SCHOOL OF ENGINEERING
INSTITUTE OF TECHNOLOGY, SLIGO

An Investigation of the Use of Vision Systems for the Robotic Control of Automated Vehicles

Research Student
Seán Mullery BEng.

Submitted for the Degree of Master of Engineering

Research Supervisor
Fergal Henry B.E. MEng Sc.

Submitted to the Institute of Technology, Sligo, September 2007

ABSTRACT

An Investigation of the Use of Vision Systems for Robotic Control of Automated Vehicles

by

Seán Mullery BEng.

Humans' usage of the motor vehicle for transport and freight is ever increasing. It seems that the risks and level of accident rates associated with these traffic systems can only be lessened by increasingly complex systems, which aid the human driver with the task or take some of the task out of the human driver's control. However, any semi or fully autonomous vehicle must conform to the type of roads currently in existence. Those roads are designed to suit a human sensory system, mainly the vision sense. Therefore, it would seem that computer vision type systems would be the largest contributor to autonomous vehicles in the short to medium term future. This work investigates how vision systems have already been used in one type of autonomous vehicle task, namely "lane detection and following". It also implements algorithms that will accomplish this task from road image capture, through detection of lane markings to trajectory planning and steering controls required to traverse the planned trajectory. Each part of this overall algorithm is based on existing algorithms that are discussed in the literature review section. These algorithms are however, implemented or used in novel ways in this project. Then based on the results from running each section of this algorithm recommendations are made regarding the current usefulness of these methods and how each could be expanded upon and improved to be used as a viable solution in the future. Also given the dangers of testing new algorithms on the open highway, this work investigates the design of a model test system that could be used to test algorithms in a safe and compact controlled environment. Some small parts of this model test system are implemented in this work.

Declaration

I declare that I am the sole author of this thesis and that all the work presented in it, unless otherwise referenced, is my own. I also declare that this work has not been submitted, in whole or in part, to any other university or college for any degree or qualification.

Seán Mullery

Seán Mullery
September 2007

Fergal Henry

Fergal Henry (Supervisor)

Acknowledgements.

Sincere thanks to Fergal Henry my supervisor for his guidance and support throughout the project.

Thanks also to Jim Rogers, who formerly supervised the project before moving on from his position at I.T. Sligo.

Thanks to the following members of staff at I.T. Sligo who lent there expertise and knowledge at crucial points during the project: Sean Dalton, Gordan Muir, Liam Winters, Ian Craig, Leo Creedon, Simeon Gillan, Stephen Reid, Frank Carter, Brendan McCormack and John Hession.

Finally thank you to my family for their support throughout the project.

Abbreviations

CMU	Carnegie Mellon University
DARPA	Defence Advanced Research Projects Agency
DMA	Direct Memory Access.
FOE	Field Of Expansion
I^2C	Serial communications standard between integrated circuits
MIT	Massachusetts Institute of Technology
MLR	Multitype lane Marker Recognition
PPI	Parallel Peripheral Interface
PROMETHEUS	PROgram for a European Traffic with the Highest Efficiency and Unprecedented Safety
PVS	Personal Vehicle System
Ralph	Rapidly adapting lateral position handler
RPM	Raised Pavement Markers
RS232	Serial communications standard
RTOS	Real Time Operating System
SCCB	Serial Camera Control Bus
TFTP	Thin File Transfer Protocol
UBM	University Munich (universitat der Bundeswehr Munchen
VaMoRs	Versuchsfahr fur autonome Mobilitat un Rechnersehen
VaMP	Versuchsfahrzeug fur autonome Mobilitat PKW

Table of contents

List of Figures

List of Tables

1 Introduction

Humans' usage of the motor vehicle for transport and freight is ever increasing. It seems that the risks and level of accident rates associated with these traffic systems can only be lessened by increasingly complex systems, which aid the human driver with the task or take some of the task out of the human driver's control. However, any semi or fully autonomous vehicle must conform to the type of roads currently in existence. Those roads are designed to suit a human sensory system, mainly the vision sense. Therefore, it would seem that computer vision type systems would be the largest contributor to autonomous vehicles in the short to medium term future. The basic aim of this study is to investigate the use of vision systems for use in the robotic control of automated vehicles. This is split into several main sections. Firstly, a literature review has been carried out to explore the progression of the use of vision systems in automotive control applications over the last few decades. Secondly, since vision systems have widely varying applications in this area, one particular application, Lane Detection and Following, is considered in depth as regards development of such a system, from image acquisition to control decisions. This entails the following:

- Capture of a roadway image.
- Detection and categorisation of lane markings within the image.
- Determination of a path that the vehicle should follow in order to traverse the lane captured in the image.
- Controls needed to enable the vehicle to follow this path, i.e. steering angle and forward motion required.

Thirdly, given the expense and impracticality of testing real vehicles on real life lanes/roadways, investigation of a possible physical model system, which could be used to test and compare future algorithms in a safe and controlled environment is investigated.

The main body of work in this project concerns the implementation and testing of lane detection algorithms. The literature review looks at many different algorithms developed in the past by other researchers. These algorithms may be complete systems for Lane Detection or may concentrate on some small subsection of the Lane Detection and Following problem. This work then picks individual subsection algorithms and implements them. These individual implementations are then integrated to achieve an overall algorithm for Lane Detection and Following.

As stated in the bullet points above this work must first implement a system, which captures the roadway image. This is achieved by using a camera module connected to a DSP board and the development of software to capture the image data from the camera and store it within the DSP board memory. This data is stored in such a way that it can be analysed by the algorithms that form the next steps of the Lane Detection and Following algorithm. The next sections of the Lane Detection and Following problem are implemented on a PC therefore, necessitating the retrieval of the image data from the DSP board memory and transferring it to the PC on which the rest of the algorithms will be implemented. This means firstly that the image must be in an appropriate file format (Windows Bitmap format is chosen) and that the software implemented on the DSP board must assemble the image data captured from the camera into this format. A Linux work station can be used to communicate with the DSP board and via this the image file can be retrieved. From here it is then transferred to a Windows PC where the rest of the algorithm is implemented.

The next algorithm, which must be carried out, is the detection and categorisation of lane markings within the image. This is carried out on the bitmap image of the roadway and is implemented using the C Programming language. The algorithms are based on some of the algorithms researched in the Literature Review. However, the papers describing these algorithms do not describe the in depth low level implementation of these algorithms and so much of the implementation in this work is novel. Also the algorithm on which the lane detection and categorisation is based works on images as they would be captured from a camera. In this work the image first undergoes an Inverse Perspective Mapping to change the image into a resemblance of a "Birds Eye" view of the road ahead and so must implement a novel algorithm for Lane Detection and Categorisation for this new type of image.

The next algorithm to be carried out is the determination of a curve that follows the lane. This will be implemented in the C programming language. Again the methods for carrying out this are based on algorithms researched in the Literature Review. However, the exact implementation is a novel approach. The algorithm entails grouping lane markings into lanes and determining from these a set of target points up the centre of the lane, which it would be desirable for the vehicle to hit. Next a curve is generated to either hit or come close to hitting each of these points. It is this curve that the vehicle is expected to follow. Once the curve mentioned above is known, the next algorithm developed is one which will inform the vehicle's control system what movements to make in order for a vehicle of automobile type to traverse the curve. This algorithm is carried out in the Java programming language in order to test a number of different curves and vehicle attributes. The Java programming language has particularly good graphics capability and this enables the results of the algorithm tests to be viewed visually.

The investigation of a physical model system centres on the need for a safe, cost-effective system that can be fitted into a relatively small amount of space. This system is not actually implemented in this work though small parts of it are tested. A simple algorithm is developed in which the wheels of a model vehicle react to a straight black piece of cardboard being moved in front of a white background. Also sections of a model roadway are built in order to test the Lane Detection and Following algorithms. An image is captured of pieces of model roadway by the camera module and this image is then passed up through the system to the PC where the rest of the algorithms operate on the image.

The key achievements of this research follow:

Images are successfully captured by the camera module and transferred into the memory of the DSP board.

The DSP board is successfully configured to communicate with a Host Linux workstation. This allows for the DSP board to be controlled and files to be sent and received from the Linux workstation to the DSP board.

Software is created for the DSP board that successfully analyses the image data to determine the location of a black piece of cardboard against a white background. This software also successfully sends appropriate control signals to a servo motor attached to the wheels of the model vehicle, which resulted in the wheels of the vehicle turning to follow the movements of the piece of black cardboard.

An Inverse Perspective Mapping algorithm is successfully implemented resulting in the original image of the road being transformed into a birds eye view of the road. Some important issues with the results of this algorithm are discovered and are included in the section on results. These are also discussed in the discussion section.

An algorithm to detect and categorise Lane Markers is successfully implemented and tested on sections of model roadway. An algorithm to group Lane Markers into lanes is also successfully implemented and tested on sections of model roadway. While the results of these tests are positive the test set is quite small and some suggestions are made in the discussion and conclusion sections regarding improvements that could be made to the algorithm to ensure better robustness for a wider range of images resulting from different roadways.

An algorithm to determine a set of target points up the centre of the lane is successfully implemented. Due to the results of the previous algorithm the number of these points is dependent on the number of Lane Markers, which are successfully detected, to make up each lane. The results from this are used to determine the possible speed of vehicle versus the number of images processed per second. This could later be used to determine what speed a vehicle would be permitted to do given the processing time required for the algorithm to run.

An algorithm to generate a curve, which hit or came close to hitting each of the target points generated in the previous algorithm, was successfully implemented. These algorithms took two forms. In one form, the Java programming language was used and arbitrary points were chosen. Curves were generated to hit or come close to hitting these target points. Two types of curve were tested here, namely a cubic curve and a b-spline curve. The second form of this algorithm used C-programming and involved the output bitmap from the previous algorithm where target points were determined by the Lane Detection algorithm. The curves in this case, once calculated, were superimposed on the bit map image. This then showed the birds eye view of the roadway with the vehicle's calculated trajectory along this roadway.

Finally an algorithm to calculate the steering and forward motion movements of the vehicle in order for the vehicle to maintain position along the trajectory calculated was successfully implemented. This algorithm was demonstrated using the Java programming language and using the previous curve generator algorithm as its input, as well as specific variables related to the vehicle such as wheelbase. This enabled the tester to visually witness the forward and steering movements required for the vehicle to follow the curve.

The layout of this Thesis follows:

Chapter 1 Introduction: is an introduction to the research work undertaken in this project.

Chapter 2 Literature Review: is a review of literature related to this subject. In this chapter various algorithms developed by others in this field are introduced and briefly explained. Some of these algorithms are in fact used in the implementation of this work.

Chapter 3 Project Implementation and Development: gives an in depth explanation of how the project is executed. This includes development & implementation of algorithms as well as any hardware systems, which had to be designed or configured in order to execute the project.

Chapter 4 Results: lays out the results obtained when executing the project. This is split into many sections, each applicable to a different part of project i.e. different algorithms. In appropriate places in this chapter, key/significant results are noted for later discussion.

Chapter 5 Discussion: is an in-depth discussion of the results obtained from executing the project's algorithms. Advantages and disadvantages of the individual algorithms are noted here.

Chapter 6 Conclusions: draws conclusions based on the work carried out and also makes recommendations about future work in this area, both towards improving the algorithms implemented in this work and towards new algorithms that could be carried out in this area.

Chapter 7 Bibliography: outlines the sources used in researching this project.

Chapter 8 is the Appendices where various ancillary resources, which the reader may find useful, are stored.

An important Note to the reader follows:
Many different algorithms are investigated and incorporated into the development of this project. Due to the fact that each of these algorithms was developed independently of the other algorithms use the same variable designations to represent different variables. The reader should be mindful that particular variables referred to in any given section might only be applicable to that section of the document.

2 Literature Review

2.1 Overview

Of all the systems with which people have to deal every day, road traffic systems are the most complex and the most dangerous. World-wide, an estimated 1.2 million people are killed in road crashes each year and as many as 50 million are injured. Projections indicate that these figures will increase by about 65% over the next 20 years unless there is new commitment to prevention [1].

There is also a summary of a number of studies contained in [1] which suggest that driver fatigue caused by many factors including monotonous roads contributes significantly to high accident rates. With human error being such a prevalent cause of road traffic accidents, it seems the only way to reduce accidents is to use technology to increase the human driver's awareness of hazards or to remove control from the human driver all together.

The current state of the art in autonomous vehicles at the time of writing of this Thesis lies with those institutes and organisations taking part in the DARPA Urban Challenge, which is due to run in November 2007. This challenge asks an autonomous vehicle to negotiate city streets, merge with traffic, and avoid collisions with other vehicles. Operating at this level requires large financial budgets and a large array of sensors. For example the team from MIT have a vehicle which contains three rooftop lasers which scan the road 75 times per second and can detect objects at 50 yards to within a quarter of an inch. It also has three lasers on the front bumper to detect stationary obstacles. Above the windshield and on the hood are three radar sensors that can spot objects up to half a mile away, and video cameras in many locations around the vehicle to help pick out painted road lines and identify other vehicles by colour. Analysing and making decisions based on information from this amount of sensors needs a very large amount of computing power. In the case of the MIT vehicle it contains a supercomputer containing 40 separate one-gigahertz processors. This computer requires 4 kilowatts of power [2].

This thesis is only concerned with those systems that rely on computer vision through video camera sensors. It explores some of the systems that have already been implemented in the move towards autonomous vehicles and vehicles, which aid the driver in the advance recognition of hazards. The main focus is on systems, which keep the car in the appropriate lane autonomously or warn the human driver of lane departure.

2.2 Time line

1970's:

An Intelligent Vehicle System was developed in the mid 1970's in Japan employing machine vision for obstacle detection [3].

1986:

The PROMETHEUS (PROgraM for a European Traffic with the Highest Efficiency and Unprecedented Safety) project was initiated in 1986 [4].

1987:

The UBM (University Munich) test vehicle for autonomous mobility and computer vision VaMoRs demonstrated the capability of fully autonomous longitudinal and lateral vehicle guidance by computer vision on a free stretch of Autobahn over more than 20km at speeds up to 96 km/h (engine power limited). This result led to computer vision guidance being included in Prometheus and inductive lateral guidance by buried wires being dropped [5].

Late 80's: In Japan Mechanical Social Systems Foundation supporting Nissan and Fujitsu started a project called PVS 'Personal Vehicle System' [3].

1994:

The primary phase of PROMETHEUS concluded and was reported on in Paris in October [4]. The Final Demonstration of PROMETHEUS took place on the A1 near Paris in October 1994. Thousands of kilometres have been driven in normal three-lane traffic including convoy driving and transition into this mode from free-lane driving at speeds up to 130km/h [5].

1995

Robotics Institute of Carnegie Mellon University (CMU) had its test vehicle NavLab_5 run from East Coast Washington DC to the West Coast USA Los Angeles. It was equipped with a simple vision system for recognition of the horizontal curvature of the road and the lateral position in the lane. Longitudinal control was done by a human driver while lateral control was performed fully autonomously. 98% of the total distance of 2,850 miles could be driven without human intervention in steer angle control [5][10].

VaMP of UBM demonstrated a fully autonomous long distance drive (both lateral and longitudinal) on the Autobahn over more than 1600 km from Munich to Odense, Denmark. About 95% of the distance could be driven without intervention of the safety driver [5].

1997:
An Italian group also did a long distance test drive called 'Mille Miglia in Automatico'.

1999:
Chris Kreucher and Sridhar Lakshmanan [8] implemented a frequency domain based lane detection algorithm. It was applied to a varied set of images. The images were obtained under a variety of lighting and environmental conditions, shadowing and lane occlusion(s), solid and dashed lines, etc. When compared to an intensity gradient type algorithm it seemed to indicate that it had some advantages. It seemed especially good at not being distracted by strong non-lane edges in far range.

2000:
Juan Pable González and Umit Ozguner [7] implemented a Histogram-based segmentation algorithm on an Intel PIII-450 MHz with a Matrox Meteor acquisition board. It took 20-25ms to process each frame. The system was tested for two scenarios: initially in a road simulator and then in more than 60 minutes of recorded scenes of highways and major roads.

2002:
Otsuka, Y.; Muramatsu, S.; Takenaga, H.; Kobayashi, Y.; Monj, T., [9] implemented a lane recognition algorithm which was able to find lane markers regardless of their types such as white lines, raised pavement markers and botts dots (Cat's eyes). The system was tested with botts dots that were installed on freeways in North America. The algorithm was tested in experiments at three times (noon, evening and night). All were above 96% recognition rate

with Noon being 98.2%. Causes of recognition errors were mainly due to the change in lane width on freeway-junctions. The changing of lanes by the host car and the shadow of preceding cars also caused recognition errors. The algorithms were evaluated on a 500 MHz Pentium PC and image sizes were 640x480 pixels. Average processing time varied between 59ms and 95 ms.

2.3 Types of Control

While there have been a number of experiments done with full automatic control [4][3][5][10] none seem robust enough yet to use in a general purpose vehicle on the road. It is also likely that even when autonomous vehicles do become robust enough that they exceed the safety of the human driver, that it may take much longer to convince the public of this. With this in mind, many of the systems in design and use, are warning systems and driver aids.

2.4 Types of Detection

The use of vision systems in vehicles has split into many sections. The most useful so far is the detection of lane markings. Given the various types of lane markers, many different algorithms have been developed. Indeed algorithms will continue to be developed, with the end aim being a system which can detect all types of lane markings and reliably inform the control system of the position of the vehicle in the lane and also of the change in the lane in the forward direction e.g. bends etc.

As well as lane detection, detection of other vehicles has been investigated with both single and multiple camera solutions. This falls into the bracket of collision avoidance and adaptive cruise control (ACC). Radar is often used in conjunction with vision for this sort of application.

Other obstacles for recognition have also been looked at, e.g. pedestrians.

Reading of traffic signs has also been investigated. However, no cost effective system has so far made it to market and it is likely to be an optional extra when it does.

2.5 Algorithms

2.5.1 Methods of Lane Detection:

Given the problem of the amount of processing that is required to process images one of the greatest challenges has been to reliably detect a lane marking without doing exhaustive processing on the image.

2.5.1.1 Edge based methods

One simple algorithm, which is presented in [6], is described below.

The image of the road scene is first obtained via a vehicle-mounted camera. The image is then searched along a pre-determined number of rows placed vertically to the heading of the vehicle. Each row is split into a right and left section. The right side is to search for the right hand road lane boundary and correspondingly the left side of the row is to search for the left road lane boundary. The crossing point of the search row and the lane boundary becomes a sampled point which is then recorded and used along with the other sampled points to determine a correct course for the vehicle. In [6] it is assumed that the road is flat allowing an easy mapping correspondence of the physical plane to the image plane. To reduce detection time even further the entire search rows are not searched; rather a sub-zone of each search row is searched. The appropriate positioning of each search row can be predicted with the aid of the road edge detection in the previous control period (image). If however, there is no previous road edge detection, such as is the case when the vehicle is at initial start-up, then the search zone is centred on the midpoint between the centre of the image horizontally and the edge of the image on each side, i.e. the centre of the left search zone should be ¼ the image width in from the left hand side of the image and the centre of the right search zone should be ¼ the image width in from the right hand side of the image. This is shown in Figure 1.

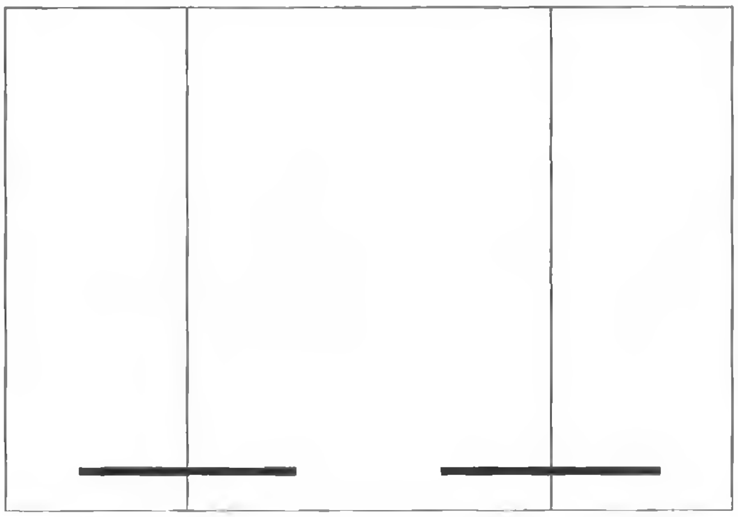


Figure 1. Search zones at initial start-up

Once the search zones are determined the lane boundary is found as discussed below.
The following second differential and smoothing filter is applied to the search zone. The coefficients of this filter are.

-1	-1	-1	2	2	2	-1	-1	-1

Applying this filter leads to the edges at the road boundaries between lane markings and roads being emphasised. After applying this operator the gradients at the zero crossing points can be found, as can the maximum and minimum points. The two zero crossing points should correspond to the edges of a lane. If the distance between these two points is appropriate and its absolute value is large enough, then the two points are regarded as a road lane edge and a sample point (midpoint of the segment connecting these points) is obtained. If the above condition is not satisfied then the sample point is not obtained.
Once all the sample points have been obtained the next job is to match up left and right sample points with each other. Each left point and right point on a given search row are matched and a point midway between the two is calculated. This results in a set of points going down the centre of the lane. By fitting a cubic curve to these points down the centre of the lane then a final path that the vehicle should take is obtained.

2.5.1.2 Histogram based methods

A method based on Histogram segmentation is given in [7]. A Description based on [7] follows.

It is assumed that the only object present in the lower scan lines of the image is the road. For most acceptable road conditions, the road variations in such a small section are small enough to assume that the histogram of the first bands is going to be unimodal and in some cases even close to Gaussian. At this point the mean value of the grey level distribution of the road, as well as the maximum and minimum value of such a distribution are calculated. Then values above the maximum are assumed to be lane markers and values below the minimum are assumed to be objects (e.g. cars etc. based on the fact that the obstacles, especially vehicles, produce shadows which are darker than the road).

As the algorithm progresses up the image, objects other than the road will appear and this will change the distribution of the histogram. Bands are processed until the vanishing point is passed, which is calculated from previous frames or from an initial guess e.g. half the screen. At this point the only interest is in finding the lane markers. So each band is thresholded with the maximum value of the road histogram. The resulting images are merged to get a black and white image containing most of the lane markers and some other objects in the scene. So the original image is segmented into three regions, namely Road, Lane-Marker candidates and Obstacle candidates.

The number of floating point operations executed by the algorithm up to this point is approximately 6 FLOPS/pixel, where most edge detectors would require between 30 and 300 FLOPS/pixel to obtain a similar result.

The most relevant features of the lane candidates need to be extracted at this stage and analysed at a higher level. Several characteristics believed to be relevant to filter out non-lane marker objects are calculated.

A Least Squares fit is applied to a straight line of each object in the image (lane candidates). From here, the estimated inclination, intersect and the correlation coefficient are calculated. These are calculated from the following:

- The Average angle formed by each point in the object is calculated as is
- The centroid of the object.
- The size of the object.

- The maximum width of the object is also calculated, as is the co-ordinate at which this maximum is located.

At this point the classification of the objects must take place to determine if an object is a lane marker or not. This process is divided in two.

2.5.1.2.1 Unary Classification.

First a confidence measurement is defined for each object. High positive measurement indicates a likely lane marker. Low positive measurement indicates the possibility of a lane marker only if other evidence corroborates this e.g. position of other high measurement lane markers or some other prior knowledge. Negative value measurement indicates non-lane marker objects and these are rejected at this stage. This classification is done in correspondence with the features extracted previously and some derived features such as:

- Maximum angle spanned by the object with respect to the co-ordinate of the vanishing point and
- Weighted size of the object, which is equivalent to the size, weighted to the inverse of the distance in y (vertical axis) to the vanishing point.

2.5.1.2.2 Relational Classification

Here relational information for groups of objects is analysed. In order to do this "lane objects" are defined i.e. groups of objects that are likely to be on the same lane marker structure (continuous or dashed line or other combination). Lane objects have the following properties:

- Base angle: the angle at which the first object on the "lane object" was found.
- Last angle: the angle at which the last object was found.
- Size: while not specified in [7], this is assumed to be size in pixels.
- Weighted size: the sum of the sizes, weighted by a function of the status flag.

At each point whether or not an object belongs to one "lane object" based on confidence measurement is evaluated. A large value of confidence indicates a highly likely lane marker. Therefore, either they are grouped together (orientations similar) or a new "lane object" is started based on the current object. A small value of confidence indicates that an object could be part or not of a lane depending on its position.

2.5.1.3 Frequency Domain approach

A frequency domain approach for detecting lane markers in images acquired from a forward-looking vehicle mounted camera is given in [8]. The method is based on a novel set of frequency domain features that capture relevant information concerning the strength and orientation of spatial edges. The frequency domain features are combined with a deformable template that is determined previously, in order to detect the lane markers of interest. The algorithm seems to detect lane markers remarkably well for a very large and varied collection of roadway images.

The method used in [8] can be outlined as follows:

An image is broken up into 8x8 pixel blocks. For each block, a frequency-domain-based feature vector is computed. This feature vector reflects the amount of "diagonally dominant edge energy" that is contained in that 8x8 block. The block feature vectors are then used in combination with a deformable template shape model of the desired lane markers. This combination is accomplished in a Bayesian setting, where the deformable template model plays the role of a prior probability and the feature vectors are used to compute a likelihood probability. The lane detection problem is reduced to finding the global maximum of a four-dimensional posterior probability density function and an exhaustive search is employed to find the global maximum.

2.5.1.3.1 Frequency Domain Features

An examination of roadway scenes obtained from a forward-looking vehicle-mounted camera easily reveals that lane markers tend to have "Diagonally dominant" orientations in the image plane due to the perspective transformation inherent in the ground plane imaging process, whereas the extraneous edges have no such preferred orientations. [8] finds the frequency domain to be a convenient method to discriminate between edges that are diagonally dominant and those that are randomly oriented.

The salient points of the method are as follows:

The image is divided into 8x8 blocks and the DCT of these is taken

There are 12 of the DCT basis functions that are most important. These are most important, as they tend to represent the diagonally dominant edges of lanes. For each original image the corresponding feature image is obtained by summing the squares of its 12 special DCT decompositions.

2.5.1.3.2 Deformable Template.

The algorithm presented in [8] uses a global shape model to predict the manner in which lane markers appear in images. As is commonly done, it is also assumed that lane markers are circular arcs on a flat ground plane.

2.5.1.4 Dealing with multiple types of lane markers.

In [9] the focus is on developing a system for recognition of multiple types of Lane Markers using local Edge detection. [9] utilises the characteristic that lane markers converge to the focus of expansion and the lane markers have edge points towards the focus of expansion. The algorithm uses only the edge points whose direction is towards the focus of expansion. Edge direction is computed by only near-neighbour elements rather than continuous line segments in order to spot lane markers even if their shapes are not continuous lines. In case of Raised Pavement Markers (RPMs), experimental results show that the average processing time is under 100ms and recognition rate is over 96%. The MLR (Multitype Lane marker Recognition) algorithm is robust for various noises such as from preceding vehicles, shadows of trees, road signs, etc.

2.5.1.4.1 Outline of approach.

The edge-based algorithm is used because it is robust to changes in brightness. Therefore, it detects the edge points of lane boundaries between lane markers and the road surface and analyses the distribution of edge points in order to calculate the directions of lane markers.

2.5.1.4.2 Two problems to be solved for RPMs

Edge Point Detection: - RPMs are thin in comparison with white lines so they give a smaller contrast when compared with white lines

Lane Marker Recognition: - Edge points of RPMs are discontinuous because the RPMs are small circles or squares. It is difficult to analyse this because of discontinuity.

2.5.1.4.3 Edge Point Detection.

One goal for the MLR algorithm is that it can detect lane markers even if there is only a little difference in brightness of lane boundaries. To prevent lost lane markers, the threshold for detecting edge points should be lowered.

2.5.1.4.4 Lane Marker Recognition.

To recognise lane markers, the distribution of edge points is analysed.

2.5.1.4.5 Implementation

2.5.1.4.5.1 Edge Detection

The algorithm extracts edges from the image and each edge point contains position and angle. To calculate the angle of each edge point, the zero-crossing method is used. The zero crossing method detects only the minimum necessary number of edge points to recognise lane markers as the method detects only the area of the maximum difference in brightness. Zero crossing can calculate information on edge points with sub-pixel accuracy. This is advantageous when dealing with small round shapes like RPMs.

2.5.1.4.5.2 Noise Elimination.

Because edge points are detected under low threshold conditions, most edge points are noise. The feature that lane markers are in the direction of the FOE (Field of Expansion) is used in order to eliminate noise. In this process the edge points of the preceding vehicle are eliminated because most edge points of the preceding vehicle are not towards the FOE, because the direction of the edge points are on the horizon or vertical. The edge points of other road noises are eliminated for the same reason. However, this assumption cannot be applied to a curved road, because the directions of the lane markers change according to their positions along the curve. Hence all distant edge points, which are close to the FOE, are eliminated even if their angles correspond to the direction of the FOE. The lane marker can be regarded as a straight line in the area near the host vehicle, so it is possible to detect the lane markers regardless of road curvature.

2.5.1.4.5.3 Histogram Analysis.

When all the edges in the scene are captured a Histogram analysis of the angles of the edges is performed. Peaks should occur at the angle of the road edges heading towards the FOE. Furthermore the histogram exhibits a significant double peak structure, where the two peaks are at about the left and right lane markers. Then lane width can also be referred to in order to exclude noise peaks, such as neighbouring lane markers, preceding cars, slip-scars (skid marks) etc. Furthermore a weight is given to each edge point

2.5.1.4.5.4 Lane Boundaries Detection.

If two lane markers are found, the algorithm implements a Hough Transform to detect the lane boundaries by using edge points which have the same angle.

2.5.1.4.5.5 FOE update.

The FOE position is updated as the intersection point of the two lane boundaries. The FOE position is used for noise elimination.

2.5.1.5 Ralph Vision System

Ralph stands for (Rapidly adapting lateral position handler) and is a vision system developed jointly by Carnegie Mellon University and AssistWare Technology Inc. Ralph decomposes vehicle steering into three steps: sampling the image, determining the road curvature and assessing the lateral offset of the vehicle relative to the lane centre [10].

2.5.1.5.1 Sampling the image.

Many parts of the image taken by the camera mounted next to the rear-view mirror are not relevant to the driving task (e.g. parts depicting the sky or showing the vehicle dashboard). Ralph eliminates these parts and only processes the portions of the scene inside a designated trapezoid (the trapezoid suits the perspective effect of the road ahead). Although the upper and lower boundaries of the trapezoid vary with vehicle velocity (moving further ahead of the vehicle, toward the top of the image, as the vehicle increases speed) they are typically 20 to 70 meters ahead of the vehicle, respectively. The second and perhaps more important aspect of the trapezoid's shape is its horizontal extent. It is configured so that its width on the ground plane is identical at each row of the image. The horizontal distance that each row of the trapezoid encompasses, is approximately 7 meters, about twice the width of a typical lane. The trapezoid is selectively sampled according to the strategy that changes it from a trapezoidal perspective image to a rectangular birds-eye view. This sampling process creates a low-resolution (30x32 pixels) image where important features such as lane markings (which converge toward the top of the original image) appear parallel (birds eye view). This image re-sampling is a simple geometric transformation and requires no explicit feature detection.

2.5.1.5.2 Curvature calculation

Transforming the image of the road features into a bird's eye view is crucial to the curvature calculation step of Ralph processing. To determine the curvature of the road ahead, Ralph hypothesises a possible curvature, subtracts it from the parallelised low-resolution image and tests to see how well the hypothesised curvature has "straightened" the lane markings in the image. The hypothesis with the closest similarity to the actual image will cause the straightest result.

