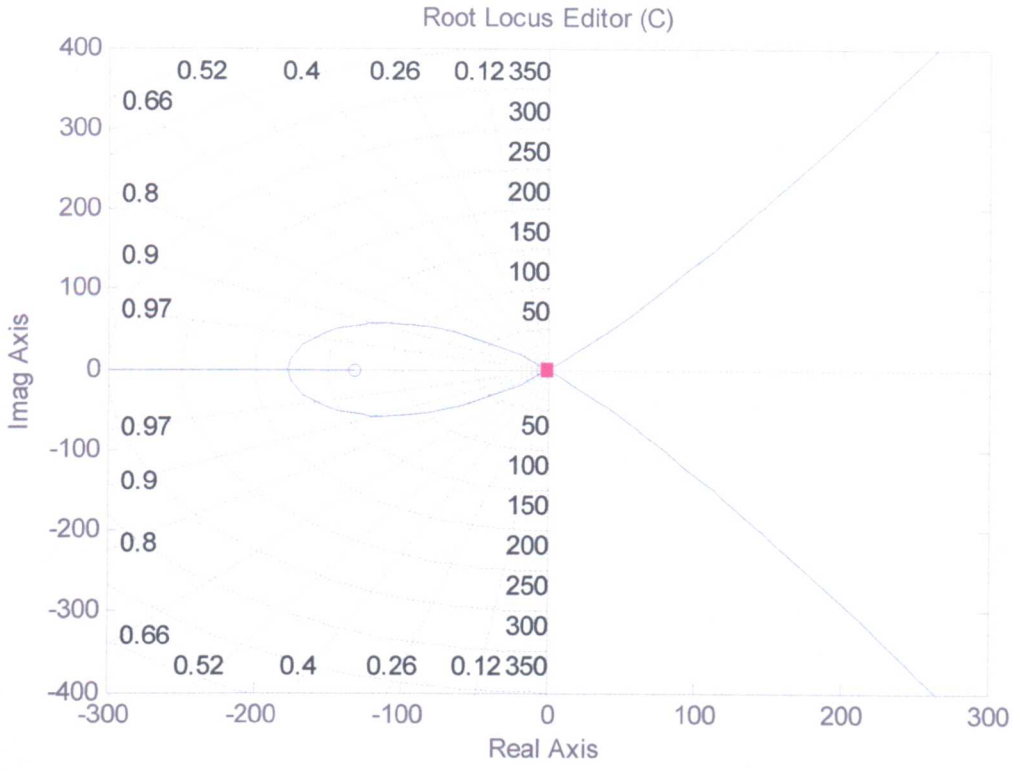


Figure 3.10: Root Locus for the UAV.

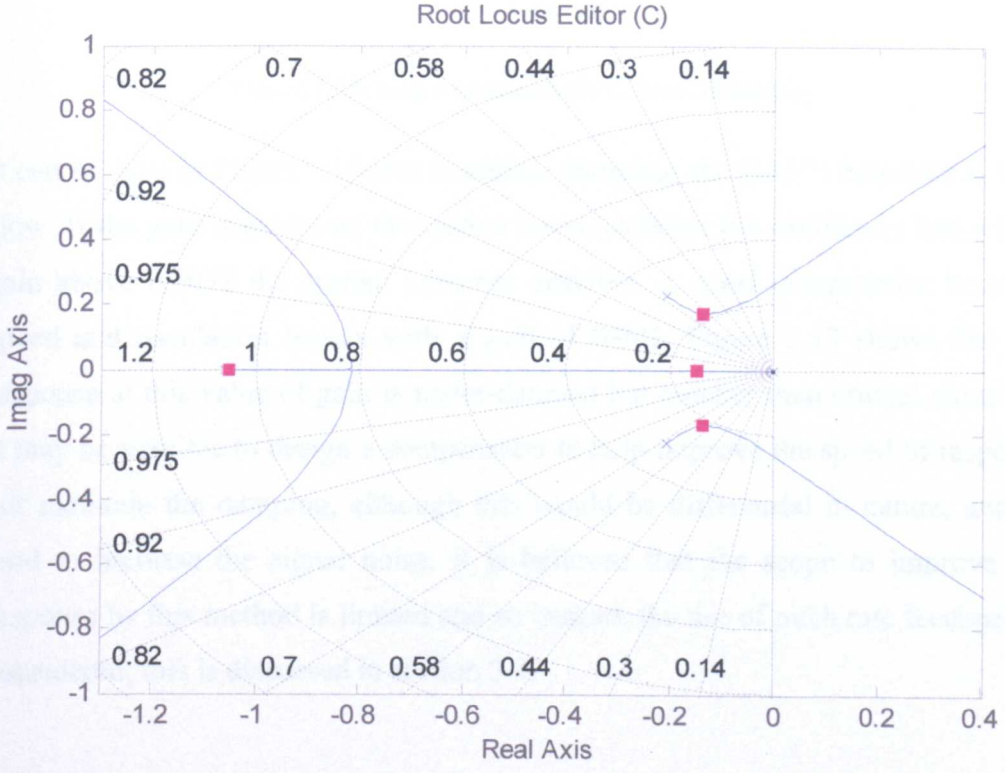


Figure 3.11: Zoomed-in Version of the Root Locus for the UAV.

Critical damping occurs with a gain of 0.00074. The step response is shown in Figure 3.12.

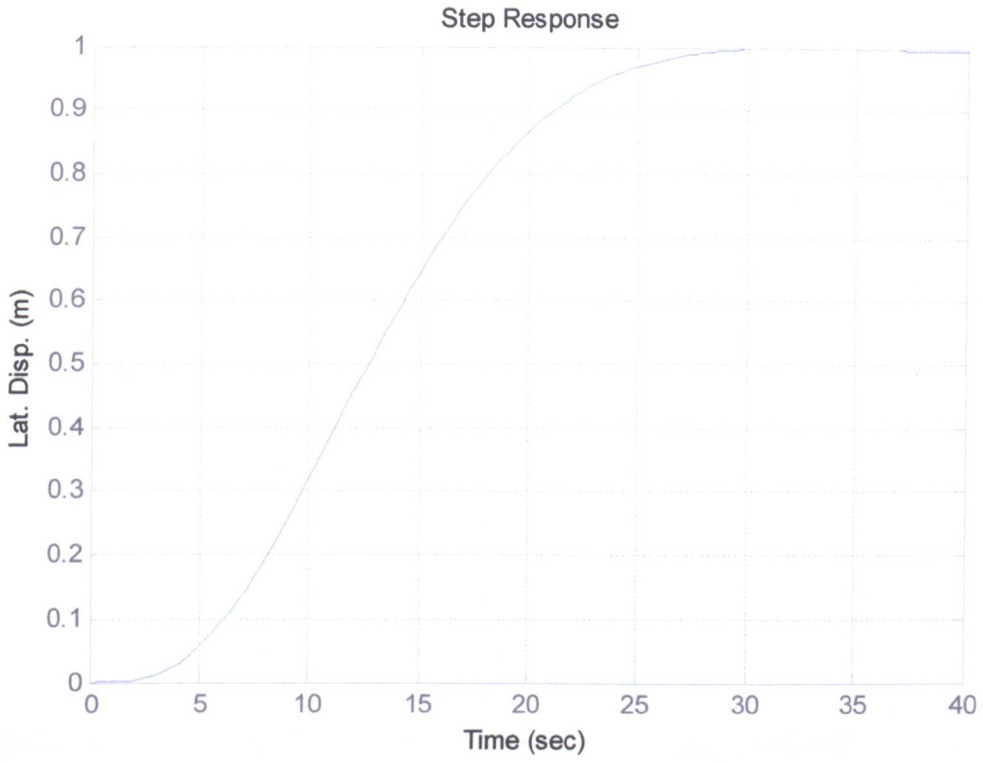


Figure 3.12: Step Response with Critical Damping.

It can be seen in Figure 3.12 that at critical damping, the UAV's response is very slow. If the gain is increased the system becomes faster but oscillatory and with a gain above 0.0032 the system becomes unstable. A good compromise between speed and oscillation occurs with a gain of 0.001. Figure 3.13 shows that the response at this value of gain is under-damped but quicker than critical damping. It may be possible to design a compensator to help improve the speed of response but maintain the damping, although this would be differential in nature, and so tend to increase the signal noise. It is believed that the scope to improve the response by this method is limited and so instead, the use of pitch rate feedback is considered; this is discussed in section 3.4.

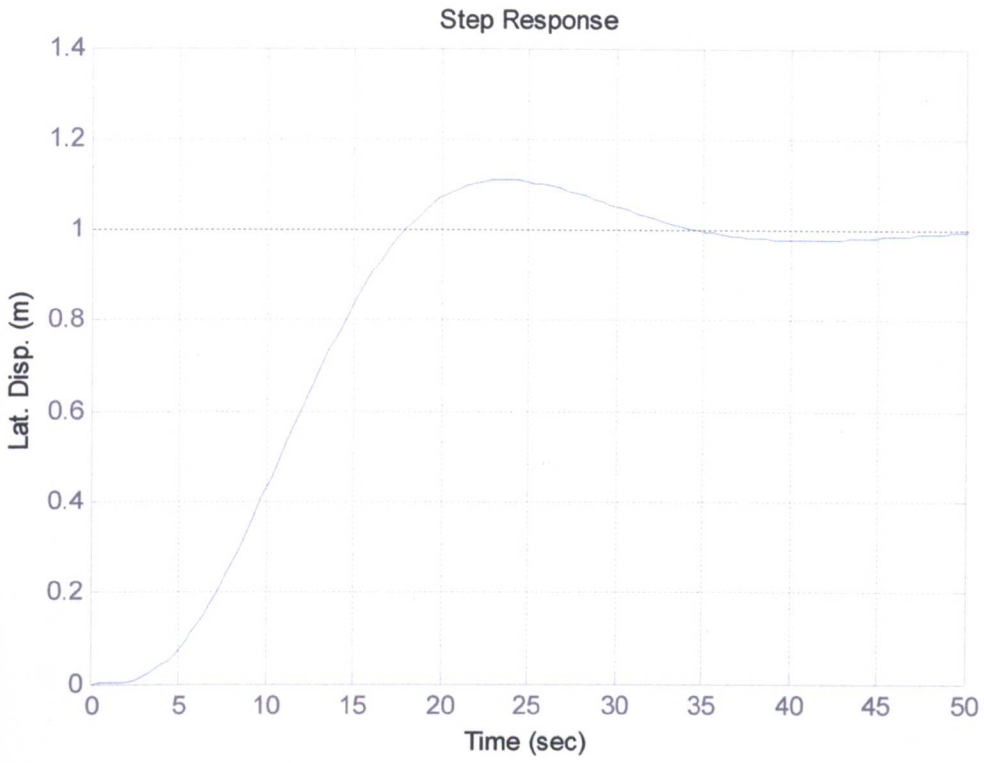


Figure 3.13: Underdamped Step Response for a Loop Gain of 0.001.

As the system is actually being run in discrete time, the continuous to discrete time conversion function of SISOtool was used to convert the root locus to discrete time. The step response for the discrete time model could then be seen. The discrete root locus is shown in Figure 3.14 and Figure 3.15 shows the root locus zoomed in on the unit circle; the discrete step response is shown in Figure 3.16.

It can be seen from Figure 3.16 that the response is virtually identical to the continuous time model. As with the continuous time case, it can be seen from Figure 3.15 that the poles are close to the stability margin.

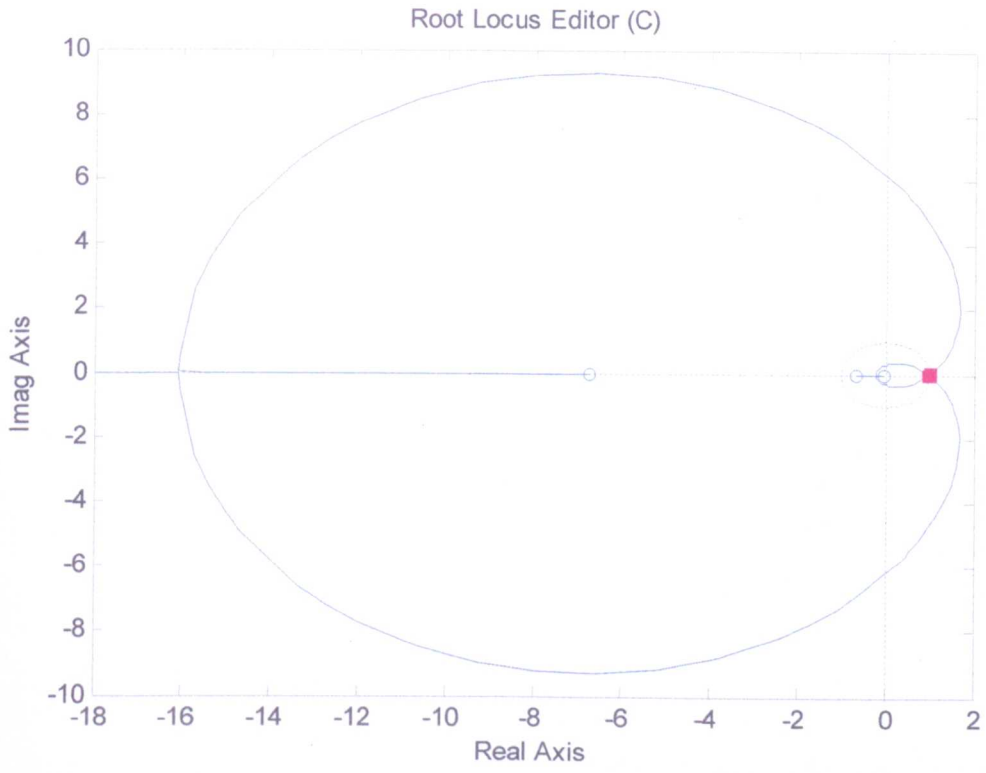


Figure 3.14: Discrete-time Root Locus for the UAV.

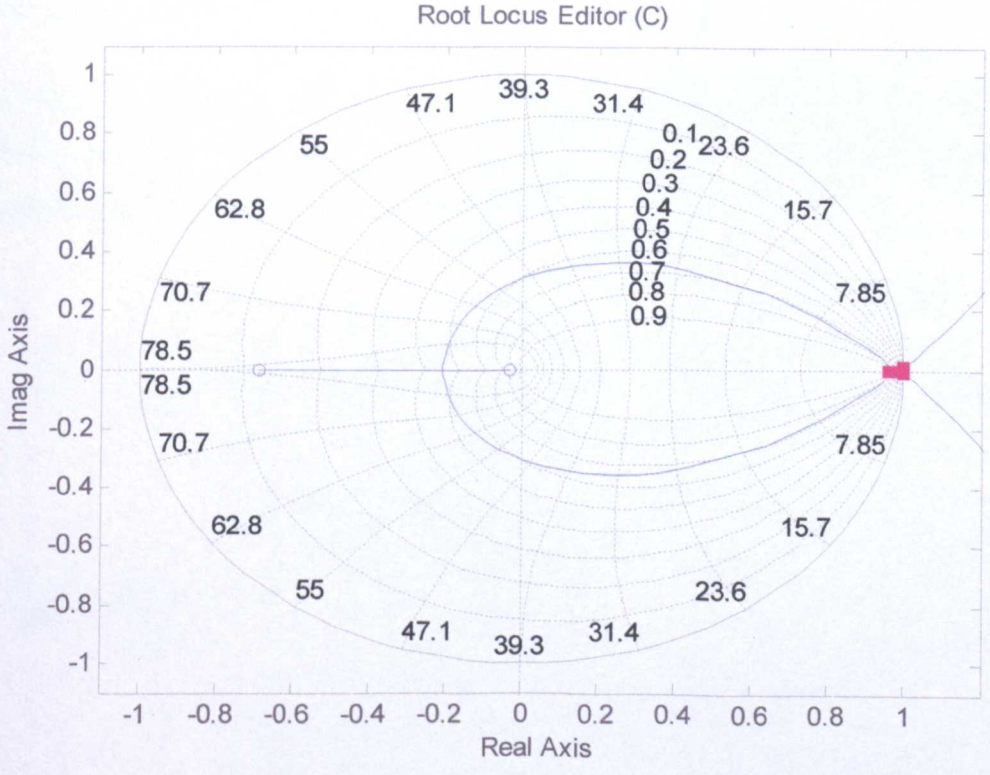


Figure 3.15: Zoomed-in Version of the Discrete-time Root Locus for the UAV.

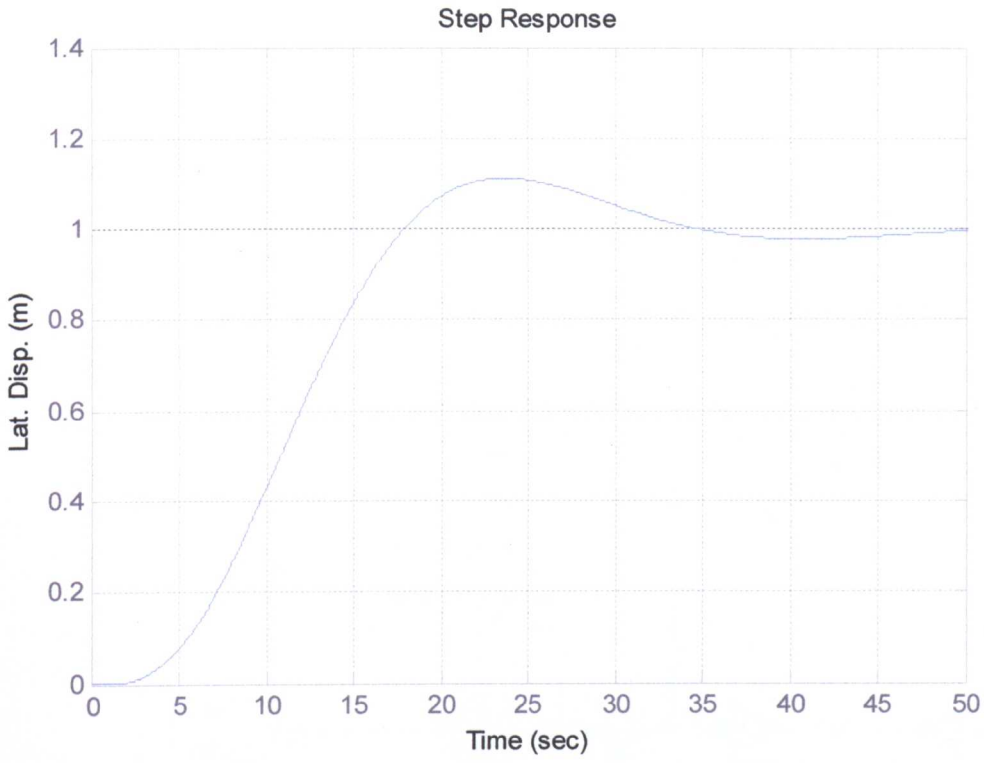


Figure 3.16: Discrete-time Step Response for a Loop gain of 0.001.

3.3.3 Simulation of Position Feedback Control of the UAV

As the UAV will be subjected to wind gusts, the model of Figure 3.9 incorporates their effect into the simulation. Figure 3.18 shows the UAV model with wind gust input while Figure 3.17 shows the position controller.

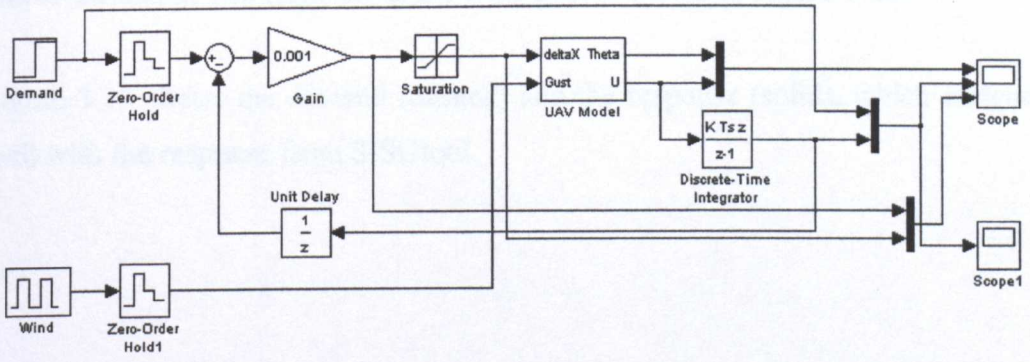


Figure 3.17: Discrete UAV Position Controller.

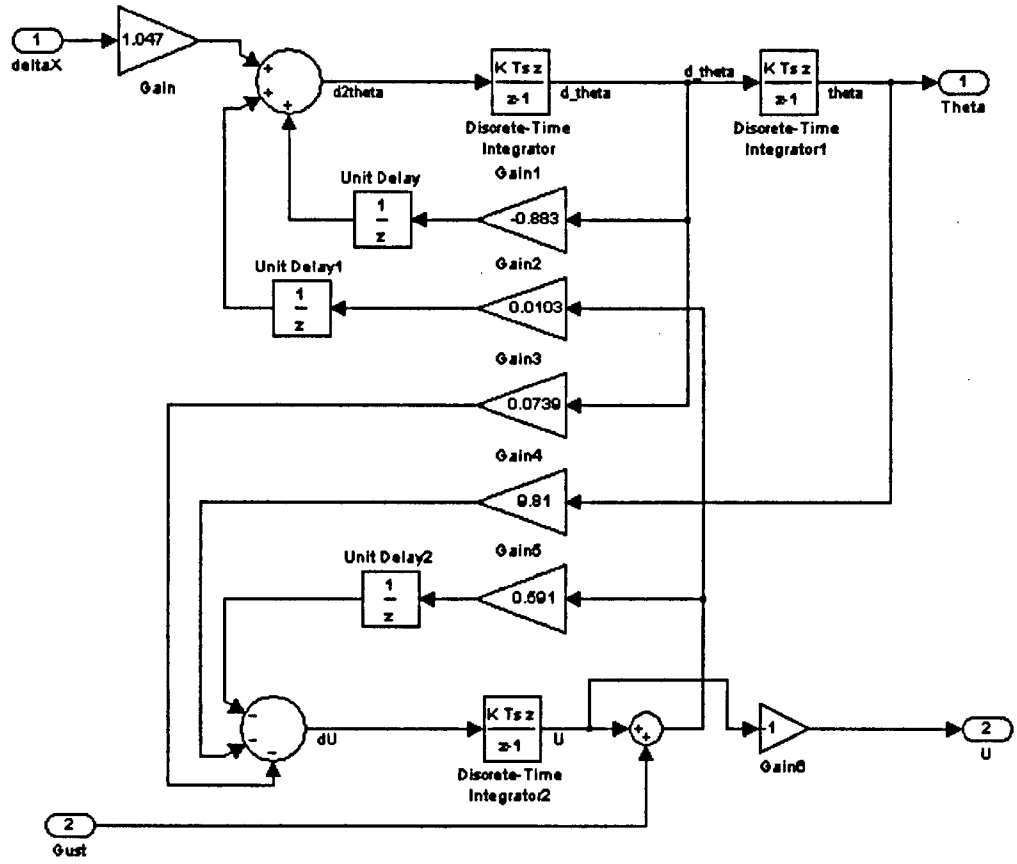


Figure 3.18: UAV Model with Wind Gust Input.

The UAV was tested with a step demand of 5m and its response is shown in Figure 3.19. In order to test the response to wind, a pulse wind gust of strength $1ms^{-1}$ for 5s was applied. This is quite a strong gust for a small UAV like the one that is modelled. The response to the wind gust is shown in Figure 3.20.

Figure 3.19 shows the demand (dashed) and the response (solid), which matches well with the response from SISOtool.

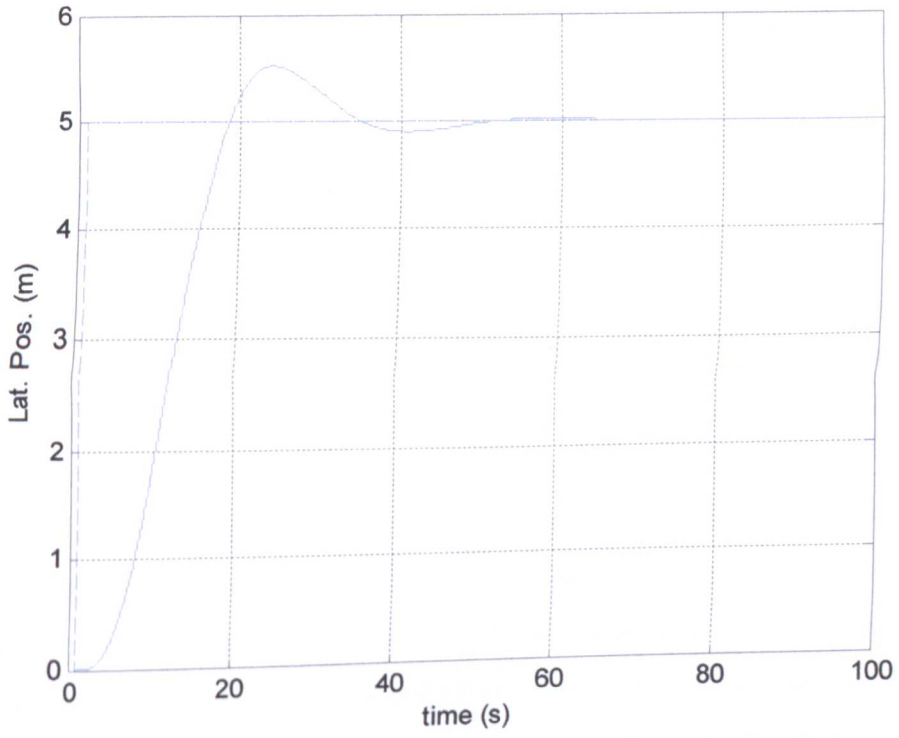


Figure 3.19: UAV Step Response.

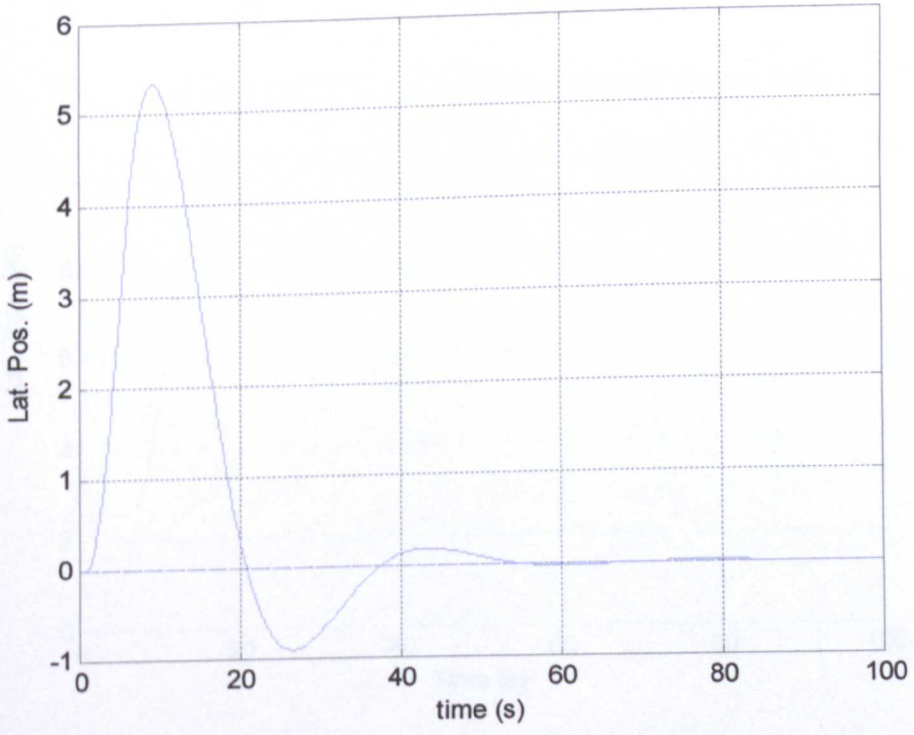


Figure 3.20: Pulse Wind Gust Response of UAV Model.

Figure 3.20 shows that the system is able to recover from a wind gust, but is quite slow to do it; it is also displaced a considerable distance from the demanded position (dashed). It should be noted that, while the wind gust to cause the effect seen in Figure 3.20 may seem small, the UAV modelled is a small laboratory demonstrator, which is much more easily affected by wind compared to a larger UAV, as would actually be used for inspection purposes.

The UAV model was programmed into the test rig control computer (to be described in Chapter 5). The position response of the UAV to a step input to the UAV's actuator (Δx) could then be obtained when it is being simulated by the laboratory test-rig. Figure 3.21 shows the response from the Simulink model (dash-dotted) and the measured rig response (solid). It can be seen that there is a good match between the two. Also shown is the response of the bare test rig (i.e. no UAV model (dashed)). It can be seen that the natural test-rig response is much faster than the UAV and so it is able to simulate the UAV's movement accurately.

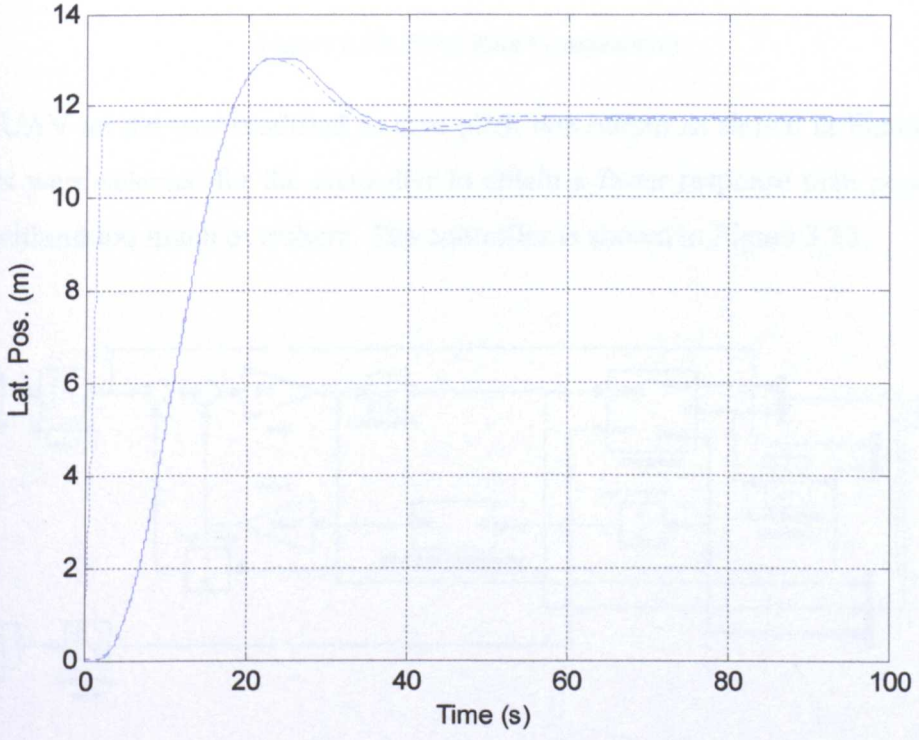


Figure 3.21: Comparison of the Step Response Produced by the Test Rig and the Off-line Simulation; the Raw Test rig Response is also shown.

3.4 Pitch Rate Feedback

The UAV's speed of response should improve with the addition of pitch rate feedback. An investigation into this was undertaken by Dr Dewi Jones, which showed an improved response speed if a pitch rate gyro and compensator are added. The gyro bandwidth is significantly faster than the UAV dynamics and so could be omitted from the model. The compensator transfer function is shown in (3.10) and the Simulink model in Figure 3.22.

$$C_{PR}(s) = \frac{0.4545s + 1}{10s^2 + 100.1s + 1} \quad (3.10)$$

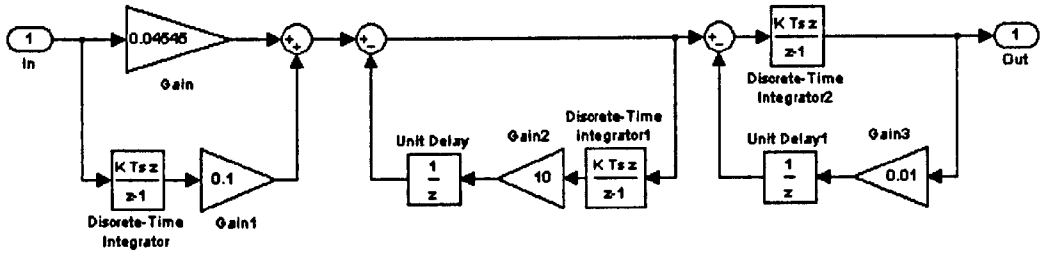


Figure 3.22: Pitch Rate Compensator.

The UAV model was modified to give pitch rate output as shown in Figure 3.24. Gains were selected for the controller to obtain a faster response than previously but without too much overshoot. The controller is shown in Figure 3.23.

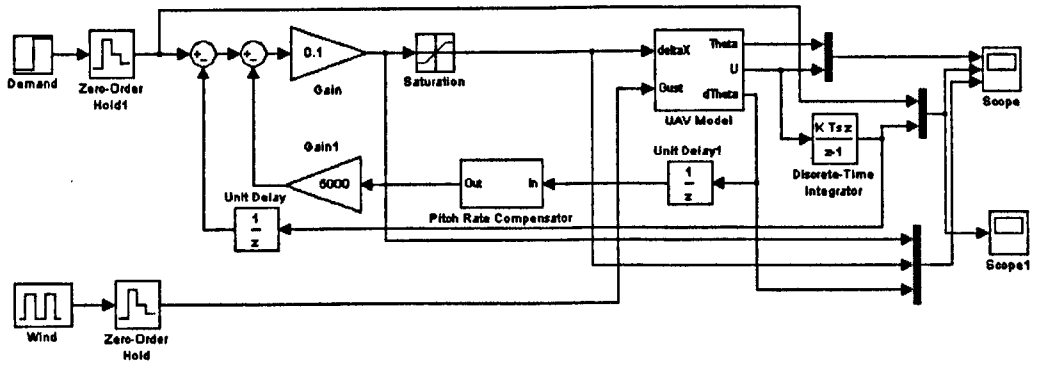


Figure 3.23: Controller with Pitch Rate Feedback.

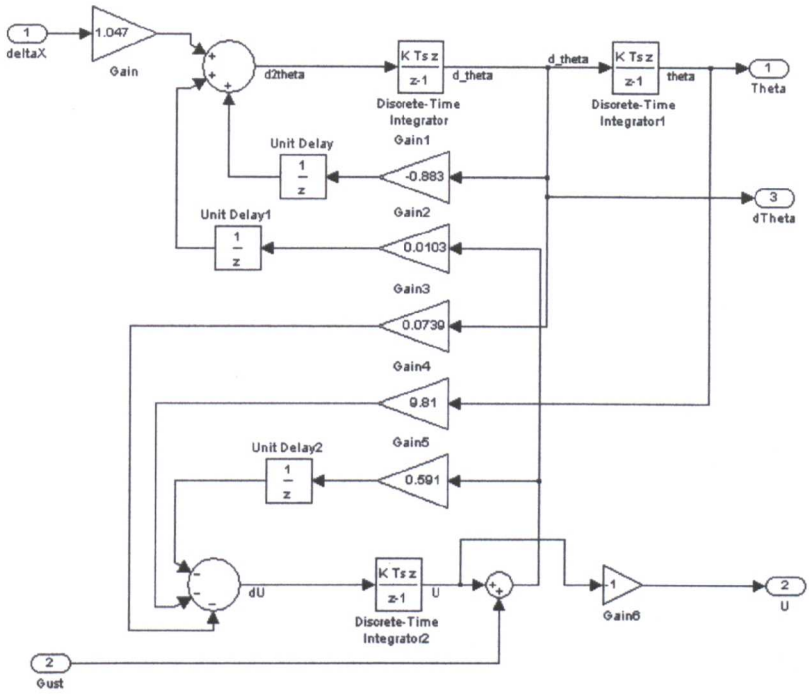


Figure 3.24: UAV Model with Pitch Rate Output.

As with the previous controller, the UAV model with pitch rate feedback was tested with a step demand of 5m and a wind gust of $1ms^{-1}$ for 5s.

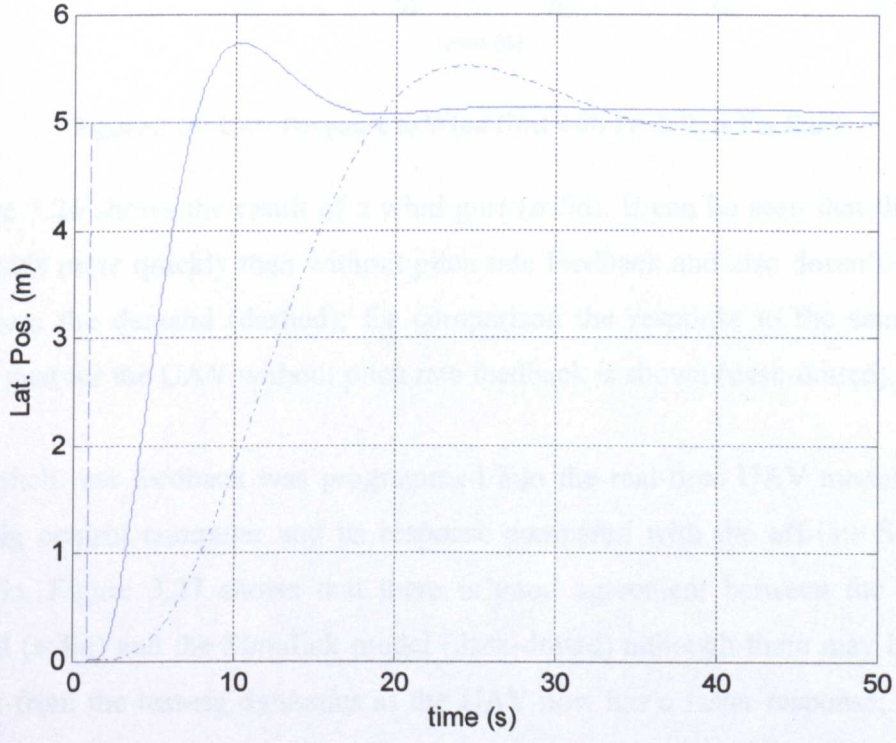


Figure 3.25: Step Response of UAV with Pitch Rate Feedback.

Figure 3.25 shows the UAV response (solid) to a step demand (dashed); for comparison the step response of the UAV without pitch rate feedback is shown (dash-dotted). It can be seen that the UAV responds in about half the time compared with the model with no pitch rate feedback, although there is now a small steady state error.

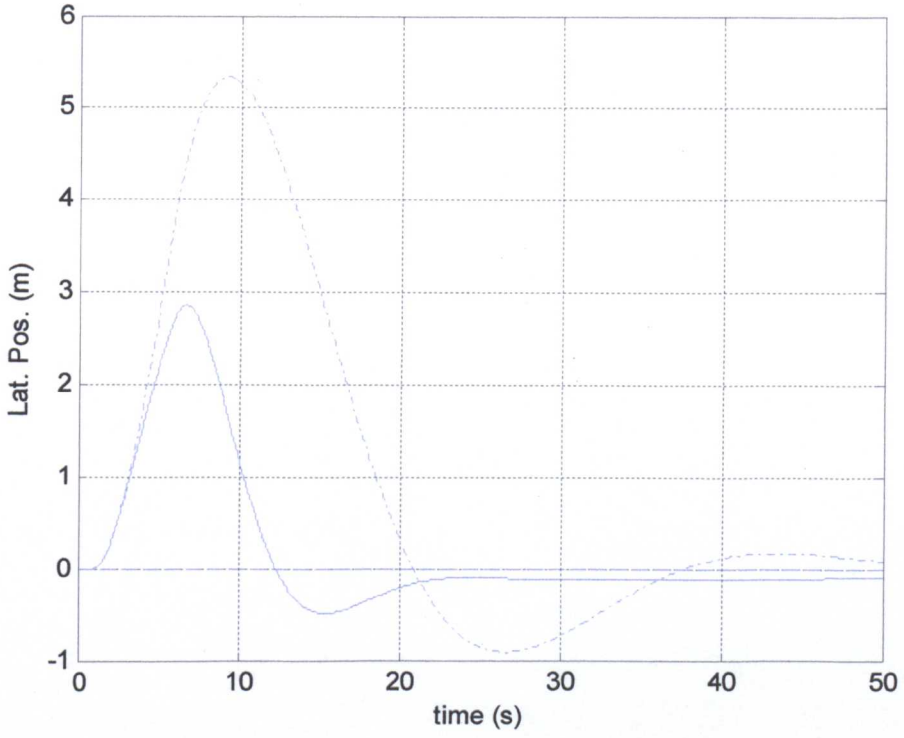


Figure 3.26: UAV Response to Wind Gust with Pitch Rate Feedback.

Figure 3.26 shows the result of a wind gust (solid). It can be seen that the UAV responds more quickly than without pitch rate feedback and also doesn't stray as far from the demand (dashed); for comparison the response to the same sized wind gust for the UAV without pitch rate feedback is shown (dash-dotted).

The pitch rate feedback was programmed into the real-time UAV model on the test rig control computer and its response compared with the off-line Simulink version. Figure 3.27 shows that there is good agreement between the test rig model (solid) and the Simulink model (dash-dotted) although there may be some effect from the test-rig dynamics as the UAV now has a faster response; the raw test rig response is shown (dotted), showing that the rig time constant remains well below that of the new UAV model. Often in Hardware-in-the-Loop

simulators an inverse transfer function of the test-rig is included in order to try to remove the dynamics of the rig; this would likely become necessary in this project if we wish to simulate a faster UAV. Also shown in Figure 3.27 is the UAV model with position feedback only (dashed), showing the improved response speed with pitch rate feedback.

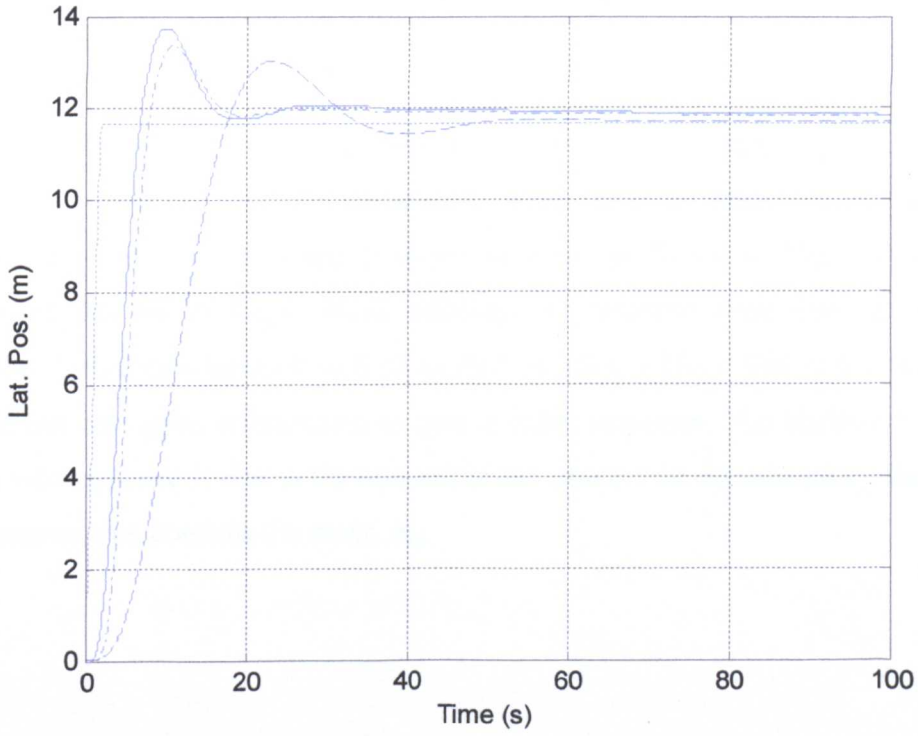


Figure 3.27: Comparison of Step Responses of UAV Model with and without Pitch Rate Feedback and Test Rig.

3.5 Conclusions

A model of the UAV has been developed in this chapter. With pitch-rate feedback it has a reasonably fast position response but with some overshoot. It will give a good basis on which to test the vision software. It should be noted that, as construction of the laboratory demonstrator is not complete, it has not been possible to validate the model with data from the real craft. Also, the model is limited to two degrees of freedom, and it is currently assumed that there will be little interaction between the forward/backward direction and the left/right model. This is likely to be a reasonable assumption, due to the symmetrical nature of the UAV, but again there is no experimental data to validate this. There will be some

interaction between pitch and lift, as when the UAV pitches, a component of the thrust acts horizontally and so the vertical thrust is reduced. It should also be noted that the model is linear and at extremes, the small angle assumptions break down. The model assumes that the aerodynamic derivatives are constant but, in practice, they change with operating conditions. What this means in practice is that you can arrange for the CG to be within a small interval that gives stability but this interval changes as the aerodynamic derivatives change, so getting true passive stability is unlikely.

The current model is for a small laboratory demonstrator, rather than a full sizes UAV, as would be used for inspection. When data for such a craft becomes available it could be simulated. It would be expected that it will be less affected by wind, due to its larger mass, although its response time may be slower. Another future development will be to look at using a UAV that is not passively stable but uses gyro stabilisation to give a faster response. The limitation on the pitch rate response is due to the non-minimum phase zero introduced by the servo mechanism that controls the mass, m_p.

Chapter 4 Image Geometry

4.1 Introduction

This chapter describes a mathematical model of the effect of UAV motion on the resulting image. The geometric transformation of the lines in the 3D world into the 2D image is modelled in order to predict how the UAV's motion affects their position in the image. Ideally, processing the image should yield the height, lateral displacement, yaw, pitch and roll of the vehicle with respect to the lines. It would not be expected to obtain its distance along the lines from image processing, as their position in the image is invariant to the UAV's longitudinal position. It is shown here that height and pitch can be estimated from processing the frames from a single camera pointing forward and angled down along the lines, but that yaw, roll and lateral displacement cannot be fully separated. It is also shown that the use of a second camera, pointing backwards, allows the separation of yaw, roll and lateral displacement.

4.2 Analysis for One Camera

The camera is assumed to be mounted on the UAV looking forwards. The UAV will pitch forward in order to travel along the lines and so this will cause the camera to be pointed down at the lines. The pitch angle that will be used for flying the UAV forward along the lines on the actual UAV used for inspection is not yet known; it may be necessary to pitch the forward camera down relative to the duct. Currently the pitch angle is assumed to be 20°. Figure 3.8 indicates that this corresponds to a forward speed of $5ms^{-1}$. This is a reasonable speed for the final UAV to travel at along the lines, although it may be on the high side. It is not currently known how this speed will scale up with a full-sized UAV. This camera configuration was chosen as it allows the lines to be seen but also to see ahead along the lines. A camera pointing vertically downwards may have some advantage for measuring yaw and lateral displacement relative to the lines, but this would need to be mounted below the duct and on the proposed UAV the payload is above the duct; in addition the image would be complicated by the power pick-up appearing in it. The position of the power lines is known in world

space, {W}. In order to see how a line will appear in the image its co-ordinates must be transformed into a reference frame centred on the camera lens, {C} as shown in Figure 4.1.

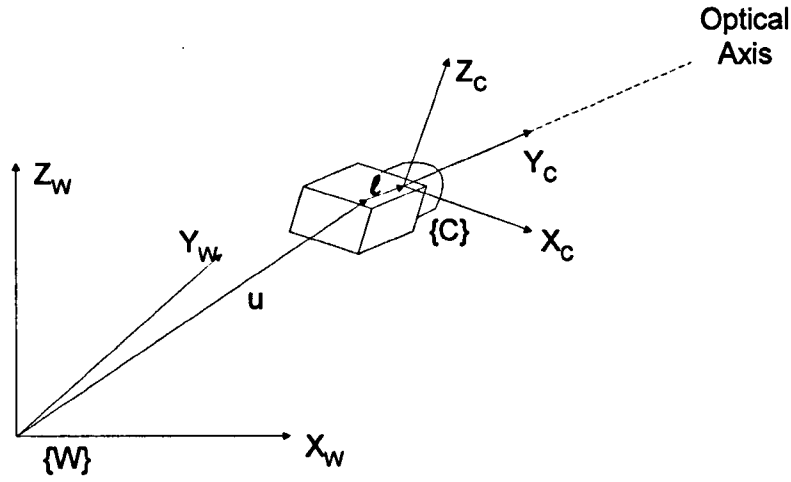


Figure 4.1: Reference Frame Definitions.

4.2.1 Mathematical Model

The relationship between the position of a point on the power line in the world space {W} and its image co-ordinates is defined by a sequence of transformations [48], based on the reference frames shown in Figure 4.1. The co-ordinates (X_C, Y_C, Z_C) of a point in {C} are related to a point in {W} by a translation, u, from the origin in {W} to the centre of the camera mount, three rotations through the Euler angles yaw (α), pitch (β) and roll (γ) to a reference frame centred on the UAV, {A}. This is followed by a translation, ℓ, to the centre of the camera lens, {C}. It should be noted that α, β and γ are being used in this case for the yaw, pitch and roll, rather than the conventional ψ, θ and ϕ, as θ is used for one of the Hough Transform variables. In order to create the image there is then a perspective transformation into the image. The transformations are given in homogeneous co-ordinates, as discussed in [48]. The sequence of transformations is given by:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\gamma & 0 & \sin\gamma & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\gamma & 0 & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -X_u \\ 0 & 1 & 0 & -Y_u \\ 0 & 0 & 1 & -Z_u \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}$$

(4.1)

where ℓ is the distance between the rotation point of the camera and the lens centre and λ is its focal length. Dividing the X_C, Y_C and Z_C co-ordinates by $-Y_C/\lambda$ gives the points in the (x, z) image plane, placed at $Y_C = -\lambda$. These are given by:

$$x = \frac{X_C}{-\frac{Y_C}{\lambda}} \text{ and } z = \frac{Z_C}{-\frac{Y_C}{\lambda}} \quad (4.2)$$

4.2.1.1 Analysis for Lateral Displacement

Applying equation (4.1) to a straight-line model of the centre conductor, placed at $X_W = 0$ and $Z_W = -Z_L$, where Z_L is the vertical height of the camera centre above the line, generates a corresponding line in the image. Applying the Hough Transform then gives ρ and θ values of the lines as a function of the camera pose. Consider, for instance, an UAV that is displaced laterally by X_u to either side of the centre line. Assume that the vehicle is flying along the lines at constant speed, and so has a fixed pitch, β. Assume also that α and γ are zero. Equation (4.1) becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & -X_u \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ Y_W \\ -Z_L \\ 1 \end{bmatrix} \quad (4.3)$$

This becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} -X_u \\ Y_W \cos\beta + Z_L \sin\beta - \ell \\ Y_W \sin\beta - Z_L \cos\beta \\ -\frac{Y_W \cos\beta + Z_L \sin\beta - \ell}{\lambda} \end{bmatrix} \quad (4.4)$$

From (4.4), using (4.2), the image co-ordinates are given by:

$$x = \frac{-\lambda X_u}{Y_W \cos\beta + Z_L \sin\beta - \ell} \text{ and } z = \frac{\lambda(Y_W \sin\beta - Z_L \cos\beta)}{Y_W \cos\beta + Z_L \sin\beta - \ell} \quad (4.5)$$

Applying the Hough Transform then gives:

$$\theta = \tan^{-1}\left(\frac{x}{z}\right) \text{ and } \rho = x\cos\theta - z\sin\theta \quad (4.6)$$

and hence:

$$\theta = \tan^{-1}\left(\frac{-X_u}{Y_W \sin\beta - Z_L \cos\beta}\right) \text{ and}$$

$$\rho = \frac{-\lambda X_u}{Y_W \cos\beta + Z_L \sin\beta - \ell}\cos\theta - \frac{\lambda(Y_W \sin\beta - Z_L \cos\beta)}{Y_W \cos\beta + Z_L \sin\beta - \ell}\sin\theta \quad (4.7)$$

Constants K_{X1}, K_{X2} and K_{X3} can be defined as follows:

$$K_{X1} = Y_W \sin\beta - Z_L \cos\beta \quad (4.8)$$

$$K_{X2} = \frac{\lambda(Y_W \sin\beta - Z_L \cos\beta)}{Y_W \cos\beta + Z_L \sin\beta - \ell} \quad (4.9)$$

$$K_{X3} = \frac{\lambda}{Y_W \cos\beta + Z_L \sin\beta - \ell} \quad (4.10)$$

Re-arranging (4.12) and evaluating the constants K_{X1}, K_{X2} and K_{X3} for fixed values of ℓ, β, λ, Y_W and Z_L gives:

$$\theta = \tan^{-1}\left(\frac{X_u}{K_{X1}}\right) \text{ and}$$

$$\rho = -\frac{X_u}{K_{X3}}\cos\left(\tan^{-1}\left(\frac{X_u}{K_{X1}}\right)\right) - K_{X2}\sin\left(\tan^{-1}\left(\frac{X_u}{K_{X1}}\right)\right) \quad (4.11)$$

This converts to:

$$\theta = \tan^{-1}\left(\frac{X_u}{K_{X1}}\right) \text{ and}$$

$$\rho = -\frac{X_u}{K_{X3}}\frac{1}{\sqrt{1+\left(\frac{X_u}{K_{X1}}\right)^2}} - K_{X2}\frac{\frac{X_u}{K_{X1}}}{\sqrt{1+\left(\frac{X_u}{K_{X1}}\right)^2}} \quad (4.12)$$

where both ρ and θ are seen to vary with lateral displacement, X_u.

4.2.1.2 Analysis for Roll

Applying the same procedure to the roll axis, γ, (assuming X_u is zero), gives the image co-ordinates as:

$$x = -\lambda\left(\frac{Y_W \sin\beta\sin\gamma - Z_L \cos\beta\sin\gamma}{Y_W \cos\beta + Z_L \sin\beta - \ell}\right) \text{ and}$$

$$z = -\lambda\left(\frac{Y_W \sin\beta\cos\gamma - Z_L \cos\beta\cos\gamma}{Y_W \cos\beta + Z_L \sin\beta - \ell}\right) \quad (4.13)$$

and applying the Hough Transform gives:

$$\theta = \gamma \text{ and } \rho = 0 \quad (4.14)$$

In (4.14) θ changes with roll angle while ρ is identically zero. Full analysis is presented in Appendix A.

4.2.1.3 Analysis for Yaw

If the same procedure is applied to the yaw axis, α, with γ and X_u kept at zero, then the image co-ordinates are:

$$x = \lambda\frac{Y_w \sin\alpha}{Y_W \cos\alpha\cos\beta + Z_L \sin\beta - \ell} \text{ and}$$

$$z = -\lambda\frac{Y_w \cos\alpha\sin\beta - Z_L \cos\beta}{Y_W \cos\alpha\cos\beta + Z_L \sin\beta - \ell} \quad (4.15)$$

and applying the Hough Transform gives:

$$\theta = \tan^{-1}\left(\frac{-Y_w \sin\alpha}{Y_w \cos\alpha\sin\beta - Z_L \cos\beta}\right) \text{ and}$$

$$\rho = \frac{-\lambda X_u}{Y_W \cos\beta + Z_L \sin\beta - \ell}\cos\theta - \frac{\lambda(Y_W \sin\beta - Z_L \cos\beta)}{Y_W \cos\beta + Z_L \sin\beta - \ell}\sin\theta \quad (4.16)$$

When the yaw angle (α) is varied, we see from (4.16) that ρ and θ vary, although it should be noted that θ changes very little. Full analysis is presented in Appendix A.

Summarising, equations (4.12), (4.14) and (4.16) show that both ρ and θ of the centre line change with lateral displacement and with yaw, although it should be noted that θ changes very little with yaw; an example showing the effect of lateral displacement and yaw of a camera is given in [49]. However, roll of the UAV only changes that value of θ of the centre line. Thus an ambiguity occurs, because any pair of values of γ and α can be chosen independently to alias a given lateral displacement, X_u. In other words, it is possible for a finite value of X_u to be produced by a combination of yaw and a roll of the UAV, despite the real value being zero. It is concluded that X_u, γ and α cannot all be estimated from just two variables, ρ and θ, obtained from a single camera.

4.2.1.4 Analysis for Height

If we apply equation (4.1) to a model of the centre conductor and vary the height of the UAV from its "normal" height above the lines, Z_L, by a distance Z_u then we get the following:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -Z_u \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ Y_W \\ -Z_L \\ 1 \end{bmatrix} \quad (4.17)$$

This becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell \\ Y_W \sin\beta - (Z_L + Z_u)\cos\beta \\ -\frac{Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell}{\lambda} \end{bmatrix} \quad (4.18)$$

From (4.18), using (4.2), the image co-ordinates are given by:

$$x = 0 \text{ and } z = -\lambda \frac{Y_W \sin\beta - (Z_L + Z_u)\cos\beta}{Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell} \quad (4.19)$$

Applying the Hough Transform then gives:

$$\theta = 0 \text{ and } \rho = 0 \quad (4.20)$$

It can be seen that the height of the UAV above the lines does not affect the position of the centre line in the image.

4.2.1.5 Analysis for Height applied to a Sideline

If the same analysis is done for a sideline at a lateral distance X_S from the centre line, then we get following:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -Z_u \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -X_S \\ Y_W \\ -Z_L \\ 1 \end{bmatrix} \quad (4.21)$$

This becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} -X_S \\ Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell \\ Y_W \sin\beta - (Z_L + Z_u)\cos\beta \\ -\frac{Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell}{\lambda} \end{bmatrix} \quad (4.22)$$

From (4.22), using (4.2), the image co-ordinates are given by:

$$x = \frac{\lambda X_S}{Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell} \text{ and}$$

$$z = -\lambda \frac{Y_W \sin\beta - (Z_L + Z_u)\cos\beta}{Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell} \quad (4.23)$$

Applying the Hough Transform then gives:

$$\theta = \tan^{-1}\left(\frac{\lambda X_S}{Y_W \sin\beta - (Z_L + Z_u)\cos\beta}\right) \text{ and}$$

$$\rho = \frac{\lambda X_S}{Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell}\cos\theta + \lambda \frac{Y_W \sin\beta - (Z_L + Z_u)\cos\beta}{Y_W \cos\beta + (Z_L + Z_u)\sin\beta - \ell}\sin\theta$$

(4.24)

It can be seen that while the centre line does not change position in the image with varying height, the positions of the sidelines do vary in both ρ and θ.

4.2.1.6 Analysis for Pitch

A similar analysis for the pitch gives no change of the centre line position in the image with changing pitch although the positions of the sidelines do vary with pitch. The image co-ordinates are given by:

$$x = \frac{\lambda X_S}{Y_W \cos\beta + Z_L \sin\beta - \ell} \text{ and}$$

$$z = -\lambda \frac{Y_W \sin\beta - Z_L \cos\beta}{Y_W \cos\beta + Z_L \sin\beta - \ell} \quad (4.25)$$

Applying the Hough Transform then gives:

$$\theta = \tan^{-1}\left(\frac{\lambda X_S}{Y_W \sin\beta - Z_L \cos\beta}\right) \text{ and}$$

$$\rho = \frac{\lambda X_S}{Y_W \cos\beta + Z_L \sin\beta - \ell}\cos\theta + \lambda \frac{Y_W \sin\beta - Z_L \cos\beta}{Y_W \cos\beta + Z_L \sin\beta - \ell}\sin\theta \quad (4.26)$$

Full analysis is presented in Appendix A. It can be seen that, while the position of the centre line in the image is unaffected by height or pitch, the positions of the sidelines do vary. Thus it should be possible to estimate the height and pitch of the UAV from the distances in ρ and θ between the sidelines and the centre line. As the two sidelines are at approximately equal distances either side of the centre line, the mean distance between a sideline and the centre line could be used (θ_d, ρ_d). Height affects both ρ_d and θ_d while pitch affects ρ_d, but has very little effect on θ_d. It should, therefore, be possible to determine both the height and pitch of the UAV from one camera. Distance along the lines does not affect the positions of the lines in the image, as would be expected.

4.2.2 Model Validation in MATLAB

In order to assess whether the geometric model is an acceptable representation of the test rig, an image was effectively synthesized by evaluating equation (4.1) for parameter values taken from the test rig, Y_W = 150mm, $\gamma = \alpha = 0$, $\beta = 20°$, ℓ = 30mm and Z_L = 80mm. A small refinement was to use a parabolic model of the lines to approximate their catenary shape. The Hough Transform of the synthesized image was then calculated as each of the UAV's six degrees of freedom were changed one at a time. Figure 4.2 shows that the model predicts ρ_C (dash-dotted) and θ_C (solid) for the centre conductor will vary almost linearly with lateral displacement X_u. Also shown in Figure 4.2 are the mean of the Hough co-ordinates of the two outer conductors relative to the centre conductor, ρ_d (dotted) and θ_d (dashed). These are calculated using:

$$\theta_d = \frac{(\theta_L - \theta_C) + (\theta_C - \theta_R)}{2}$$

$$\rho_d = \frac{(\rho_L - \rho_C) + (\rho_C - \rho_R)}{2} \quad (4.27)$$

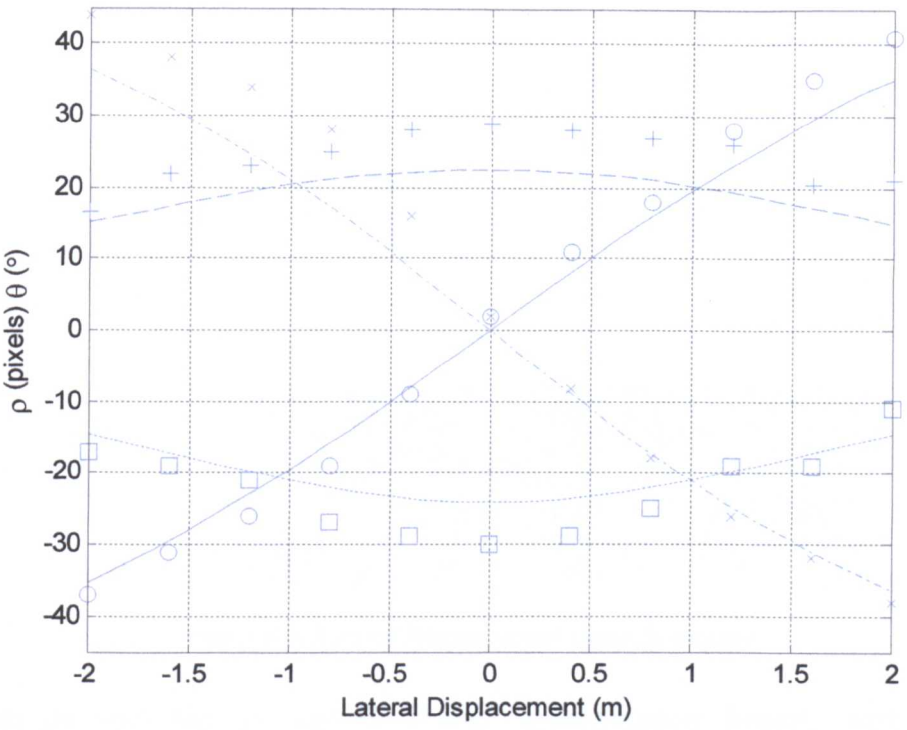


Figure 4.2: Comparison of Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) for Lateral Displacement of the Vehicle from the Centre Line.

To see how the model compares with results from the test rig, a sequence of images, a sub-sampled version of which is shown in Figure 4.3, was produced with a lateral translation of the camera on the test rig. These were then processed using the Hough Transform, described in Chapter 6, to obtain the ρ and θ values of the three lines. The data points measured directly on the test rig were plotted onto Figure 4.2 (ρ_C, x; θ_C, o; ρ_d, □; θ_d, +). There is good agreement between the data points measured directly on the test rig and the model prediction. The discrepancies between the model and the measurements in ρ_C (for negative X_u) and in θ_C (for positive X_u) are thought to be due to a slight offset of the lines at zero displacement on the test rig, and slight differences in the scaling of the image. There is also good agreement for the mean of the Hough co-ordinates of the two outer conductors relative to the centre conductor, which is seen to be relatively insensitive to lateral displacement.

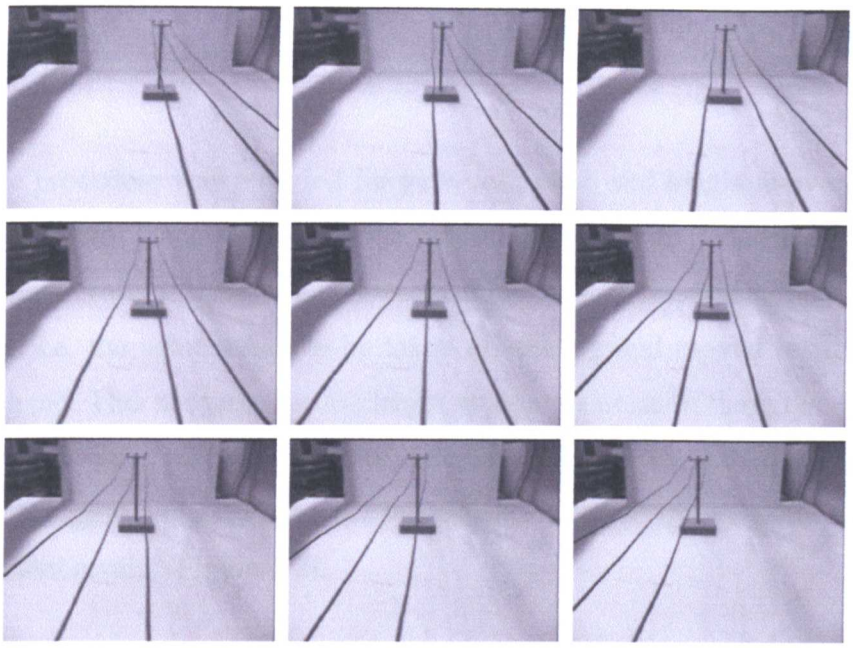


Figure 4.3: Lateral Displacement Image Sequence.

It can be seen that ρ_C and θ_C change approximately linearly with lateral displacement, X. If these are assumed to be linear then the value of the slope will relate the values of X to ρ_C and θ_C. If the inverses of each slope are defined as X_θ and X_ρ (where X_θ and X_ρ are $\frac{\partial X}{\partial \theta}$ and $\frac{\partial X}{\partial \rho}$), then the value of X can be calculated using either:

$$X = X_\theta \theta \quad (4.28)$$

$$X = X_\rho \rho \quad (4.29)$$

Measuring the values X_θ and X_ρ from the test rig data in Figure 4.2 gives:

$X_\theta = 0.048$

$X_\rho = -0.0433$

The lateral displacement could be obtained individually from either ρ or θ but for the testing done in Chapter 7 the average was used:

$$X = \frac{X_\theta \theta + X_\rho \rho}{2} \quad (4.30)$$

The same procedure was repeated for yaw, roll, pitch and height above the lines. Again image sequences were obtained from the test rig. Because the test rig doesn't have roll or height adjustment, these sequences had to be obtained manually, i.e. the camera had to be taken off the rig and moved relative to the lines by hand. This accounts for the larger amount of noise in these two data sets. The results for yaw, roll, height above the lines and pitch are shown in Figure 4.4, Figure 4.5, Figure 4.6 and Figure 4.7 respectively. Line styles are as for the lateral displacement results (Figure 4.2).

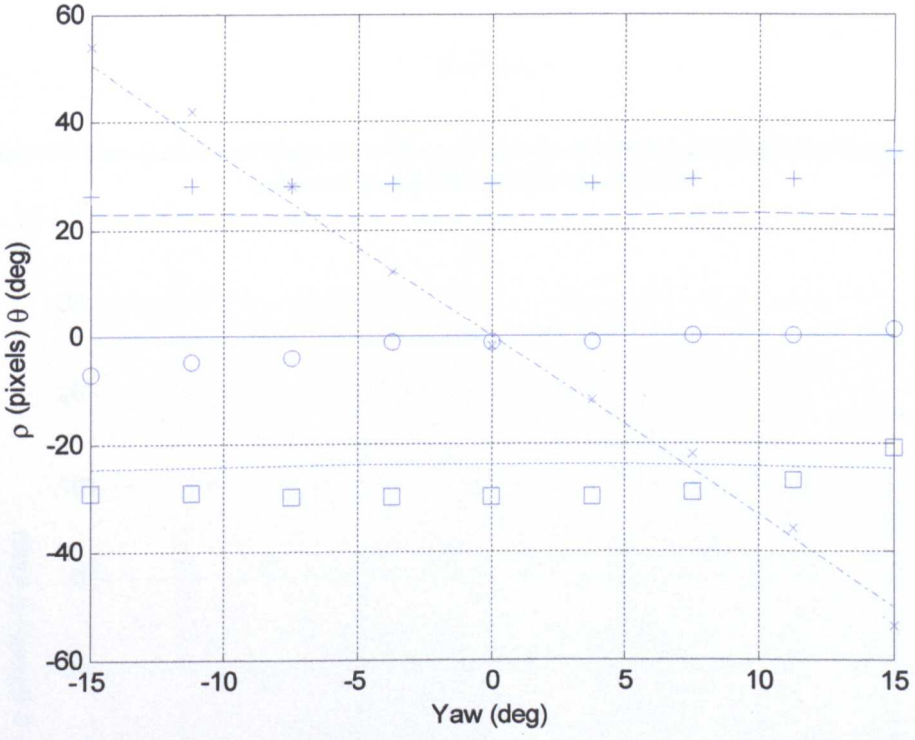


Figure 4.4: Comparison of Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) for Yaw of the Vehicle.

Figure 4.4 shows that ρ_C changes roughly linearly with yaw, as predicted, but there is virtually no change in θ_C. The positions of the outer lines relative to the centre line are insensitive to yaw. There is again good agreement between the test rig and model results.

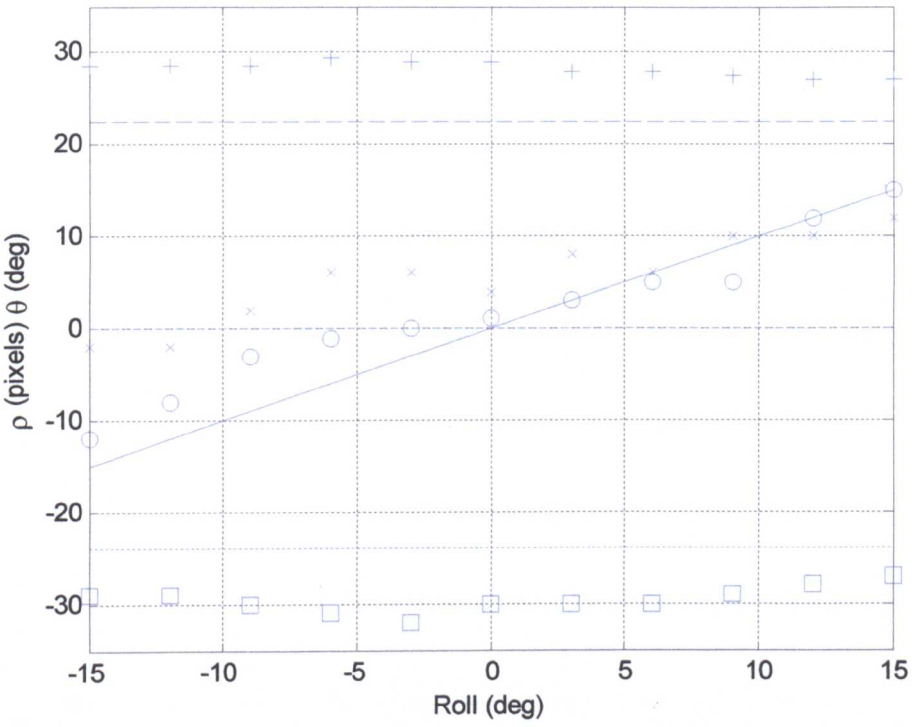


Figure 4.5: Comparison of Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) for Roll of the Vehicle.

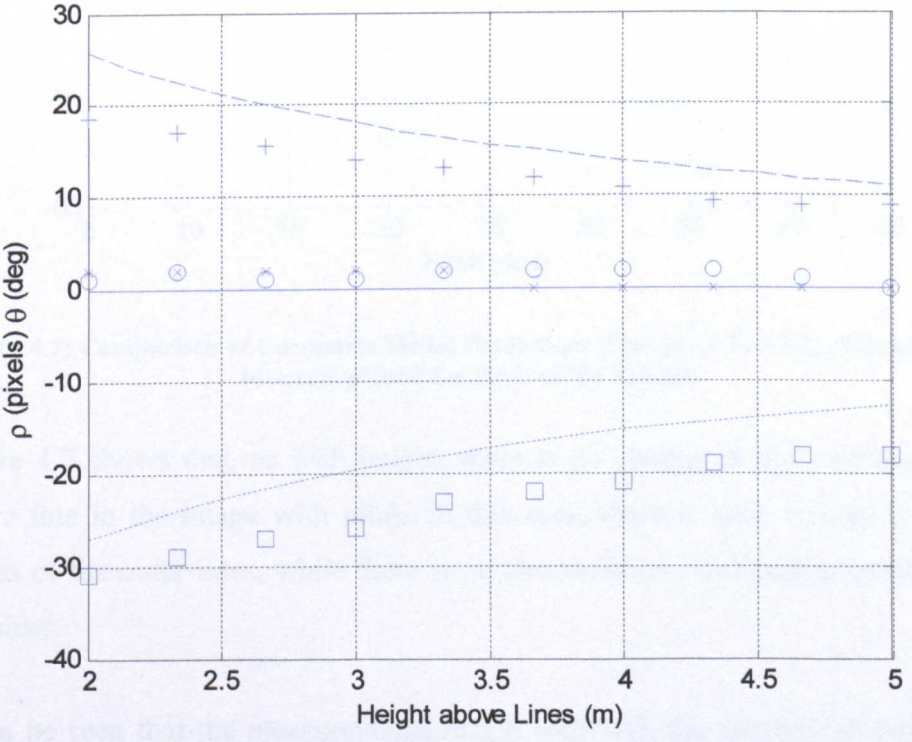


Figure 4.6: Comparison of Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) for Varying Vehicle Height above the Lines.

Figure 4.5 shows that θ_C changes linearly with roll, as predicted, but there is virtually no change in ρ_C. The positions of the outer lines relative to the centre line are insensitive to roll. There is again good agreement between the test rig and model results.

Figure 4.6 shows that there is no change in the position of the centre line(ρ_C, θ_C) with height above the lines. There is a change in the mean of the Hough co-ordinates of the two outer conductors relative to the centre conductor (ρ_d, θ_d).

