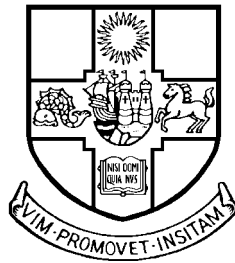


Applications of Computer Vision to Road-traffic Monitoring.

Christopher John Setchell



A thesis submitted to the University of Bristol in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering, Department of Electrical and Electronic Engineering.

September 1997

Abstract

Keywords: Traffic-Monitoring, Number-plate Recognition, Vehicle Tracking, Computer Vision, Character Recognition, Object Recognition.

Current techniques for road-traffic monitoring rely on sensors which have limited capabilities, are inflexible and often, both costly and disruptive to install. The use of video cameras (many of which are already installed to survey road networks), coupled with computer vision techniques offers an attractive alternative to current sensors. Vision based sensors have the potential to measure a far greater variety of traffic parameters compared to conventional sensors.

This thesis presents two vision based traffic-monitoring systems. The first is a number-plate recognition system. This is capable of monitoring the output from a video camera and detecting when a vehicle passes by. At this moment an image is captured and the vehicle's number-plate is located and deciphered. The second system is a generic road-traffic monitoring sensor which utilises model based techniques to track vehicles as they manoeuvre through complex road scenes. The position of the vehicle in the image is transformed to the vehicle's position in the real world enabling, among other things, vehicle speed and path to be easily measured. The development of each system is described in detail and results from testing the systems on images from real traffic scenes are presented.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or institution of learning.

C. Setchell

Acknowledgements

I must begin by thanking the members of the Vision Lab., or *boys from the lab* as they have become known. Firstly, there are the *old* boys from the lab; Neill, who is now a *girl from upstairs*, Frank and his ominous cloud formations, Simon, who has on occasion been amusingly dizzy, and Shane who has a magnificent, well polished brain and has never allowed me to become addicted to sugar or caffeine by always ensuring the coffee club is out of supplies. Shane's brain, however, was highly appreciated for the crash course in neural networks which it gave me.

Then there are the *new* boys from the lab; Angus, who will definitely start his PhD next week, Dave, who has a fantastic collection of antique jeans and Mat, who sadly is a victim of those annoying little toys which you can never find at the bottom of cornflake packets.

Finally, there are the *babies* from the lab; Ann, who has taught me the finer points of the English language (FOYW), Erik, who fortunately makes everyone else appear convincingly sane, Mark, who can always be relied on for a good story, especially the one about the plastic bag and Costas, who still has not experienced morning.

The boys from the lab have all contributed to an interesting and resourceful environment that has provided stimulation for new ideas, answers to questions, and above all, great entertainment over the last four years.

Thanks are also due to Barry Thomas who managed to direct a few pennies my way during my final, unfunded year but most of all thanks must go to Erik Dagless, my supervisor, who gave me a long enough rein to pursue the ideas and directions which I found stimulating during my research.

Of course, the love and support (emotional and financial) of my family is what put me here in the first place, so thanks to Mum, Dad, Roger and Skipper (the dog).

Thanks also to Marit, who looked after me during times of stress and supported me when my finances became depleted. I would never have finished without her help.

Finally, thanks to Golden River Traffic Limited and EPSRC for funding my studentship.

*To Mum, Dad,
and Marit.*

Contents

List of Figures	ix
List of Tables	xiii
1 Introduction	1
1.1 Road-traffic Monitoring	1
1.2 Road-traffic Monitoring and Computer Vision	4
1.3 Thesis Overview	6
2 Review	7
2.1 Introduction	7
2.2 Number-plate Recognition	8
2.2.1 Elsydel Ltd.	13
2.2.2 Computer Recognition Systems Ltd.	13
2.2.3 University of Newcastle-upon-Tyne	14
2.2.4 CSIRO and Telstra Corporation, Australia	14
2.3 Road-traffic Monitoring - Non Model-Based	15
2.3.1 TRIP	16
2.3.2 University of Tokyo	17
2.3.3 Swedish Royal Institute of Technology	17
2.3.4 California Institute of Technology - JPL	18

2.3.5	University of Bristol	18
2.3.6	CCATS	19
2.3.7	IMPACTS	19
2.3.8	Iran University of Science and Technology	20
2.3.9	University of California, Berkeley	20
2.3.10	University of Huddersfield	21
2.3.11	Autoscope	21
2.4	Road-traffic Monitoring - Model-Based	22
2.4.1	Universat Karlsruhe	28
2.4.2	University of Manchester	29
2.4.3	University of Reading	29
2.5	Summary	31
3	The Vision Hardware and Parallel Processing	34
3.1	Introduction	34
3.2	The Hardware	34
3.2.1	Digitiser/Display Module	35
3.2.2	Framestore Module	37
3.2.3	General Function Board	37
3.2.4	Input/Output/Boot Module	40
3.2.5	SCSI Transputer Module	40
3.3	The Vision Hardware and Parallel Processing	40
3.3.1	Parallel processing	42
3.3.2	Communicating Sequential Processes	45
3.3.3	Parallel C	47
3.4	The Image Processing Pipeline	48
3.5	Summary	49
4	A Number-plate Recognition System	53

4.1	Introduction	53
4.2	The Number-Plate Recognition System	54
4.2.1	Trigger	54
4.2.2	Buffer	58
4.2.3	Finder	58
4.2.4	Reader	66
4.2.5	Syntax Checker	76
4.2.6	Number-plate Database	80
4.2.7	Fax Generation	80
4.2.8	Console	80
4.3	Implementation on Vision Hardware	82
4.4	Results	84
4.4.1	Performance of Automatic Trigger	84
4.4.2	Performance of Finder	86
4.4.3	Performance of Reader	86
4.4.4	Execution times	87
4.5	Conclusions	88
4.6	Summary	93
5	A Generic Road-traffic Monitoring Sensor	95
5.1	Introduction	95
5.2	A Generic Road-traffic Monitoring Sensor	95
5.2.1	Model Based Object Recognition	96
5.2.2	The Geometric Model	98
5.2.3	Choice of Image Feature	100
5.2.4	Extraction of Image Features	100
5.2.5	Merging of Image Features	105
5.2.6	The Interpretation Tree	110

5.2.7	Tree Pruning with the View Independent Relational Model	112
5.2.8	Tree Generation with the View Independent Relational Model	114
5.2.9	Tree Pruning and Pose Recovery with the Geometric Model	116
5.2.10	Optimal Mapping Selection	122
5.3	Implementation on Vision Hardware	123
5.3.1	Digitiser Module	123
5.3.2	Framestore Module 1	125
5.3.3	General Function Module	125
5.3.4	Framestore Module 2	125
5.3.5	Framestore Module 3	128
5.4	Results	128
5.4.1	K-means Segmentation	129
5.4.2	Region Tracing	131
5.4.3	Region Merging	131
5.4.4	Vehicle Recognition	133
5.4.5	Vehicle Parameters	134
5.4.6	Execution Times	139
5.5	Conclusions	140
5.6	Summary	141
6	Conclusions	143
6.1	Future Work	145
6.2	Contribution of this Work	146
	Bibliography	147
A	Scaling by Linear Interpolation	154
B	Projection of a 3D Model into the Image	157
C	Inverse Projection of a Point in the Image to a Point in the Real World	161

D Optimisation	164
D.1 One Dimensional Optimisation	164
D.1.1 Golden Section Search	164
D.1.2 Brent’s Method	166
D.2 Multi-Dimensional Optimisation	166
D.2.1 Downhill Simplex	166
D.2.2 Powell’s Direction Set Method	167
D.2.3 Simulated Annealing	167
D.2.4 Genetic Algorithms	169