How straight is the result? This is ascertained by vertically summing the columns of the resulting transformed image to create a scan-line intensity profile. When the visible image features are correctly straightened, sharp discontinuities between adjacent columns occur in the image. In contrast, when the hypothesised curvature has shifted the image features too much or too little, there are smooth transitions between adjacent columns of the scan-line intensity profile. By summing the maximum absolute differences between intensities of adjacent columns in the scan-line intensity profile, Ralph can quantify this property to determine the curvature hypothesis that best straightens the image features.

An important attribute of this technique for determining road curvature is that it is entirely independent of the particular features present in the image. As long as visible features run parallel to the road, this technique exploits them to determine road curvature. Those features need not be at any particular position relative to the road and they need not have distinct boundaries.

2.5.1.5.3 Lateral offset calculation

Next, Ralph determines the vehicle's lateral position relative to the lane centre. It uses a template matching approach on the scan-line intensity profile generated in the curvature estimation step. The scan-line intensity profile is a one-dimensional representation of the road's appearance as seen from the vehicle's current lateral position. By comparing this current appearance with the appearance of a template created when the vehicle was centred in the lane, Ralph can estimate the vehicle's current lateral offset.

2.5.1.6 Multiple Cues method

A cue in this case is any item in a road scene that could be used to detect the lane ahead. A common characteristic of many lane detection and following systems is that they rely on only one or two cues for lane detection, that are used regardless of how well they are performing. They do not make any attempt to track the road as the conditions change from highly structured highways to semi-structured lanes to unstructured off-road conditions.

In [11] a method which uses multiple cues is considered. It is based on a Distillation Algorithm that attempts to dynamically allocate computational resources over a suite of cues to robustly track the road in a variety of situations. The system also uses particle filtering, which is a search method that represents the continuous posterior density with a set of discrete particles, or hypothesis. These particles represent the target location and are moved to positions of high probability to concentrate computational power in those areas of interest.

2.5.1.6.1 Distillation Algorithm.

This algorithm is based on a suite of cues, which are calculated from image and state information and combined to provide evidence strengthening or attenuating the belief in each hypothesis of the particle filter. Each cue's usage is evaluated over time and those cues that are performing best are distilled to a select set that can then be given the largest share of the processing time. The other cues that are not performing as well are still processed but are not given priority. These cues are not used for decision making as they are not processed fast enough but their results are monitored to see what their contribution would be to the overall tracking. If it is found that one of these cues over time has become more useful than one of those that is being processed faster, then that cue will have its priority upped so that it is given more processing power overall and is included in the tracking. In this way, as road structures change, the type of tracking used changes to a more appropriate type.

2.5.1.6.1.1 Types of cues

Each cue is developed to work independently from the others and is customised to perform better in different situations. Individually they would perform unsatisfactorily, but combined they produce a robust solution to lane tracking.

Two different classes of cues are used in the lane tracker, namely Image based cues and State based cues.

Image based cues:

Lane Marker Cue: suited to detecting roads that have lane markings.

Road Edge Cue: suited to detecting roads with lane markings or defined edges.

Road Colour Cue: suited to any roads that have a different colour than their surroundings.

Non Road Colour Cue: suited to evaluating non-road regions in the road colour probability map.

State based cues:

Road Width Cue: suitable in multi-lane roads where it is possible for the other cues to detect two or more lanes as one.

Elastic Lane Cue: is used to move objects towards the lane that the vehicle is in. The need for this cue arose when it was discovered that the lane tracking system often switched between lanes when driving on a multilane highway. This cue was introduced to favour particles that describe lanes that the vehicle was in.

2.5.1.6.2 Experimental results.

This method has proved very effective in its proficiency for target detection, the particle filter moves seamlessly from detection to tracking without any additional computation required for the detection phase. The lane tracker was tested in several different scenarios including highway driving with light traffic, outer city driving with high curvature roads and inner city driving with moderate levels of traffic. Cue fusion was found to dramatically increase the robustness of the algorithm due to the variety of conditions the cues were suited to. The initial set of cues were limited to the image based cues (Lane Marker, Road Edge, Road Colour and Non-road Colour Cues), which contained no prior information except for a transformation between a particle and its road model in the image space. Using the initial cues the particle filter often converged to lane segments that the vehicle was not in or to the whole road instead of a single lane. This was due to the lane markings and edges of the road having stronger signals in the observations than the lane markings separating the lanes. Adding the two heuristic state based cues (Road Width Cue and Elastic Lane Cue) to strengthen hypotheses that contain the vehicle and have a road width close to the average was found to be a satisfactory solution to this problem.

2.5.1.7 Lane recognition by detecting Reflection Posts.

On country roads in some countries (Germany in the case of [12]) reflection posts occur as additional features, indicating the road course due to their close proximity to the road edge and the ability to see them from a large distance. Since human drivers would use these as an extra indicator for lane recognition it is advisable to also include them in vision/robotic based systems to improve robustness of systems on country roads.

Reflection posts have accurately defined dimensions and appearances. The whole of the reflection post, with its white and black faces is visible in daylight. However, only the small reflectors are visible at night. In [12] the authors concentrate on the detection of reflection posts in daylight.

Three different computer vision methods are investigated as to their capability to locate reflection posts in grey-scale images. These are as follows:

A correlation based method using a matched filter.

An edge based method.

A method that searches for typical patterns of reflection posts in horizontally and vertically integrated edge images.

2.5.1.7.1 Correlation Method

The first method (correlation) is a matched filter method, which correlates the captured image with a matrix, which represents the white and black regions of a reflection post. This method is based on the simple structure of the reflection posts. 2D correlation can be very computationally intensive so to counter this the matrix is separated into a vertical and horizontal search to reduce the computational complexity. A local peak in the result of this operation represents a potential reflection post.

The advantages of this algorithm are that it is relatively fast in computation terms and is suitable for detection of partly occluded objects. Disadvantages of this algorithm are that light spots in the camera image also result in local peaks of the filter results and the filter results blur around the exact position of the reflection posts resulting in disturbances in the tracking process. Also because the size of objects in the image must be known in advance for this method to work, it is of limited use. However, it is suggested that an iterative application of the matched filter for searching diverse sizes of reflection posts is one solution to make the method more usable.

2.5.1.7.2 Edge Extraction Method.

An edge extraction algorithm can also be applied to the reflection posts problem as the bright reflection posts can be separated from the typically dark background. The first step in this detection algorithm is to generate two lists of vectors, which represent the horizontal upward and downward edges in the image. In the second step, groups of edges, that match the characteristic parallel edges of reflection posts, are combined. This method allows the calculation of the size and position of the detected reflection post in the image precisely. By combining the vectors it is possible to detect reflection posts standing at any angle beside the road. However, unlike the correlation algorithm above the recognition of partly occluded reflection posts is difficult.

2.5.1.7.3 Vertical and Horizontal Integration Method

The third method is based on vertical and horizontal integration of image regions. In the case of an ideal vertical reflection post the vertical integration of the edge image would result in perfect straight-line peaks. However, in a situation where the reflection post is slightly skewed the peaks would be blurred but still visible (see Figure 2). The reflection post can be detected and horizontally located by searching the characteristic group of peaks in the integral. The horizontal integral is then applied to a limited area of the image to determine the vertical position of the reflection post in the image. Similar to reflection posts that are not in correct upright position, partial occlusions decrease the peaks in the integrals.

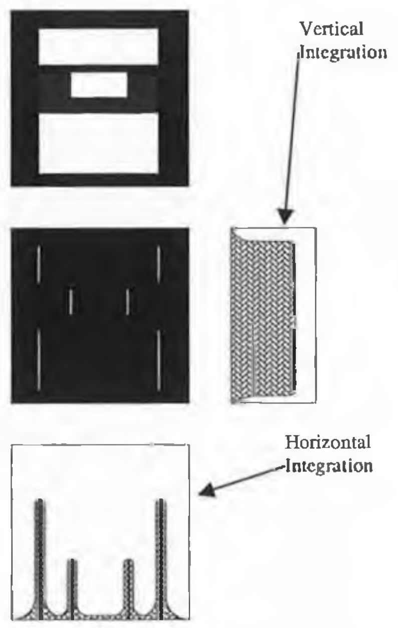


Figure 2. Vertical and Horizontal integration

With this method it is difficult to detect reflection posts at larger distances. The method is however, suitable for tracking the objects in the image sequences because these features can be found again easily.

2.5.1.8 Lane recognition using Hough transform.

In [13] a system using a Hough transform to fit extracted numerical white line data to a straight line for the lane boundary is used. The algorithm for the lane recognition includes processes for the threshold setting, the white line extraction, the lane recognition and the prediction of the lane boundary when the lane recognition fails.

2.5.1.8.1 Threshold setting.

First, nine scanning lines, with limited length are chosen for each side, left and right and the maximum value of the brightness signal along each scanning line is detected. The length of each scanning line is 80 pixels long and the centre of each line corresponds to the location of the recognised lane for the previous image data (see Figure 3).

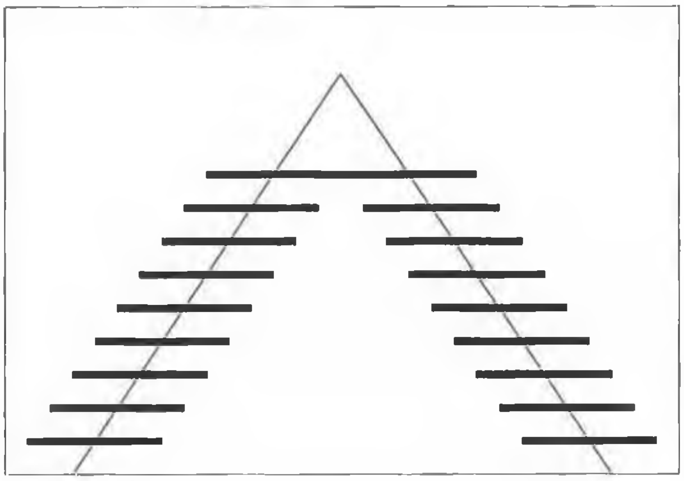


Figure 3. Scan lines

Next, the average value of those nine maximum values is calculated for each side and they are called Pl_{var} for the left and Pr_{var} for the right. The threshold value is calculated as a weighted average of Pl_{var} or Pr_{var} and the average brightness P_{var} over the limited scanning lines. This enables evaluation of the brightness of the white line relative to the average brightness of the road surface.

2.5.1.8.2 Extraction of outline of the white line.

The outline of the white line is determined by using the threshold value. First a parallelogram shaped area, 80 pixels wide and its height corresponding to 150 scanning lines across is cut for each side (see Figure 4).

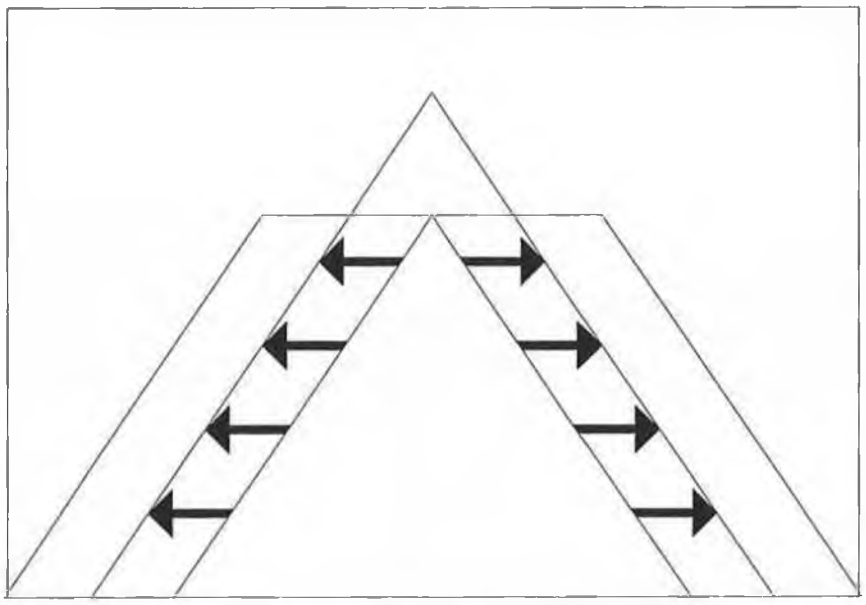


Figure 4. Extracting the outline of white line.

The centre of the width corresponds to the recognised white line in the previous image frame. The brightness data is then compared to the threshold value along each scanning line from the inner end to the outer end. This is continued along each scanning line until the first pixel that is brighter than the threshold is found or the compared pixel reaches the outer end of the limited scanning line and then moves to the next scanning line. Finally the outline of the white line is obtained as the location of 150 pixels for each side, 300 altogether at most.

2.5.1.8.3 Straight-line fitting.

The numerical white line data is fitted to a straight line by using a Hough Transform. The Hough transform is a projection of a point in the x-y plane into a curve in the $\rho - \theta$ plane as given in the following equation.

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta \qquad (1)$$

The characteristic of the transform is that any of the points on the same straight line in the x-y plane are projected into the curves in $\rho - \theta$ plane, which cross each other at the same point, the unique point. This point represents the straight line in the x-y plane and this is used to recognise the white line in the x-y plane. Here the distribution of crossing points are obtained instead of a unique point as the recognised numerical white line data are not exactly on a straight line. Thus the densest point is chosen as the unique point. Then the straight line in the x-y plane can be determined. Disadvantages of this method are that it is time consuming and noise also causes problems. However, these problems can be solved when the calculated area in the $\rho-\theta$ plane is restricted to a minimum as follows. First the origin of the co-ordinate system in the road image is set at the centre of the lower side of the image with the x-axis along the horizontal and the y-axis along the vertical. In general an image of a roadway will have the left line appearing in the left half of the image and the right line appearing in the right side of the image. So the θ domain for the left line can be limited to [90°, 180°] and [0°, 90°] for the right. The Hough transform is applied to only this small area and the new unique point is found quickly. This is possible if the sampling rate is high enough to keep the change of the location of the unique point in the limited rectangular area. The algorithm is effective in reducing the processing time significantly and also avoids the interference of data between the two lines on either side of the lane.

2.5.2 Inverse Perspective Mapping.

In using a camera to sense the roadway, a major issue that must be dealt with is the perspective effect caused when mapping a 3D scene onto a 2D image. In [10][14][15][16][17][18][19] this problem has been given a lot of consideration. Due to the perspective effect, the road markings appear to vary their width and length according to their distance from the camera.

The model roadway shown in Figure 5 is a straight piece of road with each of the road markings being the same size and distance apart. The two sets of lane markers are parallel.

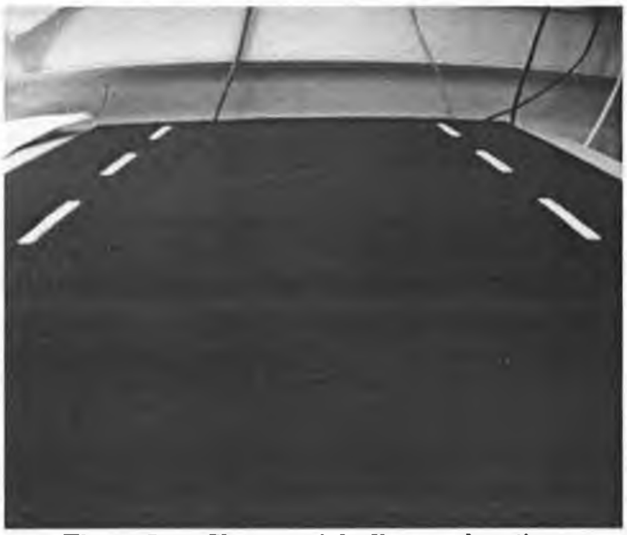


Figure 5. Short straight-line road section.

As can be seen from the sets of road markings that appear in this 2D representation of the roadway, the right and left lane markers do not appear to be in parallel, rather they seem to converge at some vanishing point in the distance. The perspective effect associates different meanings to different image pixels, depending on their position in the image [14]. A Pixel in the lower part of the image represents a much smaller physical area than a pixel in the upper part of the image. This can be seen in Figure 6.

Part (b) shows both a plan and side view of the camera and roadway. The camera in this situation is at a downward angle of 30° and the angular aperture of the camera is 60°. From the side view lines can be seen extending out from the camera at 10° intervals over the 60° of the angular aperture. The distance each of these extends along the x-axis varies. The same can be seen in the plan view. Again lines can be seen extending out from the camera at 10° intervals over the 60° of the angular aperture.

Now looking at the boundary contained between the bottom 10° of the horizontal direction compared with the boundary contained with the top 10°, it can be seen that these two boundaries contain vastly different areas of the 3D scene. However, each of these areas will cover the same amount of pixels in the captured image. Three different sectors are shaded out in the plan view of the 3D scene and can be seen to cover varying areas, but each of these areas are shown again in part (a) of the diagram and are all of the same area.

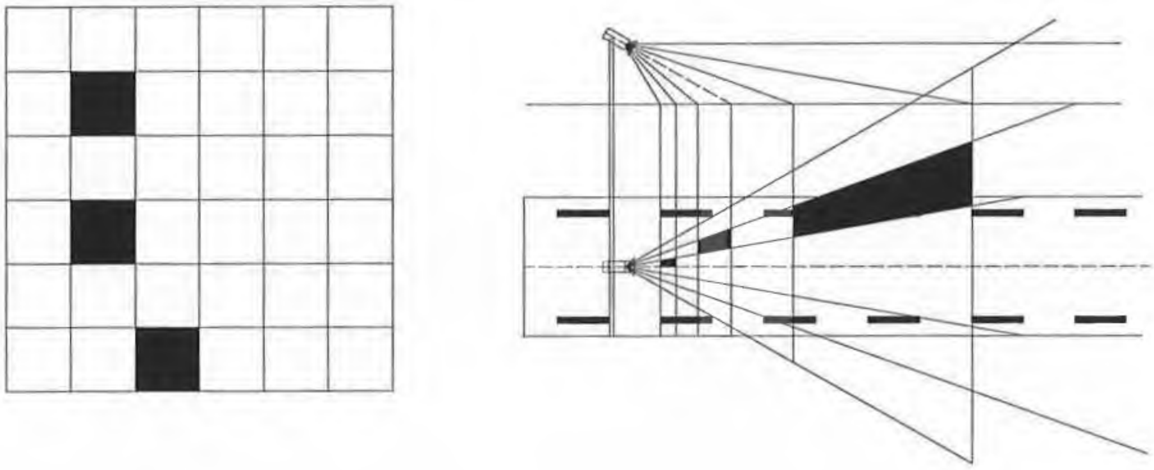


Figure 6. **Perspective effect, (a) shows 36 equal sectors of a captured image, (b) shows the varying areas in the real world scene that go to making up each of those sectors.**

This can make the detection of lane markings and the judgement of distances more complex to ascertain. This inevitably leads to more processing power being required to make decisions based on the image data.

2.5.2.1 Methods of counteracting Perspective effect.

One method for removing the perspective effect is put forward in [10] and is briefly explained in section 2.5.1.5.1. [14][15][16][17][18][19] also suggest a mechanism to remove this perspective effect, with the result being a birds eye view (plan view) of the roadway. Having the birds eye view means that each pixel represents the same portion of the road, allowing homogeneous distribution of the information among all pixels. This should lead to a less complex method of detecting lane markers and determining a target trajectory for the autonomous vehicle to traverse. Less complexity may lead to less processing power being required to implement the algorithm. If this is the case, then it must be considered whether the simpler algorithm added to the processing required to perform the removal of the perspective effect, is in total, less than the processing required by the more complex algorithm for detection of lane markings and trajectory planing based on the original captured image. This is suggested by [14], which states: "The removal of the perspective effect allows to detect road markings through an extremely simple and fast morphological processing that can be efficiently implemented on massively parallel SIMD architectures."

The mathematical formula given by [14] to remove the perspective effect is as follows:

$$u(x, y, 0) = \frac{\tan^{-1}\left[\dfrac{h \sin \gamma(x, y, 0)}{y - d}\right] - \left(\bar{\theta} - \alpha\right)}{\dfrac{2\alpha}{n-1}} \tag{2}$$

$$v(x, y, 0) = \frac{\tan^{-1}\left[\dfrac{y - d}{y - l}\right] - \left(\bar{\gamma} - \alpha\right)}{\dfrac{2\alpha}{n-1}} \tag{3}$$

The important parameters are as follows:

The co-ordinates of the real world scene are (x, y, z), the birds eye (plan view) will show the road as a 2D (x, y) space where it is assumed that z = 0;

The x-axis is parallel to the heading of the vehicle. The y-axis is perpendicular to this.

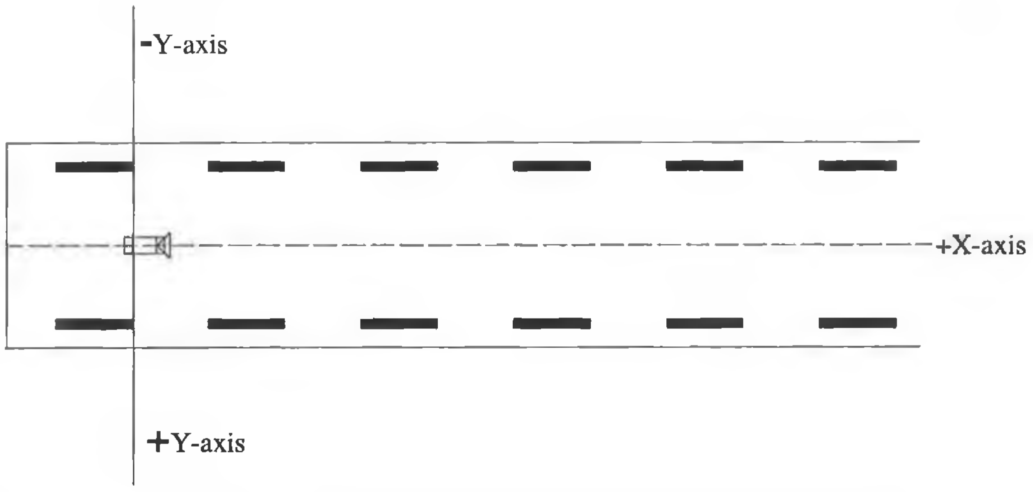


Figure 7. Co-ordinate space of the real world scene in plan view.

u and *v* are the axes of the captured image, which contains the perspective effect that must be undone.

$\bar{\gamma}$ is the angle between the optical axis of the camera and the heading of the vehicle.

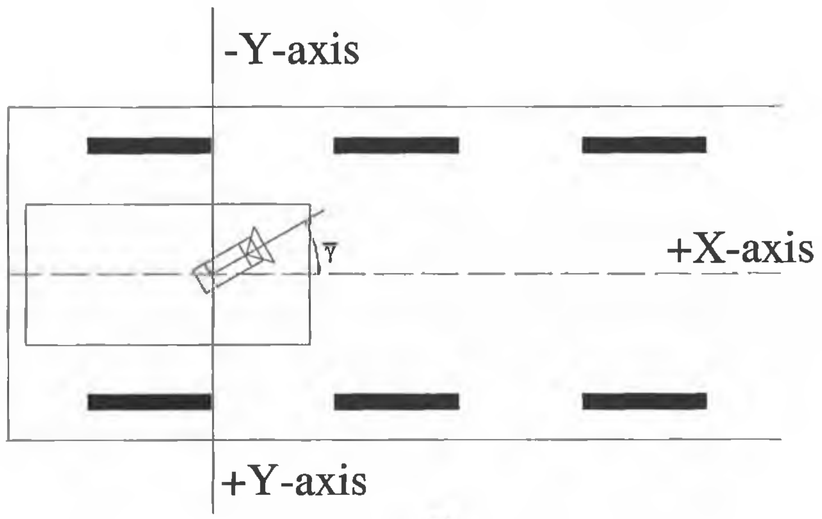


Figure 8. The $\bar{\gamma}$ angle.

$\bar{\theta}$ is the angle between the horizontal, parallel to the (x, y) plane and the optical axis of the camera.

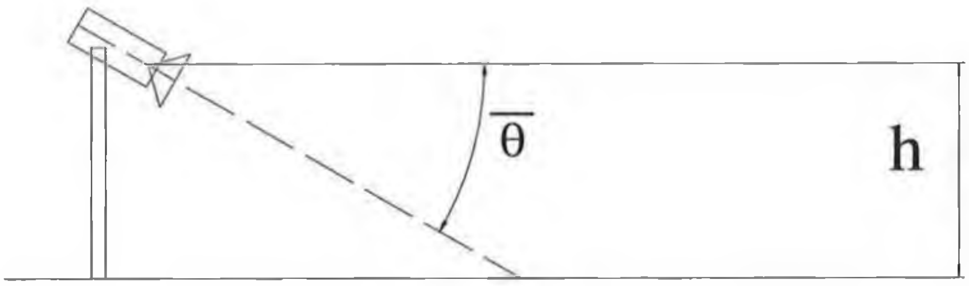


Figure 9. The $\bar{\theta}$ angle and the *h (height) variable.*

The larger this angle, the closer to the vehicle the camera will be able to view. This would be advantageous for close quarter manoeuvring and sharp curves. However, this would lead to less of the road ahead being visible, which may require the vehicle to move at slower speeds as the control system will have much shorter space and therefore, shorter time to react to changing circumstances.

h is the height of the camera on the z-axis, with z=0 being the level of the (x, y) plane on which the road is assumed to be. It is important to note that the units of height are in pixels.

This is best explained with an example. If lane markings are 1 meter in length and the height of the camera is 1.5 meters, then for example if h in the algorithm is set to 30 pixels each lane marking should be 20 pixels long. Changing the *h* value to 60 would cause the lane markings to appear to be 40 pixels long etc.

2α is the angular aperture of the camera.

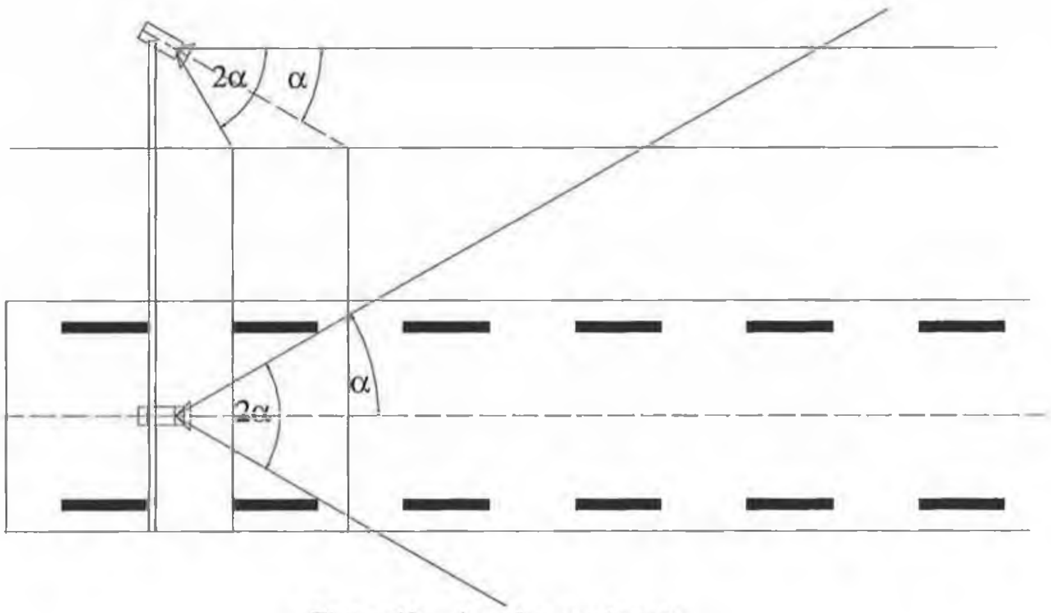


Figure 10. Angular aperture α.

Resolution of the camera is given as *n* x *n* where *n* is the number of pixels horizontally and vertically on the image sensor.

d is the distance along the y-axis that the camera is placed. This is important for determining where the vehicle is positioned in the lane between the lane markers. As with *h* above, *d* is measured in pixels in the algorithm and should be set to match *h* in a ratio of the real world situation.

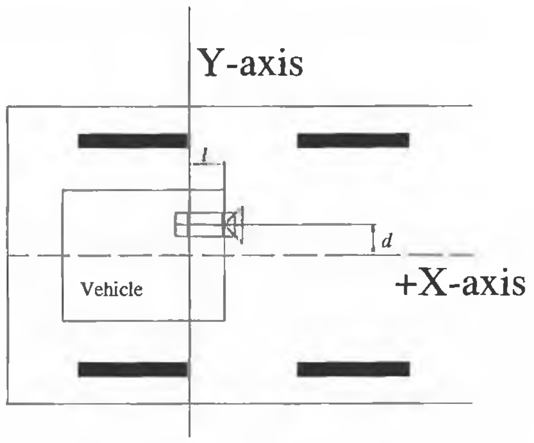


Figure 11. The *l* and *d* variables.

l is the point on the x-axis where the camera lens is positioned. If the camera is assumed to be positioned at the origin (0,0), then both *l* and *d* will be zero. As with *h* and *d* above, *l* is measured in pixels.

2.5.3 Lane Trajectory Planning and Steering algorithms

After determining where the lane markers and/or the lane are positioned, the next action is to determine how the vehicle should move in order to follow the lane. Assuming a two dimensional flat plane for the vehicle to drive on, this will usually take the form of starting with a number of target points which the vehicle should hit while moving from its current position to final destination. In order to get the vehicle to hit these points a curve should first be fitted to hit or come close to each of these points and the final task is to determine how the vehicle can follow this curve. This entails taking into account the kinematics of automobile type vehicles, in particular as the vehicle changes its position the steering angle required to maintain the vehicle's position along the curve that has been generated from the target points.

2.5.3.1 Trajectory planing by cubic curve

In [6] such a target point following algorithm is described. The dynamics of a vehicle of automobile type are described as follows:

$$\dot{x} = v\cos\theta \tag{4}$$
$$\dot{y} = v\sin\theta \tag{5}$$

$$\dot{\theta} = \frac{v}{l} \tan \alpha \qquad (6)$$

Where (x, y) is the position of the vehicle, θ is the heading of the vehicle, v is the speed of the vehicle, α is the steering angle and l is the wheelbase of the vehicle. The relations hold when the vehicle drives without slip. Let (X_0, Y_0) and θ_0 be the current position and heading respectively of the vehicle in the fixed reference frame, the X-Y system. Also let (X_1, Y_1) be the target point and θ_1 be the expected heading of the vehicle at the target point as shown in Figure 12.

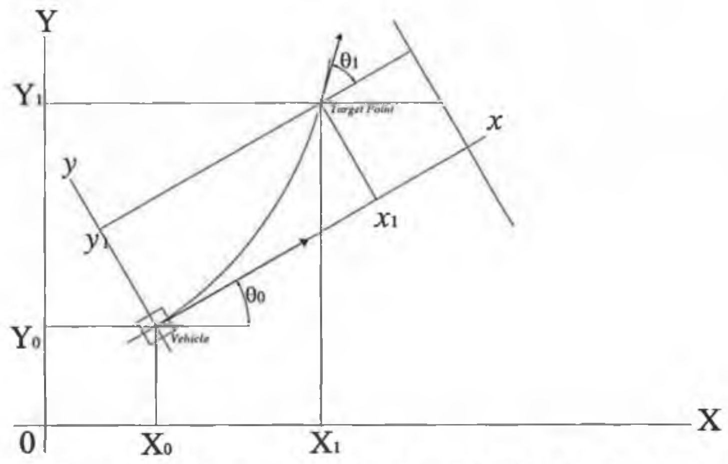


Figure 12. Co-ordinate system for Target Point Following Algorithm.