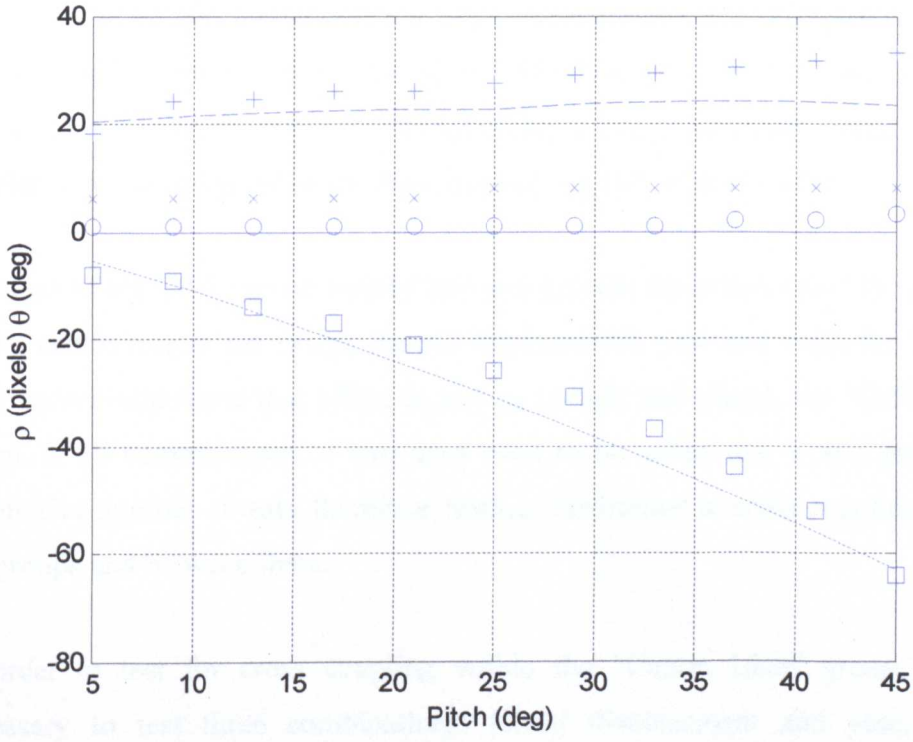


Figure 4.7: Comparison of Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) for Pitch of the Vehicle.

Figure 4.7 shows that, as with height, there is no change in the position of the centre line in the image with pitch. In this case, there is little change in the θ_d values of the outer lines, while there is an almost linear relationship between the ρ_d values.

It can be seen that the measurements match well with the theoretical model. As predicted, θ_d and ρ_d are only significantly affected by pitch and height meaning that it should be possible to estimate the height and pitch from θ_d and ρ_d. The

position of the centre line in the image is affected by the lateral position, yaw and roll. Roll only affects the value of θ_C while yaw only significantly affects the value of ρ_C. Lateral displacement has an effect on both θ_C and ρ_C. As such it is possible for a combination of a yaw and a roll to have an equivalent effect on the image as a lateral translation. It is not, therefore, possible to separate these three values from the two variables, ρ_C and θ_C. The solution to this problem is discussed in section 4.3.

4.2.3 Two-axis Modelling in MATLAB

The previous section considered varying individual axes and observed the effect on the image. However the results do not show the effect of any cross coupling between axes. Here the effect of varying multiple axes is assessed, using the same procedure as was used for single axes, but varying two axes at a time.

The axes of the UAV can be formed into two groups: those that affect the position of the centre line in the image (lateral displacement, yaw and roll), the "Centre Line" group and those that affect θ_d and ρ_d (height and pitch), the "Difference" group. If all combinations of two axes were to be tested this would produce a prohibitive number of tests, therefore, testing was limited to cross coupling within the groups and between them.

In order to test for cross coupling within the "Centre Line" group, it was necessary to test three combinations: lateral displacement and yaw, lateral displacement and roll and roll and yaw.

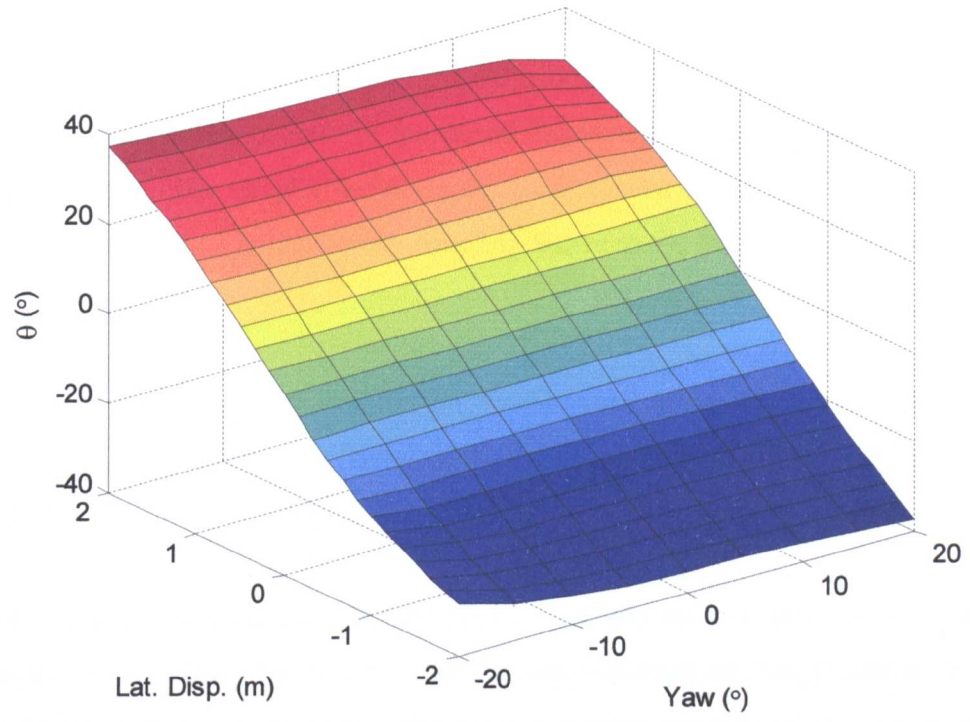


Figure 4.8: The Effect of Varying both Yaw and Lateral Displacement on θ_C.

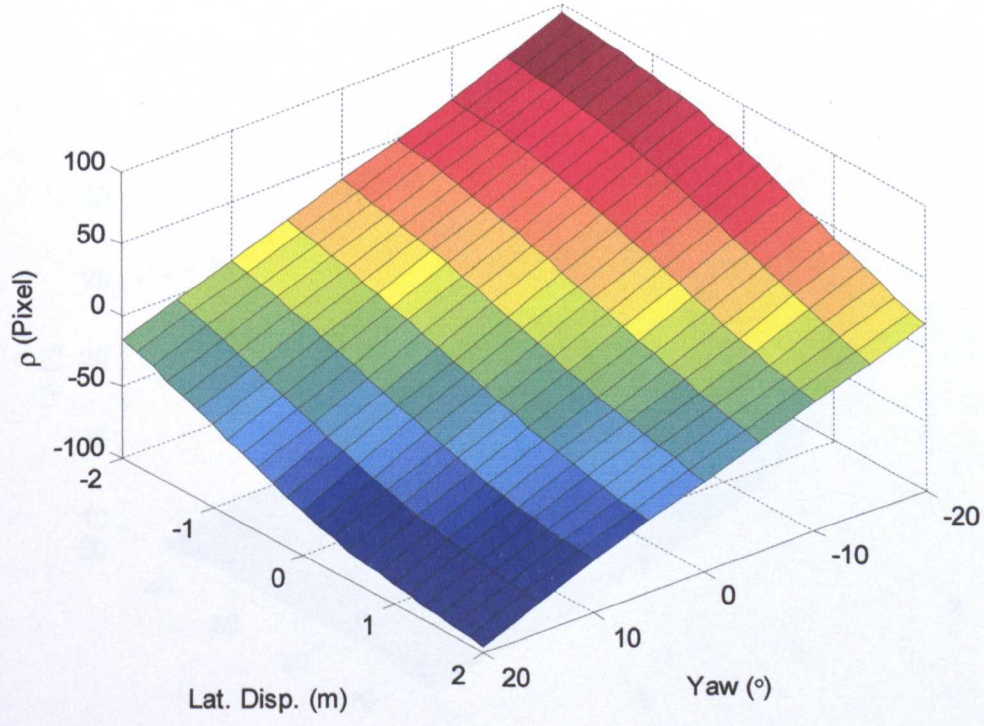


Figure 4.9: The Effect of Varying both Yaw and Lateral Displacement on ρ_C.

It can be seen from Figure 4.8 and Figure 4.9 that there is little cross coupling between the lateral displacement and yaw axes; the θ_C and ρ_C values are approximately equal to the sum of the contributions to θ_C and ρ_C from each axis. This procedure was repeated for the lateral displacement and roll axes and the roll and yaw axes. The graphs for these tests are shown in Appendix B. It can be concluded that there is little cross coupling between the lateral displacement, yaw and roll axes.

There is only one test required for the "Difference" group: pitch and height. This test looks for cross coupling between θ_d and ρ_d, as neither axis affects the values of θ_C and ρ_C.

It can be seen from Figure 4.10 and Figure 4.11 that there is more cross coupling than there was within the "Centre Line" group, although the height and pitch axes should be reasonably separable. The cross coupling should be less of a problem with pitch and height, as the aim would be to maintain the pitch and height within tight limits.

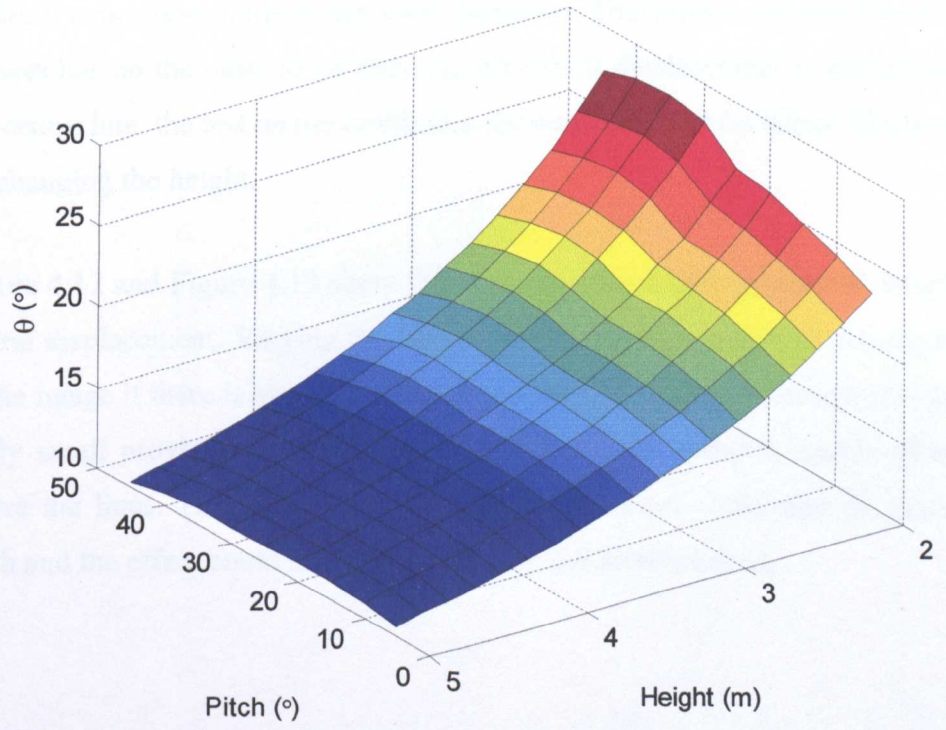


Figure 4.10: The Effect of Varying both Pitch and Height on θ_d.

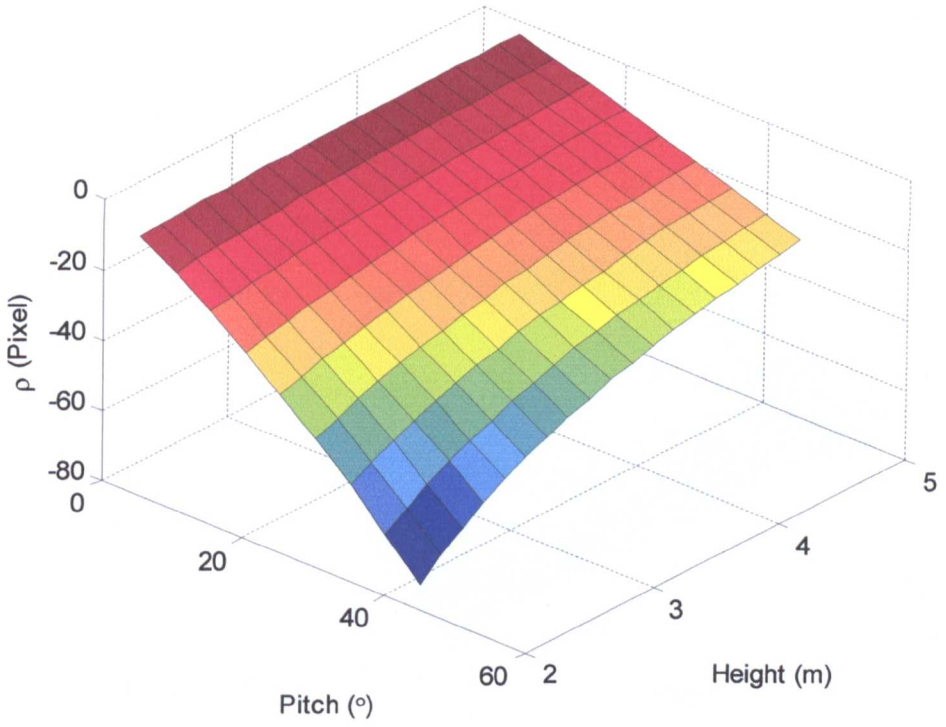


Figure 4.11: The Effect of Varying both Pitch and Height on ρ_d.

In order to test for cross coupling between the two groups of axes, the lateral displacement and height were varied, and the effect on both θ_d and ρ_d and the centre line position in the image were measured. This allows the effect each group of axes has on the other to be seen. As the lateral displacement is measured from the centre line, the test on the centre line showed the effect on lateral displacement by changing the height.

Figure 4.12 and Figure 4.13 show that there is cross coupling between height and lateral displacement. Varying the height will affect the position of the centre line in the image if there is also lateral displacement of the UAV, although the effect is fairly small provided the UAV height is maintained within a couple of metres above the lines. The effect on the lateral displacement could also be caused by pitch and the effect could also affect yaw and roll measurement.

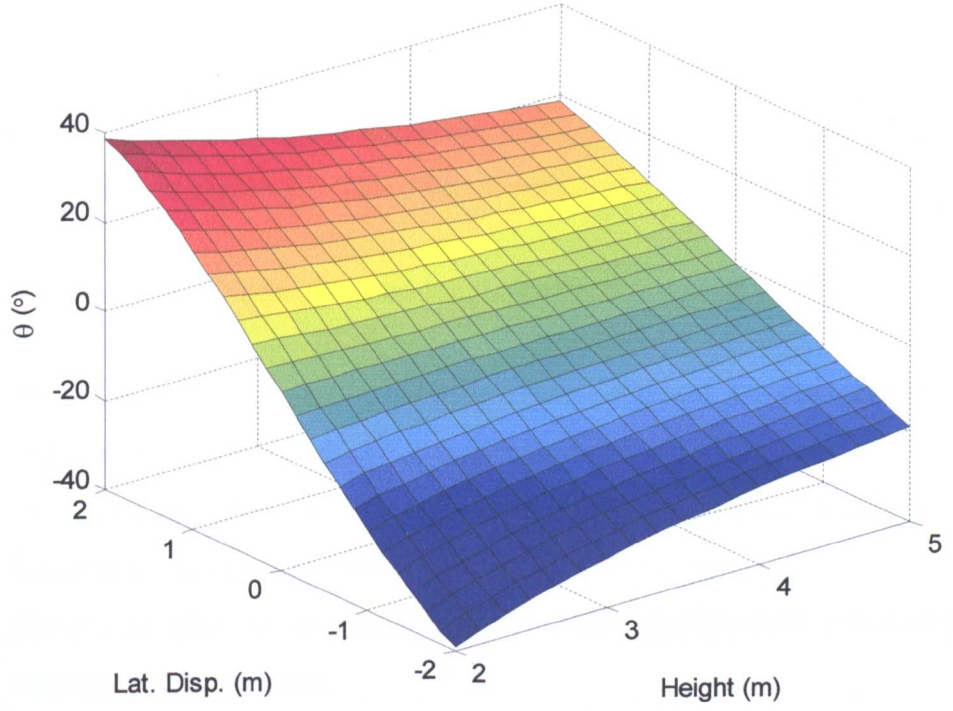


Figure 4.12: The Effect of Varying both Height and Lateral Displacement on θ_C.

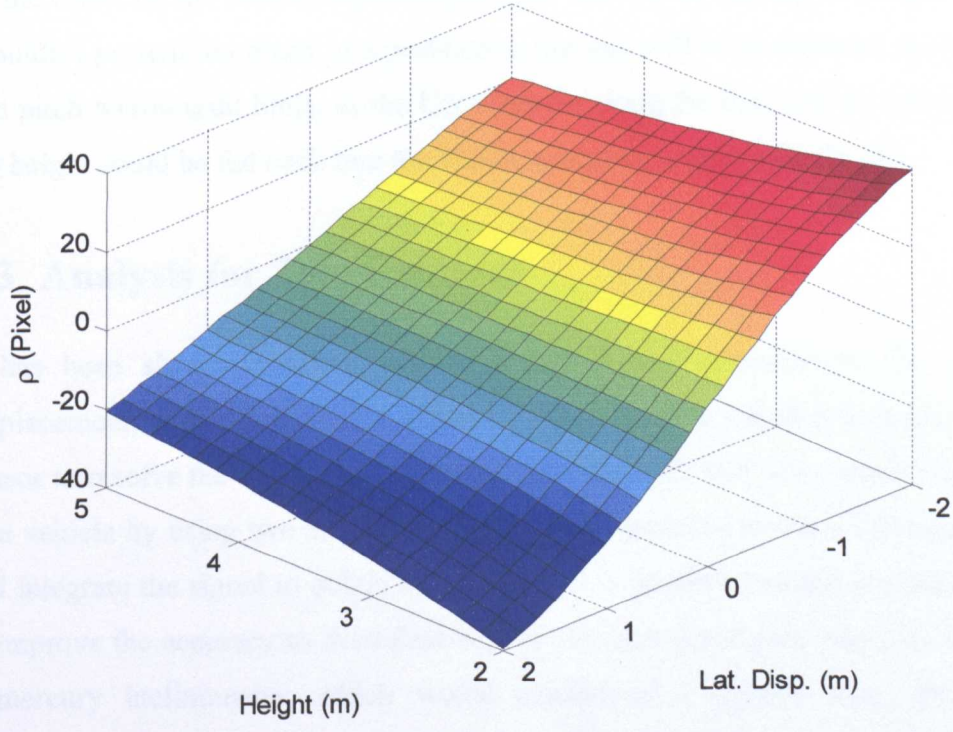


Figure 4.13: The Effect of Varying both Height and Lateral Displacement on ρ_C.

As the height is measured from θ_d and ρ_d, the test on θ_d and ρ_d showed the effect on height by changing the lateral displacement. The graphs showing this are shown in Appendix B. This shows that there is some effect on the height measurement by changing the lateral displacement. This could be predicted by the θ_d and ρ_d lines in Figure 4.2. This effect is not caused by yaw or roll.

To summarise, the following cross-coupling effects are seen:

- There is little cross coupling between lateral displacement, yaw and roll.
- There is little cross coupling between height and pitch.
- There is some effect on the measurement of height or pitch by changing the lateral displacement but not by changing the roll or yaw.
- There is an effect on the measurement of lateral displacement, yaw and roll by varying height and pitch.

It should be possible to extract the information about all five axes from the image. There is little cross coupling between most of the axes. The main concern would be the effect on the lateral displacement, yaw and roll of varying the height. This shouldn't present too much of a problem as the aim will be to maintain the height and pitch within tight limits as the UAV travels along the line, and an estimate of the height could be fed back into the vision system to compensate for this.

4.3 Analysis for Two Cameras

It has been shown that one camera is insufficient to determine the lateral displacement, yaw and roll of the UAV. One strategy is to use an independent roll sensor to resolve the ambiguity. There are forms of GPS that can estimate the roll of a vehicle by using two antennas or it would be possible to use a roll rate gyro and integrate the signal to obtain the position. It is possible to combine these two to improve the accuracy as described in [50]. Another possibility would be to use a mercury inclinometer, which would consist of a circular tube, mounted vertically aligned left to right, with a number of contacts spread around its length, inside the tube, and small amount of mercury, inside the tube; roll could be measured by which contacts were connected by the mercury. Mercury

inclinometers are sensitive to all acceleration rather than just gravity. All of these sensors do have a problem with noise and limited accuracy. The known effect of the roll could then be subtracted from the θ_C value from the image and the yaw and lateral displacement can be extracted from the resulting ρ_C and θ_C values. Another possibility to separate lateral displacement, yaw and roll is to use a second camera, pointing backward, and pitched downward relative to the duct, as shown in Figure 4.14. This should allow the separation of lateral displacement, yaw and roll as they will affect the position of the centre line in the image from the backward camera differently to its position in the image from the forward camera.

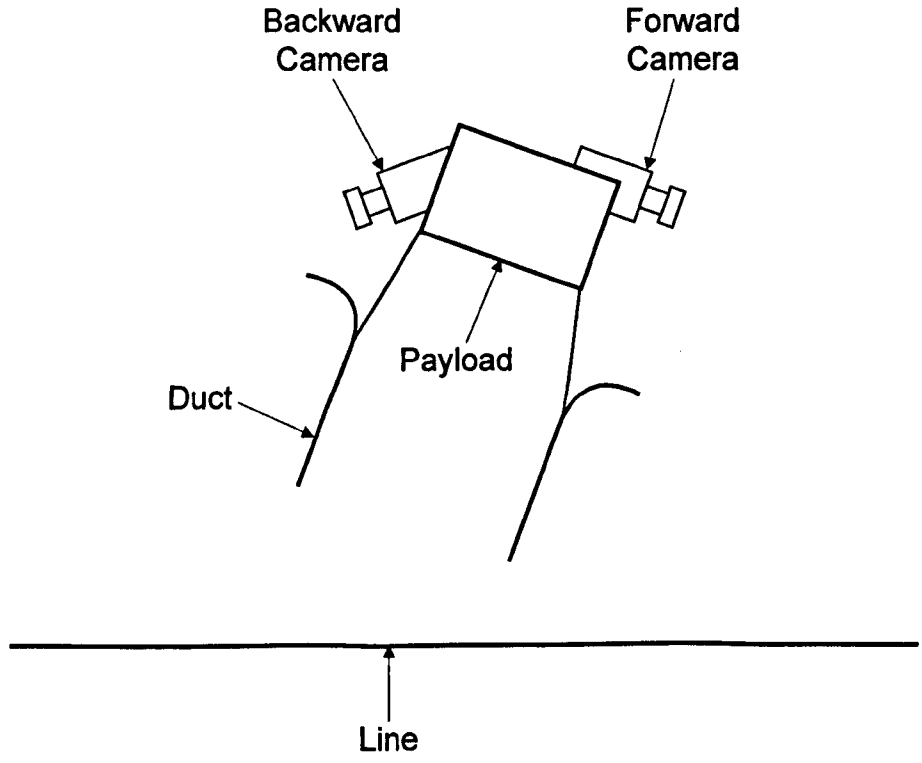


Figure 4.14: Mounting of Twin Cameras on the Duct.

The reference frames for two cameras are shown in Figure 4.15. The geometric model for the forward camera is given by (4.1) and the analysis proceeds in similar fashion to Section 4.2, except that the value of ℓ is larger. This is due to the cameras being back to back, requiring them to be placed further from the vertical axis of rotation. The geometric model for the rear-pointing camera is similar to (4.1) and is shown in (4.31). The differences are: the camera is pitched down from the rotorcraft by an angle β_C, giving an extra transformation; the yaw

angle is incremented by π due to the camera pointing rearward and the roll and pitch are in the opposite direction.

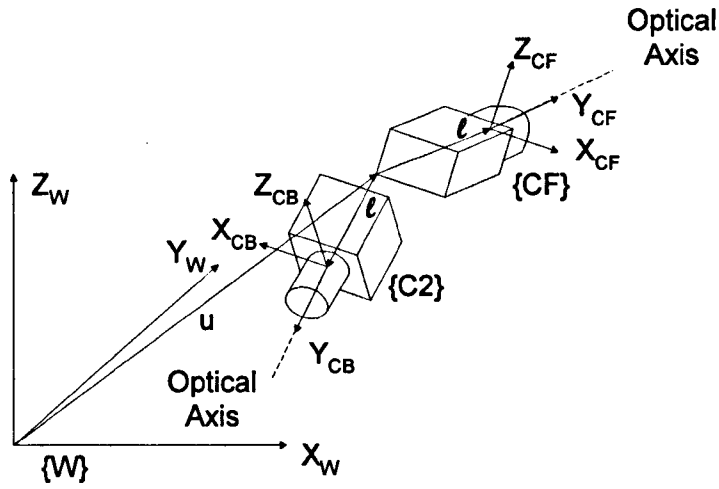


Figure 4.15: Reference Frame Definitions for twin Cameras.

$$\begin{bmatrix} X_{CB} \\ Y_{CB} \\ Z_{CB} \\ \frac{-Y_{CB}}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta_C & -\sin\beta_C & 0 \\ 0 & \sin\beta_C & \cos\beta_C & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} \cos(-\gamma) & 0 & \sin(-\gamma) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\gamma) & 0 & \cos(-\gamma) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} \cos(\alpha+\pi) & -\sin(\alpha+\pi) & 0 & 0 \\ \sin(\alpha+\pi) & \cos(\alpha+\pi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -X_u \\ 0 & 1 & 0 & -Y_u \\ 0 & 0 & 1 & -Z_u \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \quad (4.31)$$

Again, the Hough Transforms for the dual camera case were calculated for a synthetic image. The simulation was repeated, as for the case of one camera, for lateral displacement, yaw and roll. The values of ρ and θ for both cameras were plotted against lateral displacement, yaw and roll.

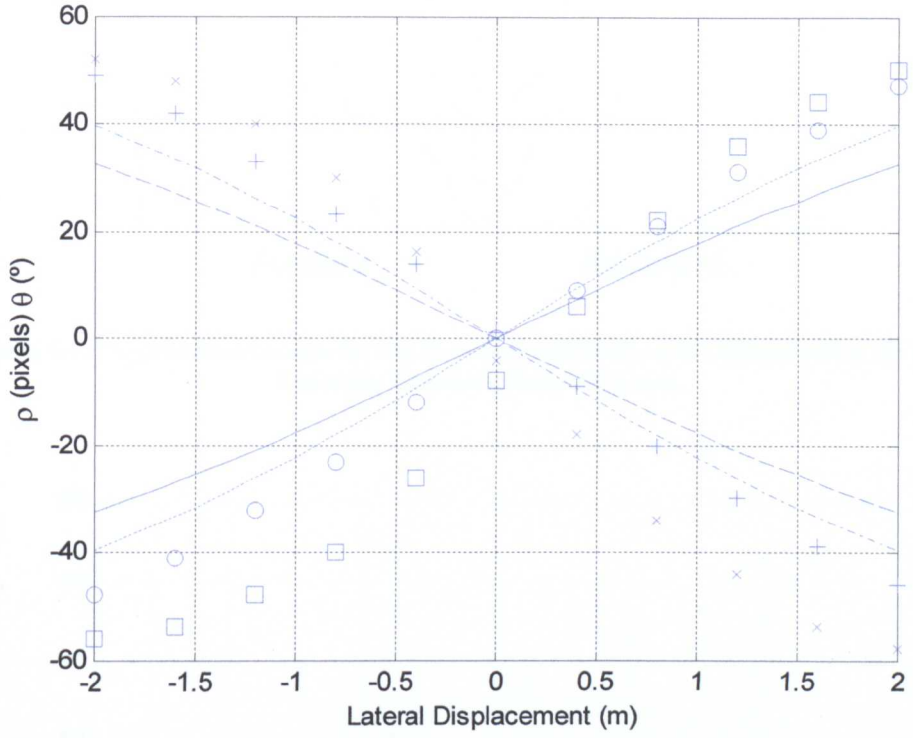


Figure 4.16: Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) of ρ_C and θ_C for Varying Lateral Displacement (X_u) with Twin Cameras.

Figure 4.16 shows that each camera gives opposite results: as lateral displacement increases, ρ_{CF} (dash-dotted) and θ_{CB} (dashed) decrease while ρ_{CB} (dotted) and θ_{CF} (solid) increase. To see how the model compares with results from the test rig, a sequence of images was produced, in the same way as for one camera, with a lateral translation of the cameras on the test rig. These were then processed using the Aggregated Hough Transform, described in Chapter 6, to obtain the ρ and θ values of the three lines. The data points measured directly on the test rig were plotted onto Figure 4.16 (ρ_{CF}, x; θ_{CF}, o; ρ_{CB}, □; θ_{CB}, +). This effect would be expected because when the UAV moves to the left, the forward-pointing camera moves to its left but the rearward-pointing camera moves to its right. Example synthesised images for the two cameras, where the UAV is to the left of the lines, are shown in Figure 4.17.

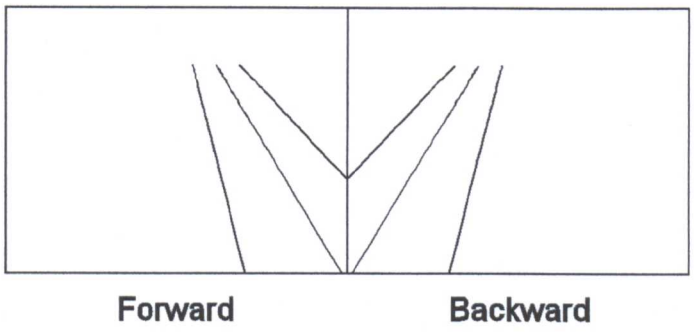


Figure 4.17: Synthesised Images for the Forward and Backward Camera when the UAV is Laterally Displaced from the Lines.

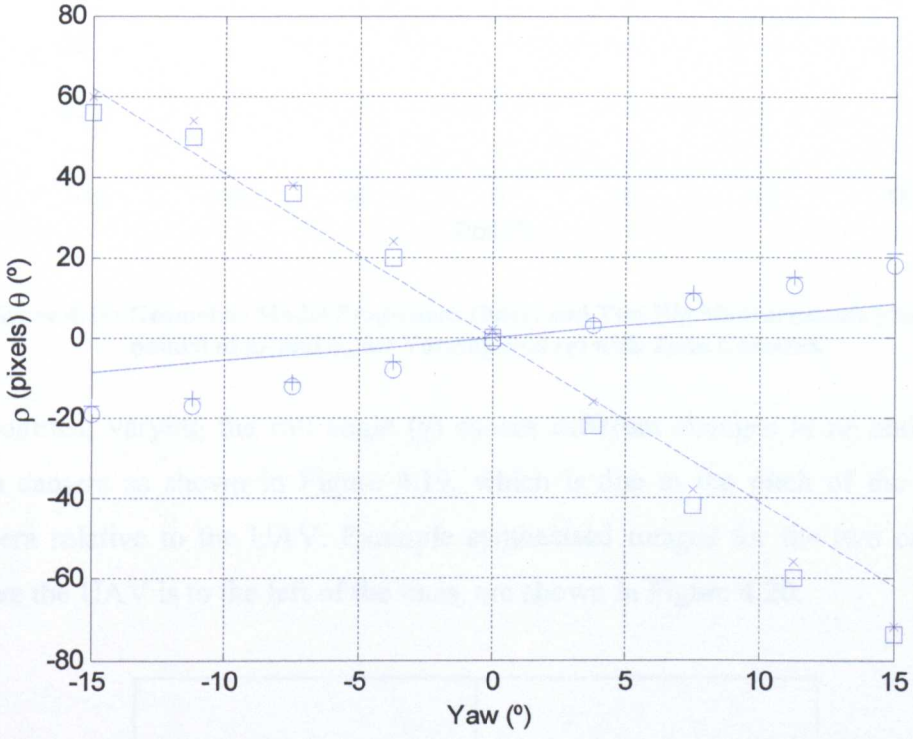


Figure 4.18: Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) of ρ_C and θ_C for Varying Yaw (α) with Twin Cameras.

Figure 4.18 shows the result for varying yaw angle (α), where it is clear that changes in ρ_C and θ_C are the same for both cameras; the backward camera results (dotted, dashed) overlap the camera 1 results in the figure. This is as expected because the rotation occurs in the same sense for both cameras. It can also be seen that there is a small change in the value of θ with yaw, compared to the single camera, where there was very little change; this is due to the longer value of ℓ in the two-camera configuration.

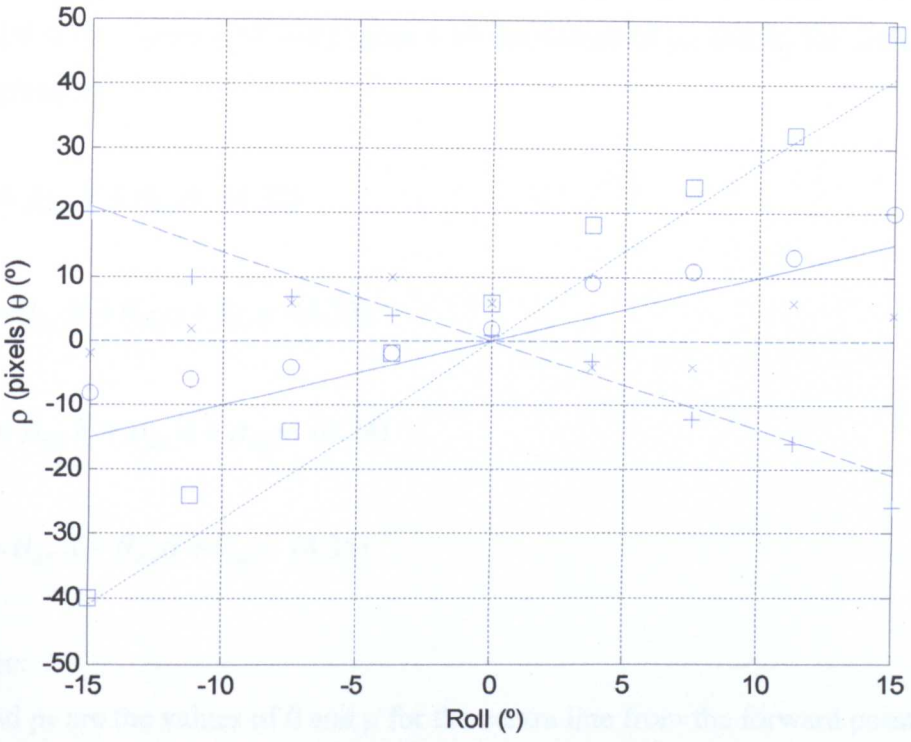


Figure 4.19: Geometric Model Predictions (lines) and Test Rig Measurements (discrete points) of ρ_C and θ_C for Varying Roll (γ) with Twin Cameras.

In contrast, varying the roll angle (γ) causes different changes in ρ_C and θ_C for each camera as shown in Figure 4.19, which is due to the pitch of the second camera relative to the UAV. Example synthesised images for the two cameras, where the UAV is to the left of the lines, are shown in Figure 4.20.

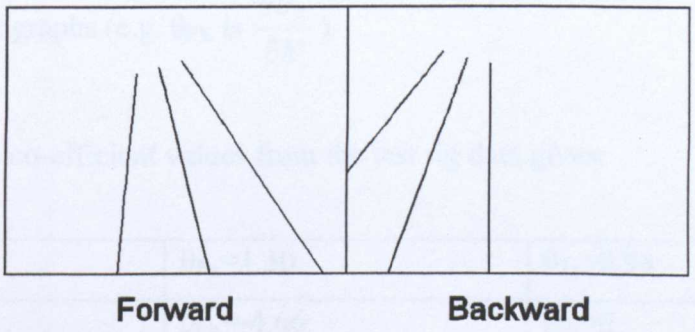


Figure 4.20: Synthesised Images for the Forward and Backward Camera when the UAV Rolls.

Because the two cameras yield different results for varying lateral displacement, yaw and roll, there are now four equations and three unknowns. Estimates of the position and orientation of the UAV relative to the lines can therefore be made.

By defining co-efficients that are the values of the slopes of each of the lines in Figure 4.16, Figure 4.18 and Figure 4.19 the values of ρ_C and θ_C for each camera are given by:

$$\rho_F = \rho_{FX}X + \rho_{F\alpha}\alpha \quad (4.32)$$

$$\theta_F = \theta_{FX}X + \theta_{F\alpha}\alpha + \theta_{F\gamma}\gamma \quad (4.33)$$

$$\rho_B = \rho_{BX}X + \rho_{B\alpha}\alpha + \rho_{B\gamma}\gamma \quad (4.34)$$

$$\theta_B = \theta_{BX}X + \theta_{B\alpha}\alpha + \theta_{B\gamma}\gamma \quad (4.35)$$

where:

θ_F and ρ_F are the values of θ and ρ for the centre line from the forward pointing camera.

θ_B and ρ_B are the values of θ and ρ for the centre line from the backward pointing camera.

X is the lateral displacement of the UAV

α is the yaw of the UAV

γ is the roll of the UAV

θ_{FX}, ρ_{FX}, $\theta_{F\alpha}$, $\rho_{F\alpha}$ etc. are the coefficients: these are equal to the values of the slope of the relevant graphs (e.g. θ_{FX} is $\frac{\partial \theta_F}{\partial X}$)

Measuring the co-efficient values from the test rig data gives:

θ_{FX}=24.9	$\theta_{F\alpha}$=1.30	$\theta_{F\gamma}$=0.93
ρ_{FX}=-30.7	$\rho_{F\alpha}$=-4.66	$\rho_{F\gamma}$=0
θ_{BX}=-24.9	$\theta_{B\alpha}$=1.30	$\theta_{B\gamma}$=-1.36
ρ_{BX}=30.7	$\rho_{B\alpha}$=-4.66	$\rho_{B\gamma}$=2.74

Table 4.1: Lateral Displacement, Yaw Roll Co-efficients.

Height and pitch information can be obtained from ρ_d and θ_d from the forward camera.

4.4 Conclusions

It has been shown in this chapter that it is theoretically possible to estimate the pose and position of the UAV relative to the lines. Lateral displacement, yaw and roll affect the position of the centre line, while pitch and height above the lines affect the distance between the outer and centre line in Hough space. It is possible to obtain estimates of the pitch and height from one camera, but in order to obtain the lateral displacement, yaw and roll it is necessary to combine the results from two cameras or one camera and a roll sensor. Measurements from the test rig agreed well with the predictions of the geometric model.

Chapter 5 Test Rig

5.1 Introduction and Overview

In order to perform real-time experimental testing of the image processing, tracking, controller and UAV model, it was necessary to construct a test rig that simulates the UAV. While the UAV itself has six degrees of freedom, it was decided, for simplicity, to limit the rig to the three degrees of freedom: X, Y and Yaw. These axes were chosen because it is necessary to control the horizontal position and heading of the UAV relative to the lines. The pitch and height relative to the lines should remain within tight limits when the UAV flies along a line, and so can be assumed to be constant for the purposes of this project. The roll axis would be useful for simulation, but would add significantly to the complexity of the rig; a roll axis would be the choice if a fourth axis were to be added.

The rig used for this project is a large custom-built X-Y table, which carries a camera. The mechanical subsystem was originally designed by Matthew Williams [17] but has been extensively modified for use with this project. The test rig measures approximately 2.5x1.1m and is shown in Figure 5.1. In addition to the table, the rig comprises a custom-made controller board to control the position of the camera, and three PCs to run the image processing, tracking, controller and UAV model.

The rig has position control of the X and Y axes and the Yaw of the camera, provided by a custom written software controller. Underneath the table is a 30:1 scale model power line. This gives an idealised version of the scene that would be seen by the UAV. The camera can be steered to simulate the yaw of the rotorcraft, and has a fixed forward pitch. As the UAV travels forwards along the line at a constant speed it will maintain a constant pitch in order to achieve this. Although the pitch angle of the rig's camera can't be changed actively during simulation it is possible to change the pitch angle manually. The X and Y axes are both driven through timing belts driven by motors. The position of the camera in the X direction is sensed by an optical encoder driven by the belt, while the Y position is

sensed by an optical encoder on the motor shaft. The Yaw axis is directly driven by a motor with an optical encoder attached.

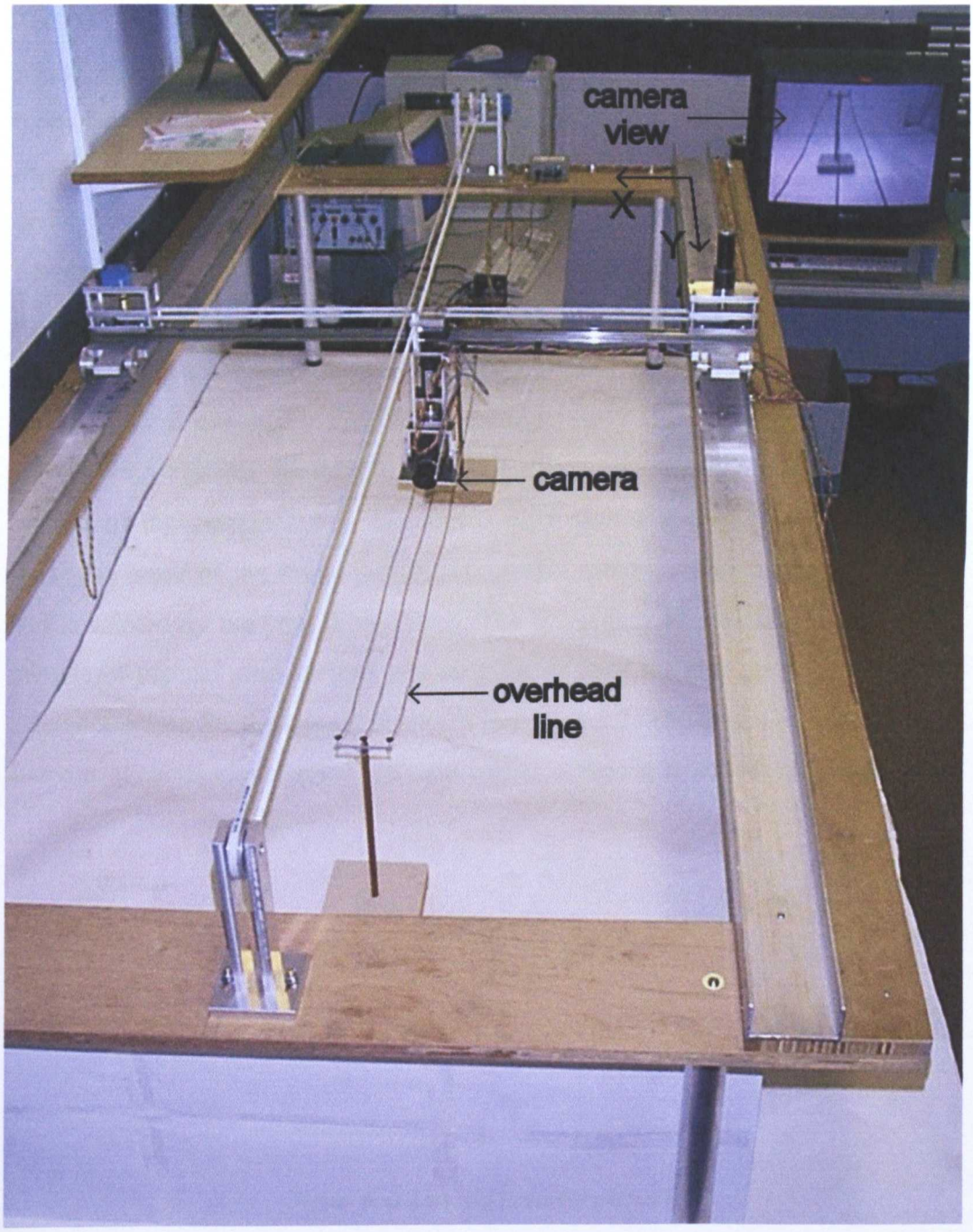


Figure 5.1: Test Rig.

The test rig retains the original table's mechanical components, but has new X and Y drives. The camera mount and Yaw axis drive are also new. The whole rig has been raised on legs, in order to place the camera above the lines. The interface

between the drives and the controller is identical for each axis, which simplifies the writing of the control software.

The X-Y table consists of two parallel aluminium channels mounted on a wooden frame. A carriage runs on each channel with a THK linear V rail fixed between them. Mounted on each carriage is a pulley housing. Between the two pulleys there is a toothed belt, which is attached to the camera mount and provides the X drive for the table. The drive motor itself is mounted on one of the pulley housings. The position of the belt is sensed by a rotary optical encoder, which is mounted on the other pulley housing. A second toothed belt runs in the Y direction between two pulleys mounted at either end of the wooden frame. The linear V rail is attached to this belt to provide the Y drive. Both the drive motor and position encoder are attached to the pulley at one end of the rig. The upper section of the camera mount contains a drive motor and an optical encoder to sense the angle of the lower part of the camera mount, which carries the camera and is rotated by the yaw drive motor. The lower part of the camera mount also includes a manual pitch setting, allowing the pitch of the camera to be fixed. The reference frame for the test rig is shown in Figure 5.2. The origin of this reference frame is at camera level at the top right corner of the rig as shown in Figure 5.1.

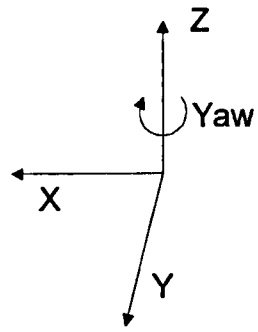


Figure 5.2: Test Rig Reference Frame.

The position and orientation of the camera is controlled by a dedicated controller. This circuit board carries a PIC microcontroller that runs the position control software. This interfaces to the rig motors through DACs and power amplifiers, and the encoders via dedicated counter chips. The microcontroller receives position demands from the control PC via a serial connection. This PC runs a model of the UAV and the visual servoing.

The remaining sections describe in more detail the mechanical design of the rig modifications, the modelling of the rig and the electronic and software design of the controller.

5.2 Mechanical Design

The author designed all the modifications to the test rig. The School's Mechanical Workshop carried out the majority of the construction

5.2.1 X Drive

The X drive required the simplest modification, involving the replacement of the drive belt and the fitting of a rotary encoder to the pulley housing at the non-drive end of the linear V rail. The existing drive motor and pulley wheels were used. Figure 5.3 shows the design of the encoder end of the X drive. Figure 5.4 shows the X drive system.

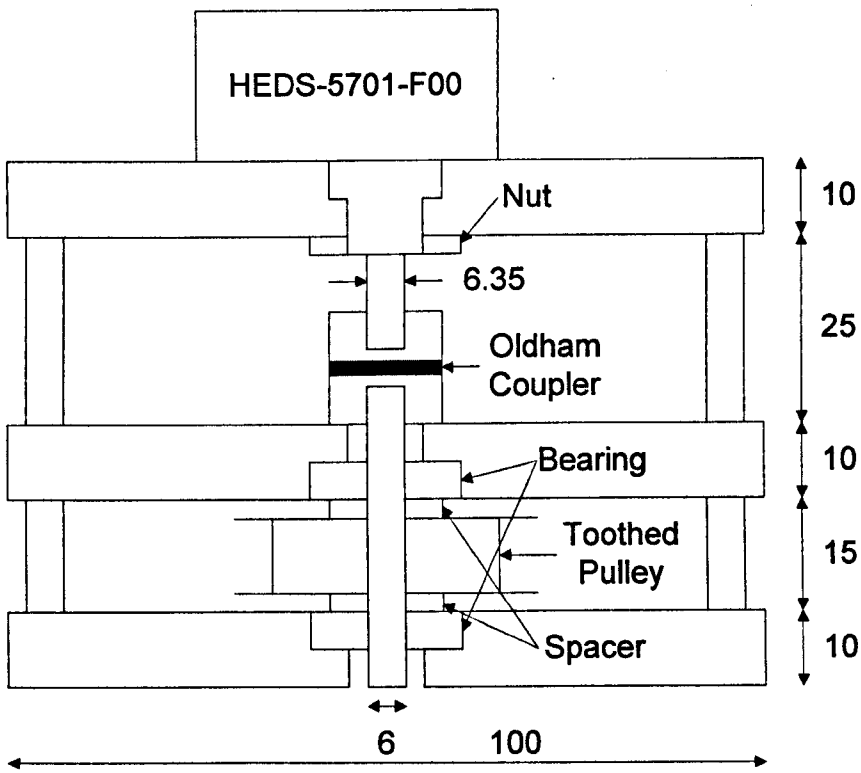


Figure 5.3: Design of the X Drive Encoder Pulley Housing.

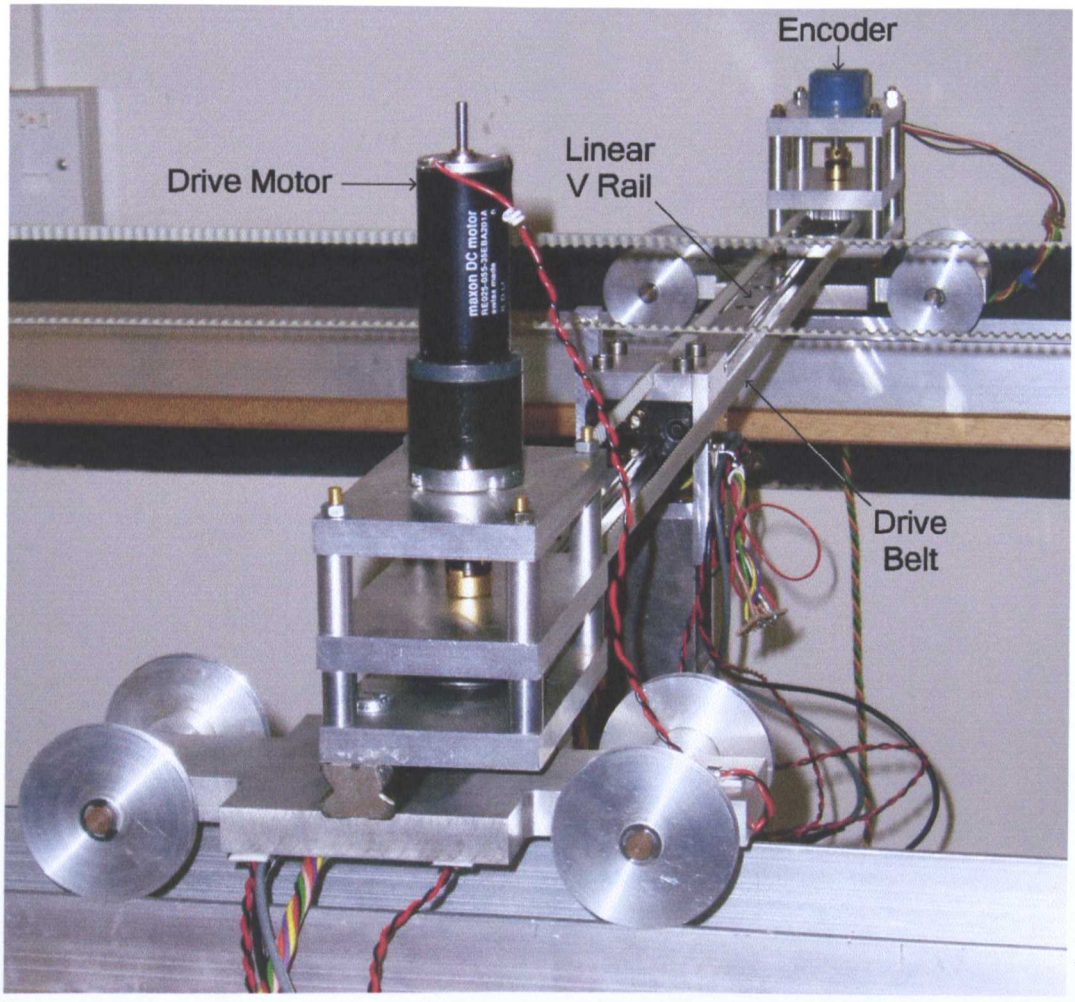


Figure 5.4: X Drive.

5.2.2 Y Drive

The Y drive needed to be replaced completely, with the exception of re-using the drive motor. It was decided that the best option was to use a toothed belt design as used in the X drive. This required a pulley mount at one end, and a pulley mount with the drive motor and position optical encoder at the other. A method of linking the belt to the linear V rail was required, which needed to avoid the X drive belt and the camera mount.

Figure 5.5 shows the Y drive pulley housing design while the design for the non-drive pulley housing and linear V rail link are shown in Figure 5.6. Figure 5.7 is a photograph of the drive end of the Y mechanism.

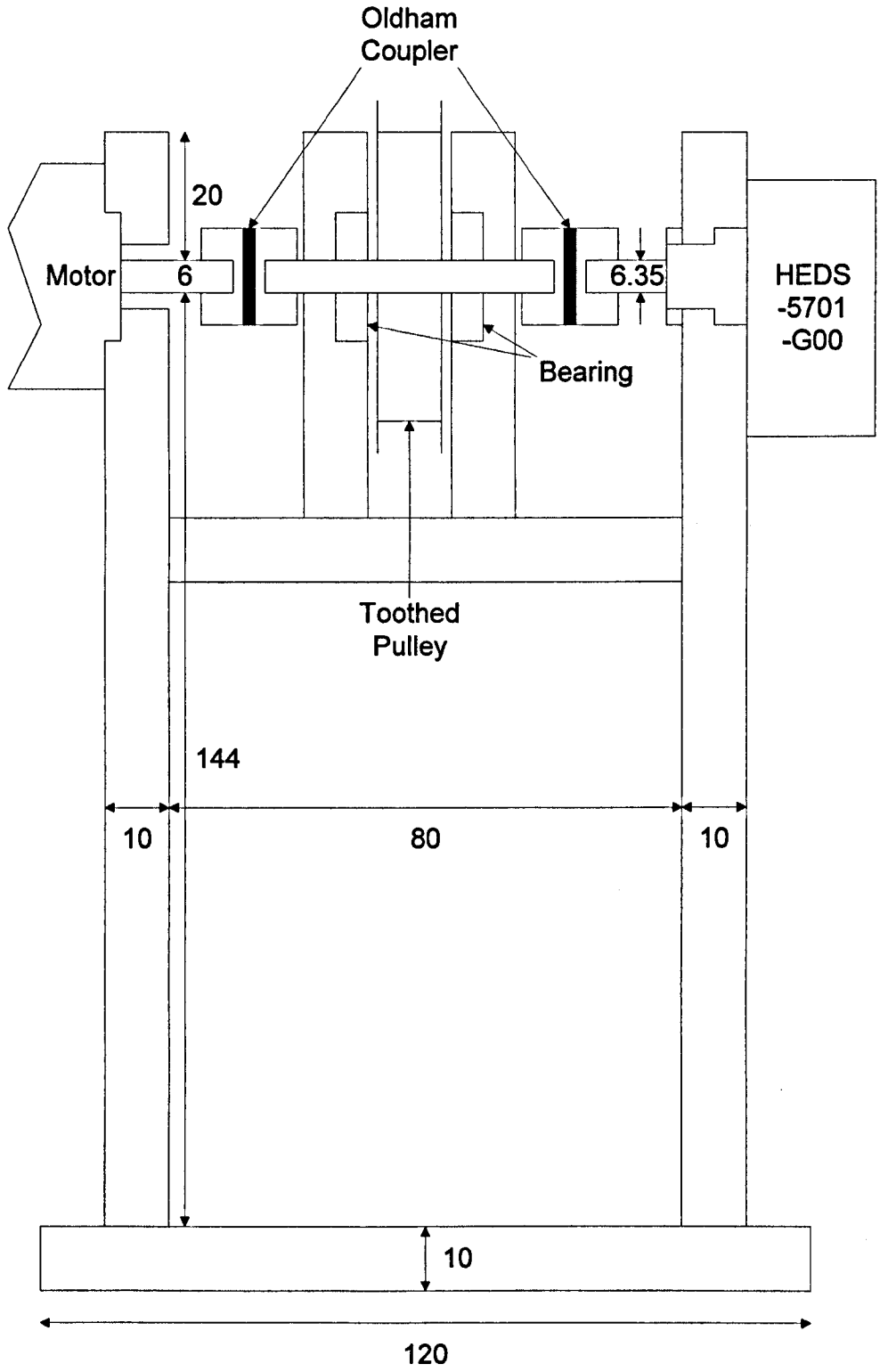


Figure 5.5: Drive Pulley Housing for the Y Drive.

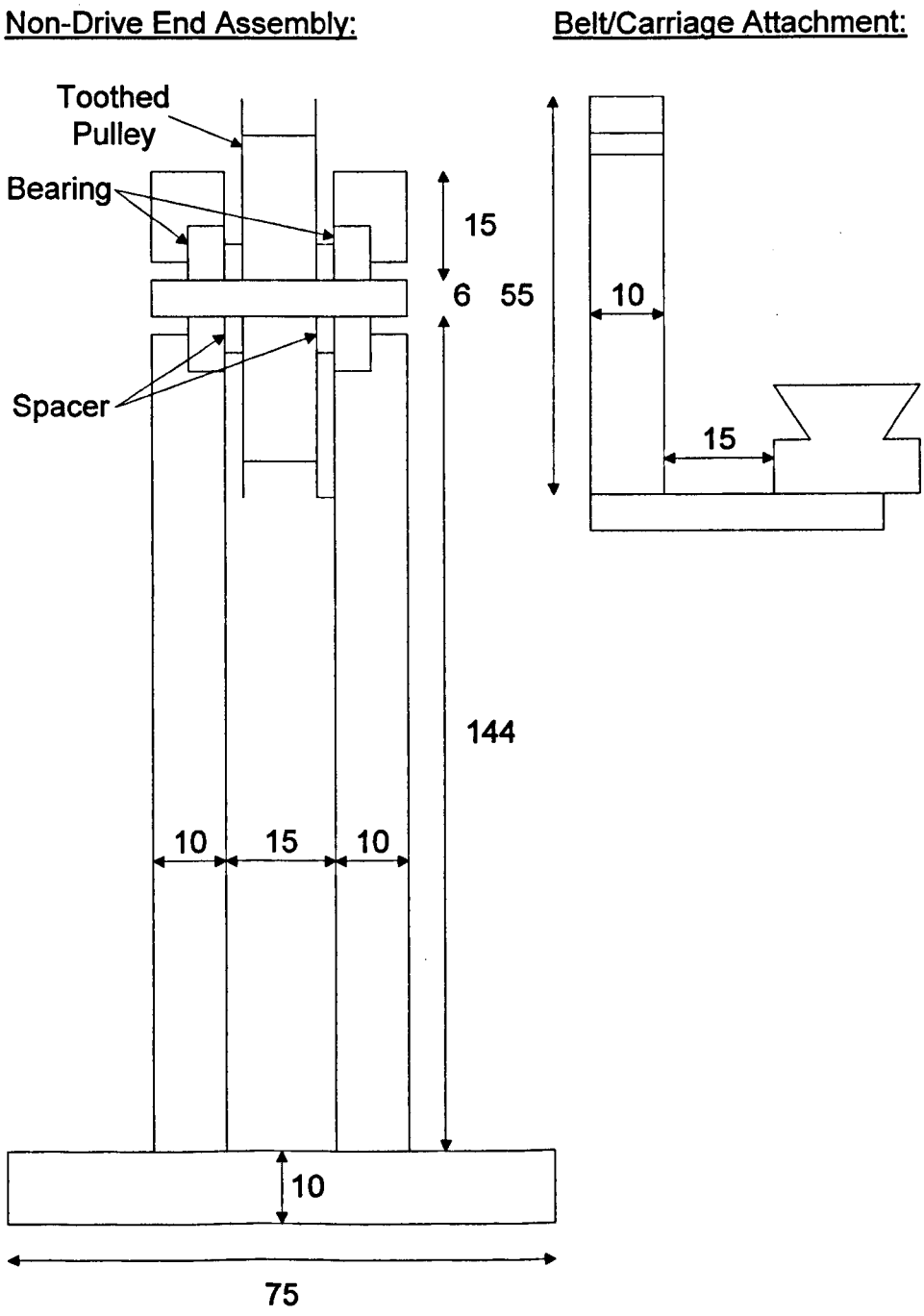


Figure 5.6: Y Drive Non-drive End Pulley Housing and the Link to the Linear V Rail.

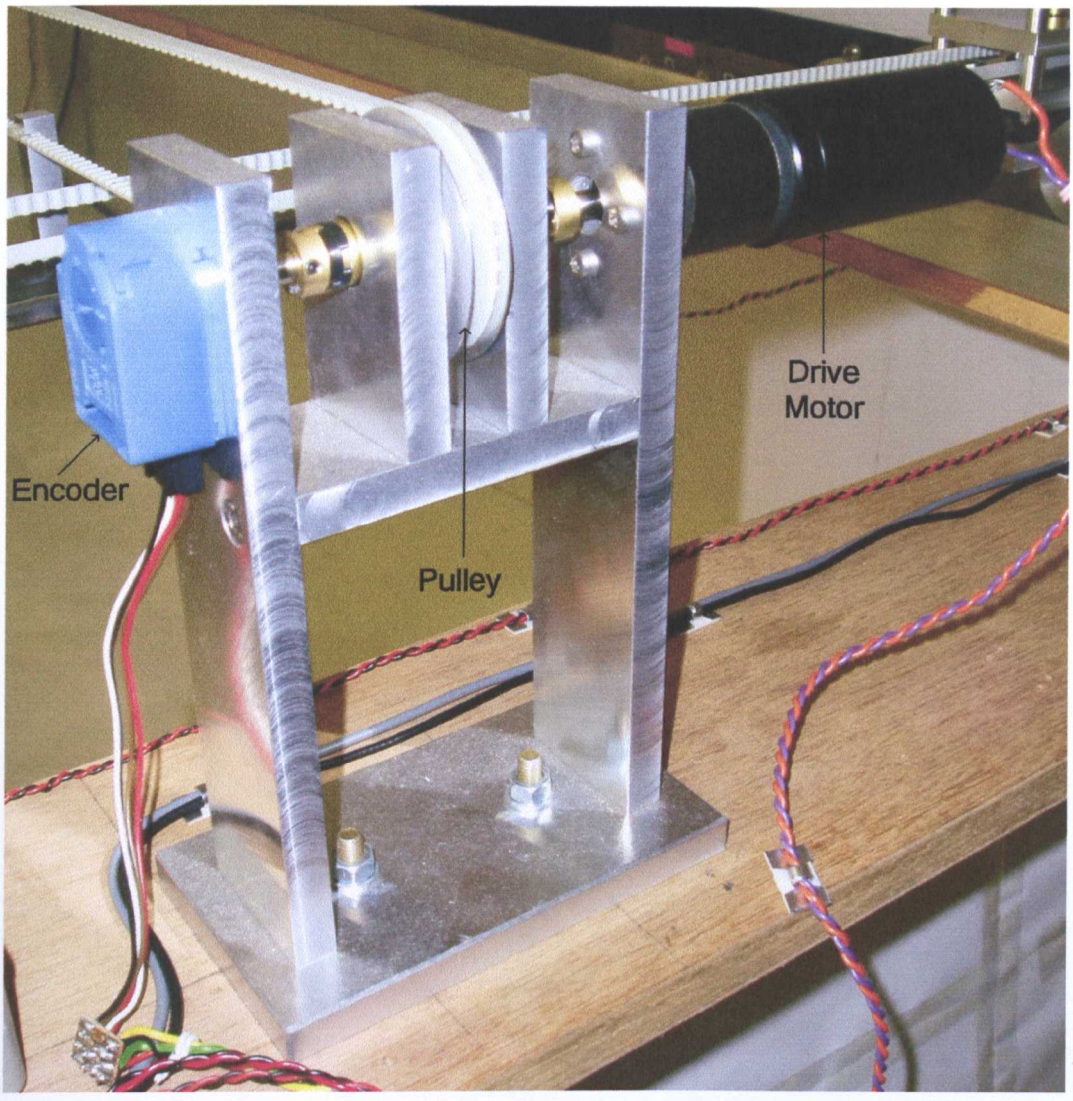


Figure 5.7: Y Drive.

5.2.3 Camera Mount/Yaw Drive

A new camera mount was required to hold a camera looking down onto the lines and act as a base for the yaw axis. The design has the motor housing fixed to the running block so as to be below the linear V rail. The camera mount is attached to the motor shaft underneath the motor housing. Figure 5.8 shows the yaw assembly and camera mount and a picture of the camera assembly is shown in Figure 5.9.

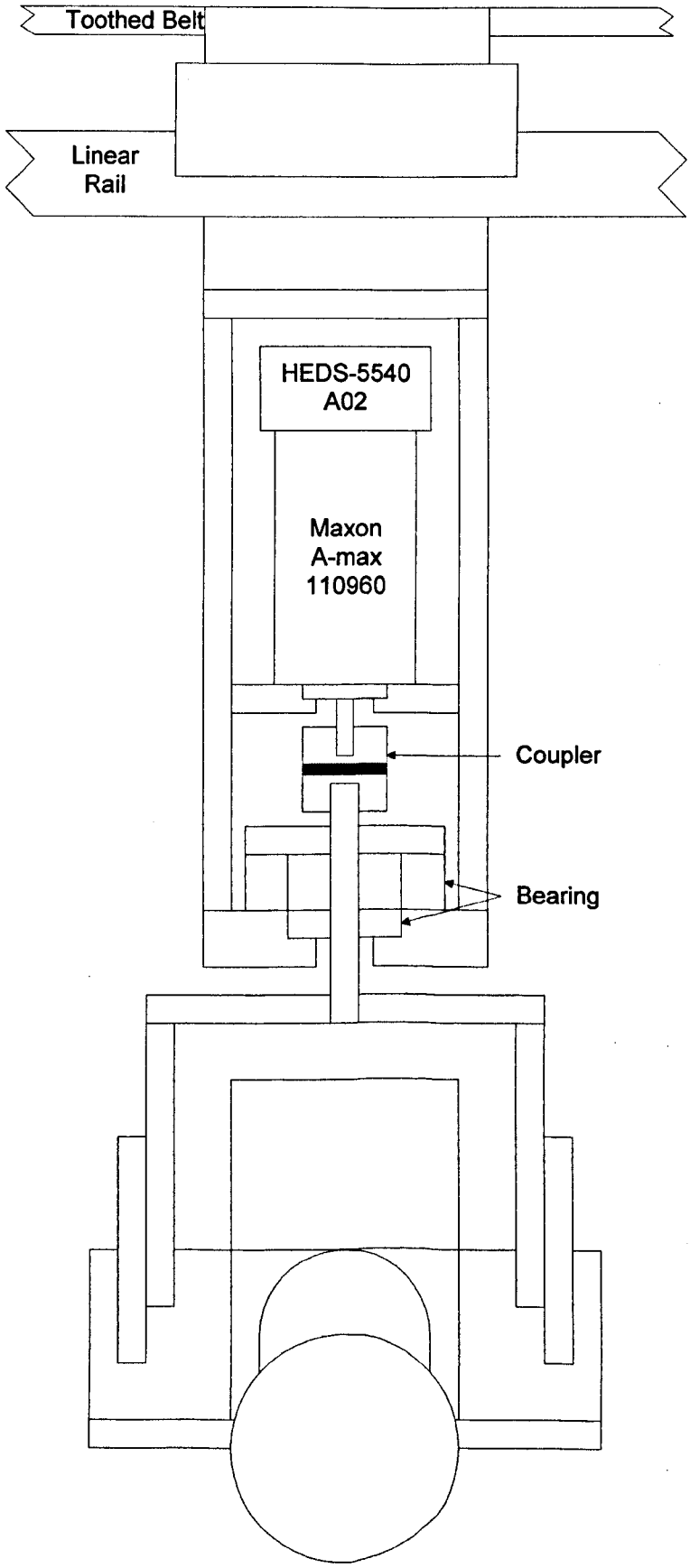


Figure 5.8: Yaw Assembly and Camera Mount Design.

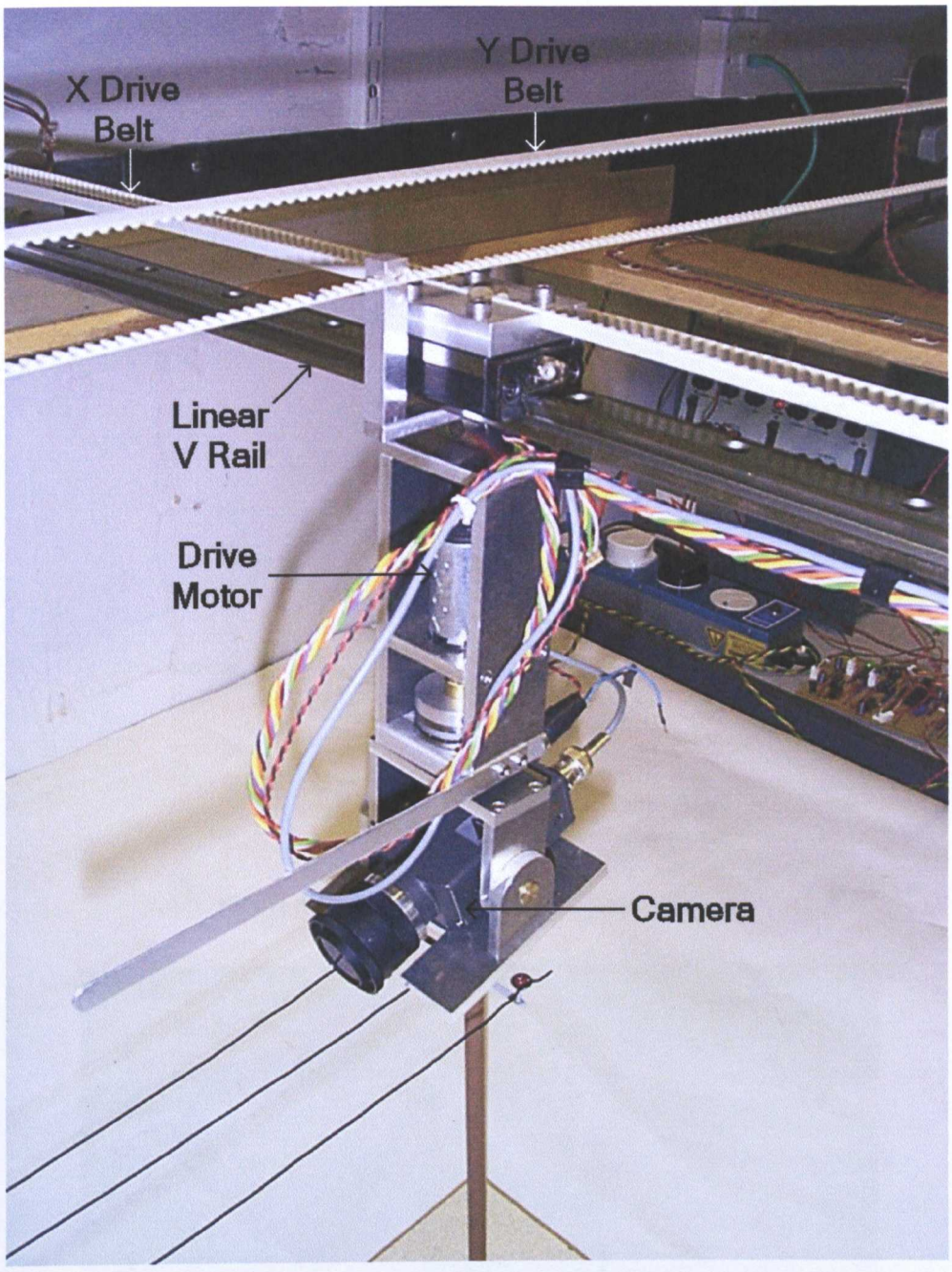


Figure 5.9: Camera Mount.

The choice of motor for the yaw axis was limited by the requirement to have a shaft at both ends, in order to allow the optical encoder to be fitted. The motor also needed to be approximately 25 mm in diameter and run from 12V. This limited the choice to a Maxon RE25 or a Maxon A-max 26 motor. It was necessary to calculate how fast each motor could turn the camera mount, to ensure it would be fast enough to simulate the rotation of the rotorcraft; a maximum rate of 1 rad/s was estimated. Both motors were found to be capable of this rate so the cheaper A-max motor was chosen.

5.2.4 Twin Camera Mount

For later experiments, two cameras were required, one pointing forwards and one pointing backwards. A second camera mount was designed and built to do this. Figure 5.10 shows the design of the mount while Figure 5.11 shows a photograph of it.

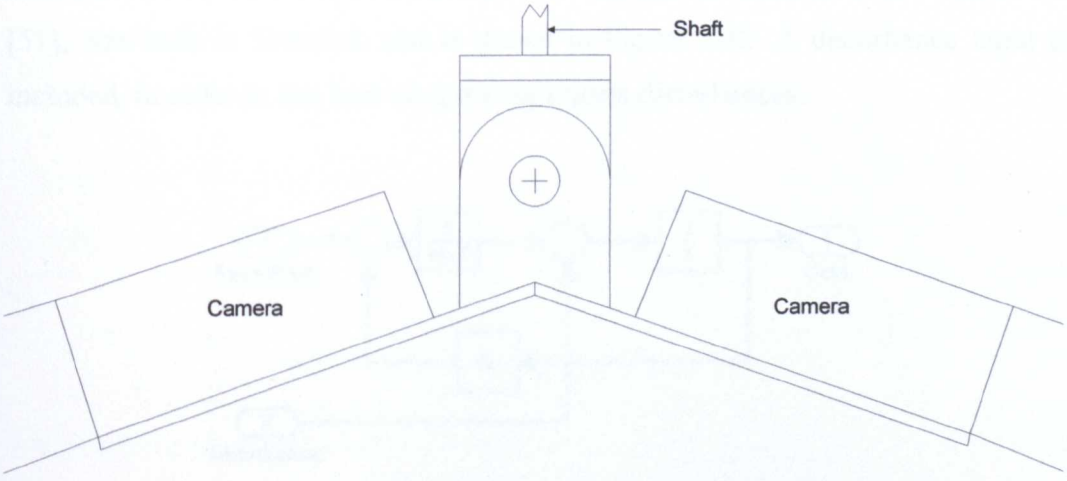


Figure 5.10: Design of the Twin Camera Mount.