List of Figures

1.1	Sensors currently in use for road-traffic monitoring	3
1.2	A typical vision system for road-traffic monitoring	5
2.1	Template matching	10
2.2	Features for OCR: Strokes and bays	11
2.3	OCR using a neural network	12
2.4	CAD or wireframe model of a car	23
2.5	The cause of many local optima in parameter space	27
3.1	The digitiser/display module	36
3.2	The framestore module	38
3.3	The general function board	39
3.4	The input/output/boot module	41
3.5	Domain decomposition: Data driven. The data is divided into n portions, one for each of the processors	44
3.6	Domain decomposition: Demand driven. The data is divided far more finely into a number of tasks. When a processor becomes ready, the task manager provides it with a new task, thus minimising idle time	44
3.7	Algorithmic decomposition	45
3.8	Two processes communicating via a channel	45
3.9	Deadlock: The dining philosophers	46
3.10	The multiple stages of a typical image processing system	49

3.11	The image processing pipeline	50
3.12	The vision hardware configured in an image processing pipeline	51
4.1	An overview of the number-plate recognition system	55
4.2	Virtual trip-wires overlayed on an image of a traffic scene	56
4.3	Determining state of trip-wire from median deviation of low-pass filtered cross-section of image	57
4.4	State machine used in the automatic trigger	59
4.5	Image of a number-plate together with a cross-section through the plate . .	61
4.6	Estimating top of plate by counting the number of vertical edges under the sliding bar	63
4.7	Estimates of the top and bottom of a number-plate	64
4.8	Simple model of a number-plate	64
4.9	LSE: Poor fit of model due to outliers	65
4.10	Several possible plates (a), the estimates for their tops and bottoms (b) and the best fits of the number-plate model (c)	67
4.11	Large neural network for character recognition	69
4.12	Multiple, small neural networks for character recognition	70
4.13	Typical number-plate character and its histogram	72
4.14	Transfer function used to normalise grey-levels of number-plate characters .	73
4.15	Network error for training and validation sets against number of iterations .	74
4.16	n network outputs giving confidence in match of n possible characters . . .	75
4.17	A possible implementation of a syntax checker using a finite state machine .	77
4.18	A typical fax generated by the number-plate recognition system	81
4.19	Distribution of number-plate system processes onto vision hardware	83
4.20	A selection of the three thousand number-plate images	85
4.21	Distribution of times for the finder	89
4.22	Distribution of times of the reader (a) and syntax checker (b) for template matching	90

4.23	Distribution of times of the reader (a) and syntax checker (b) for n small neural networks	91
4.24	Distribution of times of the reader (a) and syntax checker (b) for the large neural network	92
5.1	Model containing knowledge of vehicle appearance and camera position within coordinate system	99
5.2	Simple model of a cuboid together with number of features extracted for both straight lines and regions. The table shows the number of combinations to be explored in each case	101
5.3	Region segmentation based on image histogram	103
5.4	Image histogram, positions of k means and corresponding mapping	104
5.5	An over-segmented cube, a list of merged image features and, for each model feature, a list of the merged image features which look like it	107
5.6	A rectangular feature and its shape descriptor based on a histogram of angles	109
5.7	Interpretation tree for recognising a cube	111
5.8	View independent relational model of a vehicle	113
5.9	Tree generation with the VIRM	115
5.10	Centre of mapped image features is found by calculating the average coordinates of all points on the external boundary	117
5.11	Centres of mapped image features and corresponding projected model features	118
5.12	Length and width of features are measured along major and minor axis respectively. Dimensions of image feature a are shown here.	119
5.13	Internal boundary between two image features and corresponding projected model line	120
5.14	Error surface for matching a point to the line: (25,25),(75,25)	121
5.15	External boundary of an image feature may correspond to any of the external lines of the model feature to which it is mapped	122
5.16	Configuration of hardware and mapping of software processes onto hardware for generic road-traffic monitoring sensor	124
5.17	Region boundary made up of a number of traces	126
5.18	Data structure representing the K-means image	127
5.19	A selection of images (a) together with their K-means segmentation (b)	130
5.20	Traced region boundaries within region of interest	132

5.21	Theoretical maximum and actual size of interpretation tree against number of image regions	134
5.22	Recognised vehicle overlaid on a selection of images from sequences A and B	135
5.23	Vehicle path overlaid on map of T-junction for vehicle in sequence A	136
5.24	Vehicle path overlaid on map of T-junction for vehicle in sequence B	137
5.25	Vehicle speed against time for vehicle in sequence A	138
5.26	Vehicle speed against time for vehicle in sequence B	138
A.1	Intensity at (x, y) must be derived by interpolating between nearest pixels .	155
B.1	Vehicle model's coordinate system	158
B.2	The camera's coordinate system	159
B.3	Projection of a point (x_c, y_c, z_c) in the camera's coordinate system to a point (i, j) in the image, where, f is the focal length of the camera, and C_i, C_j are the number of pixels per millimetre on the image sensor in the i and j directions respectively	160
C.1	Inverse projection of a point in the image to a line in the real world, where, $(x_{cam}, y_{cam}, z_{cam})$ is the camera position, α_{cam} is the incline of the camera from horizontal, C_i, C_j are the number of pixels per millimetre on the image sensor in the i and j directions respectively, f is the focal length of the camera, and (i, j) is the point in the image to be projected to a point on the road	163
D.1	Bracketing of a minimum for golden section search	165
D.2	Transformations undergone by the downhill simplex	168

List of Tables

4.1	Format of table produced by reader	69
4.2	Performance of automatic trigger	86
4.3	Performance of reader	87
4.4	Execution times for template matching (seconds)	88
4.5	Execution times for n small neural networks (seconds)	88
4.6	Execution times for single, large neural network (seconds)	88
5.1	Number of combinations against max	105
5.2	Percentage of images for which over-segmentation occurs together with average number of regions produced for several values of K	131
5.3	Average number of generated and average number of accepted merged features per image for several values of max_{merged}	133
5.4	Percentage of images in which the vehicle was correctly recognised together with average VIRM score per image for several values of max_{merged}	133
5.5	Average execution time per image for each stage of the traffic monitoring sensor	139

Chapter 1

Introduction

In this thesis two applications of computer vision are presented. Both are concerned with road-traffic monitoring, the first being a number-plate recognition system and the second a generic road-traffic monitoring sensor. This chapter will set the scene by defining road-traffic monitoring and what it is used for. The technologies which are currently in use for road-traffic monitoring will be described. The use of computer vision to create more powerful and flexible monitoring systems will then be discussed. The chapter will finish with a brief overview of the rest of this thesis.

1.1 Road-traffic Monitoring

Road-traffic monitoring involves the collection of data describing the characteristics of vehicles and their movement through road networks. Vehicle counts, vehicle speed, vehicle path, flow rates, vehicle density, vehicle length, weight, class (car, van, bus) and vehicle identity via the number plate are all examples of useful data. Such data may be used for one of four purposes:

- **Law enforcement:** Speeding vehicles, dangerous driving, illegal use of bus lanes, detection of stolen or wanted vehicles.

- **Automatic toll gates:** Manual toll gates require the vehicle to stop and the driver to pay an appropriate tariff. In an automatic system the vehicle would no longer need to stop. As it passes the toll gate it would be automatically classified in order to calculate the correct tariff. The vehicle's number-plate would be automatically deciphered and the owner sent a monthly bill.
- **Congestion & Incident detection:** Traffic queues, accidents and slow vehicles are potentially hazardous to approaching vehicles. If such incidents can be detected then variable message signs and speed limits can be set up-stream in order to warn approaching drivers.
- **Increasing road capacity:** Increasing the capacity of existing roads is an attractive alternative to building new roads. Given sufficient information about the status of a road network it is possible to automatically route traffic along the least congested roads at a controlled speed in order to optimise the overall capacity of the network.