In the new co-ordinate system, the x-y system (see Figure 12), where the position of the vehicle is the origin and its heading is zero, let (x_1, y_1) be the current target point and θ_1 be the heading (assume that $\theta_1 \neq \pm\pi/2$). The headings are assumed to be tangential angles of a curve going through the origin and the target point at these points. Then, a cubic curve that goes through the two points, (0,0) and (x_1, y_1), is uniquely defined as follows:

$$y = ax^3 + bx^2 \qquad (7)$$

Where

$$a = \frac{x_1 \tan\theta_1 - 2y_1}{x_1^3} \qquad (8)$$

$$b = \frac{3y_1 - x_1 \tan\theta_1}{x_1^2} \qquad (9)$$

Then the steering control angle at the origin in the x-y system that leads the vehicle along the cubic curve to hit the point (x_1, y_1), with the heading θ_l is given as follows:

$$\alpha = \tan^{-1}(2lb) \qquad \textbf{(10)}$$

Where l is the wheelbase of the vehicle. Wheelbase is the distance from the front axle to the back axle of the vehicle.

2.5.3.2 Trajectory planning by template matching.

In [10] a method of trajectory planning via template matching is described. There is a brief description provided in section 2.5.1.5.2.

2.6 Conclusions of the Literature Review.

From the literature review two things become apparent. Firstly the number of types of objects within a road scene that can be used to determine the trajectory of the lane. Secondly the varying methods that have been employed to analyse road scene images to determine the whereabouts of these objects that will help determine the trajectory of the lane ahead.

Of the objects that can be used to detect a lane in a roadway scene it would seem that lane markings, which are usually white or yellow painted markings, are the most prevalent throughout the world and therefore, deserve the most attention. While all roads have edges, this is not necessarily the best determinant of where the lane that applies to the vehicle is going. The other objects such as RPMs and Reflection posts may be a good backup system to increase the robustness of a more general system but would only be useful where these objects are prevalent.

As mentioned above there are a large number of ways of analysing images of roadways. Edge detection is popular but appears to be computationally intensive. The Histogram method described in section 2.5.1.2 claims a much less computationally intensive method than most edge detection methods. The frequency domain method in section 2.5.1.3 seems quite a novel approach to the problem. Not much is said about its performance from a processing standpoint but it is claimed to work well with a very large and varied collection of roadway images. This seems to be the most important attribute for the systems described in sections 2.5.1.4 and 2.5.1.6 where dealing with multiple types of lane markers is considered. Clearly this would be computationally intensive but is likely to be the way any viable system for use in a real world situation would need to be implemented. However, a system like this may well have to wait for powerful enough hardware at a price acceptable to the market before it becomes viable.

Finally the Hough transform method described in section 2.5.1.8 seems to adopt a similar approach to the frequency domain method though the transforms are different. This is however, likely to be as computationally intensive and this is suggested although it does describe ways to minimise this.

This work requires the implementation of a system from road scene image capture to determination of the controls for the vehicle to correctly traverse the road scene. For this reason some of the algorithms from this literature review are entirely or partially used to implement this complete system.

In order to keep build complexity of the model road system low for this project only white lane marking systems are considered and this rules out the systems mentioned previously that used other types of objects, such as road edges, reflection posts or raised pavement markers.

At some point in every system the trajectory must be mapped to the real world plane that the vehicle is traversing. The camera does not show this plane exactly but instead shows a distorted view of it due to the perspective effect. Given that at some point this mapping must be carried out it would seem a high priority to be implemented as part of the work of this project. Many algorithms may well determine the few target points that define the trajectory for the car to follow using the perspective distorted image and then map these few to a real world plane thus requiring fewer operations. However, this may well result in detection of lane markings and determination of target points becoming more difficult (both in terms of complexity and processing power) than if the image was already displayed as a birds eye view before the detection of lane markings commences.

From the point of view of this thesis the inverse perspective mapping is investigated significantly as it is a sub-algorithm that can be very useful to the implementation and development of future lane detection and following algorithms.

A number of methods for the actual detection of lane markings and determination of the lane are outlined in the literature review and any of these would suffice for this work. However, in order to add a certain amount of novelty to the system the algorithm that is implemented for this part is based on that outlined in the histogram method from section 2.5.1.2. This is a somewhat novel approach as the ideas put forward in this algorithm are based on the road scene image that has not had the perspective effect corrected. Also [7] which describes this method doesn't detail in depth the implementation details of how the algorithm works and so in implementing this system some novel solutions are devised.
Once the lane markers that make up the lane have been determined, a set of target points for the car to hit needs to be determined. In section 2.5.1.1 a relatively simple system for carrying out this operation is described. However, this system is used along with edge detection to determine the lane edges, whereas in this project the lane's edges have already been determined earlier. For this reason parts of that algorithm described in section 2.5.1.1 are used but modified to suit the information available from the previous sections implemented in this work.
Once target points have been decided, a curve which links these points is calculated. In section 2.5.3.1 a mathematical formula is outlined for a curve that a vehicle can follow, which can join two points. Implementation and testing is carried out on this system and also on one other type of curve not mentioned in the literature review, namely the b-spline curve. Having achieved a curve for the vehicle to follow, the final task is to determine the control signals to be given to the vehicle for it to traverse this curve. Section 2.5.3.1 outlines the links between the curve and the steering angles and forward motion needed to traverse a curve. This is the system implemented in this work for that purpose.

3 Project Implementation and Development.

3.1 Introduction.

This chapter describes the implementation and development that took place in this project. This includes the ideas for devising a future complete model system and also those parts of the model system that are implemented as part of this work. It also describes in-depth the different sub algorithms that are implemented towards an overall system that would encompass all necessary operations from image acquisition to the generation of appropriate control signals for a future model vehicle.

The structure of the chapter is as follows:

A model system is proposed and described. Some small parts of the model system are implemented and these are described.

The implementation of Inverse Perspective algorithm is described as is the set-up of the system for testing it.

The implementation of the Lane Detection and Lane Marking Recognition algorithms are described in detail. This deals with all operations between the output of the Inverse Perspective algorithm to the determination of a set of target points that the vehicle should follow.

The chapter ends with a description of the implementation of a Path Trajectory and Steering Algorithm, which bridges the gap between obtaining target points for the vehicle to follow and how the vehicle actually follows these target points.

3.2 Towards the development of a Model System

What is required is a physical model that could be used in the future for testing and comparing various lane following algorithms as well as other vision controlled automated vehicle algorithms. It is required that this be small enough to allow testing inside a laboratory setting. In order to keep costs low a standard off the shelf 1:10 radio controlled model car is used as the test vehicle. This means that all other items can be designed to be a 1/10 scale of their life-sized equivalents. The vehicle must be capable of housing a

computer board, stand-alone. This rules out the use of a PC Desktop or Laptop, as both would be too large and heavy for the task. While a system sending data from the car to a host computer via radio link for analysis could be used, it would cause unnecessary extra development work and could be affected by RF interference.
The model must also be capable of housing a camera module. This camera is used to capture the images of the road/lane and send the image data to the computer board inside the model. The model vehicle must have a motor for the driving wheels and servo control of the steering. The computer board must have some way of controlling both of these.

3.2.1 The Model Vehicle

The model vehicle used is a NIKKO radio controlled car approximately 1:10 scale model. This approximation of scale is as a result of NIKKO using one size of chassis but different body covers that relate to real life cars that vary in size. However, the bodywork cover is not important to the project and so a 1:10 scale will be assumed. The vehicle has front and rear suspension, a differential gear system and a DC motor for the driving wheels that uses a 7.2-volt battery pack to drive it. The steering control however, uses a system whereby the steering, under direction from the remote control operator will turn fully left or right and when not given an input will revert to a central position. This will be exchanged for a servo motor system whose angle of rotation is determined by the width of a control pulse supplied to it.

Figure 13. The NIKKO Model Vehicle.

3.2.2 Processing and Controller Board

Since the main board of the RC car is for the specific purpose of controlling the car via radio controlled signals and not capable of analysing image data for control decisions, it is replaced with a DSP board. Also the drive system for the driving wheels motor simply switches a relay allowing direct connection from the battery source to the motor when the operator moves that control on the RC controller. This means that the RC car moves at full speed all the time until the operator takes their finger off the control. This means that there is no way for this board to vary or control the speed of the drive motor. So the new board must have a way of controlling the speed of this drive motor.

Digital Signal Processors are highly efficient at processing image data and therefore, the perfect choice for this type of application. Cost and flexibility were the most significant concerns when determining a DSP board to use for this project. With this in mind a low cost board that has support from an open source Real Time Operating System was chosen.

3.2.2.1 BF533 Stamp Board.

This is a low-cost development platform for the ADSP-BF533 Blackfin device. The STAMP board is part of the Blackfin/uClinux open source project.

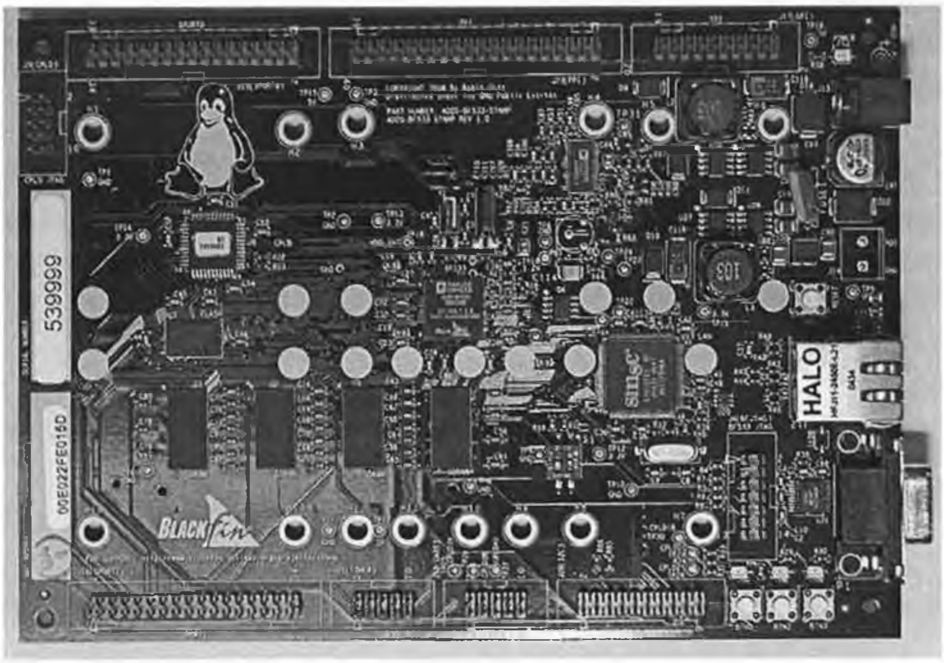


Figure 14. The Blackfin BF533 STAMP board.

An overview of some of the STAMP board's features is given below:

- ADSP-BF533 Blackfin device with JTAG interface.
- 500MHz core clock.
- 133MHz system clock (SCLK).
- 64M x 16bit external SDRAM (128MBytes).
- 2M x 16bit external flash (4MBytes).
- 10/100 Mbps Ethernet Interface.
- UART interface with DB9 serial connector.
- 270-pin expansion interface.
- CPLD with JTAG interface allowing for custom configuration of the Ethernet interface, external memory and programmable flags.
- Connectors to various Blackfin peripherals: PPI, SPI, SPORT0, SPORT1, IrDA, I^2C and Timers.

3.2.2.2 The Blackfin BF533 Processor

The following description of the Blackfin BF533 Processor is taken from [20]:

The ADSP-BF533 processor is an enhanced member of the Blackfin processor family that offers significantly higher performance and lower power than previous Blackfin processors while retaining their ease-of-use and code compatibility benefits. The BF533 processor is completely pin compatible, differing only in its performance and on-chip memory, mitigating many risks associated with new product development. The Blackfin processor core architecture combines a dual MAC signal processing engine, an orthogonal RISC-like microprocessor instruction set, flexible Single Instruction, Multiple Data (SIMD) capabilities and multimedia features into a single instruction set architecture.

The BF533 peripherals include:

- Parallel Peripheral Interface (PPI).
- Serial Ports (SPORTs).
- Serial Peripheral Interface (SPI).
- General-purpose timers.
- Universal Asynchronous Receiver Transmitter (UART).
- Real-Time Clock (RTC).
- Watchdog timer.
- General-purpose I/O (programmable flags).

These peripherals are connected to the core via several high bandwidth buses as shown in Figure 15.

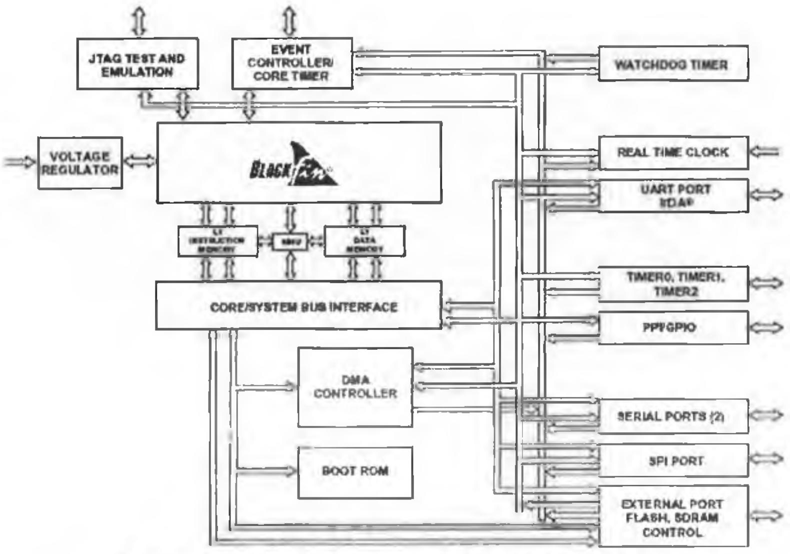


Figure 15. Block Diagram of the Blackfin Processor.

All of the peripherals, except for general-purpose I/O, Real-Time Clock and Timers, are supported by a flexible DMA structure. There are also two separate memory DMA channels dedicated to data transfers between the processor's memory spaces, which include external SDRAM and asynchronous memory. Multiple on-chip buses provide enough bandwidth to keep the processor core running even when there is also activity on all of the on-chip and external peripherals.

The important peripherals from the point of view of this project are the PPI, General-Purpose timers, General-purpose I/O. The Serial Port is also important for debugging and monitoring the controlling of the target board from the Linux based host machine. As will be detailed later in section 3.2.5.1 the PPI interface was used to interface to the Camera Module. This incorporates I^2C to read and write registers on the Camera Module and DMA for transferring the picture data from the Camera Module to the Stamp board for processing. General Purpose I/O (programmable flags) are used to take inputs from push button switches on the board and output to status LEDs on the board. Some of the General Purpose Timers have External pins and can be configured as timers or PWM (Pulse Width Modulation) outputs. The steering control for the Model car uses a servomotor, which moves to a given angular position based on a specific PWM signal being sent to it. Also the drive motor for the model car will have its speed controlled by PWM.

A short description of these peripherals and DMA can be found in Appendix B.

3.2.3 Software for the Target Board.

Since any system for testing would need to be flexible and perhaps also run several algorithms at the same time, having a RTOS and development environment for the board is advantageous. One of the main advantages of the Stamp BF533 board is that there is a port of the uClinux free open source operating system available for it.

3.2.3.1 uClinux

The original uClinux was a derivative of Linux 2.0 kernel intended for microcontrollers without Memory Management Units (MMUs). However, the Linux/Microcontroller Project has grown both in brand recognition and coverage of processor architectures. Today's uClinux as an operating system includes Linux Kernel releases for 2.0, 2.4 and 2.6 as well as a collection of user applications, libraries and tool chains. A port of the uClinux operating system is available for the Blackfin processor. The uClinux distribution comes with a menu configuration system that allows the developer to select the Blackfin BF533 stamp as the target [21].

Embedded system software can be characterised by two extreme cases. One case is where the software driving the system might be a totally customised application specific package, with very narrow focus. Alternatively an application specific software package may lie on top of an underlying, somewhat general purpose embedded OS, which provides various capabilities for the application from its general tool set [22].

3.2.4 The Camera Module

A camera module is required to capture the image of the road scene. Since only grey scale values are required a Black and White or Mono camera suits the application fine. However, it would be advantageous to have a digital output, as an analogue output would require extra A/D converters.

The camera module chosen for the application was the M3188A 1/3" B/W Camera Module with digital output.

General description and features taken from [23] follows:

The M3188 is a 1/3" B/W camera module with digital output. It uses OmniVisions CMOS image sensor OV7110.

The digital video port supplies a continuous 8 bit-wide image data stream. All camera function, such as exposure, gamma, gain, windowing, are programmable through I^2C interface.

Features:

307,200 pixels, VGA/CIF format.

- Small size: 40 x 28 mm.
- Lens: f = 6mm(Optional).
- Read out – progressive/ interlace.

- Data format – 8 bit video data.
- I^2C interface.
- Electronic exposure/Gain/Control.
- Image enhancement – brightness, contrast, gamma, sharpness, window, etc.
- Internal/external synchronisation scheme.
- Frame exposure/ line exposure option.
- Single 5V operation.
- Low power consumption (< 300mW).

The camera module comes with a 32-pin output.
The pin out and short description of each pin are detailed in Appendix A.

3.2.5 Integration of Development System.

3.2.5.1 Connecting Camera Module to Stamp board.

Much work on drivers and applications had already been carried out for the Blackfin port of uClinux by developers in the open source community and much of these were useful to carry out this project.
Since the project involves connecting a camera module to the stamp board, one driver and test application was particularly relevant, the ppi_driver. This driver can be used to capture video Frames from the PPI port.

The interface to the camera sensor is shown in Appendix A.
This driver also has the following features:

- High speed data input from the PPI device via DMA.
- Configuration of the PPI device.
- Interrupts Transfer Complete and Error Detection.
- Fasync notification of Transfer Complete.

The PPI Input device has a number of configurable features; this driver uses the following:

- POL_S - invert Frame Sync 1 and Frame Sync 2.

- POL_C - invert Clock.
- PPI_DATA_LEN - port width.
- PPI_PACKING - Pack 2x8-bit words into a single 16-bit output.
- CFG_GP_Input_3Syncs - Selects full (Hsync, Vsync and Frame) or single Frame Sync.
- GP_Input_Mode - selects input mode.

The PPI Transfer can use a GPIO pin to start the transfer process [25].

This ppi_driver proved very useful in this project. Even though it was written for another camera module, it had a similar interface and worked in much the same way. The ppi_driver that was already developed used the Micron MT9M001 1.3 Mega pixel colour image sensor, whereas this project used the Omni-vision OV7110 0.3 Mega pixel monochrome image sensor.

Most of the pin connections shown in the interface diagram for the Micron camera module had an equivalent connection pin on the Omni-vision camera module though under a different name. The pin connections used to connect the Omni-vision camera module are detailed in Appendix A.

Two major omissions from the interface diagram for the Micron camera module that were required when connecting up the Omni-vision camera module are as follows.

1. Pull up resistors are required for I^2C data and clock lines, namely SDA and SCL.

The formula for calculating the value of the pull up resistors is

$$R = \frac{50}{d} \qquad \textbf{(11)}$$

Where R is the required resistance in kΩ.

d is the number of devices on the bus.

2. The pin PF3 requires a pull down resistor. When using RX mode with 3 external frame syncs and only 2 syncs are needed, configure the PPI for three-frame-sync operation and provide an external pull-down to GND for the PPI_FS3 pin [20].

The PPI driver did not need any modification to work with the Omni-vision camera module. However, the application code that uses the driver did need modification. The uClinux port included a test application for the PPI driver, which assumed the PPI port was connected to a Micron MT9M001 camera module. This test application needed some minor modifications in order to operate the Omni-vision OV7110. These were mainly in configuration of the PPI port, the DMA and buffer set up and sending appropriate commands to the camera module. The commands for the camera module are sent via I^2C link to the camera chip. Note that the camera module chip calls this bus SCCB rather than I^2C. However, the I^2C bus from the main board works perfectly with this bus system. The correct register and setting must be determined based on information from [24].

The test application allowed for one or more frames of video to be captured. It also stored the latest image in a bitmap file whose name could be specified at the command line.

3.2.5.2 Running the Test Application.

In order to run the test application the application had to either be included in the overall build image of the RTOS or compiled separately and then downloaded to the target after boot up. The uClinux system has a user friendly configuration interface for including or omitting functionality or applications from the RTOS image. Although in some cases it may require direct modification of make-files or configuration files, which later in the process generate make-files.

The stamp board has two communication links with the host PC, serial RS232 and Ethernet. The RS232 can be used to download a RTOS image to the target but is very slow. Using Ethernet, this can be achieved in less than 10 seconds, although this does depend on the size of the image. The serial connection is mainly used for monitoring what is happening on the target board. All standard output is sent from the target board to the host Linux PC via RS232 and can then be displayed in a shell. A command line for the target board can also be controlled from a shell on the host Linux PC and this information is sent via RS232. In order for this communication to take place there must be a communication mechanism set up on the Host such as Telnet or Kermit etc. In this project Kermit was used.

Once the RTOS has booted, either after download from the Host PC or from boot memory on the board, the user is presented with a command prompt. From here the user can run applications included in the compiled image or download files/applications from the Host PC. Downloading requires that a file transfer protocol be set up on the target. A TFTP application can be included in the configuration and so in the final image the TFTP application will be present. This allows the download and upload of files and applications. As mentioned earlier, the test application for the camera attached to the PPI port can specify a bitmap file in which to store an acquired image. Since the interface to the target is a text only interface, this bitmap cannot be viewed. However, using TFTP the bitmap can be uploaded to the Host machine for viewing. This bitmap file can then be used for host based testing of image processing algorithms.

While no major algorithm tests were carried out using the Stamp Board and camera, one short algorithm was run in order to prove that a continuous stream of images could be analysed and control outputs given, based on the current image frame.

The camera module was placed inside a white box. Then using a straight black piece of cardboard, the black cardboard was waved back and fourth in front of the camera.

Each image frame analysed the image, by splitting the image into horizontal sections and determining the darkest point along the horizontal. Then all darkest points relating to all of the horizontal sections were averaged. This average was assumed to be the average position of the black cardboard in the image scene. The position is then used in the determination of an angle, based on the left side of the image being the largest negative angle and the right side being the largest positive angle. The centre of the image is the angle zero.

This angle was then converted to an appropriate pulse width to turn a servo motor to that angle. The servo was attached to the steering mechanism of the test vehicle. So when the application was run, moving the black cardboard in front of the camera resulted in a proportional change in the steering angle of the front wheels of the vehicle. A short video clip of this can be found on the CD, accompanying this Thesis, in the "Video" directory.

3.2.6 Implementation on the STAMP board.

The following is a list of the work implemented on the STAMP board:

Images are successfully captured by the camera module and transferred into the memory of the DSP board. This entailed the hardware link up of the camera module to the STAMP board as well as the software required to communicate between the two.

The DSP board is successfully configured to communicate with a Host Linux workstation. This allows for the DSP board to be controlled and files to be sent and received from the Linux workstation to the DSP board.

Software is created for the DSP board that successfully analyses the image data to determine the location of a black piece of cardboard against a white background. This software also successfully sends appropriate control signals to a servo motor attached to the wheels of the model vehicle, which resulted in the wheels of the vehicle turning to follow the movements of the piece of black cardboard.

As is outlined in the Discussion section of this Thesis some issues would need to be resolved before the STAMP board could be used as a good system for implementing vision system applications. To ease development and implementation of algorithms all algorithms, from this point on in the Project, are implemented on IBM based PCs.

3.3 Implementing the Inverse Perspective Algorithm.

The original image captured by the camera was chosen to be 400 X 400 pixels.

The image captured by the camera is actually 640 X 480 but is cropped to 400 X 400.

The test image below (see Figure 16) shows a 640 X 480 image taken with the camera module.

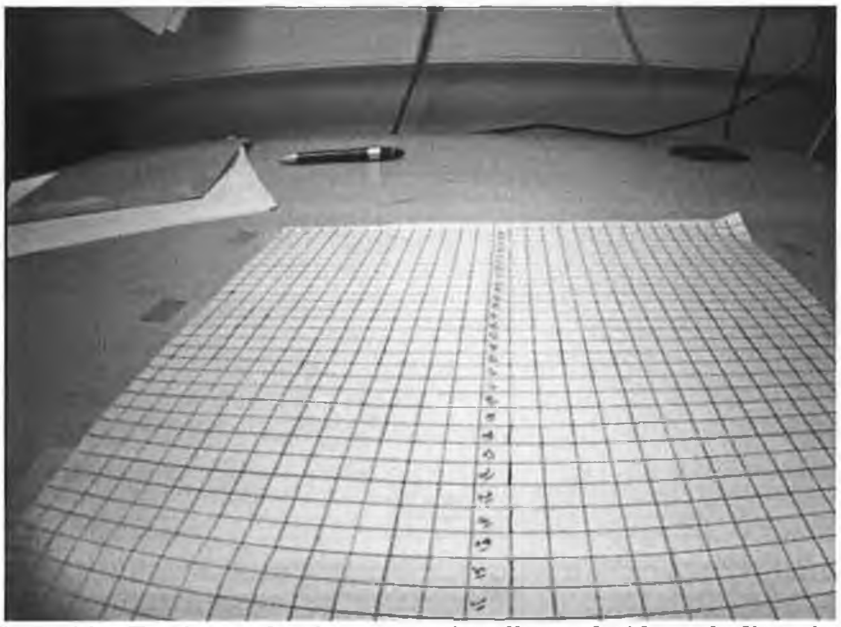


Figure 16. Test image showing perspective effect and wide-angle distortion.

Despite appearances each of the horizontal lines are straight but wide-angle distortion has caused these to appear as arcs. The same distortion can be noted at the top of the image where the white electrical conduit is arcing in the other direction. The other important thing to note here though, is that the centre of this distortion does not appear to be in the centre of the image but is instead to the right of centre. Another telltale sign of this is the darkened edges on the left particularly at the bottom. This suggests that the lens is not centred over the CMOS image sensor, as demonstrated in Figure 17.

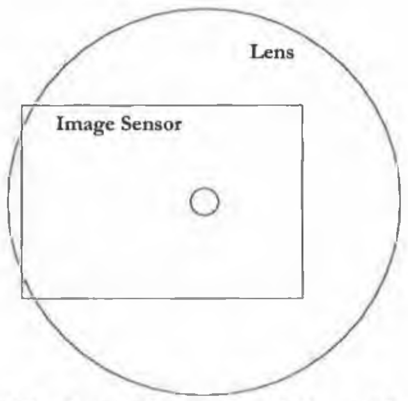


Figure 17. Lens off centre with Image Sensor.

This means that taking a 400 X 400 crop of the image in the exact centre of the image would cause complications in the inverse perspective mapping as the algorithm assumes that the centre of the image is on the optical axis i.e. the centre of the lens.

The inverse perspective mapping algorithm used in this project is based on [14][15][16][17][18][19] which are described in section 2.5.2.

From this point on, the image captured by the camera will be referred to as the captured image and the remapped image that contains the result of the inverse perspective mapping will be known as the remapped image. Each pixel in the remapped image must take a pixel from the captured image or be left blank. The algorithm cycles through each pixel of the remapped image and determines where this pixel would be taken from in the captured image. If it finds that the pixel that this specifies is from outside the boundaries of the captured image then this pixel is left blank. Blank will be assumed to be the colour black.

The set-up for this system used a sine table, (a heavy and cumbersome piece of equipment) with the camera module mounted on it. The sine table was set up to give a θ value of 20°. The model roadway is made from fibreboard painted black and the lane markings are white duct tape.

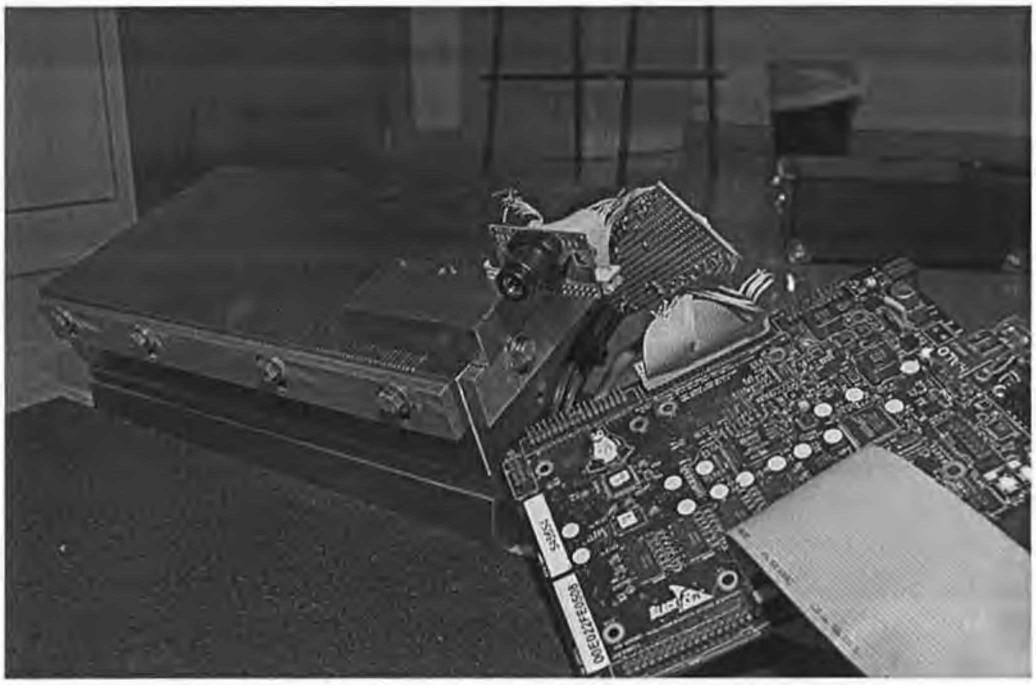


Figure 18. The set-up of the system with camera module mounted on Sine Table and connected to STAMP board.

Under these circumstances the θ angle could be guaranteed to within the tolerance of the sine table but no other parameters could be guaranteed, such as height, *l*, *d*, γ etc.
More information on the meaning of the variables used in this algorithm can be found in section 2.5.2.1.

$\bar{\gamma}$ is the angle between the optical axis of the camera and the heading of the vehicle. It is intended in this project to have $\bar{\gamma} = 0$. However, due to tolerances in mechanical set up it may be found that it is not exactly zero and that this must be compensated for in software. This will have to be calibrated through trial and error with a known target.

$\bar{\theta}$: this is the angle between the horizontal, parallel to the (x, y) plane and the optical axis of the camera. The larger this angle, the closer to the vehicle the camera will be able to view. This would be advantageous for close quarter manoeuvring and sharp curves. However, this would lead to less of the road ahead being visible, which may require the vehicle to move at slower speeds as the control system will have much shorter space and therefore, shorter time to react to changing circumstances. A compromise will have to be found here.

h is the height of the camera on the z-axis, with z=0 being the level of the (x, y) plane on which the road is assumed to be. Important to note that the units of height are in pixels.

2α this is the angular aperture of the camera. A definite value for this for the camera module used was difficult to find, but through fine tuning of a known target the value α = 0.30 rads (17.19°) was used.

Resolution of the camera is given as *n* x *n*. *n* was set at 400 pixels for this project. The camera has an actual resolution of 640 X 480 but was cropped to 400 x 400.

d is the distance along the y-axis that the camera is placed. This is important for determining where the vehicle is positioned in the lane between the lane markers. As with *h* above, *d* is measured in pixels in the algorithm and should be set to match *h* in a ratio of the real world situation.

l is the point on the x-axis where the camera lens is positioned. If the camera is assumed to be positioned at the origin (0,0), then both *l* and *d* will be zero. As with *h* and *d* above, *l* is measured in pixels.