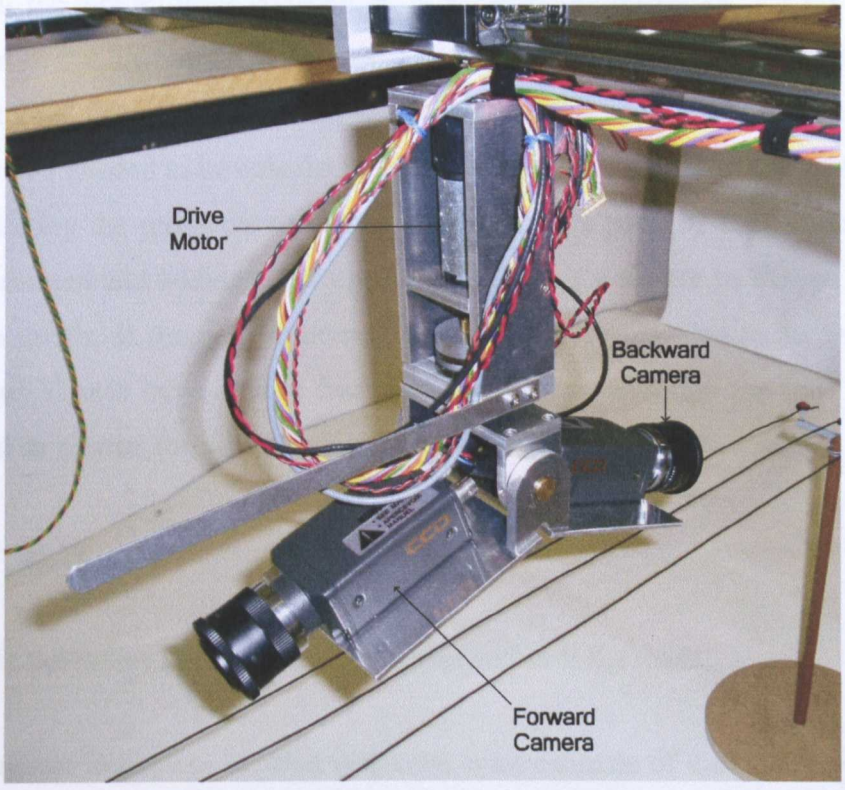


Figure 5.11: Twin Camera Mount.

5.3 Modelling of the Test Rig

5.3.1 Position Feedback Controller

In order to write the control software to provide position control of the test rig, it was necessary to develop a dynamic model of the rig. The parts of the rig being moved by each axis form a load, which places an effective moment of inertia on each motor shaft. A mathematical model of the motor, based on a servo model in [51], was built in Simulink and is shown in Figure 5.12. A disturbance input is included, in order to test how well the rig rejects disturbances.

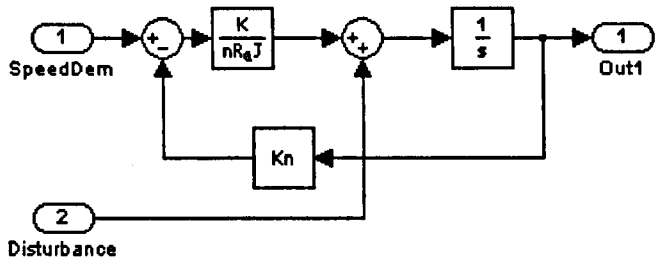


Figure 5.12: Motor Model.

Here K is the motor torque constant, R_a is the armature resistance, J is the moment of inertia of the load and n is the gearbox ratio. Values of K and R_a for each axis were obtained from the motor data sheets, while n is known for each gearbox. An estimate of J needed to be calculated for each axis. For the yaw axis this was done by calculating the moments of inertia for each of the parts of the camera mount and the camera and adding them up. As the X and Y axes are rectilinear, J is the effective inertia. It should also be noted that the loads presented to the motors by the X and Y axes have a large frictional component, although the load is being modelled as inertial for this exercise. The inertia can be calculated using:

$$J = mr^2 \quad (5.1)$$

where r is the radius of the belt pulley wheel and m is the mass.

The value or r is known, so what is needed is an estimate of the effective mass in order to calculate the effective inertia. In order to estimate the mass, a string was

attached to the drive belt. This was run parallel to the belt until the edge of the test rig, where it passed over a pulley. Masses were attached to the end of the string and the mass increased until it was sufficient to move the slider block in the case of the X axis or the linear V rail in the case of the Y axis. The moment of inertia was then calculated for the two axes using:

The values for the three axes are shown in Table 5.1.

Axis	Yaw	X	Y
K (NmA^{-1})	0.0146	0.0163	0.0525
J (kgm^2)	$5.1x10^{-4}$	$4.5x10^{-4}$	$52.42x10^{-4}$
R_a (Ω)	2.5	1.23	2.07
n	1	35	27

Table 5.1: Values for the Motor Model for each Axis.

Initially a rig model was built with only the X and Yaw axes, because the Y axis is expected to behave in a similar manner to the X axis. Figure 5.13 shows this model, which includes a model for the motor stiction at the input to each motor, shown in Figure 5.14.

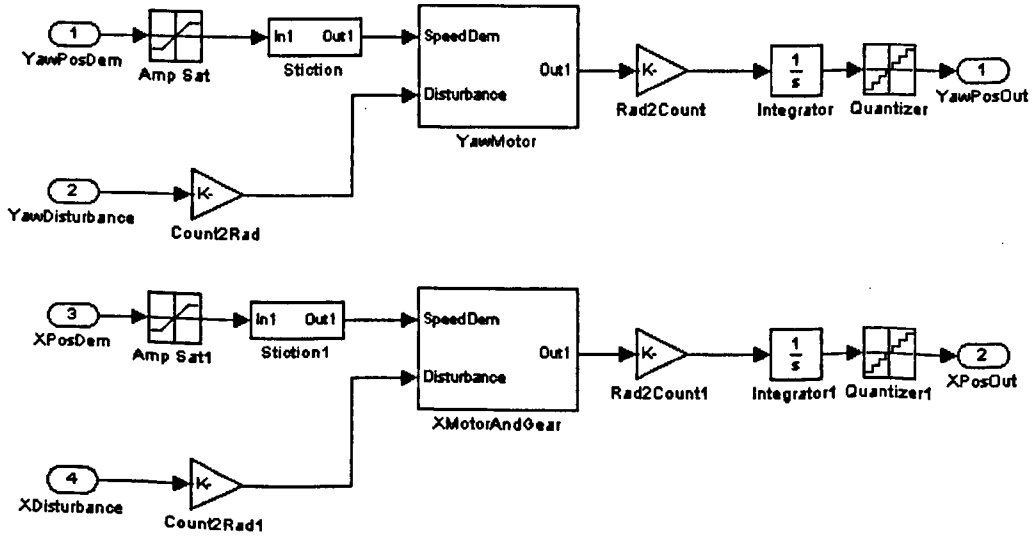


Figure 5.13: Test Rig Model.

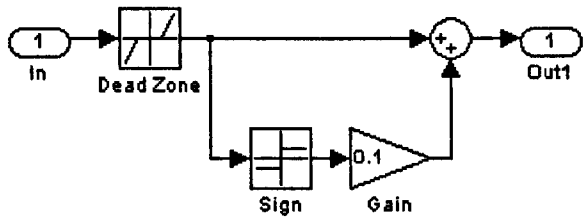


Figure 5.14: Simulink Stiction Model.

A controller model based on position feedback was implemented. Optimum gains to give a critically damped response were found using SISOtool and included in the Simulink model. Due to the non-linearity in the motor model, a higher gain was needed for the yaw axis than the optimum determined by linear analysis. Figure 5.15 shows the controller model. This was tested with square wave inputs into both the X and Yaw axes, to test the step response. The magnitudes of the demand inputs are ±2m (scaled) for the X axis and ±45° for the Yaw axis and the results are shown in Figure 5.16.

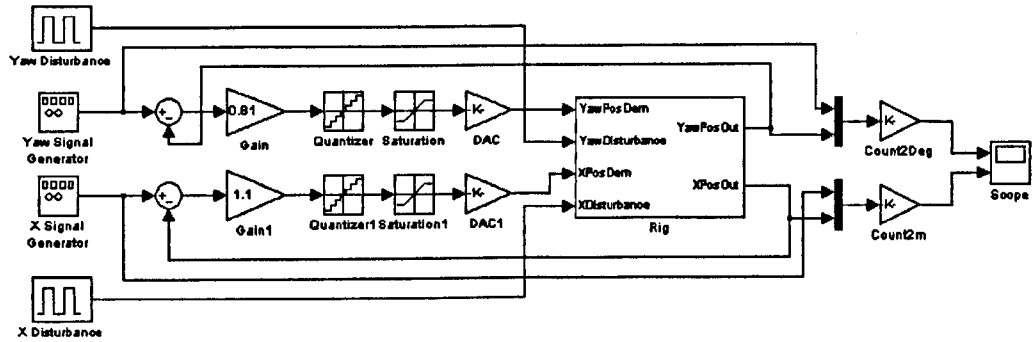


Figure 5.15: Test Rig Position Feedback Controller Model.

In Figure 5.16 it can be seen that we get a critically damped response (solid) to the step demand (dotted) with the X axis, as predicted by SISOtool. The yaw axis gives a very under-damped response to the step input.

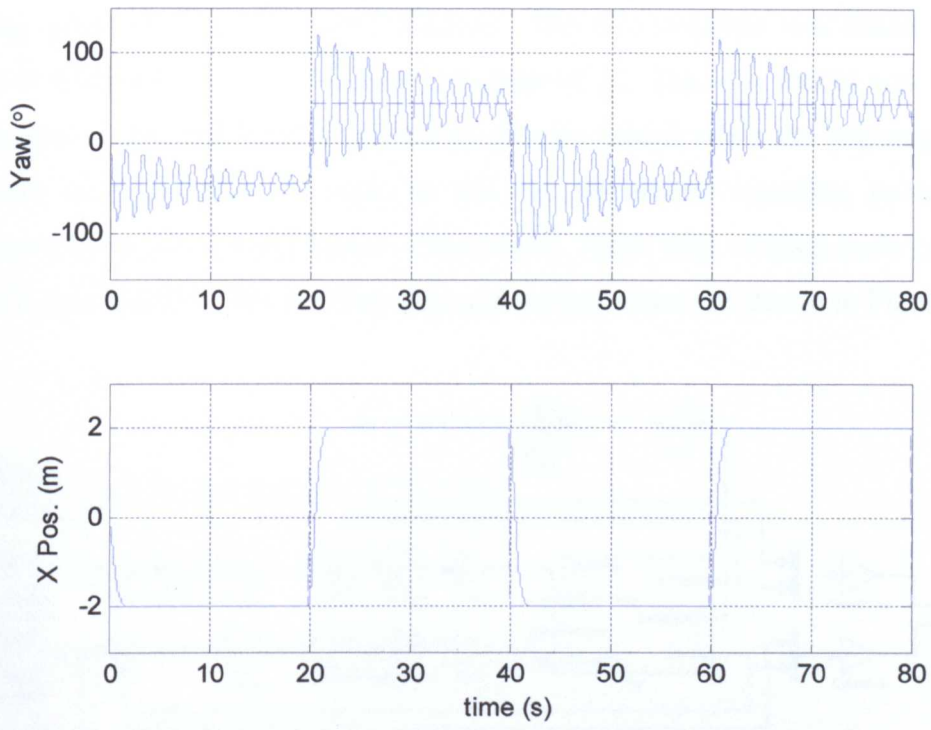


Figure 5.16: Position Feedback Controller Output.

5.3.2 Speed Feedback for the Yaw Axis

Due to the oscillatory response, it was necessary to alter the yaw controller. Speed feedback was added to the yaw axis. The model was adjusted to give the motor speed, as shown in Figure 5.17, although on the test rig the speed would be obtained by differentiating the position signal.

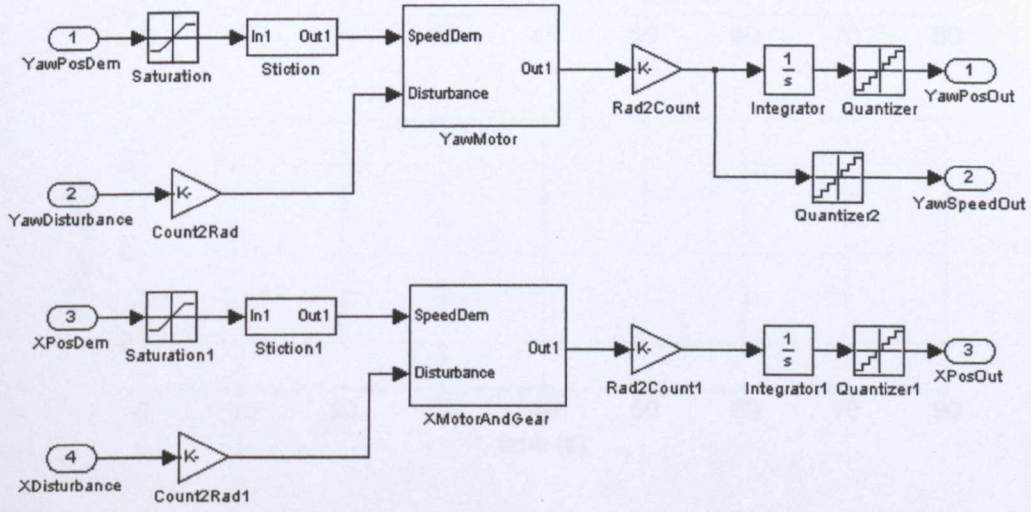


Figure 5.17: Test Rig Model With Yaw Speed Output.

To select the value of the speed feedback, the yaw model was put into SISO tool, using different values of speed feedback. The best response was found using a speed feedback gain of 0.1 and a loop gain of 22. The new model was built in Simulink and tested. Figure 5.18 shows the new model, while the step response is shown in Figure 5.19. In order to test the disturbance rejection, pulses were applied to the disturbance inputs of the model. These were of magnitude $1ms^{-2}$ for the X axis and $45°s^{-2}$ for the Yaw axis and the responses are shown in Figure 5.20.

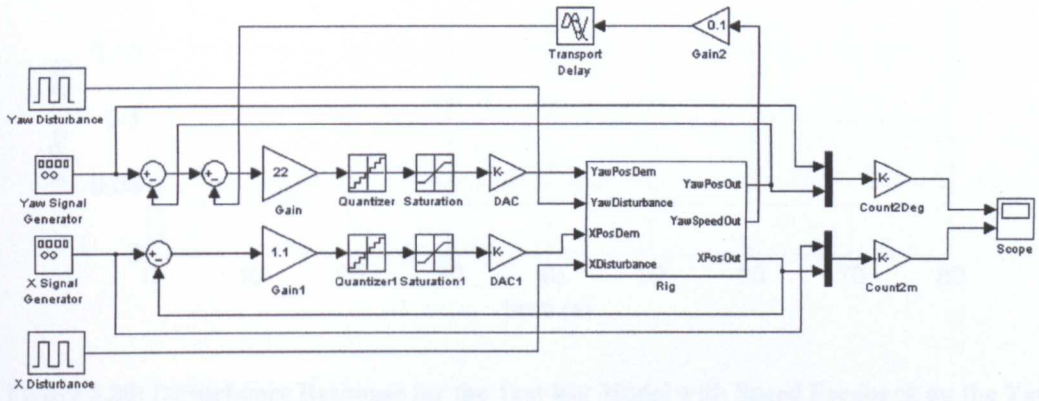


Figure 5.18: Test Rig Controller with Speed Feedback on Yaw Axis.

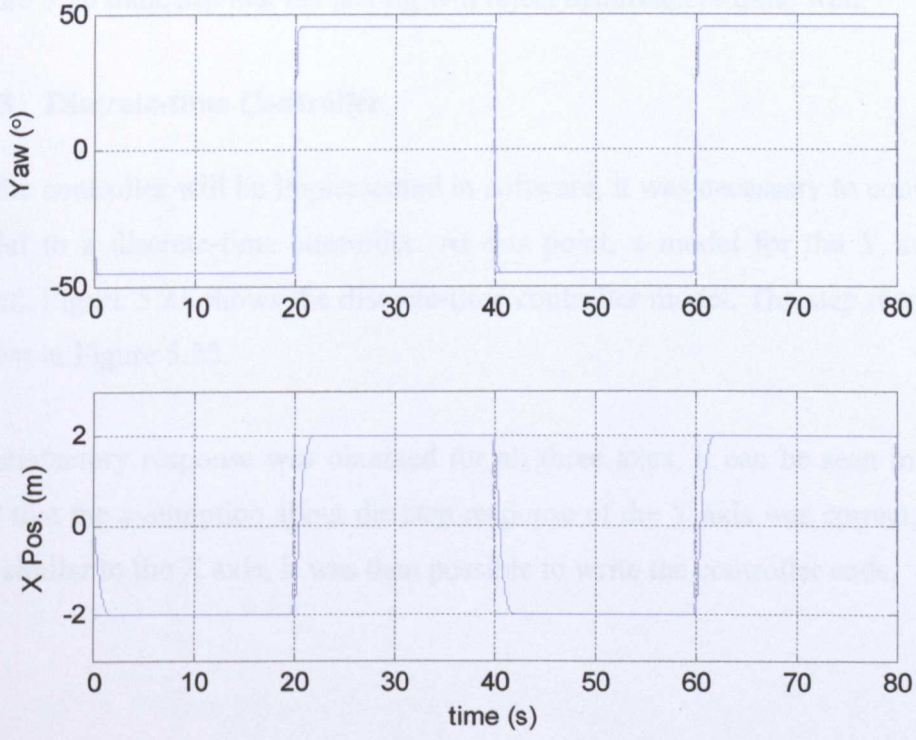


Figure 5.19: Step Response for the Test Rig Model with Speed Feedback on the Yaw Axis.

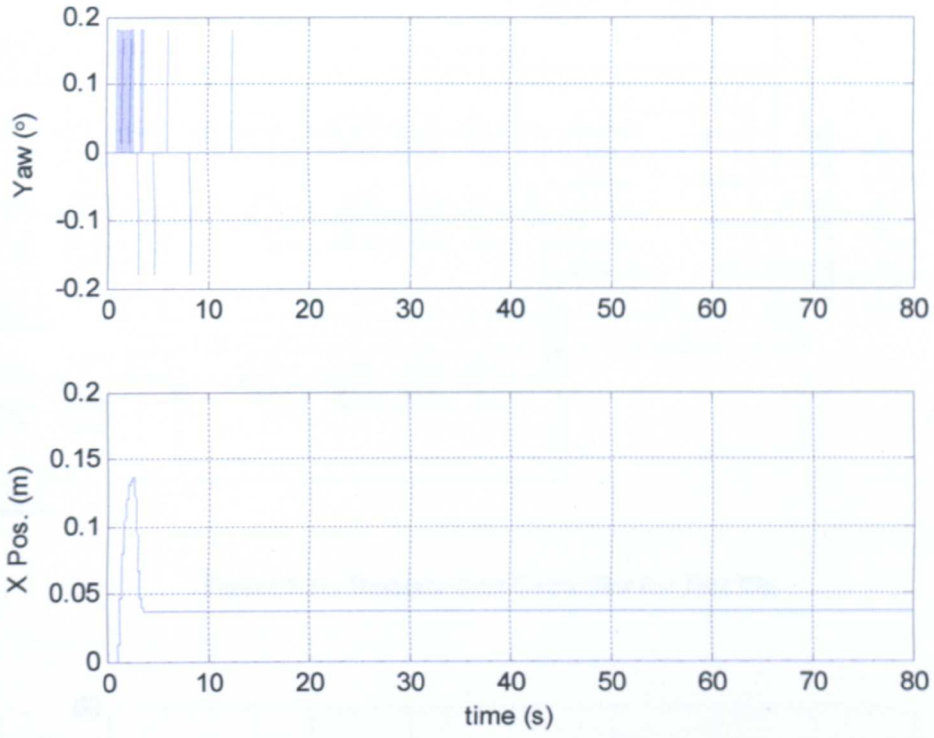


Figure 5.20: Disturbance Response for the Test Rig Model with Speed Feedback on the Yaw Axis.

It can be seen from Figure 5.19 that the Yaw response is very much improved. Figure 5.20 indicates that the test rig will reject disturbances quite well.

5.3.3 Discrete-time Controller

As the controller will be implemented in software, it was necessary to convert the model to a discrete-time controller. At this point, a model for the Y axis was added. Figure 5.21 shows the discrete-time controller model. The step response is shown in Figure 5.22.

A satisfactory response was obtained for all three axes. It can be seen in Figure 5.22 that the assumption about the step response of the Y axis was correct, in that it is similar to the X axis. It was then possible to write the controller code.

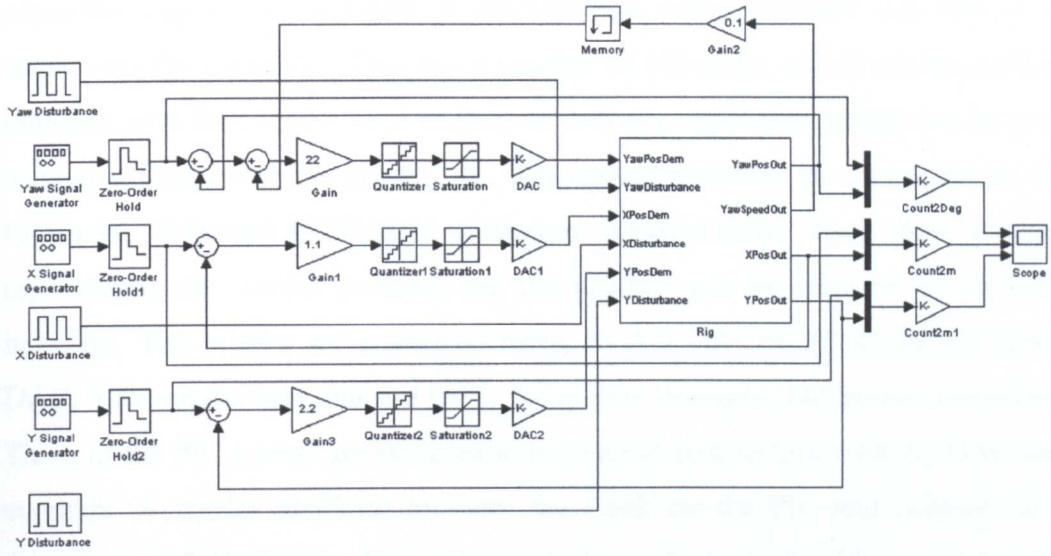


Figure 5.21: Descrete-time Controller for Test Rig.

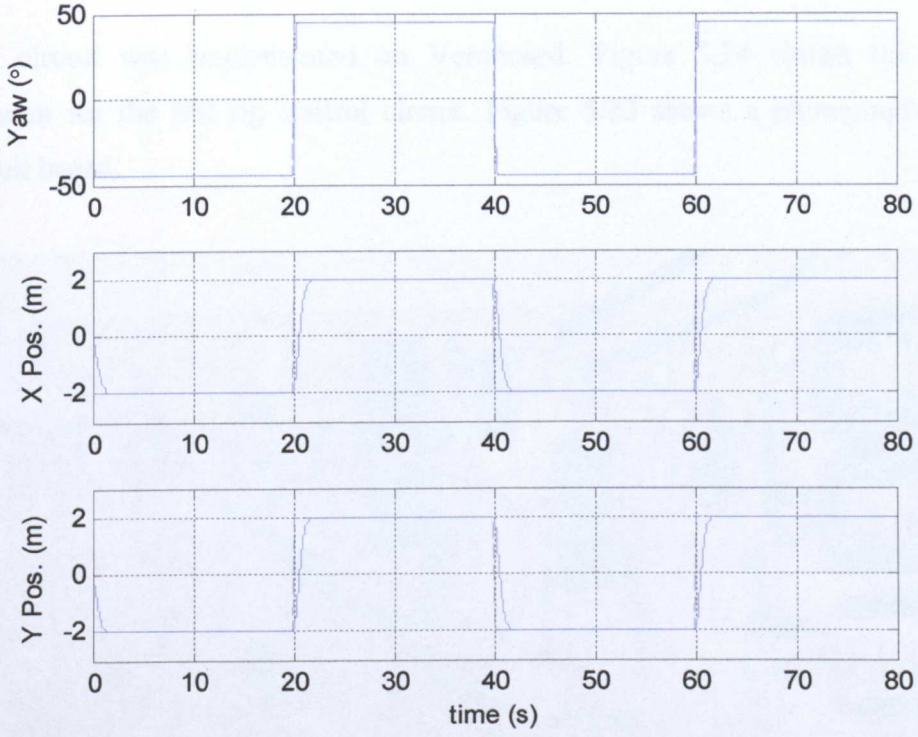


Figure 5.22: Output for All Three Axes for the Discrete-time Controller.

5.4 Electronic Design

The position sensors on the rig are HP HEDS optical encoders, which produce digital output and the rig controller needs to communicate with the control PC. For these reasons, it was decided that it would be easiest to implement the

controller digitally in software. A PIC16F874A microcontroller was chosen to implement the controller. This has a number of I/O ports, which can be used to interface with the encoders and motors, as well as a serial port, which can be used to communicate with the control PC. The optical encoders are connected to the PIC using dedicated HCTL2016 quadrature decoder chips. These hold a count representing the current position, and the position can be read out by an 8-bit interface. The motors are connected using an AD DAC8412F 4-channel 12-bit DAC. The outputs from this are fed to the motors through L165 power amplifiers. Three of the PIC's ports are assigned to form a bus to interface with the DAC and counters. A crystal oscillator provides the clock for the PIC and counters at a frequency of 3.6864MHz. This value was chosen because the PIC generates the baud clock for the serial port from this clock and this value divides down to give the 38400baud required.

The circuit was implemented on Veroboard. Figure 5.24 shows the circuit diagram for the test rig control circuit. Figure 5.23 shows a photograph of the circuit board.

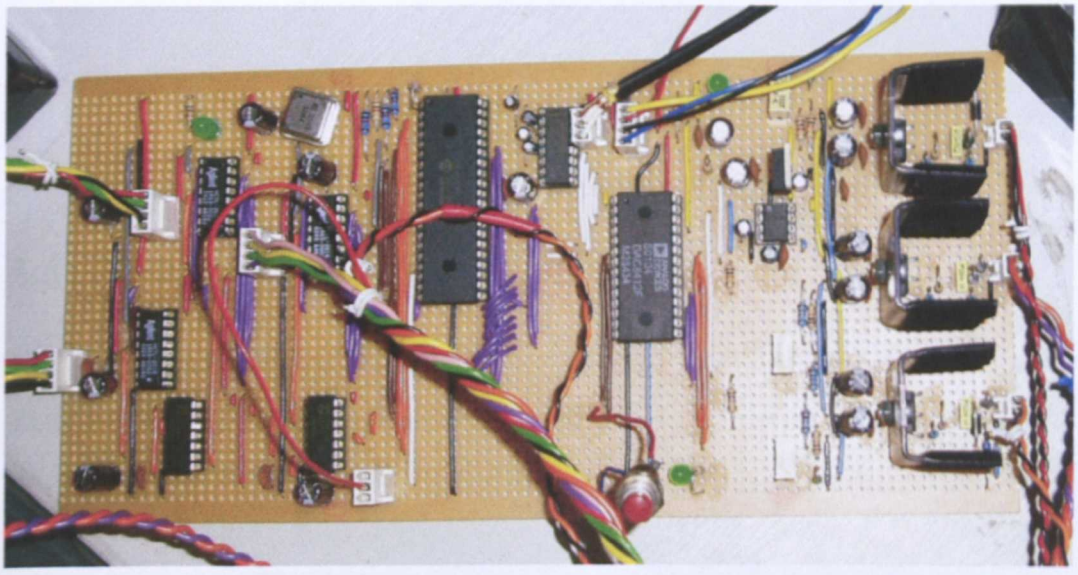


Figure 5.23: Test Rig Control Circuit Board.

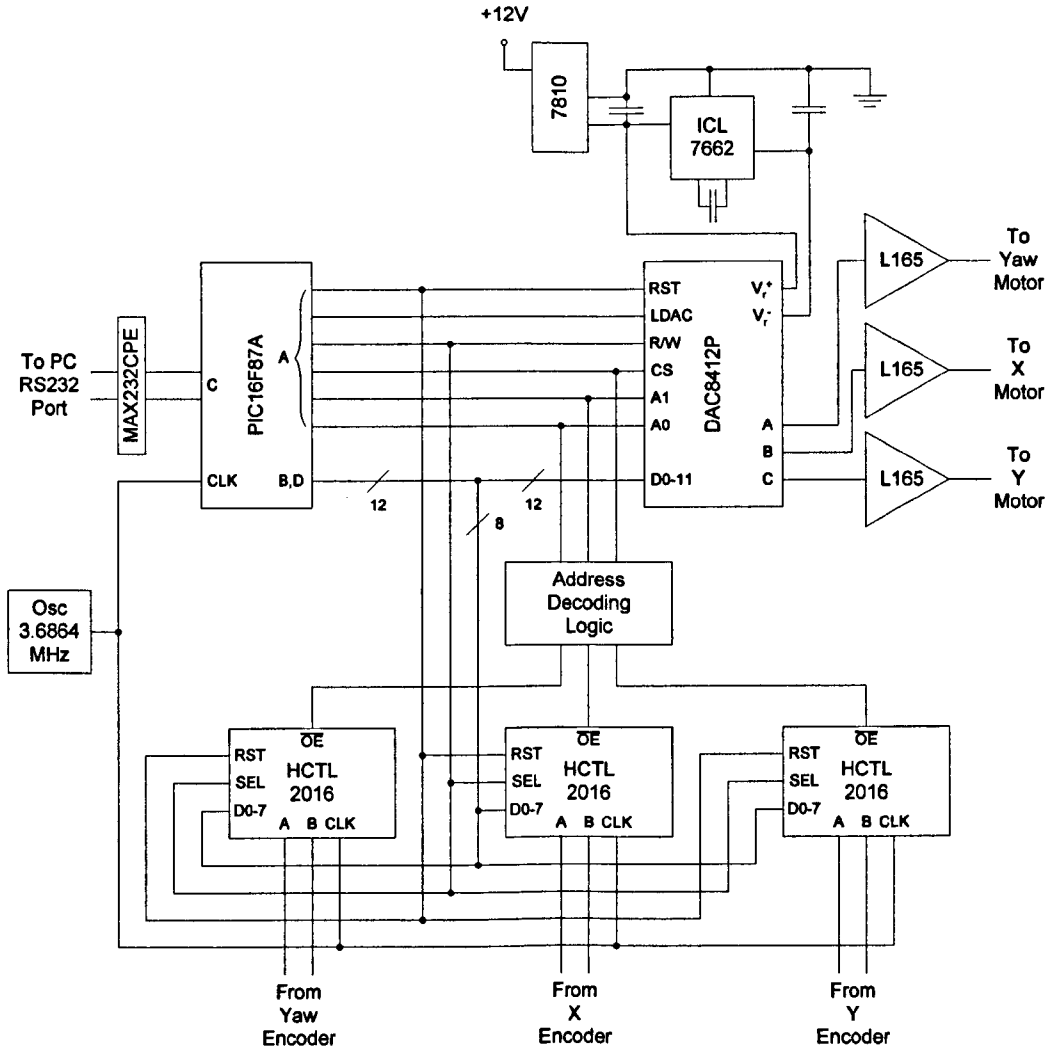


Figure 5.24: Test Rig Circuit Diagram.

5.5 Embedded Software

The PIC microcontroller had to be programmed to perform the control. Its main functions are to implement the controller and to communicate with the control PC. These two functions must run concurrently. This was implemented using the PIC interrupt. The control loop is the most time critical of the functions and was run at interrupt level, with the communication routine run at normal level. The microcontroller software is written in MPASM, the assembly language for PIC microcontrollers. Figure 5.25 shows the state transition diagram of the embedded software. It should be noted that in Figure 5.25 there are two transitions leaving the Control Algorithm state marked end. This is possible because these transitions

are a "return from interrupt" and so the software returns to the state it was in before the interrupt.

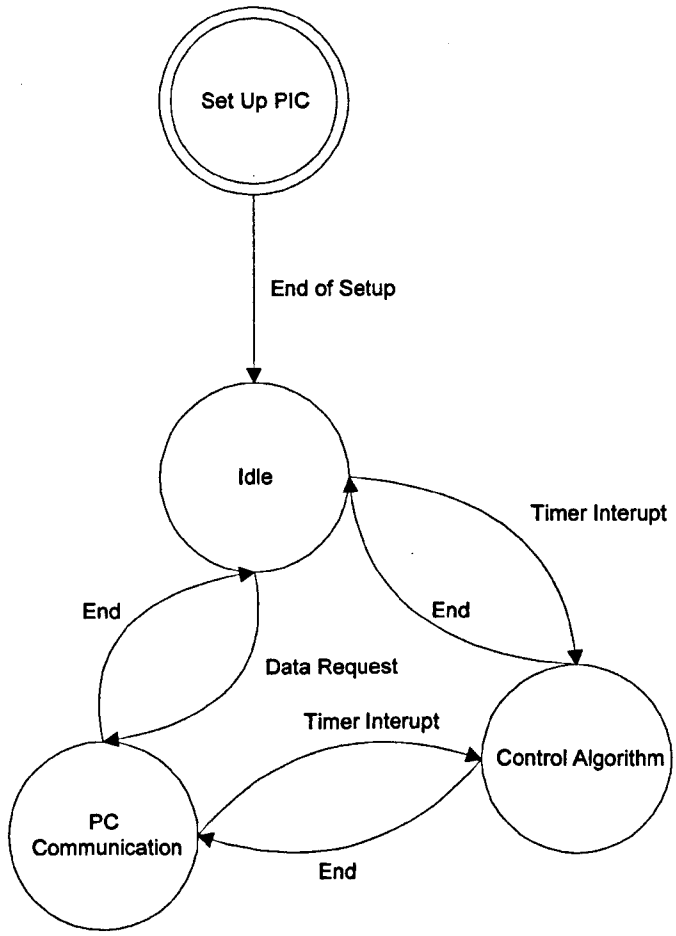


Figure 5.25: Embedded Software State Transition Diagram.

The PIC contains a timer and this is set to trigger the interrupt at a frequency of 450Hz. This value had to be sufficiently high in order to control the Yaw axis, which has a response time of 0.5s. There is an upper limit to the frequency caused by the clock frequency driving the PIC. The specific value of 450Hz was chosen due to the frequency of the crystal that drives the system, 3.6864MHz, which itself was chosen in order to provide the correct baud rate for the serial port.

The controller must provide position control for the X and Y axes and both position and speed feedback on the Yaw axis. Because the Yaw speed is obtained by differentiating the position signal, it is quite noisy. This caused some chatter of

the Yaw axis so a first order infinite-impulse response low pass filter [52] on the Yaw demand signal was used to remove it.

In order to test the validity of the model of the test rig described in section 5.3, the step response of the test rig was measured for all three axes and the results are shown in Figure 5.26.

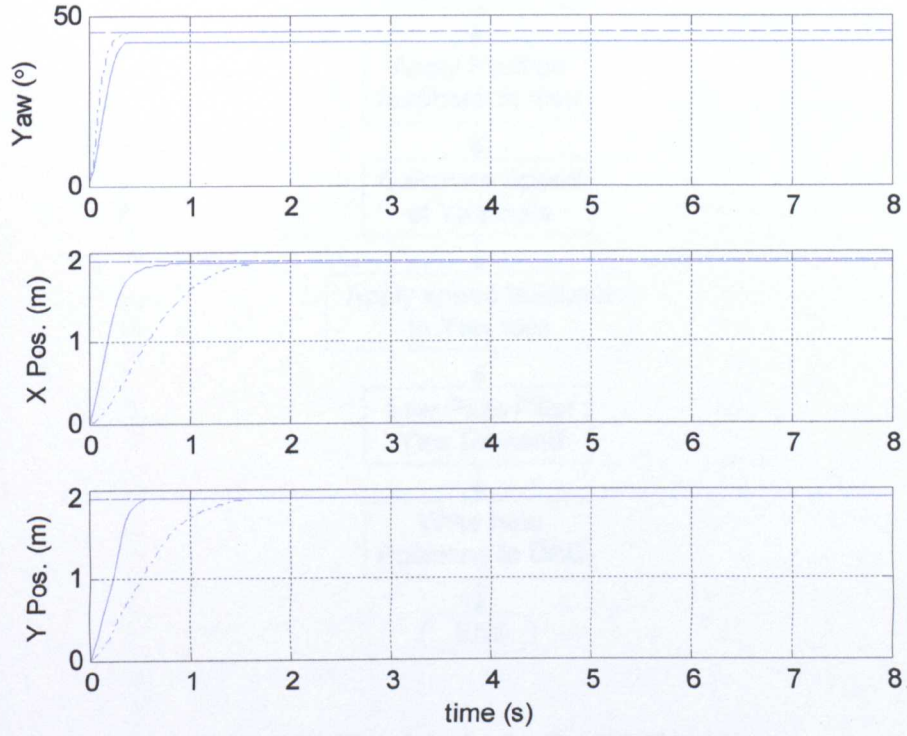


Figure 5.26: Comparison between the Step Responses of Rig and Simulink Model of the Test Rig.

Figure 5.26 shows that the step response of the test rig's yaw axis (solid) matched well with the predicted model (dash-dotted), although there is a position offset. This is due to the cables going to the camera acting as a spring. There was some difference in the X and Y response in that the rig's response time is quicker than predicted. This is due to the fact that the load was modelled as inertial while it is partly frictional. While inertia tends to hinder both acceleration and deceleration, friction aids deceleration. This allowed a higher loop gain to be used than predicted without causing an overshoot, thus allowing the faster response. The gains were rounded to powers of two, in order to speed up the operation of the software. Figure 5.27 is a flowchart showing how the control algorithm works.

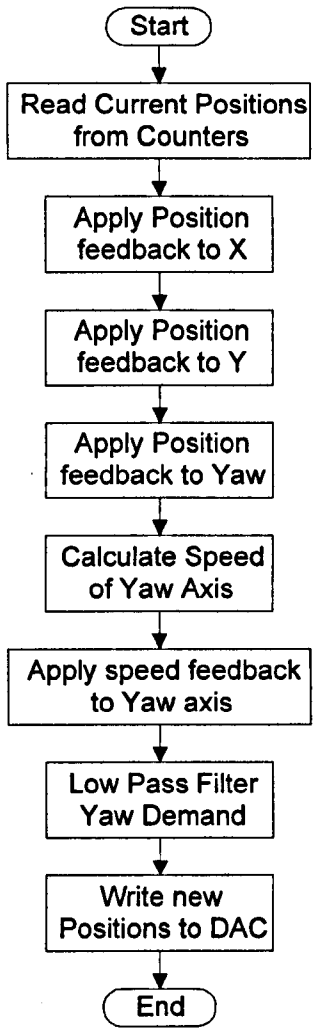


Figure 5.27: Flowchart for the Control Algorithm.

In between calls to the control algorithm, the PIC needs to monitor the serial port for requests from the control PC. In most cases this will be the PC sending new position demands and read back the current positions. In addition, the PC needs to be able to check that the communication channel is working, and reset the rig position to zero. Figure 5.28 shows a flowchart for the communication routine.

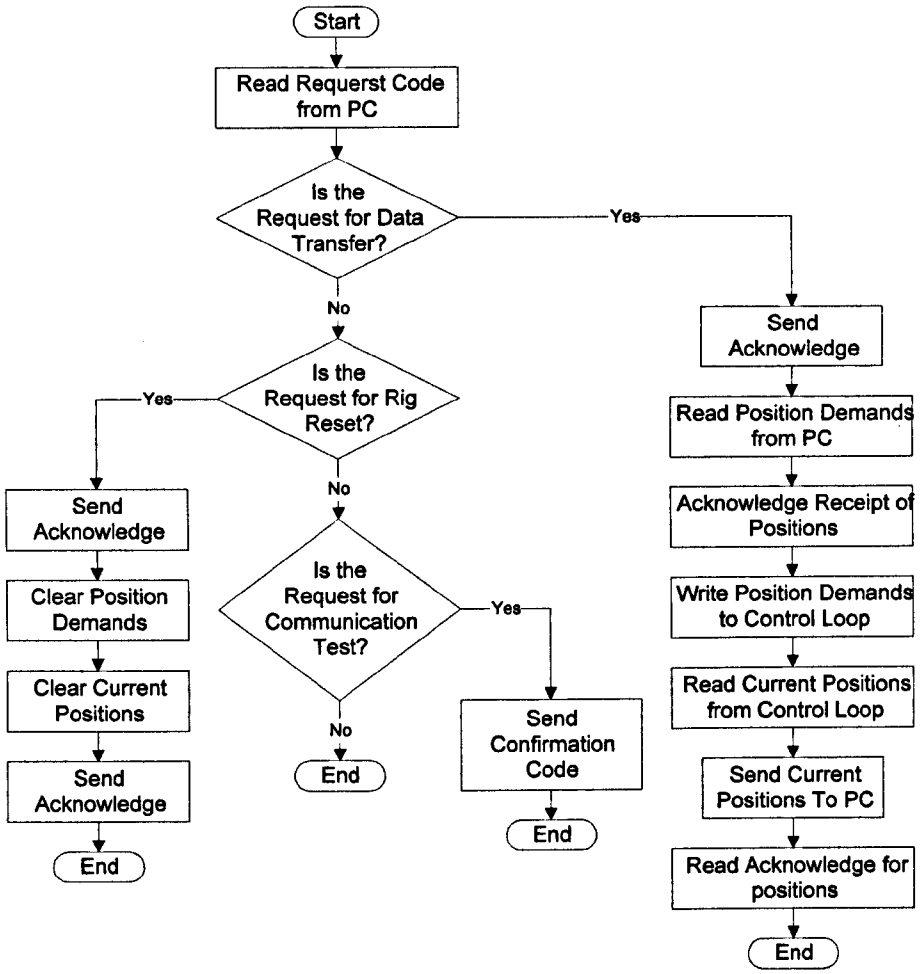


Figure 5.28: Flowchart for the Serial Communication Routine.

5.6 Control Software

The control software provides simulation of the UAV and the image processing required to do visual control. This software runs on a PC under Microsoft Windows XP and was written using Microsoft Visual C++ 6.0. In order to grab images from the test rig's camera, the control PC has a Matrox Meteor framegrabber card in it. This is accessed from the C code using the supplied MIL software.

The PC software is multithreaded. One thread operates the user interface, while the other two run the control and UAV model, described in Chapter 3, and the image processing and tracking described in Chapters 6, 7 and 8. The vision processing thread is started and stopped by the control thread, which also reads

the line positions from the vision thread. The control thread is itself started and stopped by the user interface. The tasks performed by each thread and the thread ownership is shown in Figure 5.29

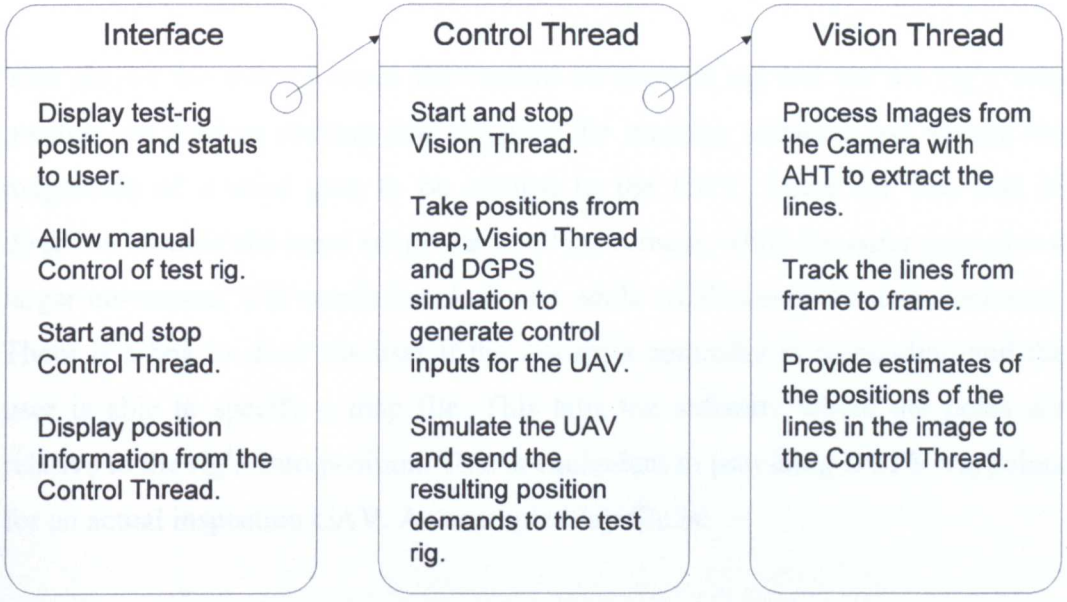


Figure 5.29: Separation of tasks within the Control Software.

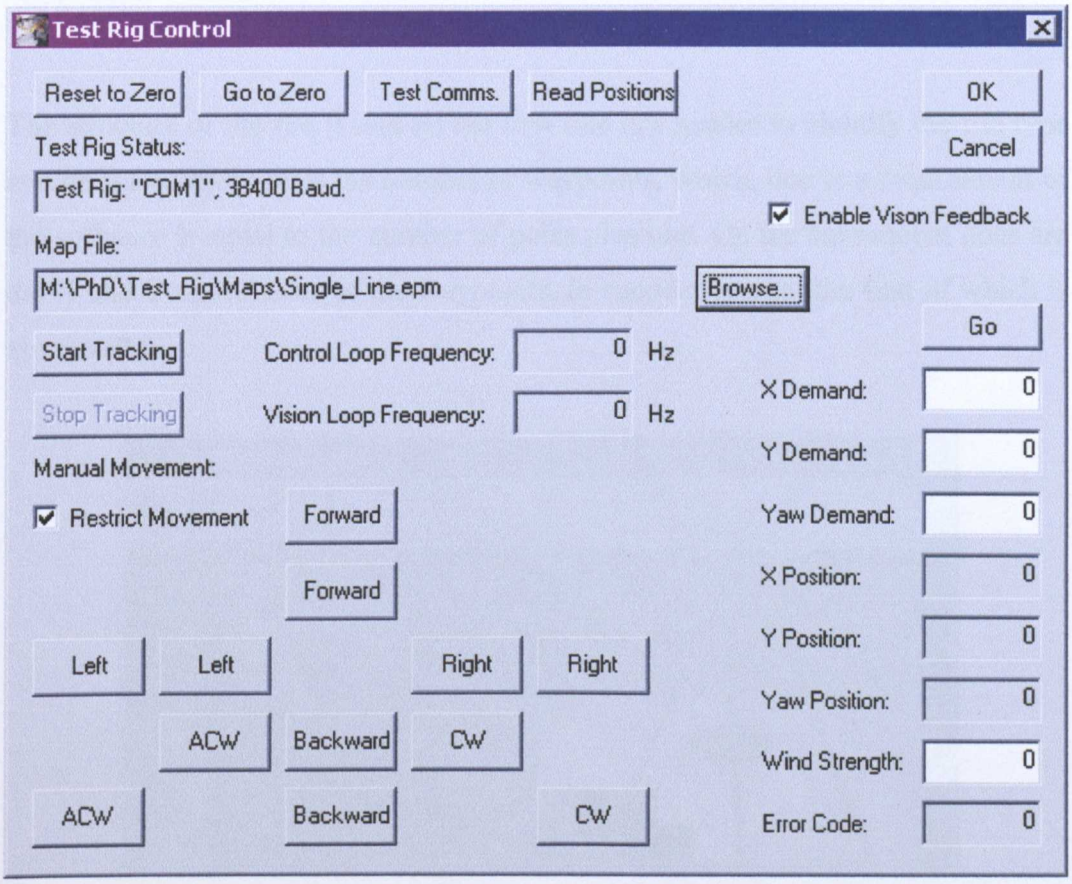


Figure 5.30: Test Rig Control Software Interface.

The interface was put together using the tools within the Visual C++ development environment. The control and vision threads implement the control and vision algorithms described elsewhere in this thesis. Figure 5.30 shows the user interface.

This allows the user to move the camera on the test rig and set the rig's zero position, as well as starting and stopping the tracking software and setting the magnitude of a wind gust to be applied to the UAV. There are two sets of direction buttons: the inner set give a small movement, while the outer ones give a larger movement. CW stands for clockwise while ACW stands for anti-clockwise. There is a box to show the user if the test-rig's controller is responding and the user is able to specify a map file. This tells the software where the poles are relative to the rig's zero position. This is equivalent to providing DGPS waypoints for an actual inspection UAV. An example Map file is:

```
EPM1
3
0 0
3515 3350
3515 14840
```

The structure of the file is that on the first line is a header to identify the file type and on the second line is the number of waypoints, which, due to a requirement of the software is equal to the number of poles plus one. On the subsequent lines are the X and Y coordinates of the waypoints, in encoder counts, the first of which is always (0,0).

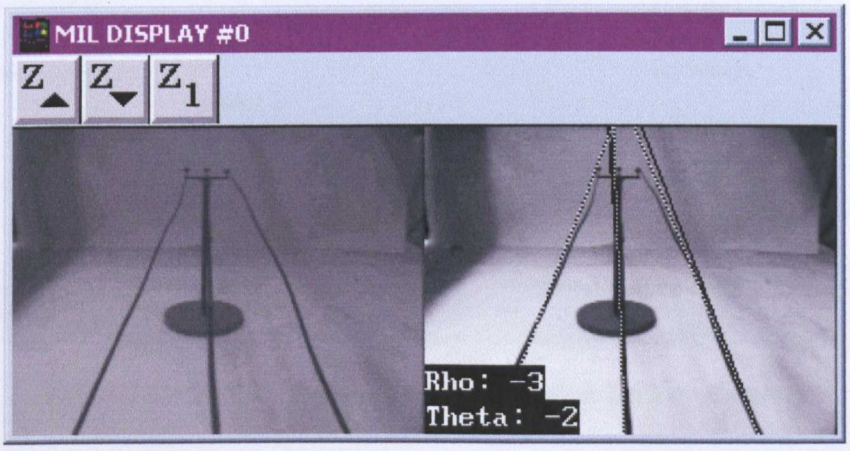


Figure 5.31: Example Image and the Lines Found by the Hough Transform.

Figure 5.31 shows an example frame with the processed image next to it.

For later experiments two cameras were used, which required two image processing computers, one per camera. The software was split to run on three computers, two running the vision threads and one performing the control. The vision and control parts of the software were left unchanged. The change to three computers only required modifications to the user interfaces and the addition of code to handle the communication between them. For simplicity and because the amount of data to be transferred between the PCs was quite small, serial communication was used. Figure 5.32 shows the new control interface, which is similar to the interface for the single camera software, apart from the addition of the boxes to display the connection status of the vision PCs. Figure 5.33 shows the interface for the vision part of the software. This allows the user to start and stop the vision thread and specify whether captured images are written to disk.

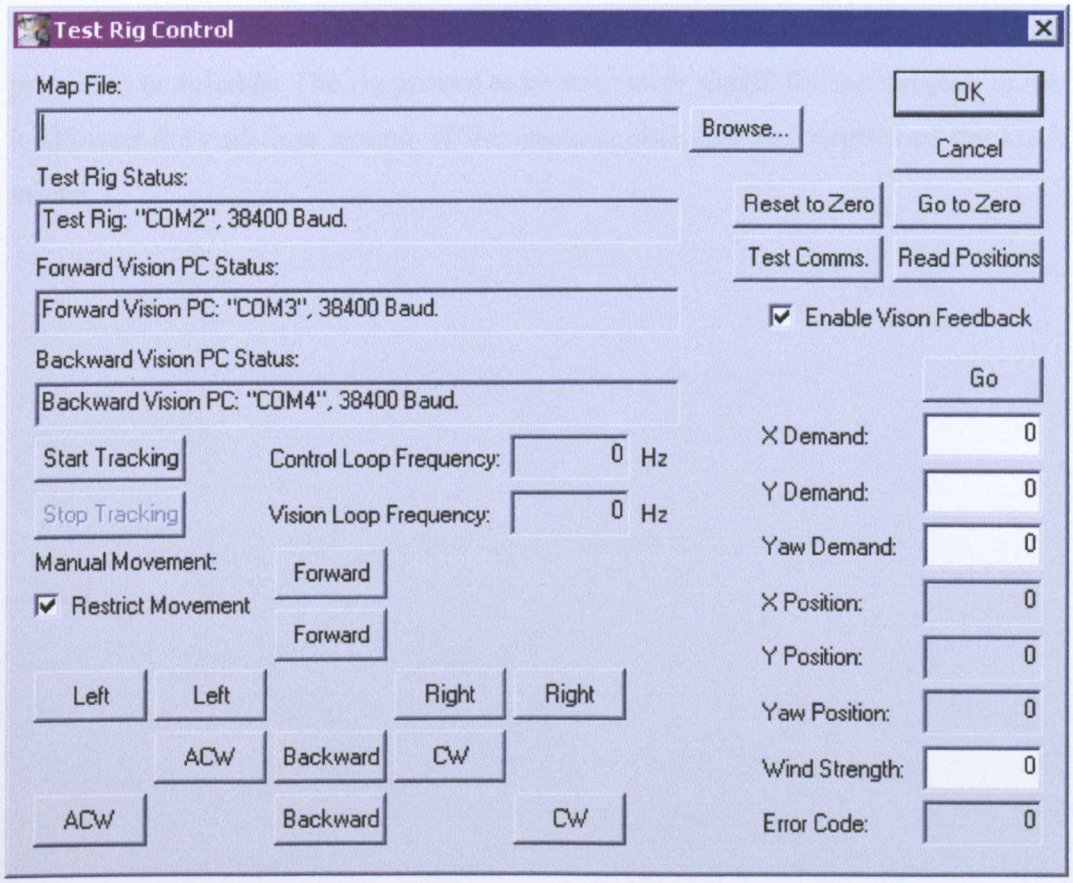


Figure 5.32: Test Rig Control Software Interface for Two Cameras.

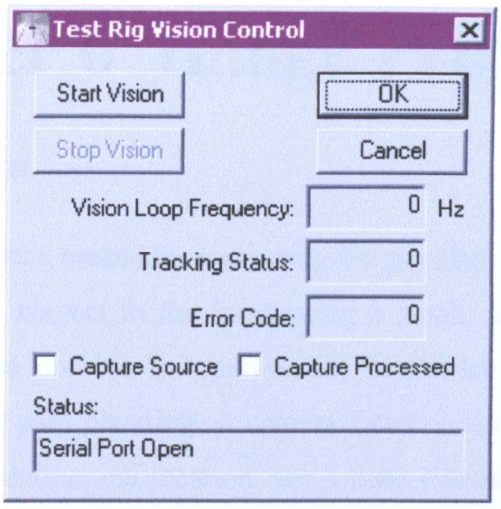


Figure 5.33: Test Rig Vision Software Interface.

5.7 Conclusions

The test rig successfully gives accurate position control of the X and Y position and the Yaw of the camera. The test rig did take quite a lot of time to design and build, but once this was done very little maintenance was needed and the rig proved to be reliable. The rig proved to be extremely useful for this project, in that it allowed the real-time testing of the tracking software and control of the UAV model.

Chapter 6 Image Processing

6.1 Introduction

Image processing offers a means for measuring the position and orientation (pose) of the rotorcraft with respect to the lines using a small, lightweight and cheap sensor which can also provide information for higher-level functions, such as obstacle detection and path planning. A camera fixed on the rotorcraft body will, depending on its orientation and location, see 3 lines extending into the distance, converging due to perspective distortion. Although they are in fact catenaries, from overhead they are approximated reasonably well by straight lines.

The UAV needs to be able to track the lines from frame to frame in order to maintain lock onto the lines. In order to do this, the lines must be located in the image. Each frame needs to be processed in order to locate the lines within it and this must be done with a good degree of accuracy. The lines appear up to six pixels wide in the image and the test rig lines do have a few kinks that would not be present on the actual lines. While most of the time the estimated position of the line from the image processing should appear within a few pixels of the actual line, some may be up to 10 pixels away. This occurs due to kinks in the model lines. Such kinks don't occur on full sized power lines due to the weight of the cable. That the estimate of the line positions occasionally appear up to 10 pixels away from there actual position in the image should not cause a problem as filtering is included in the tracker. The slow response time of the UAV means that it is unable to respond to high frequency noise. As this is a real time application, the image processing must also be fast. The sample time for the vision feedback must be considerably smaller than the response time of the UAV. A frame rate of ten frames per second should be sufficient for this application. A compromise between these two criteria is necessary. The Hough Transform [48, 53] finds straight lines in an image and is also a relatively fast method, computationally, so this is used as the basis of the image processing. Successful application of the Hough transform requires pre-processing of the raw image and post-processing in the transform space. The image processing is summarised in Figure 6.1. On the

current hardware this takes approximately 8ms with the first three tasks taking around 28% of that time each.

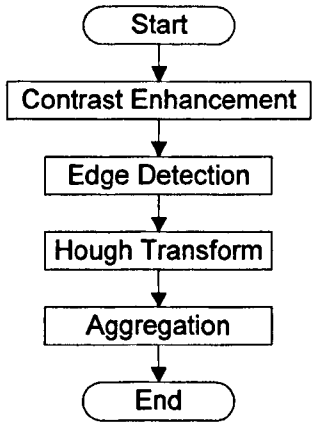


Figure 6.1: Flowchart showing the Operation of the Image Processing.

6.2 Contrast Enhancement

The varying light levels encountered in this application greatly affect the image processing. The images from the camera do not use the full dynamic range of intensity representation: in low light levels the images from the camera use only the lower end of the dynamic range while, with high light levels, only the upper end of the dynamic range is used. Contrast enhancement can help with this by adjusting the brightness of each pixel such that the full dynamic range is used to store the image. Methods of doing this include contrast-stretching, histogram equalisation and histogram specification [48]. For speed, a very simple contrast-stretching algorithm was used. This works by first finding the lightest and darkest pixel values within the image, P_{min} and P_{max}. The pixel values between these two values are then stretched to fill the range 0-255. The grey level transformation function is shown in Figure 6.2. Contrast enhancement improves the image in low light levels, as can be seen in Figure 6.3.

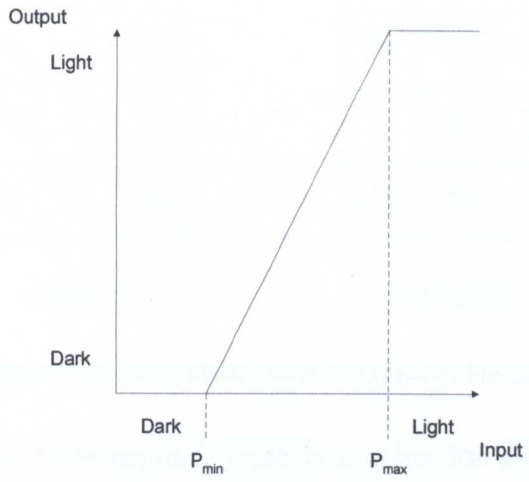


Figure 6.2: Grey Level Transformation Function.

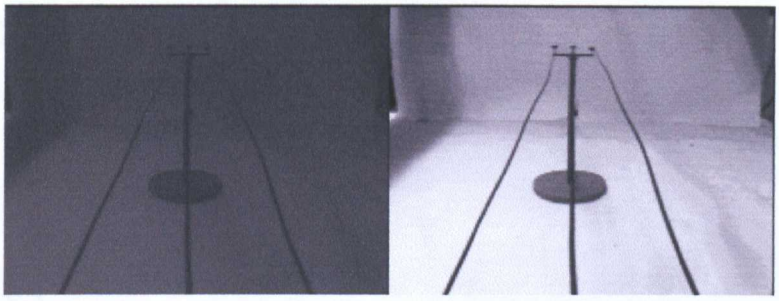


Figure 6.3: Image Before and After Contrast Enhancement.

6.3 Edge Detector

Before the Hough Transform is applied, an edge detection algorithm is used to pick out lines in the image: i.e. boundaries of rapid transition between light and dark. A variety of different algorithms for this purpose are described in the literature. These include the Roberts [48], Prewitt [48] and Sobel [54] mask edge detectors, the Laplacian of Gaussian edge detector [55] and the Canny detector [56]. Two of the most commonly used are the Sobel detector and the Canny detector. Although the Canny detector can produce better results, it is more computationally demanding so the Sobel detector was used.

The Sobel edge detector is a mask operator. The mask is applied to the image both horizontally and vertically, which finds both the horizontal and vertical components of a line. Figure 6.4 shows the masks used.

-¼	0	¼
-½	0	½
-¼	0	¼

Vertical

¼	½	¼
0	0	0
-¼	-½	-¼

Horizontal

Figure 6.4: Vertical and Horizontal Sobel Masks.

After the masks have been applied, there is a value for the horizontal, X, and vertical, Y, image gradient for each pixel. The vertical and horizontal gradients are then combined for each pixel to give the square of the gradient magnitude using (6.1) The angle perpendicular to the gradient at each pixel is calculated using (6.2), as this is needed for the transform stage.

$$G^2 = X^2 + Y^2 \quad (6.1)$$

$$\theta = \tan^{-1}\left(\frac{Y}{X}\right) \quad (6.2)$$

A sample result is shown in Figure 6.5. Figure 6.5a is the raw image and b shows the results of applying the Sobel masks. In order to pick out the stronger edges, a threshold, called the gradient threshold, is applied, which gives image c. Looking at image c, it can be seen that the edges vary in thickness. This is because, with very strong edges, the values of the gradient is higher than the threshold either side of the lines The lines can be reduced to 1 pixel wide using non-maximum suppression [57]. This is a thinning technique and it works by rejecting a pixel as an edge if it is next to an edge pixel with a higher gradient value, perpendicular to the edge direction. Figure 6.5d shows the final result. In this application we are not looking for horizontal lines, as the power lines should appear reasonably close to vertical in the image. Lines whose angle is within ±5° of the horizontal have been removed from image d.

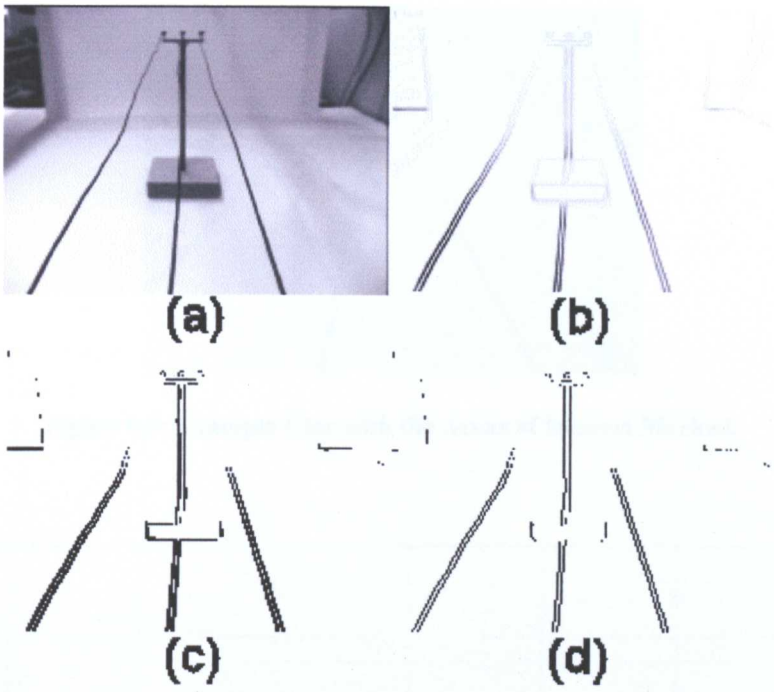


Figure 6.5: An Image at Different Stages of Edge Detection.

The optimum value for the gradient threshold needs to be found for the edge detector. This needs to be low enough to include the desired lines but high enough to reject background noise. The method used has been described previously by Golightly & Jones [8], for corner detector algorithms. The method works by taking a number of sample images and marking areas around the features of interest. A threshold value is determined for the number of pixels that are expected to make up each feature and then the actual number of pixels comprising the feature in each image is counted and compared to the threshold value. In this case eleven frames were selected from a sequence of images. A MATLAB program was written to count the number of pixels making up sections of the three lines in each frame. Figure 6.6 shows an example frame used; marked on this image are three boxes highlighting the sections of line that are of interest. The program counts the number of pixels making up the lines within these boxes.

The program plots the total number of pixels found for each threshold value, across all eleven frames. The ideal number of pixels for each line, based on finding two unbroken 1-pixel-wide lines, is also plotted. Figure 6.7 shows the results.

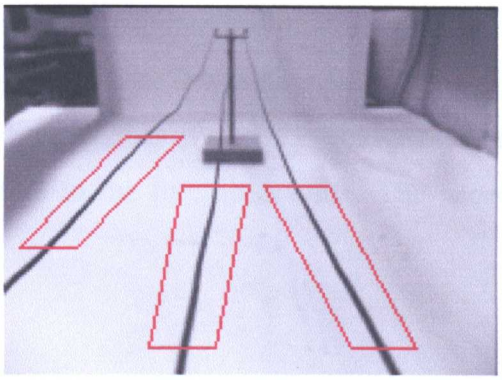


Figure 6.6: Example Line with the Areas of Interest Marked.

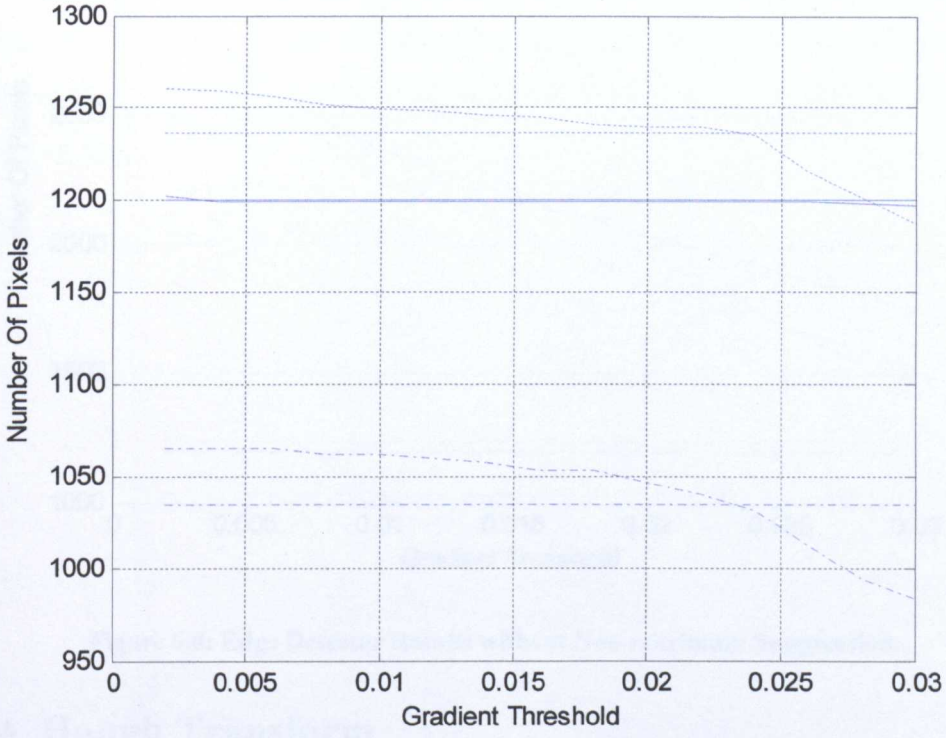


Figure 6.7: Edge Detector Results with Non-maximum Suppression.

Figure 6.7 shows the number of pixels actually found for the left (dash-dotted), centre (solid) and right (dotted) lines while the ideal number of pixels for each line is shown with horizontal lines in the same style. It can be seen that changing the gradient threshold does not have a large effect on the result: looking at the centre line, the number of pixels is nearly ideal over the full range of gradient threshold values. For the left and right lines there is some change and the crossover point is at around 0.023. This value will be used as the threshold.

The effect of non-maximum suppression may be assessed by running the same experiment on edge images where it was not used. Figure 6.8 shows the results. It can be seen that the target number of pixels is not reached, and that the number of pixels found is sensitive to the threshold value. The benefits of non-maximum suppression are clear.

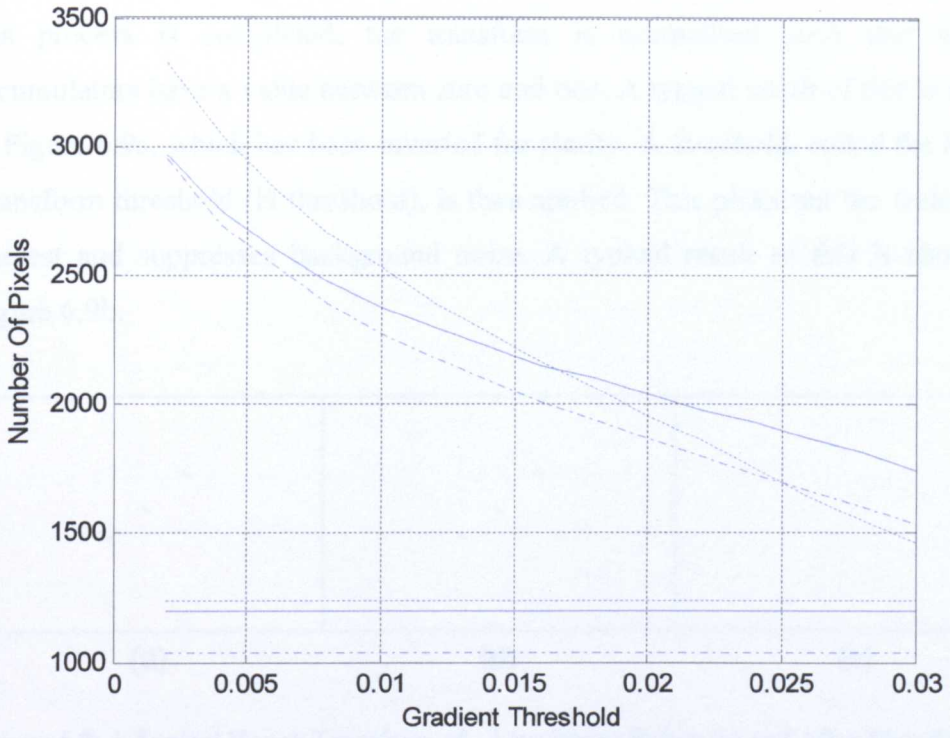


Figure 6.8: Edge Detector Results without Non-maximum Suppression.

6.4 Hough Transform

The Hough Transform is a well-known method for extracting lines that match parameterized functions from an image. The most common case is classifying straight lines in normal form according to their angle (θ) and distance from the image centre (ρ).

In order to create the transform, each line in the edge map must be classified by its angle and distance from the image centre. In order to do this, the edge map is stepped through. When a pixel, which is part of an edge, is discovered, its angle (θ), as previously calculated by the edge detector, is retrieved. The perpendicular distance from the image centre, ρ, can then be calculated using (6.3):

$$\rho = x\cos(\theta) + y\sin(\theta) \quad (6.3)$$

where:

x and y are the cartesian co-ordinates of the pixel in the image

The transform consists of a 2D array of accumulators addressed by ρ and θ. For each edge pixel in the edge map, the relevant accumulator is incremented. After this process is completed, the transform is normalised such that all the accumulators have a value between zero and one. A typical result of this is shown in Figure 6.9a, which has been inverted for clarity. A threshold, called the Hough Transform threshold (H threshold), is then applied. This picks out the features of interest and suppresses background noise. A typical result of this is shown in Figure 6.9b.

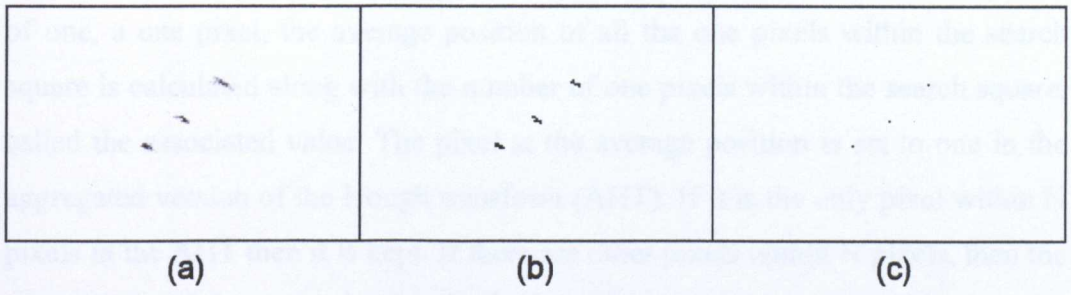


Figure 6.9: A Typical Hough Transform of a Line Image Before (a) and After Thresholding (b) and After Aggregation (c).

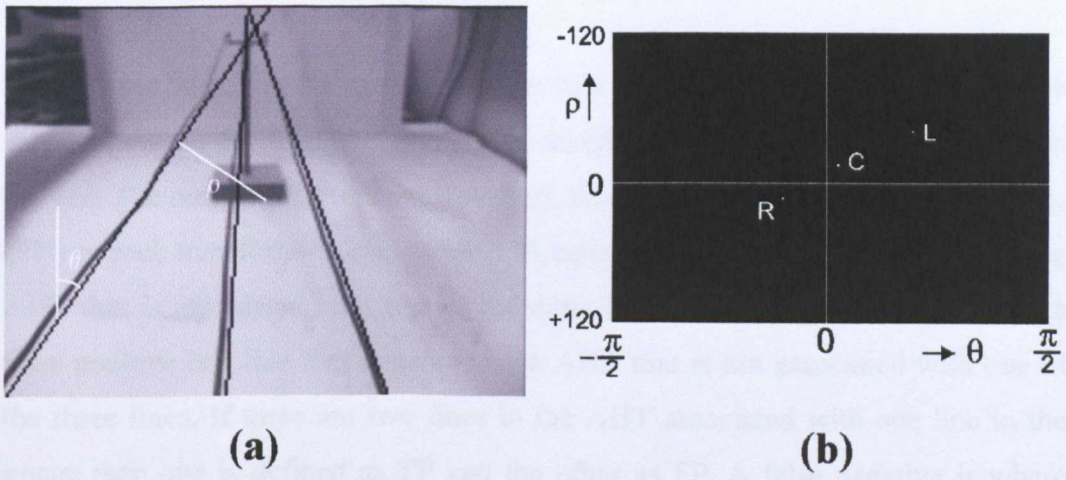


Figure 6.10: Typical Result showing: (a) the Three Overhead Lines Overlaid with the Straight Lines Generated by the Hough Transform and (b) the Corresponding Points in the Hough Transform Space.

This leaves clusters of points that need to be reduced to form single points with a typical result after reduction shown in Figure 6.9c. An example image and the resulting transform are shown in Figure 6.10.

The optimum threshold for the Hough transform needs to be found. In addition, there are alternative methods to reduce the clusters in the thresholded Hough transform to single points:

- An Aggregation algorithm described in [8].
- Non-maximum suppression, as used in the edge detector.

The aggregation algorithm works by stepping through the transform with a NxN search square. The search square is centred on each pixel in turn. If the pixel has a value of zero, a zero pixel, then no further action is taken. If the pixel has a value of one, a one pixel, the average position of all the one pixels within the search square is calculated along with the number of one pixels within the search square, called the associated value. The pixel at the average position is set to one in the aggregated version of the Hough transform (AHT). If it is the only pixel within N pixels in the AHT then it is kept. If there are other pixels within N pixels, then the one with the highest associated value is kept. This leaves a pixel whose position is at the centre of the cluster. The Non-maximum suppression works as in the edge detector, i.e. only points that are a local maximum are retained.