Currently, road-traffic monitoring relies on the technology of sensors based on radar, microwaves, tubes or loop detectors (Figure 1.1):

- **Radar:** For accurately measuring vehicle speed.
- **Microwave detectors:** These are usually mounted on a bridge or gantry such that they point vertically down over a lane of traffic. The device emits microwaves which are reflected off the road surface and bounced back towards the sensor. A vehicle passing under the sensor will cause interference to the reflected microwaves which enables the vehicle to be detected.
- **Tubes:** A rubber tube fixed to the road surface across the width of a lane of traffic forms the basis of this sensor. One end of the tube is closed and the other is connected to a pressure sensor. As each wheel of a vehicle runs over the tube it causes a pressure fluctuation inside the tube which is detected by the pressure sensor. Each pressure fluctuation represents one axle of a vehicle passing over the sensor. Tubes count the number of vehicle axles which pass a particular point on the road allowing vehicle count, vehicle length and class to be deduced.

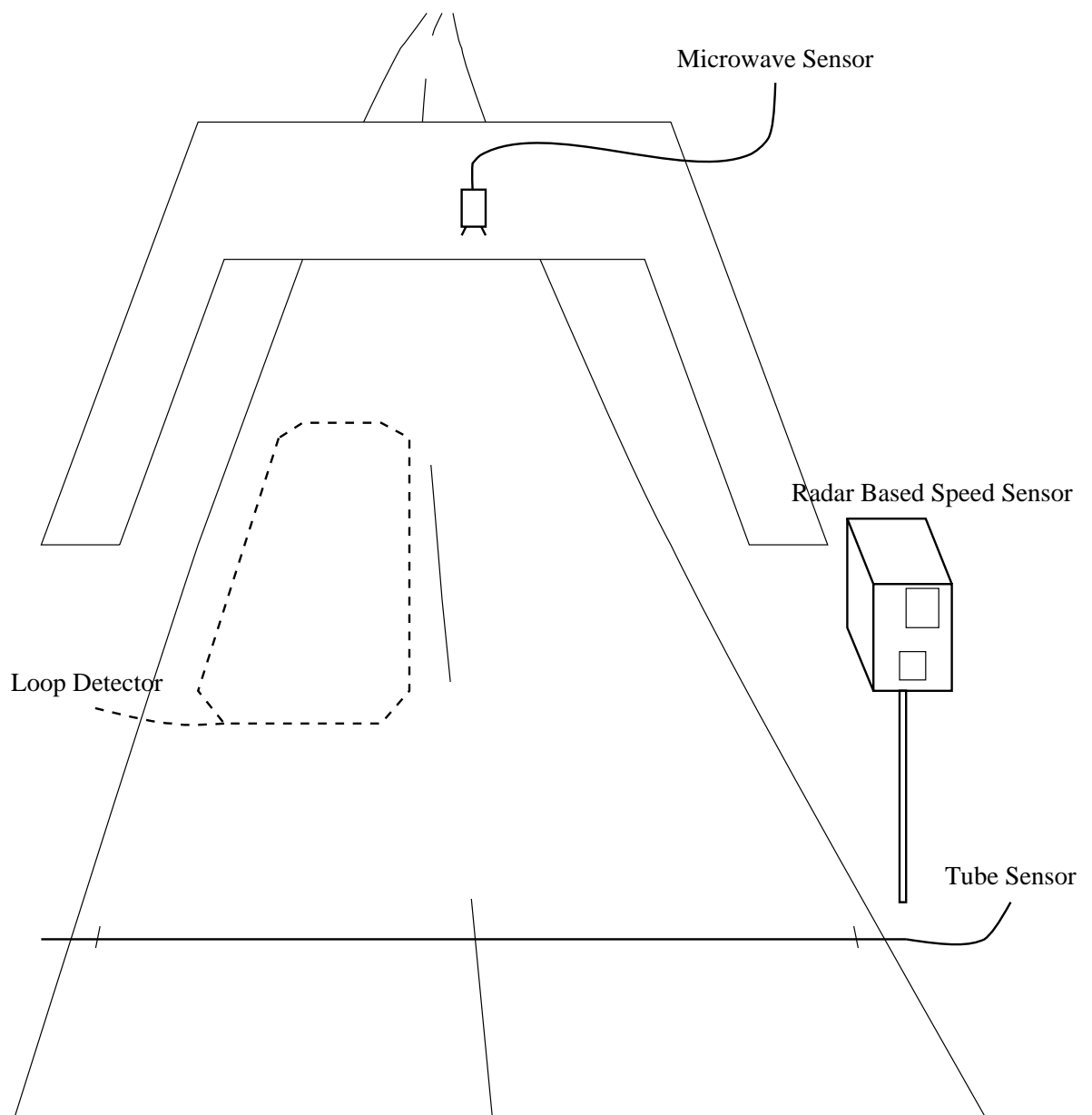


Figure 1.1: Sensors currently in use for road-traffic monitoring

- **Loop detectors:** These consist of a large coil of wire buried just below the road surface. As vehicles pass over the coil, the inductance of the coil changes and the vehicle can be detected.

From this range of sensors, loop detectors are the most prominent and are used almost universally in traffic light systems. Although an individual detector merely signals the presence or absence of a vehicle, the outputs of several detectors may be collated to deduce information such as vehicle speed, length, flow rates and density. There are several disadvantages in using such sensors. As they are only capable of detecting vehicles directly overhead, a typical road junction requires the installation of many sensors in order to cover all entry/exit points. They are highly inflexible, once installed they may not be moved. Installation is costly and disruptive. Loop detectors are vulnerable to resurfacing or road works, in the USA 30% are out of operation at any one time.

Computer vision based monitoring systems will overcome many of these disadvantages.

1.2 Road-traffic Monitoring and Computer Vision

Computer vision is the process of using a computer to extract high level information from a digital image. A typical vision system for road traffic monitoring might appear as in Figure 1.2. The CCD camera provides live video which is digitised and fed into the computer which may well contain some special purpose hardware to cope with the extremely high data rates (~ 10 MBytes/s). Computer vision algorithms then perform vehicle detection, tracking, classification or identification via number-plate recognition.

Vision is potentially more powerful than any other sensor currently available. The installation of video cameras to monitor road networks is cheaper and less disruptive than installing other sensors. In fact, large numbers of cameras are already installed on road networks for surveillance purposes. A single camera is able to monitor more than one lane of traffic along several hundred metres of road. Vision based systems have the potential to extract a much richer variety of information such as precise vehicle path, vehicle shape, dimensions and colour. With suitable positioning of the camera, a vision system is capable of tracking vehi-