3.4 Implementing the Lane Detection Algorithm

It is useful to have only black and white pixels in the image in order to differentiate between objects and the ordinary surface of the road. Therefore, a threshold operation is carried out on the image and any pixel values above the threshold value are assumed to be white and assigned the value 255 (brightest 8-bit value). If a pixel is found to be below the threshold it is assumed to be black and changed to the value 0 (darkest 8-bit value). Determining the appropriate threshold value can be achieved through trial and error and picking an appropriate value for best results given the particular lighting of a scene. [7] also describes a good method for determining an appropriate value for the threshold that is outlined in section 2.5.1.2 of this document. After thresholding has been performed, all pixels in the image will be either black or white. There are now no shades of grey in between. However, each pixel is still stand-alone i.e. black or white. There is no relationship between white pixels that could be said to make up lane markers or other objects. With this in mind the first thing that must be achieved in this lane recognition algorithm is to group white pixels together into objects. After that each of the objects found must be categorised into possible lane marker objects and other non-lane marker objects.

Once all the possible lane marker objects have been determined, the lane marker objects must be categorised into left and right lane markers. Once this is achieved, target points along the centre of the lane can be determined and used in a point following algorithm.

3.5 Lane Marking Recognition.

This must be done in a methodical fashion following a set of rules to determine whether any white pixel should constitute a new object or be part of a previous object. It may also occur that at some point two objects meet and should become a single object. Before an algorithm is designed for this, it is important to decide what attributes might be important to describe any given object. For example, with a lane marker object it would be advantageous to know the direction the marker is pointed as this would indicate the direction of the lane at that point etc.

Since objects will be built up pixel by pixel there must be a way of determining the overall attributes of the object. This can be done either by way of dynamically recording all the pixels belonging to a particular object, so that later these can be looked at together to determine the attributes of the object, or by way of updating all the object's attributes as each pixel is added.

After consideration of the problem to be solved, it was determined that the important attributes to be considered for each object are as follows:

Smallest X value.

Smallest Y value.

Largest X value.

Largest Y value.

Size of the object in Pixels.

In order to get an idea of the direction of the lane marking, a least squares fit to a straight line is used. This means the following information will be required for each object:

Sum of all X values of the pixels in the object.

Sum of all Y values of the pixels in the object.

Sum of the square of all X values of the pixels in the object.

Sum of all X x Y values of each of the pixels in the object.

The least-squares line uses the equation for a line given by

$$y = a + bx \tag{12}$$

Where *a* and *b* are given as follows [26]

$$a = \frac{\left(\sum y\right)\left(\sum x^2\right) - \left(\sum x\right)\left(\sum xy\right)}{n\sum x^2 - \left(\sum x\right)^2} \tag{13}$$

$$b = \frac{n\sum xy - \left(\sum x\right)\left(\sum y\right)}{n\sum x^2 - \left(\sum x\right)^2} \tag{14}$$

Where *n* is the number of co-ordinates being used to obtain the line. In the case of this project *n* will be the number of pixels in any given object. This is recorded as the size attribute of the object. With this in mind each object has an *a* and *b* attribute associated with it.

With lane markings tending to be in a vertical direction in the image, knowing an approximate point at the near end and far end of the lane marker is advantageous. The Smallest X and Largest X values are applicable here but corresponding Y values to go with these are required. Note the Smallest and Largest Y values are not appropriate for this, as those inform us more about width in the horizontal direction of the image. So *Near end Y* and *Far end Y* attributes are also recorded for each lane object.

A *centre co-ordinate* for each object is also recorded, as this will be beneficial to best determine the position of the lane marker.

To give each object an identity, each object must have a *unique number* associated with it.

Finally it cannot be known in advance how many objects will occur throughout the image and how many lane markings will be linked as left and right lane markers. For this reason linked lists are used to link up objects. Therefore, each object will have an attribute which will *link it with another object* so that they can be daisy-chained together.

So the C struct will look as follows:

```
struct object{
        int number;
        int smallestX;
        int smallestY;
        int biggestX;
        int biggestY;
        int nearEndY;
        int farEndY;
        int CX; /*centre X */
        int CY; /*centre Y */
        int size;
        float sumY;
        float sumX;
        float sumX2;
        float sumXY;
        float a;
        float b;
        struct object* next;
};
```

3.5.1 Calculating a Value for each of the Attributes.

Rather than trying to calculate values for the entire object's attributes after each has been fully created, it was decided to do this as the scanning of the pixels for object generation was in process.

As each new pixel is added to an object the following attributes of that object are updated:

Sum of Y.

Sum of X.

Sum of X^2.

Sum of $X \times Y$.

Size of the object increases by one.

The following variables are checked to see if the new pixel is a more appropriate value than the current value

Smallest X.

Smallest Y.

Biggest X.

Biggest Y.

Should it be determined that a new pixel results in two previously created objects being joined then one of the objects will always take precedence and the other will be destroyed. Before the other is destroyed the following attributes will be added to the same attribute in the surviving object:

Sum of Y.

Sum of X.

Sum of X^2.

Sum of X x Y.

Size of object.

And again before the object is destroyed, the following variables are checked to see if the object due for demolition has more appropriate values than the surviving object.

Smallest X.

Smallest Y.

Biggest X.

Biggest Y.

When the algorithm is finished scanning the image, it will be left with a number of objects, which it now needs to categorise. The other attributes don't need to be calculated until the algorithm has determined that they are possible lane markers.

3.5.2 Grouping Pixels together as Objects.

Up to now it has been assumed that there is some way of determining whether a pixel should belong to a given object or not. The process of how this is achieved is now discussed.

The algorithm cycles through each of the pixels in the image starting at the bottom left of the image and going along the line of pixels horizontally until the right side of the image is reached. Then the algorithm starts on the left pixel second from the bottom and again moves from left to right. This continues until it reaches the top right of the image.

An object in the context of this algorithm is defined as one or more white pixels that are touching each other. Touching is defined as two pixels being near neighbours of each other.

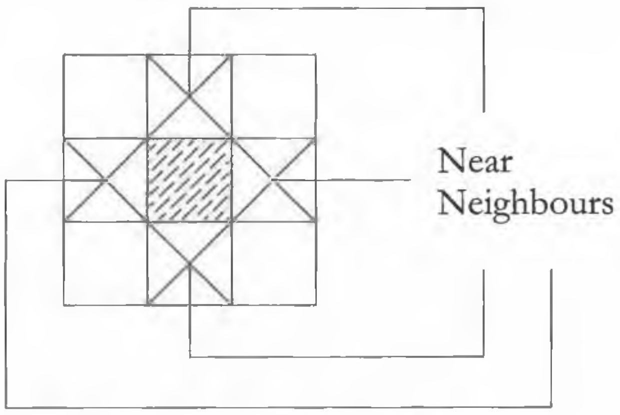


Figure 19. Near Neighbour pixels.

As the algorithm looks at each new pixel it must first be ascertained whether it is black or white. If it is white it must then be determined if it is touching another white pixel, or if not, to start a new object. Given the sequence in which the pixels are being scanned, it is determined that it is only necessary to check the near neighbours to the left and below, as being possible white pixels that might be touching. The near neighbours to the top and right have not been scanned yet and will be dealt with when the scanning gets to that part of the image.

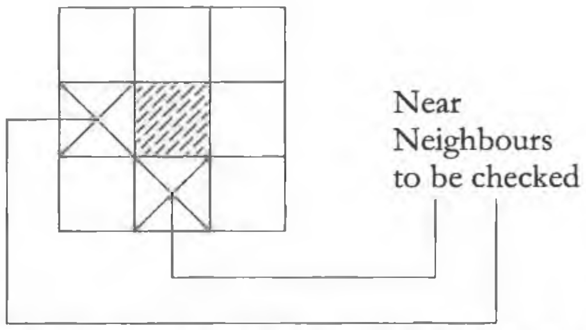


Figure 20. The two Near Neighbours that will be checked.

Since the image is held in an array, care must be taken not to try to access elements outside of the array. Therefore, special care must be taken with pixels on bottom and left edges, as this is where the algorithm could test for a near neighbour that is outside the bounds of the image. Hence the very first pixel, bottom left, must be dealt with as a special case as it has

no near neighbours below or to the left. All other pixels on the bottom row are treated as another special case as they can only have near neighbours to the left, but not below.
From that point on the first pixel in each row is treated as a special case as it can only have near neighbours below and not to the left.
There is also a special case when the end of a row is reached. This will be explained later.
If a new object is to be created, first a temporary object is created. As the algorithm moves along the line, if it comes across more white pixels touching each other, they are added to the temporary object. When a black pixel is finally met the temporary object is made into an ordinary object and added to the list. This should also happen if the end of a line is reached. Should it be found at any-point that a member pixel of the temporary object is a near neighbour of an already established object, then the temporary object is assimilated by the established object and the temporary object is zeroed and made ready for re-use. There is only ever one temporary object at a time, so it can be re-used. The temporary object will only ever have member pixels on one horizontal line (row of pixels).

3.5.2.1 Bottom Corner Pixel

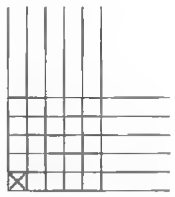


Figure 21. Bottom corner Pixel

<u>Is pixel white?</u>

Yes White:- Initialise the temporary object with information from this pixel.

No Black:- Ignore and move on.

3.5.2.2 Remainder of the pixels in the Bottom Row.

Figure 22. The remainder of the pixels on the bottom row.

<u>**Is pixel White?**</u>

***Yes White*:-** <u>**Is near Neighbour to left White?**</u>

  ***Yes White*:-** Update the temporary object with info from this pixel.

  ***No Black*:-** Initialise the temporary object with info from this pixel.

***No Black*:-** <u>**Is near Neighbour to left White?**</u>

  ***Yes White*:-** Copy info from temporary object into full object and add to list of objects. Set temporary object back to default zeroed values.

  ***No Black*:-** Ignore and move on.

3.5.2.3 Dealing with all other pixels.

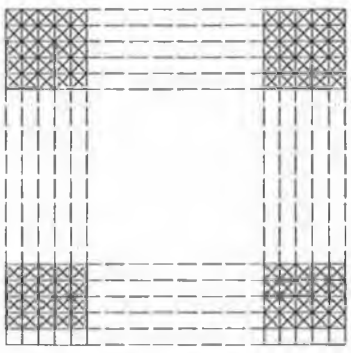


Figure 23. All other Pixels

<u>**Is pixel white?**</u>

***Yes White*:-** Is near neighbour below white?

  ***Yes White*:-** Square below is part of established object, update this object with info from this pixel.

  <u>**If there is a near neighbour to the left, is it white?**</u>

  ***Yes White*:-** Check if it is part of a temporary object and if it is, assimilate the temporary object into the established object. If near neighbour to left is also part of an established object, check that it is not the same object this pixel is attached to, if it isn't then merge the object that the near neighbour to the left is part of to the object the current pixel is part of.

  ***No Black*:-** Ignore and move on.

  ***No Black*:-** <u>**If there is a near neighbour to the left, is it white?**</u>

  ***Yes White*:-** Check if it is part of a temp object and if it is, add this current pixel to it. If it's an established object also add this current pixel to it.

No Black:-This current white pixel has no white pixel near neighbour below or to the left, so initialise temporary object.

No Black:- **If there is a near neighbour to the left check if it's white?**

Yes White:- **Is it a temporary object?**

Yes temporary object:- Copy info from temporary object into full object and add to list of objects. Set temporary object back to default zeroed values.

No not a temporary object:- do nothing.

No Black:- ignore and move on.

Finally after each row has finished the following must be performed in case there is a temporary object right up to the end of the row that has not been closed off and made into an ordinary object yet.

Check size of temporary object, is it non zero?

Yes non zero:- Copy info from temporary object into full object and add to list of objects. Set temporary object back to default zeroed values.

After this algorithm is run there should be a list of objects for the image, which include all the white pixels in the image.

3.5.3 Criteria for determining lane markers from other objects.

Once all the objects are inserted in a linked list, the first operation to be carried out is to remove any objects that are definitely not lane markers. These would include any objects that have very few pixels (for example less than 8) and those that have far too many pixels (for example more than 200). This eliminates many objects from the list, which will speed up searching at later stages.

3.5.4 Sorting Lane markers into Right and Left Lanes

3.5.4.1 Finding first lane markers in Left and Right lanes

In order to build up the set of lane markers, it is first necessary to find the first lane marker objects on each side of the lane. Once this is done then knowledge gained about these first lane markers can be used to more accurately determine where the other lane markers are. Of course incorrect identification of the first lane markers will lead to problems and errors in identification of further lane markers.

While not investigated as part of this algorithm, information from previous image frames could be used to help correctly determine the first lane markers. So in order to find the first lane markers the algorithm must be given criteria to determine which, of all the objects still available, is one of the first two lane markers. The following strategy was decided upon to find the first lane markers.

Search through all the objects and find the object that has the smallest *SmallestX* value but that also has a size larger than some pre-determined value (for example 30 pixels). The reason for picking a size value is not that the lane marker must be larger than this size, but often the first lane marker may be mostly cut off by what is not visible by the camera. This means that the object chosen may have too few pixels to get accurate information concerning it, such as slope. Since finding all other lane markers will rely on the details of these first lane markers it is important that the details be as accurate and complete as possible and so should be above a certain number of pixels in size. The first lane markers are also used to determine the current heading and position of the vehicle and so it is more important to get accurate information from these lane markers than from any others. Once this object has been identified, it should be determined whether the lane marker is on the right or the left of the lane. This is achieved simply by checking in which half, right or left, the lane object lies. Once this is determined, the algorithm will try to find the first lane marker on the other side. This is found by searching through the list of objects for an object that has a centre Y value of between 80 and 120 pixels away. These limits are based on the measured number of pixels separating the lane markers in images (approx. 90 pixels see section 4.2.3.2). Objects outside of this range are assumed not to be the lowest lane marker on the other side of the road. As in the last case the lane marker is expected to be above a certain size (e.g. 30 pixels) for the same reasons as given before. Because more than one object may fall into these aforementioned criteria the object that is closest to the bottom of the image is chosen from all those that fit the criteria.

At this point, the algorithm has now determined the right and left lane markers recorded in the image that are closest to the camera.

3.5.4.2 Sorting the remainder of the lane markers into Right and Left lanes

Since the two objects above have now been assigned as the first lane markers in the left and right sides of the lane, they are each put at the head of their own object list, known as the left lane list and right lane list. Since they are now in lists of their own they are removed from the original overall list, which means they will not need to be checked in any further searches.

The next task for the algorithm is to determine all of the lane markers in the left side of the lane. This entails cycling through the list continuously until all left lane markers are found. On each cycle through the list the most appropriate lane marker candidate will be picked and then removed from the main list and a new search will begin. The algorithm will know when to stop looking for new left lane markers when a cycle through the main list reveals no appropriate candidates that meet the criteria. After the first cycle through the main list the left lane marker closest to the first one should be chosen as the most appropriate candidate. On the next cycle, the next lane marker on the left should be chosen and so on. Due to the fact that there are expected to be many lane markers on each side, the criteria must be chosen in such a way that only one of these be identified on each cycle. Therefore, a priority system is used which assigns points to each object that meets particular criteria. Some criteria are considered higher priority than others and so at the end it is expected that one object will have acquired more points than all other objects and as such is deemed to be the most appropriate object on that pass through the algorithm. The criteria chosen and points awarded are given as follows:

Note: *lastObject* is the previous chosen object to be added to the lane list. This is the most important lane marker object towards finding the next lane marker.

searchObject is the object currently being tested for appropriateness to be added to the list of lane markers.

Criterion 1: Is the searchObject's smallestX bigger than the lastObjects biggestX value? Awarded 8 points. This is weighted the highest; objects not meeting this couldn't possibly be the next lane marker.

Criterion 2: Is the searchObject's centreY less than 40 pixels from the centreY of the lastObjects centreY? Awarded 4 points. This is based on the measured value of approx. 90 pixels for the width of the lane (see section 4.2.3.2). If the lane marker is more than 40 pixels along the Y-axis away it is highly unlikely to be the next lane marker. It is not impossible that it is in the same lane, as a curve could cause the lane markers on one side to progress towards the other side of the image, but it is unlikely that the centre value of two consecutive lane markers would have centreY values this far apart.

Criterion 3: Is the distance from the far end of the lastObject to the near end of the searchObject less than 50 pixels? Awarded 2 points. It would be expected that most contiguous markers would be in this range and so one found in this range should be given a higher priority over the others.

Criterion 4: Calculate the slope between the far end of the last object to the near end of the search object and determine if this slope is less than 0.3 rads different from the slope of the last object. Points awarded 1. This 0.3 rads was picked as an arbitrary value, though if the maximum curvature of a given road system is known it could be calculated what the maximum difference in slope between two lane markings would be. In the straight-line road it is expected that all lane markers including those on the other side of the lane could meet this criterion and therefore, it is given a low number of points.

The different points awarded make it easy to pick one lane marker out of all the possibilities because the object that meets the most criteria in the list will be chosen on any given search. If a search reveals no lane marker that meets all the criteria then the searching stops.

After each new marker is identified as being the most appropriate lane marker found in that search, that lane marker is removed from the main list and added to the end of the left lane list. In this way, new searches do not need to check these again and the left lane list is now in the order starting with the marker closest to the camera and progressing to the marker furthest away from the camera.

Next the right hand side of the lane is looked at. Already the algorithm has the first lane marker on the right hand side. The main list now only contains right hand lane markers and

non lane markers. All left-hand lane markers should have been removed by now. This means less objects overall to search, which entails less processing time.
The search for right lane markers takes the same form as that shown for the left lane markers above.

3.5.5 Determining a set of target points to follow.

Given an image that is 800 pixels in height (along X-axis), an arbitrary figure of 50 pixels was chosen for calculation of each point vertically on the image. Therefore, the target points should each have an X value that is a multiple of 50 pixels. The y value must then be calculated for each of these 50 pixel lines. Each of these lines should intersect with the line of the left lane and the line of the right lane. However, the line may intersect with left or right at a point where a lane marking is absent. If this is the case the algorithm must interpolate between the lane markers above and below the point of intersection in order to determine an accurate point of intersection of the overall lane way. In fact a similar scheme must be determined when the 50-pixel lines intersect with a lane marker. While the lane marker is considered a single object, only certain physical attributes are recorded about it. The *a* and *b* values allow a mathematical equation for a straight line through the lane marker to be determined. If an equation for each 50-pixel line could also be determined, then through mathematical manipulation, the point of intersection could be determined. However, since there is a chance that it won't intersect a lane marker, a general system for finding the point of intersection between the 50-pixel line and the lane edge is used.
Each lane marker has three known points i.e. Near end, far end and centre. These all have both X and Y values associated with them. Taking the left lane edge for each 50-pixel line, the algorithm cycles through each lane marker object and determines the closest known point below and above the 50-pixel line. Then it determines the point of intersection between the 50-pixel line and a line joining the two determined points. This then gives a set of points, which should follow the left edge of the lane. The same is then performed for the right lane edge and results in a set of points, which should follow the right lane edge.
Once this is complete then the centre point between each pair of corresponding left and right points is determined. This results in a set of points, which should follow the centre of

the lane along its path. Before progressing further however, a number of potential exceptional scenarios of this system need to be dealt with.

Firstly there may be no lane markers for the first 100-150 pixels. In these cases, points should be plotted where the algorithm has determined the vehicle is positioned. For example if the camera module is situated in the exact centre of the vehicle then the front position of the vehicle is at the centre bottom of the image i.e. (x, y) point (0, 200). So where no lane markers have yet been found the vehicle should maintain this line e.g. (0,200), (50,200), (100,200) etc.

The next exceptional scenario can occur when there is a lane marker visible on one side of the lane but not on the other. In this case the central point should be calculated as being 45 pixels away from the lane edge intersection with the 50-pixel line in the appropriate direction along the y-axis. The figure of 45 is chosen as the lane width was found to be approximately 90 pixels (see section 4.2.3.2).

The last exceptional scenario is towards the top of the image where there are no longer any lane marker objects from which to calculate points. In this case it was determined that the best course of action was to continue plotting points based on the direction of a straight line between the last two properly calculated points. However, only one of these is actually needed or useful for the generation of a curve that follows all these target points as will be discussed in section 4.3.7. Continuing to follow points along this straight line would not be wise and the algorithm should not consider points beyond this. Any points beyond this must come from the acquisition of another image frame.

3.6 Path Trajectory and Steering Algorithm.

3.6.1 Introduction

After it has been ascertained where it is intended the vehicle should go, the next challenge is to get the vehicle to actually go there. Assuming a two dimensional flat plane for the vehicle to drive on, this will usually take the form of starting with a number of target points, which the vehicle should hit while moving from its current position to its final destination. In order to get the vehicle to hit these points a curve should first be fitted to hit each of these points and the final task is to determine how the vehicle can follow this curve. This entails taking into account the kinematics of automobile type vehicles. As the vehicle changes position, the steering angle required to maintain its position along the curve must be generated from the target points.

3.6.2 Using Cubic curve method

In [6] such a target point following algorithm is described. See section 2.5.3.1 for more information on this algorithm.

3.6.2.1 Generating the curve

To implement this algorithm on computer/digital hardware it must be remembered that a computer cannot deal with continuous mathematical functions. For this reason the mathematical functions described in [6] and section 2.5.3.1 must be implemented in a discrete form. Also the functions are given in terms of velocity which doesn't affect trajectory shape, assuming no slip, so the equations were changed to a form which specified the trajectory in terms of distance and not time. The pseudo code below describes how the algorithm is used to draw the cubic curve from (0,0) to (x_1, y_1) operates

```
*Loop until ($x_{new}$ is greater than $x_1$ AND $y_{new}$ is greater than $y_1$)*
    $v_{total} <= v_{total} + v$
    $x_{new} <= x_{previous} + v\cos\theta_{previous}$
    $y_{new} <= y_{previous} + v\sin\theta_{previous}$
    $\theta_{new} <= \tan^{-1}(3ax_{new}^2 + 2bx_{new})$

    *draw a straight line between ($x_{previous}$,$y_{previous}$) and ($x_{new}$,$y_{new}$).*
    $y_{previous} <= y_{new}$
    $y_{previous} <= y_{new}$
    $\theta_{previous} <= \theta_{new}$

*Back to beginning of loop*
```

Note that v is now a distance rather than a speed. v_{total} is the total distance that has been travelled along the curve up to that point. Any value can be chosen for v, however, the larger the value of v chosen, the less like a continuous curve the result will be. This will lead to a lack of precision in the result. If a very small value of v is chosen, then this will be much more like the continuous curve but will require more processing. Should this algorithm turn out to be a bottleneck in an overall system then changing this value might lead to important compromises between speed and precision. Also note in the pseudo code above that θ_{new} is the new angle of slope of the tangent to the curve i.e. $\tan^{-1}\left(\frac{dy}{dx}\right)$

In section 2.5.3.1 it was seen that $y = ax^3 + bx^2$ so $\frac{dy}{dx} = 3ax^2 + 2bx$

So from this it can be seen that $\theta = \tan^{-1}(3ax^2 + 2bx)$.

The algorithm is implemented in the Java programming language given that language's powerful graphics libraries.

3.6.2.2 Determining the steering angle.

In [6] it is stated that the steering control angle at the origin in the x-y system that leads the vehicle along the cubic curve to hit the point (x_1, y_1), with the heading θ_1 is given as follows:

$$\alpha = \tan^{-1}(2lb) \qquad \textbf{(15)}$$

However, in most cases the steering angle must change as the vehicle traverses the curve. So in order to determine the required steering angle at each point along the curve another method was used and is described below.

Since a continuous curve is not being dealt with here but a discrete approximation to the curve then a number of discrete steering angles must be obtained. The steering angle can be obtained as shown in Figure 24, for any given point along a curve.

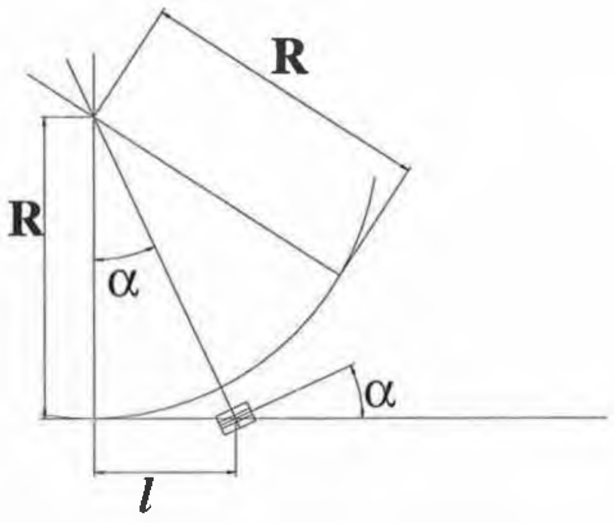


Figure 24. Wheel base *l*, steering angle α and Radius R

The computer must of course calculate the angle by means other than diagram method. The wheelbase *l* will be known for any given vehicle but the radius R must be calculated. To calculate R its relationship with the curve must first be understood. The radius is given as follows

$$R = {}^{1}\!/_{Curvature} \tag{16}$$

Where Curvature is defined as the second differential of *y* with respect to *x*

i.e. $$curvature = \frac{d^2y}{dx^2} \tag{17}$$

The first differential $\frac{dy}{dx}$ is the instantaneous slope of the tangent to the curve, i.e. what has earlier been referred to as θ. So the second differential must be the instantaneous change in θ. Since a discrete system is being dealt with here then curvature must be the change in θ over the distance in which this change takes place, i.e. what has been referred to earlier as *v*. Therefore,

$$curvature = \frac{(theta_{new} - theta_{previous})}{v} \tag{18}$$

So the radius is

$$R = \frac{1}{\frac{(theta_{new} - theta_{previous})}{v}} \tag{19}$$

$$\text{or } R = \frac{v}{(theta_{new} - theta_{previous})} \qquad \textbf{(20)}$$

This will give a set of discrete radii for each discrete change in θ along the discrete approximation of the curve. From Figure 24 above it can be seen that the steering angle can then be calculated as

$$\alpha = \tan^{-1}\left(\frac{l}{R}\right) \qquad \textbf{(21)}$$

Again it is important to note that these will be discrete steering angles and not continuously changing steering angles.

3.6.3 Using a B-spline method.

A quick reminder is perhaps desirable at this point as to what information will be available before the trajectory is calculated. The lane detection algorithms will generate a set of target points, which the vehicle will be expected to hit. The previous algorithm only shows how to get between any two of these points. More specifically the algorithm works on the assumption that the first point is the origin (0,0) and that the heading of the vehicle θ is 0 rads at the first point. One of the examples in the results section 4.4.1 for the cubic curve shows the vehicle starting at the origin with a heading of zero and moving to the point (12,8) with a final heading of 1.3 rads. Now consider three target points, i.e. (0,0) (θ = 0), (12,8) (θ =1.3) and (20,6) (θ=-0.4). In this case it has already been shown how to get from (0,0) to (12,8) but how to then proceed on to (20, 6) poses a problem. To use the same algorithm as before it must first be assumed that (12,8) is actually (0,0), this would mean that the relative position of (20,6) would be (8, -2). The second assumption however, would change this again. It states that the heading at the first point is assumed to be 0. This would mean that the position (8, -2) must be rotated around the origin by –1.3 rads. This is a far from insurmountable problem but if for example, five target points are considered then after the second point is reached all three of the other points must be transposed in the same way. Then after the third point the last two must be transposed etc. Add to this the fact that new target points are likely to be added on the fly as the vehicle moves along a roadway and the camera picks up further points along the road.

Since each frame of video will take time to process, during this time the vehicle will have continued to move and may have performed one or more transposes before the new targets are calculated. This will require noting the co-ordinate space at the moment the frame was captured and trying to link these new points into the co-ordinate space of the targets already in the system. Thus the problem becomes much more complex and more importantly, heavier in computational terms.

3.6.3.1 B-spline.

In the previous case having the vehicle hit each point by generating an approximate curve between each point and then patching each of these curves together was considered. Matching the endpoints of the curve segments in this way is not considered the most acceptable way. It is important also to match the gradients and defining curves by the points through which they pass. This does not lend itself very well to patching [28].

Before looking at the B-spline, consider the more general Bezier curve. Bezier curves are defined using four control points, known as knots. Two of these are the end points of the curve, while the other two effectively define the gradient at the end points. These two points control the shape of the curve. The curve is actually a blend of the knots [28].

One disadvantage of Bezier curves is that they do not have much local control. This means that as the number of points increases the effect of any individual point/knot along the way is reduced. This is due to the nature of the blending used for Bezier Curves. They combine all the points to create the curve. The obvious solution is to combine only those points nearest to the current parameter.

These points are labelled internally from 0 to (number of points) –1. To calculate the curve at any parameter t a gaussian curve is placed over the parameter space. This curve is actually an approximation to a gaussian curve; it does not extend to infinity at each end, just to +/- 2 [29]

$$B(u) = \frac{1}{6}(2+u)^3 \qquad -2 < u \leq -1 \tag{22}$$

$$B(u)=\frac{1}{6}(4-6u^2-3u^3) \qquad -1<u\le 0 \tag{23}$$

$$B(u)=\frac{1}{6}(4-6u^2+3u^3) \qquad 0<u\le 1 \tag{24}$$

$$B(u)=\frac{1}{6}(2-u)^3 \qquad 1<u\le 2 \tag{25}$$

$$B(u)=0 \qquad \textit{otherwise} \tag{26}$$

It is helpful to put this in a general form where *u* can only vary between 0 and 1. In the first case above substitute *(u-2)* for *u*. This gives,

$$B(u)=\frac{1}{6}(2+u-2)^3=\frac{1}{6}(u^2)=\frac{1}{6}(u^3+0u^2+0u+0) \tag{27}$$

In the second case above substitute *(u-1)* for *u*.

$$B(u)=\frac{1}{6}(4-6u^2-3u^3)=\frac{1}{6}(4-6(u-1)^2-3(u-1)^3)=\frac{1}{6}(-3u^3+3u^2+3u+1) \tag{28}$$

In the third case above *u* is already in the appropriate range, so simply put in the same end form as the others.

$$B(u)=\frac{1}{6}(4-6u^2+3u^3)=\frac{1}{6}(3u^3-6u^2+1u+4) \tag{29}$$

In the final case above substitute *(u+1)* for *u*.

$$B(u)=\frac{1}{6}(2-u)^3\Rightarrow\frac{1}{6}(2-u-1)^3=\frac{1}{6}(1-u)^3=\frac{1}{6}(-1u^3+3u^2-3u+1) \tag{30}$$

A general matrix form for the cubic B-spline can be constructed from these new equations which match that found in [30]. It is given as:

$$S_i(u)=\begin{bmatrix}u^3 & u^2 & u & 1\end{bmatrix}\frac{1}{6}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}P_{i-1}\\ P_i\\ P_{i+1}\\ P_{i+2}\end{bmatrix} \text{for } u\in(0,1) \tag{31}$$

For the x and y co-ordinates specifically these are given as

$$x_{cur} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{P_{i-1}} \\ x_{P_i} \\ x_{P_{i+1}} \\ x_{P_{i+2}} \end{bmatrix} \quad \textbf{(32)}$$

$$y_{cur} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_{P_{i-1}} \\ y_{P_i} \\ y_{P_{i+1}} \\ y_{P_{i+2}} \end{bmatrix} \quad \textbf{(33)}$$

u is a fractional part of 1, i.e. it must always lie somewhere between 0 and 1. Given that what will be generated is an approximation to a curve made up from small single line segments, if *u* is chosen to increase in increments of 0.1 for example, then this will mean that there will be 10 straight-line segments between each of the points approximating the curve. If *u* is chosen as 0.01 there will be 100 straight-line segments between each of the points approximating the curve. Again a trade off between precision in approximating a continuous curve and the processing power available is clearly seen to exist here.
Matrix calculations like those above require mainly multiplication and accumulations.
The multiply-accumulate is a basic building block in DSP. In fact, the speed of DSP-based computers is often specified by how long it takes to perform a multiply-accumulate operation [27].