To find out which is the best to use for this application, both were tested with different values of threshold. In order to do this, a measure of detection quality is needed. The number of true positives (TP), false positives (FP) and false negatives (FN) in each transform can be counted. A true positive is a line that appears in the AHT that is associated with one of the three lines in the original image, while a false positive is a line that appears in the AHT that is not associated with one of the three lines. If there are two lines in the AHT associated with one line in the image then one is defined as TP and the other as FP. A false negative is where there is a line in the image that doesn't have an associated line in the AHT. The fraction of true positives (f_{TP}) is then defined in (6.4), while the detection success (DS) is defined as in (6.5).

$$f_{TP} = \frac{TP}{TP + FP} \quad (6.4)$$

$$DS = \frac{TP}{FN + 1} \quad (6.5)$$

As FN+TP≡3, DS can be defined as in (6.6):

$$DS = \frac{TP}{4 - TP} \quad (6.6)$$

In order to assess how well each method works, a measure is needed that includes both DS and f_{TP}. The detection quality (DQ) is defined as the product of the two, resulting in (6.7):

$$DQ = \frac{TP^2}{(4 - TP)(TP + FP)} \quad (6.7)$$

The DQ varies from zero to three, with zero representing a bad detection quality and 3 being good.

Figure 6.11 shows the result of how detection quality changes with H threshold, averaged for five frames. It can be seen that the aggregation method (solid) gives better results than non-maximum suppression (dotted), the best threshold value being 0.55. This is because the clusters contain a number of maxima and so the non-maximum suppression tends to give a number of points in the AHT to represent one line in the original image. Occasionally there are two points in the AHT for one line in the image with the aggregation method, although this is a much rarer occurrence than with non-maximum suppression.

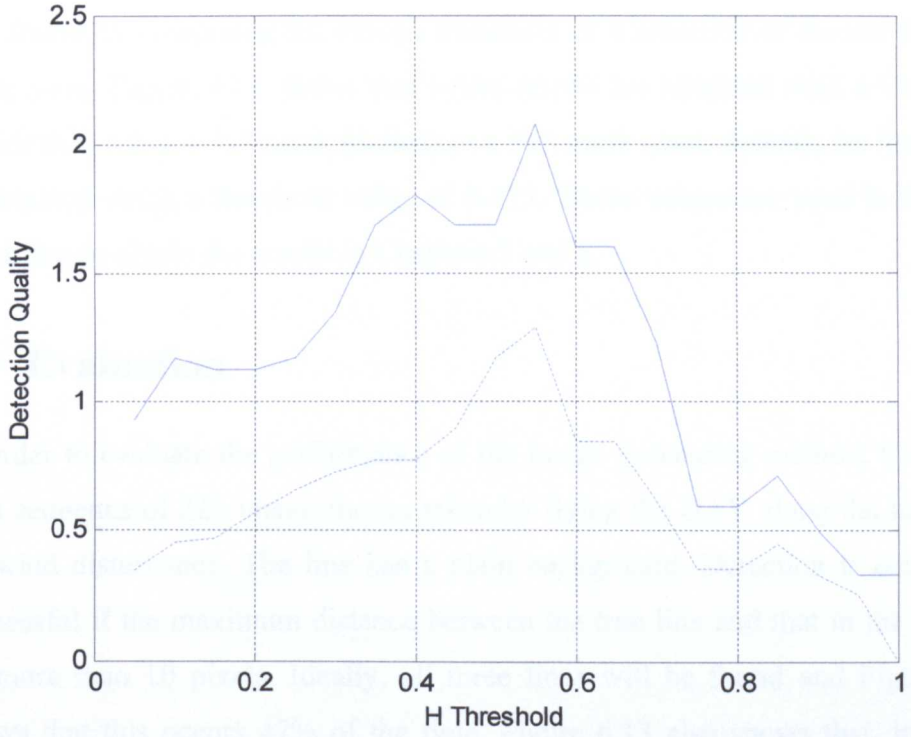


Figure 6.11: Detection Quality against Hough Transform Threshold for the Aggregation and Non-maximum Suppression Methods.

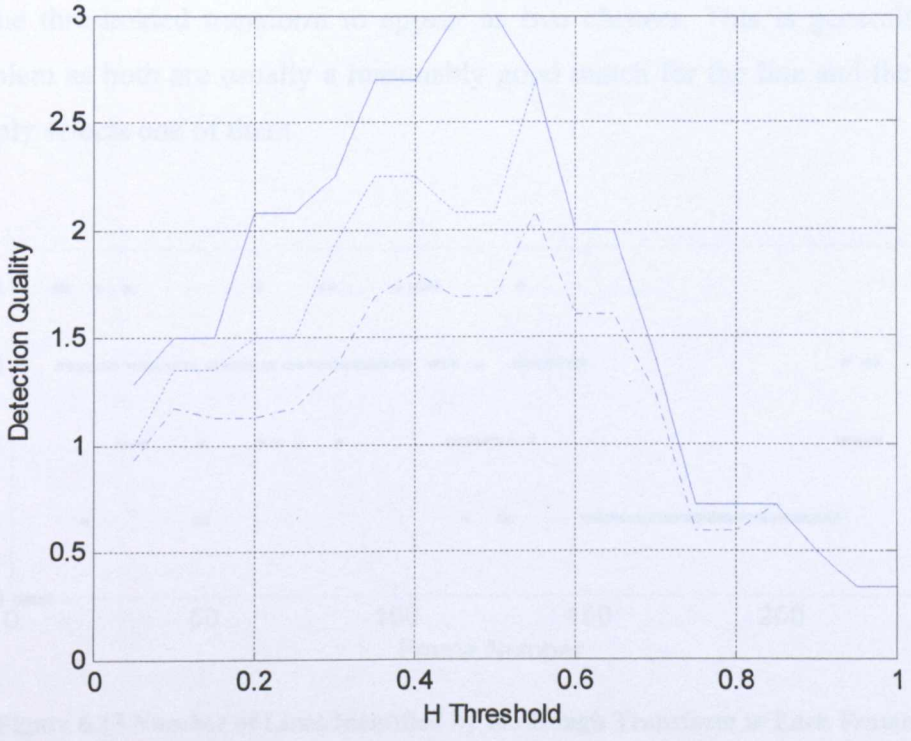


Figure 6.12: Detection Quality against Hough Transform Threshold for Different Mask Sizes.

The aggregation algorithm can be used with varying mask sizes. A suitable size was found by computing the Hough transform of a selection of frames for three mask sizes. Figure 6.12 shows that better results are obtained with a 9x9 mask (solid) than using a 7x7 mask (dotted) or a 5x5 mask (dash-dotted); the best result is obtained using a threshold value of 0.475. These values are used in the final algorithm to obtain the results in Chapters 7 and 8.

6.5 Evaluation

In order to evaluate the performance of the image processing method, it was run on a sequence of 225 image frames taken by flying the UAV along the line, with no wind disturbance. The line has a plain background. Detection is considered successful if the maximum distance between the true line and that in the AHT is no more than 10 pixels. Ideally, all three lines will be found and Figure 6.13 shows that this occurs 47% of the time. Figure 6.13 also shows that, in 7% of images, more than 3 lines are identified. This occurs when a line in the original image is mapped to more than one point in the AHT sufficiently far apart that they are not aggregated. This can occur if there is a kink in the line, causing the cluster in the thresholded transform to appear as two clusters. This is generally not a problem as both are usually a reasonably good match for the line and the tracker simply selects one of them.

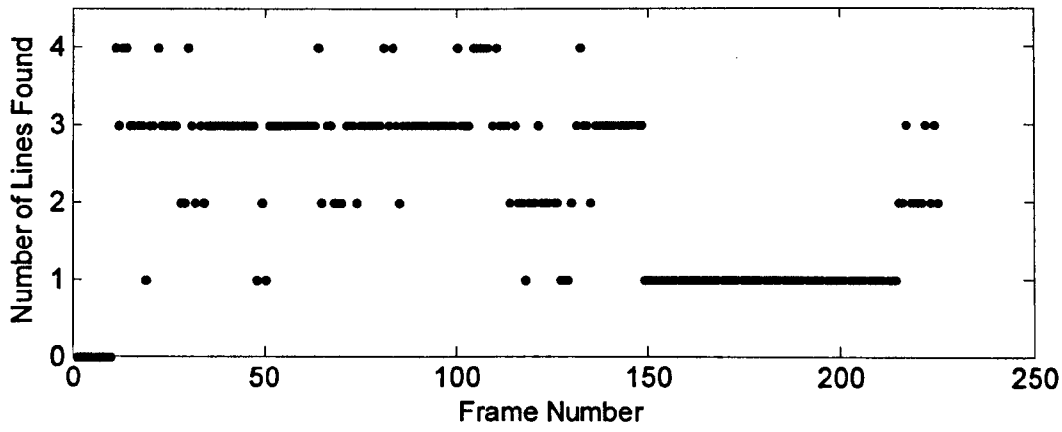


Figure 6.13 Number of Lines Identified by the Hough Transform in Each Frame of a Sequence.

Reliable tracking requires that at least two lines in the image are found because the position of the third line can then be estimated. This assumes that the lines are spaced an equal distance apart. This is very common although is not always the case. It would be possible to supply a priori information to the tracking system in cases where the conductors are not equally spaced. Finding at least two lines happens 60% of the time in Figure 6.13. If only one line is found, it is difficult to distinguish it from single lines generated by the background and the system becomes vulnerable to tracking false targets.

A significant problem occurs as the distant support pole is approached and becomes more prominent in the image. The Hough transform then tends to find one of the strong edges associated with the pole, rather than the overhead lines. This edge is an acceptable substitute for the centre line, with which it is nearly co-linear. Unfortunately, its relative strength causes the thresholding algorithm to suppress the Hough transforms of the two outer lines and tracking is adversely affected. This may be seen in Figure 6.13 between frames 150 and 210, where only one line is consistently identified. For the preceding section, before the distant pole becomes prominent, at least two lines are found 95% of the time, with all three lines being found 77% of the time.

6.6 Conclusions

It can be seen that this form of image processing yields the positions of the lines in the image with sufficient frequency to be tracked from frame to frame. It would be expected that similar results would be obtained with other image sequences with plain backgrounds. When cluttered backgrounds are present, the performance is likely to be less good. Background features will produce additional lines in the AHT. Very dark or very bright features in the background may affect the performance of the contrast enhancement. It may be necessary to adjust the values of parameters when used with realistic backgrounds although the structure of the image processing software shouldn't need to change in order to process cluttered backgrounds except when dealing with very cluttered scenes.

The tracking algorithm and estimation of the vehicle's position is discussed in Chapter 7.

Chapter 7 Tracking

7.1 Introduction

In order for the UAV to follow the lines, it is necessary to track the lines from frame to frame. This involves finding the lines in the current frame and then finding which ones correspond to the three lines found in the previous frame. According to Davison [41] there are four main tracking methods:

- Exhaustive search, where the entire image, or the transform space in this case, is searched for a match between the object being tracked in the previous frame and the current frame.
- Local search: this is similar to exhaustive search except that only the local area around the point at which the object would be expected to be found is searched.
- Kalman Filter [42]: this attempts to find a best estimate of the object's position by combining a prediction from the previous frames with the measurement of the object's position in the current frame. All the errors are assumed to have a Gaussian distribution and are used form a weighting factor to combine the prediction and measurement.
- Particle Filter [43]: this also uses errors to produce position estimates. However, unlike the Kalman filter, the errors are not assumed to be Gaussian. Instead the error function is represented by a number of particles, which allows more complex error functions to be represented, including multi-modal functions, allowing multiple-hypothesis testing.

The most basic form, exhaustive search, is likely to produce many tracking errors; this is because the pattern we are searching for is relatively simple (three dots in the transform space) and so could be lost in the background noise. In addition the exhaustive search is more computationally demanding than a local search tracker because all of the transform must be searched at every sample. For these reasons, a local search tracker was used initially.

This chapter discusses the development of the tracking software. This starts with a local search tracker and the development of a model-based acquisition routine, to find the lines in the early frames in order to initialise the tracking. Section 7.4 considers the use of fuzzy logic, to detect whether the tracker erroneously thinks a sideline is the centre line or whether the lines have been lost. Finally, the inclusion of a Kalman filter is described. This chapter concentrates on obtaining the lateral displacement of the UAV relative to the lines, although some results for tracking height are presented. Tracking of roll, yaw and lateral displacement using two cameras is described in Chapter 8.

7.2 Early Tracker

7.2.1 Description of the Early Tracker

A relatively simple tracker was implemented to start with, giving a broad assessment of tracking performance as well as a basis on which to build a more advanced tracker.

Lateral displacement of the rotorcraft with respect to the lines leads to the pattern shown in Figure 7.1 where the 3 points are seen to slide along a straight-line locus in the Hough transform space, as predicted in chapter 4.

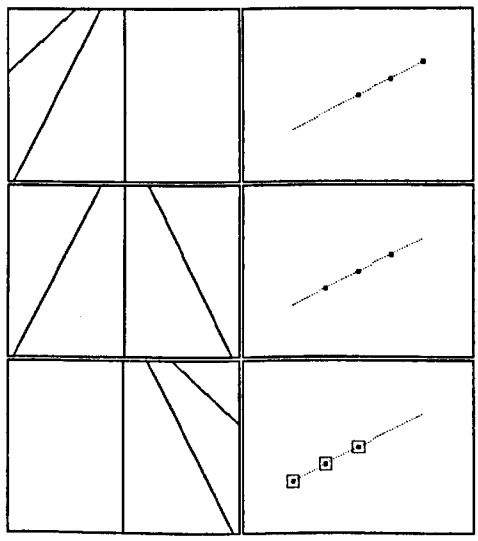


Figure 7.1: Three Examples of Image Line Patterns (left) and their Corresponding Hough Transform Points (right).

Assuming that yaw, roll, pitch and height remain constant, the displacement relative to the centre line is given by (7.1). Chapter 4 showed that either the θ or ρ value can be used to obtain the lateral displacement so equation (7.1) uses an average of the two.

$$X = \frac{X_\theta \theta + X_\rho \rho}{2} \quad (7.1)$$

where X_θ and X_ρ are $\frac{\partial X}{\partial \theta}$ and $\frac{\partial X}{\partial \rho}$.

The θ and ρ scaling factors are obtained from the analysis in section 4.2.2. Substituting these values into (7.1) gives:

$$X = \frac{0.048\theta - 0.0433\rho}{2} \quad (7.2)$$

It is reasonable to assume that yaw pitch and height will generally remain within tight limits when the rotorcraft is flying along a line section. In practice the UAV will roll when it moves from side to side. However, because roll and lateral displacement independently affect the line positions in the image (as shown in section 4.2.2) lateral displacement can be considered independently.

Once the line co-ordinates have been extracted, it is necessary to track them from frame to frame. The tracker is initialised by taking off-line measurements of the positions of the lines in the early frames of each image sequence. The averages of these measurements were used as the start points. Clearly, this is not a practical method of initialising tracker on an actual UAV; an automated method of initialisation is described in section 7.3. Once tracking has started, it is necessary to search the local area around the predicted position of each line in the transform space. Initially, the search area is chosen to be symmetric in θ and ρ: this gives a choice between either a circular or square shaped search area. In order to compare the two, a square and a circle, with a diameter equal to the side of the square, were generated in the transform space. Each point within the square and circle represent a line in the image space. All the lines associated with these points were drawn in

the image space. Figure 7.2 shows the two HT search areas, on the left, and all the lines in the corresponding image shown on the right in white. The red lines show the limits for the square search area, for comparison. Both search areas are seen to produce a similar set of lines in the image space. Squares are easier to represent, computationally, and so square search areas, hereinafter called search-squares, will be used.

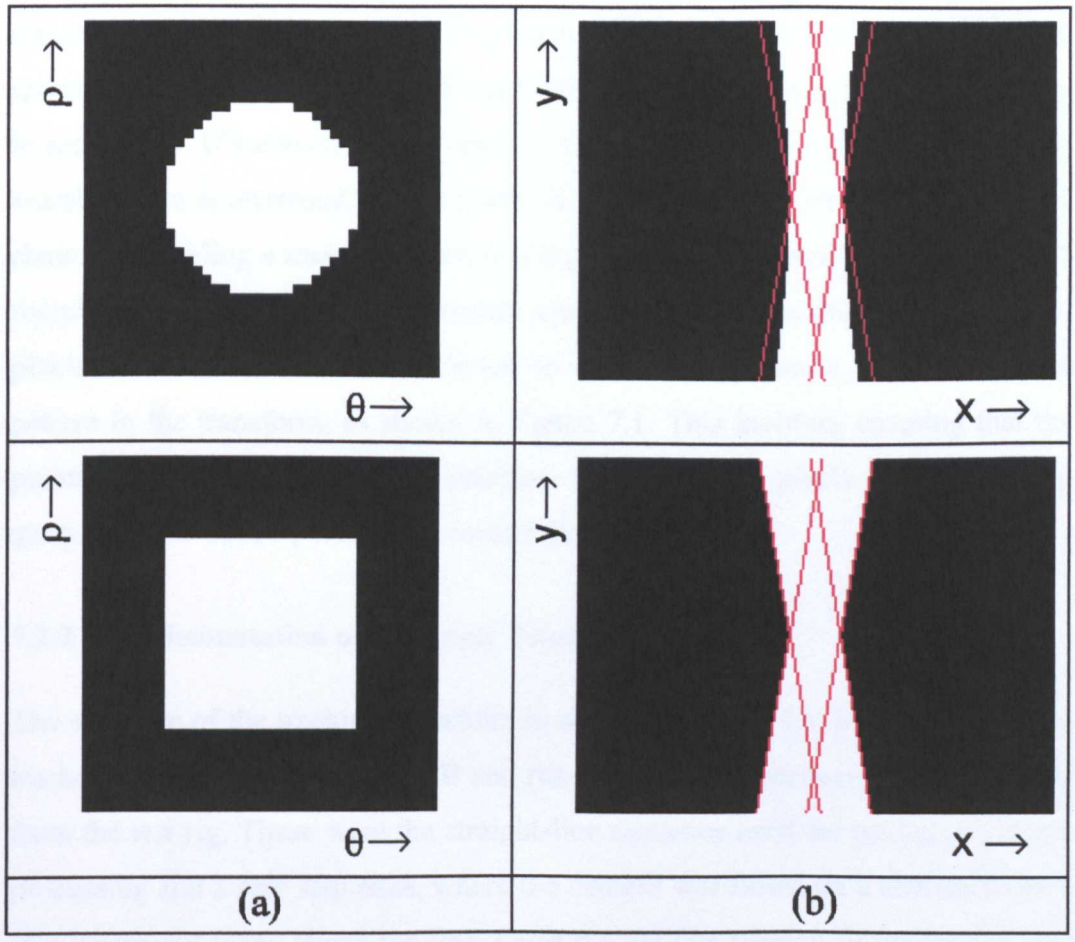


Figure 7.2: Square and Circle Search Spaces in the HT with Corresponding Lines in the Image Space.

Search-squares, centred on the predictions from the previous frame, are placed on the transform of the current frame and searched for matches to the three lines. When flying straight above the lines, each frame should appear substantially the same as the previous frame, although there is in fact a small change due to the catenary shape of the lines. With the assumption that the pattern in Hough transform space is invariant with position along the line, the current line positions can then be used as reasonable predictions for the next frame. For most overhead

lines, the pattern should be symmetrical. However, there are lines that are not constructed symmetrically, but this information is known and can be accounted for in the tracker algorithm. In some cases, one or more lines are not found. Merging into the background, glare or occlusion can cause this. If only one line is missing, then its position can be predicted from the other two. If fewer than two lines are found, then the previous predictions are simply carried forward and used as the next prediction. Initially the size of the search-squares is set to the minimum size, in this case it is 21 pixels (10 pixels either side) in a transform space of 181x120 pixels; a more formal choice of search square size is undertaken in section 7.4. If a corresponding line is not found in a given frame, the size of the search square is increased by two pixels in the next frame in order to increase the chances of finding a match, up to a maximum size of 41 pixels. When a match is found in a frame the size of the search square is reset to the minimum size of 21 pixels. In addition rules are included to ensure conformance to the three-line pattern in the transform, as shown in Figure 7.1. This involves ensuring that the points corresponding to the two sidelines are at least 10 pixels in both θ and ρ away from the centre pixel in the correct direction.

7.2.2 Implementation of the Early Tracker

The structure of the tracking algorithm is shown in Figure 7.3. In order to test the tracker, it was coded in MATLAB and run off-line on sequences of images taken from the test rig. These were the straight-line sequence used for testing the image processing and a sine sequence, where the camera was flown on a sinusoidal path in a horizontal plane above the line. Once the off-line version of the tracker was working, it was ported into C++ and incorporated into the test rig control software.

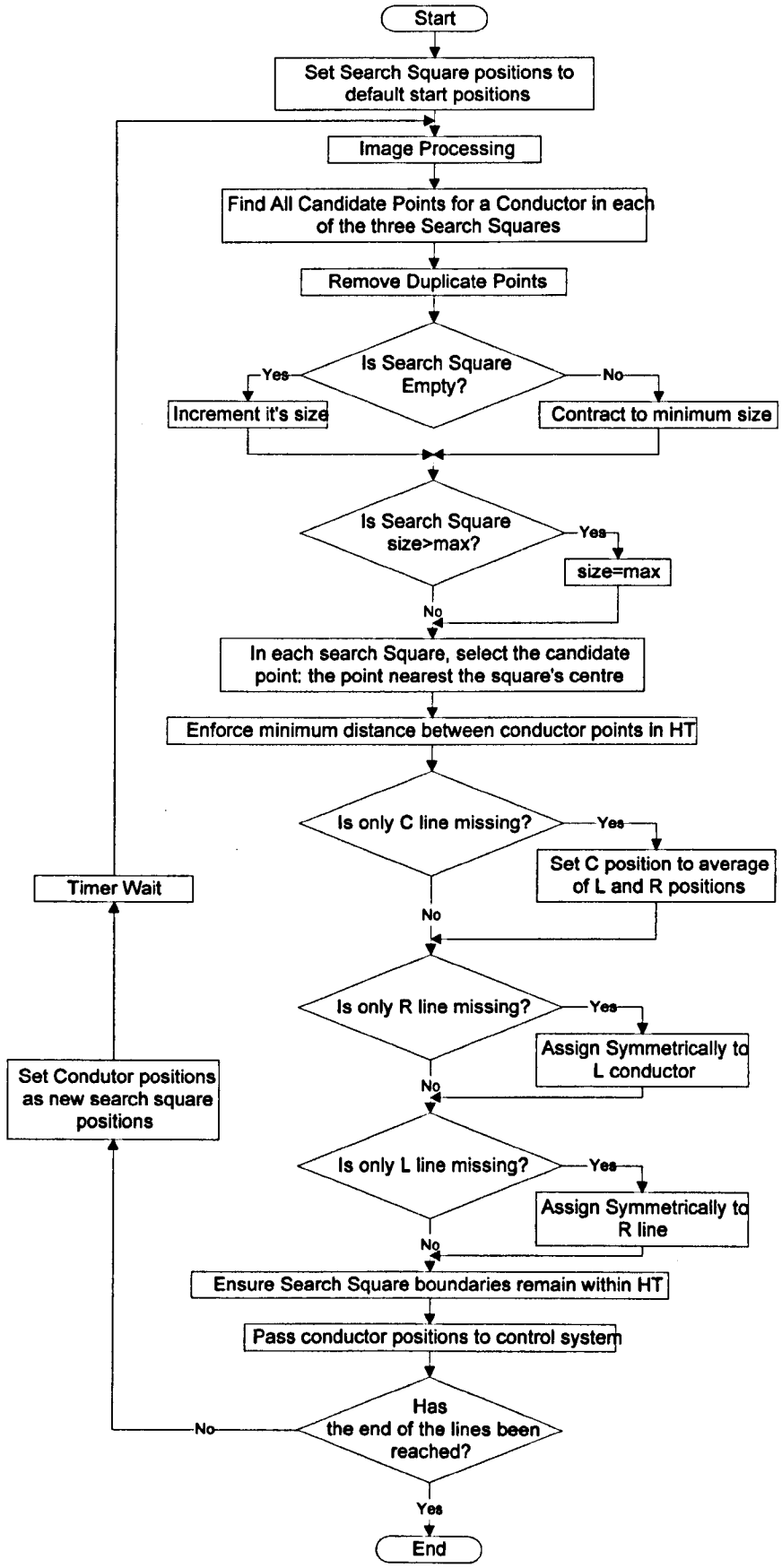


Figure 7.3: Flowchart showing the Operation of the Early Tracker.

The aim of the image processing and tracking is to provide the local position feedback signal for the UAV when it is close to the overhead lines. In order to bring the UAV into the vicinity of the lines, or if the visual tracker loses the lines, Differential Global Positioning System (DGPS) can be used to provide the position signal. The model of the system is shown in Figure 7.4, which is an extended version of Figure 3.9 that includes the pitch rate feedback described in section 3.4. The rotorcraft model and pitch rate compensator run at a sample rate of 25Hz, while the vision processing runs at 10Hz and the DGPS runs at 1Hz. The vision processing and DGPS together form the position feedback loop for the UAV. The model includes a switching condition to select between vision and DGPS feedback. The system uses the DGPS to bring the UAV to the vicinity of the lines and then switches to using vision when the lines are acquired. Currently the system is set to switch back to DGPS if the UAV strays more than 2m from the line, as measured from the test-rig. On an actual inspection UAV this envelope limit measured from the DGPS and would likely be set further from the lines, in order to stop DGPS errors erroneously switching the system away from vision feedback. When the UAV comes back within 2m of the line, the system attempts to re-acquire the lines. The system will also switch to using DGPS in the event of the tracker losing lock on the lines.

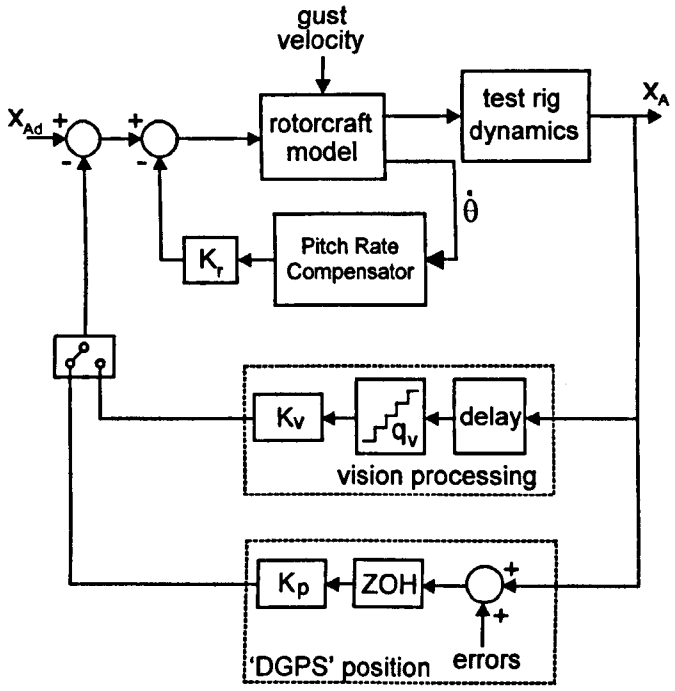


Figure 7.4: High Level System Model.

It was necessary to model the DGPS position errors in order to show the effect of DGPS feedback. Position fixes from DGPS contain random correlated errors. Figure 7.5 shows a sequence of East-West DGPS errors obtained by Earp [7] using a Trimble Pro XR receiver at a fixed reference point – the error distribution and power spectral density are also to be found in [7].

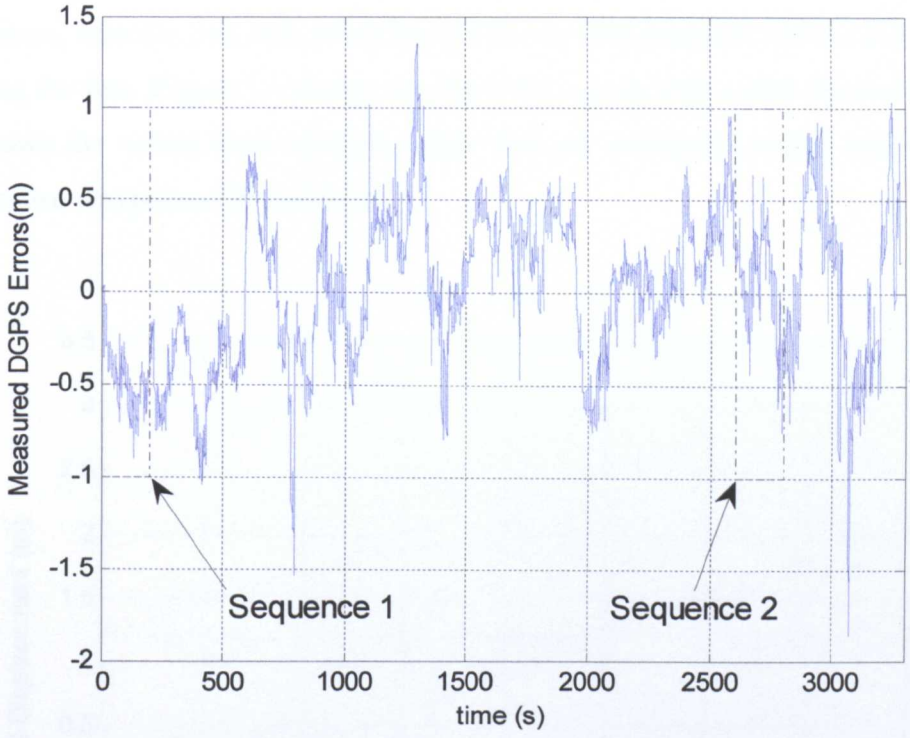


Figure 7.5: Measured DGPS Errors at 1Hz Sampling Rate showing the Two Sections of Data Used in the Test.

In order to simulate DGPS, two sequences of errors were selected from the error set shown in Figure 7.5. Sequence 1 has a distinct bias offset, while sequence 2 has no bias. These error sequences are added into measurements from the test rig to produce simulated DGPS position fixes. Using two error sequences allows testing of the effect of different errors.

7.2.3 Results with Early Tracker

7.2.3.1 Still-air Responses

In order to use DGPS to guide the UAV along the lines, a number of waypoints would be set; these would almost always be at pole locations. In practice

electricity companies only know the locations of the poles to within a few metres. If DGPS were used to guide the UAV, it would blindly follow the recorded pole locations. Using vision feedback means that the UAV would follow the actual line, provided that the initialisation is correct, i.e. that the UAV is sufficiently near to the lines for visual acquisition to occur.

In the first test, the second pole on the test rig was offset from its recorded position, with the first pole being kept at its recorded position, and the UAV flown along the line. Figure 7.6 shows that the UAV's path with vision feedback (solid) follows the actual lines (dotted) rather than an erroneous course based on the recorded waypoints (dash-dotted).

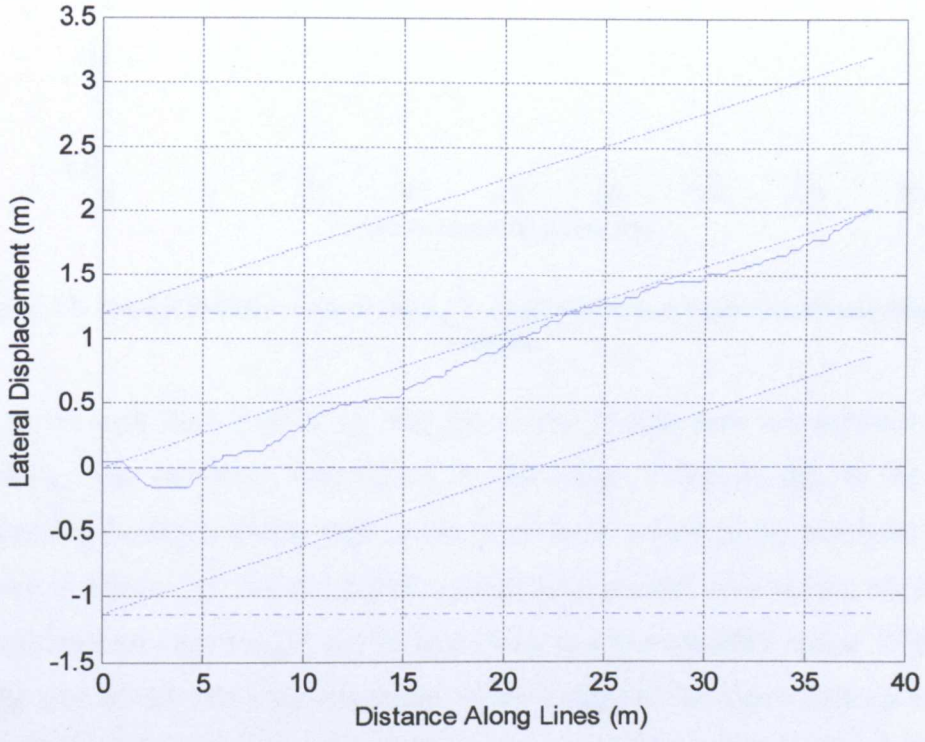


Figure 7.6: Lateral Displacement of the UAV using Vision Feedback with an Offset Pole.

In addition to the waypoint errors, DGPS has errors in the fixes it gives, as discussed in subsection 7.2.2. To quantify their effect, the UAV is flown along the lines with lateral position feedback from the DGPS loop alone. The path of the UAV, using the two different DGPS error sets (dashed, sequence 1; dash-dotted, sequence 2) are shown in Figure 7.7, which shows that the varying errors in the DGPS position fixes cause significant perturbations of the flight path. The UAV

was then flown along the lines with vision providing the position feedback (solid). This flight path tracks the centre line more closely than the DGPS flight paths.

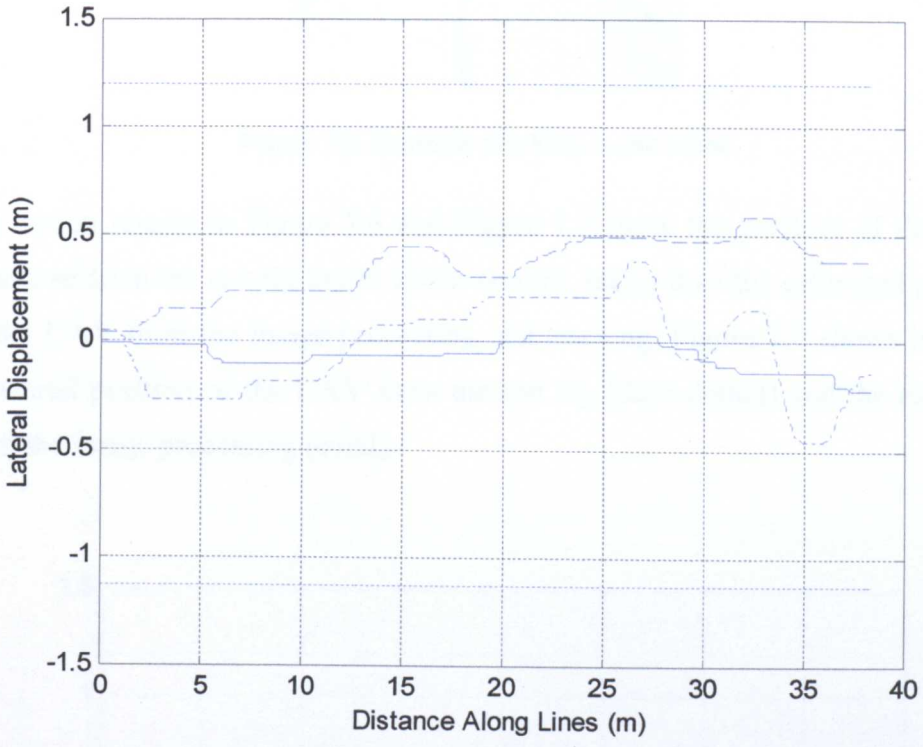


Figure 7.7: Lateral Displacement of the UAV using DGPS and Vision Feedback; Zero Wind Gusting.

It can be seen from Figure 7.7 that the vision system does not produce perfect tracking. The deviation that occurs in the range 5-20m is due to the image processing finding a strong edge in the direction of a kink in the overhead line, as shown in Figure 7.8, which is then tracked for a period. This occurs because the model lines are lightweight and so don't keep to a true catenary shape. With a line in the real-world, this does not occur, as the weight of the line scales up far more than its diameter relative to our rig lines; this causes the weight of the line to keep it in catenary shape. The second feature of the response is a small offset, which forms at about 30m along the line and persists to the end of the run. This is caused by the detection and tracking of a strong edge on the upcoming support pole, whose width becomes significant as it is approached. The camera therefore tracks slightly to one side of the centre line. These results shows that we should get better tracking using vision, rather than relying on DGPS alone; DGPS is essential to bring the UAV into the vicinity of the lines, however.

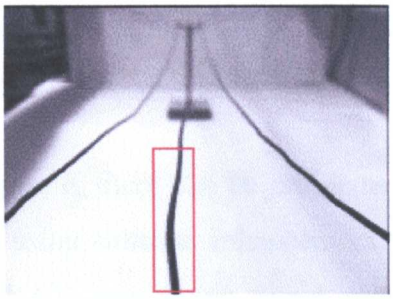


Figure 7.8: Example of a Kink in the Lines.

The vision results in Figure 7.6 and Figure 7.7 show the position of the UAV measured from the test rig under vision control, rather than the estimated position of the UAV from the image processing and tracking. Figure 7.9 shows both the measured position of the UAV from the test rig (dash-dotted) and the estimates from the image processing (solid).

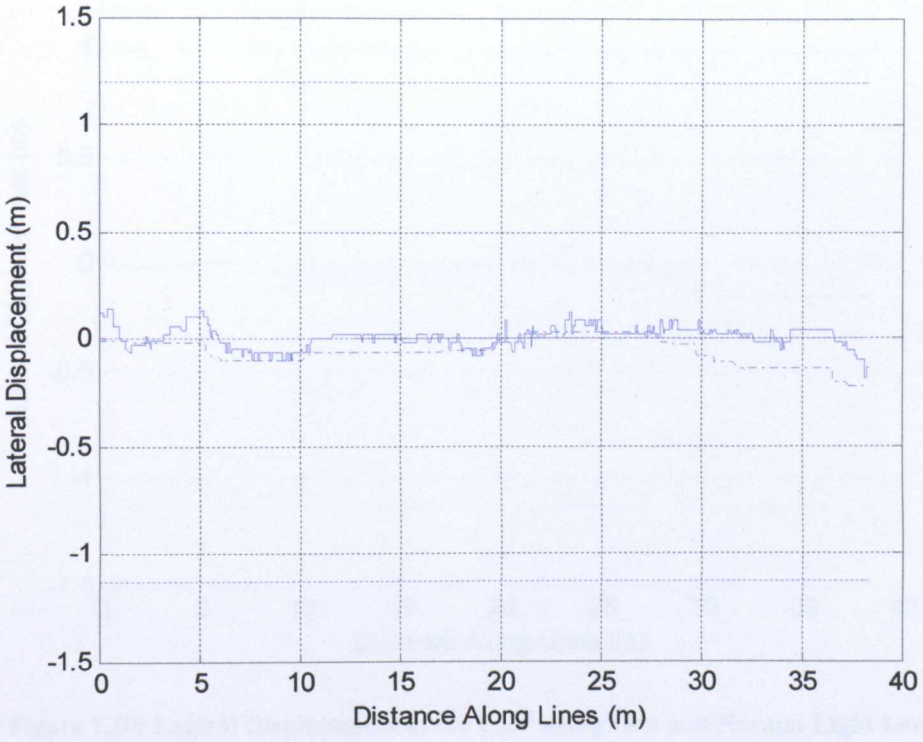


Figure 7.9: Measured UAV Position and Estimated UAV Position from the Image Processing.

Figure 7.9 shows that the estimated position from the image processing matches well with the measured UAV position, although it is noisier. This would be expected as the line positions in the image do change slightly from frame to

frame. Due to the slow response time of the UAV, this noise is smoothed out and so doesn't appear on the measured UAV position graph.

As was discussed in Chapter 6, there can be problems associated with low light levels. This is countered using contrast enhancement. In order to test how this affects the tracker, the UAV was flown along the lines in low light level conditions. Figure 7.10 shows that the tracker works as well in low light levels (solid) as in normal light levels (dash-dotted): Normal lighting levels refers to the test rig being illuminated by the overhead laboratory fluorescent lights plus a small amount of varying daylight and low light refers to the small amount of daylight entering the laboratory with the window blind closed.

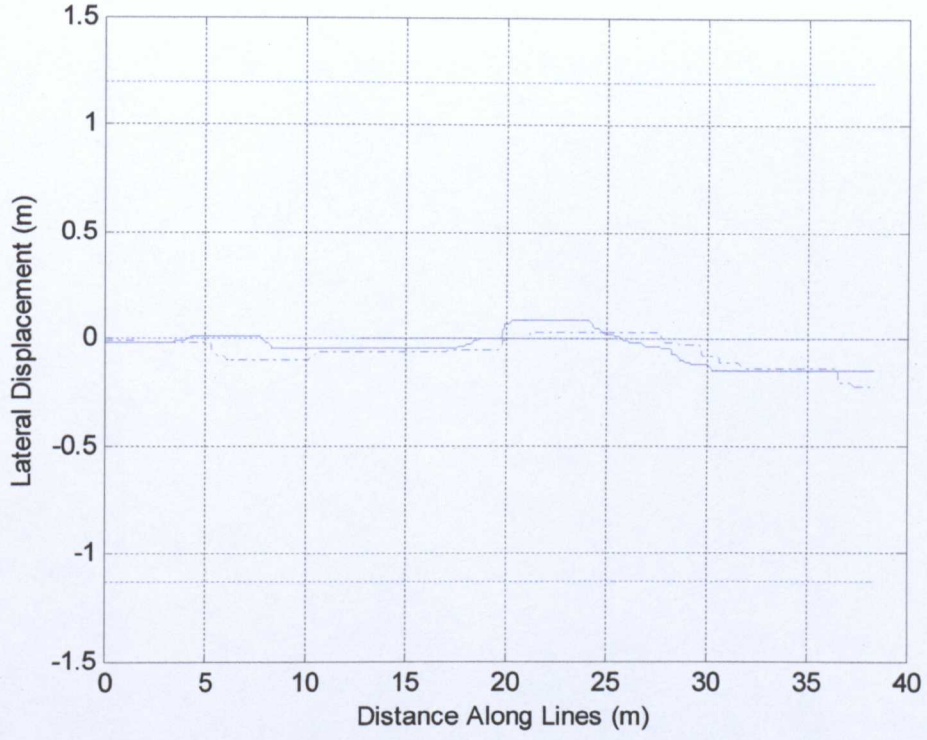


Figure 7.10: Lateral Displacement of the UAV using Low and Normal Light Levels.

7.2.3.2 Wind gust Responses

The response of the vision feedback system to a simulated wind gust was investigated. In order to test this, a 3s pulse of wind of $1ms^{-1}$ velocity, which represents a significant wind gust for this size of UAV, was input between approximately 3 and 5m along the line. As the UAV model being used in

simulation is for the lab demonstrator, rather than the final UAV, which is likely to be around 5x bigger, it is more affected by wind. How the wind gust scales with the size of the UAV is currently unknown, although if it scales linearly, the $1ms^{-1}$ wind gust would be equivalent to around 11mph for a full size UAV. The modelling of a full size UAV and the effect of wind gusts on it will need to be done later in the project.

It should be noted that the wind gust was simulated on the control PC and was input into the dynamic model simulated on that PC via the gust disturbance input of the model, as discussed in section 3.4; the position of the UAV is then relayed to the test-rig. Figure 7.11 shows two results, for a positive and negative wind gust (solid lines).

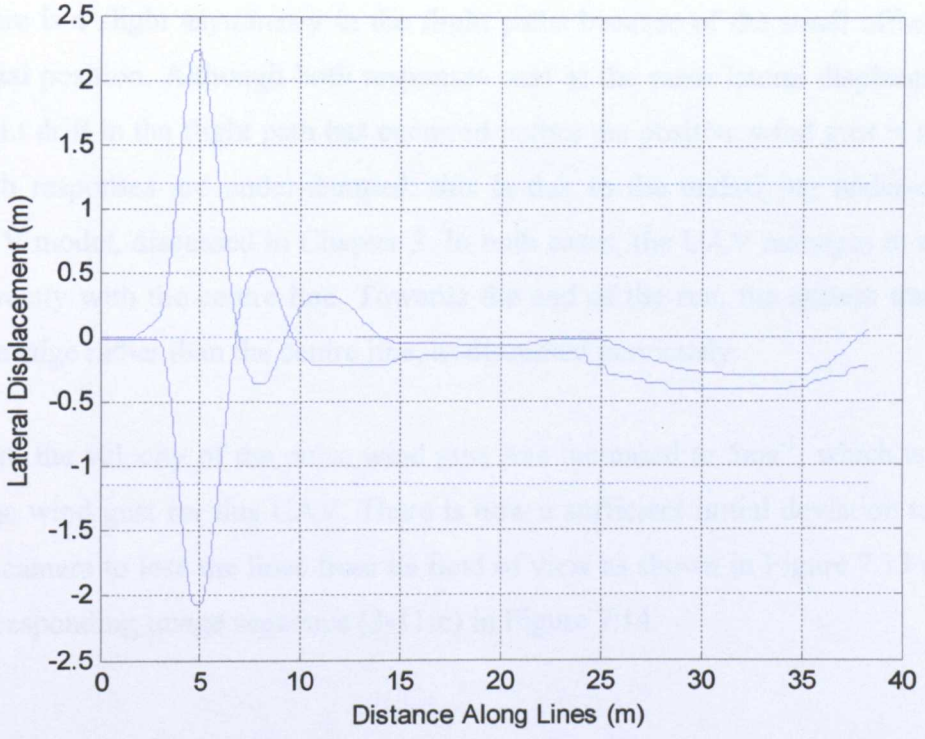


Figure 7.11: Lateral Displacement of the UAV in Response to a Pulse Wind Gust.

Figure 7.12 shows samples of the images recorded, during the run with the positive wind gust, sampled at equal intervals in the range 3-10m along the lines.

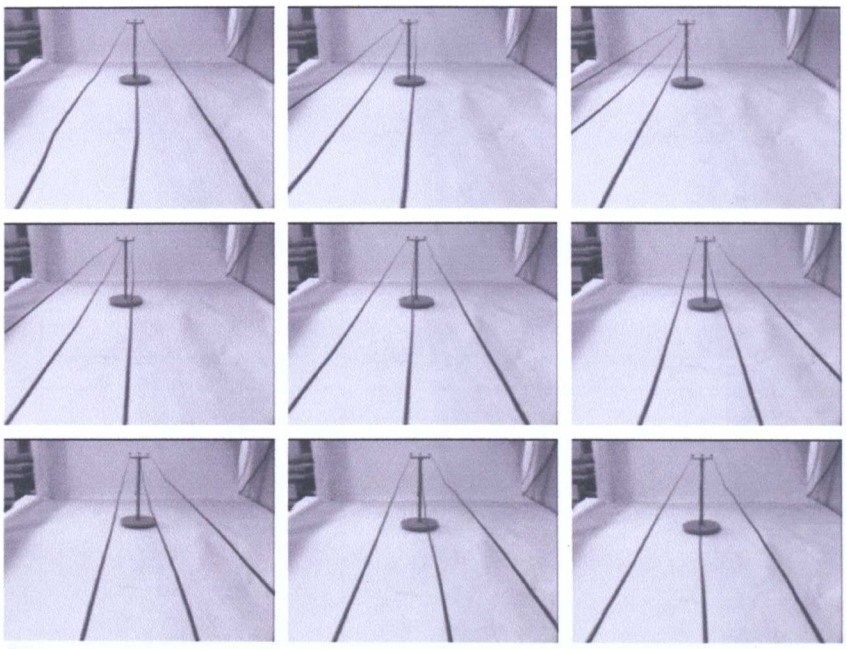


Figure 7.12: Sequence of Camera Views taken during the Run with Positive Wind Gust.

There is a slight asymmetry in the flight paths because of the small offset in the initial position. Although both responses start at the same lateral displacement, a slight drift in the flight path has occurred before the positive wind gust is applied. Both responses are under-damped; this is due to the underlying under-damped UAV model, discussed in Chapter 3. In both cases, the UAV manages to re-align correctly with the centre line. Towards the end of the run, the system tracks the pole edge rather than the centre line, as discussed previously.

Next, the velocity of the pulse wind gust was increased to $5ms^{-1}$, which is a very large wind gust for this UAV. There is now a sufficient initial deviation to cause the camera to lose the lines from its field of view as shown in Figure 7.13 and the corresponding image sequence (3-11m) in Figure 7.14.

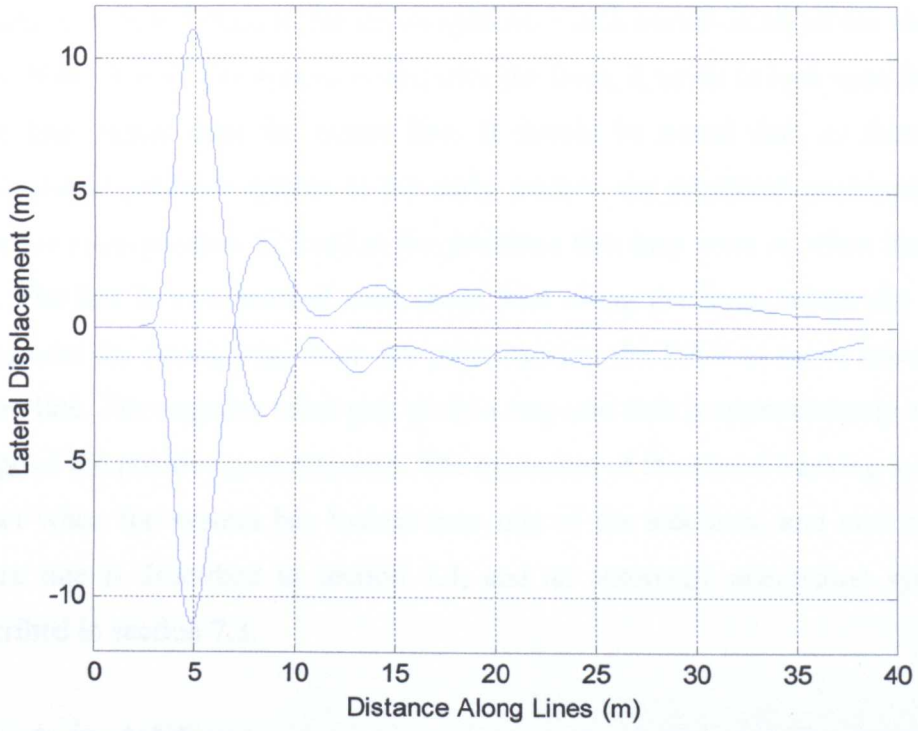


Figure 7.13: Lateral Displacement of the UAV in Response to a Large Pulse Wind Gust.

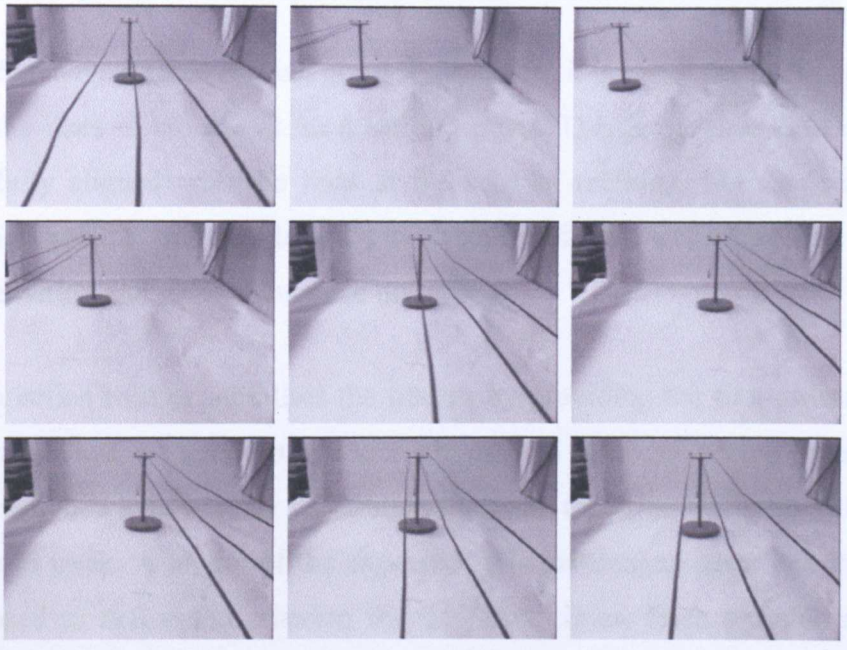


Figure 7.14: Sequence of Camera Views taken during the Run with Large Positive Wind Gust.

The initial deviation is now about 12m from the centre line, which removes all except the furthest portion of the span from the image. At 2m lateral displacement, the position feedback is switched to the DGPS loop, which returns

the camera back to the vicinity of the lines. As the 2m threshold is crossed again, control is switched back to the vision system, which occurs at about 6m along the lines. When the vision system re-acquires the lines, it tends to lock onto the right hand line, rather than the centre line. It should be noted that, as there is no automatic acquisition system in the early tracker, the predicted positions of the lines for re-acquisition is fixed at the positions that they were at when they were lost. The line is then tracked until about 30m along the lines, where the system locks onto the strong edge from the pole, causing the UAV to move towards the centre line. The negative wind gust gives a response that is approximately a mirror image of the positive gust response. The extension of the visual tracking system to detect when the system has locked onto one of the sidelines, and switch to the centre line is described in section 7.4, and an automatic acquisition system is described in section 7.3.

7.3 Acquisition

7.3.1 Description of the Acquisition Routine

Rather than have a fixed starting point for tracking, it is better for the system itself to find the lines in the image for a starting point. This is because the UAV may not be fully aligned with the lines at the start of tracking. For this reason, an acquisition routine was developed. First, the principle is described and then the implementation and performance are discussed.

The acquisition routine initialises the tracker by providing the first estimates of θ and ρ for each of the three lines, which are used to place the search-squares. The AHT is searched exhaustively for straight lines and the "best" lines are selected as the lines to track. A model of the expected AHT pattern, as described in section 6.4, is used to define the criterion for the "best" lines. Each possible match is given a score depending on how well it fits the model and the best of these is chosen. At least two of the three lines need to be found for a match, with three lines being given preference over two.

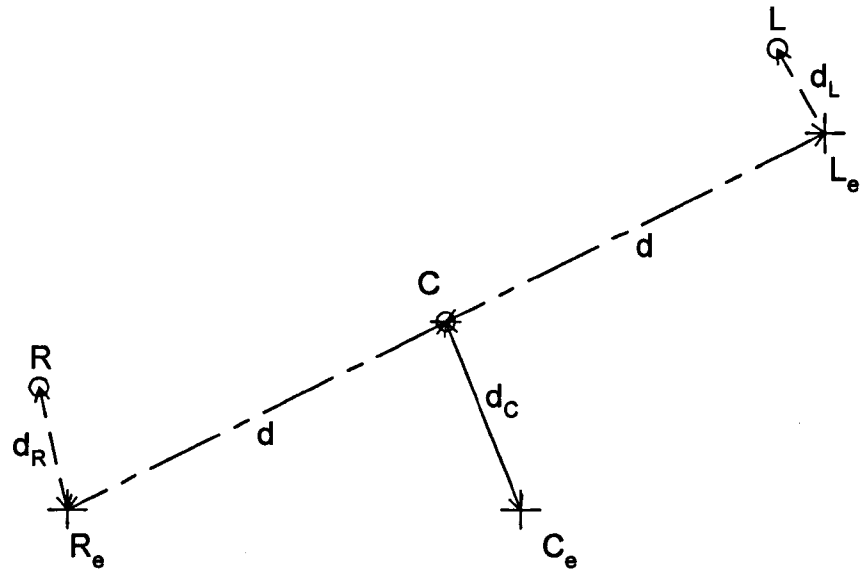


Figure 7.15: Points in AHT and the Distances between them; Crosses Represent the Positions where the Three Points are Expected to be, while the Circles Represent Examples of Actual Points Found.

Ideally the point representing the centre line should appear at the centre of the Aggregated Hough transform (AHT), point C_e in Figure 7.15. In practice the centre line, C, will be found a short distance, d_C, from C_e. Ideally we would then expect to find the left and right line at points L_e and R_e, a distance d from C. Again, in practice these will be found a short distance from L_e and R_e at L and R; these are distances d_L and d_R from C respectively. For any set of lines (R, C and L) we can define an improbability measure, P, as in (7.3):

$$P = S\left(1 + f_S\left(d_L^2 + d_R^2\right) + f_C d_C^2\right) \quad (7.3)$$

where:

f_S is the sideline factor

f_C is the centre factor

S is the symmetry measure

This measure incorporates three important factors about the best match model: the distance from the AHT centre, how far the sidelines deviate from their expected positions and how symmetrical the lines are. The best match is the most symmetrical with the sidelines close to their expected position. In addition it should be close to the centre of the AHT, although this is less critical, as reflected

in the value of f_C in (7.3). The candidate set of lines which minimises (7.3) is selected.

The values of the sideline and centre line factors were determined experimentally and f_C=0.0001 and f_S=0.002 were found to give the best results. In the case where all three lines are found the symmetry measure is given by (7.4)

$$S = (R - C + L - C)^2 \quad (7.4)$$

If either the right or left line is not found, its position has to be estimated from the other two. In these cases, (7.4) fails and an alternative measure for S, based on the predicted distance, d is used. This measure is also designed to give higher values of S than (7.4) in order to give preference to line sets where all three lines are found. In the cases where only two lines are found, S is defined as in (7.5) when the left line is missing, or (7.6) when the right line is missing.

$$S = (abs(d) + abs(R - C))^2 \quad (7.5)$$

$$S = (abs(d) + abs(L - C))^2 \quad (7.6)$$

7.3.2 Implementation of the Acquisition Routine

The acquisition routine needs to check every point in the AHT to see if it is a possible candidate for the centre line by scoring it according to (7.3). Figure 7.16 shows the operation of the acquisition routine. In order to test it, it was coded in MATLAB and run on the same test sequences used to test the early tracker. After testing the acquisition routine, it was ported to C++ and incorporated into the real-time vision software; the code is shown in Appendix D.

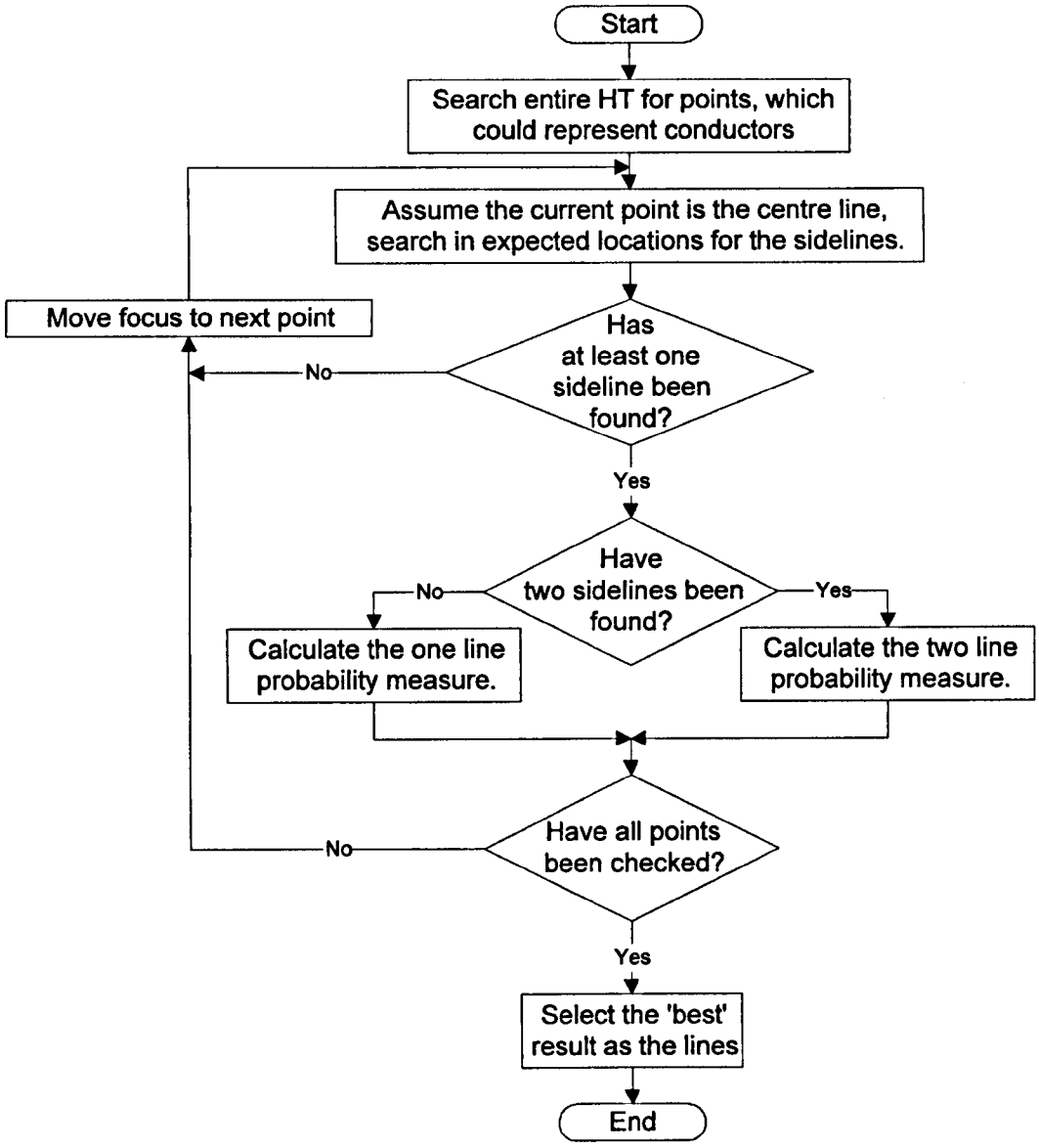


Figure 7.16: Flowchart showing the Operation of the Acquisition Routine.

7.3.3 Results with the Acquisition Routine

Ideally, the acquisition routine should find the three lines from the AHT, as shown in Figure 7.17. However, this is not always the case and we can define six different types of result:

- Found All Correct: where all three lines are found and match well to the actual lines.
- Found 2 Correct: where two of the three lines are found and match well to the actual lines, with the third line predicted from the other two, an example of

which is shown in Figure 7.18. Here the white lines represent the result from the transform and the dotted line represents the predicted third line.

- Found Sideline: where two of the three lines are found and match well to actual lines, but the third line is predicted such that the estimated centre line erroneously corresponds to one of the sidelines, rather than the centre line; an example is shown in Figure 7.19. Here the white lines represent the result from the transform, the dotted line represents the predicted third line, while the black line corresponds to a line that appears in the AHT but that isn't in the set from the acquisition output. In this case it is the other sideline.
- Found Pole: where the centre line in the result corresponds with a pole edge rather than the centre line in the image, an example is shown in Figure 7.20; the line styles are as before.
- Not Found: where no result is found by the acquisition routine, an example is shown in Figure 7.21, where the lines found by the AHT are shown in black, but they don't fit the line model sufficiently for a match.
- Found Incorrect: where a result is found but it does not correspond to the actual lines. This would typically lock onto straight-line sections produced by the insulators on a pole top; an example is shown in Figure 7.22. As with previous examples, the white and dotted lines represent the result and black lines are other lines in the AHT.

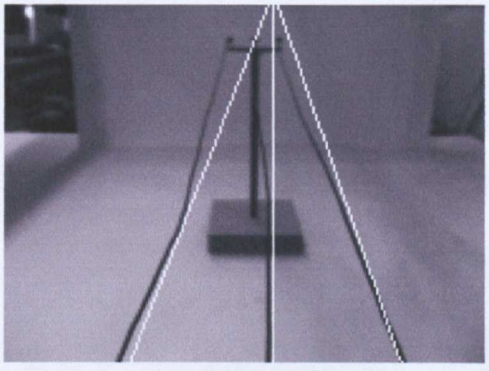


Figure 7.17: An Example of a "Found All Correct" Result.

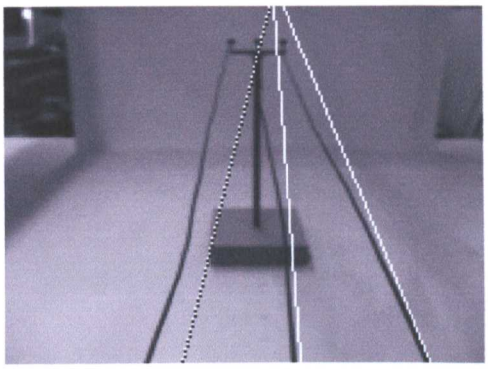


Figure 7.18: An Example of a "Found 2 Correct" Result.

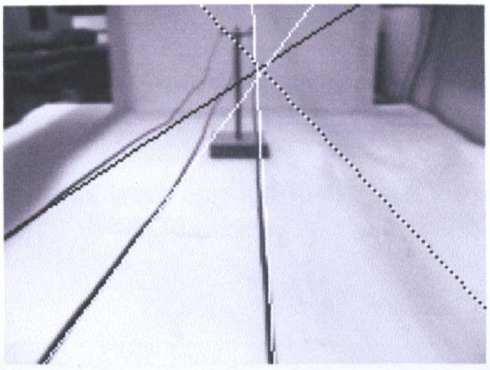


Figure 7.19: An Example of a "Found Sideline" Result.

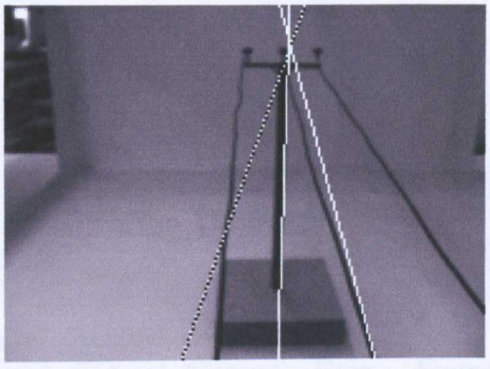


Figure 7.20: An Example of a "Found Pole" Result.

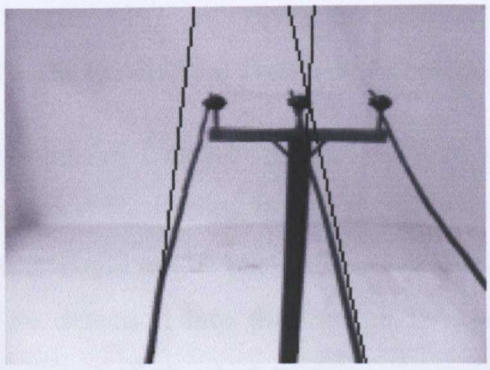


Figure 7.21: An Example of a "Not Found" Result.

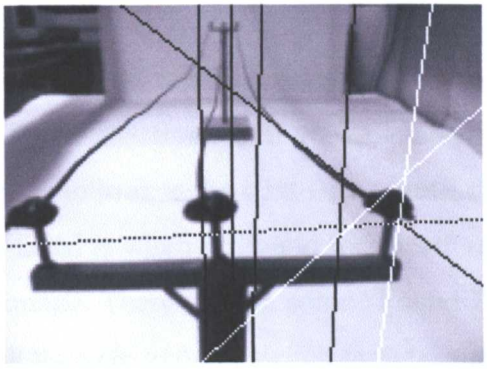


Figure 7.22: An Example of a "Found Incorrect" Result.

The acquisition routine was tested on five image sequences: a straight flight down the lines, the UAV flying down following a sinusoidal path and three sequences taken from an early version of the on-line version of the tracking software, where different strength wind gusts were applied to the UAV. The results for the five runs are as follows:

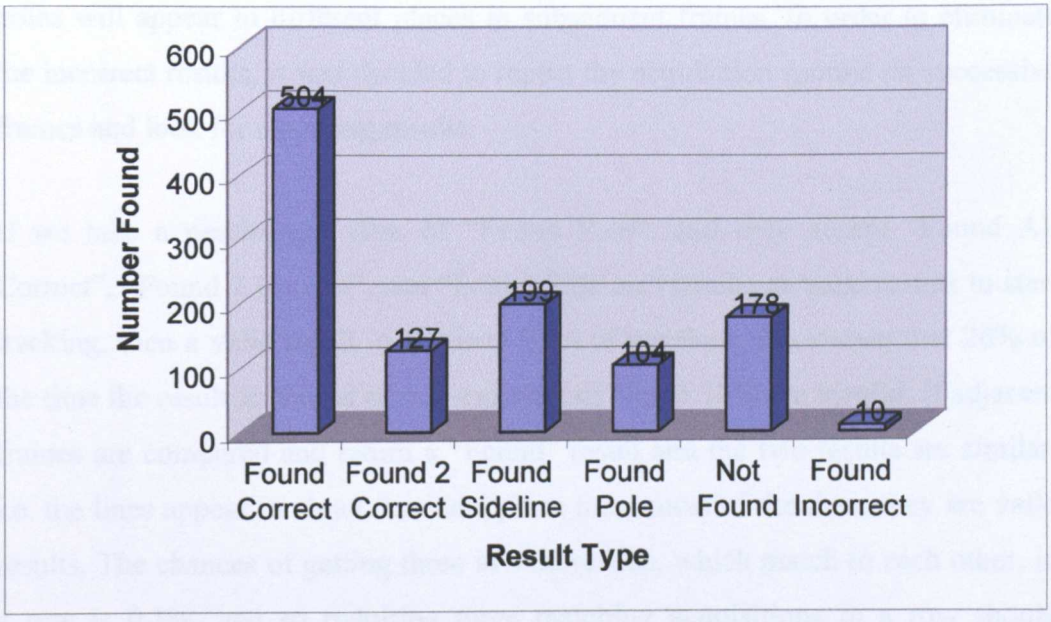


Figure 7.23: The Results from Testing the Acquisition Routine.

The desired result, "Found All Correct", occurs 45% of the time. A "Found 2 Correct" result is just as good for starting tracking as a "Found All Correct" result; a "Found Sideline" result would not be useful for starting the early tracker, but the incorporation of sideline detection into the tracker, as described in section 7.4, allows such a result to successfully start tracking. If these two result types are included, tracking would be successfully started 74% of the time. A "Found Pole"

result will often be able to switch to the lines and so may or may not be a valid start to tracking; if "Found Pole" results are included as valid, tracking would be started successfully 83% of the time. If the acquisition routine returns a "Not Found", the only option is to look at the next frame, until the lines are found. The only time a problem is faced is when a "Found Incorrect" result starts tracking the wrong features in the image. There will be some "Found Pole" results that cause the system to only track the pole and not switch to the centre line. As described in the next section, it was necessary to extend the acquisition routine in order to alleviate this problem.

7.3.4 Implementation of the Repeat Acquisition Routine

If the acquisition routine is run on successive frames, it would be expected that the actual lines will appear in approximately the same place in each frame whereas results associated with the background or the insulators on top of the poles will appear in different places in subsequent frames. In order to eliminate the incorrect results, it was decided to repeat the acquisition routine on successive frames and look for matching results.

If we take a pessimistic view of "Found Pole", and only accept "Found All Correct", "Found 2 Correct", and "Found Sideline" results as valid results to start tracking, then a valid result is obtained 74% of the time; this means that 26% of the time the result is invalid or non-existent, of which 10% are invalid. If adjacent frames are compared and return a "Found" result and the two results are similar, i.e. the lines appear in about the same place then, most of the time, they are valid results. The chances of getting three invalid results, which match to each other, in a row is 0.1%, and so requiring three matching acquisitions in a row should almost always give a valid result. In order to implement this, a count variable is used, if the results match, this is incremented; otherwise, it is decremented. Using the count allows hypothesis testing. If a match for the lines is found, it is uncertain whether it is a true or false match. Looking at subsequent frames allows the testing of the hypothesis that the match is a true match. If the subsequent frames return similar matches, then this indicates that the match is a true match. If the count goes below zero then the acquisition routine starts again but if the count

reaches three, the result is used to start tracking. Figure 7.24 shows the operation of the repeat acquisition routine.

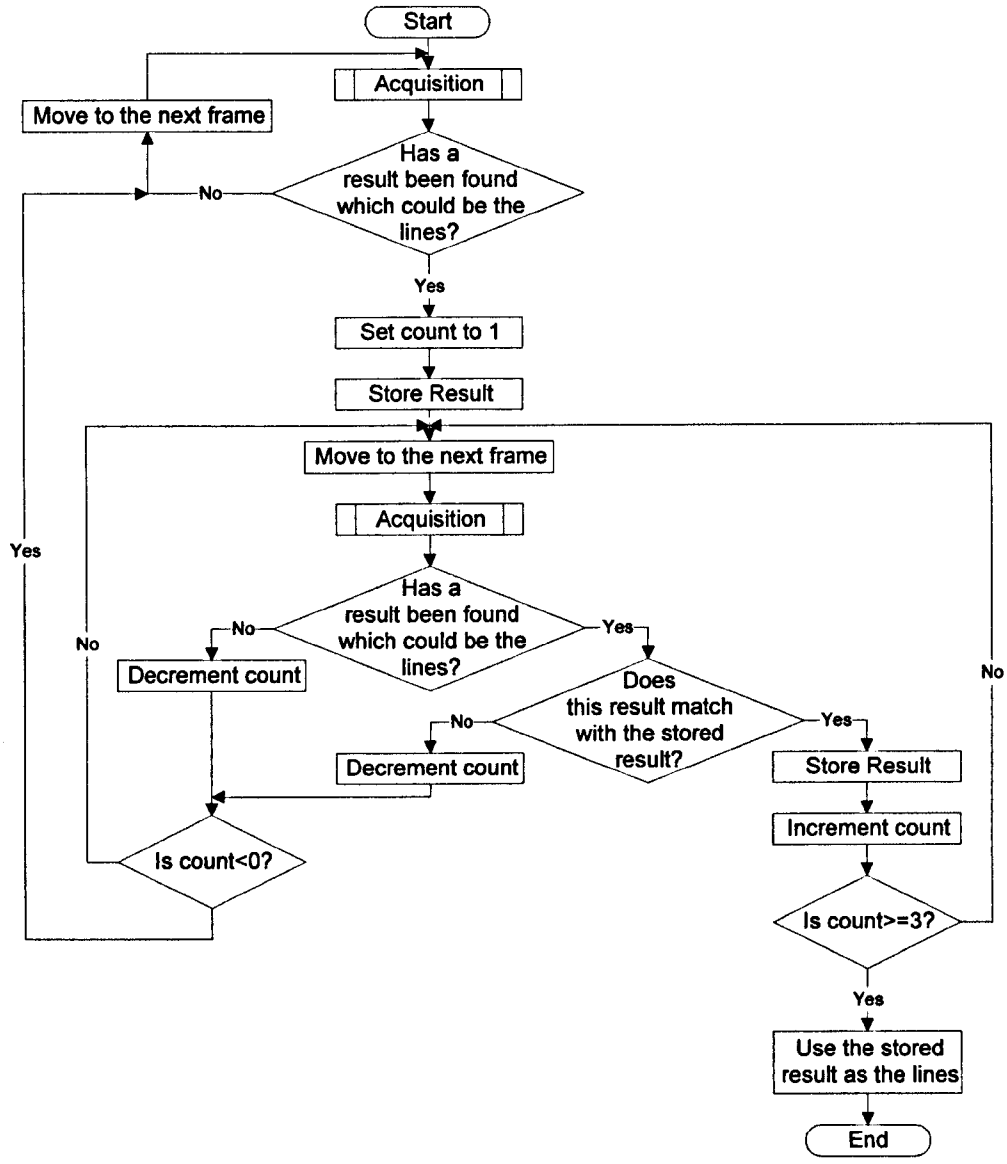


Figure 7.24: Repeat Acquisition Flowchart.