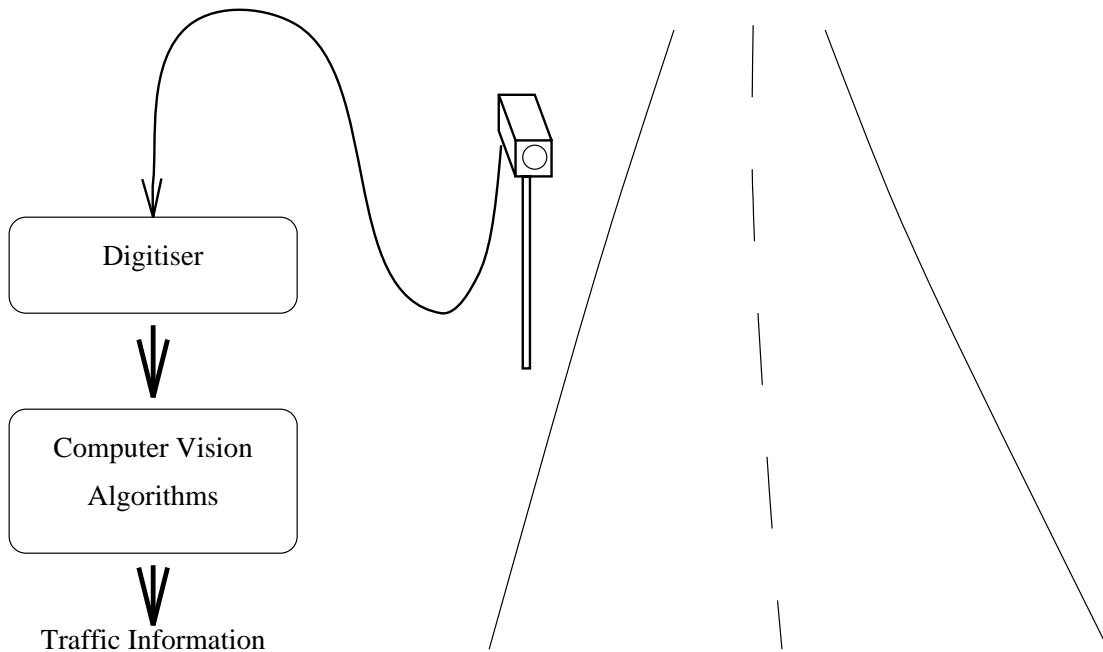


Figure 1.2: A typical vision system for road-traffic monitoring

cles as they manoeuvre through complex road junctions or along relatively long stretches of road. A vision system could theoretically have the same powers of observation as a human observer but without the detrimental effects of tiredness and boredom.

A fundamental requirement for the success of a vision based traffic-monitoring system is that it operates in real time. If each image is 720 by 512 pixels and the camera is producing 25 frames per second then the data rate is in the order of 10 MBytes/s. This may be coped with by the use of special purpose, possibly parallel, hardware. Such hardware tends to implement low level functions such as filtering (convolution) or pixelwise operators which involve very simple operations that must be repeated many times per image. The alternative way of coping with the high data rates is by data reduction, spatially or temporally. Spatial data reduction involves processing only small portions of each image known as *regions of interest*. In a typical traffic scene, much of the image is of little interest as it contains buildings, vegetation or pavement. These areas are never likely to contain a vehicle and so it is ludicrous to waste processor time on them. Temporal data reduction is achieved by only processing every n th. frame. The amount of temporal data reduction that may be applied is dependent on the particular application. A system for measuring queue length

at a set of traffic lights might only need to operate at one frame every few seconds whereas a system for tracking vehicles through junctions must process at least several frames per second.

1.3 Thesis Overview

This thesis will describe how computer vision techniques have been used to implement two road-traffic monitoring systems on a transputer based platform developed at the University of Bristol. The thesis is structured as follows:

- **Chapter 2:** A review of number-plate recognition and road-traffic monitoring systems that have been developed around the world. Relevant computer vision techniques are also introduced.
- **Chapter 3:** The applications developed in this thesis are implemented on general purpose, transputer based, vision hardware developed in the Advanced Computing Research Center at the University of Bristol. This chapter contains a description of the hardware and a brief discussion of the associated parallel programming issues.
- **Chapter 4:** The number-plate recognition system is presented in this chapter together with results from testing of the system.
- **Chapter 5:** This chapter describes the generic road-traffic monitoring sensor. It contains a specification for the sensor and describes how the specification may be accomplished using the methods of model based object recognition. It gives details of the implementation and results of testing.
- **Chapter 6:** Conclusions about the work that has been done and possible work for the future.

Chapter 2

Review

2.1 Introduction

This thesis presents the development of two applications of computer vision:

- A Number-Plate Recognition System
- A Generic Road-Traffic Monitoring Sensor

This chapter contains a review of recent research relevant to these two areas. The chapter is split into three sections. The first gives an introduction to the principal techniques of number-plate recognition, namely optical character recognition. Several number-plate recognition systems which have been developed around the world are then reviewed.

The second section deals with road-traffic monitoring systems which are *non-model based*. Non-model based systems have no idea of what a vehicle looks like and are therefore unable to achieve any image understanding. They are able to detect and track objects in the scene but are unable to recognise them. The consequence of this is that one of these systems would respond to an elephant walking down the road as if it were a car. These systems merely detect and track groups of image pixels without understanding what the pixels represent in the real world. Again, relevant techniques are introduced, in this case, motion detection. A

review is then given of several non-model based road-traffic monitoring systems which have been developed.

The third section is concerned with *model-based* road-traffic monitoring systems. The systems described in this section are different because they actually begin to gain an understanding of what is happening in the scene. Using information about the position of the camera relative to the road and knowledge of what vehicles look like, the image is transformed into a full 3D description of the scene. Not only are these systems able to locate objects in 3D real world coordinates but they are also able to recognise vehicles. These systems would not be fooled by an elephant walking down the road. The location and recognition processes are able to extract far more information than the non-model based systems, i.e. vehicle dimensions and shape, direction and precise path. Vehicle dimensions and shape can be used to classify vehicles as car, van, bus, etc. Knowledge about the vehicle shape and scene geometry is represented in *models* and the techniques of *model-based object recognition* are used to locate and track vehicles through sequences of images. This section therefore contains an introduction to the methods of model-based object recognition followed by a review of research into model-based traffic monitoring systems.

2.2 Number-plate Recognition

In order to achieve number-plate recognition, two processes must be performed. The first is to locate the number-plate and its constituent characters in the image. There are no established methods for doing this and developers are reluctant to publish details of their systems due to the commercial nature of the problem. It can be regarded as the most challenging aspect of number-plate recognition. The few systems described in the literature seem to adopt one of two approaches. The first is based on thresholding the image such that number-plate characters are black and the background white. The image is then searched for regions containing several adjacent black blobs which all have similar dimensions to the expected number-plate characters. The second approach is to utilise neural networks although the details of exactly how do not seem to have been published.

The second process is character recognition. This is a fairly well developed field in com-

puter vision and several techniques are available. The review of Govindan and Shivaprasad [1] forms the basis of the following discussion which describes the techniques of Template matching, Feature based character recognition and Neural Networks for character recognition.

Template matching: This involves the use of a database of characters or *templates*. There is a separate template for each possible input character. Recognition is achieved by comparing the current input character to each template in order to find the one which matches the best. If $I(x, y)$ is the input character, $T_n(x, y)$ is template n , then the matching function $s(I, T_n)$ will return a value indicating how well template n matches the input character (Figure 2.1). Several common matching functions are:

City block

$$s(I, T_n) = \sum_{i=0}^w \sum_{j=0}^h |I(i, j) - T_n(i, j)| \quad (2.1)$$

Euclidean

$$s(I, T_n) = \sum_{i=0}^w \sum_{j=0}^h (I(i, j) - T_n(i, j))^2 \quad (2.2)$$

Cross Correlation

$$s(I, T_n) = \sum_{i=0}^w \sum_{j=0}^h I(i, j)T_n(i, j) \quad (2.3)$$

Normalised Correlation

$$s(I, T_n) = \frac{\sum_{i=0}^w \sum_{j=0}^h (I(i, j) - \bar{I})(T_n(i, j) - \bar{T})}{\sqrt{\sum_{i=0}^w \sum_{j=0}^h (I(i, j) - \bar{I})^2} \sqrt{\sum_{i=0}^w \sum_{j=0}^h (T_n(i, j) - \bar{T})^2}} \quad (2.4)$$

Character recognition is achieved by identifying which T_n gives the best value of matching function, $s(I, T_n)$. The method can only be successful if the input character and the stored templates are of the same (or at least very similar) font. Template matching can be performed on binary, thresholded characters or on grey-level characters. In the latter case, comparison functions such as Normalised Correlation are usually used as they provide improved immunity to variations in brightness and contrast between the input character and the stored template.