4 Results

4.1 Introduction

This chapter looks at the results of the running the algorithms that are implemented in this project

A set of test images is used in the testing. These compose of a number of roadway images as follows

An image of a straight stretch of roadway.

An image of a section of roadway with an S-bend ahead.

An image of roadway partway through the S-bend.

Each of these images are considered first, with regard to the Inverse Perspective Mapping algorithm. Once these have been passed through the Inverse Perspective the images will have changed. So after each resulting image has been discussed with regard to the Inverse Perspective the resulting image is then used as the input test image to the next algorithm, namely the Detection of Lane Markings algorithm.

So the three road images mentioned above have now all had the perspective effect removed and are now input in the Detection of the Lane Markings algorithm. The results for the three sections of roadway are then discussed.

Again this algorithm will produce a set of results on the three images, which can be used as the input information into the next algorithm namely the Path Trajectory and Steering algorithm. This algorithm should produce as a result a curve that defines the trajectory for the vehicle to follow. The three different curves for the three different road sections are then discussed.

In order to test the algorithm for Steering and further look at the algorithm for Path Trajectory, the algorithm is tested in a more general way by using the Java programming language with a graphical output. This enables different types of curves to be looked at as well as arbitrary target points to be chosen. It also permits the varying of vehicle properties to be modified to see the effect this has on steering control. The results of running these algorithms are discussed at the end of this chapter.

4.2 Inverse Perspective Mapping Results

In this section, results of applying the inverse perspective to test images captured by the camera module are looked at. Images of three sections of model road are looked at. The first is entirely straight; the second has a left turn followed by a right turn and the third shows just a right turn.

In the image examples, the captured image will be 400 X 400 pixels but the remapped image will be 400 X 800. This is simply to make the results easier to look at. In a final running of the algorithm 400 X 400 or other resolutions (e.g. 200 X 400) could be used to save processing time.

4.2.1 A Straight Piece of Road Way

The image in Figure 25 shows a straight stretch of road. The perspective effect is very noticeable. It would also appear that the camera module is not centred in the lane.

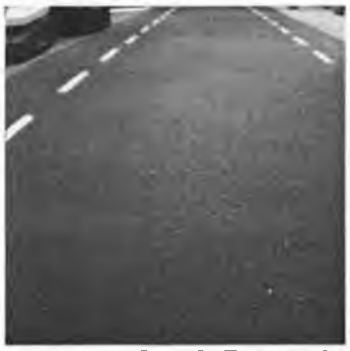


Figure 25. Long straight section of road. Perspective effect is clearly visible.

Applying the inverse perspective mapping algorithm to this image the output obtained is shown in Figure 26.

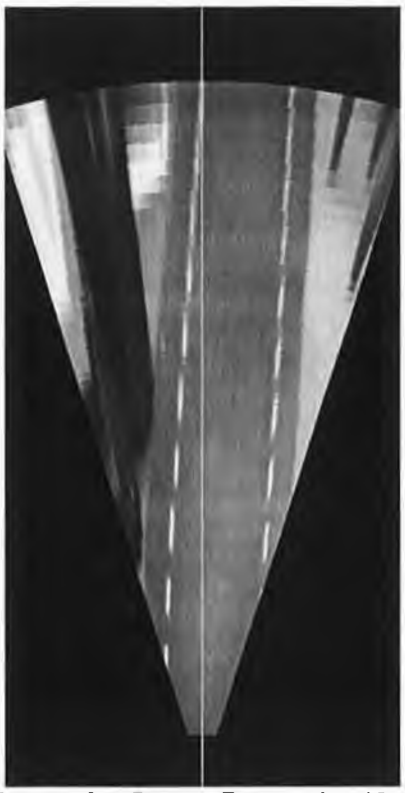


Figure 26. Corrected image after Inverse Perspective Algorithm has been applied

The pure black pixels are pixels on the x-y plane that the camera cannot see. All other pixels are within range of the camera module.

Attempts were made to line up the camera module to be in a straight line with the lane (i.e. $\gamma=0$) and it would not appear from the captured image (see Figure 25) that there was a significant error in this. However, it can be seen with the remapped image (see Figure 26) that the road, while appearing to be straight is at an angle to the x-axis, which is shown as the white line up the centre of the image ($y = 0$). If the camera module was mounted and calibrated mechanically on a vehicle to be directly in line with the heading of the vehicle, then this result would inform the control system that the vehicle is not on the correct heading in order to traverse this stretch of roadway. This would mean that steering control would be required to bring the vehicle back in line.

4.2.2 Calibration

This section looks at the resulting effects of changing the values of variables that were explained in sections 2.5.2.1 and 3.3.

4.2.2.1 Effect of γ value

In section 4.2.1 it was assumed that the camera was perfectly calibrated mechanically and the reason the road didn't proceed vertically up the image was that the camera (and therefore, vehicle) were not perfectly lined up in parallel with the lane markings. However, starting with the situation where it is known that the roadway is straight and that the car is on an exact straight heading to traverse this, then it can be inferred from the image above, that the camera is not mounted on the vehicle in line with the vehicle heading. In other words it would show that the γ value is not zero. In this case mechanical adjustments could be made to the camera mounting to calibrate it. Within reason this could also be calibrated in software. For example looking at the situation where the γ value used for the inverse perspective mapping is changed from 0 to –0.06 rads, the following is the result (see Figure 27).

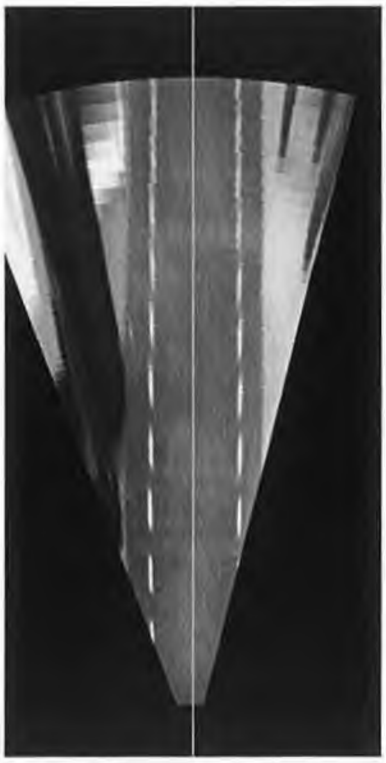


Figure 27. Effect of changing γ value, in this case a value of –0.06 rads is used.

It can be seen that the roadway is now straight ahead. In effect the image has been rotated, compensating for the γ value.

4.2.2.2 Effect of *d* value

It was noted earlier that the camera module didn't appear to be centred in the middle of the lane. This may or may not be a problem. The following should be taken into account.

Is the camera mounted in the centre of the vehicle?

If the camera is mounted in the centre of the vehicle then the above result informs the algorithm that the vehicle is also off centre in the lane and the control system can take this into account in its next control sequence.

If the camera is not centred on the vehicle however, then for the vehicle to be centred in the lane the camera must by virtue of this be off centre in the lane. From the control system's point of view, the positioning of the vehicle (not the camera) is what is important. For this reason it would be useful for the centre of the image to also be the central position of the vehicle. This can be dealt with by modifying the *d* value in the algorithm. This won't change what the camera sees but will change how the remapped image is positioned. The following three diagrams (see Figure 28) show a *d* value of + 10, –10 and – 4. A minus value signifies that the camera is to the left of the centre of the vehicle.

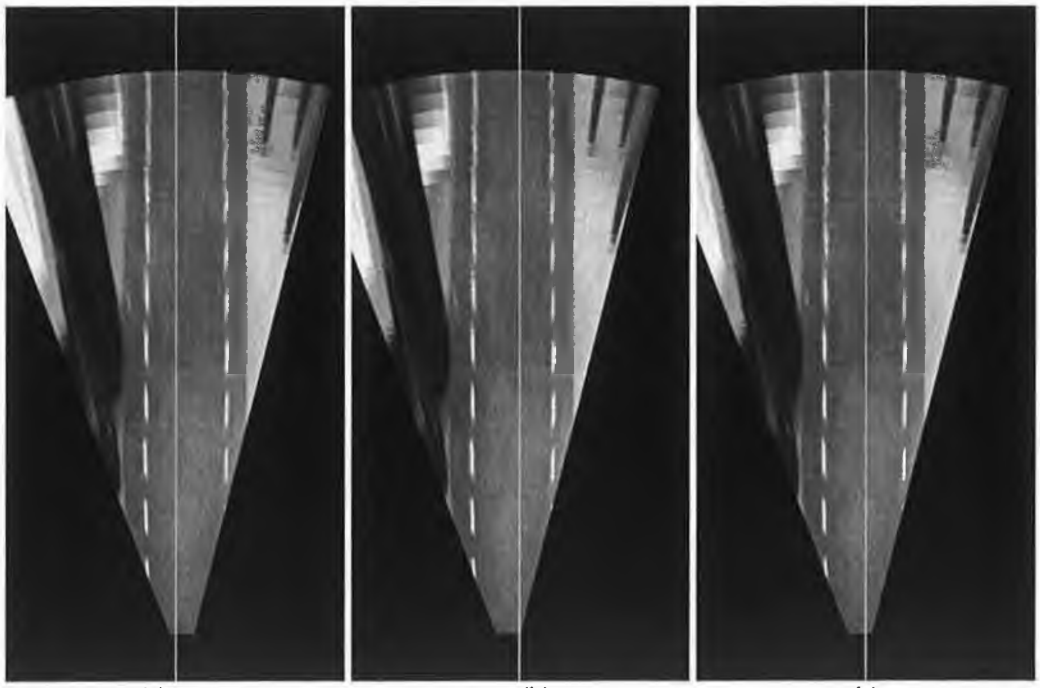


Figure 28. Effect of varying the *d* variable. (a) uses a value of +10, (b) uses a value of –10 and (c) uses a value of –4.

Therefore, if it is known that the camera is a value of 10 to the right of centre of the vehicle, the image informs the algorithm that the vehicle is currently well to the left of centre of the lane (as shown in the first remapped image). If however, it is known that the camera is mounted a value of 10 to the left of the centre of the vehicle then the remapped image informs the algorithm that the vehicle is slightly to the right of centre of the lane. If it is next considered that the camera is mounted a value of 4 to the left of centre of the vehicle then the remapped image informs the algorithm that the vehicle is in the exact centre of the image. It is important therefore, that the positioning of the camera on the vehicle be known in advance so that the re-mapping algorithm can compensate for it.

4.2.2.3 Effect of *l* value

The next variable to look at is the *l* variable, which as mentioned earlier represents the position on the x-axis that the camera is positioned. If this variable is left at zero, it is assumed that the camera module is positioned at x=0. Three examples follow, with *l* values of 0, -40 and +40 respectively (see Figure 29).

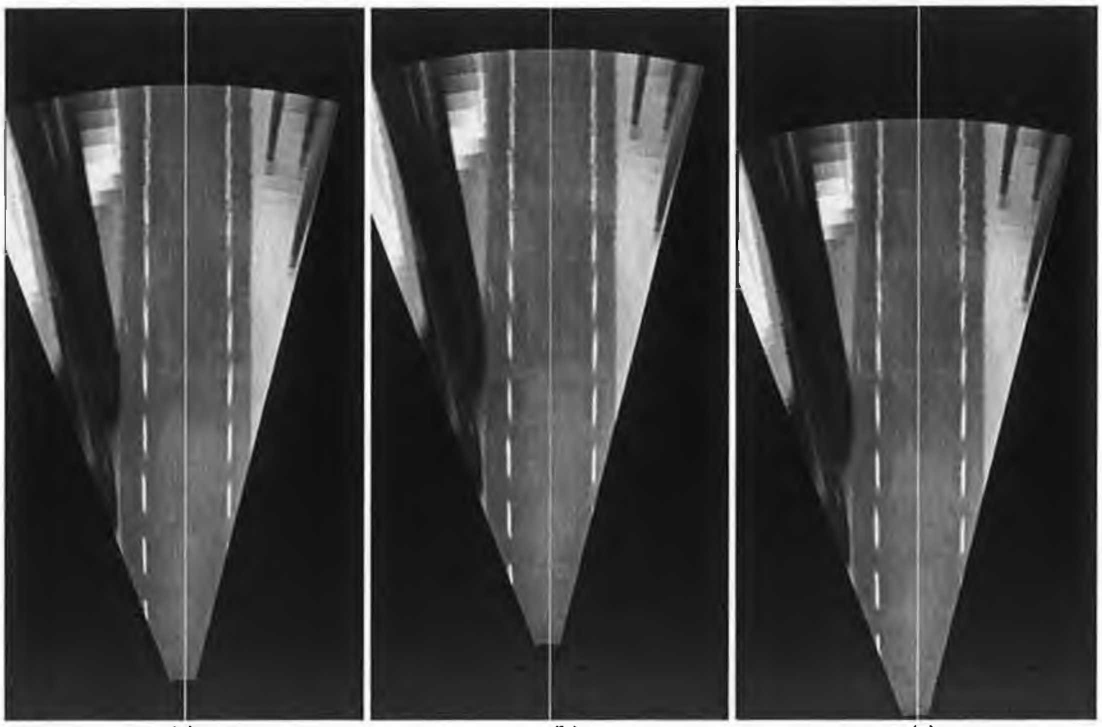

(a) (b) (c)

Figure 29. Effect of *l* variable, (a) uses a value of 0, (b) uses a value of –40 and (c) uses a value of +40.

As can be seen from the results, changing the value of *l* simply moves the valid pixels up or down the x-axis.

4.2.2.4 Effect of *h* value

Next the effect of changing the value of *h* (height) will be investigated. The following three diagrams (see Figure 30) show *h* as a value of 35, 10 and 70 respectively.

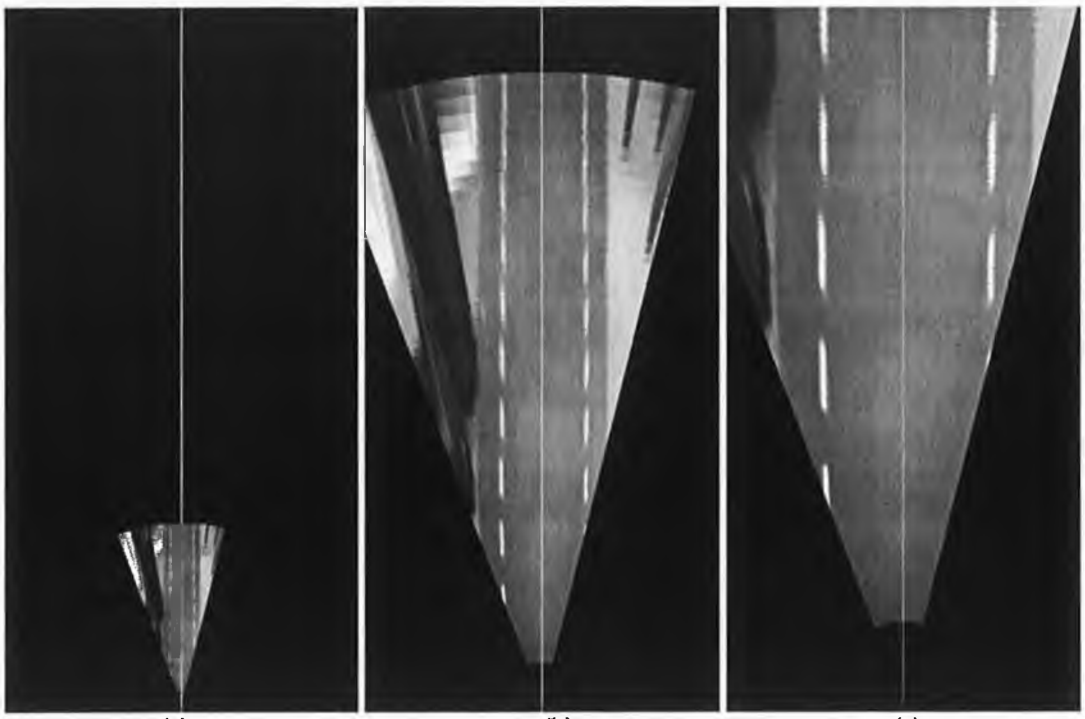


Figure 30. The effect of *h* variable, (a) uses value of 35, (b) uses value of 10 and (c) uses a value of 70.

As stated earlier, *h* is in units of pixels, so changing the value of *h* from 10 to 35 to 70 has a proportional effect on the scene visible to the camera module and will increase or decrease all items in the scene in size proportionally.

4.2.2.5 Effect of α value

2α is the angular aperture of the camera. A definite value for α was difficult to obtain for the camera module used in the experiments. However, through trial and error the value used was 0.3 rads. The following diagrams demonstrate the effect of changing the value of α used in the algorithm to 0.28 rads and 0.32 rads (see Figure 31).

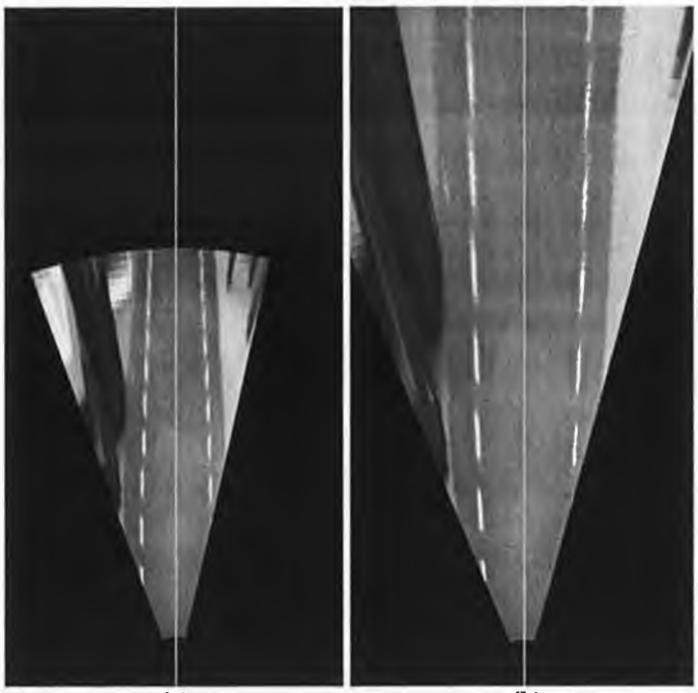


Figure 31. Effect of the α variable, (a) uses a value of 0.28 rads and (b) uses a value of 0.32 rads.

It can be seen in the first case that the algorithm has failed to compensate fully for the perspective effect, as the lane markings are still converging. In the second, the algorithm has over compensated for the perspective effect; the lane markings are now in fact diverging.

4.2.2.6 Effect of θ value

The θ value is 20° or 0.349 rads. The next two diagrams show the effect of changing this value to 0.3 rads and 0.39 rads (see Figure 32).

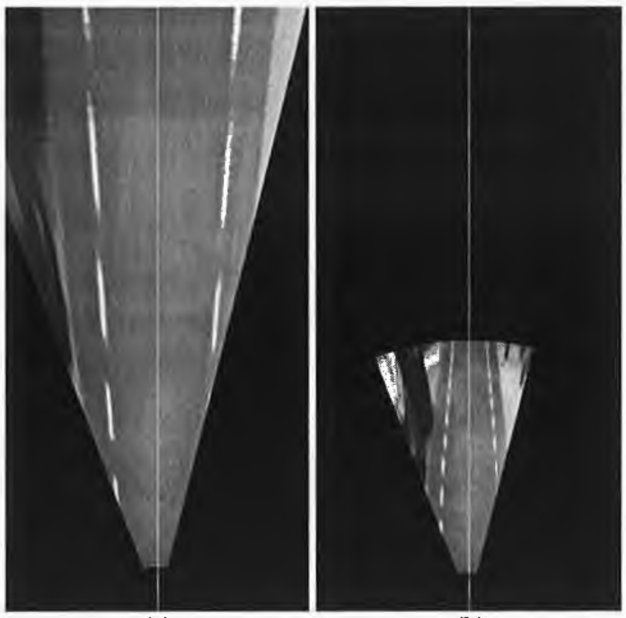

(a) (b)

Figure 32. Effect of θ variable, (a) uses a value of 0.3 rads and (b) uses a value of 0.39 rads.

In the first diagram the lane markings appear to diverge and also it would appear that much less of the road is visible in the remapped image. This is because the algorithm has been given the information that the camera module is at a shallower angle and can therefore, see much further into the distance, so in the section of the x-y plane that is being looked at much less of the original image is seen.

The opposite is the case when the algorithm is given the information that the camera is at a steeper angle. This assumes the image takes up much less space on the x-y plane, as if the camera is looking down at a lower angle. Therefore, it will not see as far into the distance. This leaves an incorrect result with lane markings that converge.

In any mechanical set-up of a camera module on a vehicle it may be required that the θ value be modified in software by small amounts. This compensates for tolerances in the mechanical set-up of the camera.

4.2.3 Using the Results to Measure Distances.

In this section, how the positioning of an individual pixel can relate to a position on the real-life x-y plane is considered.

4.2.3.1 Vertical Distances

Firstly some sort of reference must be used to equate a pixel to a real world measurement unit. In this case the height of the camera module (with respect to the x-y road plane) at the centre of the lens was known to be approximately 130mm. The value of *h* used in the algorithm is 35 pixels. This equates to each pixel representing 3.7mm. Given that each of the lane markings was designed to be approximately 100mm long the expectation is that following the algorithm they would be 100/3.7=27 pixels in length. However, a look at the result shows that the lane markings appear to be 35 pixels long as can be seen from Figure 33.

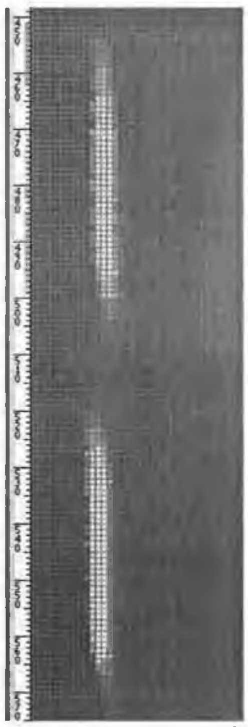


Figure 33. Close up of lane markings in the vertical direction, grid is at 1 pixel resolution.

It is difficult to determine exactly where the edge of the lane marking is, to determine the length of the lane marking. However, all lane markings appear to be approximately of uniform length. The distance between each of the lane markings is also 100mm, so this should give an even better idea of how the transition from white to grey/black is not well defined. This doesn't match with what is expected given the height value in the algorithm though this doesn't pose a major problem since they are uniform, the length at any point can be calculated by multiplying by a pre-determined scaling factor. In this case the value would be 35 pixels = 100mm. This means that each pixel equates to 2.86mm. Later however, it will be seen that this scaling factor is only appropriate for pixels in the vertical direction.

4.2.3.2 Horizontal Distances

Next the horizontal distances are examined; in particular the distance between the lane markings is looked at i.e. the lane width. Given the *h* value of 35 corresponding to 3.7 mm per pixel and the prior knowledge that the lane width is 360mm, the expectation is that the lane width should be approximately 97 pixels. However, as Figure 34 shows this is not what results. Unlike the vertical distances, where the measured value was larger than the calculated value, in this case the measured value is less than the calculated value. The measured lane width comes in at approximately 90 pixels wide.

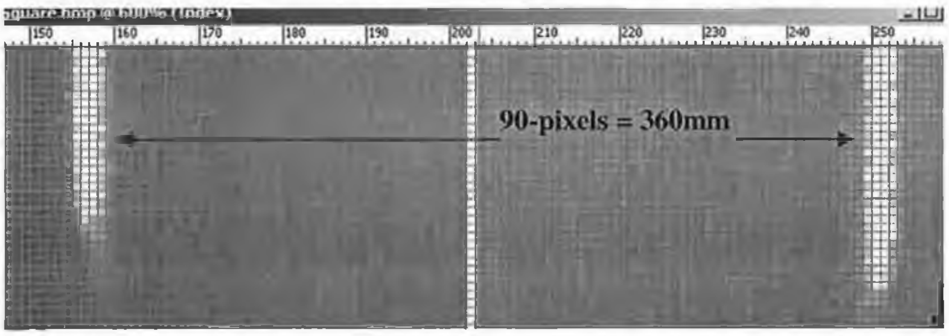


Figure 34. Close up of lane markings in the horizontal direction, grid is at 1 pixel resolution.

Again given the uniform width along the lane this is not a major issue and can be corrected when any measurement is taken by multiplying by a constant value. In this case this means that 90 pixels = 360mm. This means that each pixel equates to 4mm in the horizontal direction.

4.2.4 Roadway section with S Bend Ahead

Now a section of roadway with an S bend is considered. Figure 35 shows the captured image of the curved road.

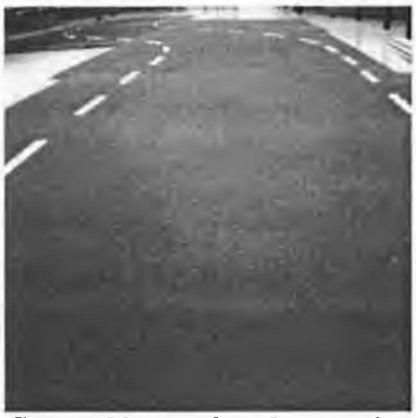


Figure 35. Captured image of roadway section with S-bend

As can bee seen from the image the road proceeds straight and then takes a left turn, followed by a right turn and continues straight again. The remapped image is shown in Figure 36.

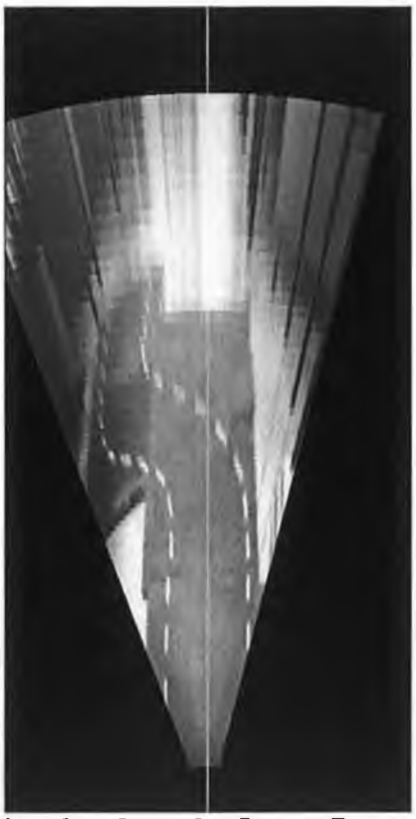


Figure 36. S-bend section of roadway after Inverse Perspective Algorithm applied.

While this gives quite a good representation of the road, the further towards the top of the image and the further towards the side of the image the less clear the road markings become. This is due to the fact that in the original image there are much fewer pixels to detail these

parts and therefore, it cannot be expected that these will be as clear. In the lane detection algorithm section later (see 4.3.1.2 and 4.3.4) it will be seen how successful a computer algorithm is at collecting appropriate information from these parts of the diagram.
One other observation is that the width of the lane should be more or less uniform along the length of the lane. However, comparing the lower part of the lane straight in front of the vehicle to the far off part of the lane, after the s-curve it can be seen that there is clearly a difference in lane width. By measurement the lane width in the lower part of the image seems to be slightly over 80 pixels whereas in the top of the image it seems to be only half that at approximately 40 pixels. This number is only an estimate as the poor resolution in the upper parts of the image makes it very difficult to ascertain the value accurately. This combined with the lack of definition in those lane markings may make them unusable for detection and path determination.

4.2.5 Roadway section half way through S Bend.

In the next example the camera module has again been moved on further along the lane and is this time placed part way through the curve with every effort made, by hand, to put it in parallel with the curve. Figure 37 shows the captured image and the remapped image.

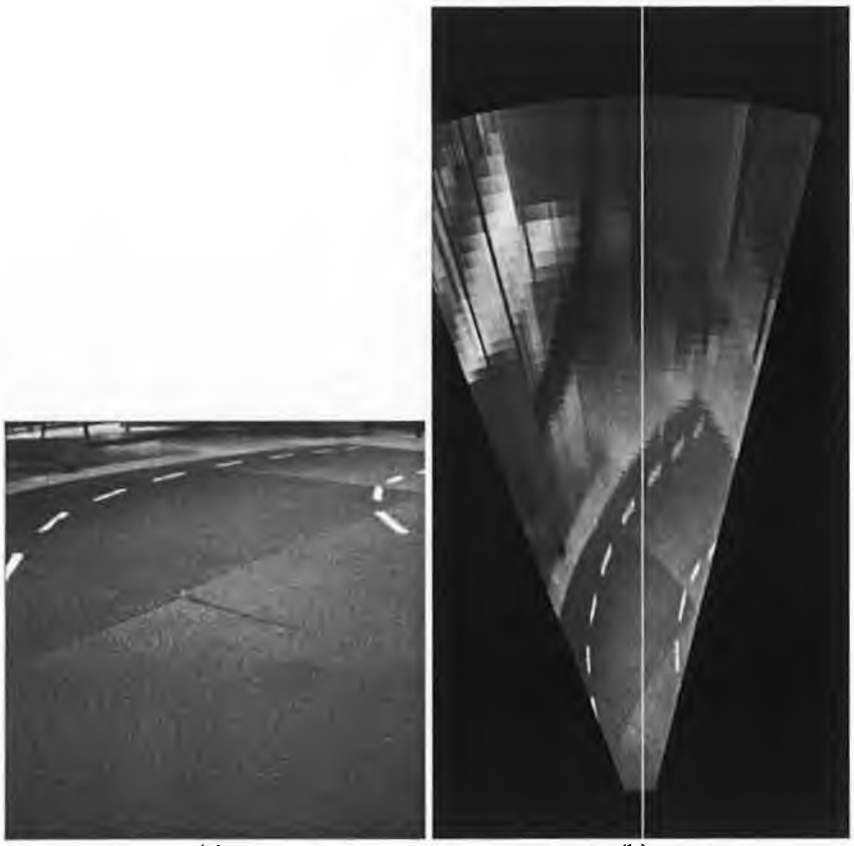

(a) (b)

Figure 37. Roadway section halfway through S-bend, (a) shows captured image and (b) shows image after Inverse Perspective Algorithm has been applied.

It is noted here that part of one set of lane markings is missing. In this case it is the right side lane markings. This is due to those lane markings not being in the field of view of the camera.

4.3 Detecting Lane markings

The next set of results examined are those of the Lane detection algorithm. Each of the road sections shown in the results of the inverse perspective mapping (see section 4.2) is analysed by this algorithm to see how successful it is at determining the lane ahead.

4.3.1 Finding all Objects in the Scene

The first task of this algorithm is to determine all the objects in the scene before they can then be categorised into lane markers and non-lane markers.

4.3.1.1 Straight Section of Roadway

The images before and after thresholding for the straight section of roadway are shown in Figure 38.