7.3.5 Results from Repeated Acquisition

Due to the nature of the repeated acquisition routine, it has to give a "Found" result eventually. The only time that this would not happen is if the tracker were to lose the lines close to the end of a line and attempt to re-acquire them; in this case, the UAV may reach the end of a line before successful acquisition occurs. In practice, the UAV could be made to hover above the lines until they are re-acquired. This, however, is not an option for off-line testing as image sequences

are used. What is of interest in this case is the number of frames that it takes to acquire the lines. In the ideal case, this will be three frames; it is not possible to acquire the lines in less than three frames due to the nature of the repeated acquisition routine. In addition, the number of invalid results also needs to be found. In order to test these, the repeat acquisition routine was run on the five test sequences, starting the routine at each frame of each sequence. The number of frames required to acquire the lines from each starting frame was recorded, along with whether the routine finished or ran out of frames; this only happened towards the end of each sequence. From these the mean number of frames for acquisition could be calculated, along with the percentage of acquisitions that happened in only three frames. The results of the test showed that the mean number of frames for acquisition is 7.4 and 50% of the acquisitions happened in 3 frames. In total, 87% of the results gave valid matches to the lines, while 13% had locked onto the distant pole. The acquisition routine returned no results associated with the background or insulators.

7.4 Rule-based Tracker

7.4.1 Problems with the Early Tracker

The results in section 7.2 exhibit a number of tracking problems. The main problem occurs when the tracker thinks that one of the sidelines is the centre line and so tracks it as if it were the centre line, an example of which is shown in Figure 7.25a. Other problems include the tracker losing the lines altogether and, very occasionally, when the tracker thinks that the centre line and one of the sidelines are the two sidelines. The centre line is then predicted incorrectly to be mid-way between them, as shown in Figure 7.29. In addition the optimum size of the search-squares needs to be chosen.

7.4.1.1 Sideline Tracking Detection

In this section a fuzzy logic rule is developed to detect erroneous tracking of a sideline. If only the centre line and one sideline are found (white), there are two possibilities as to why:

- The tracker has locked onto the left hand sideline and believes the centre line is the right hand line, as shown in Figure 7.25a, so the predicted left hand line (dotted) doesn't correspond to any of the actual lines. The mirror image of this can also occur.
- One of the sidelines is missing from the AHT. As shown in Figure 7.25b, only the centre line and the other sideline (white) have been found while the predicted third line (dotted) is close to the position of the actual line in the image.

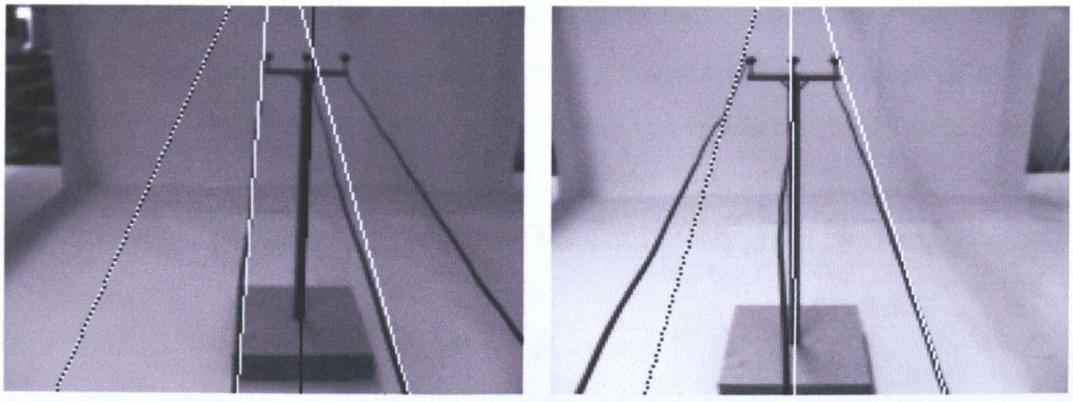


Figure 7.25: Possible Causes of only Finding One Sideline: a: Tracking Sideline, b: Other Sideline Missing from the AHT.

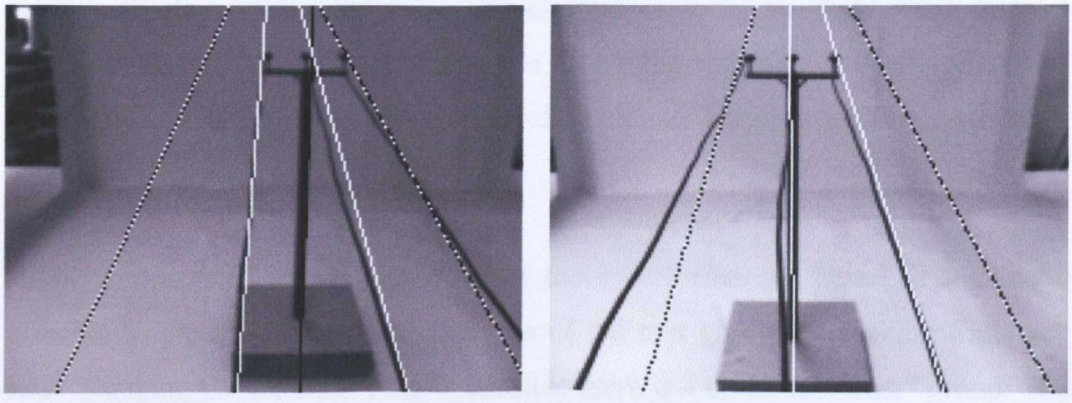


Figure 7.26: Fourth Line Predictions for the Case where Only One Sideline is Found: a: Tracking Sideline, b: Other Sideline Missing from the AHT.

The two patterns in Figure 7.25 are indistinguishable from the tracker's point of view so it is necessary to search for a fourth line (dash-dotted), which could be the missing sideline, as shown in Figure 7.26.

This fourth line, or "extra line" as it is referred to in the software, is predicted from the two lines that the tracker has found, in a similar fashion to predicting a missing sideline. Analysis in section 4.2.2 indicates that lateral displacement of the UAV relative to the overhead lines causes them to appear closer together in the image. In cases where the tracker had locked onto a sideline the "extra line" was found experimentally to appear at approximately 0.8 times the distance between the centre line and the sidelines in the tracker. The θ and ρ values are, therefore, multiplied by 0.8 In the case where the left hand line is missing, the position of the extra line is given by (7.7); if the right hand line is missing, the mirror image is used, and the position of the extra line is given by (7.8).

$$\theta_e = 0.8*((2*\theta_r)+\theta_c) \quad \rho_e = 0.8*((2*\rho_r)+\rho_c) \quad (7.7)$$

$$\theta_e = 0.8*((2*\theta_l)+\theta_c) \quad \rho_e = 0.8*((2*\rho_l)+\rho_c) \quad (7.8)$$

In this case it can be seen that when the tracker has locked onto the left-hand sideline (Figure 7.26a), the extra line appears in about the right place to be a match for the right-hand line, whereas in case b the extra line doesn't correspond with anything in the image. In order to discriminate between the two cases, it is necessary to look at subsequent frames.

Ideally, each time only a sideline and the centre line are found, it would be possible to place each instance into one of the two groups: "Tracking Sideline" and "Tracking Centre Line", but in practice there is a third "Not Sure" group. The three groups: "Tracking Sideline Very Likely", "Not Sure" and "Tracking Centre Line Very Likely" form a fuzzy set, as discussed in Chapter 2. A simple fuzzy logic rule can be used to detect which group an individual instance belongs to.

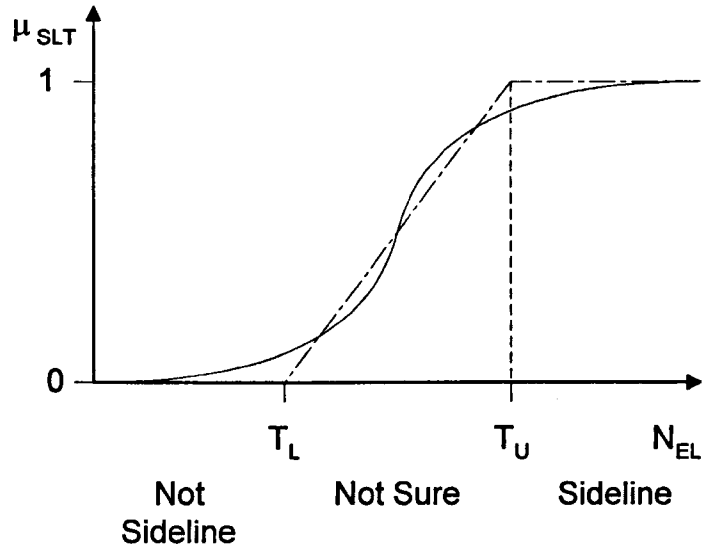


Figure 7.27: Fuzzy Membership Function for the Sideline Detection and its Piecwise Linear Approximation.

When this two-line case occurs, a count variable, N_{EL}, is set to 0. Figure 7.27 shows how the fuzzy set membership function, μ_{SLT}, changes with the value of N_{EL}. In order to de-fuzzify the set, the membership function is approximated by a piecewise-linear μ_{SLT} function. If a match is found in subsequent frames for the fourth line, this value is incremented, otherwise it is decremented. In addition if the third line reappears in the transform and the tracker finds it, N_{EL} is decremented. What this does is to test the hypothesis that the tracker is tracking a sideline rather than the centre line. When the extra line is found, this reinforces the hypothesis, while failing to find the extra line or re-acquiring the missing sideline tends to disprove the hypothesis and indicates that the tracker is actually tracking the centre line and the sideline has merely disappeared from the transform. Two threshold values are defined for the piecewise linear μ function: T_L and T_U. While $T_L<N_{EL}<T_U$ remains the case, the system remains in the not sure state. If $N_{EL} <= T_L$ then the system enters the Tracking Centre Line Very Likely state and the tracking of the fourth line is stopped and if $N_{EL} >= T_U$ then the system enters the Tracking Sideline Very Likely state and switches to the centre line. It was found that $T_L=-3$ and $T_U=3$ were sufficient to separate the two cases. In the case where the sideline is being tracked the tracker switches as shown in Figure 7.28

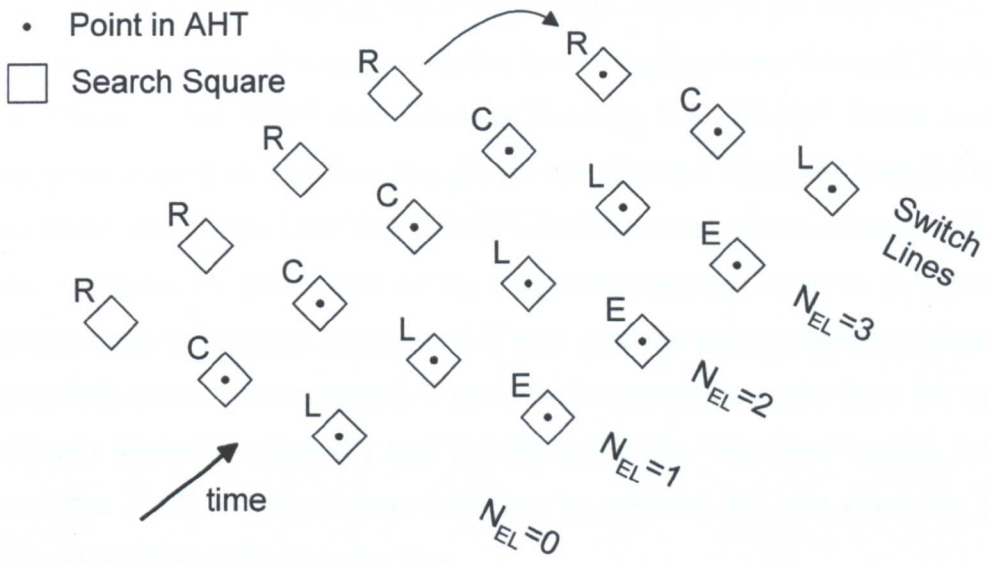


Figure 7.28: Switching Lines When a Sideline is being Tracked.

7.4.1.2 Detection of the Loss of the Centre Line or Loss of the Lines

There are two other problem cases. The first is loss of the centre line, i.e. what the tracker thinks are the two sidelines, are in fact the centre line and a sideline, as shown in Figure 7.29. In Figure 7.29 the white lines correspond to the lines the tracker thinks are the sidelines, while the dotted line corresponds to the predicted centre line.

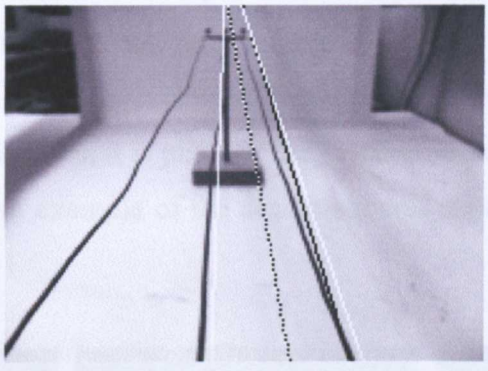


Figure 7.29: Example Case where the Tracker Thinks that the Centre Line and a Sideline Correspond to the Two Sidelines.

The second problem is a loss of the lines, defined as finding no lines or just one line. Finding only one line is insufficient for tracking, as it could easily correspond to a line from the background.

When these cases are detected, fuzzy logic rules, similar to the sideline rule are invoked. In the case of a missing centre line, the groups are "Correct Tracking Very Likely", "Not Sure" and "Incorrect Tracking Very Likely". In the case of finding no more than one line, the groups are "Correct Tracking Very Likely", "Not Sure" and "Lines Lost Very Likely". In these cases we are starting with the count variables, N_L (lines lost) or N_{IT} (incorrect tracking) equal to 0. Tracking continues into subsequent frames, and if they give the same conclusion, then the appropriate count is incremented, if not it is decremented. Again there are upper and lower threshold values (T_L and T_U) that define the "Not Sure" region. It was found that T_L=0 and T_U=5 were sufficient to separate the two cases for both incorrect tracking and losing the lines.

7.4.1.3 Selection of the Optimum Search-square Size

It was necessary to select optimum search-square sizes. If the search-squares are too small, then some matches won't be made as the UAV moves to one side. On the other hand if the search-squares are too big the tracker is more likely to switch to an incorrect line. This most often happens if the centre line and pole are not quite co-linear and the centre line disappears for a frame-the tracker then switches to the pole. An example of this happening is shown in Figure 7.30 where the white lines indicate the tracker's output that correspond to lines from the AHT, while the black lines indicate other lines present in the AHT, which aren't used by the tracker. These lines are correctly found by the Hough transform and correspond to image features, it is just that they are not selected as matches for the lines by the tracker. An example of the search-squares superimposed on an AHT is shown in Figure 7.31.

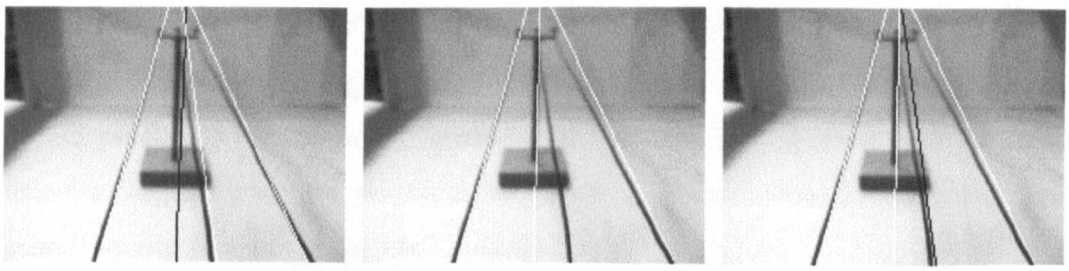


Figure 7.30: Images of the Lines with the Tracker Output Superimposed when the Tracker Switches from the Centre Line to the Pole.

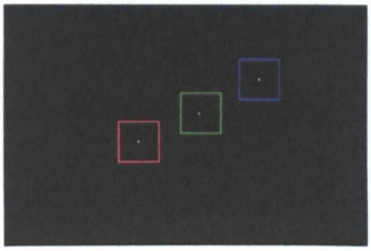


Figure 7.31: Tracking Squares Superimposed on an AHT.

The first part of the selection process involved selecting the minimum search square size. In order to do this the Rule-based tracker was run off-line with minimum search square sizes varying from 3x3 - 25x25 pixels on two wind sequences. These sequences include both the situation where the tracking switches to the pole and the situation where the UAV is displaced laterally. If the search square is too large then the tracker is more likely to switch to the pole and if the search square is too small then tracking breaks down as the UAV moves laterally. The optimum search square size is a compromise between these two and can be determined by looking at the resulting processed image frames and tracking data to see where the tracker has estimated the line positions. The optimum minimum search square size was found to be 15x15 pixels.

After the minimum search square size had been chosen, it was necessary to select the maximum search square size. Again the rule-based tracker was run off-line on the same two image sequences, this time varying the maximum search square size from 17x17-45x45 pixels. If one of the lines has been missing for a few frames and continues to be predicted, the prediction may well drift away from the actual line. When the actual line reappears in the AHT, the search square can at times be too small to enclose and recapture it. The results showed that, with smaller maximum search square sizes, the line is occasionally not recaptured. This is shown in Figure 7.32a. With a larger maximum search square size, the line is recaptured as shown in Figure 7.32b. With a maximum search square size above 35x35 pixels, there was no difference in the results, and so there is little point using a larger maximum search square size; for these reasons, the maximum search square size was set at 35x35 pixels.

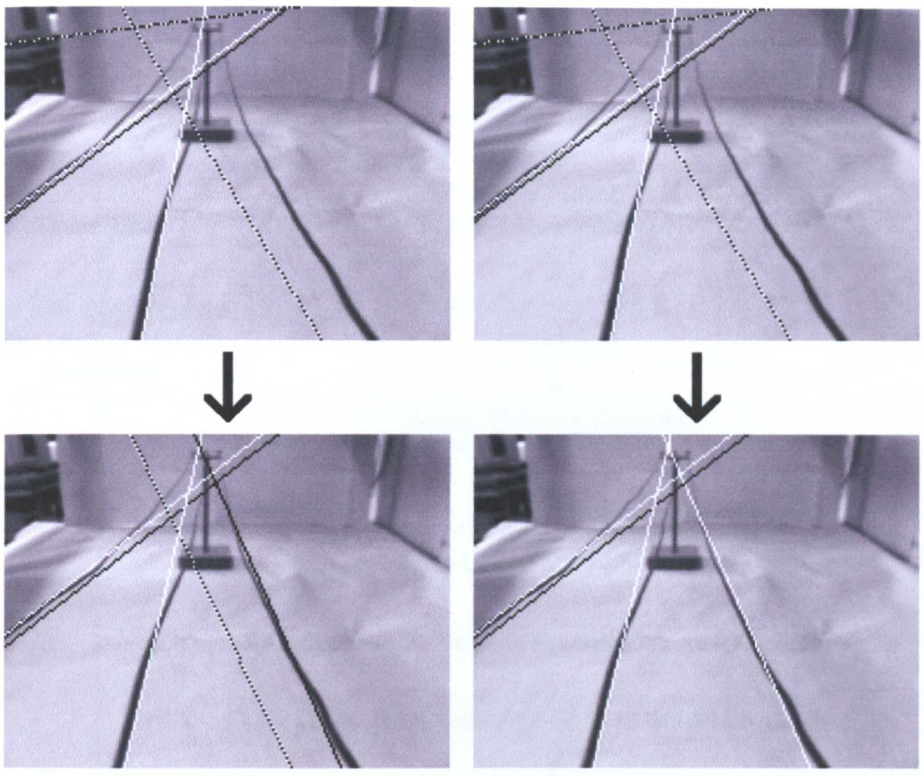


Figure 7.32: Finding Maximum Search-square Size; the Maximum Search-square Sizes used are: a: 33x33 pixels, b: 35x35 pixels.

7.4.2 Implementation of the Rule-based Tracker

Testing of the rule-based tracker was done in a similar manner to the early tracker. It was coded in MATLAB and run off-line on the same image sequences as were used for testing the acquisition routine. Once the off-line version of the tracker was working, it was ported into C++ and incorporated into the test rig control software.

Figure 7.33 shows the operation of the rule-based tracker. A state machine is used in the program in order to control which parts of the tracker run. This state machine is shown in Figure 7.36. The repeated acquisition process is described in section 7.3, while the image processing, search of the AHT and third line prediction are as described in section 7.2; their flowcharts are shown in Appendix C. Flowcharts showing the operation of the Fuzzy Logic rules are in Figure 7.34 and Figure 7.35.

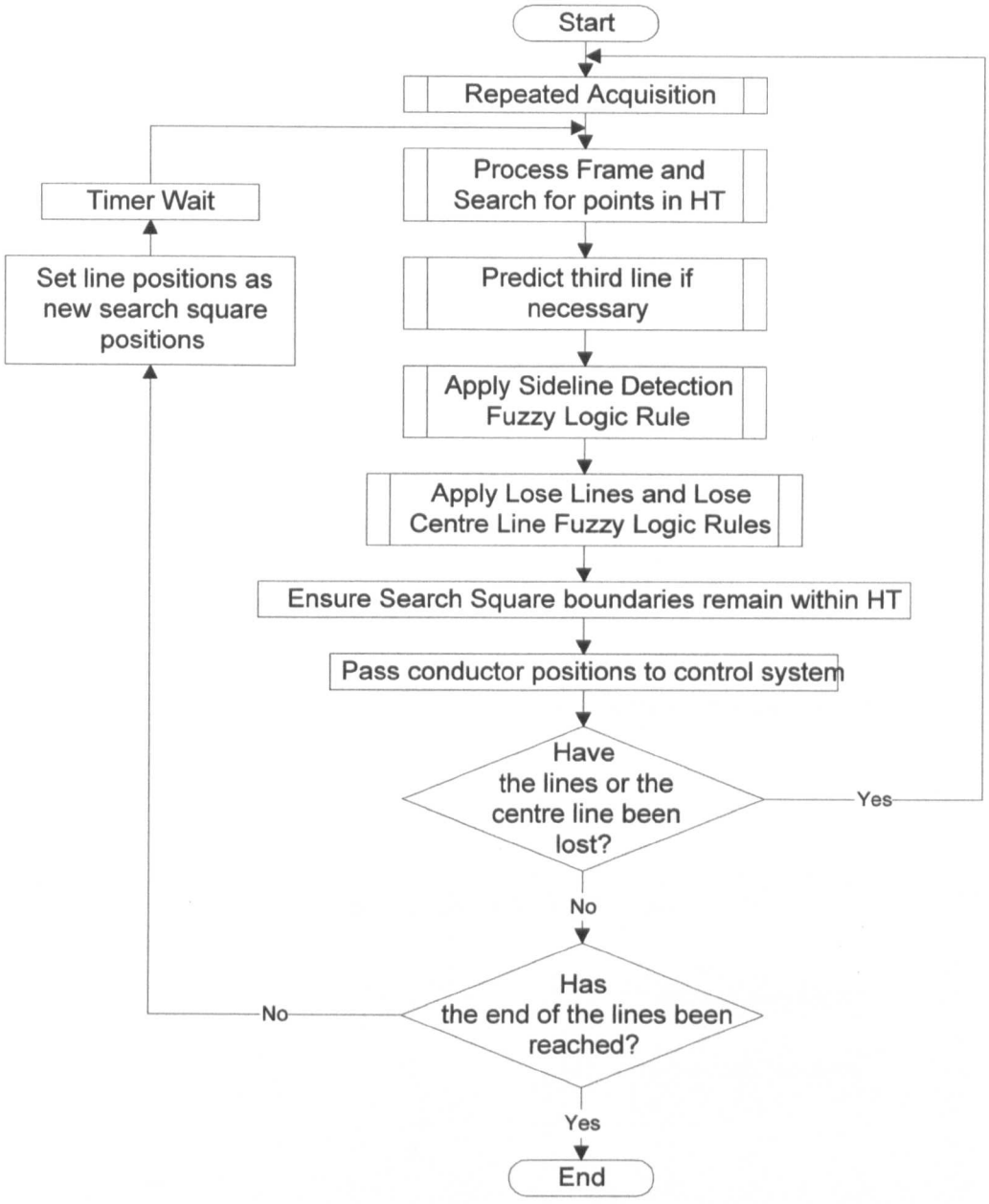


Figure 7.33: Flowchart for the Rule-based Tracker.

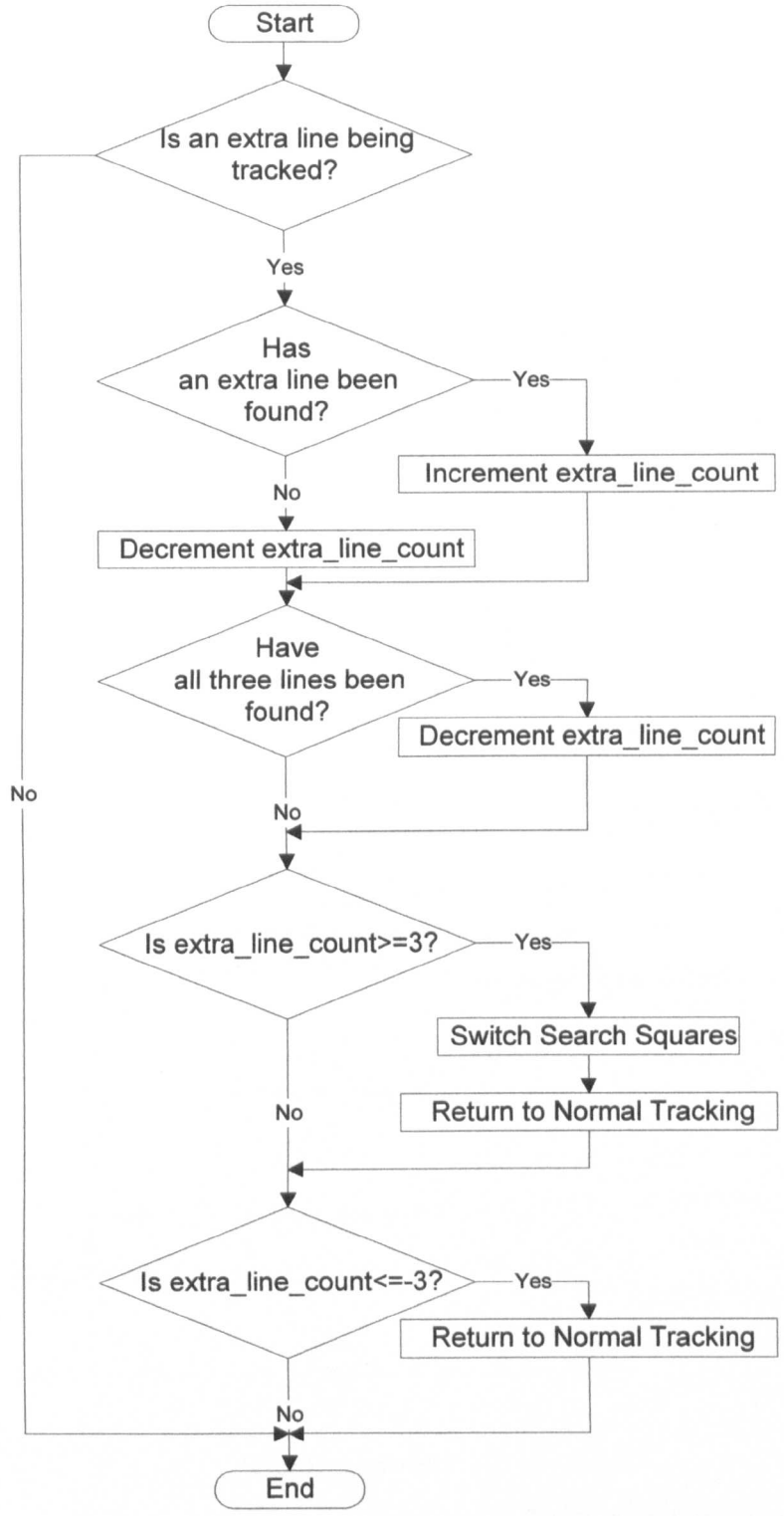


Figure 7.34: Flowchart showing the Operation of the Sideline Detection Rule.

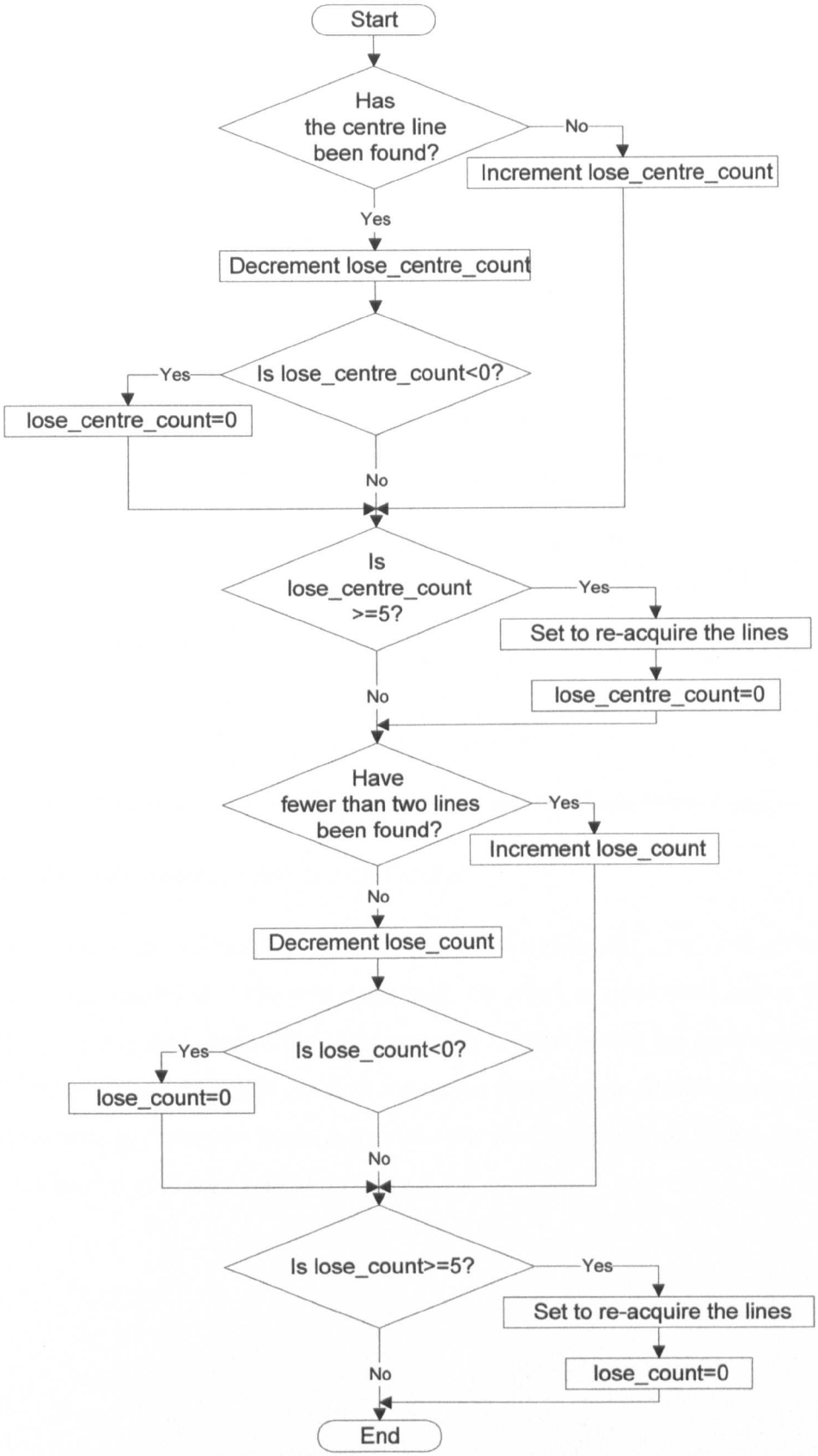


Figure 7.35: Flowchart showing the Operation of the Lose Rules.

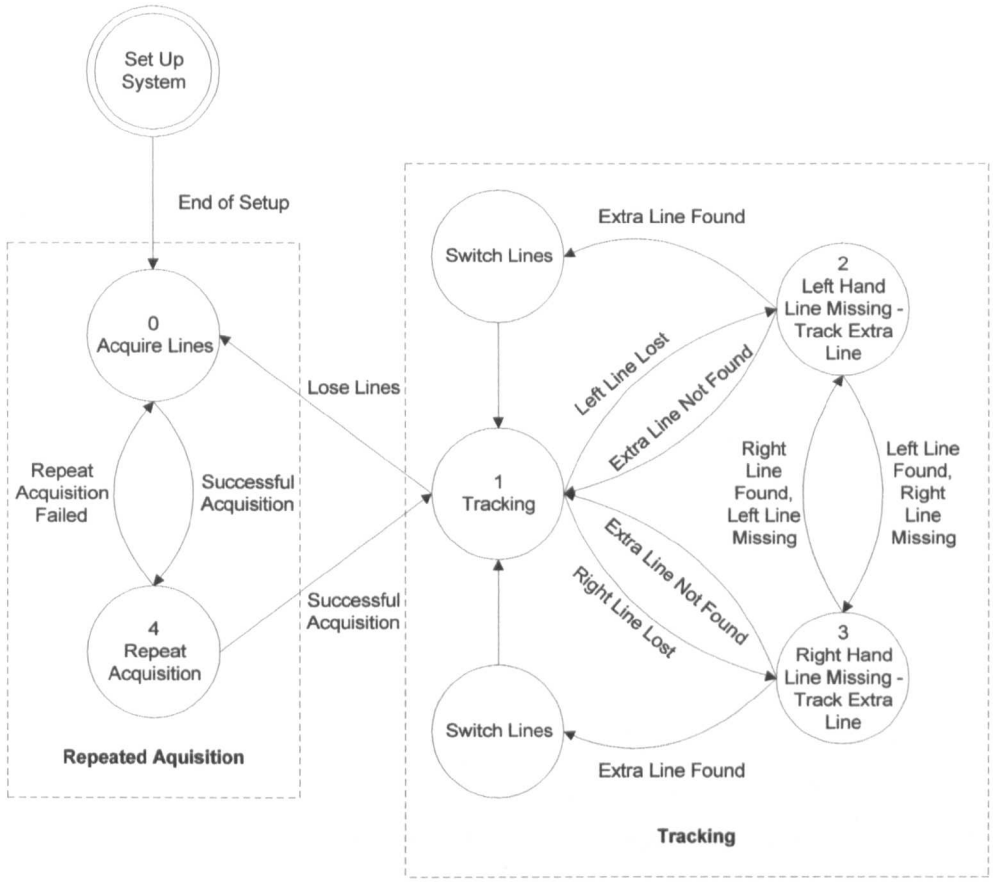


Figure 7.36: State Transition Diagram showing how the Tracker State Changes.

7.4.3 Results from the Rule-based Tracker

In order to test the new tracker, the UAV was flown along the lines, and its lateral displacement recorded. This was done with no wind, a wind gust and a larger wind gust: these gusts are of the same strength as used to test the early tracker. It would be expected that with the first two cases should give similar results to the previous tracker, because there is a relatively low incidence of losing the line. This confirmed in Figure 7.37 and Figure 7.38.

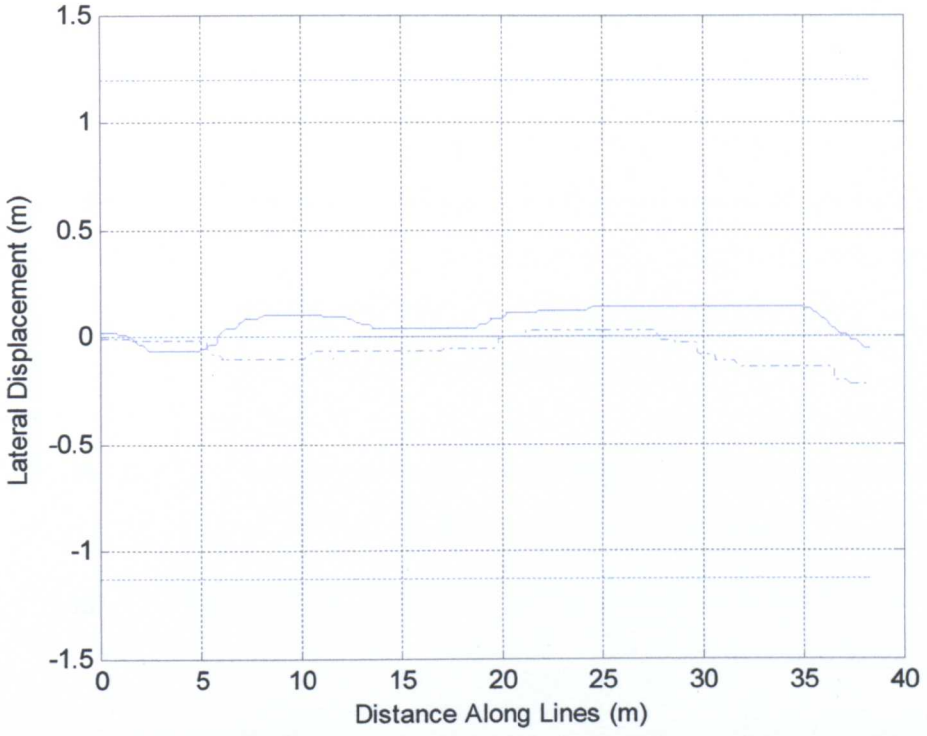


Figure 7.37: Lateral Displacement of the UAV with Rule-based Tracker Subject to No Wind.

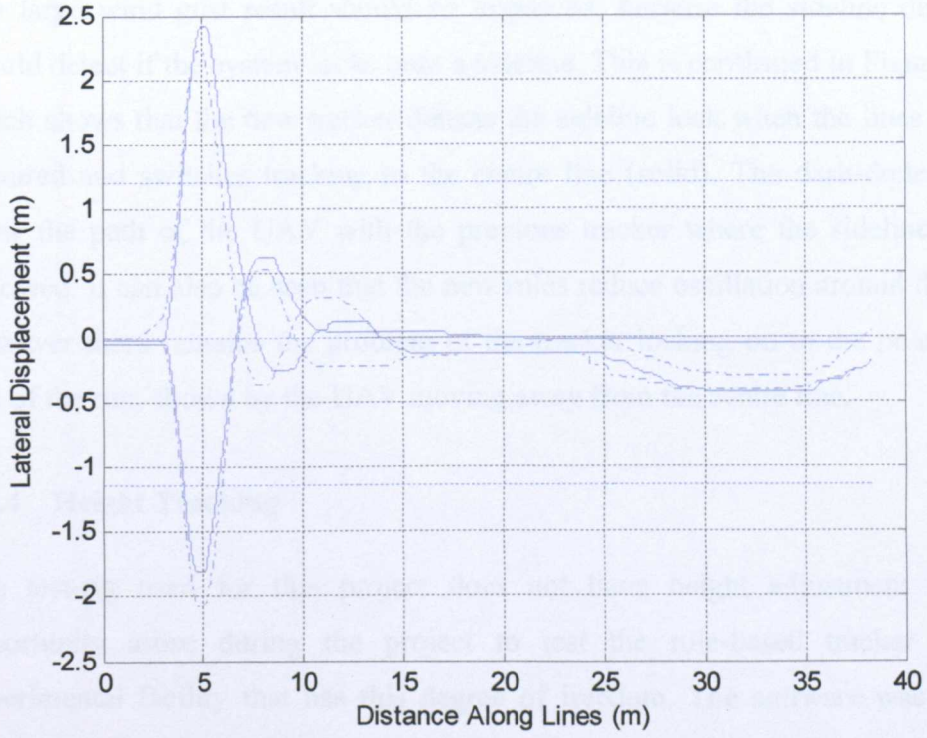


Figure 7.38: Lateral Displacement of the UAV with Rule-based Tracker in Response to a Pulse Wind Gust.

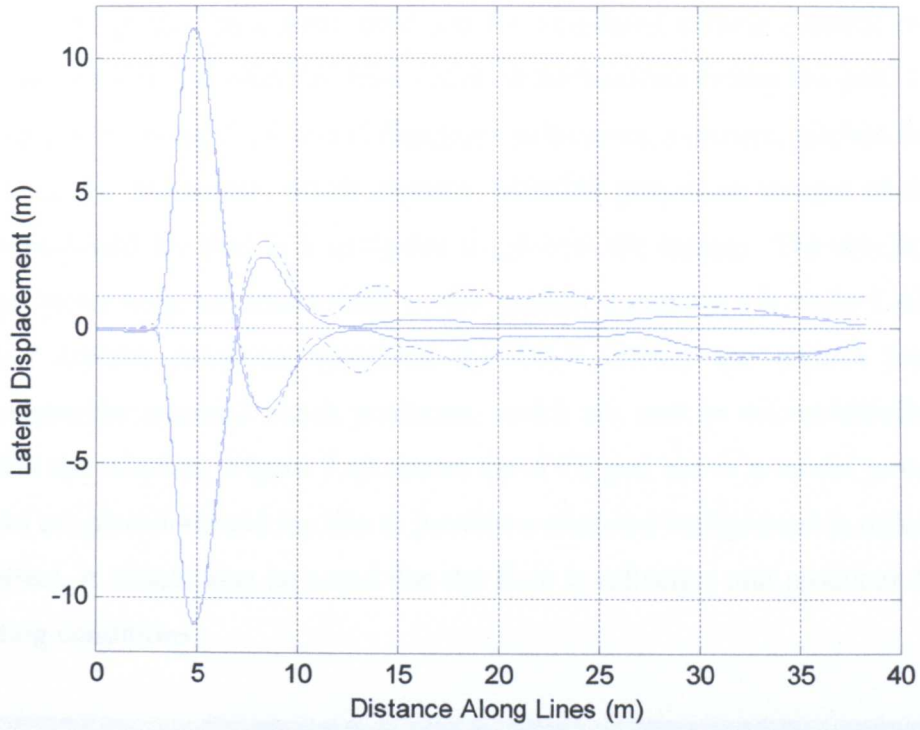


Figure 7.39: Lateral Displacement of the UAV with Rule-based Tracker in Response to a Large Pulse Wind Gust.

The large wind gust result should be improved, because the sideline detection should detect if the system locks onto a sideline. This is confirmed in Figure 7.39, which shows that the new tracker detects the sideline lock when the lines are re-acquired and switches tracking to the centre line (solid). The dash-dotted lines show the path of the UAV with the previous tracker where the sidelines were followed. It can also be seen that the new rules reduce oscillation around the line. However there remains the problem of the tracker locking on to the pole at the end of the run, shown by the UAV moving away from the centre line.

7.4.4 Height Tracking

The test-rig used for this project does not have height adjustment but an opportunity arose during the project to test the rule-based tracker on an experimental facility that has this degree of freedom. The software was ported onto the Air Vehicle Simulator (AVS) [39, 40] at the Autonomous Systems Laboratory, CSIRO, Australia and tested there by Dr Dewi Jones. The AVS is essentially a larger scale version of the laboratory test-rig described in Chapter 5,

measuring approximately ten metres long by eight metres wide by six metres high. This operates in a large shed and the simulated vehicle consists of a pod suspended from four winches; movement of the winches moves the pod. The pod can be moved in the X, Y and Z directions and carries a camera, pitched down by 20° from the horizontal, which captures 640x480 grey-scale images of the line. Also on-board the pod is a computer to process the images. The results of the image processing are transmitted to the control computer via radio LAN. The central control computer simulates the UAV, closes the control loop and calculates the required winch positions, which are sent to microcontrollers that control the winches. Figure 7.40 shows the AVS pod above a model power line. Rocks are placed around the line to provide a cluttered background in order to test its effect. It should also be noted that the floor is reflective and produces difficult lighting conditions.

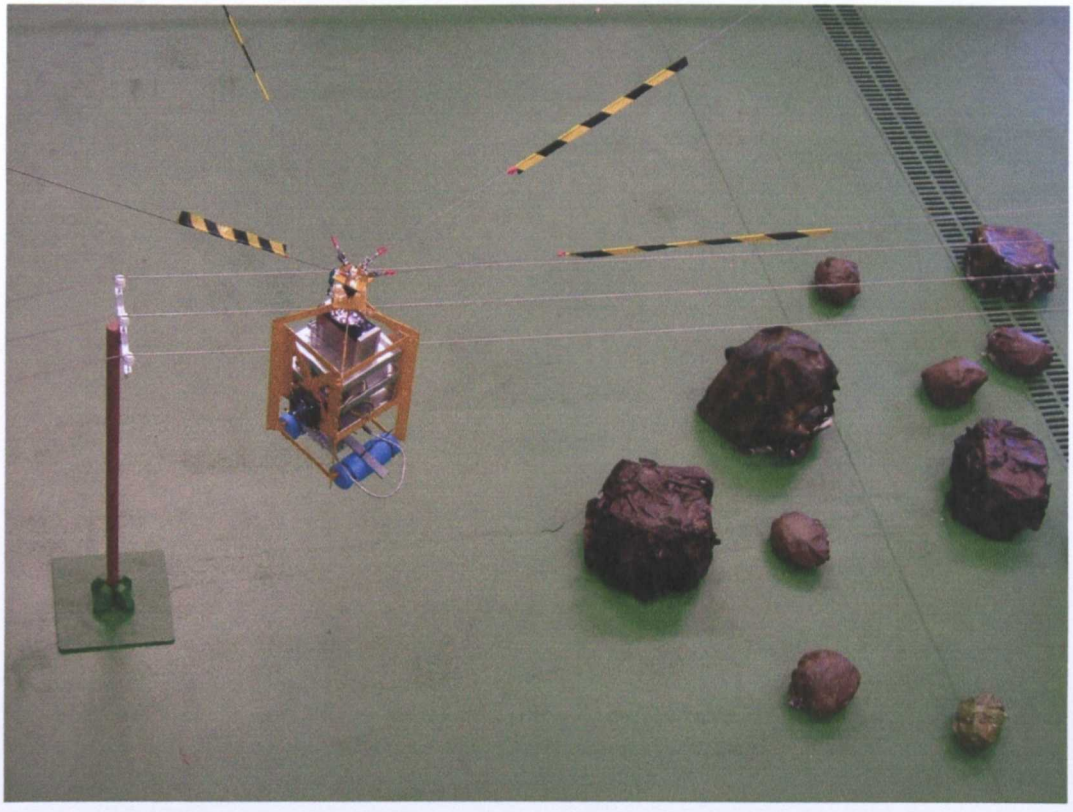


Figure 7.40: The AVS Pod suspended above a Model Power Line.

The UAV model was also ported onto the AVS software and vision feedback used to track the pod's lateral displacement from the centre line position. The average distance between the sidelines and the centre line was used for tracking the height

of the pod. Calibration was performed using the AVS; yielding the following equations for the lateral displacement, X, (7.9) and the height, Z (7.10).

$$X = -0.34315 + (0.0012288\rho_c) + (0.0040401\theta_c) \quad (7.9)$$

$$Z = -0.56246 + \frac{35.308}{d} \quad (7.10)$$

where:

ρ_c and θ_c are the r and q values associated with the centre line

d is the average distance between the sidelines and the centre line

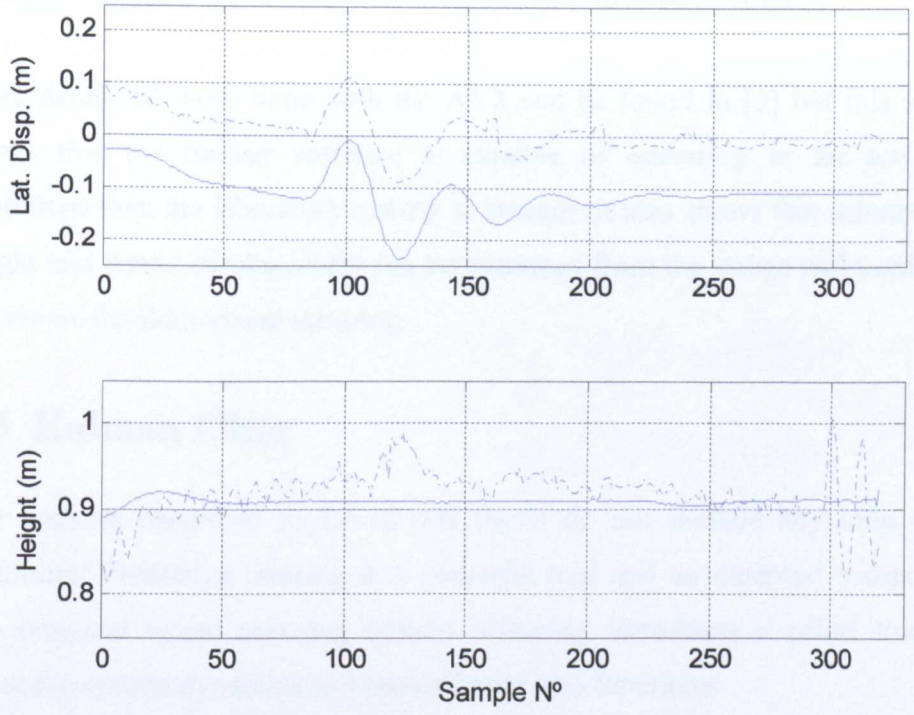


Figure 7.41: Results of tracking both Lateral Displacement and Height on the AVS; Sample N° indicates distance along the lines.

Figure 7.41 shows how well the pod lateral position and height as measured by the AVS (solid) matches to the demand (dotted); the dash-dotted line shows the measurement of lateral displacement and height from the visual tracker. It can be seen that the vision output for the lateral displacement is noisier than the measured position. The UAV is able to re-align with the lines. With the height

measurement it can be seen that the position follows the demanded height well although the output from the tracker is quite noisy, particularly at the end of the run. This is due to the AVS pod pitching as it comes to a stop. It is to be expected that the vision output for the height would be more noisy than the lateral displacement as it is calculated from the difference between points in the AHT and so there will be a larger error than for the lateral displacement, which is calculated from the position of the point representing the centre line. It can be seen from the vision height measurement during the small wind gust that there is some effect on the height measurement from the lateral displacement. This coupling would be expected because the lines appear closer together as the UAV is displaced laterally. This was predicted in section 4.2.2. Looking at the lateral displacement results, there is a distinct offset between the AVS and vision measurement; this is probably due to an offset in the line position.

More details of work done with the AVS can be found in [3] but this summary shows that the tracker software is capable of operating in far less benign conditions than the laboratory test-rig at Bangor. It also shows that information on height and lateral displacement can be extracted from the image and used for two degree-of-freedom visual servoing.

7.5 Kalman Filter

The trackers described so far in this thesis do not include any kind of filter algorithm. Predictive filtering is a powerful tool and an essential component of any practical visual servoing system. Filtering introduces a priori knowledge about the system dynamics and basically has two functions:

- It places dynamic constraints on the possible motion of the features between one image frame and the next.
- It allows better predictions, based on past measurements and the internal dynamic model of where image features will be located in future frames.

Filtering therefore contributes a great deal to the robustness of a tracker and it is somewhat remarkable that the preceding results have been obtained without its

inclusion. Nevertheless, improved robustness is still a necessary goal and this section considers the addition of a Kalman filter to the tracking algorithm.

7.5.1 Description of the Kalman Filter Tracker

The Kalman Filter is a recursive filter as discussed in section 2.4. It gives an optimal solution if the errors in the measurements being filtered have a Gaussian distribution. With other error distributions, the Kalman filter can still give good results but in order to use it in these situations, the errors are assumed to be Gaussian.

For this application Kalman filters are used to filter the ρ and θ values of each line. It should be noted that the Kalman filter is include in addition to the fuzzy logic rules described in section 7.4. The Kalman filter has four stages:

- Calculate the Kalman gain: this determines the fraction of the estimate that is from the prediction and the fraction from the measurement.
- Update the estimate.
- Update the error (variance) associated with the estimate.
- Calculate the prediction for the next frame and its associated error (variance).

Using equations from [46], at sample k the Kalman gain is given by

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T \left(\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k \right)^{-1} \quad (7.11)$$

where:

$\mathbf{P}^-_k$ is the error covariance matrix associated with the prediction

$\mathbf{H}_k$ is matrix giving the ideal connection between the measurement and the state vector

$\mathbf{R}_k$ is the error covariance matrix associated with the measurement

The estimate is given by

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^- \right) \quad (7.12)$$

where:

$\hat{\mathbf{x}}_k^-$ is the prediction vector

$\mathbf{z}_k$ is the measurement vector

The variance associated with the estimate is given by:

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_k^- \quad (7.13)$$

where:

I is the identity matrix

The prediction and its associated variance are given by:

$$\hat{\mathbf{x}}_{k+1}^- = \varphi_k \hat{\mathbf{x}}_k$$

$$\mathbf{P}_{k+1}^- = \varphi_k \mathbf{P}_k \varphi_k^T + \mathbf{Q}_k \quad (7.14)$$

where:

φ_k is the matrix relating the current estimate to the next estimate.

$\mathbf{Q}_k$ is the error covariance matrix associated with φ_k

In our Kalman filters $\hat{\mathbf{x}}$, $\hat{\mathbf{x}}^-$ and $\mathbf{z}$ represent the estimate, prediction and measurement of ρ or θ, and are scalars. Ideally, in this application the measurement should be equal to the state vector, as it is the measurement that is being filtered. For this reason H is equal to 1. There is now a measure of the accuracy of the prediction: the variance of the prediction, P_k^-. This can be used to set the search square size. As 99.7% of results are within three standard deviations of the mean, the search-square size, S_{SS}, is calculated using:

$$S_{SS} = ceil\left(1 + \left(6 * \sqrt{P_k^-}\right)\right) \quad (7.15)$$

The Kalman filter equations (7.11), (7.12) and (7.13) become for θ; there is an identical Kalman filter for ρ:

$$K_k = \frac{P_k^-}{P_k^- + R_k} \quad (7.16)$$

$$\hat{\theta}_k = \hat{\theta}_k^- + K_k\left(\theta_{mk} - \hat{\theta}_k^-\right) \quad (7.17)$$

$$P_k = (I - K_k)P_k^- \quad (7.18)$$

For the early tracker and rule-based tracker, the predictor, ϕ, used was a zero order predictor. This type of predictor is known as a trivial predictor and simply uses the current measurement as the prediction. This predictor works well when the UAV isn't moving laterally. When the UAV is moving laterally, the zero order predictor is not as good as the lines do not appear in the same place in consecutive frames. However it still works if the UAV doesn't move too far between frames. Prediction can be improved for these situations by changing to a first order predictor. A first order predictor is of the form:

$$\theta_{k+1} = \theta_k + \frac{\partial\theta}{\partial U}UT$$

$$\rho_{k+1} = \rho_k + \frac{\partial\rho}{\partial U}UT \quad (7.19)$$

where:

U is the UAV lateral velocity

T is the sample time

$\frac{\partial\theta}{\partial U}$ and $\frac{\partial\rho}{\partial U}$ are the Jacobian relating the speed to θ and ρ

It is possible to estimate the velocity of the UAV from the changing θ and ρ values. This allows the predictor to be simplified as it gives an estimate of the velocity in Hough co-ordinates, meaning that there is no need to calculate the Jacobian. When the UAV moves laterally, the values of θ and ρ change for the three lines by amounts $\Delta\theta$ and $\Delta\rho$ respectively. Assuming that the speed of the

UAV changes little from frame to frame to frame, the predictior in equation (7.19) becomes:

$$\theta_{k+1} = \theta_k + \Delta\theta$$

$$\rho_{k+1} = \rho_k + \Delta\rho \quad (7.20)$$

A higher order predictor could be used. This would be based on the UAV model described in Chapter 3 and could predict where the UAV would be based on known control inputs. Such a predictor would be complex to implement as the UAV motion is described in a Cartesian reference frame, while the Kalman filter is carried out in the AHT's reference frame. In order to use such a second order predictor, the two have to be related by the appropriate Jacobian, which includes that of the Hough transform. Moving to higher order predictors increases the complexity and is subject to the law of diminishing returns.

It was decided to use a first order predictor. In order to perform this with the Kalman filter equations, the prediction and estimate have to be put in normal form for this step. For θ the predictor, φ, is:

$$\varphi = \begin{bmatrix} 1 & \Delta\theta \\ 0 & 1 \end{bmatrix} \quad (7.21)$$

There is an equivalent predictor for ρ.

Equation (7.14) becomes:

$$\begin{bmatrix} \hat{\theta}_{k+1}^- \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \Delta\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\theta}_k^- \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} P_{k+1}^- & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & \Delta\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_k^- & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & \Delta\theta \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} Q & 0 \\ 0 & 0 \end{bmatrix} \quad (7.22)$$

Testing of the first order predictor showed it worked well. When one of the lines was missing from the AHT, meaning that its position was estimated from the other two, the calculated values of $\Delta\theta$ and $\Delta\rho$ were found to be a poor predictor for that line. It was found that better results could be obtained by estimating the values of $\Delta\theta$ and $\Delta\rho$ for the missing line from the values of $\Delta\theta$ and $\Delta\rho$ for the other two lines. This is done by taking the average of $\Delta\theta$ and $\Delta\rho$ for the other two lines. For example, if the right line is missing, $\Delta\theta_R$ and $\Delta\rho_R$ are given by:

$$\Delta\theta_R = \frac{\Delta\theta_C + \Delta\theta_L}{2} \quad \Delta\rho_R = \frac{\Delta\rho_C + \Delta\rho_L}{2} \quad (7.23)$$

7.5.2 Implementation of the Kalman Filter Tracker

The matrix equations were easy to code in MATLAB as it has matrix support, but C++ does not have this support natively. To allow for easy coding, the matrix equations were broken down into scalar equations. Equation (7.22) becomes:

$$\hat{\theta}_{k+1}^{-} = \hat{\theta}_k + \Delta\theta$$

$$P_{k+1}^{-} = P_k + Q \quad (7.24)$$

In addition to the equations for the Kalman filter, it is necessary to measure the error variances associated with the measurement (R) and the prediction (Q). In order to obtain an estimate of R, ten processed frames were selected from the image sequences used to test the trackers. These had the output lines from the AHT superimposed on them along with a marker of the image centre. Lines were marked onto the images, showing an estimate of the line position by eye. An example of this is shown in Figure 7.42

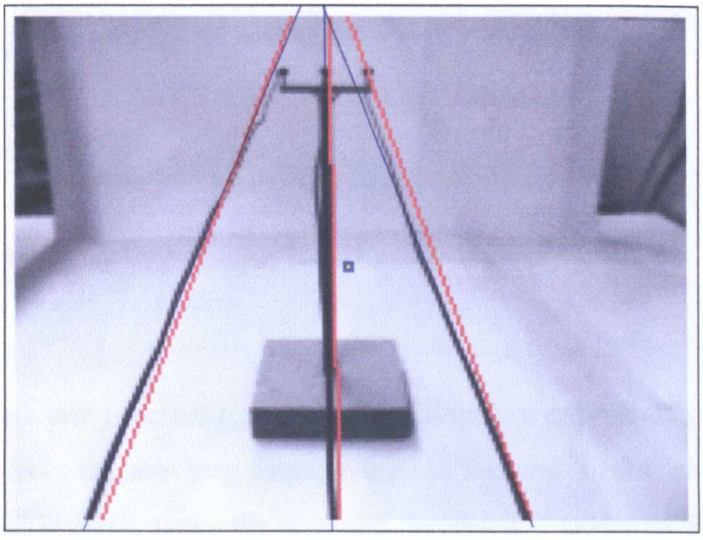


Figure 7.42: Example Frame Used to Measure R.

The difference between the ρ and θ values for each line in each of the ten frames was recorded. The variance was then calculated for both θ and ρ. The values of R were as follows and are used in all the Kalman filter tests:

θ: $R=3.9^{o2}$

ρ: $R=3.13pixel^2$ (7.25)

These values of the measurement apply when all three lines are found in a frame. However, during normal operation, lines are missing from some frames. In these cases, the measured value of R in (7.25) will not be representative of the error present in any estimate used as the measurement. The value of R needs to vary with time to reflect the effect on the error caused by the estimation of line positions.

In the case where one line is missing from the AHT, its position is estimated from the positions of other two. The value of R associated with the estimated line will be higher than the measured values as the error in the estimated position is higher than if it were present in the transform. The value of R for an estimated line position is calculated by adding up the standard deviations of the line positions the estimate is calculated from. For an estimated centre line, R_C is calculated by equation (7.26) and when the sideline is estimated, R is given by (7.27) for an estimated left line and (7.28) for an estimated right line.

$$R_C = \left(\sqrt{R_L} + \sqrt{R_R}\right)^2 \quad (7.26)$$

$$R_L = \left(\left(2 * \sqrt{R_C}\right) + \sqrt{R_R}\right)^2 \quad (7.27)$$

$$R_R = \left(\left(2 * \sqrt{R_C}\right) + \sqrt{R_L}\right)^2 \quad (7.28)$$

If more than one line is missing, then line positions are not estimated and instead, the prediction from the previous frame is carried forward. In this case, in order to indicate the higher error, twice the standard deviation is assumed. R is calculated using:

$$R = 4 * R_{DEF} \quad (7.29)$$

where:
R_{DEF} is the measured value of R given in (7.25)

To measure the value of Q the off-line tracker was programmed to record the prediction used and the actual change in the estimates of ρ and θ. The error between the two was recorded for each frame. A program was then written in MATLAB to extract the prediction errors and calculate the variance for both θ and ρ. The values of Q were as follows and are used in all the Kalman filter tests:

θ: $Q=1.16^{o2}$

ρ: $Q=0.791 pixel^2$ (7.30)

Testing of the Kalman filter tracker was done in a similar manner to the rule-based tracker. It was first coded in MATLAB and run off-line on the same image sequences as were used for the rule-based tracker. Once the off-line version of the tracker was working, it was ported into C++ and incorporated into the test rig control software and Figure 7.43 shows its operation. The repeated acquisition process is as described in section 7.3, while the image processing, search of the AHT and third line prediction are again as described in section 7.2. The fuzzy

logic rules are as described in section 7.4. A flowchart showing the operation of the Kalman filter itself is shown in Figure 7.44.

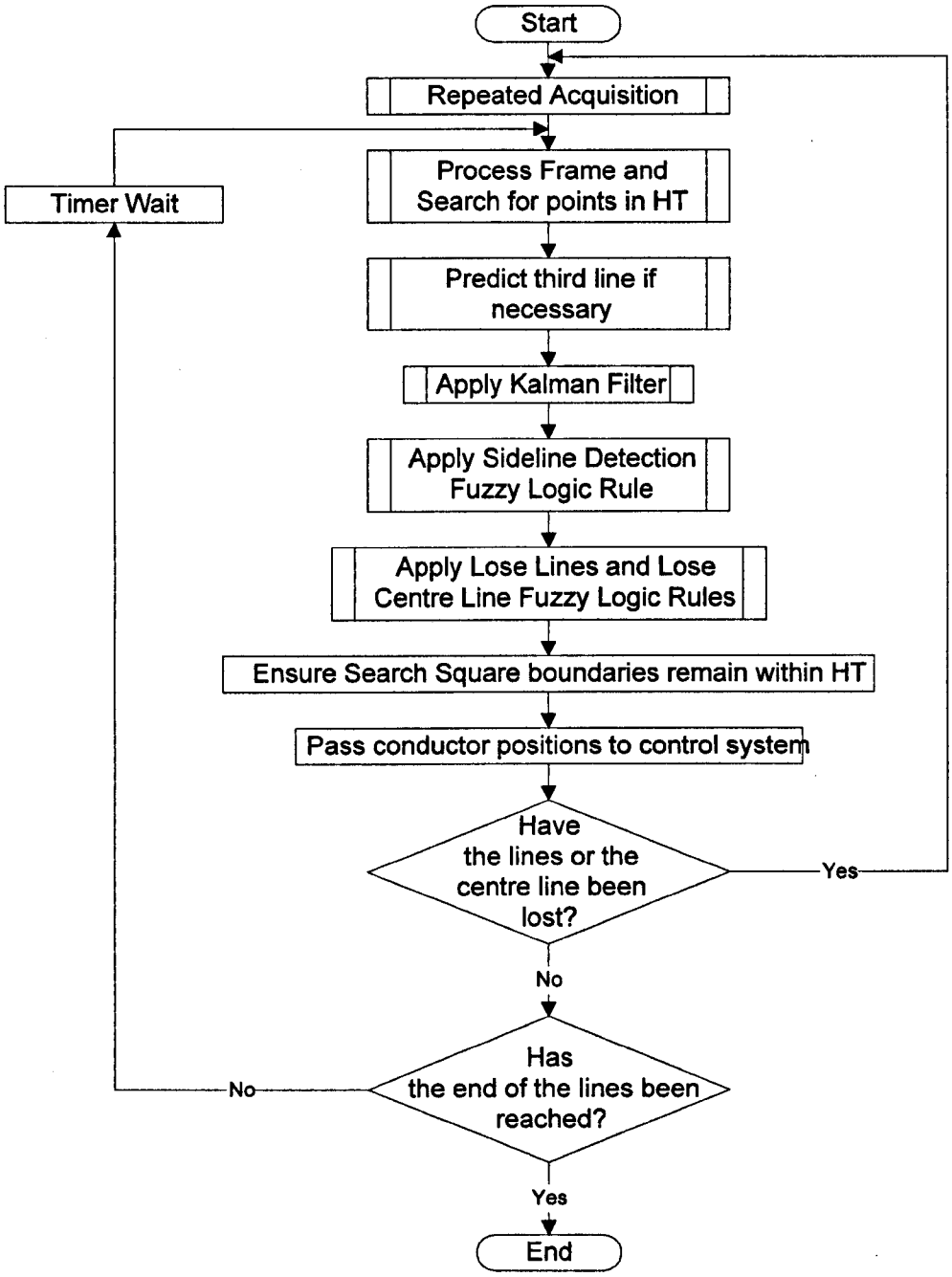


Figure 7.43: Flowchart for the Kalman Filter Tracker.

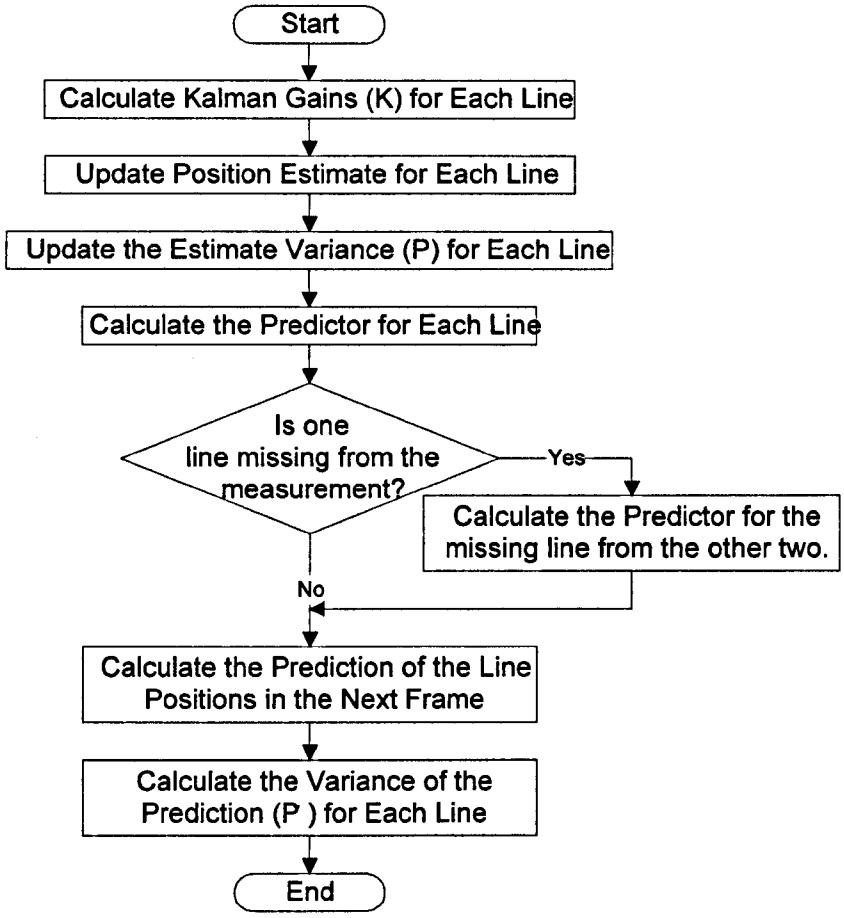


Figure 7.44: Flowchart for the Kalman Filter.

7.5.3 Results with Kalman Filter Tracker

In order to test the Kalman filter tracker, the UAV was flown along the lines, and its lateral displacement recorded. This was done with no wind, a wind gust and a large wind gust: again these gusts are of the same strength as used to test the early tracker (see section 7.2.3).

Figure 7.45 shows that the UAV tracks the lines well with the Kalman filter tracker. The results with the Kalman filter tracker (solid) are very similar to those obtained with the rule-based tracker (dash-dotted). It should be noted that the Kalman filter tracker does include the fuzzy logic rules. This would be expected because there is a plain background to the lines and so there is little noise and only small UAV lateral velocity, so there is little opportunity for the Kalman filter to show an advantage.

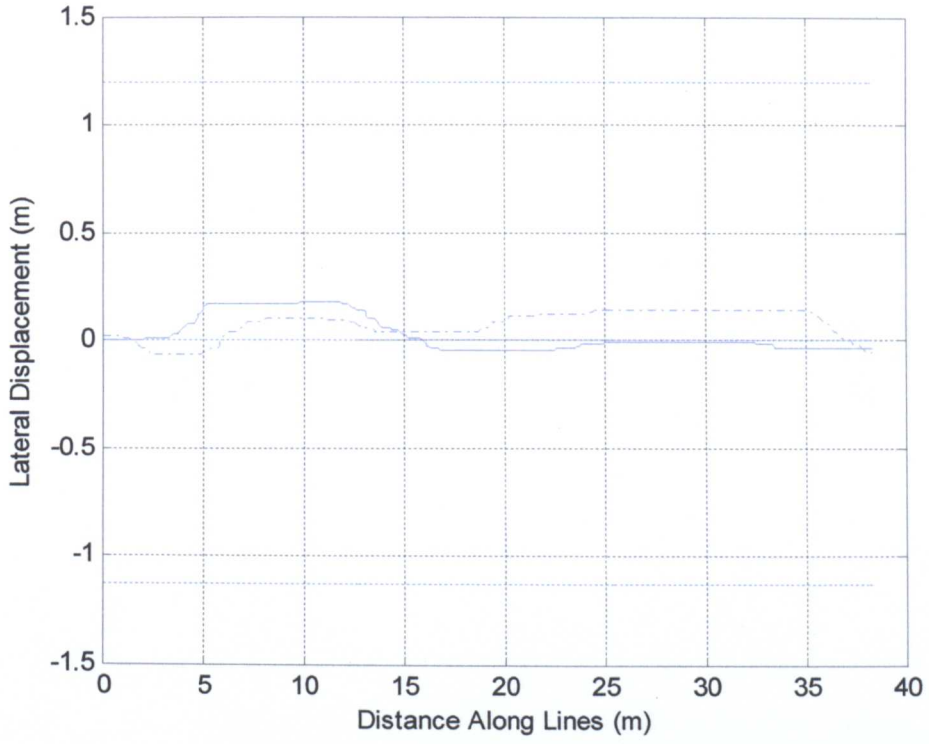


Figure 7.45: Lateral Displacement of the UAV with Kalman Filter Tracker Subject to No Wind.

In order to see the effect of the Kalman filter, it is necessary to look at the estimate of the lateral displacement of the UAV from the Kalman filter tracker, rather than the measured UAV position from the test rig, and compare it to the unfiltered lateral displacement estimate from the Rule-based tracker. Figure 7.46 shows that the output from the Kalman filter tracker (solid) generally has less high frequency content compared to the Rule-based tracker output (dash-dotted). This can be seen more clearly by zooming in on the area of the plot that doesn't include the sections affected by the poles at either end of the run, as shown in Figure 7.47.

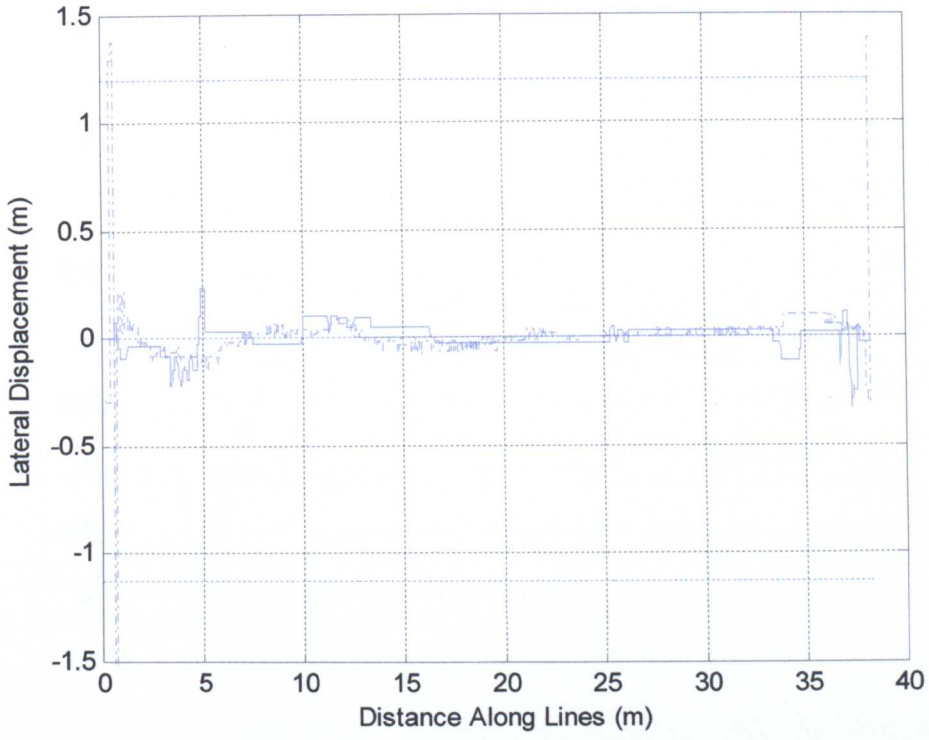


Figure 7.46: Estimated Lateral Displacement of the UAV from the Kalman Filter Tracker Output with the UAV Subject to No Wind.