Feature based character recognition: This is performed by first extracting significant features from the input character. These features are then compared to a database of feature descriptors for all of the possible input characters. The best matching descriptor provides recognition. The type of feature extracted may be classed as one of the following:

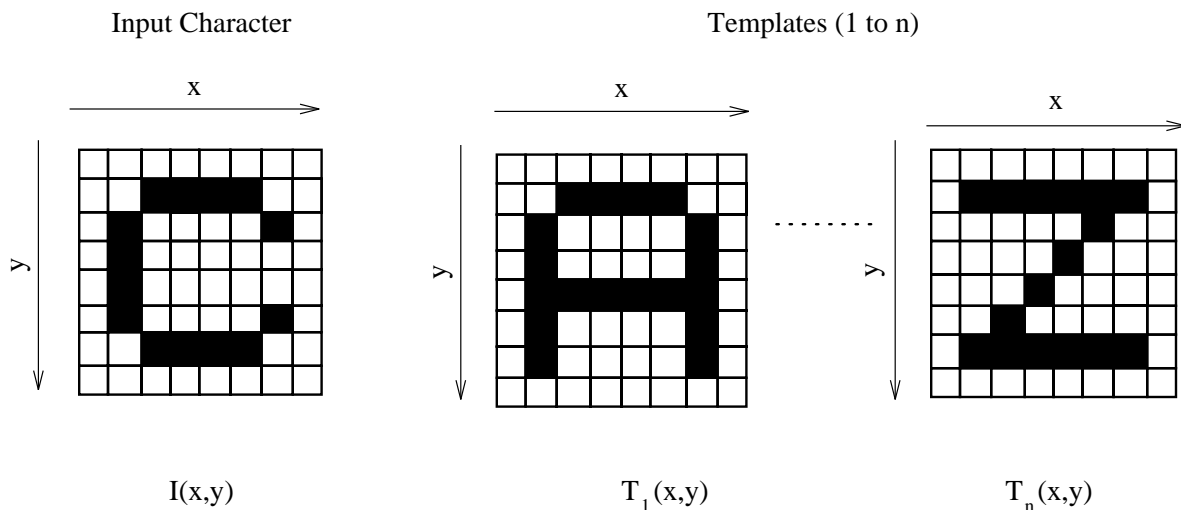


Figure 2.1: Template matching

- Features produced by global transformations and series expansions.
- Features derived from the statistical distribution of points.
- Geometrical and topological features.

Global transformations and series expansions reduce the dimensionality of the feature vector and can provide some invariance to translation, scale and rotation. Examples include Fourier, Walsh, Haar, Hadamard series expansions and Hough transform, chain-code transform and principal axis transform.

Features derived from the statistical distribution of points include Zoning, Moments, n -tuples, Characteristic Loci and Crossing and Distances. These features provide some immunity to small translation and rotation distortions as well as variations in font.

Geometrical and topological features provide high immunity to changes in font and are insensitive to small amounts of translation and rotation. Typical features might include strokes and bays in various directions (Figure 2.2), end points, intersections of lines, loops and angular relations between lines.

Neural networks: These can be used effectively for classification [2, 3, 4] and are therefore

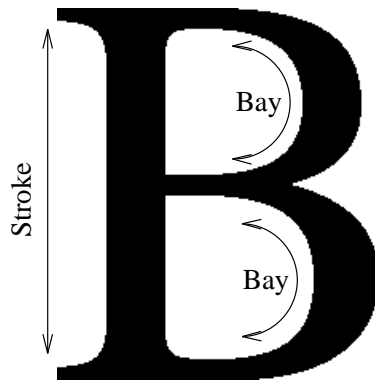


Figure 2.2: Features for OCR: Strokes and bays

suitable for character recognition. They are firstly trained on a set of example characters and are then able to classify previously unseen characters. Input to the network will either be a scaled sub-image of the character or a set of features (such as those described above) extracted from the character. In the former case, the network architecture would be as in Figure 2.3.

The network consists of a number of nodes called *neurons*. The output from each neuron is derived by passing a weighted sum of its inputs through a non-linear transfer function. The weights associated with each neuron are derived during the training phase using algorithms such as *back propagation* which use gradient methods to minimise an error function.

The network is divided into an *input layer*, an *output layer* and one or more *hidden layers*. The input layer will have one neuron for each feature in the input vector. In the case of character recognition the output layer will have one neuron for each possible character.

When an input vector is applied to the input layer, the output node which gives the strongest response yields the character's identity.

There now follows a review of some of the number-plate recognition systems which have been developed.