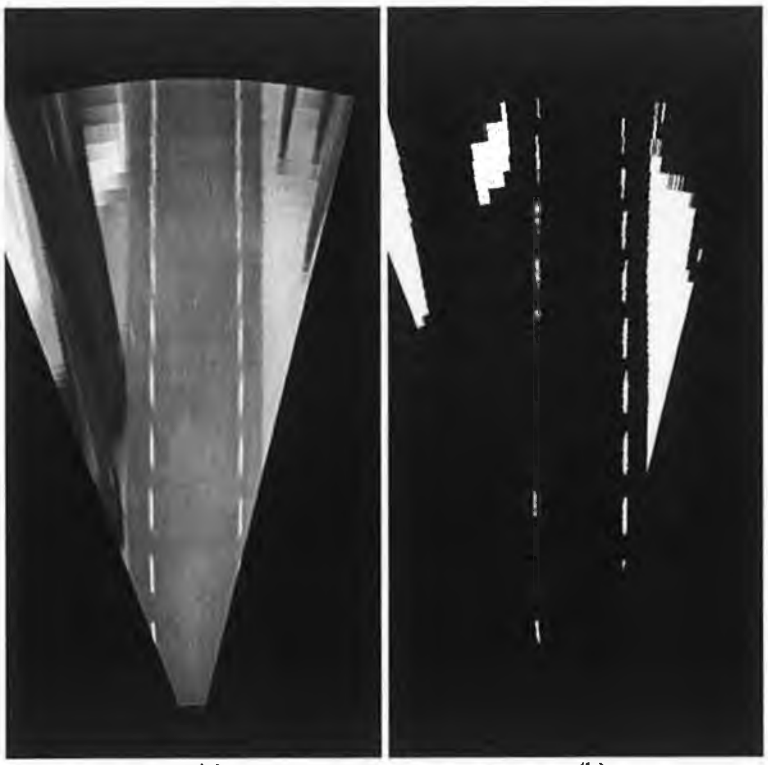


Figure 38. Threshold operation on straight road image. (a) shows image before thresholding and (b) shows image after thresholding.

The list of objects found in this image are as follows.

Object Number	Size in Pixels	Centre	Object Number	Size in Pixels	Centre
0	52	(133,158)	25	4	(597,323)
1	109	(194,156)	26	1	(600,324)
2	11	(204,249)	27	5	(607,311)
3	141	(255,250)	28	6	(612,312)
4	127	(263,156)	29	13	(618,160)
5	165	(321,251)	30	3	(617,313)
6	3	(305,274)	31	85	(642,159)
7	10064	(503,294)	32	34	(649,278)
8	117	(329,157)	33	8	(647,249)
9	161	(386,251)	34	7	(647,283)
10	118	(397,157)	35	7	(648,288)
11	148	(446,251)	36	72	(667,282)
12	152	(463,157)	37	7	(655,284)
13	4055	(554,18)	38	6	(654,289)
14	1	(461,42)	39	9	(657,250)
15	2	(463,41)	40	7	(664,105)
16	2	(466,40)	41	38	(671,249)
17	127	(505,250)	42	7	(671,104)
18	138	(520,158)	43	1	(675,103)
19	31	(534,329)	44	41	(686,291)
20	120	(562,250)	45	6	(680,282)
21	119	(577,158)	46	22	(691,160)
22	2582	(638,111)	47	15	(690,283)
23	4	(592,322)	48	8	(687,285)
24	86	(613,249)	49	6	(694,286)

Table 1. All the objects found in the straight section of roadway.

There are 50 objects in total found in this image. They range in sizes from 1 pixel to the largest at 10064 pixels. By inspection of Figure 38, 10 of these should be Left lane markers and 9 of them should be right lane markers. However, a number of issues could cause problems with this. A look at the lowest right lane marker shows that it is very small in size due to the fact that only a small amount of the marker is visible. In the list it is Object Number 2 and is only 11 pixels in size. Realistically there is not enough information in this lane marker to reliably inform the algorithm about the lane ahead. In particular, a least squares fit to a straight line of so few pixels may result in an incorrect assumption about the slope of the line of the lane marker. Also a closer look at some of the other lane markers reveals that some of them may be split into more than one object (see Figure 39).

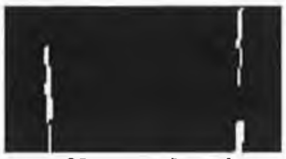


Figure 39. Close up of lane makers in straight road image.

In the Left-hand marker in this image, this marker will actually be made up of two objects. The lane marker is split into Object Numbers 29 and 31 from the list. The lane marker in the top left of Figure 39 above may appear to be two objects but is in fact three, as a further zoom in will show (see Figure 40) that this is not the case.

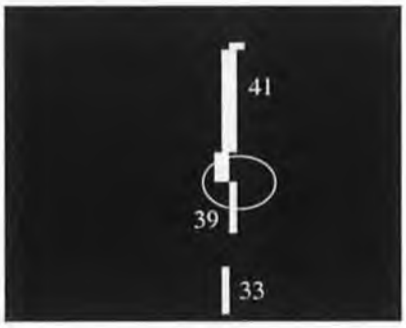


Figure 40. Relationship between objects 33, 39 and 41 in straight road image.

There is no near neighbour relationship between the pixels, as outlined in white, these are far neighbours. So these objects are in the list as Object numbers 33, 39 and 41.

4.3.1.2 Roadway with S-bend Ahead.

The example with the roadway section with S-bend ahead is now examined.

The images before and after thresholding are shown in Figure 41.

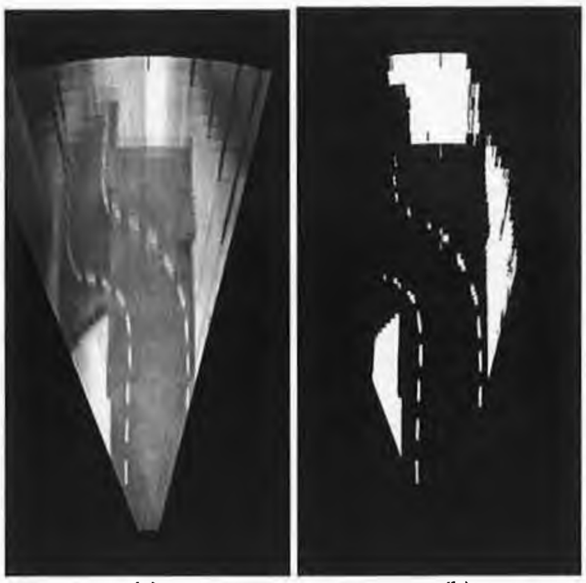


Figure 41. Threshold operation on Roadway with S-bend ahead. (a) shows image before threshold operation and (b) shows image after threshold operation.

The list of objects found in this image is as follows:

Object Number	Size in Pixels	Centre	Slope	Object Number	Size in Pixels	Centre	Slope
0	73	(146,170)	-0.015388	23	13	(419,125)	-0.091580
2	91	(205,171)	0.028406	24	125	(440,229)	-0.338209
3	78	(252,256)	0.014248	28	9	(464,202)	0.000000
4	90	(259,172)	0.019669	29	9	(475,204)	0.000000
5	1	(277,267)		30	63	(475,209)	0.000000
6	7927	(458,279)	0.003497	31	9	(484,201)	0.000000
7	1	(283,107)		32	9	(484,203)	0.000000
8	79	(302,256)	-0.017787	33	27	(492,186)	-0.059454
9	103	(310,175)	0.002228	34	31	(511,159)	-0.071208
11	2	(301,109)	0.000000	35	25	(532,142)	-0.106918
12	4730	(281,131)	-0.064519	36	22	(555,138)	-0.121145
14	101	(349,254)	-0.051849	38	4	(569,295)	0.000000
15	94	(357,173)	-0.111034	39	5	(578,141)	0.000000
16	2	(358,287)	0.000000	40	7	(586,140)	0.000000
18	123	(395,247)	-0.233326	45	8	(614,266)	0.000000
19	86	(392,162)	-0.437757	47	7	(675,149)	0.000000
21	17	(409,146)	-0.170735	50	1	(673,241)	
22	31	(414,140)	-0.147122	53	11996	(673,197)	-1.304891

Table 2. All of the objects found in the roadway section with s-bend ahead.

From the numbering it appears that there are 54 objects in this image but closer inspection reveals there are actually only 36 objects. Many of the object numbers between 0 and 53 do not appear. The reason for this is that in some cases one object has been assimilated by another object. As was described previously (see section 3.5.2) this leads to the original object being destroyed but the pixels of that object are now all linked into a larger object. One major concern evident from these results is in the curve. A close up is shown in Figure 42.

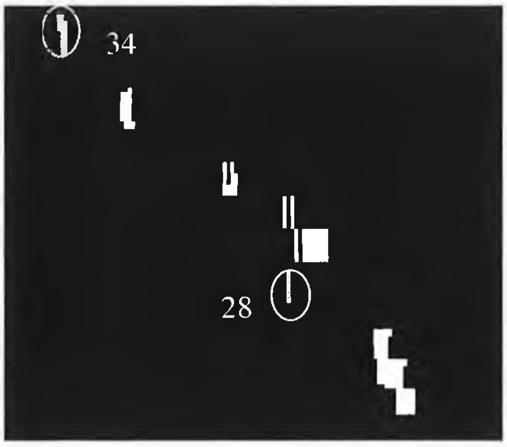


Figure 42. Close up of lane marker objects in curve.

While these do appear to follow the curve as a group, looking at each individually, there is no indication that the slope of the best-fit line through each object is approximate to the slope of the curve at that point. For example object number 34 has a slope of -0.071208 rads and object number 28 has a slope of 0 rads. 0 rads is vertical on the image or straight-ahead from the vehicle's point of view. The angle at this point in the curve would be expected to be somewhere between –0.5236 and –0.785 rads. This suggests that objects this far from the camera source cannot be relied upon to give an accurate indication of the lane markings.

4.3.1.3 Roadway Section Half way through S Bend.

A third example is now looked at. The images before and after thresholding from the algorithm are shown in Figure 43.

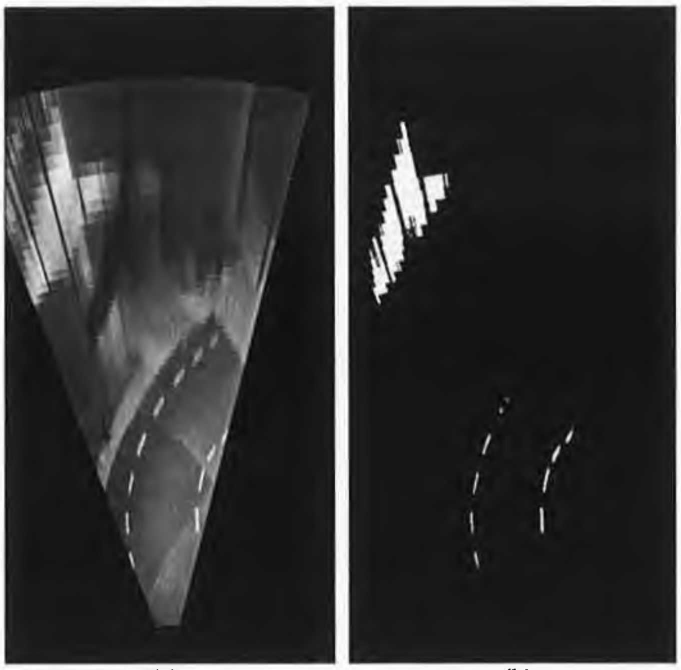

(a) (b)

Figure 43. Threshold operation on Roadway image halfway through S-bend. (a) shows image before threshold operation and () shows image after threshold operation.

The list of objects detected in this image are as follows

Object Number	Size in Pixels	Centre	Slope	Object Number	Size in Pixels	Centre	Slope
0	1	(115,159)		16	2	(460,33)	0.000000
1	3	(118,158)	0.000000	17	2	(463,32)	0.000000
2	26	(127,155)	-0.193261	18	3	(466,31)	0.000000
3	40	(172,150)	-0.041946	19	1886	(522,55)	-0.061549
4	83	(174,234)	-0.028891	20	1	(468,30)	
5	4	(209,153)	0.266252	21	2	(498,35)	0.000000
6	70	(223,239)	0.193318	22	2	(501,34)	0.000000
7	32	(221,155)	0.183409	23	2	(504,33)	0.000000
8	68	(255,251)	0.403270	24	2	(519,79)	0.000000
9	12	(258,164)	0.113792	25	4	(521,75)	0.000000
10	39	(270,169)	0.331142	26	4	(522,78)	0.000000
11	44	(276,267)	0.396676	29	4	(525,74)	0.000000
12	2	(285,272)	0.000000	31	3	(527,77)	0.000000
13	9	(308,185)	-0.165149	32	2776	(580,75)	-0.151943
14	4	(321,193)	0.000000	35	789	(576,106)	0.022789
15	830	(476,39)	-0.098154	36	42	(589,121)	-0.142147

Table 3. All the objects found in road-way image halfway through s-bend.

Again a few object numbers are missing as they have been assimilated into other objects. One thing to note here is the size of the lane markers in pixels appears to be a little smaller than in previous cases. In some of the cases there is no value for the slope, this occurs where the object has only one pixel. A slope cannot be calculated from a single point and therefore, the slope for such an object is undefined.

4.3.2 Criteria for Determining Lane Markers from Other Objects.

Once all the objects are inserted in a linked list, the first operation to be carried out is to remove any objects that are definitely not lane markers. These would include any objects that have very few pixels (e.g. less than 8) and those that have far too many pixels (e.g. more than 200). This eliminates many objects from the list, which will speed up searching at later stages.

4.3.3 Sorting Lane Markers into Right and Left Lanes

4.3.3.1 Finding First Lane Markers in Left and Right Lanes

How the algorithm goes about finding the first lane markers on the Left and Right side of the lane is discussed in section 3.5.4.1.

Running the algorithm to this point on the straight road image reveals the following lane markers as being the first Left and Right lane markers.

The attributes of the first Left and first Right lane markers are:

```
The First Left Lane Marker is number 0
Object number 0     Size is: 52     Centre (X, Y):(144,160)     slope: -0.078009
The First Right Lane marker is number 3
Object number 3     Size is: 141    Centre (X, Y):(265,252)     slope: 0.017131
```

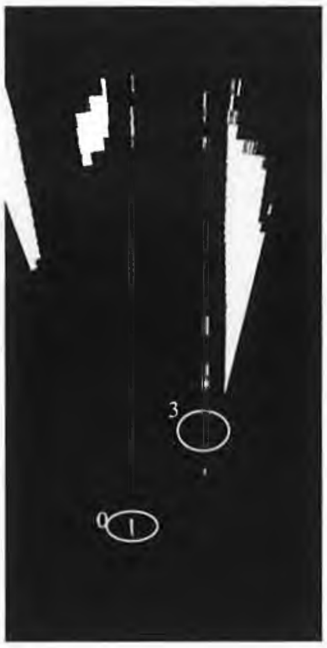


Figure 44. Finding the first lane markers on left and right sides of the lane.

Notice that just below object number 3 is object number 2. As it happens, this is actually a small section of the first Right lane marker that is visible to the camera but this object is only 10 pixels in size and such a small object couldn't reliably inform the algorithm about the lane position or direction. A look at the slope of object number 2 shows it is 0.161837 rads, which is considerably different to lane marker object number 3, proving this point.

4.3.3.2 Sorting the remainder of the Lane Markers into Right and Left Lanes

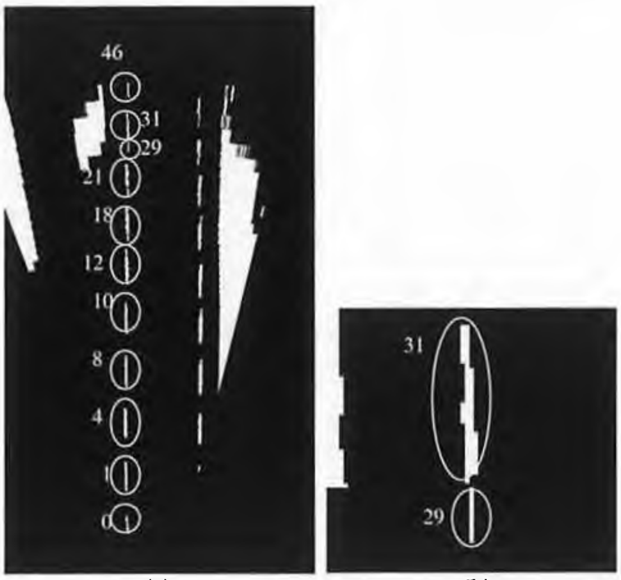
The results of sorting the remainder of the lane markers into right and left lanes following the criteria put forward in section 3.5.4.2 are now considered.

The results on the straight lane section are given below. The full list of objects in the image can be seen in Table 1.

The list of Left lane markers is as follows

Object Number	Size in Pixels	Centre	Slope
0	52	(133,158)	-0.088533
1	109	(194,156)	-0.020644
4	127	(263,156)	-0.014257
8	117	(329,157)	-0.016483
10	118	(397,157)	-0.030369
12	152	(463,157)	-0.030563
18	138	(520,158)	-0.039070
21	119	(577,158)	-0.046723
29	13	(618,160)	0.000000
31	85	(642,159)	-0.044251
46	22	(691,160)	-0.054826

Table 4. List of Left lane markers in straight section of roadway.



(a) (b)

Figure 45. Finding all the Left lane markers in the straight road image, (a) shows all the lane markers identified in the Left lane and (b) shows a close up of objects 29 and 31.

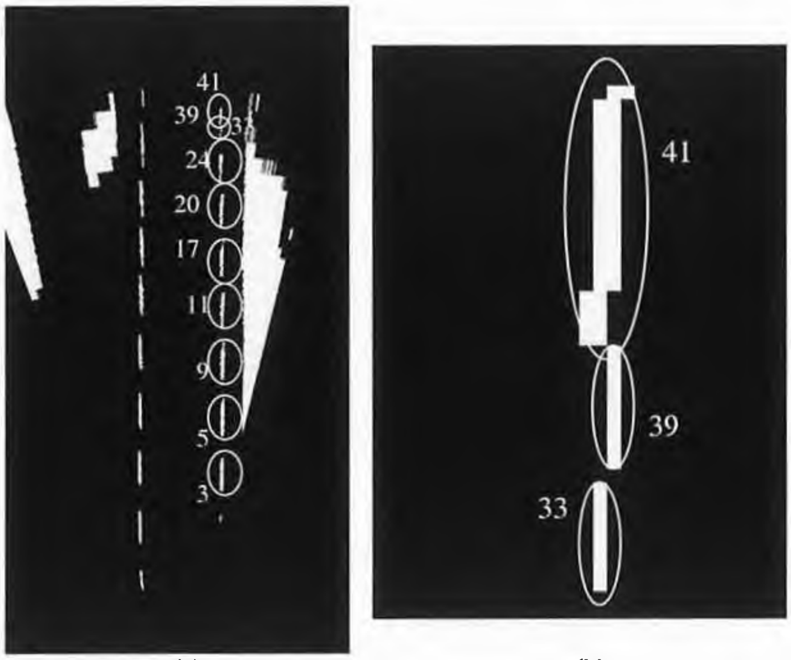
Of particular note here is the relationship between 29 and 31 (see Figure 45). It is likely that these were part of the one lane marker but became separated during the processing.

The search for Right lane markers takes the same form as that shown for the Left lane markers above.

The results are as follows.

Object Number	Size in Pixels	Centre	Slope
3	141	(255,250)	0.016587
5	165	(321,251)	0.020921
9	161	(386,251)	0.025997
11	148	(446,251)	0.032536
17	127	(505,250)	0.044690
20	120	(562,250)	0.052208
24	86	(613,249)	0.036544
33	8	(647,249)	0.000000
39	9	(657,250)	0.000000
41	38	(671,249)	0.068315

Table 5. List of Right lane markers in straight section of roadway.



(a) (b)

Figure 46. Finding all the Right lane makers in the straight road image (a) shows all the lane markers identified in the Right lane and (b) shows a close up of objects 33, 39 and 41.

Part (b) of Figure 46 shows a closer look at the situation for lane markers 33, 39 and 41. It may appear that object number 39 should be part of object number 41 but where the two meet is not a near neighbour relationship and so they are not considered to be the one object. In reality objects 33, 39 and 41 all come from the one lane marker on the original roadway.

4.3.4 Finding the Lane Markers on Roadway section with S-bend ahead.

Now the whole algorithm is run on a piece of curved roadway. The roadway image and results follow.

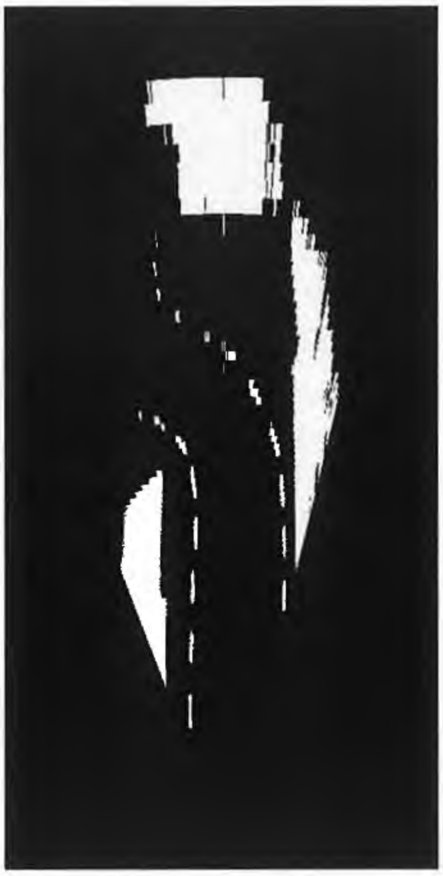


Figure 47. Roadway with S-bend ahead, after Threshold Operation.

The complete list of objects found in this section of roadway can be seen in Table 2.

The list of Left lane objects is as follows:

Object Number	Size in Pixels	Centre	Slope
0	73	(146,170)	-0.015388
2	91	(205,171)	0.028406
4	90	(259,172)	0.019669
9	103	(310,175)	0.002228
15	94	(357,173)	-0.111034
19	86	(392,162)	-0.437757

Table 6. List of Left lane markers in section of roadway with s-bend ahead.

The list of Right lane objects is as follows:

Object Number	Size in Pixels	Centre	Slope
3	78	(252,256)	0.014248
8	79	(302,256)	-0.017787
14	101	(349,254)	-0.051849
18	123	(395,247)	-0.233326
24	125	(440,229)	-0.338209
29	9	(475,204)	0.000000
32	9	(484,203)	0.000000

Table 7. List of Right lane markers in section of roadway with s-bend ahead.

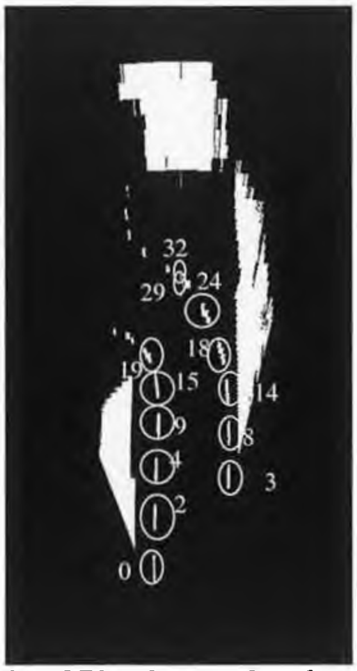


Figure 48. All the detected Left and Right lane markers found in the image of roadway with S-bend ahead.

The Left lane seems to stop short of where the human observer would expect. However, the next object above object 19, which is object 21 (not marked on image), just narrowly fails the criterion 4 (see section 3.5.4.2). Its value for this is 0.317347 rads and the limit is 0.3 rads. However, the 0.3 was an arbitrary value and could be modified if deemed beneficial. As mentioned in section 3.5.4.2, Criterion 4 could be determined more definitely if the maximum curvature of a given road is known. For the general case here it is just left at the arbitrary value 0.3 rads.

The Right lane seems to also stop short of where a human observer would expect. This is due to the fact mentioned earlier that the pixels at this point do not seem to group together properly to form the correct lines made up by the lane markers and so will not meet the criteria. Objects 29 and 32 meet the criteria though as can be seen from the results their slope is a lot different from what would be expected given the trend. This error is due to the fact that they have very few pixels (only 9 each) and should clearly be part of some larger object, which would give a slope with higher precision. Again through exhaustive testing it may be found necessary to deal with this problem by eliminating objects below a larger threshold than is currently being used. Currently this threshold is set to 8 pixels but this may need to be increased.

4.3.5 Finding the Lane Markers on Roadway section halfway through S-bend ahead.

The next curve examined is a right hand curve as the image was taken part way into the curve shown in the previous image. The camera module would have been placed some where between marker objects 15 and 18 of the previous image.

Results:

The complete list of objects found in the image can be seen in Table 3.

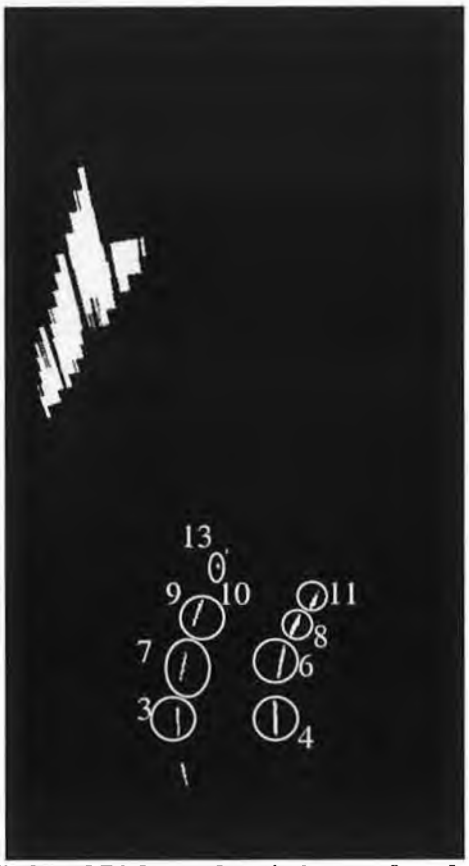


Figure 49. All the detected Left and Right markers in image of roadway halfway through S-bend.

The list of Left lane objects is as follows:

Object Number	Size in Pixels	Centre	Slope
3	40	(172,150)	-0.041946
7	32	(221,155)	0.183409
9	12	(258,164)	0.113792
10	39	(270,169)	0.331142
13	9	(308,185)	-0.165149

Table 8. List of Left lane markers in section of roadway half way through s-bend.

The list of Right lane objects is as follows:

Object Number	Size in Pixels	Centre	Slope
4	83	(174,234)	-0.028891
6	70	(223,239)	0.193318
8	68	(255,251)	0.403270
11	44	(276,267)	0.396676

Table 9. List of Right lane markers in section of roadway halfway through s-bend.

There are a number of points to note here.

Objects 9 and 10 are connected as far neighbours and appear to be one object but are actually two objects. They are circled as one in Figure 49.

Some of the markers in the Left lane have a very small number of pixels. Object 9 has 12 pixels and object 13 has 9 pixels. Having fewer pixels like this leads to unusual results in the slope of the object, which may be out of line with the trend of the other markers, which it is in this case. However, recognition of the objects making up the lane is all that is important here. The algorithm will work as long as the slope is not out of line with the others by enough to prevent the algorithm from correctly identifying the subsequent lane markers in the image.

It is also noted that the first Left lane marker is clearly not the first marker that a human observer would have picked out. A closer look will show that it doesn't meet the criteria set out for the first-in-lane markers of being over 30 pixels in size. It is in fact only 26 pixels. The 30-pixel value was picked as an arbitrary value and could be changed. However, changing this value could lead to objects with very few pixels being used as the first lane marker and thus making it difficult to determine the position and orientation of the vehicle in the lane.

It is also noted that when lane markers are at a more extreme angle with respect to the heading of the vehicle, as seen in this last example, that there is a greater tendency for lane markers to be split into different objects which may appear to the human observer to be one object. This relates to objects that touch via far neighbour pixels. The algorithm doesn't consider these to be touching, only near neighbour pixels can be acceptable criteria for connecting objects. This decision was made on the basis of processing power concerns as adding checks for a further two pixels during object creation would double the processing required to carry out the algorithm. However, as can be noted from the results these only happen in a few places and as such a mechanism to cycle through all the objects and see if any are far neighbours of each other and then amalgamate them into one larger object could be considered. A large amount of testing would be required to determine if this is necessary and worth while.

4.3.6 Determining a Set of Target Points to follow.

Following the scheme set out in section 3.5.5 the algorithm determines target points for the vehicle to follow. These points can be seen in Figure 50 at points where the curve does not actually hit the target.

4.3.7 Curve Following the Target Points.

A B-spline curve was used to follow the points generated. This is described in section 3.6.3 and results of tests are given in section 4.4.3.

As described in section 4.4.3 points before and after the first and last targets are often beneficial to having a complete curve that traverses all the target points. This is why the algorithm takes one point after there are no more lane markers to determine target points. This extra point, which is calculated to be in line with the last valid point is vital for allowing the curve to reach as far as the last valid point. Also in order for the curve to start on or before the first valid point, points must be placed before the first valid point. In the discussion of the algorithm above, a point was placed where the camera and vehicle front is assumed to be, i.e. (0, 200). In order for the curve to pass through this point, another point is assumed to be behind the vehicle but in line with the heading. In this case it was assumed to be (-50, 200). This allows a complete curve to be generated. The reasons for these are explained in greater detail in section 4.4.3.

Results for the three sections of road shown throughout this section are shown in Figure 50.

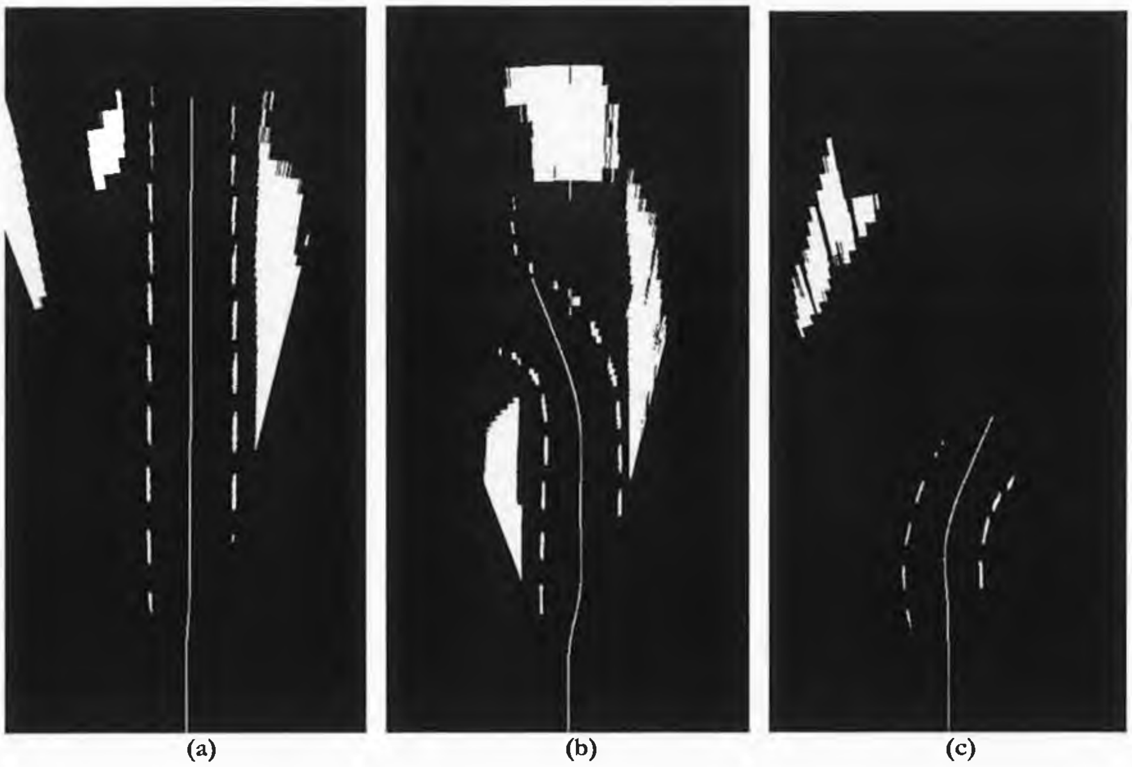


Figure 50. The trajectory curve calculated for the three roadway sections, (a) is the straight section of roadway, (b) is the section with S-bend ahead and (c) is the section halfway through the S-bend.