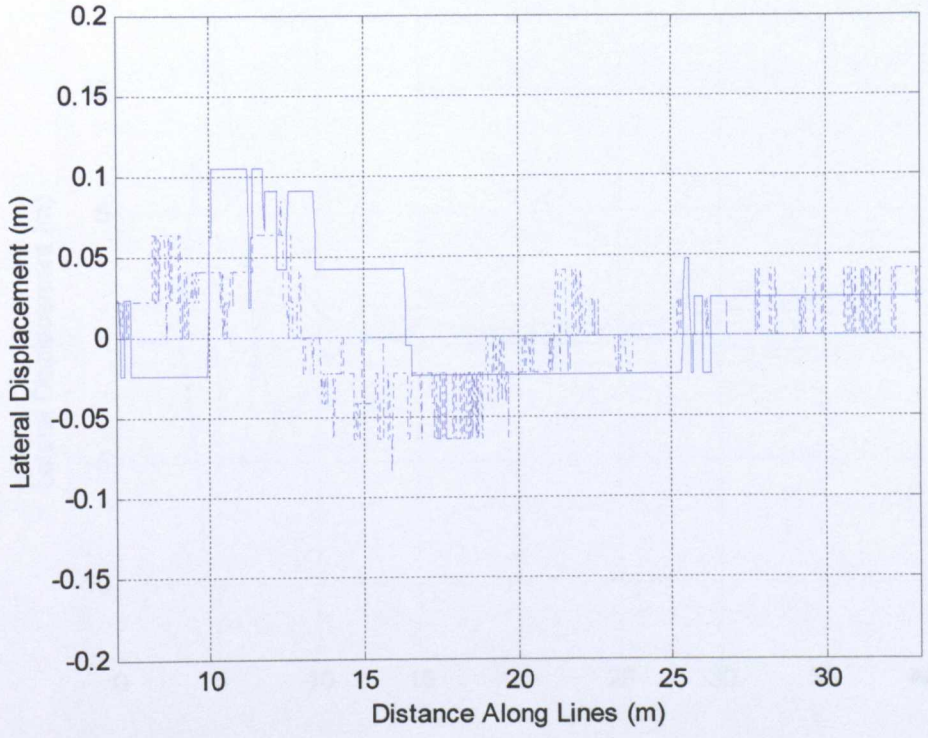


Figure 7.47: Estimated Lateral Displacement of the UAV from the Kalman Filter Tracker Output with the UAV Subject to No Wind (Zoomed In).

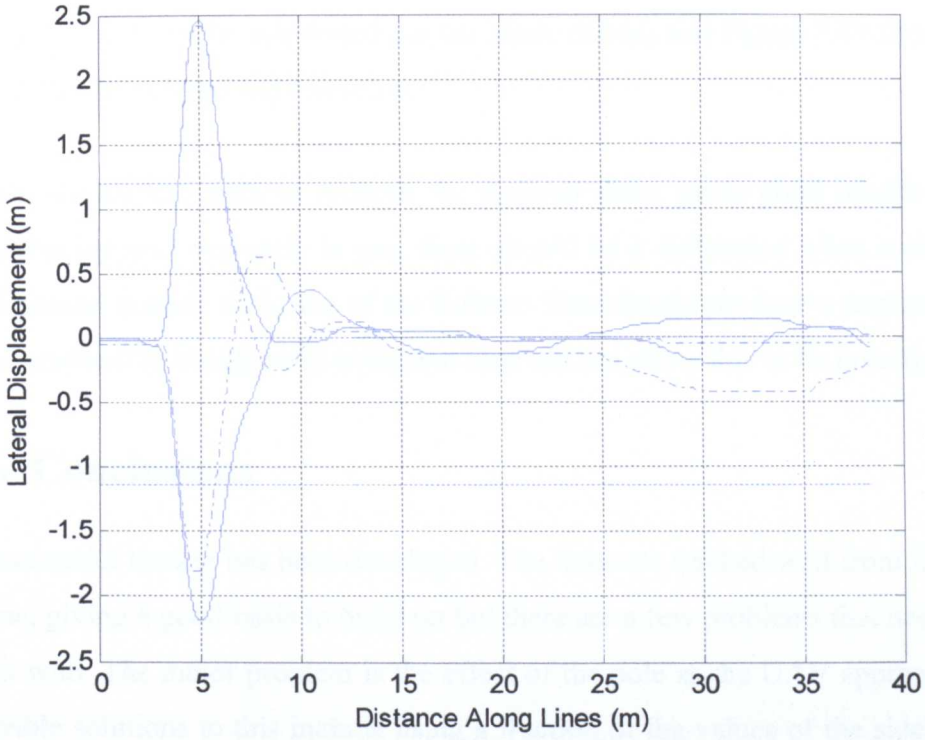


Figure 7.48: Lateral Displacement of the UAV with Kalman Filter Tracker in Response to a Pulse Wind Gust.

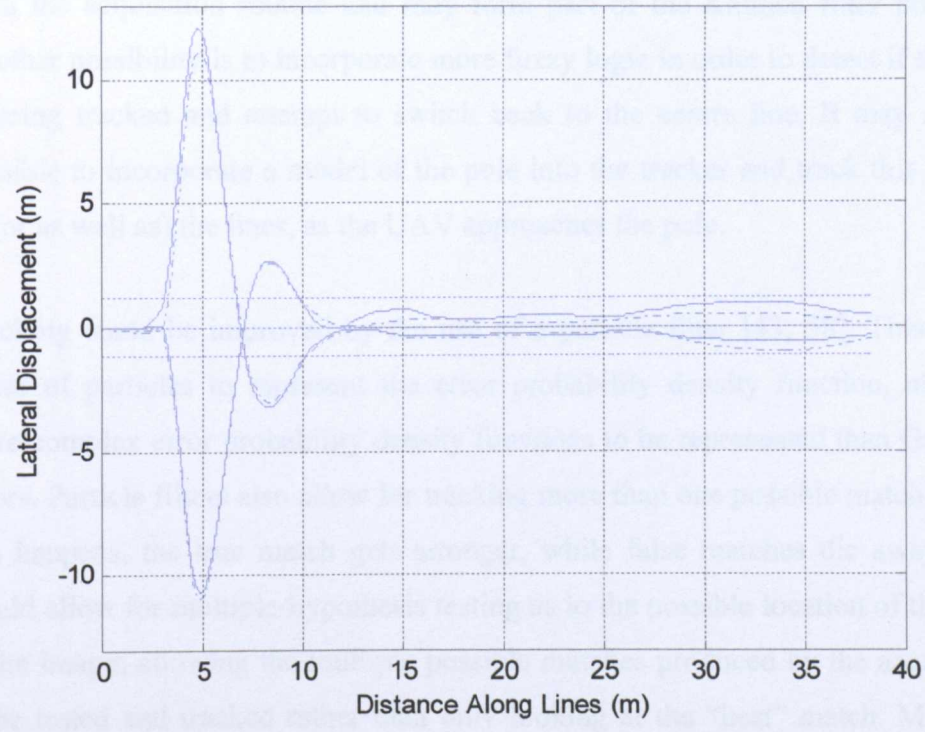


Figure 7.49: Lateral Displacement of the UAV with Kalman Filter Tracker in Response to a Large Pulse Wind Gust.

Figure 7.48 shows that the results when a wind gust is applied (solid) are similar to with those from the rule-based tracker (dash-dotted) and Figure 7.49 shows that this is true for a large wind gust too.

While the tracker, with or without the Kalman filter, gives good results on the plain background currently in use, there should be a difference when a cluttered background is used. Inclusion of the Kalman filter should produce a tracker that is more resilient to background noise, but time did not allow this to be investigated.

7.6 Conclusions

A successful tracker has been developed. The lines are tracked well from frame to frame, giving a good basis to build on but there are a few problems that need to be dealt with. The major problem is the effect of the pole as the UAV approaches it. Possible solutions to this include using a fraction of the values of the sidelines in the prediction of the centre line. This would help as it would tend to keep the line pattern more symmetrical; when the tracker locks onto the pole, the line pattern becomes quite non-symmetric. This could make use of the symmetry measure from the acquisition routine and may form part of the Kalman filter predictor. Another possibility is to incorporate more fuzzy logic in order to detect if the pole is being tracked and attempt to switch back to the centre line. It may also be possible to incorporate a model of the pole into the tracker and track this instead of (or as well as) the lines, as the UAV approaches the pole.

Tracking could be improved by the use of a particle filter [43, 58]. This uses a series of particles to represent the error probability density function, allowing more complex error probability density functions to be represented than Gaussian errors. Particle filters also allow for tracking more than one possible match. When this happens, the true match gets stronger, while false matches die away. This would allow for multiple-hypothesis testing as to the possible location of the lines in the image; allowing the multiple possible matches produced by the acquisition to be tested and tracked rather than only looking at the "best" match. Multiple-hypothesis testing should also help when the tracker is used in the real environment where the background contains clutter. The effect of a cluttered

background needs to be investigated. Other possible lines of investigation include the affine techniques described by Shapiro [59], more fuzzy logic, neural networks [60] and the use of optical flow to detect the upcoming pole.

Chapter 8 Multi-axis Control

8.1 Introduction

Chapter 7 discussed control of the lateral displacement and height of the UAV using visual servoing. The geometric analysis in Chapter 4 showed that lateral displacement, yaw and roll could be measured from the position of the centre line in the image while the height and pitch could be measured from the average distance between the outer lines and the centre line. Chapter 4 also showed that in order to track lateral displacement, yaw and roll, two cameras were required. In this chapter, control of multiple axes is discussed. This will focus on the use of two cameras to provide information on the lateral displacement, yaw and roll of the UAV.

In order to do this the two-camera assembly was fitted to the test-rig and the multiple computer version of the tracking software was used, as described in section 5.6. The vision part of this software used the Kalman filter based tracker described in section 7.5, while the control part of the software used the UAV model described in Chapter 3. The equations used to extract the lateral displacement, yaw and roll are described in this chapter. The code for the control and vision parts of the software are given in Appendix D.

In order to build up to measuring all three axes from vision, each axis was added in turn. To start with, the system was tested with only lateral displacement tracking. Lateral displacement was extracted from the forward camera using the equation (7.1) as used in Chapter 7. As the twin-camera mount is being used, the values of X_θ and X_ρ are taken from section 4.3 rather than section 4.2.2. Testing only the lateral displacement, initially, was done to show that the system worked with the two-camera configuration as it did with one camera. Following this, the system was expanded to extract both the lateral displacement and yaw from the forward camera. This allowed control of both the lateral displacement and yaw. As the UAV model from Chapter 3 doesn't include a model for yaw, it was modelled as a first order system. Finally, the system was expanded to use input from both cameras to give the lateral displacement, yaw and roll. Note that the

test-rig doesn't have a roll axis so this variable can't be controlled. An additional complication is that on the actual UAV the same actuator will control roll and lateral displacement (it is under-actuated). In this case, the lateral displacement and yaw were controlled while the roll measurement was simply recorded. This allowed the output to be assessed as to whether it could be used to control the roll axis.

8.2 Lateral Displacement Control

8.2.1 Design

In the first test, only the lateral displacement, X, is measured using images from the forward camera of the twin-camera mount. The measure of X is estimated from the forward image using (8.1)

$$X = \frac{X_\theta \theta + X_\rho \rho}{2} \quad (8.1)$$

where X_θ and X_ρ are $\frac{1}{\frac{\partial \theta_F}{\partial X}}$ and $\frac{1}{\frac{\partial \rho_F}{\partial X}}$.

The θ and ρ scaling factors are obtained from the analysis in section 4.3. Substituting these values into (8.1) gives:

$$X = \frac{0.0401\theta - 0.0314\rho}{2} \quad (8.2)$$

Lateral displacement using the twin-camera assembly could then be tested.

8.2.2 Results

In order to test the lateral displacement tracking, the same tests were used as in Chapter 7. First the UAV was flown along the line subject to no wind gust. This was then repeated with a small and large wind gust.

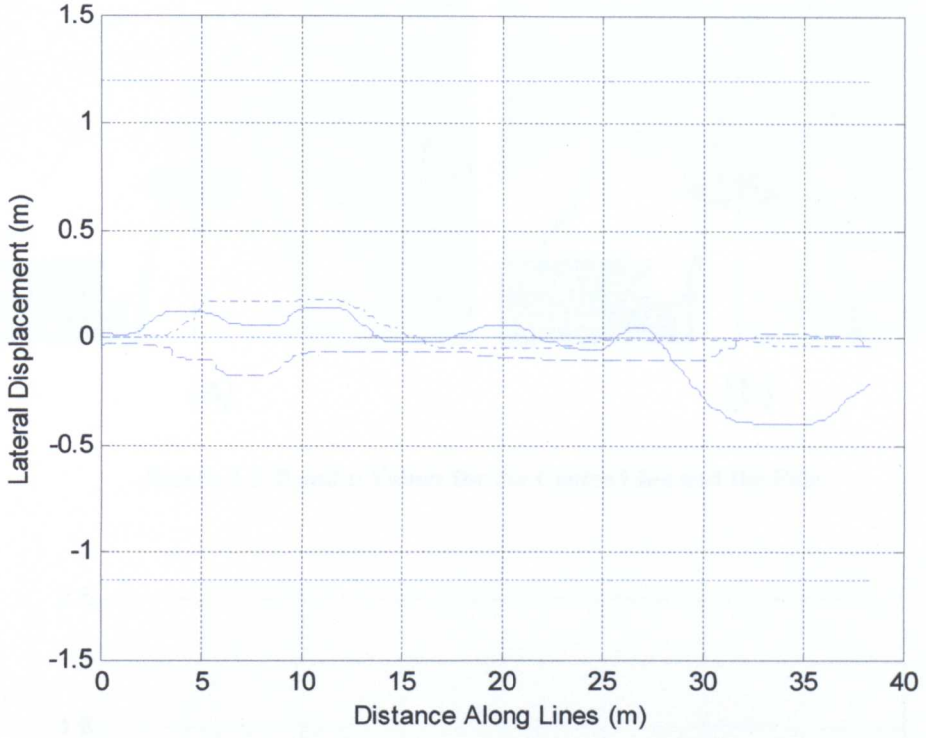


Figure 8.1: Lateral Displacement of the UAV with Kalman Filter Tracker Subject to No Wind.

We can see from Figure 8.1 that for most of the length of the line section tracking is good with the twin-camera setup (solid) although the effect of switching to the pole at the end of the run is considerably worse than with the single camera result from section 7.5.3 (dash-dotted). Figure 8.2 shows the effect of mistaking the pole for the centre line. The values of both θ and ρ are affected by tracking the pole, although the effect on θ is more significant as its sign changes, which introduces a certain amount of positive feedback. The position estimate from the twin camera mount comes more from the θ value than the position estimate from the single camera, hence the larger effect. In order to show that the deviation at the end of the run is caused by the pole, it was covered such as to blend in with the background. The UAV was then flown along the line again. Figure 8.1 shows that the UAV doesn't deviate at the end of the line (dashed).

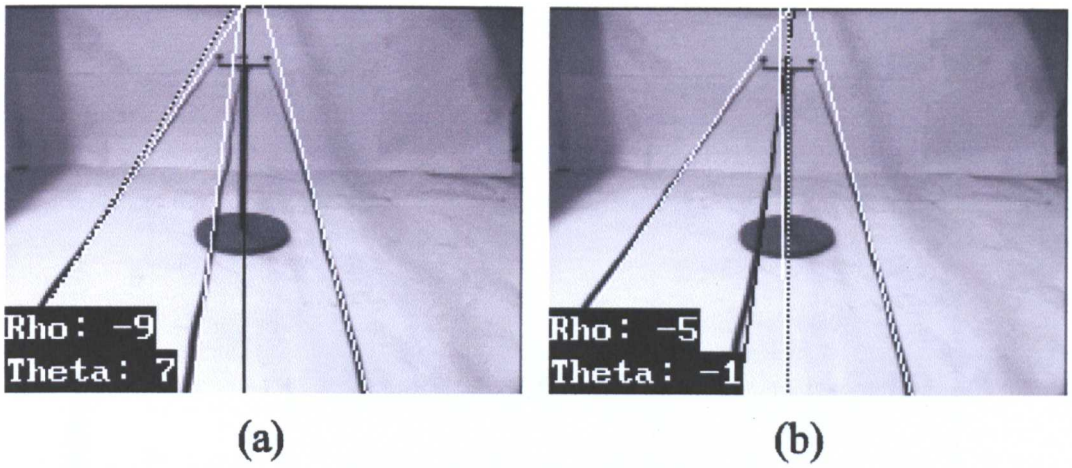


Figure 8.2: θ and ρ Values for the Centre Line and the Pole.

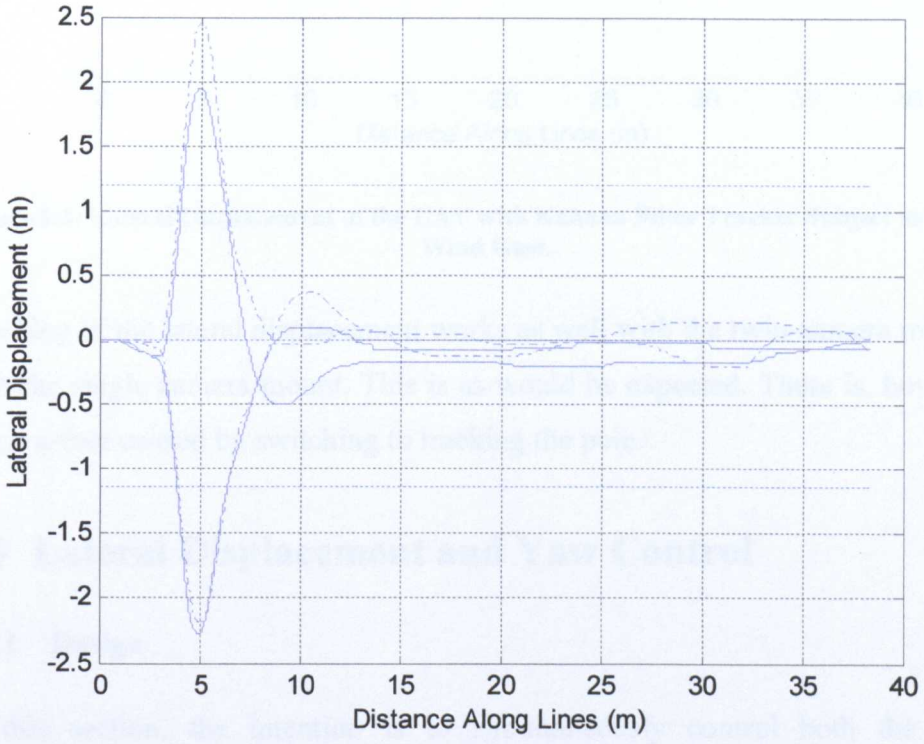


Figure 8.3: Lateral Displacement of the UAV with Kalman Filter Tracker Subject to a Pulse Wind Gust.

Figure 8.3 shows that with the small wind gust, the results for the two-camera setup with the poles covered (solid) match well with the one camera system (dash-dotted) with a pulse wind gust. Figure 8.4 shows a similar result with a large pulse wind gust.

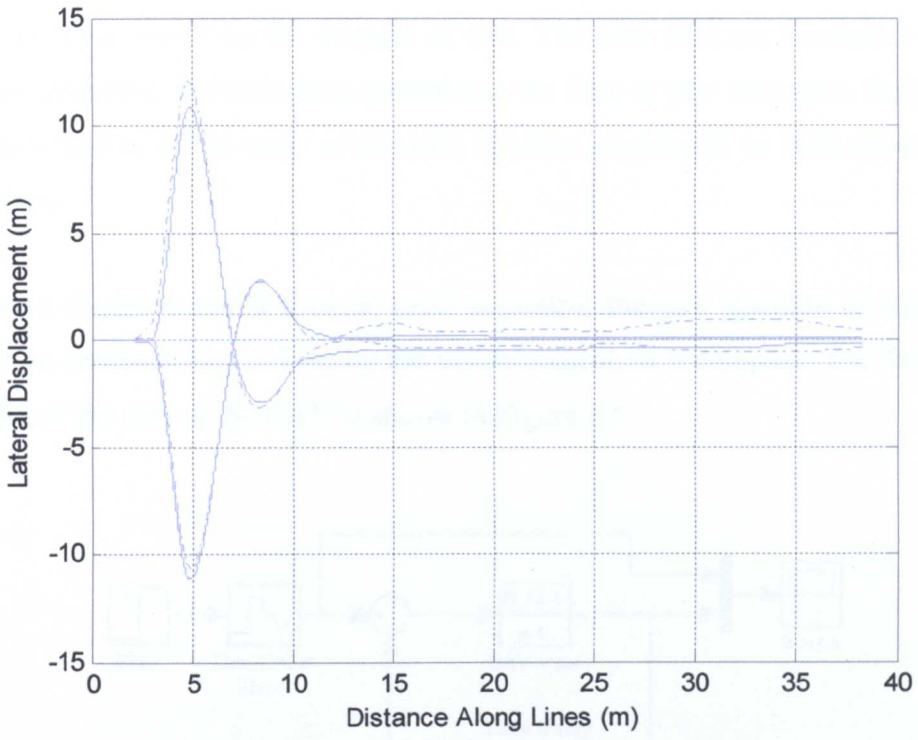


Figure 8.4: Lateral Displacement of the UAV with Kalman Filter Tracker Subject to a Large Wind Gust.

Tracking of the lateral displacement works as well with the twin-camera mount as with the single camera mount. This is as would be expected. There is, however a larger effect caused by switching to tracking the pole.

8.3 Lateral Displacement and Yaw Control

8.3.1 Design

In this section, the intention is to simultaneously control both the lateral displacement, X, and the yaw, α, of the UAV. In order to do this, both the α and X need to be extracted from the image. Also, it is necessary to model the yaw of the rotorcraft.

8.3.1.1 Yaw Model

The yaw of the UAV would be provided by adjusting the relative speeds of the two rotor blades. Unfortunately, the data required to model the yaw of the UAV was not available. However, the speed of the yaw should be proportional to the

difference in the speeds of the rotor blades, over short periods of time. Hence the yaw position would be the integral of this. The yaw axis can be modelled as a single integrator. It would be expected that the time to yaw through a fairly small angle would be of the order of seconds. A better model will be built when data is available.

Position feedback could then be used to control the yaw position of the UAV. Position feedback is provided by the vision control, or a compass. The first order model of the yaw of the UAV is shown in Figure 8.5

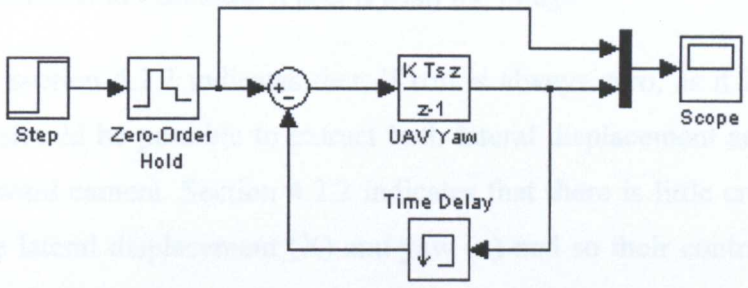


Figure 8.5: First Order Yaw Model.

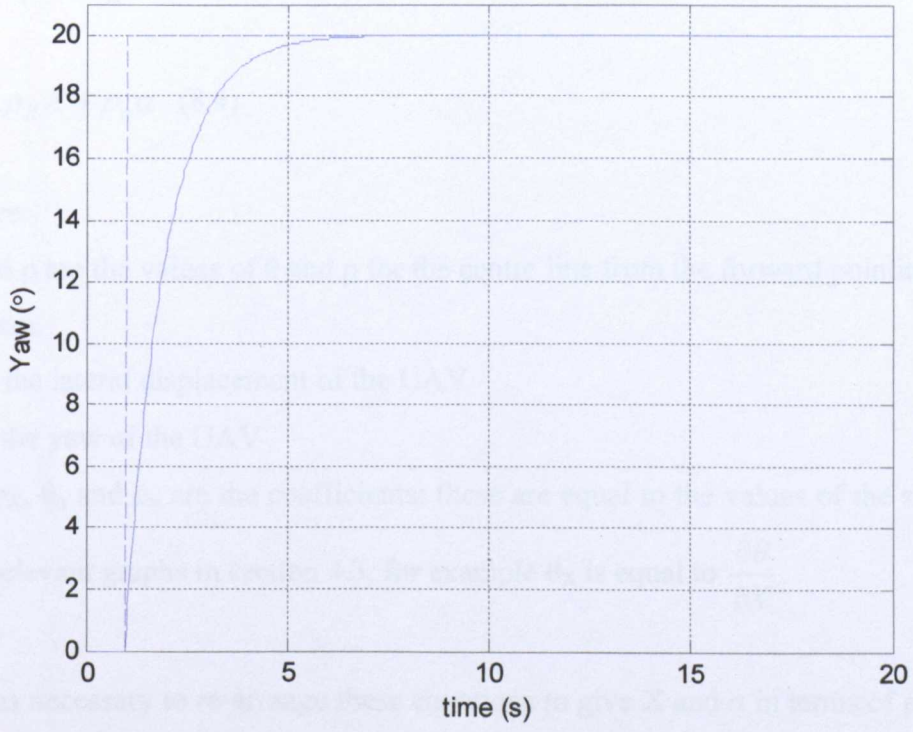


Figure 8.6: Yaw Model Position Response.

Figure 8.6 shows the response of the first order yaw model (solid) to a step input (dashed). The first order yaw model was coded into the control part of the test rig software (ControlThread.cpp: see Appendix D)

It should be noted that as the UAV yaws, its heading will change and so the direction of lateral displacement on the rig will change. As the yaw is expected to stay at around zero, the lateral displacement was kept aligned with the X axis of the test rig rather than rotating it with the yaw of the UAV. This is something that will have to be done in the future.

8.3.1.2 Equations to Calculate X and α from the Image

Analysis in section 4.2.2 indicates that, if roll is always zero, as it is on the test rig, then it should be possible to extract both lateral displacement and yaw from just the forward camera. Section 4.2.2 indicates that there is little cross coupling between the lateral displacement (X) and yaw (α) and so their contributions to θ and ρ can be added together. The values of ρ and θ for any given X and α are given in (8.3) and (8.4).

$$\theta = \theta_X X + \theta_\alpha \alpha \quad (8.3)$$

$$\rho = \rho_X X + \rho_\alpha \alpha \quad (8.4)$$

where:
θ and ρ are the values of θ and ρ for the centre line from the forward pointing camera.
X is the lateral displacement of the UAV
α is the yaw of the UAV
θ_X, ρ_X, θ_α and ρ_α are the coefficients: these are equal to the values of the slope of the relevant graphs in section 4.3, for example θ_X is equal to $\frac{\partial \theta}{\partial X}$.

It was necessary to re-arrange these equations to give X and α in terms of ρ and θ. If (8.3) is re-arranged to give α this gives:

$$\alpha = \frac{\theta - \theta_X X}{\theta_\alpha} \quad (8.5)$$

Substituting (8.5) into (8.4) gives:

$$\rho = \rho_X X + \frac{\rho_\alpha}{\theta_\alpha}\theta - \frac{\rho_\alpha \theta_X}{\theta_\alpha} X \quad (8.6)$$

Re-arranging (8.6) gives:

$$X = \frac{1}{\rho_X - \frac{\rho_\alpha \theta_X}{\theta_\alpha}}\rho - \frac{\frac{\rho_\alpha}{\theta_\alpha}}{\rho_X - \frac{\rho_\alpha \theta_X}{\theta_\alpha}}\theta \quad (8.7)$$

Sub (8.7) into (8.3) gives:

$$\theta = \theta_X \left(\frac{1}{\rho_X - \frac{\rho_\alpha \theta_X}{\theta_\alpha}}\rho - \frac{\frac{\rho_\alpha}{\theta_\alpha}}{\rho_X - \frac{\rho_\alpha \theta_X}{\theta_\alpha}}\theta \right) + \theta_\alpha \alpha \quad (8.8)$$

Re-arranging (8.8) gives:

$$\alpha = -\frac{1}{\frac{\rho_X \theta_\alpha}{\theta_X} - \rho_\alpha}\rho + \frac{\frac{\rho_X}{\theta_X}}{\frac{\rho_X \theta_\alpha}{\theta_X} - \rho_\alpha}\theta \quad (8.9)$$

This gives equations for X and α. Before these can be programmed into the control software, the values of the coefficients need to be determined. From analysis in section 4.3 the values are:

$\theta_X = 24.9$

$\rho_X = -30.7$

$\theta_\alpha = 1.30$

$\rho_\alpha = -4.66$

It should be noted that while θ_α is almost zero for the single camera setup, for the twin camera, it has a significant non-zero value. Putting these values into (8.7) and (8.9) gives:

$$X = 0.0171\rho + 0.0612\theta \quad (8.10)$$

$$\alpha = -0.328\rho - 0.403\theta \quad (8.11)$$

These equations and the first order UAV model were added to the control software. This could then be tested.

8.3.2 Results

First the UAV was flown along the line, subject to no wind gust. Following this, the test was repeated using a small and then a large wind gust.

Figure 8.7 shows that for the first part of the run, both the lateral displacement and the yaw track quite well (solid), although there is a slight offset on the lateral position. As the UAV approaches the pole at the end of the line, two-axis tracker causes a larger lateral displacement to occur. This is because the tracker locks onto the pole and the erroneous ρ and θ values cause both lateral displacement and yaw tracking to be affected. The rotorcraft's yaw then compounds the effect on the lateral displacement. Again, in order to show that it is the pole that causes the deviation, the run was repeated with the pole covered up (dash-dotted).

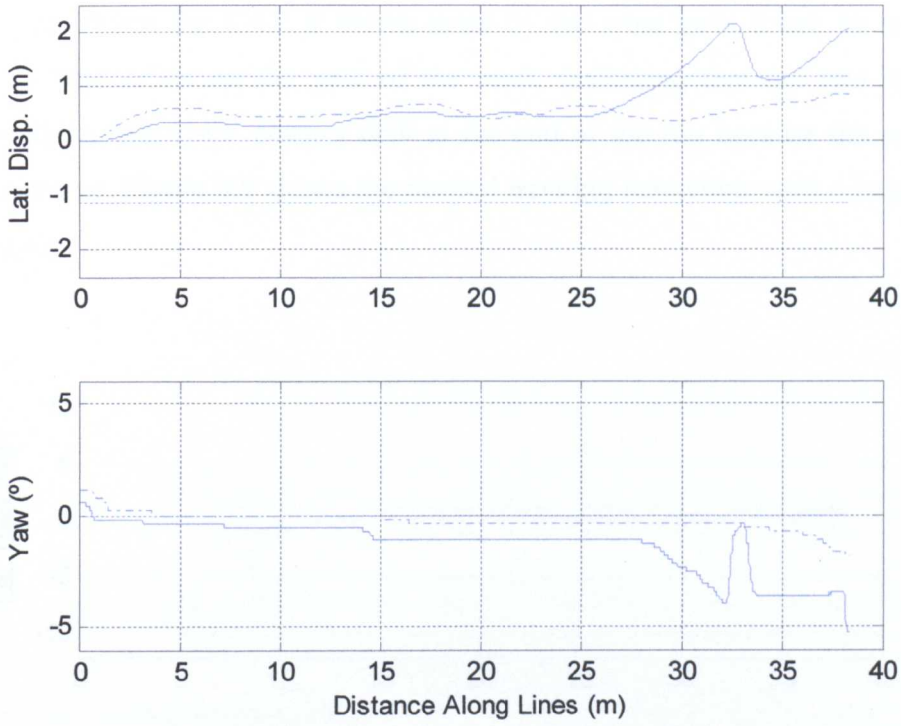


Figure 8.7: Lateral Displacement and Yaw of the UAV with Kalman Filter Tracker Subject to No Wind.

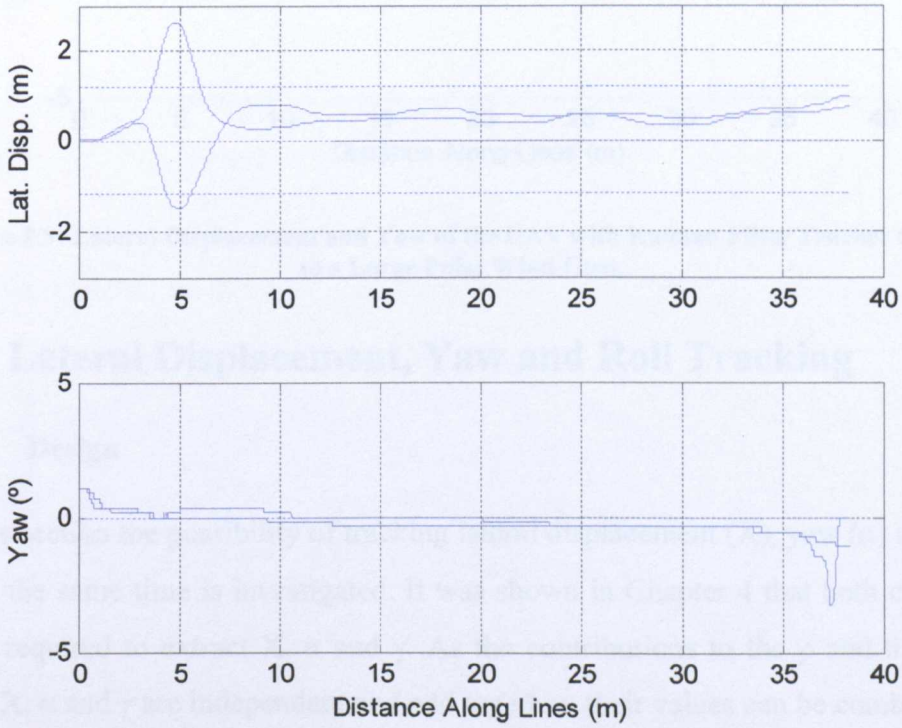


Figure 8.8: Lateral Displacement and Yaw of the UAV with Kalman Filter Tracker Subject to a Pulse Wind Gust.

Figure 8.8 shows that the UAV is able to restore lock onto the line after a pulse wind gust. When the UAV is blown aside by the wind gust, it can be seen that there is little effect on the yaw of the craft, indicating that the two axes are independent. The UAV doesn't drift at the end of the run because the pole had been covered. Figure 8.9 shows the desired tracking behaviour with a large pulse wind gust.

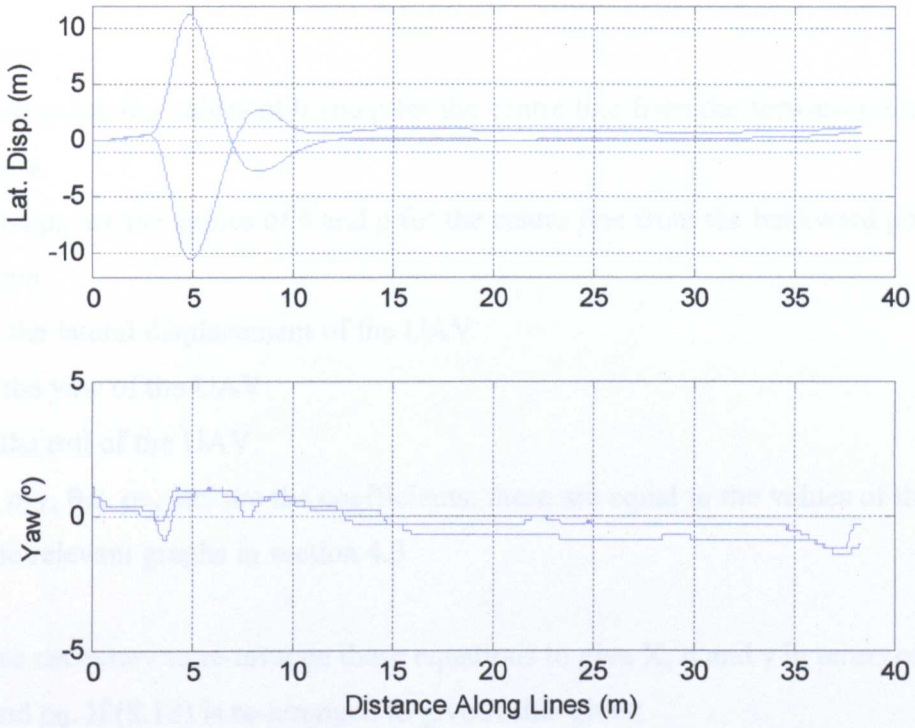


Figure 8.9: Lateral Displacement and Yaw of the UAV with Kalman Filter Tracker Subject to a Large Pulse Wind Gust.

8.4 Lateral Displacement, Yaw and Roll Tracking

8.4.1 Design

In this section the possibility of tracking lateral displacement (X), yaw (α) and roll (γ) at the same time is investigated. It was shown in Chapter 4 that both cameras were required to extract X, α and γ. As the contributions to the ρ and θ values from X, α and γ are independent and add together, their values can be combined to give ρ and θ for each camera. The values of ρ and θ for the forward facing camera for any given X, α and γ are given in (8.12) and (8.13) while those for the backward facing camera are given in (8.14) and (8.15).

$$\rho_F = \rho_{FX} X + \rho_{F\alpha} \alpha \quad (8.12)$$

$$\theta_F = \theta_{FX} X + \theta_{F\alpha} \alpha + \theta_{F\gamma} \gamma \quad (8.13)$$

$$\rho_B = \rho_{BX} X + \rho_{B\alpha} \alpha + \rho_{B\gamma} \gamma \quad (8.14)$$

$$\theta_B = \theta_{BX} X + \theta_{B\alpha} \alpha + \theta_{B\gamma} \gamma \quad (8.15)$$

where:

θ_F and ρ_F are the values of θ and ρ for the centre line from the forward pointing camera.

θ_B and ρ_B are the values of θ and ρ for the centre line from the backward pointing camera.

X is the lateral displacement of the UAV

α is the yaw of the UAV

γ is the roll of the UAV

θ_{FX}, ρ_{FX}, $\theta_{F\alpha}$, $\rho_{F\alpha}$ etc. are the coefficients: these are equal to the values of the slope of the relevant graphs in section 4.3

It was necessary to re-arrange these equations to give X, α and γ in terms of θ_F, ρ_F, θ_B and ρ_B. If (8.12) is re-arranged to give X this gives:

$$X = \frac{\rho_F - \rho_{F\alpha} \alpha}{\rho_{FX}} \quad (8.16)$$

Substituting (8.16) into (8.13) gives:

$$\theta_F = \theta_{FX} \left(\frac{\rho_F - \rho_{F\alpha} \alpha}{\rho_{FX}} \right) + \theta_{F\alpha} \alpha + \theta_{F\gamma} \gamma \quad (8.17)$$

Re-arranging for γ gives:

$$\gamma = \frac{1}{\theta_{F\gamma}} \theta_F - \frac{\theta_{FX}}{\theta_{F\gamma}} \left(\frac{\rho_F - \rho_{F\alpha} \alpha}{\rho_{FX}} \right) + \frac{\theta_{F\alpha}}{\theta_{F\gamma}} \alpha \quad (8.18)$$

Substituting (8.18) and (8.16) into (8.14) gives:

$$\rho_B = \rho_{BX}\left(\frac{\rho_F - \rho_{F\alpha}\alpha}{\rho_{FX}}\right) + \rho_{B\alpha}\alpha + \rho_{B\gamma}\left(\frac{1}{\theta_{F\gamma}}\theta_F - \frac{\theta_{FX}}{\theta_{F\gamma}}\left(\frac{\rho_F - \rho_{F\alpha}\alpha}{\rho_{FX}}\right) + \frac{\theta_{F\alpha}}{\theta_{F\gamma}}\alpha\right) \quad (8.19)$$

Re-arranging (8.19) for α gives:

$$\alpha = \frac{\rho_B + \left(\frac{\rho_{B\gamma}\theta_{FX}}{\rho_{FX}\theta_{F\gamma}} - \frac{\rho_{BX}}{\rho_{FX}}\right)\rho_F - \frac{\rho_{B\gamma}}{\theta_{F\gamma}}\theta_F}{\rho_{B\alpha} - \frac{\rho_{BX}\rho_{F\alpha}}{\rho_{FX}} + \frac{\rho_{B\gamma}\theta_{FX}\rho_{F\alpha}}{\theta_{F\gamma}\rho_{FX}} - \frac{\rho_{B\gamma}\theta_{F\alpha}}{\theta_{F\gamma}}} \quad (8.20)$$

Substituting (8.20) into (8.12) gives:

$$\rho_F = \rho_{FX}X + \rho_{F\alpha}\left(\frac{\rho_B + \left(\frac{\rho_{B\gamma}\theta_{FX}}{\rho_{FX}\theta_{F\gamma}} - \frac{\rho_{BX}}{\rho_{FX}}\right)\rho_F - \frac{\rho_{B\gamma}}{\theta_{F\gamma}}\theta_F}{\rho_{B\alpha} - \frac{\rho_{BX}\rho_{F\alpha}}{\rho_{FX}} + \frac{\rho_{B\gamma}\theta_{FX}\rho_{F\alpha}}{\theta_{F\gamma}\rho_{FX}} - \frac{\rho_{B\gamma}\theta_{F\alpha}}{\theta_{F\gamma}}}\right) \quad (8.21)$$

Re-arranging (8.21) for X gives:

$$X = \frac{-\rho_B + \left(\frac{\rho_{B\alpha}}{\rho_{F\alpha}} - \frac{\rho_{B\gamma}\theta_{F\alpha}}{\rho_{F\alpha}\theta_{F\gamma}}\right)\rho_F + \frac{\rho_{B\gamma}}{\theta_{F\gamma}}\theta_F}{\frac{\rho_{B\alpha}\rho_{FX}}{\rho_{F\alpha}} - \rho_{BX} + \frac{\rho_{B\gamma}\theta_{FX}}{\theta_{F\gamma}} - \frac{\rho_{FX}\rho_{B\gamma}\theta_{F\alpha}}{\theta_{F\gamma}\rho_{F\alpha}}} \quad (8.22)$$

Substituting (8.22) and (8.20) into (8.15) and re-arranging for γ gives:

$$\gamma = \frac{1}{\theta_{B\gamma}}\theta_B - \frac{\theta_{BX}}{\theta_{B\gamma}}\left(\frac{-\rho_B + \left(\frac{\rho_{B\alpha}}{\rho_{F\alpha}} - \frac{\rho_{B\gamma}\theta_{F\alpha}}{\rho_{F\alpha}\theta_{F\gamma}}\right)\rho_F + \frac{\rho_{B\gamma}}{\theta_{F\gamma}}\theta_F}{\frac{\rho_{B\alpha}\rho_{FX}}{\rho_{F\alpha}} - \rho_{BX} + \frac{\rho_{B\gamma}\theta_{FX}}{\theta_{F\gamma}} - \frac{\rho_{FX}\rho_{B\gamma}\theta_{F\alpha}}{\theta_{F\gamma}\rho_{F\alpha}}}\right)$$

$$-\frac{\theta_{B\alpha}}{\theta_{B\gamma}}\left(\frac{\rho_B+\left(\frac{\rho_{B\gamma}\theta_{FX}}{\rho_{FX}\theta_{F\gamma}}-\frac{\rho_{BX}}{\rho_{FX}}\right)\rho_F-\frac{\rho_{B\gamma}}{\theta_{F\gamma}}\theta_F}{\rho_{B\alpha}-\frac{\rho_{BX}\rho_{F\alpha}}{\rho_{FX}}+\frac{\rho_{B\gamma}\theta_{FX}\rho_{F\alpha}}{\theta_{F\gamma}\rho_{FX}}-\frac{\rho_{B\gamma}\theta_{F\alpha}}{\theta_{F\gamma}}}\right) \quad (8.23)$$

This gives equations for X, α and γ. Before these can be programmed into the control software, the values of the coefficients need to be determined. From analysis in section 4.3 the values are:

θ_{FX}=24.9	$\theta_{F\alpha}$=1.30	$\theta_{F\gamma}$=0.93
ρ_{FX}=-30.7	$\rho_{F\alpha}$=-4.66	$\rho_{F\gamma}$=0
θ_{BX}=-24.9	$\theta_{B\alpha}$=1.30	$\theta_{B\gamma}$=-1.36
ρ_{BX}=30.7	$\rho_{B\alpha}$=-4.66	$\rho_{B\gamma}$=2.74

Table 8.1: Lateral Displacement, Yaw Roll Co-efficients.

Putting these values into (8.22), (8.20) and (8.23) gives:

$$X = 0.0754\rho_B - 0.137\rho_F - 0.222\theta_F \quad (8.24)$$

$$\alpha = -0.496\rho_B + 0.69\rho_F + 1.46\theta_F \quad (8.25)$$

$$\gamma = -0.735\theta_B - 1.85\rho_B + 3.17\rho_F + 5.46\theta_F \quad (8.26)$$

These equations were added to the control software, which was then tested.

8.4.2 Results

Again the UAV was first flown along the line, subject to no wind gust and with the poles covered. The lateral displacement and yaw were controlled by the vision feedback, while the roll can only be measured from the images. Ideally we would expect to get a constant zero measurement for the roll, while the lateral displacement and yaw should track as before.

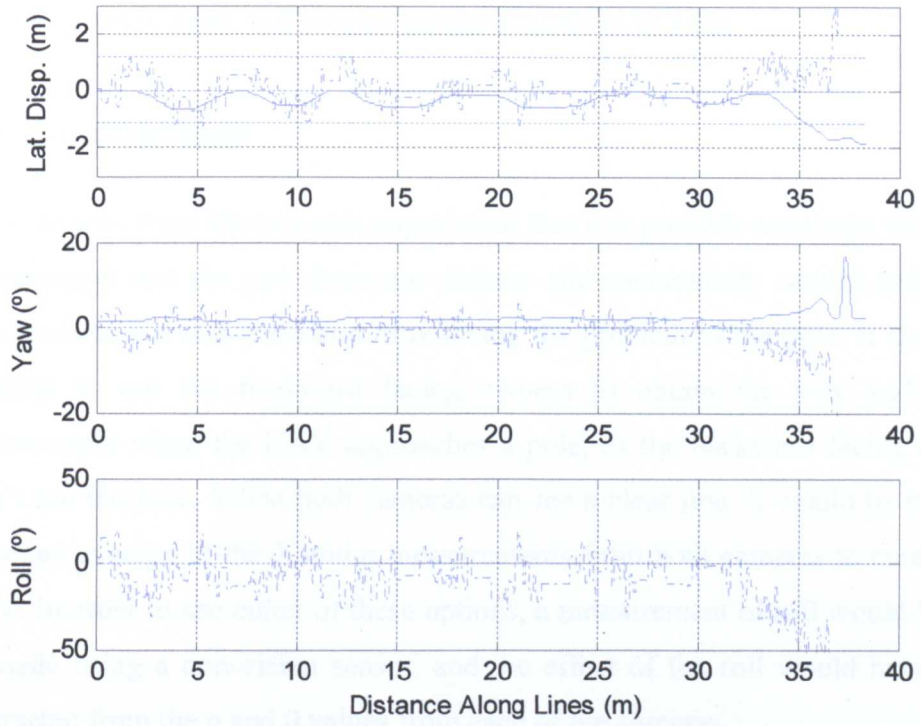


Figure 8.10: Lateral Displacement, Yaw and Roll of the UAV with Kalman Filter Tracker Subject to No Wind.

Figure 8.10 shows the lateral position and yaw of the UAV under vision control (solid) and the estimate of the lateral displacement, yaw and roll of the UAV from the image processing (dash-dotted). The UAV tracks the lines quite well. There is some oscillation in the lateral displacement of the UAV. Yaw is around zero, as would be expected. The estimates of the lateral displacement and yaw from the image processing have more noise than the measured positions from the rig. There is also a reasonable amount of noise on the roll measurement. At the very end of the run, the tracker loses the lines, as there aren't more lines following on from the pole, hence the deviation from the lines at the very end of the run.

Possible causes of the oscillation in the lateral displacement and the noise are that separating three measurements from the θ and ρ values and using two cameras brings in more noise sources. Also there will be a larger error in the $\theta_{F\gamma}$, $\theta_{B\gamma}$ and $\rho_{B\gamma}$ values, compared to the values of the other coefficients because there is no roll axis on the rig and it was necessary to make the measurements by hand. Changing the lateral displacement causes opposite effect on the forward and backward image while roll causes opposite but unequal effects on the two images. It may,

therefore be the case that these two axes are not as easy to separate as separating the yaw measurement, and so may be more sensitive to noise.

8.5 Conclusions

It can be seen from the two-axis experiment that it is possible to extract the lateral displacement and the yaw from one camera and successfully control both axes. This leads to a possible method of reducing the problem of the pole. It should be possible to use the backward facing camera to obtain the yaw and lateral displacement when the UAV approaches a pole, as the backward facing camera won't see the pole. When both cameras can see a clear line, it would be possible to use an average of the X and α measurements from both cameras to control the UAV. In order to use either of these options, a measurement of roll would have to be made using a non-vision sensor, and the effect of the roll would have to be subtracted from the ρ and θ values from each of the cameras.

The three-axis experiment indicates that it should be possible to control all three axes from vision, if both cameras are working, although the results are noisier than controlling just yaw and lateral displacement. In order to fully test this, the test-rig would need the addition of a roll axis in order to simulate the roll and thus see the effect on the image. In addition, as the same actuator will control both roll and lateral displacement on the UAV, there will, at times, need to be a non-zero roll demand, in order to stop the attempt to obtain zero roll affecting lateral displacement. A method of combining the lateral displacement and roll error signals to feed to the actuator on the UAV will need to be developed.

Chapter 9 Conclusions and Future Work

9.1 Conclusions

The project has produced encouraging results for the prospect of a visually guided UAV to inspect electricity distribution lines. It has been shown that the three lines do form a useful beacon for servoing the UAV. The project met its aims and has given a good foundation on which to build an improved tracker.

A model of the UAV has been developed, although validation tests will need to be done once a UAV has been constructed to see how well it conforms to the real UAV. The ducted fan type of UAV described suffers from a poor wind gust response and there is also the limitation that the model is of the laboratory demonstrator and so is more sensitive to wind than a full size UAV would be. The UAV also has the limitation of having a slow response time. The tracking results show this slow response, and indicate that a different design of UAV, with a faster response time will very likely be needed for this project. Changing to a different type of UAV would require a new UAV model but shouldn't significantly affect the operation of the vision system.

The test rig, on which experiments were performed, was constructed successfully, although this took quite a lot of time and work. Despite its small scale, it produces satisfactorily realistic image sequences of the overhead lines. The rig provides good three-axis movement but it is clear that additional axes would need to be added for future experiments. The addition of a roll axis and changing the pitch to a controllable axis, rather than just a manually settable axis would be necessary in order to better simulate the UAV.

The Aggregated Hough Transform was developed and shown to find the three lines well. Within the AHT there are often lines that are not associated with the overhead lines. These primarily come from the pole and the insulators on top of it, when they are prominent in the image. There are occasional lines from the

background. These don't generally cause a problem with tracking, as they tend to appear in areas of the AHT that the tracker doesn't search. The exception to this is the vertical pole, which did tend to cause tracking problems when it was not co-linear with the centre line. When more realistic backgrounds are used, the number of such lines will increase, and so it is predicted that tracking will be affected more.

The use of fuzzy logic rules and a Kalman filter provided successful tracking of the lines from frame to frame. Visual control of the lateral displacement of the UAV was demonstrated and the UAV was able to maintain alignment with the centre line and re-acquire the lines and return to the centre line after a wind gust.

In Chapter 8 simultaneous control of both lateral displacement and yaw was demonstrated. This showed that both could successfully be controlled by visual servoing with video from one camera. The possibility of extracting information about the roll of the UAV from vision, as well as the lateral displacement and yaw, was demonstrated using two cameras, although this gave a much noisier result than when only the lateral displacement and yaw were tracked. Consideration must also be given as to how to use the roll and lateral displacement signals, because the same actuator must control both the roll and lateral displacement axes. The experiment in section 7.4.4 showed that in addition to extracting the lateral position of the UAV it was also possible to simultaneously extract an estimate of the height above the lines and control both the lateral displacement and height of the UAV simultaneously. In order to combine this and the yaw and roll control, it would be necessary to add a height axis to the test rig as well as a roll and pitch axis.

It has been shown that the UAV will follow the lines and that it is possible to control multiple axes using vision feedback. Though the results of this project demonstrate the principle of controlling the UAV using visual servoing, there are significant problems that remain to be solved before commercial development is appropriate. At least in the short term, it is likely that other sensors that assist visual servoing will be needed. These could include ultrasonic, laser and millimetre wave radar sensors. Using these would give an associated power,

weight and cost penalty. Currently the system is run in an idealised environment with a plain background, with the exception of the test performed on the AVS. Tests will need to be done using cluttered backgrounds and eventually in the real world. This will almost certainly cause tracking to be less reliable than in the current idealised case, and so strategies to deal with this must be found.

9.2 Future Work

In order to take the project from its current early state to completion, a lot of work is going to be required. The problems of tracking the pole and tracking problems associated with realistic backgrounds will need to be solved. This could possibly be done by the use of fuzzy logic rules to detect a switch followed by an attempt to move back to the centre line. Alternatively, a percentage of the positions of the outer two lines, when both are present, could be included in the prediction of the centre line. A particle filter [58], the affine techniques described by Shapiro [59] and/or the use of neural networks [60] could also help with this. Template matching may also help with locating the pole in the image, which could allow the system to switch back to the centre line. A possible method for locating the pole in the image is given in [61]. The location of the pole top could also be used for servoing the UAV. An additional line of investigation is to look at the augmentation of information by other sensors, such as an electric field sensor to detect the position of the lines relative to the craft, roll and pitch sensors, a compass and radar combined with the machine vision method, thus improving the estimates of position and leading to improved control of the UAV. The use of a downward looking camera could be investigated. Because it would look directly downward, it wouldn't be affected by the distant pole. Primarily, this should give information on the lateral displacement and yaw of the UAV. Due to the direction this camera points, small values of roll and lateral displacement will produce the same apparent motion in the image. Consideration would be needed in to how to mount such a camera on to the UAV such that it has a clear view of the lines un-obscured by the power pick up.

It is know that the ducted fan type UAV that is planned as a laboratory demonstrator suffers from a poor wind gust response. It was chosen because it is

convenient to have a UAV that is dynamically stable and marginally statically stable. Further work is required to design a replacement for this UAV that is larger, so as to be useful for inspection purposes, has a better wind gust response and has a faster response time. It is also necessary to investigate how the effect of a wind gust scales with the size of the UAV. It is possible that using an inherently unstable UAV with a dedicated stabilising control system may help speed up the response to a wind gust and allow a critically damped response, rather than the current slow, under-damped response. Work also needs to be done to develop the power pick-up system.

The research effort reported in this thesis has made a significant contribution to the feasibility of the overall concept but it is clear that several other difficult areas of work must be undertaken before a practical system for power line inspection emerges.

Appendix A Geometric Analysis

A.1 Analysis for the Roll Axis

Applying equation (4.1) to a straight-line model of the centre conductor, placed at $X_W = 0$ and $Z_W = -Z_L$, where Z_L is the vertical height of the camera centre above the line, generates a corresponding line in the image. Applying the Hough Transform then gives ρ and θ as a function of the camera pose. The UAV that rolls by an angle γ. Assume that the vehicle is flying along the lines at constant speed, and so has a fixed pitch, β. Assume also that α and X_u are zero. Equation (4.1) becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\gamma & 0 & \sin\gamma & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\gamma & 0 & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ Y_W \\ -Z_L \\ 1 \end{bmatrix} \quad \text{(A.1)}$$

This becomes

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} Y_w \sin\beta\sin\gamma - Z_L \cos\beta\sin\gamma \\ Y_W \cos\beta + Z_L \sin\beta - \ell \\ Y_w \sin\beta\cos\gamma - Z_L \cos\beta\cos\gamma \\ -\frac{Y_W \cos\beta + Z_L \sin\beta - \ell}{\lambda} \end{bmatrix} \quad \text{(A.2)}$$

From (A.2), using (4.2), the image co-ordinates are given by:

$$x = -\lambda \frac{Y_w \sin\beta\sin\gamma - Z_L \cos\beta\sin\gamma}{Y_W \cos\beta + Z_L \sin\beta - \ell} \text{ and}$$

$$z = -\lambda \frac{Y_w \sin\beta\cos\gamma - Z_L \cos\beta\cos\gamma}{Y_W \cos\beta + Z_L \sin\beta - \ell} \quad \text{(A.3)}$$

Applying the Hough Transform (4.6) then gives:

$$\theta = \tan^{-1}\left(\frac{Y_w \sin\beta\sin\gamma - Z_L \cos\beta\sin\gamma}{Y_w \sin\beta\cos\gamma - Z_L \cos\beta\cos\gamma}\right) = \tan^{-1}\left(\frac{\sin\gamma}{\cos\gamma}\right) \text{ and}$$

$$\rho = -\lambda \frac{Y_w \sin\beta\sin\gamma - Z_L \cos\beta\sin\gamma}{Y_W \cos\beta + Z_L \sin\beta - \ell}\cos\gamma + \lambda \frac{Y_w \sin\beta\sin\gamma - Z_L \cos\beta\sin\gamma}{Y_W \cos\beta + Z_L \sin\beta - \ell}\sin\gamma$$

(A.4)

These simplify to

$$\theta = \gamma \text{ and } \rho = 0 \quad \text{(A.5)}$$

It can be seen that only θ changes with roll angle while ρ is identically zero.

A.2 Analysis for the Yaw Axis

Applying equation (4.1) to a straight-line model of the centre conductor, placed at $X_W = 0$ and $Z_W = -Z_L$, where Z_L is the vertical height of the camera centre above the line, generates a corresponding line in the image. Applying the Hough Transform then gives ρ and θ as a function of the camera pose. The UAV that yaws by an angle α. Assume that the vehicle is flying along the lines at constant speed, and so has a fixed pitch, β. Assume also that γ and X_u are zero. Equation (4.1) becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ Y_W \\ -Z_L \\ 1 \end{bmatrix} \quad \text{(A.6)}$$

This becomes

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} -Y_w \sin\alpha \\ Y_W \cos\alpha\cos\beta + Z_L \sin\beta - \ell \\ Y_w \cos\alpha \sin\beta - Z_L \cos\beta \\ -\frac{Y_W \cos\alpha\cos\beta + Z_L \sin\beta - \ell}{\lambda} \end{bmatrix} \quad \text{(A.7)}$$

From (A.7), using (4.2), the image co-ordinates are given by:

$$x = \lambda \frac{Y_w \sin\alpha}{Y_W \cos\alpha\cos\beta + Z_L \sin\beta - \ell} \text{ and}$$

$$z = -\lambda \frac{Y_w \cos\alpha \sin\beta - Z_L \cos\beta}{Y_W \cos\alpha\cos\beta + Z_L \sin\beta - \ell} \quad \text{(A.8)}$$

Applying the Hough Transform (4.6) then gives:

$$\theta = \tan^{-1}\left(\frac{-Y_w \sin\alpha}{Y_w \cos\alpha \sin\beta - Z_L \cos\beta}\right) \text{ and}$$

$$\rho = \frac{-\lambda X_u}{Y_W \cos\beta + Z_L \sin\beta - \ell}\cos\theta - \frac{\lambda(Y_W \sin\beta - Z_L \cos\beta)}{Y_W \cos\beta + Z_L \sin\beta - \ell}\sin\theta \quad \text{(A.9)}$$

It can be seen that both ρ and θ change with yaw.

A.3 Analysis for the Pitch Axis

If we apply equation (4.1) to a model of the centre conductor and vary the pitch (β) of the UAV from the "normal" pitch value required for the UAV to move forward then we get the following:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ Y_W \\ -Z_L \\ 1 \end{bmatrix}$$

(A.10)

This becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ Y_W \cos\beta + Z_L \sin\beta - \ell \\ Y_W \sin\beta - Z_L \cos\beta \\ -\frac{Y_W \cos\beta + Z_L \sin\beta - \ell}{\lambda} \end{bmatrix} \quad \text{(A.11)}$$

From (A.11), using (4.2), the image co-ordinates are given by:

$$x = 0 \text{ and } z = -\lambda \frac{Y_W \sin\beta - Z_L \cos\beta}{Y_W \cos\beta + Z_L \sin\beta - \ell} \quad \text{(A.12)}$$

Applying the Hough Transform then gives:

$$\theta = 0 \text{ and } \rho = 0 \quad \text{(A.13)}$$

It can be seen that the pitch of the UAV does not affect the position of the centre line in the image, however if the same analysis is done for a sideline a lateral distance X_S from the centre line then we get following:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{\lambda} & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\ell \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -X_S \\ Y_W \\ -Z_L \\ 1 \end{bmatrix}$$

(A.14)

This becomes:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ \frac{-Y_C}{\lambda} \end{bmatrix} = \begin{bmatrix} -X_S \\ Y_W \cos\beta + Z_L \sin\beta - \ell \\ Y_W \sin\beta - Z_L \cos\beta \\ -\frac{Y_W \cos\beta + Z_L \sin\beta - \ell}{\lambda} \end{bmatrix} \quad \text{(A.15)}$$

From (A.15), using (4.2), the image co-ordinates are given by:

$$x = \frac{\lambda X_S}{Y_W \cos\beta + Z_L \sin\beta - \ell} \text{ and}$$

$$z = -\lambda \frac{Y_W \sin\beta - Z_L \cos\beta}{Y_W \cos\beta + Z_L \sin\beta - \ell} \quad \text{(A.16)}$$

Applying the Hough Transform then gives:

$$\theta = \tan^{-1}\left(\frac{\lambda X_S}{Y_W \sin\beta - Z_L \cos\beta}\right) \text{ and}$$

$$\rho = \frac{\lambda X_S}{Y_W \cos\beta + Z_L \sin\beta - \ell}\cos\theta + \lambda \frac{Y_W \sin\beta - Z_L \cos\beta}{Y_W \cos\beta + Z_L \sin\beta - \ell}\sin\theta \quad \text{(A.17)}$$

It can be seen that both ρ and θ for a sideline vary with pitch angle (β).

Appendix B Two-Axis Analysis

Shown in this appendix are graphs showing the effect of varying both roll and lateral displacement and yaw and roll on the values of θ_C and ρ_C and the effect of varying both Height and Lateral Displacement on θ_d and ρ_d.

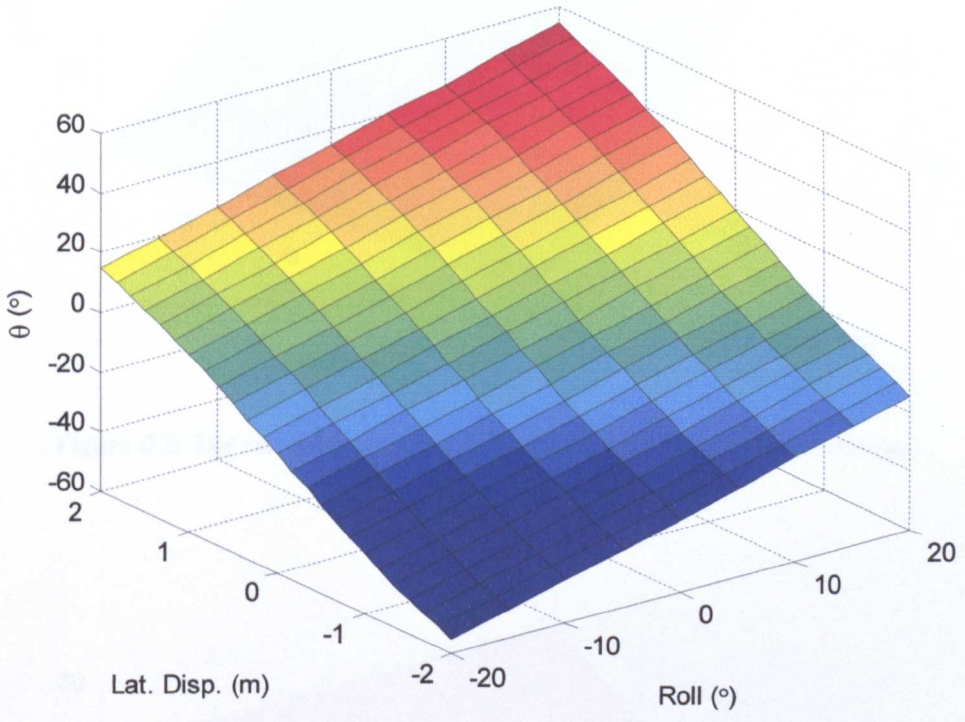


Figure B.1: The effect of varying both Roll and Lateral Displacement on θ_C.

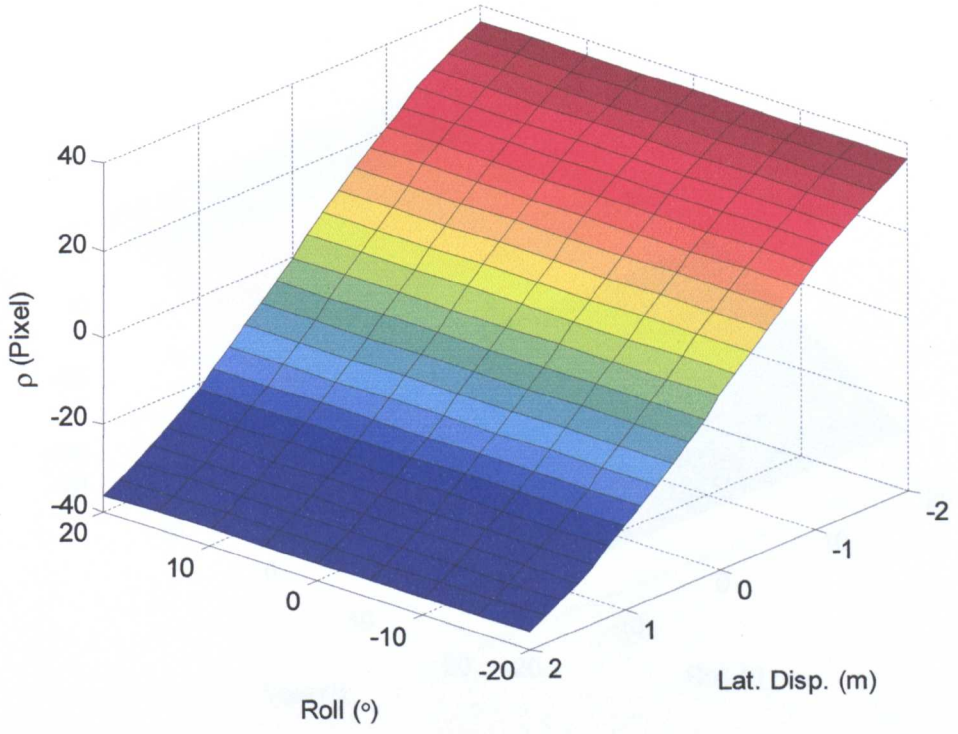


Figure B.2: The effect of varying both Roll and Lateral Displacement on ρ_C.

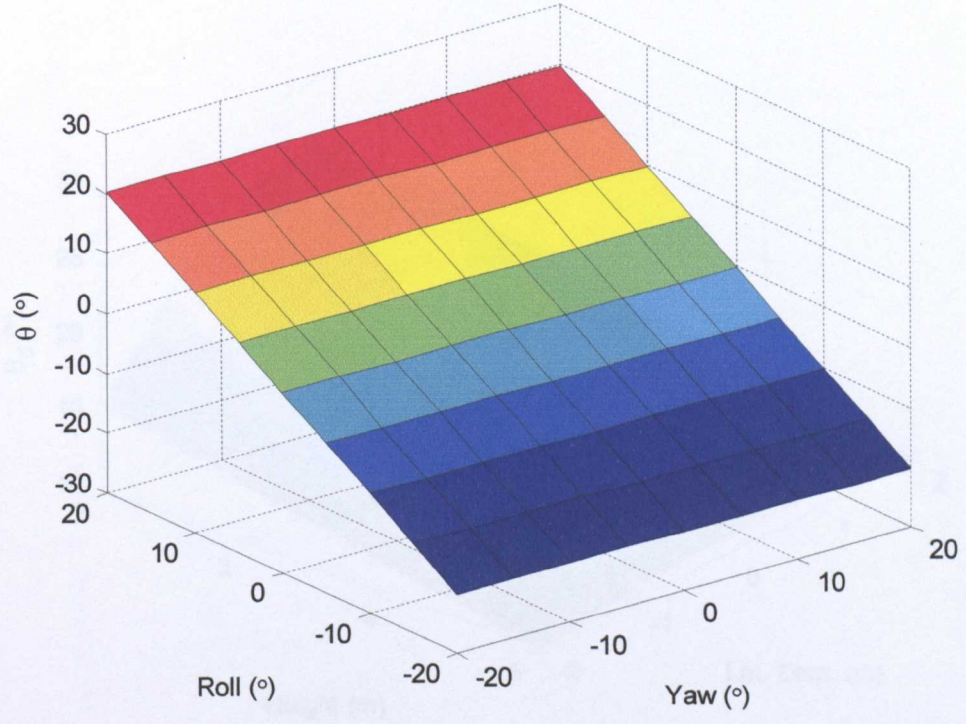


Figure B.3: The effect of varying both Yaw and Roll on θ_C.

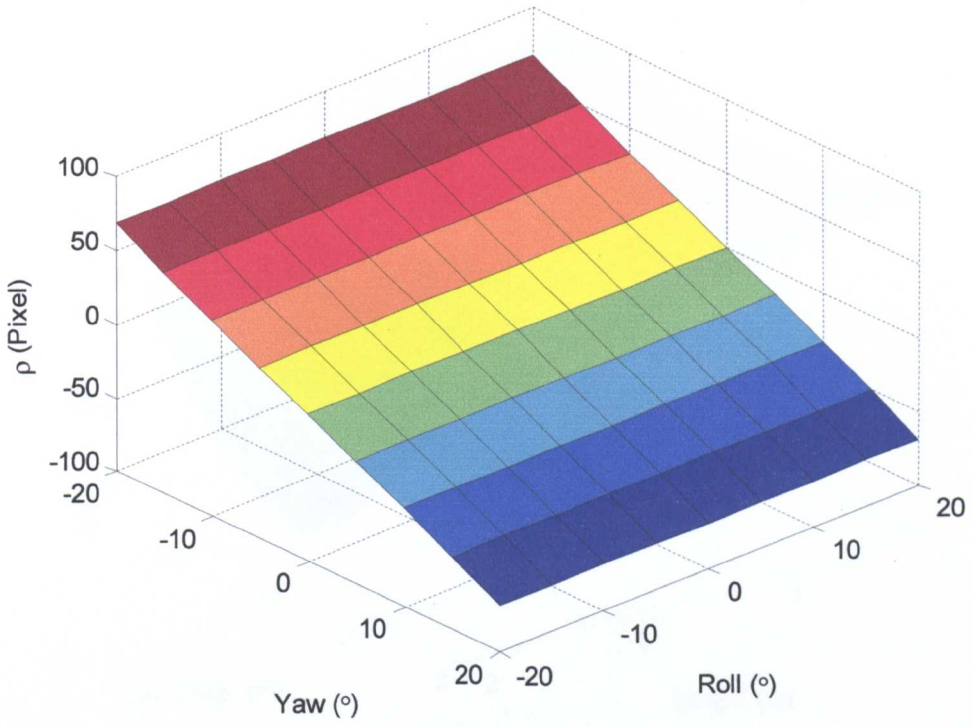


Figure B.4: The effect of varying both Yaw and Roll on ρ_C.

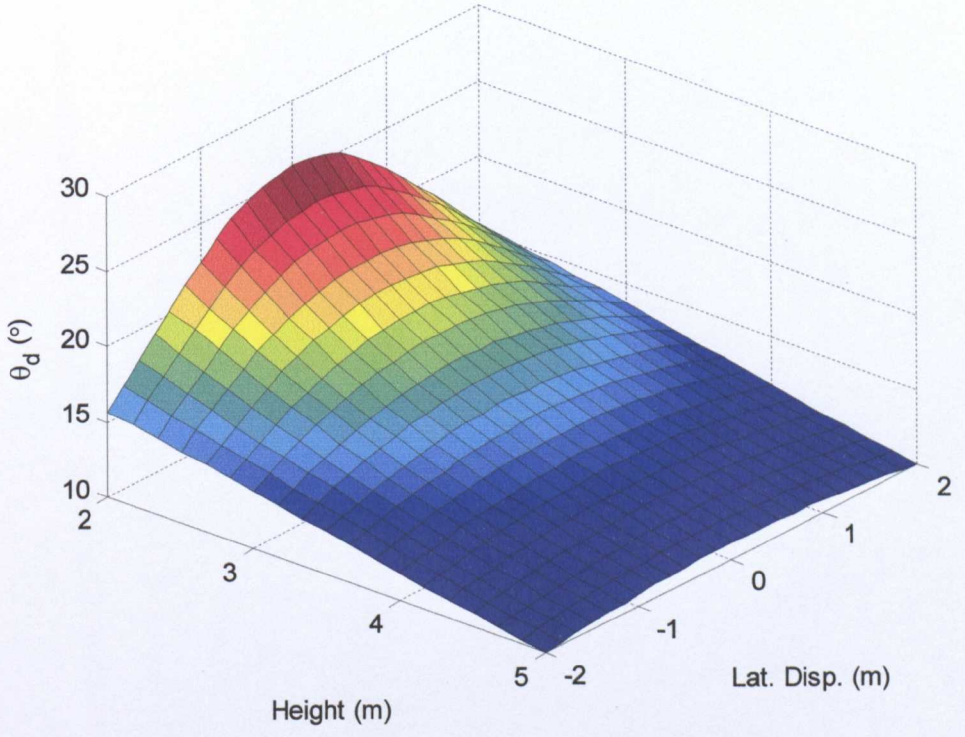


Figure B.5: The effect of varying both Height and Lateral Displacement on θ_d.

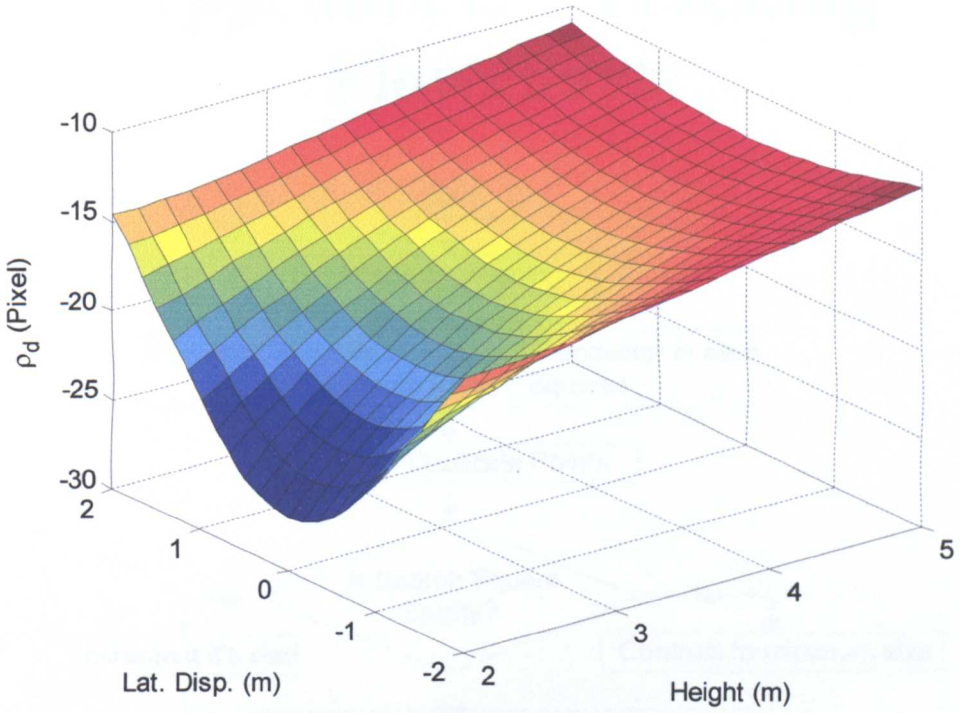


Figure B.6: The effect of varying both Height and Lateral Displacement on ρ_d.