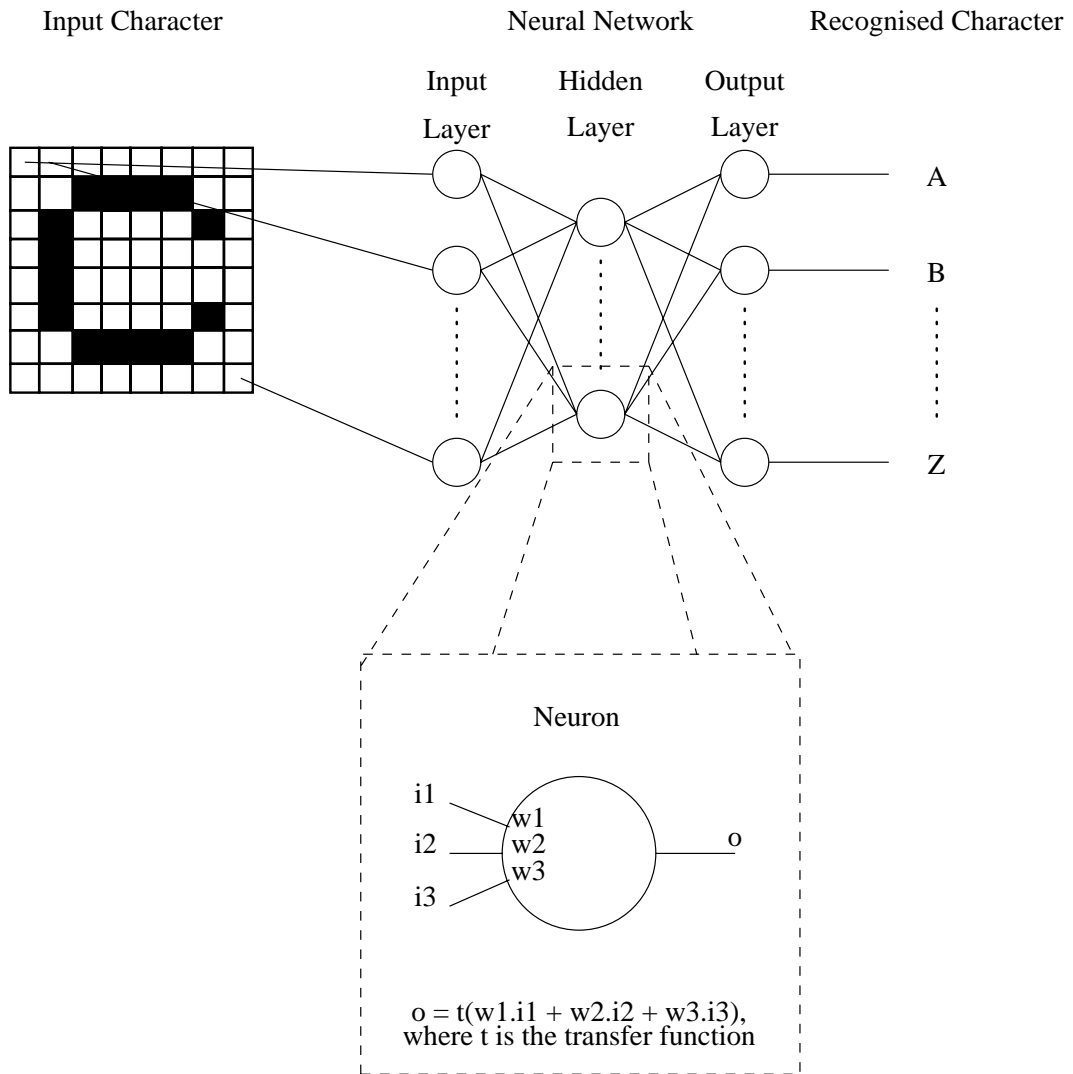


Figure 2.3: OCR using a neural network

2.2.1 Elsydel Ltd.

The number-plate recognition system developed by Elsydel Ltd. in the late 1980's [5] is suitable for use at toll gates. Vehicles approaching at approximately 20 kph. are detected by optical sensors. This triggers two flashbulbs which illuminate the vehicle's front number-plate while a CCD camera grabs an image. The number-plate is then located in the image, boundaries of individual characters are found and then the number-plate is deciphered (no more details are available). The system was installed at a toll station on a French motorway in June 1988 for testing purposes. From some three thousand nine hundred plates that were regarded as legible, 85% were read correctly, 13% with one character wrong and 2% with two or more characters wrong. Processing time for each plate was in the order of two seconds.

2.2.2 Computer Recognition Systems Ltd.

Computer Recognition Systems Ltd. have developed a number-plate recognition system which includes a syntax forcing algorithm. Details of the implementation are not available but, in 1989 a thorough evaluation of its performance was undertaken at the University of Leeds [6]. Testing was undertaken both in the laboratory and in the field. The laboratory tests involved the use of a computer to generate images of number-plates which would be read by the system. Artificial noise was added and the presence of mounting rivets was simulated. In the case of no noise, 93% of plates were correctly identified, 4% had one character wrong and 1% had two characters wrong. In the presence of noise (to simulate dead flies, etc.), performance dropped considerably to 26% completely correct, 26% with one character wrong, 18% with two characters wrong and 30% with three or more characters wrong. The addition of simulated rivets resulted in a performance ranging from 88% completely correct to 53% completely correct, depending on the location of the rivets. Another test showed that the syntax forcing algorithm was able to improve performance from 48% of plates completely correct to 63% completely correct. Results from the field trials are not available.

2.2.3 University of Newcastle-upon-Tyne

The number-plate recognition system developed during the early 1990's at the University of Newcastle-upon-Tyne [7] is activated by an external trigger. The trigger causes an image from the camera to be grabbed. This image is thresholded using an adaptive thresholding method to produce a binary image in which number-plate characters will be black, while background will be white. The number plate is then located in this image by searching for certain features (details are not given). Individual characters are located using a constrained run-length labelling algorithm which uniquely labels isolated regions in the binary image. Each character is then scaled to a 16 by 16 grid ready for input to a neural network which performs character recognition.

No results are given for the performance of the system.

2.2.4 CSIRO and Telstra Corporation, Australia

Safe-T-Cam, a product of the collaboration between CSIRO Division of Manufacturing Technology and Telstra Corporation Australia, is a traffic monitoring system for use on multi-lane highways [8]. The project was initiated in 1991 and its primary aim was to improve road safety by enforcing speed limits and maximum driving hours for drivers of large vehicles. The system consists of a number of remote sites, each able to capture high quality images of vehicles travelling at speeds up to 160km/h. The high quality image is then processed by a number-plate recognition module which transmits the deciphered number-plate and the original image to a central control station which is able to track the progress of a vehicle as it passes a number of remote sites along a highway. The central control centre is therefore able to calculate average speeds of vehicles as they travel between remote sites as well as providing a measure of the hours that a driver has driven continuously.

A remote site comprises a vehicle detection and classification module (VDCM), an image acquisition module (IAM) and the number-plate recognition module (NPRM). The VDCM uses frame differencing between the incoming image and a background reference frame to segment moving vehicles. The segmented vehicles are tracked until they reach a suitable

point on the road when the IAM is triggered to capture a high quality image of the front of the vehicle using a second camera. This process includes infra-red flood lighting to enable operation at night. Neural network techniques are then used to both locate and decipher the number-plate in the image.

Field testing of Safe-T-Cam has shown the IAM to capture suitable high quality images in over 90% of cases and of these, the NPRM read 90% correctly.

2.3 Road-traffic Monitoring - Non Model-Based

Many non-model based traffic monitoring systems rely on motion detection to segment moving regions from the image. If the regions have suitable characteristics, they are deemed to be vehicles and may then be counted or tracked as desired. There are several well established techniques for motion detection, two of which are frequently used in road-traffic monitoring systems.

- Frame differencing.
- Feature based motion detection.

Frame differencing [9] is based on the fact that the difference between two frames captured at different times will reveal regions of motion. A *difference image*, $d(i, j)$ is generated by calculating the absolute difference between two frames f_1, f_2 and then thresholding the result.

$$d(i, j) = \begin{cases} 0 & \text{if } |f_1(i, j) - f_2(i, j)| \leq T, \text{ where } T \text{ is a suitable threshold.} \\ 1 & \text{otherwise} \end{cases} \quad (2.5)$$

In the case of road-traffic monitoring it is usual for f_1 to be the incoming frame and f_2 to be a *reference* or *background* frame. The reference frame is merely an image of the scene with no vehicles in. If the incoming frame contains no vehicles then it will be identical to the reference frame and the difference frame will contain only zeros. However, if the incoming frame does contain vehicles, then these will show up in the difference frame. The purpose

of the threshold, T , is to reduce the effects of noise and changes in scene illumination. The latter is a big problem and a simple threshold is not usually sufficient to overcome it. Most systems employ a method of dynamically updating the reference frame so it adapts to changes in scene illumination.

Feature based motion detection [10] works by tracking prominent features from frame to frame. The first step is to identify suitable features. These should be areas of the image which stand out from their surroundings. Corners are a frequently used feature. After the features have been identified in all frames, the second step is to use a matching procedure to find the correspondence between these points in consecutive frames. The search for the correct correspondence is not a trivial one and iterative techniques are often used.

Below is a review of some road-traffic monitoring systems which adopt either frame differencing or feature based motion detection to detect and track objects in traffic scenes.

2.3.1 TRIP

Traffic Research using Image Processing (TRIP) was a joint project between the University of Manchester, Institute of Science and Technology and the University of Sheffield. This work began in the early 1980's and was one of the earliest attempts at using vision for road-traffic monitoring, so initial results were not particularly impressive [11]. The system was designed merely to count vehicles travelling along a bi-directional highway. The camera is mounted above the road, looking vertically down on the scene. A *background* or *reference* image is captured which contains no vehicles. The incoming images are then differenced with this reference image and the result thresholded. This gives a binary image in which only the moving vehicles will appear. The threshold is to reduce the effects of noise and changes in scene illumination. A thirty minute sequence was videoed and a vehicle count performed manually. The video was then processed by the vision system which was implemented on a special purpose, pipeline processor. A human then viewed the binary images produced by the system and counted the number of blobs which travelled along the road. The ratio of the number of blobs detected to the number of vehicles counted gave an accuracy of 89% over the two lanes.

In its final form the TRIP system was reported to attain vehicle count errors of 1%, and vehicle speed errors between 8% and 17% [12].

The TRIP system has since been upgraded by utilising improved hardware [13, 12]. Results for TRIP II have not been published.

2.3.2 University of Tokyo

The system developed at Tokyo in the early 1980's uses what Takaba calls *Sample Points* [14]. Each sample point is capable of detecting the presence of a vehicle and so by positioning a sample point in each lane of traffic it is possible to perform vehicle counts. Vehicle speed can be measured by placing two sample points in each lane of traffic and measuring the time delay between the triggering of the first and the second sample point in each lane. The sample points achieve vehicle detection by what is essentially a frame differencing technique applied only to the pixels in the sample point. Testing was done using a forty minute video recording, obtained on a clear day. The system's vehicle detection error was less than 5% and the maximum velocity error was 10%.