It is noted particularly in part (b) of Figure 50 that the trajectory seems to go wrong part way through the curve. A look back at Figure 48 and Table 7, shows that the markers on the Right side of the lane give the wrong impression of where the lane is going and the trajectory is following this. Also as stated previously an extra target will be set continuing in the same direction as the trajectory was previously going. This makes the trajectory error appear even worse. However, if the algorithm were to be run on a real vehicle system new images would be taken as the vehicle travels and this would mean that new points that would likely not have error would be added and thus remove the problem shown in Figure 50. Even if new target points are not added the trajectory line stops short of leaving the lane entirely.

4.3.8 Vehicle Speed Vs Algorithm Speed.

Now that a trajectory has been decided upon it should be considered how fast a vehicle should travel (assuming no slip) and still have enough information to follow the lane. In effect this means determining how far ahead of the vehicle the algorithm can reliably predict and how often new data will be introduced to the algorithm. Unfortunately there are unknowns in this. Much will depend on the type and speed of the hardware used, to determine how quickly the algorithm can be run and therefore, how many image frames can be dealt with each second. This determines how often new data is available to the control system. With this in mind the three stretches of roadway are considered at three different frame rates. Then taking three different vehicle speeds a table shows how many image frames should occur within the distance over which one image frame can reliably determine the lane. If less than one frame can occur in this space then this will show that the algorithm is not viable at this vehicle speed/frame rate combination. The more frames that occur in this space the more reliable and safe the system is considered to be.

In each of the three roadway sections only the x-axis movement will be considered as opposed to calculating the distance by following all the way around the curve. The x-axis movement will be the worst case scenario in any case. The point picked as the final point that can be reliably determined will be the centre point of the furthest lane marker. Given the vertical distance calculated in section 4.2.3.1 each pixel was determined to represent 2.86mm. Given the 1:10 scale this would mean in an ordinary road car each pixel would represent 28.6mm. So for example if the centre point of the lane marker was 300 pixels on the x-axis then this would be equivalent to 8580mm from the vehicle i.e. 8.58 metres from the vehicle.

The five frame rates used are 4fps, 5fps, 10fps, 15fps and 30fps.

The three speeds used are 50kph, 100kph and 130kph.

Straight Section of road way	50km/h	100km/h	130km/h
4fps	5.73	2.85	2.19
5fps	7.16	3.56	2.74
10fps	14.23	7.11	5.47
15fps	21.45	10.67	8.2
30fps	42.69	21.34	16.4
Section with s-bend ahead			
4fps	3.98	1.99	1.54
5fps	4.98	2.49	1.92
10fps	9.97	4.98	3.83
15fps	14.95	7.48	5.75
30fps	29.9	14.95	11.5
Section half way through s-bend			
4fps	2.54	1.26	0.976
5fps	3.17	1.58	1.22
10fps	6.34	3.17	2.44
15fps	9.5	4.76	3.66
30fps	19	9.5	7.32

Table 10. Frame rates Vs Vehicle speed

What does the table above mean for the algorithm used in this project? Looking at the lowest number in the table i.e. 0.976, this value occurs at 130 km/h using 4fps on the section of roadway halfway through the s-bend. This means that the algorithm is able to see 0.976 image frames into the distance if it can operate at 4fps and the vehicle is moving at 130km/h. In other words if the vehicle is travelling at 130 km/h in this section it will have travelled completely the trajectory that it has calculated and then have to stop and wait for the next image frame to be analysed before it can move on. However, if the algorithm is capable of operating at 5fps then the figure is 1.22 which means that the vehicle will have travelled (1/1.22=0.82) 82% of the trajectory it has calculated from the previous image frame before the algorithm supplies it with new information. 5fps would mean that the algorithm must be complete within 200ms.

Note: In driving 82% of the calculated trajectory it is assumed that the trajectory is correct and obstacle free, as this algorithm is not capable of recognising obstacles in any event. The greater the number in the table above the less the effect of trajectory miscalculation, as any miscalculation should be corrected in the next frame and the vehicle will only have travelled a short distance in the intervening period of time, e.g. in the Straight road section if driving at 50km/h. using 30fps the vehicle will only have travelled (1/42.69=0.0234) 2.3% of the calculated trajectory before the next data is available.

If the algorithm could be improved to see further into the distance then this would mean lower frame rates could be tolerated. Of course improving the algorithm might lead to more processing that will take up more of the extra processing time that has been gained.

These figures will be very useful in determining if a piece of hardware is capable of carrying out the algorithm at a particular vehicle speed.

4.4 *Path Trajectory and Steering Algorithm.*

4.4.1 Cubic Curve tests.

The following two graphics show the results based on different (x_1, y_1) and θ_1 values. In each case (x_0, y_0) was (0,0) and θ_0 was also 0. The *v* value used in each case was 0.01.

Example 1

(x_1,y_1) is (16,5); θ_1 is 0.8 rads.

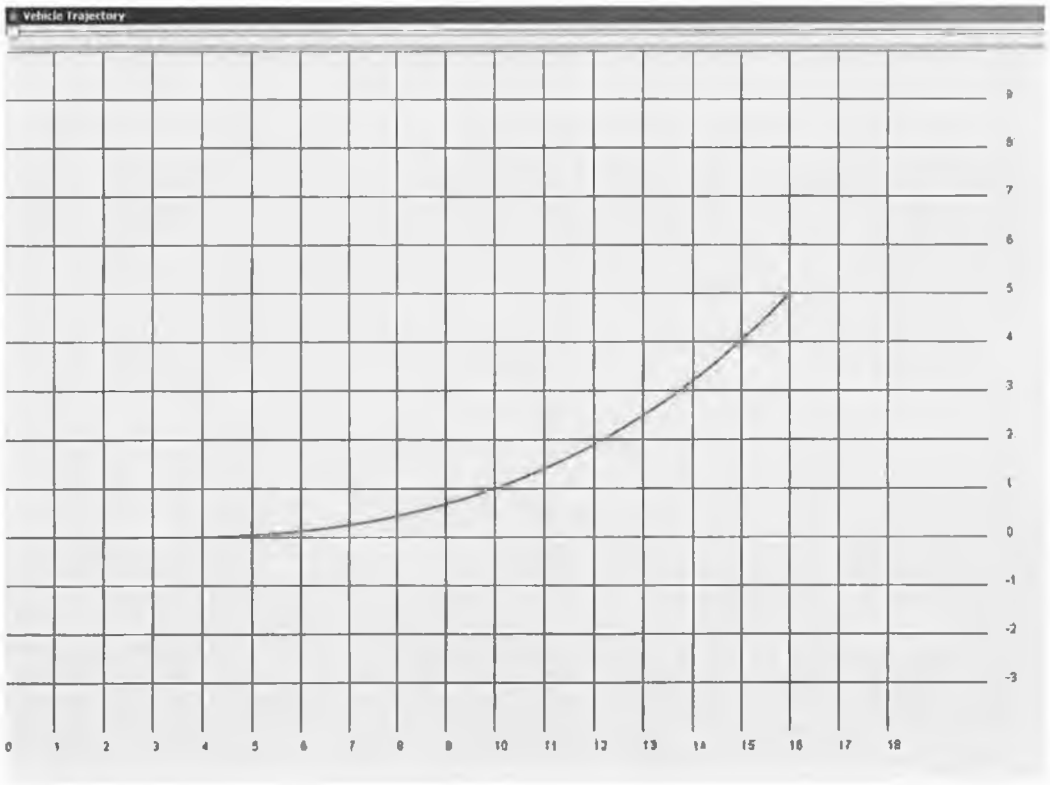


Figure 51. Curve for Example 1, (x_1, y_1) is (16, 5) and θ_1 is 0.8 rads, *v* is 0.01.

As can be seen from example 1 (see Figure 51) quite a gentle curve results. From the point of view of a vehicle, the steering change would be gradual and all changes would be in the one direction.

Example 2

(x_1,y_1) is (12,8); θ_1 is 1.3 rads.

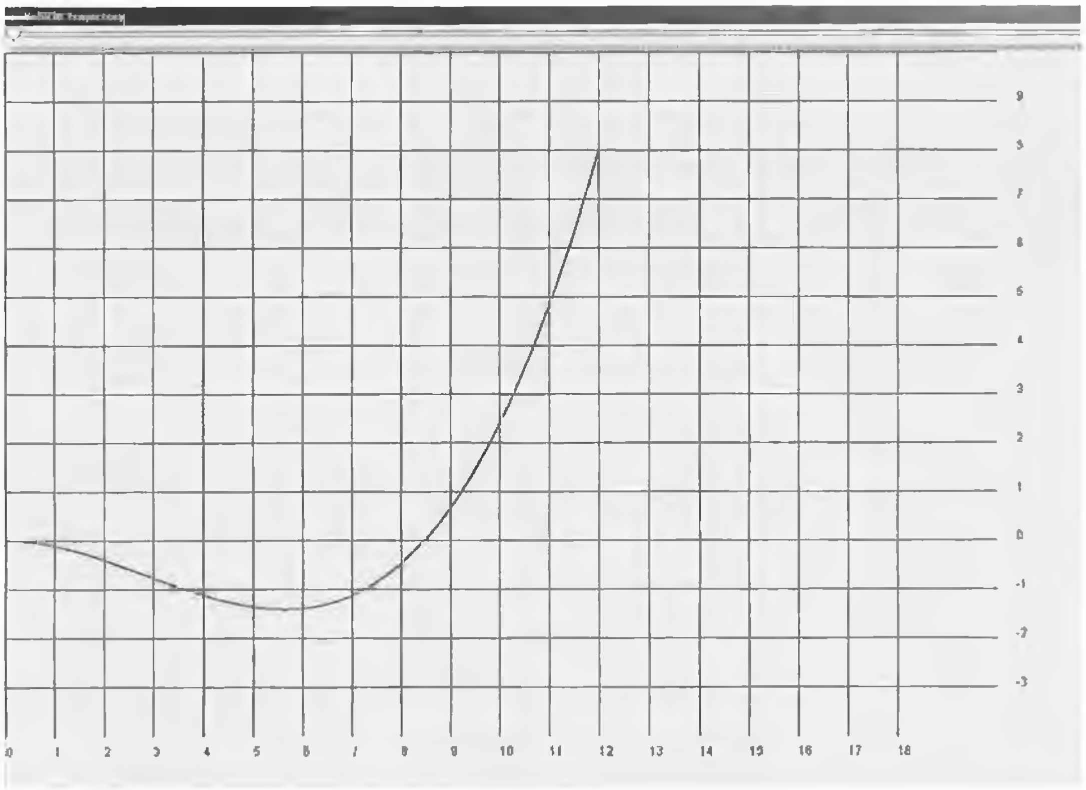


Figure 52. Curve for Example 2, (x_1, y_1) is (12, 8) and θ_1 is 1.3 rads, v is 0.01.

Example 2 (see Figure 52) shows a much more extreme curve. In order to have the final θ value of 1.3 in such a short space the curve must first go down before turning back up. From a vehicle point of view this would mean steering to the right before quickly changing the steering to the left.

A third example is now shown (see Figure 53) which is the same as Example 2 except that the *v* value has changed to 2 rather than 0.01. This is purposely chosen as unacceptably high to show the effect of this variable on the precision of the curve estimation.

Example 3.

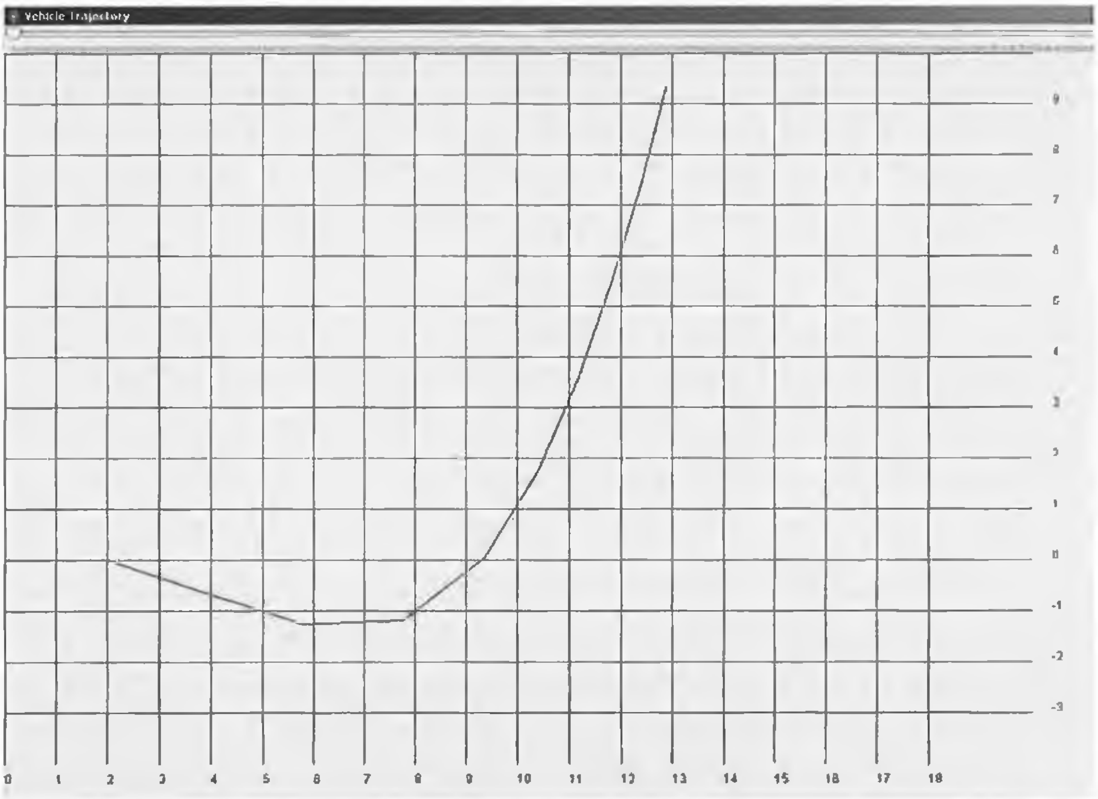


Figure 53. Curve for Example 3, (x_1, y_1) is (12, 8) and θ_1 is 1.3 rads, *v* is 2.

Take particular note that the curve should end as close as possible to the (x_1, y_1) position. In this case this should be (12,8) but because of the lack of precision the end point is approximately (12.88, 9.35) which is a long way from the expected value. For comparison in Example 2 the end values are (12.004, 8.005).

4.4.2 Effect of Wheelbase on Steering Angle

Two examples now follow, which will show the effect of wheelbase on steering angle. Both examples will use the curve that starts at (0,0) with a θ of 0 and ends at (12,8) and θ_1 of 1.3 rads.

Example 1 (see Figure 54)

In this case a wheelbase of 2 is used. The vehicle is pictured near the point on the curve where the radius is the smallest, i.e. largest curvature, so that the steering angle is most pronounced.

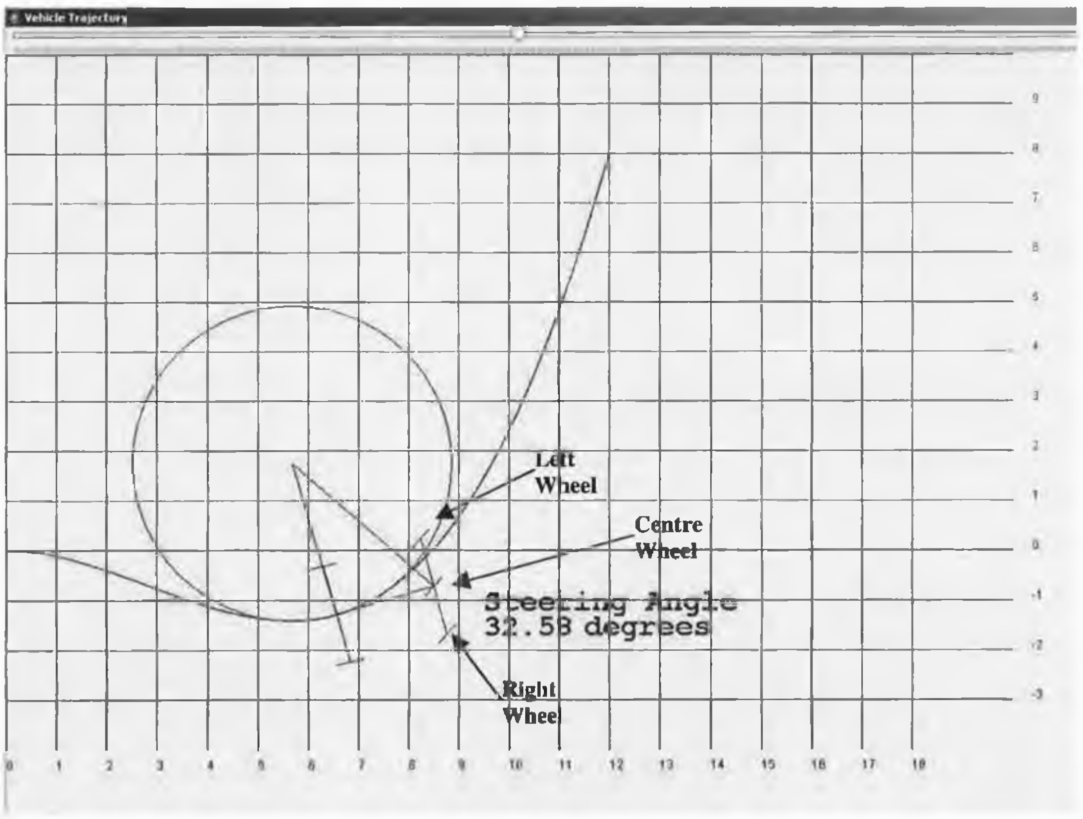


Figure 54. Steering angle using wheelbase value of 2.

In the diagram it can be seen that the car has three front wheels. The left and right wheels are there to give the effect of an actual automobile. The centre wheel shows the average steering angle. The left and right wheel show the exact same angle though on a real vehicle of this type these wheels would be at different angles. The reason for this is that each wheel is traversing a different trajectory. If they were at the same angle this would lead to excessive tyre slip and wear as well as difficult handling. However, these differing angles will be sorted out by mechanical elements in the car so from the point of view of this project the average angle is all that is important. By moving the slider at the top of the window above the vehicle can traverse the curve and the changing steering angle can be viewed as the vehicle moves along the curve.

Example 2 shows the same curve but this time being traversed by a vehicle with a wheelbase of 5. Again it is pictured at the point along the curve where the curvature is the greatest.

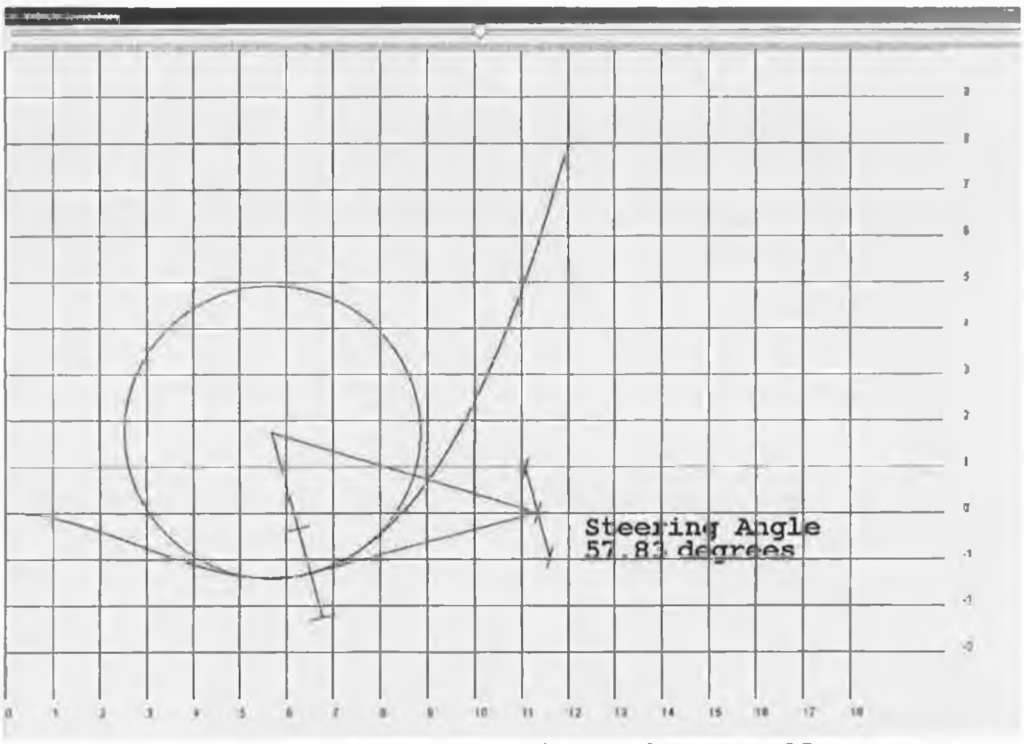


Figure 55. Steering angle using wheelbase value of 5.

Note here that the radius of curvature is the same as in the last example, this should not be a surprise as the curve itself has not changed and again the vehicle is positioned at the point of greatest curvature. However, the wheelbase of the vehicle is different from the last example and so in order to keep the vehicle on this trajectory the steering angle must be much greater than in the previous case. This would be a limiting factor in the maximum curvature that a vehicle could traverse given a maximum steering angle that a given vehicle would be capable of.

4.4.3 B-spline tests.

To show this algorithm, again Java has been used, given its powerful graphics capabilities. In the example that follows ten target points have been chosen as follows

(0,0) (1,1) (2,4) (3,4) (5,4) (7,4) (9,2) (11, -2) (12, -1) (14, -2)

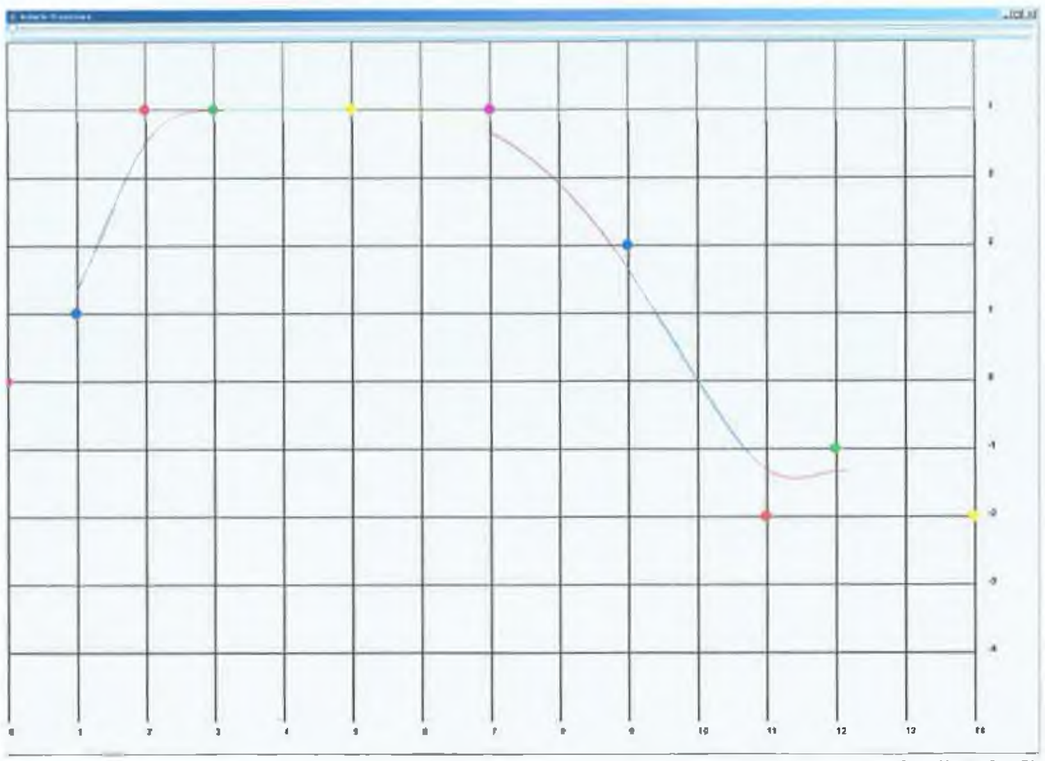


Figure 56. B-spline using points (0,0) (1,1) (2,4) (3,4) (5,4) (7,4) (9,2) (11, -2) (12, -1) (14, -2)

There are a number of important things to note regarding Figure 56.

The coloured dots represent the ten different points listed above. The curve also changes colour across its length, the colour of the segment at any point indicates which coloured dot/target point is the current local control point, i.e. the point P_i mentioned in the general matrix form previously (see section 3.6.3.1).

The curve doesn't extend from the first dot to the last dot. It cannot properly extend from the first dot (Magenta), as when P_i is the first dot then P_{i-1} does not exist and therefore, the matrix calculation cannot be carried out or could at best be carried out on incomplete data. A solution to this will be discussed in a moment. This is also the case for the final two dots (Green and Yellow) as when P_i is the second last dot then P_{i+2} doesn't exist and when P_i is the final dot neither P_{i+1} nor P_{i+2} exist. So again the matrix calculation cannot be carried out in its entirety. However, in this project this should only happen if and when the vehicle runs out of road and it is likely to be desirable that the vehicle stops short of this point in any case.

Another important point to note here also is that the B-spline does not always pass through the target point but instead passes close to it. This is not necessarily a problem in this application. If the target point is always the centre of a lane, a human driver will regularly deviate from the centre of a lane while travelling through bends but will tend to stay to the centre of a lane while travelling in a straight line. In the diagram the first red, green, yellow and blue dots are in a straight line. For the middle two of these the curve hits both target points similar to a vehicle travelling in a straight line.

As can be noted from the second red dot the curve misses this target by a considerable distance. Would this be a large enough difference in this application to cause the vehicle to leave the intended lane? The answer to this question depends on a number of factors. Firstly it depends on the relative size of the vehicle compared to the scale of the curve. It also depends on the relative distance between the points compared to the size of the vehicle. In any case roads are designed for vehicles to travel around the lanes safely. Therefore, these problems can be solved by increasing the number of target points and by virtue of this, having the target points close to each other relative to the size of the vehicle. The overall number of target points can always be traded off against the resolution between each point, the incremental value of *u* as mentioned previously, to gain a curve that comes sufficiently close to each target point to maintain position of the vehicle in its own lane. On the subject of the incremental value of *u*, it should also be noted that the different colour segments of the curve in the diagram are not of uniform length. The main problem that this causes is that the incremental value of *u* cannot be used to gauge distance travelled. Remember that the incremental value of *u* is a fractional value between 0 and 1 and not a measure of distance as was *v* in the target point following algorithm discussed earlier in section 4.4.1 and [6]. The reason that this is an issue is that controlling a robotic vehicle to follow a lane not only requires steering output but also drive output. *u* cannot be used to inform the vehicle control system how far to move forward for each incremental change. However, this can be solved as follows. The curve as generated is an approximation to a curve made up of short straight-line segments. Due to the problem mentioned above however, these are not of uniform length. However, each individual length can be obtained by co-ordinate geometry since each of the individual end points of these lines are calculated by the matrix calculation. Each of these lengths can then be used to determine how far to move the car forward for each incremental change in *u*.

This is also how the steering angle is obtained for each point along the curve. Each straight-line segment has a slope. This is effectively the slope of the tangent to the approximated curve at that point. Likewise the change in slope between two points divided by the length of the straight line segment between those two points gives us the curvature at that point on the approximated curve. From this the radius of the curve at that point can be determined. Then given the wheelbase of the vehicle the appropriate steering angle can be obtained. Earlier it was mentioned that the curve doesn't start at the first point for stated reasons. This raises the question of how the vehicle should go about moving towards where the curve does start, in order to then traverse along it. As there is no information to guide the vehicle between its current position and where the curve starts this would be impossible given the situation as it stands. One solution is to have the first point somewhere behind the vehicle and in this way it can be used in calculations but does not need to be an actual target point for the car.

An example follows (see Figure 57). Assume the vehicle is on the second (blue) target point at position (1,0).

The points used in this example are

(0,0) (1,0) (2,1) (3,2) (5,4) (7,4) (9,2) (11, -2) (12, -1) (14, -2)

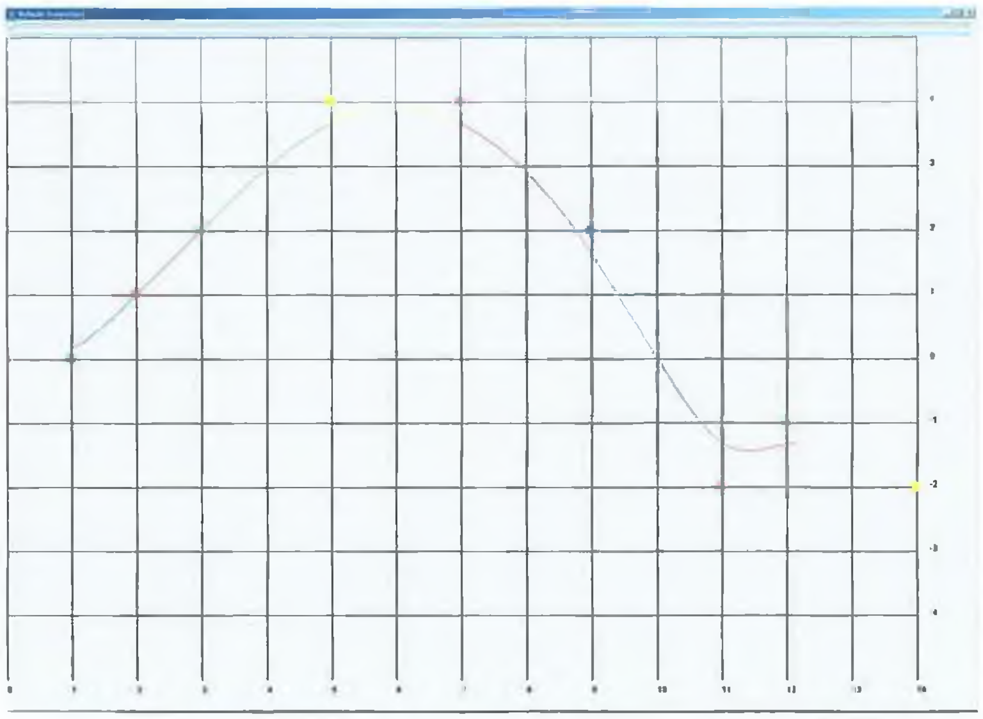


Figure 57. B-spline using points (0,0) (1,0) (2,1) (3,2) (5,4) (7,4) (9,2) (11, -2) (12, -1) (14, -2)

The result obtained here is certainly better than in the previous case but is still problematic. The blue line doesn't actually connect with the blue target point. The other big issue is that unless by luck the heading of the car just happened to coincide with the starting slope of the blue section of the curve then the vehicle will be pointed in the wrong direction. Again this leaves the situation that there is no information available to get the vehicle from its start point to the curve and so a further solution must be found.

The more complete solution is as follows. Firstly assume the first target point to be directly behind the vehicle in line with the heading of the vehicle. Secondly assume the third target point to be directly in front of the vehicle again in line with the heading of the vehicle. Since the co-ordinate space that will be used will be obtained from the first frame of video from the vehicle, it is fair to assume that the vehicle start point is at (1,0) and its heading is $\theta = 0$ rads. Although the mounted camera cannot see behind the vehicle, it can be assumed that the first point is directly behind the vehicle, for example (0,0).