Appendix C Tracking Flowcharts

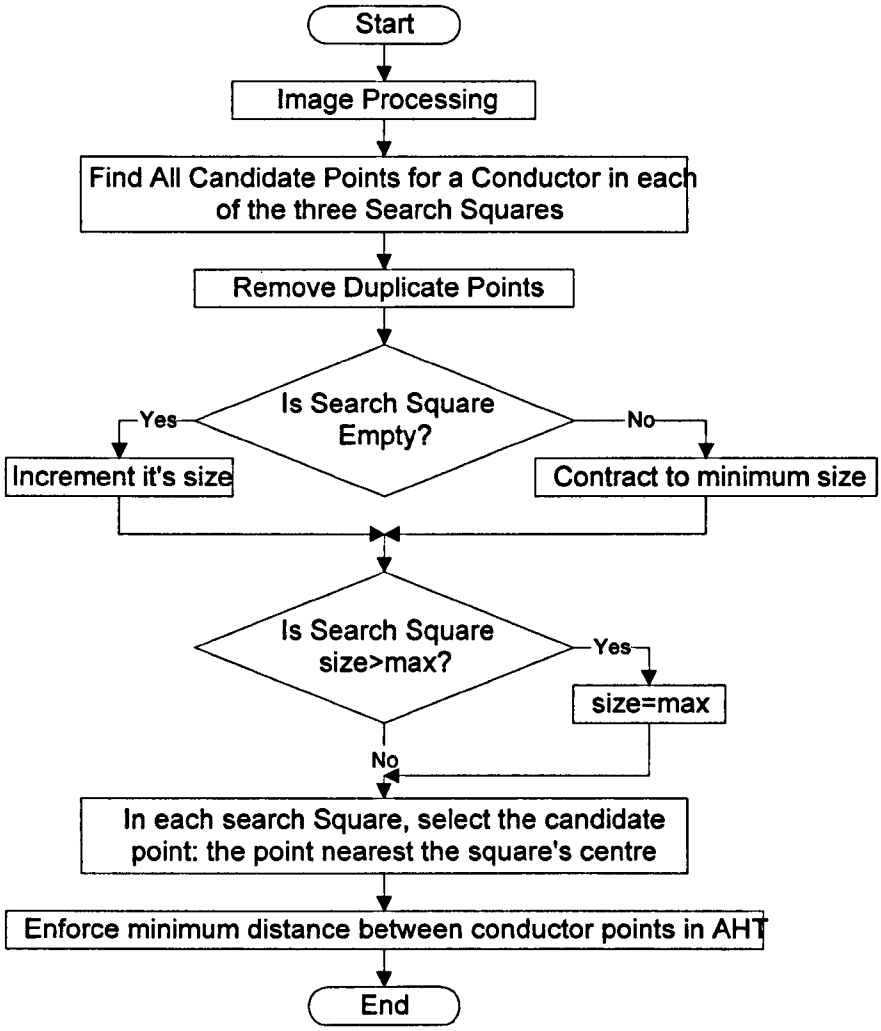


Figure C.1: Flowchart showing the operation of image processing and search of the AHT

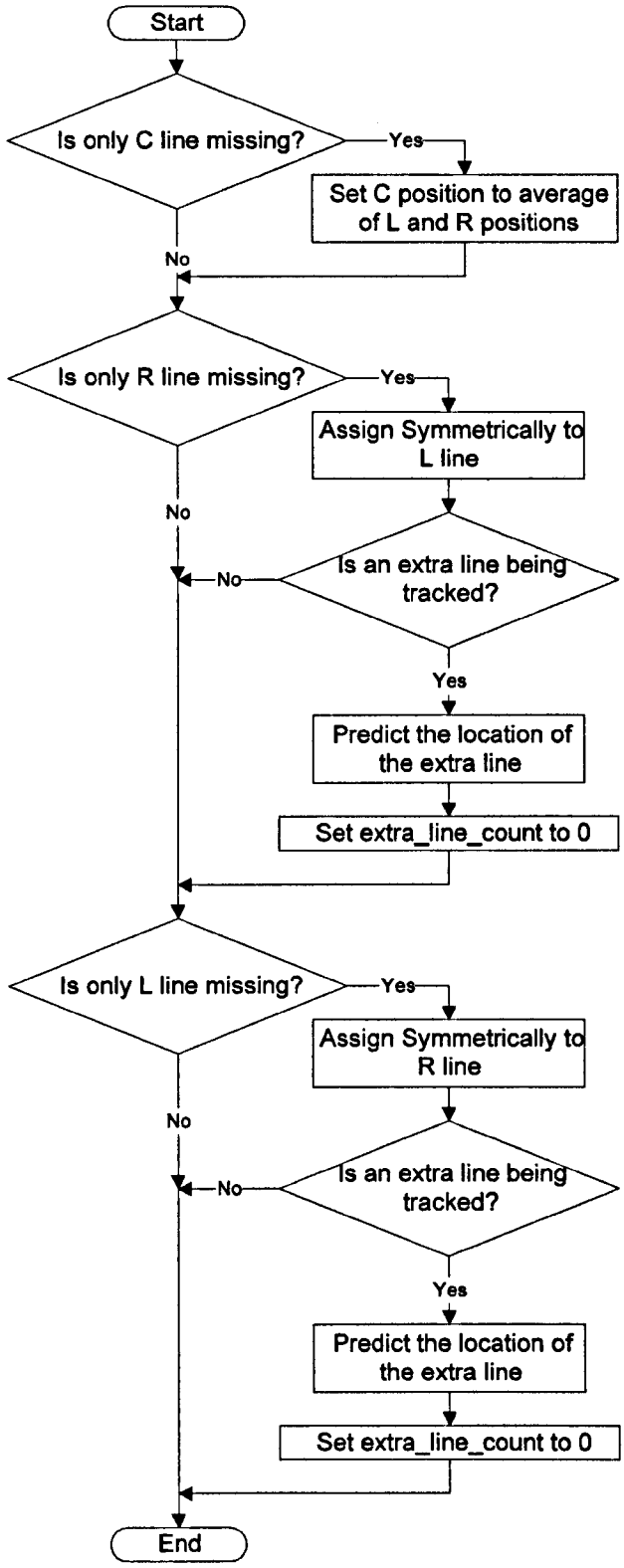


Figure C.2: Flowchart showing the third line prediction

Appendix D C++ Source Code

D.1 Header Files

D.1.1 ControlThread.h

```
#if
!defined(AFX_CONTROLTHREAD_H__DF15EFD5_E059_4367_9C39_54FE9FFE31B7
__INCLUDED_)
#define
AFX_CONTROLTHREAD_H__DF15EFD5_E059_4367_9C39_54FE9FFE31B7__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ControlThread.h : header file
//

#include "MapObject.h"
#include "Serial.h"

/////////////////////////////////////////////////////////////////////
///////////
// ControlThread thread

class ControlThread : public CWinThread
{
      DECLARE_DYNCREATE(ControlThread)
protected:
      ControlThread();           // protected constructor used by
dynamic creation
      //constructor
public:
      ControlThread(short x_p, short y_p, short yaw_p, short x_d,
short y_d, short yaw_d, MapObject* m, BOOL m_exist, Serial*
trig_s, Serial* fvis_s, Serial* bvis_s, BOOL s_exist, short x_min,
short x_max, short y_min, short y_max, short yaw_min, short
yaw_max, double wind, BOOL vis_en);

// Attributes
public:
      HANDLE m_hEventKill;
      HANDLE m_hEventDead;

      //Critical section for access to the tracking data by
external thread  static CRITICAL_SECTION accessLock;

      //**************Demand and Test rig output
variables**********

      //position varaibles
      short x_pos, y_pos, yaw_pos;

      //demand variables
      short x_dem, y_dem, yaw_dem;
```

```
	//output variables
	short x_out, yaw_out;

	//************************UAV Model
variables*****************

	//d2theta/dt2
	double d2theta;

	//d_theta/dt and delayed version
	double d_theta, d_theta_n1;

	//theta value and delayed version
	double uav_theta, uav_theta_n1;

	//dU/dt
	double dU;

	//U and delayed version
	double U, outU, U_n1;

	//intermediate value for pitch rate feedback
	double dtfb1;

	//integral of intermediate value for pitch rate feedback and
delayed version
	double int_dtfb1, int_dtfb1_n1;

	//pitch rate feedback value and delayed version
	double dtfb2, dtfb2out, dtfb2_n1;

	//double versions of demand and output variables
	double temp_x_dem, temp_y_dem, temp_yaw_dem, temp_x_out,
temp_yaw_out;

	//delayed values of temp_x_out and temp_yaw_out
	double temp_x_out_n1, temp_yaw_out_n1;

	//position error in x direction
	double x_p_err, x_p_err_n1;

	//integral of position error in x direction
	double int_x_p_err, int_x_p_err_n1;

	//************************Yaw part of UAV
model******************

	//position error in Yaw
	double yaw_p_err;

	//pre-integrated yaw output
	double d_yaw;

	//****************************Other
Variables********************

	//wind gust magnitude
	double windGust;

	//enable vision feedback
	BOOL visionEnable;
```

```
	//storage for gps error data
	short* gps_errors;
	short last_gps_fix;
	int gps_errors_length;

	//error code storage
	int err_code, ext_err_code;

	//tracking enable flag
	BOOL trackEnable;

	//gust delay count
	int d_count;

	//set to align with the lines before proceeding along the
line
	BOOL align;

	//position step increments
	double dx, dy;

	//flag to indicate if tracking has finished
	BOOL finished;

	//file directory for position output file
	char* directory;

	//copies of position variables for external access
	short ext_x_pos, ext_y_pos, ext_yaw_pos, ext_x_dem,
ext_y_dem, ext_yaw_dem;

	//Map of poles
	MapObject* map;
	//flag to indicate if the map has been created correctly
	BOOL mExist;

	//Serial Ports
	Serial* sp_trig;
	Serial* sp_fvis;
	Serial* sp_bvis;
	//flag to indicate if the serial ports have been created
correctly
	BOOL sExist;

	//movement restrictions
	short xmin, xmax, ymin, ymax, yawmin, yawmax;

	//tracking status
	int trackStatus;

	//copies of frequency variables for external access
	unsigned int ext_c_freq, ext_v_freq;

	//Control Loop Frequency in Hz
	unsigned int c_frequency;
	//Vision Loop Frequency in Hz
	unsigned int v_frequency;
	//variable to show if the system has a high performance
counter
	BOOL hpc;
```

```
	//storage structure to read out frequency
	LARGE_INTEGER temp_freq;
	//performance counter frequency
	_int64 HPC_freq;
	//storage for start time of current iteration
	LARGE_INTEGER curStartTime;
	//storage for start time of previous iteration
	LARGE_INTEGER prevStartTime;
	//difference between the start times
	_int64 elapsedCounts;
	//difference between the times in milliseconds
	unsigned int elapsedMS;
	//sampling period in milliseconds
	unsigned int sampleTime;

	//sampling period in seconds
	double T;

	//pi
	double pi;

	//rho and theta values from vision computers
	int frho, ftheta, brho, btheta;

	//tracking staus from vision computers
	int vis_track_stat_f, vis_track_stat_b;

// Operations
public:
	//function to run each step of the controller and UAV model
	void SingleStep();
	//function to allow access to the tracking data by the
interface thread
	void AccessData(short* x_p, short* y_p, short* yaw_p, short*
x_d, short* y_d, short* yaw_d, unsigned int* freq, unsigned int*
v_freq, BOOL* fin, int* errcode);
	//function to send demands to the test rig
	int DataTrans();
	//function to kill the control thread
	void KillThread();

	//apply GPS errors to position values from test rig
	short GPS_Error(short value, int* count);

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(ControlThread)
	public:
	virtual BOOL InitInstance();
	virtual int ExitInstance();
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~ControlThread();
	virtual void Delete();

	// Generated message map functions
	//{{AFX_MSG(ControlThread)
		// NOTE - the ClassWizard will add and remove member
functions here.
```

```
	//}}AFX_MSG

	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////
///////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_CONTROLTHREAD_H__DF15EFD5_E059_4367_9C39_54FE9FFE31B7
__INCLUDED_)
```

D.1.2 VisionThread.h

```
#if
!defined(AFX_VISIONTHREAD_H__9B965A29_66B7_43D2_A212_37F70FF328FF_
_INCLUDED_)
#define
AFX_VISIONTHREAD_H__9B965A29_66B7_43D2_A212_37F70FF328FF__INCLUDED
_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// VisionThread.h : header file
//

#include <mil.h>

#include "ImageObject.h"
#include "EdgeMap.h"
#include "HoughTransform.h"
#include "kalman.h"

/////////////////////////////////////////////////////////////////////////
///////////
// VisionThread thread

class VisionThread : public CWinThread
{
	DECLARE_DYNCREATE(VisionThread)
protected:
	VisionThread();
public:
	VisionThread(BOOL cap_Source, BOOL cap_Proc);

// Attributes
public:

	HANDLE m_hEventKill;
	HANDLE m_hEventDead;

	//Critical section for access to the tracking data by
external thread
	static CRITICAL_SECTION accessLock;
```

```
      //error code storage
      int err_code, ext_err_code;

      //copy of frequency variable for external access
      unsigned int ext_v_freq;

      //Vision Loop Frequency in Hz
      unsigned int v_frequency;
      //variable to show if the system has a high performance
counter
      BOOL hpc;
      //storage structure to read out frequency
      LARGE_INTEGER temp_freq;
      //performance counter frequency
      _int64 HPC_freq;
      //storage for start time of current iteration
      LARGE_INTEGER curStartTime;
      //storage for start time of previous iteration
      LARGE_INTEGER prevStartTime;
      //difference between the start times
      _int64 elapsedCounts;
      //difference between the times in milliseconds
      unsigned int elapsedMS;
      //sampling period in milliseconds
      unsigned int sampleTime;

      //************Handles to Access the Frame Grabber
Card**************

      MIL_ID MilApplication, /*Application identifier.*/
           MilSystem,      /*System identifier.*/
           MilDisplay,     /*Display identifier.*/
              MilDigitizer,   /*Digitizer identifier.*/
              MilParentBuff,  /*Parent buffer which contains the
grab and display buffer.*/
           MilGrabImage,   /*Grab Image buffer identifier.*/
             MilDispImage;   /*Display Image buffer identifier.*/

      int channel;                //digitizer channel

      double imScale;             //image grabing scle factor

      ImageObject* SourceImage;   //image captured from digitizer
      ImageObject* DestImage;     //final processed image

      EdgeMap* edgeMap;           //Edge Map of captured image

      HoughTransform* ht;         //Hough Transform of captured
image

      BOOL capSource, capProc;    //whether to capture source and
processed images.

      int frameNumber;            //storage for current frame
number

      //pointers to strings for saving image data
      char *directory, *Fname, *baseFname, *extn, *Fname2,
*baseFname2;

      //strings to print rho theta values
```

```
      char* rhoOutString;
      char* thetaOutString;

      //external acces to rho and theta values from HT
      int ext_rhoOut, ext_thetaOut;

      //search squares
      struct kalman lRhoKal;
      struct kalman lThetaKal;
      struct kalman cRhoKal;
      struct kalman cThetaKal;
      struct kalman rRhoKal;
      struct kalman rThetaKal;
      struct kalman eRhoKal;
      struct kalman eThetaKal;

      //count to see if we need to switch lines
      int switch_count;

      //count to see if we have lost the lines
      int lose_count;

      //count to see if we have lost the centre line
      int lose_centre_count;

      //count to aquire the lines
      int aquire_count;

      //storage for tracking status
      int track_stat, ext_track_stat;

// Operations
public:
      //process an individual frame
      void SingleStep();
      //Contrast Enhance an image
      int ContEnhance(ImageObject* source);
      //Edge Detector
      int EdgeDetect(ImageObject* source, EdgeMap* dest, int
edgeGap, double gThres);
      //Hough Transform
      int Hough_Transform(EdgeMap* source, HoughTransform* dest,
int edgeGap, double threshold, int a_size);
      //Superimpose the lines from an AHT on to an image (not a
true inverse)
      int InvHoughTransform(HoughTransform* soruce, ImageObject*
dest);
      //Track lines from frame to frame
      int TrackLines(HoughTransform* source, struct kalman*
lthetakal, struct kalman* lrhokal, struct kalman* cthetakal,
struct kalman* crhokal, struct kalman* rthetakal, struct kalman*
rrhokal, struct kalman* ethetakal, struct kalman* erhokal,
int*trackStat, int* switchCount, int* loseCount, int*
loseCentreCount);
      //Acquisiiton routine
      int Acquisition(int* bestposition, int* bestfound,
HoughTransform* htr, int thetadiff, int rhodiff, int thetavar, int
rhovar, double sideline_fact, double centre_fact, int fnum);
      //Calculate  image gradient
      double Grad(ImageObject* image, int xpos, int ypos, double*
dx, double* dy);
```

```
      //find a point to represent a cluster of points in AHT
      unsigned int FindPoint(int* xout, int* yout, HoughTransform*
trans, int xbase, int ybase, int c_size);
      //Find points in a specified area of the AHT
      int FindPoints(int* outpoints, int* outnumpoints,
HoughTransform* trans, int thetabase, int rhobase, int theta_size,
int rho_size, int centre, int mode, int* HTAccessError);
      //Draw a line on an image
      int DrawLine(ImageObject* image, int theta, int rho,
unsigned char intensity1, unsigned char intensity2);
      //copy an image
      int CopyImage(ImageObject* source, ImageObject* dest);
      //allow access to the tracking data by external thread
      void AccessData(unsigned int* v_freq, int* errcode, int*
thetaout, int* rhoout);
      //allow access to the tracking status by external thread
      void ReadWriteTrackStat(int* trStat, BOOL read);
      //kill the thread
      void KillThread();

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(VisionThread)
      public:
      virtual BOOL InitInstance();
      virtual int ExitInstance();
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~VisionThread();
      virtual void Delete();

      // Generated message map functions
      //{{AFX_MSG(VisionThread)
            // NOTE - the ClassWizard will add and remove member
functions here.
      //}}AFX_MSG

      DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////
///////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_VISIONTHREAD_H__9B965A29_66B7_43D2_A212_37F70FF328FF_
_INCLUDED_)
```

D.1.3 EdgeMap.h

```
// EdgeMap.h: interface for the EdgeMap class.
//
////////////////////////////////////////////////////////////////////
////
```

```
#if
!defined(AFX_EDGEMAP_H__3B004A20_4348_11D7_BAAC_0004769EA59C__INCL
UDED_)
#define
AFX_EDGEMAP_H__3B004A20_4348_11D7_BAAC_0004769EA59C__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class EdgeMap
{
      //constructor/descructor
public:
      EdgeMap();
      EdgeMap(int xSize, int ySize);
      virtual ~EdgeMap();

      //attributes
public:
      //storage for edge strength data
      double** edge_strength_data;
      //storage for edge angle data
      double** edge_angle_data;
      //storage for the thresholdded edge map
      unsigned char** thres_edge_data;
      //dimensions of the edge map
private:
      int xsize, ysize;

      //methods
public:
      //read the X dimension
      int GetXSize();
      //read the Y dimension
      int GetYSize();
      //set the edge map to zero
      void SetZeros();
};

#endif //
!defined(AFX_EDGEMAP_H__3B004A20_4348_11D7_BAAC_0004769EA59C__INCL
UDED_)
```

D.1.4 HoughTransform.h

```
// HoughTransform.h: interface for the HoughTransform class.
//
////////////////////////////////////////////////////////////////////
////

#if
!defined(AFX_HOUGHTRANSFORM_H__3B004A20_4348_11D7_BAAC_0004769EA59
C__INCLUDED_)
#define
AFX_HOUGHTRANSFORM_H__3B004A20_4348_11D7_BAAC_0004769EA59C__INCLUD
ED_

#if _MSC_VER >= 1000
#pragma once
```

```
#endif // _MSC_VER >= 1000

class HoughTransform
{
      //constructor/destructor
public:
      HoughTransform();
      HoughTransform(int xSize, int ySize, int del_Rho, int
del_Theta);

      virtual ~HoughTransform();

      //attributes
public:
      //storage for:
      unsigned int** HT_data;          //Hough transform data
      unsigned char** thres_HT_data;   //Thresholded Hough
transform
      unsigned int** aggreg_values;    //Values used in
aggregation
      unsigned char** aggreg_HT_data;  //Aggregated Hough
transform
private:
      int xsize, ysize;               //size of the image that was
transformed.
      int numRho, numTheta;           //size of x and y axes of
hough space, theta is on x axis, rho is on y axis
      int delRho, delTheta;           //rho and theta
quantisation.

      //methods
public:
      //read the X size of the transformed image
      int GetXSize();
      //read the Y size of the transformed image
      int GetYSize();
      //read the Rho size of the transform
      int GetRhoSize();
      //read the Theta size of the transform
      int GetThetaSize();
      //read the Rho quantisation of the transform
      int GetRhoQuant();
      //read the Theta quantisation of the transform
      int GetThetaQuant();

      void SetZeros();
      unsigned int GetMax();
};

#endif //
!defined(AFX_HOUGHTRANSFORM_H__3B004A20_4348_11D7_BAAC_0004769EA59
C__INCLUDED_)
```

D 1.5 ImageObject.h

```
// ImageObject.h: interface for the ImageObject class.
//
//////////////////////////////////////////////////////////////////
////
```

```
#if
!defined(AFX_IMAGEOBJECT_H__3B004A20_4348_11D7_BAAC_0004769EA59C__
INCLUDED_)
#define
AFX_IMAGEOBJECT_H__3B004A20_4348_11D7_BAAC_0004769EA59C__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class ImageObject
{
      //contructor/destructor
public:
      ImageObject();
      ImageObject(int xSize, int ySize);
      virtual ~ImageObject();

      //attributes
public:
      //storage for the image data
      unsigned char** image_data;
private:
      //dimensions of the image
      int xsize,ysize;
      //minimum and maximum pixel values in the image
      unsigned char min, max;

      //methods
public:
      //read the X size of the image
      int GetXSize();
      //read the Y size of the image
      int GetYSize();
      //set all the image pixels to zero
      void SetZeros();
      //invert the image
      void InvertImage();
      //find the minimum and maximum pixel values in the image
      //exclude the edge of the image to a distance edgegap
      void FindMinMax(int edgegap);
      //read the minimum pixel value
      unsigned char GetMin();
      //read the maximum pixel value
      unsigned char GetMax();
};

#endif //
!defined(AFX_IMAGEOBJECT_H__3B004A20_4348_11D7_BAAC_0004769EA59C__
INCLUDED_)
```

D 1.6 kalman.h

```
//kalman filter data structure

struct kalman
{
      int estimate; //current estimate
      int prevEstimate; //previous estimate
      int prediction; //prediction
```

```
	double estProb; //variance of current estimate
	double predProb; //variance of prediction
	double H;
	double K; //Kalman Gain
	int phiOffset; //offset for generating next prediction
	double Q; //variance
	int measurement; //measurement
	double R; //variance of measurement
};
```

D.2 Selected Functions

D.2.1 ControlThread::SingleStep

```
//*****************************************************************
*******
// SingleStep
// Follow Pole Maps
// Combine Error Sources
// Send demands to test rig
//*****************************************************************
*******
void ControlThread::SingleStep()
{
	//Data Output file
	FILE* dataOut;
	//filename
	char dataFName[100];

	//set rho and theta quantisation
	int delrho = 2;
	int deltheta = 1;

	double temp_ftheta, temp_frho, temp_btheta, temp_brho,
x_ht_err, yaw_ht_err, roll_ht_err, x_ht_err2, yaw_ht_err2,
roll_ht_err2;
	short store_x_out, store_yaw_out; //xout for data file
	double visionFract = 0; //fraction of the feedback signal
that is from the vision feedback

	if(hpc)
	{
		//******************Wait until the next sample
time*************

		//only wait if in tracking mode
		if (trackStatus == 2)
		{
			//read the number of ms since the previous
iteration started
			QueryPerformanceCounter(&curStartTime);
			elapsedCounts = curStartTime.QuadPart -
prevStartTime.QuadPart;
			elapsedMS = (unsigned
int)((1000*elapsedCounts)/HPC_freq);

			//if the full sample period hsn't elapsed, then
wait for the reamaining time, rounded down to the nearest 10 ms
```

```
                if (elapsedMS < sampleTime)
                {
                        Sleep((sampleTime - elapsedMS)-((sampleTime
- elapsedMS)%10));
                }

                //wait until end of sample time
                do
                {
                        QueryPerformanceCounter(&curStartTime);
                        elapsedCounts = curStartTime.QuadPart -
prevStartTime.QuadPart;
                        elapsedMS = (unsigned
int)MathFns::Round((1000*(double)elapsedCounts)/(double)HPC_freq);
                }
                while(elapsedMS < sampleTime);

                //calculate the sampling frequency
                QueryPerformanceCounter(&curStartTime);
                elapsedCounts = curStartTime.QuadPart -
prevStartTime.QuadPart;
                c_frequency = (unsigned
int)MathFns::Round((double)HPC_freq/(double)elapsedCounts);
                //QueryPerformanceCounter(&prevStartTime);
                EnterCriticalSection(&ControlThread::accessLock);
                {
                        ext_c_freq = c_frequency;
                }
                LeaveCriticalSection(&ControlThread::accessLock);
                QueryPerformanceCounter(&prevStartTime);
        }
    }

    //read position of the centre line from vision PCs
    err_code = sp_fvis->VisReadData(&ftheta, &frho);
    err_code = sp_bvis->VisReadData(&btheta, &brho);

    //read tracking status from the vision PCs
    err_code = sp_fvis->VisReadTrackStat(&vis_track_stat_f);
    err_code = sp_bvis->VisReadTrackStat(&vis_track_stat_b);

    //set tracking enable to false if necessary
    if(!trackEnable)
    {
        sp_fvis->VisWriteTrackStat(0);
        sp_bvis->VisWriteTrackStat(0);
        err_code=4;
    }

    //perform tracking based on map
    switch (trackStatus)
    {

    //**************tracking not started need to intialise
tracking**********

    case 0:
        //get the difference in the y direction to find
direction
        dy = (((double)map->GetNextCoOrds()[1]) -
((double)map->GetCurCoOrds()[1]));
```

```
            //set the initial demands
            yaw_dem = 0;
            x_dem = map->GetCurCoOrds()[0];
            y_dem = map->GetCurCoOrds()[1];

            //set the past output values to start of line x value
            temp_x_out_n1 = 0.0033*(double)map-
>GetNextCoOrds()[0];
            temp_yaw_out_n1 = 0;

            //set curret output values to 0
            temp_x_out = 0;
            temp_yaw_out = 0;

            //set current d2theta/dt2 to 0
            d2theta = 0;

            //set current and delayed d_theta/dt to 0
            d_theta = 0;
            d_theta_n1 = 0;

            //set current and delayed theta value to 0
            uav_theta = 0;
            uav_theta_n1 = 0;

            //set current dU/dt to 0
            dU = 0;

            //set current and delayed U to 0
            U = 0;
            U_n1 = 0;

            //set the past error value to 0
            x_p_err_n1 = 0;

            //set the past integral of position error to 0
            int_x_p_err_n1 = 0;

            //set the integral of intermediate value for pitch
rate feedback and delayed version to 0
            int_dtfb1 = 0;
            int_dtfb1_n1 = 0;

            //set the pitch rate feedback value and delayed
version to 0
            dtfb2 = 0;
            dtfb2_n1 = 0;

            //set intermediate value for pitch rate feedback to 0
            dtfb1 = 0;

            //set to align x before starting along the lines
            align = TRUE;

            //set the y difference to 5 in the correct direction.
            if (dy > 0)
            {
                  dy = 5;
            }
            if (dy < 0)
```

```
            {
                  dy = -5;
            }

            //initialise the floating point version of the demands
            temp_x_dem = (double) x_dem;
            temp_y_dem = (double) y_dem;

            //send demands to test rig
            DataTrans();

            //move to the tracking state
            trackStatus = 2;
            break;

      //*****starting new line section and so need to re-
initialise******

      case 1:
            //set demands
            yaw_dem = 0;
            x_dem = map->GetCurCoOrds()[0];
            y_dem = map->GetCurCoOrds()[1];

            //set not to align x before starting along the lines
            align = FALSE;

            //set the y difference to 5 in the correct direction.
            if (dy > 0)
            {
                  dy = 5;
            }
            if (dy < 0)
            {
                  dy = -5;
            }

            //initialise the floating point version of the demands
            temp_x_dem = (double) map->GetCurCoOrds()[0];
            temp_y_dem = (double) map->GetCurCoOrds()[1];

            //send demands to test rig
            DataTrans();

            //move to the tracking state
            trackStatus = 2;
            break;

      //********************tracking between two
poles****************

      case 2:
            //read x demand from the map
            x_dem = map->GetNextCoOrds()[0];

            //calculate error signal from map demand and test rig
position
            if(trackEnable)
            {
                  x_p_err = (double)(x_dem - GPS_Error(x_pos,
&d_count));
```

```
                yaw_p_err = (double)(yaw_dem - yaw_pos);
            }
            else
            {
                x_p_err = (double)(x_dem - x_pos);
                yaw_p_err = (double)(yaw_dem - yaw_pos);
            }

            //if we are using vision feedback set vision fraction
to 1 otherwise set to 0 and reset the vision tracking
            if(visionEnable && trackEnable && (x_pos < (x_dem +
600)) && (x_pos > (x_dem - 600)))
            {

     if((vis_track_stat_f==0)||(vis_track_stat_f==4)||(vis_track_
stat_b==0)||(vis_track_stat_b==4))
                {
                    visionFract = 0;
                }
                else
                {
                    visionFract = 1;
                }
            }
            else
            {
                visionFract = 0;
                sp_fvis->VisWriteTrackStat(0);
                sp_bvis->VisWriteTrackStat(0);
                err_code=5;
            }

            //*****calculate correction factors from rho and
theta*******

            //apply HT quantisation factors
            temp_frho = frho * delrho;
            temp_ftheta = ftheta * deltheta;
            temp_brho = brho * delrho;
            temp_btheta = btheta * deltheta;

            //calculate x, yaw and roll errors
            x_ht_err = (0.0754*temp_brho)-(0.137*temp_frho)-
(0.222*temp_ftheta);
            yaw_ht_err = (0.690*temp_frho)+(1.46*temp_ftheta)-
(0.496*temp_brho);
            roll_ht_err = (3.17*temp_frho)+(5.46*temp_ftheta)-
(0.735*temp_btheta)-(1.85*temp_brho);

            //store ht output
            x_ht_err2=x_ht_err;
            yaw_ht_err2=yaw_ht_err;
            roll_ht_err2=roll_ht_err;

            //apply vision fraction
            x_ht_err*=visionFract;
            yaw_ht_err*=visionFract;
            roll_ht_err*=visionFract;

            //******calculate correction factors from DGPS and
compass******
```

```
            //apply count to metre conversion of 0.0033 to gps
error signal
            x_p_err*=0.0033;
            //apply gps faction
            x_p_err*=(1-visionFract);

            //apply count to degree conversion of 90/500 to
compass error signal
            yaw_p_err*=90;
            yaw_p_err/=500;
            //apply compass faction
            yaw_p_err*=(1-visionFract);

            //*************combine error sources**********

            //subtract the vision correction factor to error
signal
            x_p_err-=x_ht_err;

            //subtract the vision correction factor to error
signal
            yaw_p_err-=yaw_ht_err;

            //******calculate the pitch rate feedback
fraction*******

            //apply the pitch rate compensator (first integrator
is already performed as part of the uav model
            dtfb1=(0.04545*d_theta)+(0.1*uav_theta)-
(10*int_dtfb1);
            //apply the second integrator
            int_dtfb1=int_dtfb1_n1+(T*dtfb1);
            //set the previous value for the next iteration
            int_dtfb1_n1=int_dtfb1;
            //add in feedback
            dtfb1-=(0.01*dtfb2);
            //apply the third integrator
            dtfb2=dtfb2_n1+(T*dtfb1);
            //set the previous value for the next iteration
            dtfb2_n1=dtfb2;
            //apply the pitch rate gain
            dtfb2out = 5000 * dtfb2;

            //***************combine lat. disp. error
signals****************

            //add in the pitch rate feedback
            x_p_err -= dtfb2out;

            //apply the loop gain
            x_p_err *= 0.1;

            //apply the saturation
            if (x_p_err > 0.2)
            {
                  x_p_err = 0.2;
            }
            if (x_p_err < -0.2)
            {
                  x_p_err = -0.2;
```

```
            }

            //*********Apply the U.A.V. model******************

            //*************Lat. Disp. Model*********************

            //d2theta integrator
            //combine signals for integrator
            d2theta = (1.047*x_p_err)+(0.0103*U)-(0.883*d_theta);
            //apply the integrator
            d_theta = d_theta_n1 + (T*d2theta);
            //set the previous value for the next iteration
            d_theta_n1 = d_theta;

            //d_theta integrator
            //apply the integrator
            uav_theta = uav_theta_n1 + (T*d_theta);
            //set the previous value for the next interation
            uav_theta_n1 = uav_theta;
            //multiply by the gain
            dU = (-9.81)*uav_theta;
            //add in d_theta factor
            dU-=(0.0739*d_theta);

            //dU integrator
            //combine signals for integrator
            dU-=(0.591*U); //plus is used as we are using the
inverted version of U
            //apply the integrator
            U = U_n1 + (T*dU);
            //set the previous value for the next iteration
            U_n1 = U;

            //set the output U of the model
            outU = -U;

            //add wind gust position error to demand if necessary
            if(trackEnable)
            {
                  //increment count variable
                  d_count++;
                  //add in offset if time reached
                  if((d_count >= 175)&&(d_count <= 250))
                  {
                        U += windGust;
                  }
            }

            //U integrator
            //apply the integrator
            temp_x_out = temp_x_out_n1 + (T*outU);
            //set the previous value for the next iteration
            temp_x_out_n1 = temp_x_out;

            //*******************Yaw
Model************************

            d_yaw = yaw_p_err - temp_yaw_out;
            //apply yaw integrator
            temp_yaw_out = temp_yaw_out_n1 + (T*d_yaw);
            //set the previous value for the next iteration
```

```
            temp_yaw_out_n1 = temp_yaw_out;

            //*****************Output Data to
Rig*****************

            //convert x output position back to counts
            x_out = (short)(temp_x_out/0.0033);

            //convert yaw output position back to counts
            yaw_out = (short)(temp_yaw_out/90*500);

            //increment y demand if necessary, if finished, set
demand as map coordinate
            if (((y_dem < map->GetNextCoOrds()[1]) && (dy >=0)) ||
((y_dem > map->GetNextCoOrds()[1]) && (dy < 0)))
            {
                  temp_y_dem += dy;
                  y_dem = (short)temp_y_dem;
            }
            else
            {
                  y_dem = map->GetNextCoOrds()[1];
            }

            //store output values
            store_x_out = x_out;
            store_yaw_out = yaw_out;

            //send demands to test rig
            DataTrans();

            //if we have reached a pole, set move to next pole
state
            if(align)
            {
                  if ((y_dem == map->GetNextCoOrds()[1])&&((x_out
>= (map->GetNextCoOrds()[0] - 5))&&(x_out <= (map-
>GetNextCoOrds()[0] + 5))))
                  {
                        trackStatus = 3;
                  }
            }
            else
            {
                  if (y_dem == map->GetNextCoOrds()[1])
                  {
                        trackStatus = 3;
                  }
            }
            break;

      //*****reach a pole, move to next line section, or stop if
last pole*****

      case 3:
            //enable tracking in second section
            trackEnable = TRUE;

            //if we have another line section then change to
initialisation state to track along next line section
            if (map->Advance() == 0)
```

```
            {
                  trackStatus = 1;
            }

            //if no more line sections, then stop tracking
            else
            {
                  EnterCriticalSection(&ControlThread::accessLock);
                  {
                        finished = TRUE;
                  }
                  LeaveCriticalSection(&ControlThread::accessLock);
            }
            break;
      default:
            break;
      }
      //write external variables to be read by dialog process.
      EnterCriticalSection(&ControlThread::accessLock);
      {
            ext_x_pos = x_pos;
            ext_y_pos = y_pos;
            ext_yaw_pos = yaw_pos;
            ext_x_dem = x_out;
            ext_y_dem = y_dem;
            ext_yaw_dem = yaw_out;
            ext_v_freq = v_frequency;
            ext_err_code = err_code;
      }
      LeaveCriticalSection(&ControlThread::accessLock);
}
```

D.2.2 VisionThread::SingleStep

```
//*****************************************************************
****
// Single Step
// Process an individual frame and track the lines from the
previous frame.
//*****************************************************************
****
void VisionThread::SingleStep()
{
      int old_track_stat=0;

      int err;
      int deltheta = 0;
      int delrho = 0;

      int bestFound = 0;
      int bestPosition[6];

      char Fname3[100];
      char fn[100];

      if(hpc)
      {
            //read the number of ms since the previous iteration
started
            QueryPerformanceCounter(&curStartTime);
```

```
            elapsedCounts = curStartTime.QuadPart -
prevStartTime.QuadPart;
            elapsedMS = (unsigned
int)((1000*elapsedCounts)/HPC_freq);

            //if the full sample period hsn't elapsed, then wait
for the remaining time, rounded down to the nearest 10 ms
            if (elapsedMS < sampleTime)
            {
                  Sleep((sampleTime - elapsedMS)-((sampleTime -
elapsedMS)%10));
            }

            //wait until end of sample time
            do
            {
                  QueryPerformanceCounter(&curStartTime);
                  elapsedCounts = curStartTime.QuadPart -
prevStartTime.QuadPart;
                  elapsedMS = (unsigned
int)((1000*elapsedCounts)/HPC_freq);
            }
            while(elapsedMS < sampleTime);

            //calculate the sampling frequency
            QueryPerformanceCounter(&curStartTime);
            elapsedCounts = curStartTime.QuadPart -
prevStartTime.QuadPart;
            v_frequency = (unsigned
int)MathFns::Round((double)HPC_freq/(double)elapsedCounts);
            EnterCriticalSection(&VisionThread::accessLock);
            {
                  ext_v_freq = v_frequency;
            }
            LeaveCriticalSection(&VisionThread::accessLock);
            QueryPerformanceCounter(&prevStartTime);
      }

      //read current tracking status, in case it has changed
      EnterCriticalSection(&VisionThread::accessLock);
      {
            track_stat = ext_track_stat;
      }
      LeaveCriticalSection(&VisionThread::accessLock);

      //set error code for this iteration to 0
      err_code = 0;

      //Set the file names for saving images.
      frameNumber++;
      strcpy(Fname, baseFname);
      strcpy(Fname2, baseFname2);
      strcpy(Fname3, baseFname2);
      sprintf(fn,"%04d",frameNumber);
      strcpy(extn, ".tif");
      strcat(Fname, fn);
      strcat(Fname, extn);
      strcat(Fname2, fn);
      strcat(Fname2, extn);
      strcat(Fname3, fn);
      strcat(Fname3, "_2");
```

```
      strcat(Fname3, extn);

      // Grab an image.
      MdigGrab(MilDigitizer, MilGrabImage);

      //save the image biffer if required
      if(capSource)
      {
            MbufSave(Fname,MilGrabImage);
      }

      //Copy the image to the processing array
      MbufGet(MilGrabImage, SourceImage->image_data[0]);

      //Contrast Enhance the image
      ContEnhance(SourceImage);

      //Copy image from source to destination
      CopyImage(SourceImage, DestImage);

      //Perform Edge Detection
      EdgeDetect(SourceImage, edgeMap, 2, 1495.575);

      //Calculate Hough Transform
      err_code = Hough_Transform(edgeMap, ht, 2, 0.475, 9);

      //Calculate Inverse Hough Transform
      InvHoughTransform(ht, DestImage);

      if(capProc)
      {
            MbufPut(MilDispImage, DestImage->image_data[0]);//save
the image biffer if required
            MbufSave(Fname2,MilDispImage);
      }

      //store the tracking status so we can see if it has changed
      old_track_stat=track_stat;

      //read theta and rho quantisation
      deltheta = ht->GetThetaQuant();
      delrho = ht->GetRhoQuant();

      //*************Aquire the lines*********************

      if(track_stat==0)
      {
            //clear the kalman filter values
            lThetaKal.prediction = 0;
            lRhoKal.prediction = 0;
            lThetaKal.prevEstimate = 0;
            lRhoKal.prevEstimate = 0;
            lThetaKal.predProb = DEFRTHETA;
            lRhoKal.predProb = DEFRRHO;
            lThetaKal.phiOffset = 0;
            lRhoKal.phiOffset = 0;

            cThetaKal.prediction = 0;
            cRhoKal.prediction = 0;
            cThetaKal.prevEstimate = 0;
            cRhoKal.prevEstimate = 0;
```

```
            cThetaKal.predProb = DEFRTHETA;
            cRhoKal.predProb = DEFRRHO;
            cThetaKal.phiOffset = 0;
            cRhoKal.phiOffset = 0;

            rThetaKal.prediction = 0;
            rRhoKal.prediction = 0;
            rThetaKal.prevEstimate = 0;
            rRhoKal.prevEstimate = 0;
            rThetaKal.predProb = DEFRTHETA;
            rRhoKal.predProb = DEFRRHO;
            rThetaKal.phiOffset = 0;
            rRhoKal.phiOffset = 0;

            //acquire the lines
            Acquisition(bestPosition, &bestFound, ht, THETADIFF,
RHODIFF, THETAVAR, RHOVAR, FS, FC, frameNumber);

            //if a set of lines are found, set then as the
prediction
            if(bestFound==1)
            {
                  lThetaKal.prediction = bestPosition[0];
                  lRhoKal.prediction = bestPosition[1];
                  lThetaKal.prevEstimate = bestPosition[0];
                  lRhoKal.prevEstimate = bestPosition[1];
                  lThetaKal.predProb = DEFRTHETA;
                  lRhoKal.predProb = DEFRRHO;
                  lThetaKal.phiOffset = 0;
                  lRhoKal.phiOffset = 0;

                  cThetaKal.prediction = bestPosition[2];
                  cRhoKal.prediction = bestPosition[3];
                  cThetaKal.prevEstimate = bestPosition[2];
                  cRhoKal.prevEstimate = bestPosition[3];
                  cThetaKal.predProb = DEFRTHETA;
                  cRhoKal.predProb = DEFRRHO;
                  cThetaKal.phiOffset = 0;
                  cRhoKal.phiOffset = 0;

                  rThetaKal.prediction = bestPosition[4];
                  rRhoKal.prediction = bestPosition[5];
                  rThetaKal.prevEstimate = bestPosition[4];
                  rRhoKal.prevEstimate = bestPosition[5];
                  rThetaKal.predProb = DEFRTHETA;
                  rRhoKal.predProb = DEFRRHO;
                  rThetaKal.phiOffset = 0;
                  rRhoKal.phiOffset = 0;

                  aquire_count=1;
                  track_stat=4;
            }
            DrawLine(DestImage, deltheta*lThetaKal.prediction,
delrho*lRhoKal.prediction, 255, 0);
            DrawLine(DestImage, deltheta*rThetaKal.prediction,
delrho*rRhoKal.prediction, 255, 0);
            DrawLine(DestImage, deltheta*cThetaKal.prediction,
delrho*cRhoKal.prediction, 255, 128);
      }
```

```
      //*****if we have found the lines then re-aquire them to
check they are consistant****

      else if(track_stat==4)
      {
            //acquire the lines
            Acquisition(bestPosition, &bestFound, ht, THETADIFF,
RHODIFF, THETAVAR, RHOVAR, FS, FC, frameNumber);

            //if we haven't found a match
            if(bestFound==0)
            {
                  if(aquire_count>0)
                  {
                        aquire_count--;
                  }
                  else
                  {
                        track_stat=0;
                  }
            }

            //if we've found some lines
            if(bestFound==1)
            {
                  //check that the new line positions are similar
to the previous

      if((bestPosition[0]<(lThetaKal.prediction+10))&&(bestPositio
n[0]>(lThetaKal.prediction-
10))&&(bestPosition[1]<(lRhoKal.prediction+10))&&(bestPosition[1]>
(lRhoKal.prediction-
10))&&(bestPosition[2]<(cThetaKal.prediction+10))&&(bestPosition[2
]>(cThetaKal.prediction-
10))&&(bestPosition[3]<(cRhoKal.prediction+10))&&(bestPosition[3]>
(cRhoKal.prediction-
10))&&(bestPosition[4]<(rThetaKal.prediction+10))&&(bestPosition[4
]>(rThetaKal.prediction-
10))&&(bestPosition[5]<(rRhoKal.prediction+10))&&(bestPosition[5]>
(rRhoKal.prediction-10)))
                  {
                        //if so then save the new position
                        lThetaKal.prediction = bestPosition[0];
                        lRhoKal.prediction = bestPosition[1];
                        lThetaKal.prevEstimate = bestPosition[0];
                        lRhoKal.prevEstimate = bestPosition[1];
                        lThetaKal.predProb = DEFRTHETA;
                        lRhoKal.predProb = DEFRRHO;
                        lThetaKal.phiOffset = 0;
                        lRhoKal.phiOffset = 0;

                        cThetaKal.prediction = bestPosition[2];
                        cRhoKal.prediction = bestPosition[3];
                        cThetaKal.prevEstimate = bestPosition[2];
                        cRhoKal.prevEstimate = bestPosition[3];
                        cThetaKal.predProb = DEFRTHETA;
                        cRhoKal.predProb = DEFRRHO;
                        cThetaKal.phiOffset = 0;
                        cRhoKal.phiOffset = 0;

                        rThetaKal.prediction = bestPosition[4];
```

```
                        rRhoKal.prediction = bestPosition[5];
                        rThetaKal.prevEstimate = bestPosition[4];
                        rRhoKal.prevEstimate = bestPosition[5];
                        rThetaKal.predProb = DEFRTHETA;
                        rRhoKal.predProb = DEFRRHO;
                        rThetaKal.phiOffset = 0;
                        rRhoKal.phiOffset = 0;

                        aquire_count++;
                  }

                  //if not then decrement the count and re-aquire
if necessary
                  else
                  {
                        if(aquire_count>0)
                        {
                              aquire_count--;
                        }
                        else
                        {
                              track_stat=0;
                        }
                  }

                  DrawLine(DestImage,
deltheta*lThetaKal.prediction, delrho*lRhoKal.prediction, 255, 0);
                  DrawLine(DestImage,
deltheta*rThetaKal.prediction, delrho*rRhoKal.prediction, 255, 0);
                  DrawLine(DestImage,
deltheta*cThetaKal.prediction, delrho*cRhoKal.prediction, 255, 0);

                  //if we've found three valid sets of lines then
move to tracking mode.
                  if(aquire_count>=3)
                  {
                        aquire_count=0;
                        track_stat=1;
                  }
            }
      }

      //****************Track the lines************************

      else if ((track_stat==1)||(track_stat==2)||(track_stat==3))
      {
            err = TrackLines(ht, &lThetaKal, &lRhoKal, &cThetaKal,
&cRhoKal, &rThetaKal, &rRhoKal, &eThetaKal, &eRhoKal, &track_stat,
&switch_count, &lose_count, &lose_centre_count);
            if (err == 2)
            {
                  err_code += 4;
            }
            DrawLine(DestImage, deltheta*cThetaKal.measurement,
delrho*cRhoKal.measurement, 255, 0);
            DrawLine(DestImage, deltheta*cThetaKal.estimate,
delrho*cRhoKal.estimate, 255, 255);

            //Draw Outer Lines on output image
            DrawLine(DestImage, deltheta*lThetaKal.measurement,
delrho*lRhoKal.measurement, 255, 0);
```

```
            DrawLine(DestImage, deltheta*lThetaKal.estimate,
delrho*lRhoKal.estimate, 255, 255);
            DrawLine(DestImage, deltheta*rThetaKal.measurement,
delrho*rRhoKal.measurement, 255, 0);
            DrawLine(DestImage, deltheta*rThetaKal.estimate,
delrho*rRhoKal.estimate, 255, 255);
            if((track_stat==2)||(track_stat==3))
            {
                  DrawLine(DestImage,
deltheta*eThetaKal.measurement, delrho*eRhoKal.measurement, 192,
0);
                  DrawLine(DestImage, deltheta*eThetaKal.estimate,
delrho*eRhoKal.estimate, 192, 192);
            }
      }

      //if necessary, update the tracking status
      EnterCriticalSection(&VisionThread::accessLock);
      {
            if(ext_track_stat == old_track_stat) //if the status
hasn't been changed externally
            {
                  if(old_track_stat!=track_stat)   //if the status
has been changed
                  {
                        ext_track_stat = track_stat; //update the
status
                  }
            }
      }
      LeaveCriticalSection(&VisionThread::accessLock);

      //Draw the found Centre line on the output image
      //Copy processed image to the display buffer
      MbufPut(MilDispImage, DestImage->image_data[0]);

      //Write rho and theta values on to output image
      sprintf(thetaOutString, "Theta: %d", cThetaKal.estimate);
      sprintf(rhoOutString, "Rho: %d", cRhoKal.estimate);

      MgraText(M_DEFAULT, MilDispImage, 0, DestImage->GetYSize()-
32, rhoOutString);
      MgraText(M_DEFAULT, MilDispImage, 0, DestImage->GetYSize()-
16, thetaOutString);

      //save the image biffer if required
      if(capProc)
      {
            MbufSave(Fname3,MilDispImage);
      }

      //save error code, and centre line rho and theta.
      EnterCriticalSection(&VisionThread::accessLock);
      {
            ext_err_code = err_code;
            ext_thetaOut = cThetaKal.estimate;
            ext_rhoOut = cRhoKal.estimate;
      }
      LeaveCriticalSection(&VisionThread::accessLock);
}
```

D.2.3 VisionThread::ContEnhance

```
//*****************************************************************
****
// Contrast Enhance an image by Contrast Stretching
// writes new image over the top of the old one
// input/output: source
//*****************************************************************
****
int VisionThread::ContEnhance(ImageObject* source)
{
      int max,min;
      int xsize,ysize;
      int i,j;
      double mulfact,tempval;
      int tempval2;

      //find max and min values in the image
      source->FindMinMax(15);
      max = source->GetMax();
      min = source->GetMin();

      //calculate the multiplication factor
      mulfact = 255/((double)(max-min));

      //read image sizes
      xsize = source->GetXSize();
      ysize = source->GetYSize();

      //step through each pixel in the image to calculate the new
pixel values
      for(j = 0 ; j < ysize ; j++)
      {
            for(i = 0 ; i < xsize ; i++)
            {
                  //subtract minimum pixel value and convert pixel
value to double
                  tempval = (double)(source->image_data[j][i] -
min);
                  //multiply the pixel value by the multiplication
factot
                  tempval *= mulfact;
                  //round the new value to the nearest integer
                  tempval2 = MathFns::Round(tempval);
                  //ensure that the resulting pixel value is in the
range 0-255
                  if(tempval2 > 255)
                  {
                        tempval2 = 255;
                  }
                  if(tempval2 < 0)
                  {
                        tempval2 = 0;
                  }
                  //write new pixel value to the image
                  source->image_data[j][i]=(unsigned char)tempval2;
            }
      }
      return(0);
}
```

D.2.4 VisionThread::EdgeDetect

```
//*****************************************************************
****
// Edge Detector
// Perform sobel edge detector with non-maximum suppression
// input:  source: input image
//         edgeGap: avoid looking at the edge of the image
//         gThres: gradient threshold to use
// output: dest: output edge map
//*****************************************************************
****
int VisionThread::EdgeDetect(ImageObject* source, EdgeMap* dest,
int edgeGap, double gThres)
{
      double dx=0; //value of dI/dx
      double dy=0; //value of dI/dy
      double g=0;  //value of square of image gradient
      double g1,g2; //values of ajacent squares of image gradient
      double ang=0; //value of angle
      int i, j;
      int XSize, YSize;

      //clear the edge map
      dest->SetZeros();

      //check that the edge gap is at least 1
      if (edgeGap < 1)
      {
            edgeGap = 1;
      }

      //read the size of the image to be edge detected
      XSize = source->GetXSize();
      YSize = source->GetYSize();

      //check that the edge map is the right size.
      if((XSize != dest->GetXSize())||(YSize != dest->GetYSize()))
      {
            return(1);
      }

      //*************perform edge filtering*******************

      //step through all required pixels
      for(j = edgeGap ; j < (YSize-edgeGap) ; j++)
      {
            for(i = edgeGap ; i < (XSize-edgeGap) ; i++)
            {
                  //find image gradient at point of interest.
                  g=Grad(source,i,j,&dx,&dy);

                  //if the edge is sufficiently strong, record it
in the edge map
                  if(g > gThres)
                  {
                        if(dx == 0)
                        {
                              ang = -pi/2;
```

```
                        }
                        else
                        {
                              ang = atan(dy/dx);
                        }
                        //record edge value and angle
                        dest->edge_strength_data[j][i]=g;
                        dest->edge_angle_data[j][i]=ang;
                  }
            }
      }

      //*************apply non-maximum suppression**************

      //step through all required pixels
      for(j = edgeGap ; j < (YSize-edgeGap) ; j++)
      {
            for(i = edgeGap ; i < (XSize-edgeGap) ; i++)
            {
                  //read the gradient value and angle from edge map
                  g=dest->edge_strength_data[j][i];
                  ang=dest->edge_angle_data[j][i];
                  if (g>gThres) //if the gradient is larger than
the threshold
                  {
                        //only set edge pixel if it is local
maximum
                        if ((ang < (pi/4)) && (ang >= -(pi/4)))
                        {
                              //read the values of adjacent pixels
                              g1=dest->edge_strength_data[j][i-1];
                              g2=dest->edge_strength_data[j][i+1];
                              //set the output to 255 if it local
maximum
                              if((g>=g1)&&(g>g2))
                              {
                                    dest->thres_edge_data[j][i]=255;
                              }
                        }
                        if (((ang >= (pi/4)) && (ang < (17*pi/36)))
|| ((ang < -(pi/4)) && (ang > -(17*pi/36))))
                        {
                              //read the values of adjacent pixels
                              g1=dest->edge_strength_data[j-1][i];
                              g2=dest->edge_strength_data[j+1][i];
                              //set the output to 255 if it local
maximum
                              if((g>=g1)&&(g>g2))
                              {
                                    dest->thres_edge_data[j][i]=255;
                              }
                        }
                  }
            }
      }
      return(0);
}
```

D.2.5 VisionThread::Hough_Transform

```
//*****************************************************************
****
// Perform hough transform
// Calculate an aggregated Hough transform (AHT) of an edge map.
// inputs: source: input: edge map
//                 edgeGap: amount of the edge of the edge map
that should not be processed.
//                 threshold: value for thresholding the HT
//                 a_size: size of the masks used for aggregation
// output: dest: output Hough transform
// error codes:
//  0 successs
//  1 Hough Transform wrong size.
//  2 Error accessing Hough Transform
//*****************************************************************
****
int VisionThread::Hough_Transform(EdgeMap* source, HoughTransform*
dest, int edgeGap, double threshold, int a_size)
{
      unsigned char edgePixel;
      double ang=0;
      double rho=0;
      int angIndex, rhoIndex;
      int i, j, k;
      int Theta, Rho, Theta1, Rho1, Theta2, Rho2;
      int XCentre, YCentre, XSize, YSize, delRho, delTheta,
ThetaSize, RhoSize, ThetaCentre, RhoCentre;
      unsigned int HTMax, HTThres, value;
      int ha_size;
      int c_numpoints = 81;
      int numpoints = 81;
      int points[162];

      //error checking
      int err = 0;
      int errCode = 0;

      //clear the hough transform
      dest->SetZeros();

      //check that the edge gap is at least 1
      if (edgeGap < 1)
      {
            edgeGap = 1;
      }

      //check that the threshold is in range 0 to 1
      if (threshold < 0)
      {
            threshold = 0;
      }
      if (threshold > 1)
      {
            threshold = 1;
      }

      //check that the aggregation mask size is odd and >=3
      if (a_size < 3)
```

```
	{
		a_size = 3;
	}
	if ((a_size%2) != 1)
	{
		a_size++;
	}
	ha_size = a_size/2;

	//read size info
	XSize = source->GetXSize();
	YSize = source->GetYSize();
	XCentre = MathFns::Round((double)XSize/2);
	YCentre = MathFns::Round((double)YSize/2);
	delTheta = dest->GetThetaQuant();
	delRho = dest->GetRhoQuant();
	ThetaSize = dest->GetThetaSize();
	RhoSize = dest->GetRhoSize();
	ThetaCentre = ThetaSize/2;
	RhoCentre = RhoSize/2;

	//check that the Hough transform is the right size.
	if((XSize != dest->GetXSize())||(YSize != dest->GetYSize()))
	{
		return(1);
	}

	//******************calculate the hough
transform**************

	//step through all required pixels
	for(j = edgeGap ; j < (YSize-edgeGap) ; j++)
	{
		for(i = edgeGap ; i < (XSize-edgeGap) ; i++)
		{
			//read whether current pixel is an edge
			edgePixel = source->thres_edge_data[j][i];
			ang = source->edge_angle_data[j][i];

			//if the edge is sufficiently strong, record it
in the hough transform
			if(edgePixel == 255)
			{
				//quantise angle

	angIndex=MathFns::Round((180*ang/pi)/delTheta)+ThetaCentre;
				//calculate rho
				rho=((j-YCentre)*sin(ang))+((i-
XCentre)*cos(ang));

	rhoIndex=MathFns::Round(rho/delRho)+RhoCentre;
				(dest->HT_data[rhoIndex][angIndex])++;
			}
		}
	}

	//read the max value from the Hough transform
	HTMax = dest->GetMax();
	//set the limits on the Hough Transform threshold
	if(HTMax<5)
	{
```

```
            HTMax=5;
      }
      if(HTMax>25)
      {
            HTMax=25;
      }
      //calculate the threshold to use
      HTThres = (unsigned
int)(MathFns::Round(threshold*(double)HTMax));
      //check that this value is greater than 0 (if 0 will get
image full of lines)
      if(HTThres < 1)
      {
            HTThres = 1;
      }

      //*****************threshold the Hough
Transform******************

      //step through all required pixels
      for(j = 0 ; j < RhoSize ; j++)
      {
            for(i = 0 ; i < ThetaSize ; i++)
            {
                  //threshold each pixel
                  if(dest->HT_data[j][i] >= HTThres)
                  {
                        dest->thres_HT_data[j][i]=255;
                  }
            }
      }

      //*****************aggregate the Hough
Transform*****************

      //step through all required pixels
      for(j = ha_size ; j < (RhoSize - ha_size) ; j++)
      {
            for(i = ha_size ; i < (ThetaSize - ha_size) ; i++)
            {
                  //if the current "pixel" is in the tranform
                  if(dest->thres_HT_data[j][i] == 255)
                  {
                        //find a point to represent the cluster of
points
                        value = FindPoint(&Theta, &Rho, dest, (i -
ha_size), (j - ha_size), a_size);

                        //save the representitive value associated
with the cluster
                        dest->aggreg_values[Rho][Theta] = value;

                        //set the representative point to 0 so it
isn't picked up when finding points in it's vacinity
                        dest->aggreg_HT_data[Rho][Theta] = 0;

                        //find any points around the current
representative point
                        numpoints = c_numpoints;
                        FindPoints(points, &numpoints, dest,
(Theta-ha_size),(Rho - ha_size), a_size, a_size, 0, 2, &err);
```

```
                        if (err != 0)
                        {
                              errCode |= 2;
                        }

                        //if there are no other representitive
points in the vicinity then set this point in the transform
                        if (numpoints == 0)
                        {
                              err=dest->aggreg_HT_data[Rho][Theta]
= 255;
                        }
                        //if there are other points then find the
strongest one and remove all others
                        else
                        {
                              Rho1 = Rho;
                              Theta1 = Theta;
                              for (k = 0 ; k < numpoints ; k++)
                              {
                                    //read point from the list
                                    Rho2 = points[2*k];
                                    Theta2 = points[2*k+1];
                                    //if 1 is larger than 2 set 2 to
0
                                    if(dest-
>aggreg_values[Rho1][Theta1] > dest->aggreg_values[Rho2][Theta2])
                                    {
                                          dest-
>aggreg_HT_data[Rho1][Theta1] = 255;

                                          dest-
>aggreg_HT_data[Rho2][Theta2] = 0;
                                    }
                                    else //otherwise set 2 as the
new 1
                                    {
                                          Rho1 = Rho2;
                                          Theta1 = Theta2;
                                    }
                              }
                        }
                  }
            }
      }
      return(errCode);
}
```

D.2.6 VisionThread::Acquisition

```
//*****************************************************************
****
// Acquisition function
// find the overhead lines in the image using a model of their
expected position
// inputs:  htr: the hough transform to be searched
//          thetadiff, rhodiff, thetavar, rhovar, sideline_fact,
centre_fact:
//              the parameters of the line model
// outputs: bestposition is the best position in form
```

```
//              [leftttheta leftrho centretheta centrerho
righttheta rightrho]
//          bestfound states whether a match has been found 1:
found 0: not found
// return value contains the error code*/
//*****************************************************************
****
int VisionThread::Acquisition(int* bestposition, int* bestfound,
HoughTransform* htr, int thetadiff, int rhodiff, int thetavar, int
rhovar, double sideline_fact, double centre_fact, int fnum)
{
      FILE* file;
      char fname[100];
      char fn[100];
      int err, err2, errCode;
      int i, j, k;
      int numTheta, numRho, h_numTheta, h_numRho;
      int threefound;
      int leftfound, rightfound;
      int maxthetadiff, minthetadiff, maxrhodiff, minrhodiff;
      int tempthetadiff, temprhodiff;

      int leftdist, rightdist, centredist[100];

      int symmetry[100];

      int cbestval, cbestnum;

      double value[100];

      int numpositions, positions_alloc;
      int positions[1000];
      positions_alloc = 100;
      numpositions = 0;

      int numpoints;
      int points[400];
      numpoints = 200;

      errCode=0;

      numTheta=htr->GetThetaSize();
      numRho=htr->GetRhoSize();
      h_numRho = numRho/2;
      h_numTheta = numTheta/2;

      //find points in the hough transform
      err=FindPoints(points, &numpoints, htr, (-h_numTheta), (-
h_numRho), numTheta, numRho, 1, 2, &err2);
      if (err2 != 0)
      {
            errCode |= 2;
      }
      if (err != 0)
      {
            errCode |= 4;
      }

      //set best position to 0
      bestposition[0]=0;
      bestposition[1]=0;
```

```
	bestposition[2]=0;
	bestposition[3]=0;
	bestposition[4]=0;
	bestposition[5]=0;
	positions[0]=0;
	positions[1]=0;
	positions[2]=0;
	positions[3]=0;
	positions[4]=0;
	positions[5]=0;
	positions[6]=0;
	positions[7]=0;
	positions[8]=0;
	positions[9]=0;

	//set the flag to see if we have a three line position to 0
	threefound=0;
	//set the flag for finding a 'best' position to 0
	*bestfound=0;

	//***************find all possible centre lines in the
transform*********

	if(numpoints>0)
	{
		//step through all points assume each may be the
centre line
		for(i = 0 ; i < numpoints; i++)
		{
			//set max and min rho and theta differences
			maxthetadiff=thetadiff+thetavar;
			minthetadiff=thetadiff-thetavar;
			maxrhodiff=rhodiff+rhovar;
			minrhodiff=rhodiff-rhovar;

			//assume the current point is the centre line
			positions[(10*numpositions)+2]=points[2*i];
			positions[(10*numpositions)+3]=points[(2*i)+1];

			//set temporary values for the left and right
lines.
			positions[10*numpositions]=points[2*i];
			positions[(10*numpositions)+1]=points[(2*i)+1];
			positions[(10*numpositions)+4]=points[2*i];
			positions[(10*numpositions)+5]=points[(2*i)+1];
			positions[(10*numpositions)+6]=0;

	positions[(10*numpositions)+7]=(int)((pow(rhodiff,2))+(pow(t
hetadiff,2)));

	positions[(10*numpositions)+8]=(int)((pow(rhodiff,2))+(pow(t
hetadiff,2)));
			positions[(10*numpositions)+9]=0;

			//set found flags to 0
			leftfound=0;
			rightfound=0;

			//check all other points in the transform to see
if they may be on of the sidelines
			for(j = 0 ; j < numpoints ; j++)
```

```
                {
                        //calculate the distances between the two
points being compared
                        tempthetadiff=points[2*i]-points[2*j];
                        temprhodiff=points[(2*i)+1]-
points[(2*j)+1];
                        //if the 'j' point is within the acceptable
area for the left line then record it as such

      if((tempthetadiff<=maxthetadiff)&&(tempthetadiff>=minthetadi
ff)&&(temprhodiff<=maxrhodiff)&&(temprhodiff>=minrhodiff))
                        {
                                //record that we have found a
possible left line
                                leftfound=1;
                                //calculate it's distance from the
ideal
                                leftdist=(int)(pow((tempthetadiff-
thetadiff),2)+pow((temprhodiff-rhodiff),2));
                                //if this is the best left line so
far, record it

      if(leftdist<positions[(10*numpositions)+7])
                                {

      positions[10*numpositions]=points[2*j];

      positions[(10*numpositions)+1]=points[(2*j)+1];

      positions[(10*numpositions)+7]=leftdist;
                                }
                        }
                        //if the 'j' point is within the acceptable
area for the right line then record it as such
                        if((tempthetadiff>=(-
maxthetadiff))&&(tempthetadiff<=(-minthetadiff))&&(temprhodiff>=(-
maxrhodiff))&&(temprhodiff<=(-minrhodiff)))
                        {
                                //record that we have found a
possible left line
                                rightfound=1;
                                //calculate it's distance from the
ideal

      rightdist=(int)(pow((tempthetadiff+thetadiff),2)+pow((temprh
odiff+rhodiff),2));
                                //if this is the best right line so
far, record it

      if(rightdist<positions[(10*numpositions)+8])

                                {

      positions[(10*numpositions)+4]=points[2*j];

      positions[(10*numpositions)+5]=points[(2*j)+1];

      positions[(10*numpositions)+8]=rightdist;
                                }
                        }
                }
```

```
                //if we have found a possible position
                if((leftfound==1)||(rightfound==1))
                {
                        *bestfound=1;

                        //calculate the probability measure and set
the bothlines flag
                        if((leftfound==1)&&(rightfound==1)) //both
lines found
                        {
                                positions[(10*numpositions)+9]=1;

        symmetry[numpositions]=(int)(pow((positions[10*numpositions]
-positions[(10*numpositions)+2]+positions[(10*numpositions)+4]-
positions[(10*numpositions)+2]),2)+pow((positions[(10*numpositions
)+1]-
positions[(10*numpositions)+3]+positions[(10*numpositions)+5]-
positions[(10*numpositions)+3]),2));
                        }
                        else if(leftfound==1)                //only
left line found
                        {
                                positions[(10*numpositions)+9]=2;

        symmetry[numpositions]=(int)(pow((abs(thetadiff)+abs(positio
ns[10*numpositions]-
positions[(10*numpositions)+2]+thetadiff)),2)+pow((abs(rhodiff)+ab
s(positions[(10*numpositions)+1]-
positions[(10*numpositions)+3]+rhodiff)),2));
                        }
                        else if(rightfound==1)               //only
right line found
                        {
                                positions[(10*numpositions)+9]=3;

        symmetry[numpositions]=(int)(pow((abs(thetadiff)+abs(positio
ns[(10*numpositions)+2]-
positions[(10*numpositions)+4]+thetadiff)),2)+pow((abs(rhodiff)+ab
s(positions[(10*numpositions)+3]-
positions[(10*numpositions)+5]+rhodiff)),2));
                        }

        centredist[numpositions]=(int)(pow(positions[(10*numposition
s)+2],2)+pow(positions[(10*numpositions)+3],2));

        value[numpositions]=(symmetry[numpositions]*(1+(sideline_fac
t*(positions[(10*numpositions)+7]+positions[(10*numpositions)+8]))
+(centre_fact*centredist[numpositions])));

        positions[(10*numpositions)+6]=(int)value[numpositions];
                        numpositions++;
                }
        }

        //find the 'best' position, if available
        if((*bestfound)==1)
        {
                //set the current best as the first poisition
                cbestval=positions[6];
                cbestnum=0;
                //compare to remaining positions
```

```
                for(k = 1 ; k < numpositions ; k++)
                {
                        //if the current best is not as good as the
one it is being compared to
                        if(cbestval>positions[(10*k)+6])
                        {
                                cbestnum=k;
                                cbestval=positions[(10*k)+6];
                        }
                }

                //store the best position
                bestposition[0]=positions[10*cbestnum];
                bestposition[1]=positions[(10*cbestnum)+1];
                bestposition[2]=positions[(10*cbestnum)+2];
                bestposition[3]=positions[(10*cbestnum)+3];
                bestposition[4]=positions[(10*cbestnum)+4];
                bestposition[5]=positions[(10*cbestnum)+5];

                //if we haven't found a three line position, then
estimate three lines from two that have been found
                //if the left hand line is found, estimate the
right hand line
                if(positions[(10*cbestnum)+9]==2)
                {

        bestposition[4]=bestposition[2]+bestposition[2]-
bestposition[0];

        bestposition[5]=bestposition[3]+bestposition[3]-
bestposition[1];
                }
                //if the right hand line is found, estimate the
left hand line
                if(positions[(10*cbestnum)+9]==3)
                {

        bestposition[0]=bestposition[2]+bestposition[2]-
bestposition[4];

        bestposition[1]=bestposition[3]+bestposition[3]-
bestposition[5];
                }
        }
    }

    //ensure that search window doesn't go outside the hough
transform
    if(bestposition[0]>(h_numTheta-AQ_EDGE_GAP))
    {
        bestposition[0]=h_numTheta-AQ_EDGE_GAP;
    }
    if(bestposition[0]<(AQ_EDGE_GAP-h_numTheta))
    {
        bestposition[0]=AQ_EDGE_GAP-h_numTheta;
    }
    if(bestposition[1]>(h_numRho-AQ_EDGE_GAP))
    {
        bestposition[1]=h_numRho-AQ_EDGE_GAP;
    }
    if(bestposition[1]<(AQ_EDGE_GAP-h_numRho))
```

```
        {
            bestposition[1]=AQ_EDGE_GAP-h_numRho;
        }
        if(bestposition[2]>(h_numTheta-AQ_EDGE_GAP))
        {
            bestposition[2]=h_numTheta-AQ_EDGE_GAP;
        }
        if(bestposition[2]<(AQ_EDGE_GAP-h_numTheta))
        {
            bestposition[2]=AQ_EDGE_GAP-h_numTheta;
        }
        if(bestposition[3]>(h_numRho-AQ_EDGE_GAP))
        {
            bestposition[3]=h_numRho-AQ_EDGE_GAP;
        }
        if(bestposition[3]<(AQ_EDGE_GAP-h_numRho))
        {
            bestposition[3]=AQ_EDGE_GAP-h_numRho;
        }
        if(bestposition[4]>(h_numTheta-AQ_EDGE_GAP))
        {
            bestposition[4]=h_numTheta-AQ_EDGE_GAP;
        }
        if(bestposition[4]<(AQ_EDGE_GAP-h_numTheta))
        {
            bestposition[4]=AQ_EDGE_GAP-h_numTheta;
        }
        if(bestposition[5]>(h_numRho-AQ_EDGE_GAP))
        {
            bestposition[5]=h_numRho-AQ_EDGE_GAP;
        }
        if(bestposition[5]<(AQ_EDGE_GAP-h_numRho))
        {
            bestposition[5]=AQ_EDGE_GAP-h_numRho;
        }

        return(errCode);
}
```