2.3.3 Swedish Royal Institute of Technology

In this system [15] a detection line is placed parallel to the road in each of up to three lanes of traffic. Each detection line is three pixels wide. For each detection line, background or reference intensities are stored. The current values in the detection line are differenced with the reference values and then thresholded. This process segments moving vehicles as they move along the length of the line and enables them to be tracked. Vehicle counts and approximate vehicle speed can be calculated. The use of the detection line reduces the number of pixels which have to be processed and permits operation at six frames per second.

2.3.4 California Institute of Technology - JPL

The Wide Area Detection System (WADS), developed at the Jet Propulsion Laboratory, is able to detect, count and measure the speed of moving vehicles [16]. The vehicle detection algorithm was developed for real-time implementation and therefore operates on only a small portion of the image, ie. a horizontal line of 50 pixels across each of a maximum of three lanes. The content of this line is differenced with a road-line and if the difference exceeds a threshold, a vehicle is detected.

Vehicle speed is measured using another thirty lines placed at intervals along the road. Each line signifies whether a vehicle is present or not. By examining the state of all thirty lines over a number of frames it is possible to track either the front or rear of the vehicle as it moves along the road and thus calculate the vehicle's speed.

When compared to a human observer, errors of 4% are quoted in sunny conditions and 10% during cloudy conditions for vehicle count. Measurements of vehicle speed were compared to that from a radar unit and found to be within 2% in sunny conditions.

2.3.5 University of Bristol

The Advanced Computing Research Centre (ACRC) at the University of Bristol has created a vision system capable of multiple lane vehicle counting, speed measurement and tracking of vehicles through complex road junctions [17, 18, 19, 20, 21, 22, 23]. Motion detection via frame differencing is at the heart of the system. The segmented regions of motion are tracked from frame to frame, enabling vehicle path, speed and entry/exit points to be monitored. Several methods for frame differencing have been investigated in order to minimise the effects of lighting changes while still reliably detecting motion. All algorithms are implemented on general purpose, transputer-based vision hardware, developed in-house at the ACRC [24, 25]. Low level operations such as frame differencing and the maintenance of a reference frame are carried out by the programmable hardware. This allows the system to operate at frame rates.

2.3.6 CCATS

The Camera and Computer Aided Traffic Sensor (CCATS) is a commercial system that has been developed in co-operation with the Belgian Government, the University of Leuven and Devlonics Control company [26, 27, 13]. A CCD camera is mounted looking vertically down on a roadway, giving a field of view of about twenty-five metres. The full image is digitised and stored but processing is carried out on only about twenty pixels per traffic lane. Vehicles are detected by a frame differencing technique using a threshold to cope with variations in lighting, shadows and noise. The system is able to work in both day and night.

Worst case results from testing indicate that vehicle count can be measured with an error of 4%, vehicle speed (less than 75mph) with an error of 8%, vehicle speed (over 75 mph) with an error of 16%, vehicle length (under 75 mph) with an error of 8% and vehicle length (over 75 mph) with an error of 16%.

2.3.7 IMPACTS

IMPACTS is a traffic monitoring system developed by Hoose at University College London [28, 29, 13]. It operates at a macroscopic level in that it does not attempt to identify or track individual vehicles but it provides a qualitative description of the spatial distribution of moving and stationary traffic in a scene. The system splits the image up into a number of cells, each cell being approximately the width of a lane and the length of a car (so cells over more distant parts of the road are smaller because of perspective). Each cell can be in one of three states; no vehicle, stationary vehicle or moving vehicle. Cells which have a common state are grouped together along the length of a traffic lane into objects. Objects involving stationary traffic represent queues. The state of a cell is determined using two algorithms, one to detect vehicles and one to detect movement. Vehicles are detected by looking for large numbers of strong edges within the cell. Movement is detected by performing frame differencing on the contents of a cell at two different moments in time.

The system was tested using data from a camera mounted on a gantry ten metres above a motorway. Video data was collected during summer and winter weather and lighting

conditions. The cell classification correctness was found to vary between 88.2% and 99.5% depending on weather, lighting and traffic conditions. The system detected 96.7% of all stop states while the false alarm rate (not detecting a stop state when there was one or falsely detecting a stop state when there was not one) was 0.032%.

2.3.8 Iran University of Science and Technology

Fathy's system is capable of measuring traffic queue parameters, ie. queue length, period of occurrence and rate of occurrence [30]. The system operates on small regions of the image called profiles. This enables real-time operation. Each profile is only one pixel wide and extends along the center of each lane of traffic. Queues are detected by firstly ascertaining that there is no motion along the profile. This is done with a differencing technique. Secondly, the presence of vehicles is confirmed with an edge-based vehicle detection algorithm. The presence of vehicles together with no motion signifies a queue. This condition is tested for at intervals along the profile in order to find the length of the queue. Output from the system is queue length as a function of time. Parameters such as period of occurrence and rate of occurrence may be calculated from this output. They tested the system by comparing its output with that of a human observer and found that its measurement of queue length was within 5% of that of the human.

2.3.9 University of California, Berkeley

Koller and his group have produced a system able to detect and track multiple vehicles and at the same time provide shape information which can be used for classification purposes [31, 32, 33, 34]. Frame differencing is yet again the tool for motion segmentation. The background frame is updated using a Kalman filter. The output from the frame differencing is used to initialise a contour or snake around each moving region. Kalman filters are used to track the position and shape of the contour through successive frames. Occlusion detection is performed by depth ordering the regions of each contour. The system has been demonstrated on several multi-lane traffic scenes and shown to cope well with occlusions. Implementation on a network of high performance C-40 Digital Signal Processors has shown

that near real-time performance is possible.

More recent work from the group has resulted in a feature based tracker [35]. This works by identifying corner features in the image and then tracking each feature from frame to frame using correlation. In order to group together features that come from the same vehicle, the constraint of common motion is used. This approach results in a system with greater immunity to partial occlusions. The tracker is implemented on a network of 13 C-40 Digital Signal Processors. Operation is at 7.5Hz for uncongested traffic, dropping to 2Hz for congested traffic. This drop in performance is not considered a problem as vehicle speed in congested traffic is reduced and therefore the lower tracking rate is still sufficient. Testing was performed using 44 lane/hours of video footage from a highway in California. A ground-truth was provided by loop detectors buried in the road. Measurements of vehicle velocity and flow rates were aggregated over 5 minute periods, yielding 514 samples for the traffic parameters. For velocity, 95% of the samples had an error of less than 5%. For flow rate, 91% of samples had an error of less than 20%.

2.3.10 University of Huddersfield

Cornish and Wakefield have developed a system which tracks vehicles through road scenes using the technique of frame differencing [36] to segment moving vehicles [37]. The success of frame differencing is highly dependent on the accuracy of the reference image. They have developed a method for automatically generating an accurate reference image while at the same time creating a region of interest map which reduces the portion of the image to be searched for vehicles, thus reducing the computational load. They report very satisfactory results from tests of the system on video sequences from two different road scenes.

2.3.11 Autoscope

Autoscope is a commercial system capable of measuring real-time traffic parameters [38, 39]. The project was initiated at the University of Minnesota in 1984 and has been supported by both the Minnesota Department of Transportation and the Federal Highway Adminis-

tration (U.S.). Autoscope is now commercially available via Econolite Control Product Inc. (Anaheim, CA, USA).