No matter what the information from the camera tells us about direction to travel it should be assumed that the first target point is directly ahead, meaning that the vehicle's first movements must always be straight ahead. In the example this will be given as (2,0). After this the target points obtained from the lane recognition algorithm should follow, the first seven of which will be given in the example as

(3,1) (5,4) (7,4) (9,2) (11, -2) (12, -1) (14, -2)

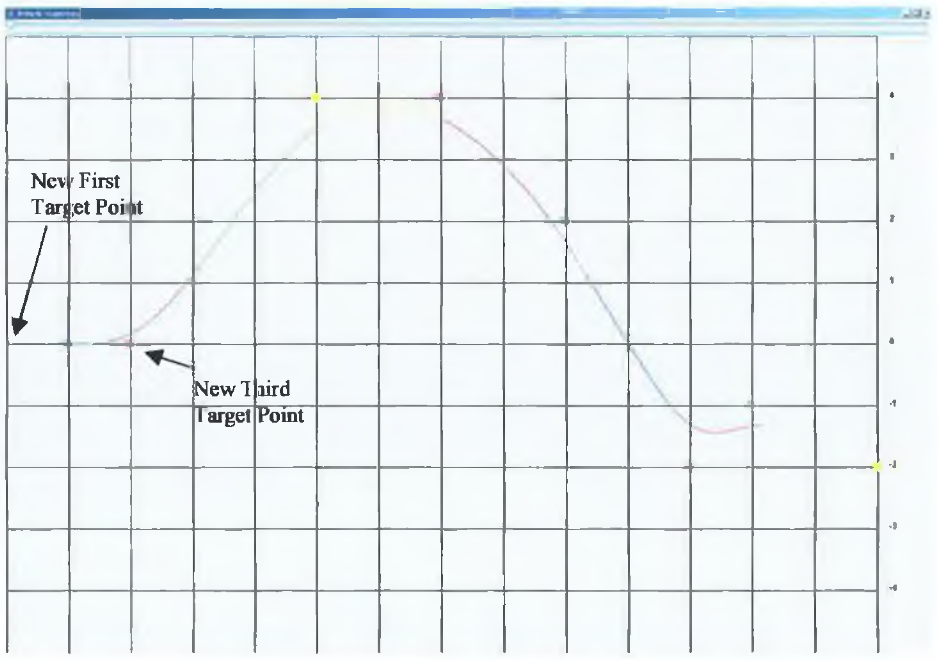


Figure 58. B-spline using points (0,0) (1,0) (2,0) (3,1) (5,4) (7,4) (9,2) (11, -2) (12, -1) (14, -2)

The results are much better in this case. The assumptions were that the vehicle is initially positioned on the second point (Blue dot) at (1,0). The first point (Magenta) at (0,0) is behind the vehicle. The third point (Red) at (2,0) is directly in front of the vehicle on the same heading. Most importantly the blue section of curve extends right to the blue starting location and the first section of the blue curve has the same slope as the heading of the vehicle, which means that the vehicle has enough information now to traverse the whole curve. It should be noted of course that the curve doesn't go all the way to the last target point. However, this is not as much of a problem as at the beginning of the curve. It is not important that the vehicle makes it all the way to the final target point for two reasons. Firstly in most cases as the vehicle moves new images will be captured by the camera, which will lead to new target points being generated and the curve will then continue on without any discontinuity. Secondly if no new images or target points are generated it is time for the vehicle to stop moving and it would be prudent for the vehicle to stop short of the final target point for safety reasons.

5 Discussion

5.1 Test Bed Model Vehicle

The test bed model vehicle was not focused on during this project. Nonetheless it is considered very important. The learning and development curve required to work on the sort of algorithms discussed in this project is so significant that it becomes difficult to concentrate on any new or novel work. If each new job of work involved in this area required the set-up of a new development system then very little advance would be made. Having a ready made test model vehicle and development environment that can be used across various algorithm types would be very beneficial to advancing novel algorithms in this area.

A number of issues are still outstanding, which would need to be worked on to get the model vehicle system operational.

5.1.1 Mechanical Issues.

The main board and camera module as well as some ancillary power electronics for the drive motors would need to be mechanically attached to the vehicle.

The main board shouldn't cause to many problems but interference from the large DC drive motor could be problematic and requires some Electro-magnetic shielding. The other issue with the main board is its power supply. Supplying power to the board from the same battery as that used to power the main drive motor could lead to serious noise problems for the main board. Hence separate battery sources for each circuit would be advisable. Again mechanically fixing these batteries to the vehicle will have to be considered particularly in light of the fact that the batteries would need to be recharged regularly.

The Camera module would need to be mechanically attached to the vehicle. Position and tilt angles would need to be reasonably accurate in order to be useful. As explained in section 4.2.2, many of these variables can be compensated for in software. However, the degree of software compensation is limited and cannot be carried out dynamically. Therefore, the camera module would have to be attached in a way that doesn't allow it to vary with the movements of the vehicle.

5.1.2 Electronics and Software issues.

The main board would need to be configured to operate applications on start up rather than wait for command line input to initiate. This is due to the fact that when the vehicle is to be used it must be stand alone and not attached to a host PC.

Test applications should be written with a time delay before start or a user input (button) before start. This is necessary so that the test personnel can have the circumstances for the test in place before the vehicle tries to react to its circumstances.

Currently the port of uClinux for the Blackfin processor doesn't have many of the common mathematical functions in its compiler libraries. While many of these problems can be worked around, for use as a general test bed a library of these would need to be developed.

5.2 Inverse Perspective Mapping.

Using some form or inverse perspective mapping seems a logical step in any lane detection and following algorithm. While many algorithms may not do it explicitly, at some point when moving from the captured image to determining lane following controls, some form of mapping must be done. However, a number of issues must be consider when using the inverse perspective mapping as part of an algorithm.

5.2.1 Processing and Memory Overheads.

The algorithm uses a considerable amount of complex mathematical operations.

Consider the case in this project where it was acting on each of 800 x 400 pixels i.e. 320,000 pixels in all. This means that all of the operations involved in the inverse perspective mapping must be carried out 320,000 times for each image frame. This takes a large chunk of any system's processing power. Those with fast maths processing instructions will of course do so quicker. As mentioned in [14], SIMD would be particularly suited to this sort of operation. Can this processing requirement be lightened? Yes, but not without compromise in another area. Looking closely at the inverse perspective mapping shows that the exact same operations and results are obtained each time the algorithm runs unless one of the variables in section 2.5.2.1 change in value. It would be expected in most set-ups that these would not change once calibration was finished. For this reason the mapping from captured image to re-mapped image could be calculated in advance and stored in a lookup table. This would mean that for each of the 320,000 pixels in each frame, the algorithm would simply look up the appropriate value in the lookup table that would tell the algorithm the appropriate pixel value to take from the captured image. All of these operations are reduced to indexing and assignment instructions. All microprocessors tend to be fast at executing these instructions. However, the downside of this is the amount of ROM required to store a lookup table of 320,000 integer values. At 2 bytes minimum this would take up 640kbytes. In fact it would take up twice that as a number would be required for both u and v values (the horizontal and vertical co-ordinates of the captured image). Altogether this is well over 1 megabyte in size. Taking the BF533 board as an example, there are only 4 Megabytes of flash ROM available and this must also store the uClinux operating system image. One other solution is to store the table in RAM, which may be more plentiful (BF533 has 128 Megabytes of RAM for example). However, this requires that the table be calculated at some point e.g. start up initialisation. One problem related to this, which was found on the BF533 board was that there were no maths functions such as *atan* in the compiler libraries for the uClinux port for the Blackfin processor, which would make the calculation of the table a non-trivial matter.

5.2.2 Using the results of the Inverse Perspective Mapping for Measurement.

As can be seen from the results the inverse perspective mapping was by no means perfect when it came to allowing the algorithm measure distances etc. from the results of the mapping. Actual horizontal and vertical distances differed from calculated values. More alarming still was the fact that when the results of the roadway image with the s-bend ahead were studied (see section 4.2.4), it was apparent that horizontal distances, which were the same in the real world scene, didn't appear the same in the re-mapped image. It is not clear whether this was the fault of the inverse perspective mapping in any way and it is assumed that this was in some part, if not solely, due to lens distortion. It would seem that any system that was to rely on inverse perspective mapping should probably include some form of lens distortion compensation.

Although not mentioned in [14], it is assumed that there are more variables that can affect the mapping. Of particular note is yaw angle. If the camera was to rotate around its optical axis this could upset the output of the algorithm considerably. While it would be unlikely any system designer would wish to introduce this rotation intentionally it could come about through imperfections in the mounting process.

5.2.3 Calibration.

Some robust, reliable and repeatable mechanism would be required to calibrate any system that was to rely on inverse perspective mapping. Be it through mechanical or software calibration, it would be necessary to compensate for imperfections in the mounting process both of the camera module to the vehicle and the lens to the camera module.

If some part of the vehicle were in the view of the camera module this calibration could be achieved based on recognisable markings on the parts of the vehicle that are in the view of the camera.

5.3 Lane Recognition Algorithm

For the small set of test images used this seemed to work reasonably well but to work in a large set of varying circumstances it is believed that this algorithm must become much more robust.

5.3.1 Lane Marker Objects with Insufficient Information.

Firstly in many cases, lane marker objects were too small to give accurate information, particularly regarding direction. Many lane markers were split into multiple objects. If these could be combined back into one object they could become more useful and reliable. Many were only split by virtue of the fact that they had no near neighbour relationship between them but did have a far neighbour relationship. Consideration of including far neighbour relationships could make a big improvement to results. This could perhaps be carried out after objects have already been created as they are currently. This could lead to this functionality being added without the need for excessive extra processing. In other cases the individual segments of a split lane marker were only split by a few pixels. Some form of region growing algorithm [27] could achieve the rejoining of these in just a few iterations. If these operations suited particular target hardware (e.g. SIMD hardware) they should be considered.

5.3.2 Priority Based Marker Recognition System

Currently the algorithm works on a priority-based system where each object is considered on merits based on particular criteria set out and marks awarded for this. This part of the algorithm in its present state was not very successful. While it did find most of the obvious markers it failed to consider many further up the image (i.e. further from the camera), which the human observer could see quite clearly. Part of the problem here is the lack of sophistication and limited amount of criteria set out. It was found that during the testing of the algorithms that to get the best results, any object that didn't attain full marks from each of the criteria was not an appropriate marker. Therefore, setting different priority marks to the different criteria proved unnecessary. However, this would seem to be more of a criticism of the number and type of criteria used to determine an appropriate marker, than of the priority based system itself. The major improvement that could be made here is to consider more general trends in the lane markers and include marks based on this. With the algorithm in its current guise, only the most recently selected marker is used in determining the next appropriate marker. This could lead to one non-marker that accidentally meets the criteria to cause the lane to carry on in the wrong direction. Also it seems particularly from section 4.3.4 that while the curve here is clearly visible to the human observer because of the trend of the markers, it is invisible to the current algorithm because of the individual attributes of the marker objects. Looking at the general trends of markers should also be considered. Therefore, improving the priority based system with points for markers that seem to fit a general trend could justify the use of the priority marks system.

6 Conclusions

It would seem from research into this area that a fully robust system for lane recognition and following, which can deal with all the road types that human drivers can deal with, is still a long way from realisation.

The work in this project was based on many constraints and assumptions, such as the road being perfectly flat and lane widths and lane markings being a definite pre-determined size. Any robust system for real-world use could not be based on such strict constraints but instead must be capable of dealing with a wide range of changing circumstances in the same way as a human driver. It would need to deal with varying types and sizes of road markings and roads that do not have lane markings at all. It must be capable of dealing with many types of road surfaces in varying weather and lighting conditions and real world roads cannot be assumed to be flat.

The literature review showed a large number of different approaches to solving the subsets of this task but always under particular constraints and assumptions. It is likely in the future that the optimum solution to any of these tasks will change as hardware advances in sophistication.

The inverse perspective mapping would seem to be a vital part of any algorithm in this area though in its current guise it is incomplete and will need other variables and mappings such as lens distortion corrections to be added to make it more useful.

Calibration methods will be very important to the accuracy of the inverse perspective mapping algorithm. It is likely that some form of measurement system to gauge this accuracy will be necessary in the future.

The issue of how the inverse perspective map is generated is one that will have to be tailored to the strengths and weaknesses of the particular hardware platform.

The lane detection algorithm used in this project was based on the system used in [7] and discussed in section 2.5.1.2. However, in that system the inverse perspective mapping was not used prior to lane detection and also the specifics of how the algorithm was carried out by the author are not published.

The algorithm as developed in this project was successful in determining the lane correctly for the test lanes used, though only to a certain distance in front of the vehicle. It is believed that this distance from the vehicle could be increased by improving the sophistication of the algorithm, to include methods to rejoin lane markings that have become split into different segments and also to include methods of looking at the general trend of the lane.

The method of path trajectory planning worked very well and properly determined the appropriate route through the lane ahead. However, the next phase in the development of this algorithm must include the matching of data from several image frames which must all be incorporated into the one general co-ordinate system. For this to be fully appraised it must be tested on a moving vehicle that is following the control signals set out by the trajectory planning algorithm. This is important as it is likely that the next generation of this trajectory planning algorithm will be required to take into account the movements of the vehicle in the intervening time since the last image frame in order to estimate the vehicle's current position.

The use of the B-spline for trajectory planning was successful and was applicable to the strengths of most DSPs, which are likely to be heavily used in vision system applications.

While it is likely that any robust lane detection and following algorithm will need to incorporate many different types of algorithms perhaps running side by side, there is still a need to compare individual algorithms against each other. This will only be possible with a standard model system on which many different algorithms can be tested in a safe, compact and controlled environment. With this in mind it is believed that the most important future work in this area is towards the development of a model vehicle and road system, which can be used to test the effectiveness of different algorithms.

The model system considered in this project would work well for this, given that it uses reasonably inexpensive off the shelf components and open source software. However, a certain amount of mechanical build would still be necessary and on the software side some standard libraries, such as common maths functions, will need to be developed as these may be necessary to enable the speedy generation of new or improved algorithms.

7 Bibliography

7.1 References

[1] Margie Peden et al, "World report on road traffic injury prevention", World Health Organisation Geneva 2004. ISBN 92 4 159131 5.

[2] Popular Science Magazine, May 2007.

[3] Tsugawa, S, "Vision-based vehicles in Japan: machine vision systems and driving control systems", Industrial Electronics, IEEE Transactions on, Volume 41, Issue 4, Aug. 1994 Page(s):398 – 405.

[4] Clarkson, Douglas, "Driving into the future", Electronics Today International November 1995.

[5] Dickmanns, E.D., "The Development of machine Vision for road Vehicles in the last decade.", IEEE Proc. Intelligent Vehicle Symposium 2002 14-17 June 2004 Page(s):54 - 59

[6] Kohji Tomita, Sadayuki Tsugawa, "Visual navigation of a vehicle along roads the algorithm and experiments", Vehicle Navigation & information systems Conference Proceedings 1994 Page(s): 419-424.

[7] Juan Pablo Gonzalez, Umit Ozguner , "Lane detection using histogram-based segmentation and decision trees." IEEE Intelligent Transportation Systems Conference Proceedings 2000 Page(s):346 – 351

[8] Kreucher, C.; Lakshmanan, S., "A frequency domain approach to lane detection in Roadway images." International Conference on Image Processing, 1999. Pages 31-35.

[9] Otsuka, Y.; Muramatsu, S.; Takenaga, H.; Kobayashi, Y.; Monj, T., "Multitype lane markers recognition using local edge direction.", IEEE Intelligent Vehicle Symposium 2002 Pages 604-609.

[10] Pomerleau, D.; Jochem, T., "Rapidly adapting machine vision for Automated vehicle steering.", IEEE Intelligent Systems and Their Applications Volume 11, Issue 2, April 1996 Page(s):19 – 27

[11] Nicholas Apostoloff, Alexander Zelinsky, "Robust Vision based Lane Tracking using Multiple Cues and Particle Filtering", IEEE proceedings from Intelligent Vehicles Symposium, 2003, 9-11 June 2003 Pages: 558-563.

[12] Michael Smuda von Trzebiatowski, Axel Gern, Uwe Franke, Uwe-Philipp Kaeppler, Paul Levi.; "Detecting Reflection Posts – Lane Recognition on Country Roads.", IEEE Intelligent Vehicles Symposium, June 2004. Pages 304-309.

[13] Akihiro Suzuki, Nobuhiko Yasui, Nobuyuki Nakano, Mamoru Kaneko, "Lane Recognition System for Guiding of Autonomous Vehicle", IEEE Intelligent Vehicles '92 Symposium proceedings, 29 June-1 July 1992, Pages 196-201.

[14] Massimo Bertozzi, Alberto Broggi, "GOLD: A Parallel Real-Time Stereo Vision System for Generic Obstacle and lane Detection", IEEE Transactions on Image Processing, Vol 7, No. 1, January 1998, Pages 62-81.

[15] Alberto Broggi, "An image reorganization procedure for automotive road following systems", IEEE Proceedings International Conference on Image Processing, 1995 Volume 3, 23-26 Oct. 1995, Pages 532-535 Vol. 3

[16] Alberto Broggi, "Robust real-time lane and road detection in critical shadow conditions", IEEE Proceedings International Symposium on Computer Vision, 1995, 21-23 Nov. 1995, Pages 353-358.

[17] Yuan Shu, Zheng Tan, "Vision Based Lane Detection in Autonomous Vehicle", IEEE Proceedings of the 5th World Congress on Intelligent Control and Automation, June 15-19, 2004 Hangzhou, Pages 5258-5260.

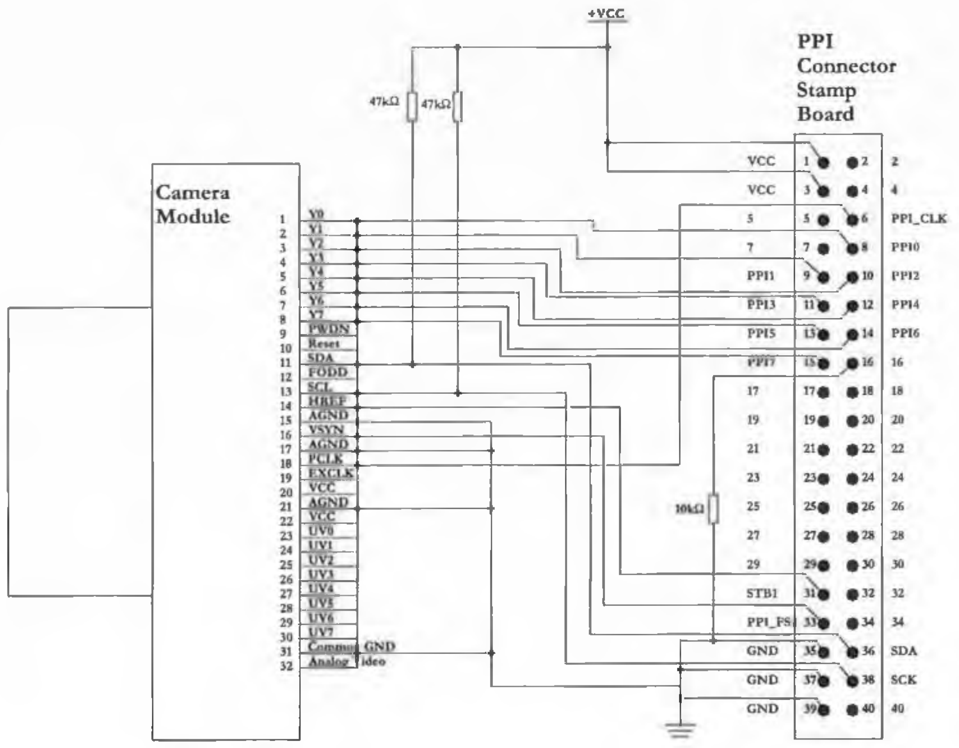
[18] Gang Yi Jiang; Tae Young Choi; Suki Kyo Hong; Jae Wook Bae; Byung Suk Song; "Lane and Obstacle Detection Based on Fast Inverse Perspective Mapping Algorithm", IEEE International Conference on Systems, Man and Cybernetics, 2000, Volume 4, 8-11 Oct. 2000. Pages 2969-2974.

[19] Anuar Mikdad Muad, Aini Hussain, Salina Abdul Samad, Mohd. Marxuki Mustaffa, Burhanuddin Yeop Majlis, "Implementation of Inverse Perspective Mapping algorithm For the Development of an Automatic Lane Tracking System", IEEE Region 10 Conference TENCON 2004, 21-24 Nov. Volume A Pages 207-210 Vol. 1.

[20] ADSP BF533 Blackfin Processor Hardware Reference Revision 3.2 July 2006
[21] http://www.uclinux.org/description/
[22] http://docs.blackfin.uclinux.org/doku.php?id=compiling_the_kernel
[23] M3188A Camera Module spec sheet
[24] Omni-Vision OV7620/OV7120 Detailed Specification document Version 2.1.
[25] http://docs.blackfin.uclinux.org/doku.php?id=sample_ppi_driver
[26] http://www.efunda.com/math/leastsquares/lstsqr1dcurve.cfm
[27] Steven W. Smith, "Digital Signal Processing – A Practical Guide for Engineers and Scientists"
[28] http://www.doc.ic.ac.uk/~dfg/AndysSplineTutorial/Beziers.html
[29] http://www.doc.ic.ac.uk/~dfg/AndysSplineTutorial/BSplines.html
[30] http://en.wikipedia.org/wiki/B-spline

8 Appendices

8.1 Appendix A- Wiring Diagram and Pin Explanation.



The Camera module pins and short explanation are as follows.

This information is mainly taken from [24], particularly pages 2 and 3.

Pins 1-8 ::Y0-Y7:- Digital Output Y Bus. The M3188A is a Black and White Camera Module which shares its interface layout with the C3188A Colour Camera Module. The Colour camera module has different modes of operation i.e. 8 bit, 16 bit etc. In 16 bit mode it would use the eight Y bits to output luminance data and a separate 8 U bits to output chrominance data. However, since the M3188A is only Black and White chrominance would have no meaning and so the Y0-Y7 data lines are used as the complete 8 bit output for the camera.

Pin 9::PWDN :- Power down mode. Defaults to logic 0, if PWDN = 1 puts the camera chip in power down (sleep) mode. Left un-connected in this project.

Pin 10 ::RST:- Reset. Can be used to reset the chip, Activated by a logic 1. Left un-connected in this project.

Pin 11::SDA:-I^2C serial data. The data line of the I^2C interface, this is used to read and write registers in the cameras chip. Camera CMOS chip considers this to be an SCCB interface in which this pin would eventually terminate at SIO-0.

Pin 12 ::FODD:- Odd Field Flag. Asserted High during the Odd field, Low during the even field. Left un-connected in this project.

Pin 13:: SCL:- I^2C serial clock output. The clock line of the I^2C interface, this is used to read and write registers in the cameras chip. Camera CMOS chip considers this to be an SCCB interface in which this pin would eventually terminate at SIO-1.

Pin 14::HREF: Horizontal window reference output. HREF is high during the active pixel window, otherwise low.

Pin 15::AGND:- Analogue Ground.

Pin 16:: VSYN: Vertical Sync output. This pin is asserted high during several scan lines in the vertical sync period.

Pin 17:: AGND: Analogue Ground.

Pin 18::PCLK:- Pixel Clock Output. By default, data is updated at the falling edge of PCLK and is stable at its rising edge. PCLK runs at the pixel rate in 16-bit bus operations and twice the pixel rate in 8 bit bus operations. For B/W mode I think this runs at the pixel rate i.e. same as 16 bit mode for colour camera.

Pin 19::EXCLK:- External clock input(remove crystal). Left un-connected in this project.

Pin 20::VCC:- Power supply 5VDC.

Pin 21::AGND:- Analogue Ground.

Pin 22::VCC:- Power supply 5VDC

Pins 23-30:: NC: Not connected. On colour camera module these would be the U data outputs for chrominance data.

Pin 31::GND: Common Ground.

Pin 32::VTO: Video Analogue output (75Ω monochrome)

8.2 Appendix B – Blackfin peripheral support information.

8.2.1 DMA Support

The following description of the DMA support comes from [20].

The processor has multiple, independent DMA controllers that support automated data transfers with minimal overhead for the core. DMA transfers can occur between the internal memories and any of its DMA-capable peripherals. Additionally, DMA transfers can be accomplished between any of the DMA-capable peripherals and external devices connected to the external memory interfaces, including the SDRAM controller and the asynchronous memory controller. DMA-capable peripherals include the SPORTs, SPI port, UART and PPI. Each individual DMA-capable peripheral has at least one dedicated DMA channel. The DMA controller supports both one-dimensional (1D) and two-dimensional (2D) DMA transfers. DMA transfer initialisation can be implemented from registers or from sets of parameters called descriptor blocks.

The 2D DMA capability supports arbitrary row and column sizes up to 64K elements by 64K elements and arbitrary row and column step sizes up to +/- 32K elements. Furthermore, the column step size can be less than the row step size; allowing implementation of interleaved data streams.

This feature is especially useful in video applications where data can be de-interleaved on the fly.

Examples of DMA types supported include:

• A single, linear buffer that stops upon completion.

• A circular, auto-refreshing buffer that interrupts on each full or fractionally full buffer.

• 1D or 2D DMA using a linked list of descriptors.

• 2D DMA using an array of descriptors specifying only the base DMA address within a common page.

8.2.2 Parallel Peripheral Interface

The following is a brief description of the PPI, taken from [20].

The processor provides a Parallel Peripheral Interface (PPI) that can connect directly to parallel A/D and D/A converters, ITU-R 601/656 video encoders and decoders and other general-purpose peripherals.

The PPI consists of a dedicated input clock pin, up to 3 frame synchronisation pins and up to 16 data pins. The input clock supports parallel data rates up to half the system clock rate. In ITU-R 656 modes, the PPI receives and parses a data stream of 8-bit or 10-bit data elements. On-chip decoding of embedded preamble control and synchronisation information is supported.

Three distinct ITU-R 656 modes are supported:

• Active Video Only - The PPI does not read in any data between the End of Active Video (EAV) and Start of Active Video (SAV) preamble symbols, or any data present during the vertical blanking intervals. In this mode, the control byte sequences are not stored to memory; they are filtered by the PPI.

• Vertical Blanking Only - The PPI only transfers Vertical Blanking Interval (VBI) data, as well as horizontal blanking information and control byte sequences on VBI lines.

• Entire Field - The entire incoming bit-stream is read in through the PPI. This includes active video, control preamble sequences and ancillary data that may be embedded in horizontal and vertical blanking intervals.

The general-purpose modes of the PPI are intended to suit a wide variety of data capture and transmission applications. The modes are divided into four main categories, each allowing up to 16 bits of data transfer per PPI_CLK cycle:

• Data Receive with Internally Generated Frame Syncs

• Data Receive with Externally Generated Frame Syncs

• Data Transmit with Internally Generated Frame Syncs

• Data Transmit with Externally Generated Frame Syncs

These modes support ADC/DAC connections, as well as video communication with hardware signalling. Many of the modes support more than one level of frame synchronisation. If desired, a programmable delay can be inserted between assertion of a frame sync and reception/transmission of data.

8.2.3 Timers

There are four general-purpose programmable timer units in the processor.

Three timers have an external pin that can be configured either as a Pulse Width Modulator (PWM) or timer output, as an input to clock the timer, or as a mechanism for measuring pulse widths of external events.

8.3 Appendix C – Contents of the Enclosed CD.

The CD attached to this Thesis document contains the source code for both the C language algorithm and the Java language algorithms.

The CD also contains a free C compiler and IDE as well as a free Java IDE.

8.3.1 The C Programme.

The C programme was compiled using Dev-C++ whose installation file is on the CD in the devC directory.
The programme may compile with other compilers but this is not guaranteed.
The C programme which implements the full algorithm from taking in the captured image to detecting the lane and determining the trajectory to follow is called fullAlgorithm.c and can be found in the C-programme directory on the CD.
To run the programme, first copy the contents of the C-programme directory to a directory on the hard drive of the computer on which it is to run. After copying from a CD the files or directories may maintain a read-only status. This should be changed, as there may be a need to modify the C file. Also the fullAlgorithm.c programme creates files in the directory that it is placed, if this directory is read only then this will not be possible. Make sure that the fullAlgorithm.c and the bitmap files are in the same directory on the hard drive.
Once this is done, open the fullAlgorithm.c from inside the Dev-C++ environment. The file can be run standalone without need to make a project file. Go to the "Execute" menu and click "Compile & Run". A DOS screen should appear momentarily with outputting text and then disappear again.
At this point in the directory where the C-file was placed on the hard drive there should be new files created.
The "intermediateOut.bmp" file contains the road image after thresholding.
The "finalOut.bmp" file contains the road image after thresholding and with the calculated trajectory superimposed on the image.
The "objectText.txt" file will contain information about the objects found in the image.

Currently the file is set to operate on the straight road image, however, it can operate on either the road image with S-bend ahead or section half way through S-bend by changing line 477 of the fullAlgorithm.c file.

To look at image with S-bend ahead change line 477 to
fpIn = fopen("longcurl20.bmp", "rb");

To look at image half way through S-bend change line 477 to
fpIn = fopen("shortcurl20.bmp", "rb");

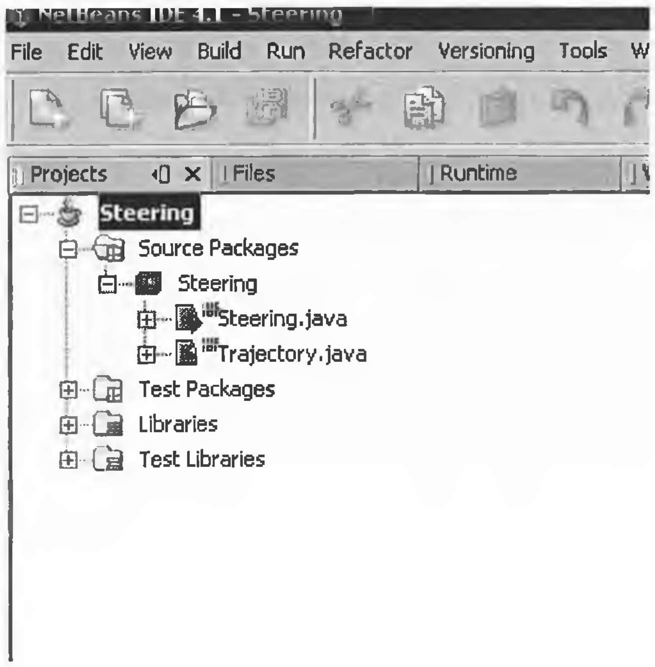
The reader should note that the original results in the thesis body are slightly different to those that will be found by running these files.

8.3.2 The Java Programmes

There are two Java programmes, one called "Steering" and one called "BSpline". These are set up as project folders as they contain more than one file.
The development environment used was Netbeans 4.1 whose installation file can be found on the CD in the Netbeans 4.1 directory.

To run the programmes copy the "Java Programmes" directory from the CD to the hard drive of the computer it is to be run on. After copying from a CD the files or directories may maintain a read-only status. This should be changed, as there may be a need to modify the Java files.

Open Netbeans IDE and go to the "file" menu and click "Open Project". Then navigate to the directory where the java programmes were copied too. Click (don't double click) on either "BSpline" or "Steering", then click "Open Project Folder", this will open either of the two programmes.
Navigate down to the source files as shown in the diagram below. Double clicking on either of the files will cause them to open in the main sub-window.



To run the programme go to the "Run" menu and then click "Run Main Project".

Making modifications to files.

The "Steering" project refers to sections 4.4.1 and 4.4.2.

In the "Steering" project the wheelbase value can be changed by modifying line 82 of Trajectory.java.

Lines 86, 87 and 88 contain the final co-ordinates and heading of the vehicle, so these can be modified by modifying these lines.

Finally the *v* value can be changed by modifying line line 95.

The "BSpline" project refers to section 4.4.3.

In the "BSpline" project the target points for the curve can be modified by changing lines 24-60 of Trajectory.java. Note that this is not the same Trajectory.java file as in the "Steering" project.