D.2.7 VisionThread::TrackLines

```
//*****************************************************************
****
// Track Lines
// Interpret and track points from the AHT
// inputs: source: the AHT
// i/o:    lthetakal, lrhokal, cthetakal, crhokal, rthetakal,
rrhokal, ethetakal, erhokal:
//             structures to hold the Kalman filter data
//          switchCount, loseCount, loseCentreCount:
//             fuzzy count values
// error codes:
//  0 success
//  2 Error accessing Hough Transform
//*****************************************************************
****
int VisionThread::TrackLines(HoughTransform* source, struct
kalman* lthetakal, struct kalman* lrhokal, struct kalman*
cthetakal, struct kalman* crhokal, struct kalman* rthetakal,
struct kalman* rrhokal, struct kalman* ethetakal, struct kalman*
```

```
erhokal, int*trackStat, int* switchCount, int* loseCount, int*
loseCentreCount)
{
      int err,err2;
      int errCode = 0;
      int i, j, k;
      FILE* dataOut;
      char dataFName[100];

      //search square sizes
      int l_rect_theta_size, l_rect_rho_size, c_rect_theta_size,
c_rect_rho_size, r_rect_theta_size, r_rect_rho_size,
e_rect_theta_size, e_rect_rho_size;

      //half the search square sizes
      int h_l_rect_theta_size, h_l_rect_rho_size,
h_c_rect_theta_size, h_c_rect_rho_size, h_r_rect_theta_size,
h_r_rect_rho_size, h_e_rect_theta_size, h_e_rect_rho_size;

      //dimensions of ht
      int theta, rho, htheta, hrho;

      //allocate storage for points found in hough transform
      int numlpoints, numcpoints, numrpoints, numepoints;
      int lpoints[600];
      int cpoints[600];
      int rpoints[600];
      int epoints[600];
      numlpoints = numcpoints = numrpoints = numepoints = 200;

      //calculate the square sizes as 3* the standard deviation
      l_rect_theta_size = (int)ceil(1+(6*sqrt(lthetakal-
>predProb)));
      l_rect_rho_size = (int)ceil(1+(6*sqrt(lrhokal->predProb)));
      c_rect_theta_size = (int)ceil(1+(6*sqrt(cthetakal-
>predProb)));
      c_rect_rho_size = (int)ceil(1+(6*sqrt(crhokal->predProb)));
      r_rect_theta_size = (int)ceil(1+(6*sqrt(rthetakal-
>predProb)));
      r_rect_rho_size = (int)ceil(1+(6*sqrt(rrhokal->predProb)));
      e_rect_theta_size = (int)ceil(1+(6*sqrt(ethetakal-
>predProb)));
      e_rect_rho_size = (int)ceil(1+(6*sqrt(erhokal->predProb)));

      //calculate the half square sizes
      h_l_rect_theta_size=l_rect_theta_size/2;
      h_l_rect_rho_size=l_rect_rho_size/2;
      h_c_rect_theta_size=c_rect_theta_size/2;
      h_c_rect_rho_size=c_rect_rho_size/2;
      h_r_rect_theta_size=r_rect_theta_size/2;
      h_r_rect_rho_size=r_rect_rho_size/2;
      h_e_rect_theta_size=e_rect_theta_size/2;
      h_e_rect_rho_size=e_rect_rho_size/2;

      //get HT dimensions
      theta = source->GetThetaSize();
      rho = source->GetRhoSize();
      htheta = MathFns::Round((double)theta/2);
      hrho = MathFns::Round((double)rho/2);

      //find points in the hough transform
```

```
      err=FindPoints(lpoints, &numlpoints, source, (lthetakal-
>prediction)-h_l_rect_theta_size, (lrhokal->prediction)-
h_l_rect_rho_size, l_rect_theta_size, l_rect_rho_size, 1, 3,
&err2);
      if (err2 != 0)
      {
            errCode |= 2;
      }
      err=FindPoints(cpoints, &numcpoints, source, (cthetakal-
>prediction)-h_c_rect_theta_size, (crhokal->prediction)-
h_c_rect_rho_size, c_rect_theta_size, c_rect_rho_size, 1, 3,
&err2);
      if (err2 != 0)
      {
            errCode |= 2;
      }
      err=FindPoints(rpoints, &numrpoints, source, (rthetakal-
>prediction)-h_r_rect_theta_size, (rrhokal->prediction)-
h_r_rect_rho_size, r_rect_theta_size, r_rect_rho_size, 1, 3,
&err2);
      if (err2 != 0)
      {
            errCode |= 2;
      }
      if((*trackStat==2)||(*trackStat==3))
      {
            err=FindPoints(epoints, &numepoints, source,
(ethetakal->prediction)-h_e_rect_theta_size, (erhokal-
>prediction)-h_e_rect_rho_size, e_rect_theta_size,
e_rect_rho_size, 1, 3, &err2);
            if (err2 != 0)
            {
                  errCode |= 2;
            }
      }

      //save data to file
      strcpy(dataFName, directory);
      strcat(dataFName, "\\points.txt");
      dataOut=fopen(dataFName, "a");
      fprintf(dataOut, "Frame %d:\n", frameNumber);
      fprintf(dataOut, "\tRaw Points: Track Status %d\n",
*trackStat);
      fprintf(dataOut, "\t\tLeft Points:\n");
      for(i = 0 ; i < numlpoints ; i++)
      {
            fprintf(dataOut, "\t\t\tTheta: %d Rho: %d\n",
lpoints[3*i], lpoints[3*i+1]);
      }
      fprintf(dataOut, "\n\t\tCentre Points:\n");
      for(i = 0 ; i < numcpoints ; i++)
      {
            fprintf(dataOut, "\t\t\tTheta: %d Rho: %d\n",
cpoints[3*i], cpoints[3*i+1]);
      }
      fprintf(dataOut, "\n\t\tRight Points:\n");
      for(i = 0 ; i < numrpoints ; i++)
      {
            fprintf(dataOut, "\t\t\tTheta: %d Rho: %d\n",
rpoints[3*i], rpoints[3*i+1]);
      }
```

```
	if(((*trackStat)==2)||((*trackStat)==3))
	{
		fprintf(dataOut, "\n\t\tExtra Points:\n");
		for(i = 0 ; i < numepoints ; i++)
		{
			fprintf(dataOut, "\t\t\tTheta: %d Rho: %d\n",
epoints[3*i], epoints[3*i+1]);
		}
	}
	fprintf(dataOut,"\n");

	//*******remove duplicate points (that appear in more than
one search square)********

	//for all points
	i=0;
	while(i < numcpoints)
	{
		j=0;
		while(j < numrpoints)
		{
			//if points match
			if((cpoints[3*i] == rpoints[3*j]) &&
(cpoints[3*i+1] == rpoints[3*j+1]))
			{
				//if centre point is less respresentative
remove it
				if(cpoints[3*i+2] > rpoints[3*j+2])
				{
					numcpoints--;
					for(k = i ; k < numcpoints ; k++)
					{
						cpoints[3*k] = cpoints[3*(k+1)];
						cpoints[3*k+1] =
cpoints[3*(k+1)+1];
						cpoints[3*k+2] =
cpoints[3*(k+1)+2];
					}
					i--; //as we will increment i as we
leave the loop, we need to decrement it as all the points in i
have moved
					break; //as point removed from c, we
don't need to test it against the rest of r
				}
				//otherwise remove the point from the right
search square
				else
				{
					numrpoints--;
					for(k = j ; k < numrpoints ; k++)
					{
						rpoints[3*k] = rpoints[3*(k+1)];
						rpoints[3*k+1] =
rpoints[3*(k+1)+1];
						rpoints[3*k+2] =
rpoints[3*(k+1)+2];
					}
				}
			}
			j++;
		}
```

```
            i++;
      }
      i=0;
      while(i < numcpoints)
      {
            j=1;
            while(j < numlpoints)
            {
                  //if points match
                  if((cpoints[3*i] == lpoints[3*j]) &&
(cpoints[3*i+1] == lpoints[3*j+1]))
                  {
                        //if centre point is less respresentative
remove it
                        if(cpoints[3*i+2] > lpoints[3*j+2])
                        {
                              numcpoints--;
                              for(k = i ; k < numcpoints ; k++)
                              {
                                    cpoints[3*k] = cpoints[3*(k+1)];
                                    cpoints[3*k+1] =
cpoints[3*(k+1)+1];
                                    cpoints[3*k+2] =
cpoints[3*(k+1)+2];
                              }
                              i--; //as we will increment i as we
leave the loop, we need to decrement it as all the points in i
have moved
                              break; //as point removed from c, we
don't need to test it against the rest of r
                        }
                        //otherwise remove the point from the left
search square
                        else
                        {
                              numlpoints--;
                              for(k = j ; k < numlpoints ; k++)
                              {
                                    lpoints[3*k] = lpoints[3*(k+1)];
                                    lpoints[3*k+1] =
lpoints[3*(k+1)+1];
                                    lpoints[3*k+2] =
lpoints[3*(k+1)+2];
                              }
                        }
                  }
                  j++;
            }
            i++;
      }
      i = 0;
      while(i < numlpoints)
      {
            j = 0;
            while(j < numrpoints)
            {
                  //if points match
                  if((lpoints[3*i] == rpoints[3*j]) &&
(lpoints[3*i+1] == rpoints[3*j+1]))
                  {
```

```
                        //if left point is less respresentative
remove it
                        if(lpoints[3*i+2] > rpoints[3*j+2])
                        {
                              numlpoints--;
                              for(k = i ; k < numlpoints ; k++)
                              {
                                    lpoints[3*k] = lpoints[3*(k+1)];
                                    lpoints[3*k+1] =
lpoints[3*(k+1)+1];
                                    lpoints[3*k+2] =
lpoints[3*(k+1)+2];
                              }
                              i--; //as we will increment i as we
leave the loop, we need to decrement it as all the points in i
have moved
                              break; //as point removed from l, we
don't need to test it against the rest of r
                        }
                        //otherwise remove the point from the right
search square
                        else
                        {
                              numrpoints--;
                              for(k = j ; k < numrpoints ; k++)
                              {
                                    rpoints[3*k] = rpoints[3*(k+1)];
                                    rpoints[3*k+1] =
rpoints[3*(k+1)+1];
                                    rpoints[3*k+2] =
rpoints[3*(k+1)+2];
                              }
                        }
                  }
                  j++;
            }
            i++;
      }

      //remove duplicate points from the extra square
      if(((*trackStat)==2)||((*trackStat)==3))
      {
            i=0;
            while(i < numcpoints)
            {
                  j=0;
                  while(j < numepoints)
                  {
                        //if points match
                        if((cpoints[3*i] == epoints[3*j]) &&
(cpoints[3*i+1] == epoints[3*j+1])) //if points match
                        {
                              //if centre point is less
respresentative remove it
                              if(cpoints[3*i+2] > epoints[3*j+2])
                              {
                                    numcpoints--;
                                    for(k = i ; k < numcpoints ;
k++)
                                    {
```

```
                                        cpoints[3*k] =
cpoints[3*(k+1)];
                                        cpoints[3*k+1] =
cpoints[3*(k+1)+1];
                                        cpoints[3*k+2] =
cpoints[3*(k+1)+2];
                                  }
                                  i--; //as we will increment i as
we leave the loop, we need to decrement it as all the points in i
have moved
                                  break; //as point removed from
c, we don't need to test it against the rest of r
                            }
                            //otherwise remove the point from the
extra search square
                            else
                            {
                                  numepoints--;
                                  for(k = j ; k < numepoints ;
k++)
                                  {
                                        epoints[3*k] =
epoints[3*(k+1)];
                                        epoints[3*k+1] =
epoints[3*(k+1)+1];
                                        epoints[3*k+2] =
epoints[3*(k+1)+2];
                                  }
                            }
                      }
                      j++;
                }
                i++;
          }
          i=0;
          while(i < numrpoints)
          {
                j=1;
                while(j < numepoints)
                {
                      //if points match
                      if((rpoints[3*i] == epoints[3*j]) &&
(rpoints[3*i+1] == epoints[3*j+1]))
                      {
                            //if right point is less
respresentative remove it
                            if(rpoints[3*i+2] > epoints[3*j+2])
                            {
                                  numrpoints--;
                                  for(k = i ; k < numrpoints ;
k++)
                                  {
                                        rpoints[3*k] =
rpoints[3*(k+1)];
                                        rpoints[3*k+1] =
rpoints[3*(k+1)+1];
                                        rpoints[3*k+2] =
rpoints[3*(k+1)+2];
                                  }
```

```
                                    i--; //as we will increment i as
we leave the loop, we need to decrement it as all the points in i
have moved
                                    break; //as point removed from
c, we don't need to test it against the rest of r
                              }
                              //otherwise remove the point from the
extra search square
                              else
                              {
                                    numepoints--;
                                    for(k = j ; k < numepoints ;
k++)
                                    {
                                          epoints[3*k] =
epoints[3*(k+1)];
                                          epoints[3*k+1] =
epoints[3*(k+1)+1];
                                          epoints[3*k+2] =
epoints[3*(k+1)+2];
                                    }
                              }
                        }
                        j++;
                  }
                  i++;
            }
            i = 0;
            while(i < numlpoints)
            {
                  j = 0;
                  while(j < numepoints)
                  {
                        //if points match
                        if((lpoints[3*i] == epoints[3*j]) &&
(lpoints[3*i+1] == epoints[3*j+1]))
                        {
                              //if left point is less
respresentative remove it
                              if(lpoints[3*i+2] > epoints[3*j+2])
                              {
                                    numlpoints--;
                                    for(k = i ; k < numlpoints ;
k++)
                                    {
                                          lpoints[3*k] =
lpoints[3*(k+1)];
                                          lpoints[3*k+1] =
lpoints[3*(k+1)+1];
                                          lpoints[3*k+2] =
lpoints[3*(k+1)+2];
                                    }
                                    i--; //as we will increment i as
we leave the loop, we need to decrement it as all the points in i
have moved
                                    break; //as point removed from
l, we don't need to test it against the rest of r
                              }
                              //otherwise remove the point from the
extra search square
                              else
```

```
                              {
                                    numepoints--;
                                    for(k = j ; k < numepoints ;
k++)
                                    {
                                          epoints[3*k] =
epoints[3*(k+1)];
                                          epoints[3*k+1] =
epoints[3*(k+1)+1];
                                          epoints[3*k+2] =
epoints[3*(k+1)+2];
                                    }
                              }
                        }
                        j++;
                  }
                  i++;
            }
      }

      //************record the 'best' point for each
line******************

      //if only 1 point then it is the best
      if(numlpoints == 1)
      {
            lthetakal->measurement = lpoints[0];
            lrhokal->measurement = lpoints[1];
            lthetakal->R = DEFRTHETA;
            lrhokal->R = DEFRRHO;
      }
      //if more than 1 then pick the best
      if(numlpoints > 1)
      {
            //repeat as long as we have more than 1 point
            while(numlpoints > 1)
            {
                  //if the first point is worse than the second,
remove the first and move the second to become the new first
                  if(lpoints[2] > lpoints[5])
                  {
                        lpoints[0]=lpoints[3];
                        lpoints[1]=lpoints[4];
                        lpoints[2]=lpoints[5];
                  }
                  //as we have removed a point, reduce the number
of points
                  numlpoints--;
                  //move the remaining points down
                  for(i = 1 ; i < numlpoints ; i++)
                  {
                        lpoints[3*i]=lpoints[3*(i+1)];
                        lpoints[3*i+1]=lpoints[3*(i+1)+1];
                        lpoints[3*i+2]=lpoints[3*(i+1)+2];
                  }
            }
            //record the best point
            lthetakal->measurement = lpoints[0];
            lrhokal->measurement = lpoints[1];
            lthetakal->R = DEFRTHETA;
            lrhokal->R = DEFRRHO;
```

```
	}

	//if only 1 point then it is the best
	if(numcpoints == 1)
	{
		cthetakal->measurement = cpoints[0];
		crhokal->measurement = cpoints[1];
		cthetakal->R = DEFRTHETA;
		crhokal->R = DEFRRHO;
	}
	//if more than 1 then pick the best
	if(numcpoints > 1)
	{
		//repeat as long as we have more than 1 point
		while(numcpoints > 1)
		{
			//if the first point is worse than the second,
remove the first and move the second to become the new first
			if(cpoints[2] > cpoints[5])
			{
				cpoints[0]=cpoints[3];
				cpoints[1]=cpoints[4];
				cpoints[2]=cpoints[5];
			}
			//as we have removed a point, reduce the number
of points
			numcpoints--;
			//move the remaining points down
			for(i = 1 ; i < numcpoints ; i++)
			{
				cpoints[3*i]=cpoints[3*(i+1)];
				cpoints[3*i+1]=cpoints[3*(i+1)+1];
				cpoints[3*i+2]=cpoints[3*(i+1)+2];
			}
		}
		//record the best point
		cthetakal->measurement = cpoints[0];
		crhokal->measurement = cpoints[1];
		cthetakal->R = DEFRTHETA;
		crhokal->R = DEFRRHO;
	}

	//if only 1 point then it is the best
	if(numrpoints == 1)
	{
		rthetakal->measurement = rpoints[0];
		rrhokal->measurement = rpoints[1];
		rthetakal->R = DEFRTHETA;
		rrhokal->R = DEFRRHO;
	}
	if(numrpoints > 1)
	{
		//repeat as long as we have more than 1 point
		while(numrpoints > 1)
		{
			//if the first point is worse than the second,
remove the first and move the second to become the new first
			if(rpoints[2] > rpoints[5])
			{
				rpoints[0]=rpoints[3];
				rpoints[1]=rpoints[4];
```

```
                        rpoints[2]=rpoints[5];
                    }
                    //as we have removed a point, reduce the number
of points
                    numrpoints--;
                    //move the remaining points down
                    for(i = 1 ; i < numrpoints ; i++)
                    {
                        rpoints[3*i]=rpoints[3*(i+1)];
                        rpoints[3*i+1]=rpoints[3*(i+1)+1];
                        rpoints[3*i+2]=rpoints[3*(i+1)+2];
                    }
                }
                //record the best point
                rthetakal->measurement = rpoints[0];
                rrhokal->measurement = rpoints[1];
                rthetakal->R = DEFRTHETA;
                rrhokal->R = DEFRRHO;
            }

            //if an extra line is being tracked then record its position
            if(((*trackStat)==2)||((*trackStat)==3))
            {
                //if only 1 point then it is the best
                if(numepoints == 1)
                {
                    ethetakal->measurement = epoints[0];
                    erhokal->measurement = epoints[1];
                    ethetakal->R = DEFRTHETA;
                    erhokal->R = DEFRRHO;
                }
                //if more than 1 then pick the best
                if(numepoints > 1)
                {
                    //repeat as long as we have more than 1 point
                    while(numepoints > 1)
                    {
                        //if the first point is worse than the
second, remove the first and move the second to become the new
first
                        if(epoints[2] > epoints[5])
                        {
                            epoints[0]=epoints[3];
                            epoints[1]=epoints[4];
                            epoints[2]=epoints[5];
                        }
                        //as we have removed a point, reduce the
number of points
                        numepoints--;
                        //move the remaining points down
                        for(i = 1 ; i < numepoints ; i++)
                        {
                            epoints[3*i]=epoints[3*(i+1)];
                            epoints[3*i+1]=epoints[3*(i+1)+1];
                            epoints[3*i+2]=epoints[3*(i+1)+2];
                        }
                    }
                    //record the best point
                    ethetakal->measurement = epoints[0];
                    erhokal->measurement = epoints[1];
                    ethetakal->R = DEFRTHETA;
```

```
                erhokal->R = DEFRRHO;
            }
        }

        //save kalman filter data to file
        fprintf(dataOut, "\tKalman Filter Data: Track Status %d\n",
*trackStat);
        fprintf(dataOut, "\t\tLeft Line: numlpoints:
%d\n",numlpoints);
        fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", lthetakal->estimate, lthetakal->prevEstimate, lthetakal-
>prediction, lthetakal->estProb, lthetakal->predProb, lthetakal-
>H, lthetakal->K, lthetakal->phiOffset, lthetakal->Q, lthetakal-
>measurement, lthetakal->R);
        fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", lrhokal->estimate, lrhokal->prevEstimate, lrhokal-
>prediction, lrhokal->estProb, lrhokal->predProb, lrhokal->H,
lrhokal->K, lrhokal->phiOffset, lrhokal->Q, lrhokal->measurement,
lrhokal->R);
        fprintf(dataOut, "\t\tCentre Line: numcpoints:
%d\n",numcpoints);
        fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", cthetakal->estimate, cthetakal->prevEstimate, cthetakal-
>prediction, cthetakal->estProb, cthetakal->predProb, cthetakal-
>H, cthetakal->K, cthetakal->phiOffset, cthetakal->Q, cthetakal-
>measurement, cthetakal->R);
        fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", crhokal->estimate, crhokal->prevEstimate, crhokal-
>prediction, crhokal->estProb, crhokal->predProb, crhokal->H,
crhokal->K, crhokal->phiOffset, crhokal->Q, crhokal->measurement,
crhokal->R);
        fprintf(dataOut, "\t\tRight Line: numrpoints:
%d\n",numrpoints);
        fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", rthetakal->estimate, rthetakal->prevEstimate, rthetakal-
>prediction, rthetakal->estProb, rthetakal->predProb, rthetakal-
>H, rthetakal->K, rthetakal->phiOffset, rthetakal->Q, rthetakal-
>measurement, rthetakal->R);
        fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", rrhokal->estimate, rrhokal->prevEstimate, rrhokal-
>prediction, rrhokal->estProb, rrhokal->predProb, rrhokal->H,
rrhokal->K, rrhokal->phiOffset, rrhokal->Q, rrhokal->measurement,
rrhokal->R);
        if(((*trackStat)==2)||((*trackStat)==3))
        {
            fprintf(dataOut, "\t\tExtra Line: numepoints:
%d\n",numepoints);
            fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d
%f %d %f\n", ethetakal->estimate, ethetakal->prevEstimate,
ethetakal->prediction, ethetakal->estProb, ethetakal->predProb,
ethetakal->H, ethetakal->K, ethetakal->phiOffset, ethetakal->Q,
ethetakal->measurement, ethetakal->R);
            fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d
%f %d %f\n", erhokal->estimate, erhokal->prevEstimate, erhokal-
>prediction, erhokal->estProb, erhokal->predProb, erhokal->H,
erhokal->K, erhokal->phiOffset, erhokal->Q, erhokal->measurement,
erhokal->R);
        }

        //*******ensure that the points conform to the minimum
distance rules**********
```

```
	//ensure the two outer points are at least the required
distance away from the centre point
	if (lthetakal->measurement<(cthetakal->measurement+10))
	{
		lthetakal->measurement=cthetakal->measurement+10;
	}
	if (lrhokal->measurement>(crhokal->measurement-8))
	{
		lrhokal->measurement=crhokal->measurement-8;
	}
	if (rthetakal->measurement>(cthetakal->measurement-10))
	{
		rthetakal->measurement=cthetakal->measurement-10;
	}
	if (rrhokal->measurement<(crhokal->measurement+8))
	{
		rrhokal->measurement=crhokal->measurement+8;
	}

	//ensure that the extra point is at least the required
distance away from the lines
	if ((*trackStat)==3)
	{
		if (ethetakal->measurement<(lthetakal->measurement+8))
		{
			ethetakal->measurement=lthetakal->measurement+8;
		}
		if (erhokal->measurement>(lrhokal->measurement-6))
		{
			erhokal->measurement=lrhokal->measurement-6;
		}
	}

	if ((*trackStat)==2)
	{
		if (ethetakal->measurement>(rthetakal->measurement-8))
		{
			ethetakal->measurement=rthetakal->measurement-8;
		}
		if (erhokal->measurement<(rrhokal->measurement+6))
		{
			erhokal->measurement=rrhokal->measurement+6;
		}
	}

	//********if points are missing, then Kalman Filter may use
spurious data, therefore use the prediction as the
measurement*****

	if(numlpoints==0)
	{
		lthetakal->measurement=lthetakal->prediction;
		lrhokal->measurement=lrhokal->prediction;
		lthetakal->R=4*DEFRTHETA;
		lrhokal->R=4*DEFRRHO;
	}

	if(numcpoints==0)
	{
		cthetakal->measurement=cthetakal->prediction;
```

```
            crhokal->measurement=crhokal->prediction;
            cthetakal->R=4*DEFRTHETA;
            crhokal->R=4*DEFRRHO;
      }

      if(numrpoints==0)
      {
            rthetakal->measurement=rthetakal->prediction;
            rrhokal->measurement=rrhokal->prediction;
            rthetakal->R=4*DEFRTHETA;
            rrhokal->R=4*DEFRRHO;
      }

      if(((*trackStat)==2)|((*trackStat)==3))
      {
            if(numepoints==0)
            {
                  ethetakal->measurement=ethetakal->prediction;
                  erhokal->measurement=erhokal->prediction;
                  ethetakal->R=4*DEFRTHETA;
                  erhokal->R=4*DEFRRHO;
            }
      }

      //****if 2 points found, use those to predict the third
point****

      //if centre point missing, average the other 2
      if((numcpoints == 0) && (numlpoints > 0) && (numrpoints >
0))
      {
            cthetakal-
>measurement=MathFns::Round(((double)lthetakal-
>measurement+(double)rthetakal->measurement)/2);
            crhokal->measurement=MathFns::Round(((double)lrhokal-
>measurement+(double)rrhokal->measurement)/2);
            cthetakal->R=pow((sqrt(lthetakal->R)+sqrt(rthetakal-
>R)),2);
            crhokal->R=pow((sqrt(lrhokal->R)+sqrt(rrhokal->R)),2);
      }

      //if right point missing calculate from other two
      if((numrpoints == 0) && (numlpoints > 0) && (numcpoints >
0))
      {
            rthetakal->measurement=cthetakal-
>measurement+cthetakal->measurement-lthetakal->measurement;
            rrhokal->measurement=crhokal->measurement+crhokal-
>measurement-lrhokal->measurement;
            rthetakal->R=pow((2*sqrt(cthetakal->R)+sqrt(lthetakal-
>R)),2);
            rrhokal->R=pow((2*sqrt(crhokal->R)+sqrt(lrhokal-
>R)),2);
      }

      //if left point missing calculate from other two
      if((numlpoints == 0) && (numrpoints > 0) && (numcpoints >
0))
      {
            lthetakal->measurement=cthetakal-
>measurement+cthetakal->measurement-rthetakal->measurement;
```

```
            lrhokal->measurement=crhokal->measurement+crhokal-
>measurement-rrhokal->measurement;
            lthetakal->R=pow((2*sqrt(cthetakal->R)+sqrt(rthetakal-
>R)),2);
            lrhokal->R=pow((2*sqrt(crhokal->R)+sqrt(rrhokal-
>R)),2);
      }

      //save kalman filter data to file
      fprintf(dataOut, "\n\tKalman Filter Data: Track Status
%d\n", *trackStat);
      fprintf(dataOut, "\t\tLeft Line: numlpoints:
%d\n",numlpoints);
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", lthetakal->estimate, lthetakal->prevEstimate, lthetakal-
>prediction, lthetakal->estProb, lthetakal->predProb, lthetakal-
>H, lthetakal->K, lthetakal->phiOffset, lthetakal->Q, lthetakal-
>measurement, lthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", lrhokal->estimate, lrhokal->prevEstimate, lrhokal-
>prediction, lrhokal->estProb, lrhokal->predProb, lrhokal->H,
lrhokal->K, lrhokal->phiOffset, lrhokal->Q, lrhokal->measurement,
lrhokal->R);
      fprintf(dataOut, "\t\tCentre Line: numcpoints:
%d\n",numcpoints);
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", cthetakal->estimate, cthetakal->prevEstimate, cthetakal-
>prediction, cthetakal->estProb, cthetakal->predProb, cthetakal-
>H, cthetakal->K, cthetakal->phiOffset, cthetakal->Q, cthetakal-
>measurement, cthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", crhokal->estimate, crhokal->prevEstimate, crhokal-
>prediction, crhokal->estProb, crhokal->predProb, crhokal->H,
crhokal->K, crhokal->phiOffset, crhokal->Q, crhokal->measurement,
crhokal->R);
      fprintf(dataOut, "\t\tRight Line: numrpoints:
%d\n",numrpoints);
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", rthetakal->estimate, rthetakal->prevEstimate, rthetakal-
>prediction, rthetakal->estProb, rthetakal->predProb, rthetakal-
>H, rthetakal->K, rthetakal->phiOffset, rthetakal->Q, rthetakal-
>measurement, rthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", rrhokal->estimate, rrhokal->prevEstimate, rrhokal-
>prediction, rrhokal->estProb, rrhokal->predProb, rrhokal->H,
rrhokal->K, rrhokal->phiOffset, rrhokal->Q, rrhokal->measurement,
rrhokal->R);
      if(((*trackStat)==2)||((*trackStat)==3))
      {
            fprintf(dataOut, "\t\tExtra Line: numepoints:
%d\n",numepoints);
            fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d
%f %d %f\n", ethetakal->estimate, ethetakal->prevEstimate,
ethetakal->prediction, ethetakal->estProb, ethetakal->predProb,
ethetakal->H, ethetakal->K, ethetakal->phiOffset, ethetakal->Q,
ethetakal->measurement, ethetakal->R);
            fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d
%f %d %f\n", erhokal->estimate, erhokal->prevEstimate, erhokal-
>prediction, erhokal->estProb, erhokal->predProb, erhokal->H,
erhokal->K, erhokal->phiOffset, erhokal->Q, erhokal->measurement,
erhokal->R);
```

```
	}

	//*********************Apply Kalman
Filter*************************

	//Compute Gain
	lthetakal->K=lthetakal->predProb*lthetakal->H/(lthetakal-
>H*lthetakal->predProb*lthetakal->H+lthetakal->R);
	lrhokal->K=lrhokal->predProb*lrhokal->H/(lrhokal->H*lrhokal-
>predProb*lrhokal->H+lrhokal->R);

	cthetakal->K=cthetakal->predProb*cthetakal->H/(cthetakal-
>H*cthetakal->predProb*cthetakal->H+cthetakal->R);
	crhokal->K=crhokal->predProb*crhokal->H/(crhokal->H*crhokal-
>predProb*crhokal->H+crhokal->R);

	rthetakal->K=rthetakal->predProb*rthetakal->H/(rthetakal-
>H*rthetakal->predProb*rthetakal->H+rthetakal->R);
	rrhokal->K=rrhokal->predProb*rrhokal->H/(rrhokal->H*rrhokal-
>predProb*rrhokal->H+rrhokal->R);

	if(((*trackStat)==2)|((*trackStat)==3))
	{
		ethetakal->K=ethetakal->predProb*ethetakal-
>H/(ethetakal->H*ethetakal->predProb*ethetakal->H+ethetakal->R);
		erhokal->K=erhokal->predProb*erhokal->H/(erhokal-
>H*erhokal->predProb*erhokal->H+erhokal->R);
	}

	//Update Estimate
	lthetakal->estimate=MathFns::Round(lthetakal-
>prediction+(lthetakal->K*(lthetakal->measurement-(lthetakal-
>H*lthetakal->prediction))));
	lrhokal->estimate=MathFns::Round(lrhokal-
>prediction+(lrhokal->K*(lrhokal->measurement-(lrhokal->H*lrhokal-
>prediction))));

	cthetakal->estimate=MathFns::Round(cthetakal-
>prediction+(cthetakal->K*(cthetakal->measurement-(cthetakal-
>H*cthetakal->prediction))));
	crhokal->estimate=MathFns::Round(crhokal-
>prediction+(crhokal->K*(crhokal->measurement-(crhokal->H*crhokal-
>prediction))));

	rthetakal->estimate=MathFns::Round(rthetakal-
>prediction+(rthetakal->K*(rthetakal->measurement-(rthetakal-
>H*rthetakal->prediction))));
	rrhokal->estimate=MathFns::Round(rrhokal-
>prediction+(rrhokal->K*(rrhokal->measurement-(rrhokal->H*rrhokal-
>prediction))));

	if(((*trackStat)==2)|((*trackStat)==3))
	{
		ethetakal->estimate=MathFns::Round(ethetakal-
>prediction+(ethetakal->K*(ethetakal->measurement-(ethetakal-
>H*ethetakal->prediction))));
		erhokal->estimate=MathFns::Round(erhokal-
>prediction+(erhokal->K*(erhokal->measurement-(erhokal->H*erhokal-
>prediction))));
	}
```

```
      //Update Estimate Variance
      lthetakal->estProb=(1-(lthetakal->K*lthetakal-
>H))*lthetakal->predProb;
      lrhokal->estProb=(1-(lrhokal->K*lrhokal->H))*lrhokal-
>predProb;

      cthetakal->estProb=(1-(cthetakal->K*cthetakal-
>H))*cthetakal->predProb;
      crhokal->estProb=(1-(crhokal->K*crhokal->H))*crhokal-
>predProb;

      rthetakal->estProb=(1-(rthetakal->K*rthetakal-
>H))*rthetakal->predProb;
      rrhokal->estProb=(1-(rrhokal->K*rrhokal->H))*rrhokal-
>predProb;

      if(((*trackStat)==2)|((*trackStat)==3))
      {
            ethetakal->estProb=(1-(ethetakal->K*ethetakal-
>H))*ethetakal->predProb;
            erhokal->estProb=(1-(erhokal->K*erhokal->H))*erhokal-
>predProb;
      }

      //calculate phi offset
      lthetakal->phiOffset=lthetakal->estimate-lthetakal-
>prevEstimate;
      lrhokal->phiOffset=lrhokal->estimate-lrhokal->prevEstimate;

      cthetakal->phiOffset=cthetakal->estimate-cthetakal-
>prevEstimate;
      crhokal->phiOffset=crhokal->estimate-crhokal->prevEstimate;

      rthetakal->phiOffset=rthetakal->estimate-rthetakal-
>prevEstimate;
      rrhokal->phiOffset=rrhokal->estimate-rrhokal->prevEstimate;

      if(((*trackStat)==2)|((*trackStat)==3))
      {
            ethetakal->phiOffset=ethetakal->estimate-ethetakal-
>prevEstimate;
            erhokal->phiOffset=erhokal->estimate-erhokal-
>prevEstimate;
      }

      //predict phi offset if one line is missing
      if((numcpoints == 0) && (numlpoints > 0) && (numrpoints >
0))
      {
            cthetakal->phiOffset=MathFns::Round((lthetakal-
>phiOffset+rthetakal->phiOffset)/2);
            crhokal->phiOffset=MathFns::Round((lrhokal-
>phiOffset+rrhokal->phiOffset)/2);
      }

      if((numrpoints == 0) && (numlpoints > 0) && (numcpoints >
0))
      {
            rthetakal->phiOffset=MathFns::Round((lthetakal-
>phiOffset+cthetakal->phiOffset)/2);
```

```
            rrhokal->phiOffset=MathFns::Round((lrhokal-
>phiOffset+crhokal->phiOffset)/2);
      }

      if((numlpoints == 0) && (numrpoints > 0) && (numcpoints >
0))
      {
            lthetakal->phiOffset=MathFns::Round((cthetakal-
>phiOffset+rthetakal->phiOffset)/2);
            lrhokal->phiOffset=MathFns::Round((crhokal-
>phiOffset+rrhokal->phiOffset)/2);
      }

      //Project Ahead
      lthetakal->prediction=MathFns::Round(lthetakal-
>phiOffset+lthetakal->estimate);
      lthetakal->predProb=lthetakal->estProb+lthetakal->Q;
      lrhokal->prediction=MathFns::Round(lrhokal-
>phiOffset+lrhokal->estimate);
      lrhokal->predProb=lrhokal->estProb+lrhokal->Q;

      cthetakal->prediction=MathFns::Round(cthetakal-
>phiOffset+cthetakal->estimate);
      cthetakal->predProb=cthetakal->estProb+cthetakal->Q;
      crhokal->prediction=MathFns::Round(crhokal-
>phiOffset+crhokal->estimate);
      crhokal->predProb=crhokal->estProb+crhokal->Q;

      rthetakal->prediction=MathFns::Round(rthetakal-
>phiOffset+rthetakal->estimate);
      rthetakal->predProb=rthetakal->estProb+rthetakal->Q;
      rrhokal->prediction=MathFns::Round(rrhokal-
>phiOffset+rrhokal->estimate);
      rrhokal->predProb=rrhokal->estProb+rrhokal->Q;

      if(((*trackStat)==2)|((*trackStat)==3))
      {
            ethetakal->prediction=MathFns::Round(ethetakal-
>phiOffset+ethetakal->estimate);
            ethetakal->predProb=ethetakal->estProb+ethetakal->Q;
            erhokal->prediction=MathFns::Round(erhokal-
>phiOffset+erhokal->estimate);
            erhokal->predProb=erhokal->estProb+erhokal->Q;
      }

      //update previous estimate
      lthetakal->prevEstimate=lthetakal->estimate;
      lrhokal->prevEstimate=lrhokal->estimate;

      cthetakal->prevEstimate=cthetakal->estimate;
      crhokal->prevEstimate=crhokal->estimate;

      rthetakal->prevEstimate=rthetakal->estimate;
      rrhokal->prevEstimate=rrhokal->estimate;

      if(((*trackStat)==2)|((*trackStat)==3))
      {
            ethetakal->prevEstimate=ethetakal->estimate;
            erhokal->prevEstimate=erhokal->estimate;
      }
```

```
      //save kalman filter data to file
      fprintf(dataOut, "\n\tKalman Filter Data after KF: Track
Status %d\n", *trackStat);
      fprintf(dataOut, "\t\tLeft Line: numlpoints:
%d\n",numlpoints);
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", lthetakal->estimate, lthetakal->prevEstimate, lthetakal-
>prediction, lthetakal->estProb, lthetakal->predProb, lthetakal-
>H, lthetakal->K, lthetakal->phiOffset, lthetakal->Q, lthetakal-
>measurement, lthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", lrhokal->estimate, lrhokal->prevEstimate, lrhokal-
>prediction, lrhokal->estProb, lrhokal->predProb, lrhokal->H,
lrhokal->K, lrhokal->phiOffset, lrhokal->Q, lrhokal->measurement,
lrhokal->R);
      fprintf(dataOut, "\t\tCentre Line: numcpoints:
%d\n",numcpoints);
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", cthetakal->estimate, cthetakal->prevEstimate, cthetakal-
>prediction, cthetakal->estProb, cthetakal->predProb, cthetakal-
>H, cthetakal->K, cthetakal->phiOffset, cthetakal->Q, cthetakal-
>measurement, cthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", crhokal->estimate, crhokal->prevEstimate, crhokal-
>prediction, crhokal->estProb, crhokal->predProb, crhokal->H,
crhokal->K, crhokal->phiOffset, crhokal->Q, crhokal->measurement,
crhokal->R);
      fprintf(dataOut, "\t\tRight Line: numrpoints:
%d\n",numrpoints);
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", rthetakal->estimate, rthetakal->prevEstimate, rthetakal-
>prediction, rthetakal->estProb, rthetakal->predProb, rthetakal-
>H, rthetakal->K, rthetakal->phiOffset, rthetakal->Q, rthetakal-
>measurement, rthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", rrhokal->estimate, rrhokal->prevEstimate, rrhokal-
>prediction, rrhokal->estProb, rrhokal->predProb, rrhokal->H,
rrhokal->K, rrhokal->phiOffset, rrhokal->Q, rrhokal->measurement,
rrhokal->R);
      if(((*trackStat)==2)||((*trackStat)==3))
      {
            fprintf(dataOut, "\t\tExtra Line: numepoints:
%d\n",numepoints);
            fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d
%f %d %f\n", ethetakal->estimate, ethetakal->prevEstimate,
ethetakal->prediction, ethetakal->estProb, ethetakal->predProb,
ethetakal->H, ethetakal->K, ethetakal->phiOffset, ethetakal->Q,
ethetakal->measurement, ethetakal->R);
            fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d
%f %d %f\n", erhokal->estimate, erhokal->prevEstimate, erhokal-
>prediction, erhokal->estProb, erhokal->predProb, erhokal->H,
erhokal->K, erhokal->phiOffset, erhokal->Q, erhokal->measurement,
erhokal->R);
      }

      //*******************Apply Fuzzy Logic**********************

      //if right point missing set tracking mode to 3 and set up
extra point if necessary
      if((numrpoints == 0) && (numlpoints > 0) && (numcpoints > 0)
&& ((*trackStat)!=3))
```

```
	{
		(*trackStat)=3;
		ethetakal->prediction=MathFns::Round(0.8*(lthetakal-
>prediction+lthetakal->prediction-cthetakal->prediction));
		erhokal->prediction=MathFns::Round(0.8*(lrhokal-
>prediction+lrhokal->prediction-crhokal->prediction));
		ethetakal->prevEstimate=MathFns::Round(0.8*(lthetakal-
>prediction+lthetakal->prediction-cthetakal->prediction));
		erhokal->prevEstimate=MathFns::Round(0.8*(lrhokal-
>prediction+lrhokal->prediction-crhokal->prediction));
		ethetakal->estimate=MathFns::Round(0.8*(lthetakal-
>prediction+lthetakal->prediction-cthetakal->prediction));
		erhokal->estimate=MathFns::Round(0.8*(lrhokal-
>prediction+lrhokal->prediction-crhokal->prediction));
		ethetakal->predProb=pow(0.8*(2*sqrt(lthetakal-
>predProb)+sqrt(cthetakal->predProb)),2);
		erhokal->predProb=pow(0.8*(2*sqrt(lrhokal-
>predProb)+sqrt(crhokal->predProb)),2);
		ethetakal->phiOffset=0;
		erhokal->phiOffset=0;
		*switchCount=0;
		numepoints=-1;
	}

	//if left point missing set tracking mode to 2 and set up
extra point if necessary
	if((numlpoints == 0) && (numrpoints > 0) && (numcpoints > 0)
&& ((*trackStat)!=2))
	{
		(*trackStat)=2;
		ethetakal->prediction=MathFns::Round(0.8*(rthetakal-
>prediction+rthetakal->prediction-cthetakal->prediction));
		erhokal->prediction=MathFns::Round(0.8*(rrhokal-
>prediction+rrhokal->prediction-crhokal->prediction));
		ethetakal->prevEstimate=MathFns::Round(0.8*(rthetakal-
>prediction+rthetakal->prediction-cthetakal->prediction));
		erhokal->prevEstimate=MathFns::Round(0.8*(rrhokal-
>prediction+rrhokal->prediction-crhokal->prediction));
		ethetakal->estimate=MathFns::Round(0.8*(rthetakal-
>prediction+rthetakal->prediction-cthetakal->prediction));
		erhokal->estimate=MathFns::Round(0.8*(rrhokal-
>prediction+rrhokal->prediction-crhokal->prediction));
		ethetakal->predProb=pow(0.8*(2*sqrt(rthetakal-
>predProb)+sqrt(cthetakal->predProb)),2);
		erhokal->predProb=pow(0.8*(2*sqrt(rrhokal-
>predProb)+sqrt(crhokal->predProb)),2);
		ethetakal->phiOffset=0;
		erhokal->phiOffset=0;
		*switchCount=0;
		numepoints=-1;
	}

	//adjust fuzzy value associated with switching from the
sideline
	if(((*trackStat)==2)||((*trackStat)==3))
	{
		//if the extra point is present then increment the
extra point count
		if(numepoints>0)
		{
			(*switchCount)++;
```

```
            }

            //if the extra point is not present then decrement the
extra point count
            if(numepoints==0)
            {
                  (*switchCount)--;
            }

            //if we have all three lines decrement extra point
count, if -3 then set to -3;
            if((numlpoints > 0) & (numrpoints > 0) & (numcpoints >
0))
            {
                  (*switchCount)--;
            }
      }

      //if we find an extra point 3 times in a row, we are
probably on a side line, therefore we need to switch to the centre
line
      if(((*trackStat)==2)&&((*switchCount)>=3))
      {
            lthetakal->estimate=cthetakal->estimate;
            lthetakal->prevEstimate=cthetakal->prevEstimate;
            lthetakal->prediction=cthetakal->prediction;
            lthetakal->estProb=cthetakal->estProb;
            lthetakal->predProb=cthetakal->predProb;
            lthetakal->H=cthetakal->H;
            lthetakal->K=cthetakal->K;
            lthetakal->phiOffset=cthetakal->phiOffset;
            lthetakal->Q=cthetakal->Q;
            lthetakal->measurement=cthetakal->measurement;
            lthetakal->R=cthetakal->R;
            lrhokal->estimate=crhokal->estimate;
            lrhokal->prevEstimate=crhokal->prevEstimate;
            lrhokal->prediction=crhokal->prediction;
            lrhokal->estProb=crhokal->estProb;
            lrhokal->predProb=crhokal->predProb;
            lrhokal->H=crhokal->H;
            lrhokal->K=crhokal->K;
            lrhokal->phiOffset=crhokal->phiOffset;
            lrhokal->Q=crhokal->Q;
            lrhokal->measurement=crhokal->measurement;
            lrhokal->R=crhokal->R;

            cthetakal->estimate=rthetakal->estimate;
            cthetakal->prevEstimate=rthetakal->prevEstimate;
            cthetakal->prediction=rthetakal->prediction;
            cthetakal->estProb=rthetakal->estProb;
            cthetakal->predProb=rthetakal->predProb;
            cthetakal->H=rthetakal->H;
            cthetakal->K=rthetakal->K;
            cthetakal->phiOffset=rthetakal->phiOffset;
            cthetakal->Q=rthetakal->Q;
            cthetakal->measurement=rthetakal->measurement;
            cthetakal->R=rthetakal->R;
            crhokal->estimate=rrhokal->estimate;
            crhokal->prevEstimate=rrhokal->prevEstimate;
            crhokal->prediction=rrhokal->prediction;
            crhokal->estProb=rrhokal->estProb;
```

```
        crhokal->predProb=rrhokal->predProb;
        crhokal->H=rrhokal->H;
        crhokal->K=rrhokal->K;
        crhokal->phiOffset=rrhokal->phiOffset;
        crhokal->Q=rrhokal->Q;
        crhokal->measurement=rrhokal->measurement;
        crhokal->R=rrhokal->R;

        rthetakal->estimate=ethetakal->estimate;
        rthetakal->prevEstimate=ethetakal->prevEstimate;
        rthetakal->prediction=ethetakal->prediction;
        rthetakal->estProb=ethetakal->estProb;
        rthetakal->predProb=ethetakal->predProb;
        rthetakal->H=ethetakal->H;
        rthetakal->K=ethetakal->K;
        rthetakal->phiOffset=ethetakal->phiOffset;
        rthetakal->Q=ethetakal->Q;
        rthetakal->measurement=ethetakal->measurement;
        rthetakal->R=ethetakal->R;
        rrhokal->estimate=erhokal->estimate;
        rrhokal->prevEstimate=erhokal->prevEstimate;
        rrhokal->prediction=erhokal->prediction;
        rrhokal->estProb=erhokal->estProb;
        rrhokal->predProb=erhokal->predProb;
        rrhokal->H=erhokal->H;
        rrhokal->K=erhokal->K;
        rrhokal->phiOffset=erhokal->phiOffset;
        rrhokal->Q=erhokal->Q;
        rrhokal->measurement=erhokal->measurement;
        rrhokal->R=erhokal->R;

        *trackStat=1;
}

if(((*trackStat)==3)&&((*switchCount)>=3))
{
        rthetakal->estimate=cthetakal->estimate;
        rthetakal->prevEstimate=cthetakal->prevEstimate;
        rthetakal->prediction=cthetakal->prediction;
        rthetakal->estProb=cthetakal->estProb;
        rthetakal->predProb=cthetakal->predProb;
        rthetakal->H=cthetakal->H;
        rthetakal->K=cthetakal->K;
        rthetakal->phiOffset=cthetakal->phiOffset;
        rthetakal->Q=cthetakal->Q;
        rthetakal->measurement=cthetakal->measurement;
        rthetakal->R=cthetakal->R;
        rrhokal->estimate=crhokal->estimate;
        rrhokal->prevEstimate=rrhokal->prevEstimate;
        rrhokal->prediction=crhokal->prediction;
        rrhokal->estProb=crhokal->estProb;
        rrhokal->predProb=crhokal->predProb;
        rrhokal->H=crhokal->H;
        rrhokal->K=crhokal->K;
        rrhokal->phiOffset=crhokal->phiOffset;
        rrhokal->Q=crhokal->Q;
        rrhokal->measurement=crhokal->measurement;
        rrhokal->R=crhokal->R;

        cthetakal->estimate=lthetakal->estimate;
        cthetakal->prevEstimate=lthetakal->prevEstimate;
```

```
            cthetakal->prediction=lthetakal->prediction;
            cthetakal->estProb=lthetakal->estProb;
            cthetakal->predProb=lthetakal->predProb;
            cthetakal->H=lthetakal->H;
            cthetakal->K=lthetakal->K;
            cthetakal->phiOffset=lthetakal->phiOffset;
            cthetakal->Q=lthetakal->Q;
            cthetakal->measurement=lthetakal->measurement;
            cthetakal->R=lthetakal->R;
            crhokal->estimate=lrhokal->estimate;
            crhokal->prevEstimate=lrhokal->prevEstimate;
            crhokal->prediction=lrhokal->prediction;
            crhokal->estProb=lrhokal->estProb;
            crhokal->predProb=lrhokal->predProb;
            crhokal->H=lrhokal->H;
            crhokal->K=lrhokal->K;
            crhokal->phiOffset=lrhokal->phiOffset;
            crhokal->Q=lrhokal->Q;
            crhokal->measurement=lrhokal->measurement;
            crhokal->R=lrhokal->R;

            lthetakal->estimate=ethetakal->estimate;
            lthetakal->prevEstimate=ethetakal->prevEstimate;
            lthetakal->prediction=ethetakal->prediction;
            lthetakal->estProb=ethetakal->estProb;
            lthetakal->predProb=ethetakal->predProb;
            lthetakal->H=ethetakal->H;
            lthetakal->K=ethetakal->K;
            lthetakal->phiOffset=ethetakal->phiOffset;
            lthetakal->Q=ethetakal->Q;
            lthetakal->measurement=ethetakal->measurement;
            lthetakal->R=ethetakal->R;
            lrhokal->estimate=erhokal->estimate;
            lrhokal->prevEstimate=erhokal->prevEstimate;
            lrhokal->prediction=erhokal->prediction;
            lrhokal->estProb=erhokal->estProb;
            lrhokal->predProb=erhokal->predProb;
            lrhokal->H=erhokal->H;
            lrhokal->K=erhokal->K;
            lrhokal->phiOffset=erhokal->phiOffset;
            lrhokal->Q=erhokal->Q;
            lrhokal->measurement=erhokal->measurement;
            lrhokal->R=erhokal->R;

            *trackStat=1;
      }

      //if we have failed to find an extra line to the side3 times
in a row then we can assume that we are tracking on the centre
line and the side line has dissappeared
      if((((*trackStat)==2)||((*trackStat)==3))&&((*switchCount)<=
-3))
      {
            *trackStat=1;
      }

      //if we have not found the centre line increment centre lose
count else decrement it
      if(numcpoints == 0)
      {
            (*loseCentreCount)++;
```

```
	}
	else
	{
		(*loseCentreCount)--;
		if((*loseCentreCount)<0)
		{
			(*loseCentreCount)=0;
		}
	}

	//if we have not found the centre line for 5 frames set to
re aquire the lines
	if((*loseCentreCount)>=5)
	{
		*trackStat=0;
		*loseCentreCount=0;
	}

	//if we have only found 1 or 0 line(s) increment a count
	if(((numlpoints >= 0) && (numrpoints == 0) && (numcpoints ==
0))||((numlpoints == 0) && (numrpoints >= 0) && (numcpoints ==
0))||((numlpoints == 0) && (numrpoints == 0) && (numcpoints >=
0)))
	{
		(*loseCount)++;
	}
	else
	{
		(*loseCount)--;
		if((*loseCount)<0)
		{
			(*loseCount)=0;
		}
	}

	//if we have only found 1 or 0 line(s) for 5 frames set to
re aquire the lines
	if((*loseCount)>=5)
	{
		*trackStat=0;
		*loseCount=0;
	}

	//save kalman filter data to file
	fprintf(dataOut, "\n\tKalman Filter Data after fuzzy rules:
Track Status %d\n", *trackStat);
	fprintf(dataOut, "\t\tLeft Line: numlpoints:
%d\n",numlpoints);
	fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", lthetakal->estimate, lthetakal->prevEstimate, lthetakal-
>prediction, lthetakal->estProb, lthetakal->predProb, lthetakal-
>H, lthetakal->K, lthetakal->phiOffset, lthetakal->Q, lthetakal-
>measurement, lthetakal->R);
	fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", lrhokal->estimate, lrhokal->prevEstimate, lrhokal-
>prediction, lrhokal->estProb, lrhokal->predProb, lrhokal->H,
lrhokal->K, lrhokal->phiOffset, lrhokal->Q, lrhokal->measurement,
lrhokal->R);
	fprintf(dataOut, "\t\tCentre Line: numcpoints:
%d\n",numcpoints);
```

```
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", cthetakal->estimate, cthetakal->prevEstimate, cthetakal-
>prediction, cthetakal->estProb, cthetakal->predProb, cthetakal-
>H, cthetakal->K, cthetakal->phiOffset, cthetakal->Q, cthetakal-
>measurement, cthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", crhokal->estimate, crhokal->prevEstimate, crhokal-
>prediction, crhokal->estProb, crhokal->predProb, crhokal->H,
crhokal->K, crhokal->phiOffset, crhokal->Q, crhokal->measurement,
crhokal->R);
      fprintf(dataOut, "\t\tRight Line: numrpoints:
%d\n",numrpoints);
      fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d %f %d
%f\n", rthetakal->estimate, rthetakal->prevEstimate, rthetakal-
>prediction, rthetakal->estProb, rthetakal->predProb, rthetakal-
>H, rthetakal->K, rthetakal->phiOffset, rthetakal->Q, rthetakal-
>measurement, rthetakal->R);
      fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d %f %d
%f\n", rrhokal->estimate, rrhokal->prevEstimate, rrhokal-
>prediction, rrhokal->estProb, rrhokal->predProb, rrhokal->H,
rrhokal->K, rrhokal->phiOffset, rrhokal->Q, rrhokal->measurement,
rrhokal->R);
      if(((*trackStat)==2)||((*trackStat)==3))
      {
            fprintf(dataOut, "\t\tExtra Line: numepoints:
%d\n",numepoints);
            fprintf(dataOut, "\t\t\tTheta: %d %d %d %f %f %f %f %d
%f %d %f\n", ethetakal->estimate, ethetakal->prevEstimate,
ethetakal->prediction, ethetakal->estProb, ethetakal->predProb,
ethetakal->H, ethetakal->K, ethetakal->phiOffset, ethetakal->Q,
ethetakal->measurement, ethetakal->R);
            fprintf(dataOut, "\t\t\tRho: %d %d %d %f %f %f %f %d
%f %d %f\n", erhokal->estimate, erhokal->prevEstimate, erhokal-
>prediction, erhokal->estProb, erhokal->predProb, erhokal->H,
erhokal->K, erhokal->phiOffset, erhokal->Q, erhokal->measurement,
erhokal->R);
      }

      //calculate the square sizes
      l_rect_theta_size = (int)ceil(1+(6*sqrt(lthetakal-
>predProb)));
      l_rect_rho_size = (int)ceil(1+(6*sqrt(lrhokal->predProb)));
      c_rect_theta_size = (int)ceil(1+(6*sqrt(cthetakal-
>predProb)));
      c_rect_rho_size = (int)ceil(1+(6*sqrt(crhokal->predProb)));
      r_rect_theta_size = (int)ceil(1+(6*sqrt(rthetakal-
>predProb)));
      r_rect_rho_size = (int)ceil(1+(6*sqrt(rrhokal->predProb)));
      e_rect_theta_size = (int)ceil(1+(6*sqrt(ethetakal-
>predProb)));
      e_rect_rho_size = (int)ceil(1+(6*sqrt(erhokal->predProb)));

      //calculate the half square sizes
      h_l_rect_theta_size=l_rect_theta_size/2;
      h_l_rect_rho_size=l_rect_rho_size/2;
      h_c_rect_theta_size=c_rect_theta_size/2;
      h_c_rect_rho_size=c_rect_rho_size/2;
      h_r_rect_theta_size=r_rect_theta_size/2;
      h_r_rect_rho_size=r_rect_rho_size/2;
      h_e_rect_theta_size=e_rect_theta_size/2;
      h_e_rect_rho_size=e_rect_rho_size/2;
```

```
	//ensure the two outer points are at least the required
distance away from the centre point
	if (lthetakal->prediction<(cthetakal->prediction+10))
	{
		lthetakal->prediction=cthetakal->prediction+10;
	}
	if (lrhokal->prediction>(crhokal->prediction-8))
	{
		lrhokal->prediction=crhokal->prediction-8;
	}
	if (rthetakal->prediction>(cthetakal->prediction-10))
	{
		rthetakal->prediction=cthetakal->prediction-10;
	}
	if (rrhokal->prediction<(crhokal->prediction+8))
	{
		rrhokal->prediction=crhokal->prediction+8;
	}

	//ensure that the extra point is at least the required
distance away from the lines
	if ((*trackStat)==3)
	{
		if (ethetakal->prediction<(lthetakal->prediction+8))
		{
			ethetakal->prediction=lthetakal->prediction+8;
		}
		if (erhokal->prediction>(lrhokal->prediction-6))
		{
			erhokal->prediction=lrhokal->prediction-6;
		}
	}

	if ((*trackStat)==2)
	{
		if (ethetakal->prediction>(rthetakal->prediction-8))
		{
			ethetakal->prediction=rthetakal->prediction-8;
		}
		if (erhokal->prediction<(rrhokal->prediction+6))
		{
			erhokal->prediction=rrhokal->prediction+6;
		}
	}

	//ensure that search window doesn't go outside the hough
transform
	if(lthetakal->prediction>(htheta-h_l_rect_theta_size))
	{
		lthetakal->prediction=htheta-h_l_rect_theta_size;
	}
	if(lthetakal->prediction<(h_l_rect_theta_size-htheta))
	{
		lthetakal->prediction=h_l_rect_theta_size-htheta;
	}
	if(lrhokal->prediction>(hrho-h_l_rect_rho_size))
	{
		lrhokal->prediction=hrho-h_l_rect_rho_size;
	}
	if(lrhokal->prediction<(h_l_rect_rho_size-hrho))
```

```
	{
		lrhokal->prediction=h_l_rect_rho_size-hrho;
	}
	if(cthetakal->prediction>(htheta-h_c_rect_theta_size))
	{
		cthetakal->prediction=htheta-h_c_rect_theta_size;
	}
	if(cthetakal->prediction<(h_c_rect_theta_size-htheta))
	{
		cthetakal->prediction=h_c_rect_theta_size-htheta;
	}
	if(crhokal->prediction>(hrho-h_c_rect_rho_size))
	{
		crhokal->prediction=hrho-h_c_rect_rho_size;
	}
	if(crhokal->prediction<(h_c_rect_rho_size-hrho))
	{
		crhokal->prediction=h_c_rect_rho_size-hrho;
	}
	if(rthetakal->prediction>(htheta-h_r_rect_theta_size))
	{
		rthetakal->prediction=htheta-h_r_rect_theta_size;
	}
	if(rthetakal->prediction<(h_r_rect_theta_size-htheta))
	{
		rthetakal->prediction=h_r_rect_theta_size-htheta;
	}
	if(rrhokal->prediction>(hrho-h_r_rect_rho_size))
	{
		rrhokal->prediction=hrho-h_r_rect_rho_size;
	}
	if(rrhokal->prediction<(h_r_rect_rho_size-hrho))
	{
		rrhokal->prediction=h_r_rect_rho_size-hrho;
	}
	if(ethetakal->prediction>(htheta-h_e_rect_theta_size))
	{
		ethetakal->prediction=htheta-h_e_rect_theta_size;
	}
	if(ethetakal->prediction<(h_e_rect_theta_size-htheta))
	{
		ethetakal->prediction=h_e_rect_theta_size-htheta;
	}
	if(erhokal->prediction>(hrho-h_e_rect_rho_size))
	{
		erhokal->prediction=hrho-h_e_rect_rho_size;
	}
	if(erhokal->prediction<(h_e_rect_rho_size-hrho))
	{
		erhokal->prediction=h_e_rect_rho_size-hrho;
	}

	//save fianl points to file
	fprintf(dataOut, "\tFinal Square Centres: Track Status
%d\n", *trackStat);
/*	fprintf(dataOut, "\t\tLeft Line:\n\t\t\tTheta: %d Rho: %d
Size: %d\n\n",sSquares[0], sSquares[1], sSquares[2]);
	fprintf(dataOut, "\t\tCentre Line:\n\t\t\tTheta: %d Rho: %d
Size: %d\n\n",sSquares[3], sSquares[4], sSquares[5]);
	fprintf(dataOut, "\t\tRight Line:\n\t\t\tTheta: %d Rho: %d
Size: %d\n\n",sSquares[6], sSquares[7], sSquares[8]);
```

```
	if(((*trackStat)==2)||((*trackStat)==3))
	{
		fprintf(dataOut, "\t\tExtra Line:\n\t\t\tTheta: %d
Rho: %d Size: %d\n",extraSquare[0], extraSquare[1],
extraSquare[2]);
	}*/

	fclose(dataOut);

	return (errCode);
}
```

D.2.8 VisionThread::Grad

```
//*****************************************************************
****
// Image Gradient Function
// Calculate the gradient at a pixel in the image
// inputs:  image: the image we want the gardient of
//          xpos, ypos: position in the image that we want the
gradient
// outputs: dx, dy: gradients in x and y direction
// return value: magnmitude of the gradient
//*****************************************************************
****
double VisionThread::Grad(ImageObject* image, int xpos, int ypos,
double* dx, double* dy)
{
	double mag,xh,yv,xd,yd,d1,d2;

	//calculate the gradient components
	xh = (((double)image->image_data[ypos][xpos+1]) -
((double)image->image_data[ypos][xpos-1]))/2; //calculate the
portion of dI/dx due to horizontal
	yv = (((double)image->image_data[ypos+1][xpos]) -
((double)image->image_data[ypos-1][xpos]))/2; //calculate the
portion of dI/dy due to vertical
	d1 = (((double)image->image_data[ypos+1][xpos+1]) -
((double)image->image_data[ypos-1][xpos-1]))/4;
	d2 = (((double)image->image_data[ypos-1][xpos+1]) -
((double)image->image_data[ypos+1][xpos-1]))/4;

	//calculate the gradients
	xd = d1+d2;
	yd = d1-d2;

	*dx = (xh+xd)/2; //calculate dI/dx
	*dy = (yv+yd)/2; //calculate dI/dy

	mag = ((*dx)*(*dx)) + ((*dy)*(*dy));

	return (mag);
}
```

D.2.9 VisionThread::FindPoint

```
//*****************************************************************
****
```

```
// Find Point
// Find a point to represent a cluster of points
// inputs:  trans: the hough transform that the point is to be
found in
//          xbase, ybase: position of the aggregation square
//          c_size: size of the aggregation square
// outputs: xout, yout: the position of the representative cluster
// return value is a representative value
//*****************************************************************
****
unsigned int VisionThread::FindPoint(int* xout, int* yout,
HoughTransform* trans, int xbase, int ybase, int c_size)
{
      unsigned int value = 0;
      int xsum = 0;
      int ysum = 0;
      int num = 0;
      for(int j = ybase ; j < (ybase + c_size) ; j++)
      {
            for(int i = xbase ; i < (xbase + c_size) ; i++)
            {
                  if (trans->thres_HT_data[j][i] == 255)
                  {
                        xsum += i;
                        ysum += j;
                        num++;
                        value++;
                  }
            }
      }
      *xout = MathFns::Round((double)xsum / (double)num);
      *yout = MathFns::Round((double)ysum / (double)num);
      return(value);
}
```

D.2.10 VisionThread::FindPoints

```
//*****************************************************************
****
// find points in the region of a point
// inputs:  trans: the hough transform to be searched for points
//          thetabase, rhobase: position of the search square
//          theta_size  rho_size: search square size
//          centre sets the centre point that thetabase and
rhobase are relative to:
//              0 = top left corner
//              1 = image centre
//          mode sets whether only the position is recorded, or
whether the square of the distance from the point and the search
square centre is also recorded:
//              2 = position only
//              3 = position and distance
//          note that in mode 2 outpoints must contain twice as
many elements as outnumpoints, in mode 3 there needs to be 3 times
as many elements as outnumpoints
// outputs: outpoints: the points found in the region of the
transform
//          outnumpoints: the number of points found
//          if there is an error accessing the HT, HTAccessError
is set to 2, otherwise it is 0
```

```
// returns 0 if successful
// returns number of extra memory elements required if not
successful
//*****************************************************************
****
int VisionThread::FindPoints(int* outpoints, int* outnumpoints,
HoughTransform* trans, int thetabase, int rhobase, int theta_size,
int rho_size, int centre, int mode, int* HTAccessError)
{
      unsigned char tempElement = 0;
      int num = 0;
      int over = 0;
      int theta, htheta, rho, hrho, thetacentre, rhocentre;

      //get HT dimensions
      theta = trans->GetThetaSize();
      rho = trans->GetRhoSize();
      htheta = MathFns::Round((double)theta/2);
      hrho = MathFns::Round((double)rho/2);

      //clear HT Access error code
      *HTAccessError = 0;

      //if we are working releative to the image centre then
adjust the base.
      if (centre == 1)
      {
            thetabase += htheta;
            rhobase += hrho;
      }

      //calculate the search square centre
      thetacentre = thetabase + (theta_size/2);
      rhocentre = rhobase + (rho_size/2);

      //for all points in the search square
      for(int j = rhobase ; j < (rhobase + rho_size) ; j++)
      {
            for(int i = thetabase ; i < (thetabase + theta_size) ;
i++)
            {
                  tempElement = trans->aggreg_HT_data[j][i];
                  if (tempElement == 1)
                  {
                        *HTAccessError = 2;
                  }
                  //if a point is found
                  if (tempElement == 255)
                  {
                        //record the points
                        if (centre == 1)
                        {
                              outpoints[mode*num] = i-htheta;
                              outpoints[mode*num+1] = j-hrho;
                        }
                        else
                        {
                              outpoints[mode*num] = i;
                              outpoints[mode*num+1] = j;
                        }
                        if (mode == 3)
```

```
                        {
                              outpoints[mode*num+2] = ((i-
thetacentre)*(i-thetacentre))+((j-rhocentre)*(j-rhocentre));
                        }
                        //if the buffer is full, record the amount
of extra space required
                        if (num < ((*outnumpoints)-1))
                        {
                              num++;
                        }
                        else
                        {
                              over++;
                        }
                  }
            }
      }
      *outnumpoints = num;
      return(over);
}
```

Bibliography

1. Golightly, I.T. and Jones, D.I., *Visual control of an unmanned aerial vehicle for power line inspection.* in *Proc. IEEE Int Conf Advanced Robotics (ICAR 2005).* 2005. Seattle, USA: pp. 288-295.
2. Jones, D., Golightly, I., Roberts, J., Usher, K. and Earp, G., *Power line inspection - a UAV concept.* in *IEE Forum on: Autonomous Systems.* 2005. London, UK.
3. Jones, D., Golightly, I., Roberts, J. and Usher, K., *Modeling and Control of a Robotic Power Line Inspection Vehicle.* in *Proc. IEEE International Conference on Control Applications (CCA 2006).* 2006. Munich, Germany.
4. Jones, D.I., *Aerial Inspection of Overhead Power Lines using Video: Estimation of Image Blurring due to Vehicle and Camera Motion.* Proc. IEE Vision, Image Signal Processing, 2000. **147**(2): pp. 157-166.
5. Jones, D.I. and Earp, G.K., *Requirements for aerial inspection of overhead electrical power lines.* in *12th International Conference on Remotely Piloted Vehicles.* 1996. Bristol: pp. Paper 4.
6. Jones, D.I. and Earp, G.K., *Camera sightline pointing requirements for aerial inspection of overhead power lines.* Electric Power Systems Research, 2001. **57**: pp. 73-82.
7. Jones, D.I., Whitworth, C.C., Earp, G.K. and Duller, A.W.G., *A Laboratory Test-Bed for an Automated Power Line Inspection System.* Control Engineering Practice, 2005. **13**(7): pp. 835-851.
8. Golightly, I. and Jones, D., *Corner detection and matching for visual tracking during power line inspection.* Image and Vision Computing, 2003. **21**(9): pp. 827-840.
9. Whitworth, C.C., Duller, A.W.G., Jones, D.I. and Earp, G.K., *Aerial Video Inspection of Overhead Power Lines.* Power Engineering Journal, 2001. **15**(1): pp. 25-32.
10. Robertson, A.C., Stuart, J. and Wagner, R.A., *Vertical take-off flying platform*, in *U. S. Patent*, 1960: U.S.A.
11. Sherman, D., Vidal, G., Marchica, G. and Johnson, R., *University of Central Florida: Technical report: AUVS International Aerial Robotics Competition.* in *AUVSI '96 Proceedings.* 1996: pp. 955-64.
12. Prouty, R.W., *Helicopter Performance, Stability and Control.* 1995, Malabar, Florida: Krieger.
13. Ando, S., *A Simple Theory on Hovering Stability of One Ducted Fan VTOL.* Transactions - Japan Society for Aeronautical and Space Sciences, 1987. **29**(86): pp. 242-250.
14. Hamel, T. and Mahony, R., *Visual Servoing of an Under-Actuated Dynamic Rigid-Body System: An Image-Based Approach.* IEEE Transactions on Robotics and Automation, 2002. **18**(2): pp. 187-198.
15. Mahony, R. and Hamel, T., *Image-Based Visual Servo Control of Aerial Robotic Systems Using Linear Image Features.* IEEE Transactions on Robotics and Automation, 2005. **21**(2): pp. 227-239.
16. CAA, *CAP 722: Unmanned Aerial Vehicle Operations in UK Airspace - Guidance.* TSO, 2004.

17. Williams, M., *Investigation of Machine Vision and Path Planning Methods for use in an Autonomous Unmanned Air Vehicle*, Ph.D. Thesis, *School of Informatics, University of Wales, Bangor*, Bangor, U.K., 2000.
18. CAA, *CAP 658: Model Aircraft: A Guide to Safe Flying*. TSO, 2003.
19. Austin, R.G. and Earp, G., *Power Line Inspection by UAV - A Business Case*. in *Proc. 19th International Conference on Unmanned Air Vehicle Systems*. 2004. Bristol U.K.: pp. 14.1-14.13.
20. Espiau, B., Chaumette, F. and Rives, P., *A New Approach to Visual Servoing in Robotics*. IEEE Transactions on Robotics and Automation, 1992. **8**(3): pp. 313-326.
21. Chaumette, F., *Image Moments: A General and Useful Set of Features for Visual Servoing*. IEEE Transactions on Robotics, 2004. **20**(4): pp. 713-723.
22. Hill, J. and Park, W.T., *Real time control of a robot with mobile camera*. in *Proc. 9th ISIR*. 1979. Washington D.C., USA: pp. 223-246.
23. Hutchinson, S., Hager, G.D. and Corke, P.I., *A Tutorial on Visual Servo Control*. IEEE Transactions on Robotics and Automation, 1996. **12**(5): pp. 651-670.
24. Davison, A.J. and Murray, D.W., *Simultaneous localization and map-building using active vision*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002. **24**(7): pp. 865-80.
25. Carelli, R., Kelly, R., Nasisi, O.H., Soria, C. and Mut, V., *Control based on perspective lines of a non-holonomic mobile robot with camera-on-board*. International Journal of Control, 2006. **79**(4): pp. 362-371.
26. Dickmanns, E.D., *Expectation-based, multi-focal, saccadic (EMS) vision for ground vehicle guidance*. Control Engineering Practice, 2002. **10**: pp. 907-915.
27. Dickmanns, E.D., *The development of machine vision for road vehicles in the last decade*. in *Proc. IEEE Intelligent Vehicle Symposium*. 2003. Piscataway, NJ, USA: pp. 268-281.
28. Hofmann, U., Rieder, A. and Dickmanns, E.D., *Radar and vision data fusion for hybrid adaptive cruise control on highways*. Machine Vision and Applications, 2003. **14**(1): pp. 42-49.
29. Kiy, K.I. and Dickmanns, E.D., *A color vision system for real-time analysis of road scenes*. in *Proc. IEEE Intelligent Vehicles Symposium*. 2004. Parma, Italy: pp. 54-59.
30. Ollero, A. and Merino, L., *Control and perception techniques for aerial robotics*. Annual Reviews in Control, 2004. **28**(2): pp. 167-178.
31. Kontitsis, M., Valavanis, K.P. and Garcia, R., *A simple low cost vision system for small unmanned VTOL vehicles*. in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005: pp. 3480-3486.
32. Hrabar, S., Sukhatme, G.S., Corke, P., Usher, K. and Roberts, J., *Combined optic-flow and stereo-based navigation of urban canyons for a UAV*. in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005: pp. 3309-3316.
33. Mejias, L., Saripalli, S., Campoy, P. and Sukhatme, G.S., *Visual Servoing of an Autonomous Helicopter in Urban Areas using Feature Tracking*. Journal of Field Robotics, 2006. **23**(3): pp. 185-199.

34. Amidi, O., Kanade, T. and Fujita, K., *A Visual Odometer For Autonomous Helicopter Flight.* Robotics and Autonomous Systems, 1998. **28**(3): pp. 185-193.
35. Amidi, O., Kanade, T. and Miler, R., *Vision-Based Autonomous Helicopter Research at Carnegie Mellon Robotics Institute 1991-1997.* in *American Helicopter Society.* 1998: pp. 1-12.
36. Del-Cerro, J., Aguirre, I. and Barrientos, A., *Development of a Low Cost Autonomous Minihelicopter for Power lines Inspections.* in *Mobile Robots; Telemanipulator and Telepresence Technologies.* 2001. Boston, MA: pp. 1-7.
37. Campoy, P., Barrientos, A., Garcia, P.J., Cerro, J. and Aguirre, I., *An autonomous helicopter guided by computer vision for visual inspection of overhead power cable.* in *5th International Conference on Live Maintenance.* 2000. Madrid, Spain.
38. Mejias, L., Campoy, P., Usher, K., Roberts, J. and Corke, P., *Two Seconds to Touchdown – Vision-Based Controlled Forced Landing.* in *Proc. IEEE/RSJ Conf Intelligent Robotics & Systems.* 2006. Beijing, China.
39. Usher, K., Winstanley, G., Roberts, J., Overs, L. and Corke, P., *AVS: Air/Aqua Vehicle Simulator*, Technical Report, *CSIRO*, Kenmore, QLD, 2005
40. Usher, K., Winstanley, G., Corke, P., Stauffacher, D. and Carnie, R., *Air Vehicle Simulator: an Application for a Cable Array Robot.* in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA2005).* 2005. Barcelona, Spain: pp. 2241-2246.
41. Davison, A.J., *Modelling the world in real time: how robots engineer information.* Philosophical Transactions of the Royal Society of London Series A-Physical Sciences & Engineering, 2003. **361**(1813): pp. 2875-2890.
42. Kalman, R.E., *A New Approach to Linear Filtering and Prediction Problems.* Transactions of the ASME Journal of Basic Engineering, 1960. **82**(Series D): pp. 35-45.
43. Gordon, N.J., Salmond, D.J. and Smith, A.F.M., *Novel Approach to nonlinear/non-Gaussian Bayesian state estimation.* IEE Proceedings-F, 1993. **140**(2): pp. 107-113.
44. El-Hawary, M.E., *Electric Power Applications of Fuzzy Systems.* 1998: IEEE Press.
45. Yen, J. and Langari, R., *Fuzzy Logic: Intelligence, Control and Information.* 1999: Prentice-Hall.
46. Brown, R.G. and Hwang, P.Y.C., *Introduction to Random Signals and Applied Kalman Filtering.* 3rd Ed. ed. 1997: Wiley.
47. Phillips, C.L. and Harbor, R.D., *Feedback Control Systems.* 4th Ed. ed. 2000: Prentice-Hall.
48. Gonzalez, R.C. and Woods, R.E., *Digital Image Processing.* World Student Series. 1993: Addison-Wesley.
49. Short Brothers and Harland Ltd, *Cathode ray tube displays for the transition and landing of V.T.O.L. aircraft, Short Brothers and Harland*, Castlereagh, 1961
50. Ryu, J., Rossetter, E.J. and Gerdes, J.C., *Vehicle Sideslip and Roll Parameter Estimation using GPS.* in *AVEC 2002 6th Int. Symposium on Advanced Vehicle Control.* 2002. Hiroshima, Japan.

51. Ogata, K., *Modern Control Engineering*. 3rd ed. 1997: Prentice-Hall.
52. Jackson, L.B., *Digital Filters and Signal Processing*. 2nd ed. 1989: Kluwer.
53. Hough, P.V.C., *Methods and Means for Recognising Complex Patterns*, in *U.S. Patent No*, 1962: U.S.A.
54. Sobel, I.E., *Camera Models and Machine Perception*, Ph.D. Thesis, *Electrical Engineering Dept., Stanford University*, Stanford, CA, 1970.
55. Haralick, R.M. and Shapiro, L.G., *Computer and Robot Vision*. Vol. 1. 1992: Addison-Wesley.
56. Canny, J., *A Computational Approach to Edge Detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986. **8**(6).
57. The Mathworks, *MATLAB Image Processing Toolbox Edge Detector: edge.m* V5.0.2, 2005
58. Doucet, A., de Freitas, N. and Gordon, N., *Sequential Monte Carlo Methods in Practice*. 2001: Springer.
59. Shapiro, L.S., *Affine analysis of image sequences*. 1995: Cambridge University Press.
60. Omidvar, O. and Elliott, D.L., *Neural Systems for Control*. 1997, San Diego: Academic Press.
61. Jones, D.I., Whitworth, C.C. and Duller, A.W.G., *Image Processing Methods for the Visual Location of Power Line Poles*. in *Proc. 7th irish Machine Vision Conference (IMVIP2003)*. 2003. Portrush, Northern Ireland: pp. 177-184.