A typical Autoscope setup will consist of up to four video cameras connected to an Autoscope 2004 which uses image processing techniques to measure various traffic parameters such as vehicle count, speed, vehicle presence, etc. The outputs of one or more Autoscope 2004 modules are fed to a supervisor computer which is able to interactively position and show the status of overlaid virtual sensors on the road image. The supervisor computer is able to communicate the status of virtual sensors to a Traffic Controller which monitors the traffic parameters in order to set traffic lights, speed limits, variable message signs, etc.

Each virtual sensor operates on a small window of the road image. This type of data reduction enables operation at real time. A virtual sensor may be in one of two states; vehicle or no vehicle. The sensor state is determined using frame differencing techniques between the current image and a background image, together with methods for detecting shadows and reflections in wet road surfaces. By monitoring the state of one or more sensors it is possible to extract, vehicle counts, vehicle speed, vehicle length, flow rate, lane occupancy and headway.

Autoscope has been subjected to a number of trials. Video sequences from six different sites, taken during light, medium and heavy traffic flow with occurrences of both static and moving shadows, were processed by the system. Ground truth vehicle counts were produced manually and vehicle speed was measured with radar units. Over the six sites accuracies for vehicle count were between 92.2% and 98.3% while for speed it was between 94.6% and 97.7%.

2.4 Road-traffic Monitoring - Model-Based

The previous section described systems which achieved no image understanding. They merely tracked groups of image pixels. The model-based systems described in this section are more advanced as they strive to gain an understanding of the image. To achieve such an understanding, models are used to represent knowledge of the appearance of vehicles

and the geometry of the traffic scene. These models usually take the form of 3D CAD or wireframe models (Figure 2.4). The techniques of *model-based object recognition* are then used to locate vehicles in images and track them from frame to frame.

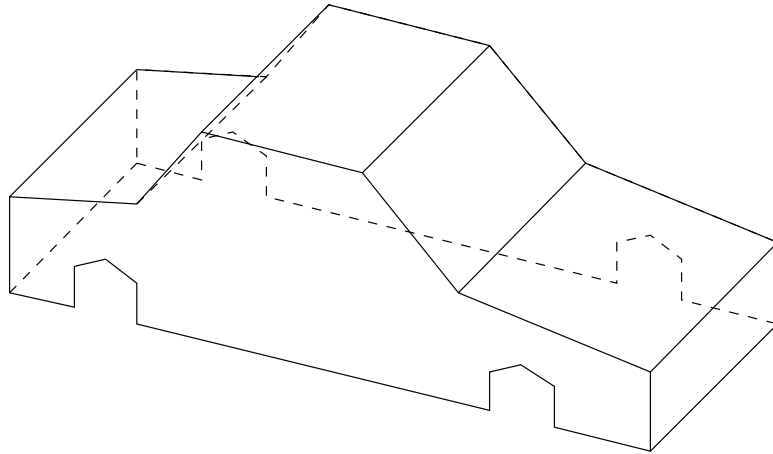


Figure 2.4: CAD or wireframe model of a car

In this chapter, the methods of model-based object recognition are introduced prior to a review of current research in the field of model-based road-traffic monitoring systems. The following paragraphs, which will provide an introduction to model-based object recognition, will cover three areas.

- History of model-based object recognition
- Model-based object recognition by searching in parameter space
- Model-based object recognition by searching in correspondence space

The following history of object recognition is based, on the excellent review of Perrott and Hamey [40].

Right from the earliest days of computer vision, the human vision system has provided inspiration for researchers of object recognition systems. Euclid, Plato and other ancient Greek philosophers believed human vision was based on something emanating from the eye; depth was sensed in a similar way to touch.

Euripides' model of vision (and taste and smell) was based on the idea that all sensed objects emitted atoms. In the case of vision, these atoms were emitted when light shone on the object. These atoms travelled to the eye in the shape of *idols* which preserved the shape of the object and were able to convey three dimensional information.

Kepler (1604) observed that the image formed on the retina of the eye is inverted and it must be the mind which performs re-inversion. It was now clear that vision was not merely a function of the eye but the mind was heavily involved.

In the early twentieth century, psychological research resulted in the *Gestalt Theory*. According to this theory, the human visual system imposes organisation on its perceptions. For example, humans would organise patterns of identical dots by forming groups of dots with small separations. If the dots are of various shades, those which are least dissimilar appear to be grouped. In a similar manner, dots which lie on a straight line or a smooth curve will be grouped together.

Roberts (1965) was the pioneer of object recognition. His system was able to recognise and locate (in real world coordinates) various objects such as cubes, prisms and wedges. This led to the term *Blocks World*, which described the environment in which all early recognition systems were limited to. Roberts' system identified lines in the image in order to extract polygons. The system then attempted to match these polygons to a set of models; cube, wedge and prism. After a match was found, the rotation, translation, scaling and projection transformations were calculated which best mapped the model vertices to the junctions in the image. In order to resolve the scale ambiguity of large, distant objects looking the same as small, near objects, Roberts used the restriction that all objects must either rest on the floor or on another object.

Waltz (1972) extended the work of Huffman (1971) and Clowes (1971) both of whom labelled junctions in the image as corresponding to a particular type of model vertex. They also labelled edges in the image as being concave, convex or occluding. When restricted to a blocks world in which the faces of all objects are planar and only three planes intersect at each vertex, only four different types of vertex exist. Depending on the viewpoint, these four types of vertex give rise to one of sixteen types of junction in the image. The labelling

of any line must be consistent with the labelling of its vertices. All possible, consistent labellings are then found by an exhaustive search. The extensions of Waltz included a greater number of labels; to include crack and shadow edges and more types of junction. He also replaced the inefficient, exhaustive search by a relaxation technique. This began by identifying all possible labels for each junction. For each junction, labels were removed from the list of possibilities if they were inconsistent with the labels of all neighbouring junctions. Neighbouring junctions were then iteratively considered such that constraints propagate through the model. This process became known as *Waltz Filtering*.

Marr describes in his book [41] an extensive theory of vision based on his understanding of the human visual system. These ideas have become known as the *Marr paradigm*. Marr presents his ideas at three levels:

- **Computational theory:** This describes the behaviour of the system in terms of the input it receives, the output it generates and the logic of how the output is generated from the input.
- **Representation and algorithm:** This level is concerned with how information is stored in the system and precise details of the algorithms which act on the information.
- **Implementation:** Details of the physical realisation of the algorithms. In computer terms; programs and hardware.

Marr states that to achieve success it is the theory that must be addressed and not the algorithms or implementation.

Marr suggests a general vision theory, based on the human visual system, which consists of three stages. Firstly, a *primal sketch* is generated. This is intended to capture significant intensity changes in the image at a range of scales. The intensity changes are grouped into tokens which are likely to represent surfaces in the real world. The second stage is the *2.5D sketch*. This is merely a depth map containing information about the distance of each surface extracted in the primal sketch from the observer. This does not constitute a three dimensional reconstruction as there is no knowledge about the occluded sides of objects. The final stage is the full *3D representation*. Marr suggests the use of volumetric primitives

to represent models which can then be searched for in the 2.5D sketch. Identification of these models will give a full three dimensional representation of the scene.

Goad has developed an object recognition algorithm [42] which recognises and locates known objects in a single monochromatic image. The first step is the extraction of straight lines from the image. A single edge from the object (model edge) is then considered and likely matches found for it in the image. Matching the model edge to the image edge constrains the object to a locus of possible poses. A second model edge is then selected. The possible positions of this edge in the image may be predicted using the locus of possible object poses. If a suitable edge can be found in the image then it can be used to further restrict the locus of possible object poses. If a suitable edge is not found then the initial match can be considered false. This procedure is repeated iteratively until either a match fails or sufficient evidence has been found to deduce that the match is correct and to establish the object's pose.

More recently, research into perceptual grouping has given promising results. These methods avoid the computational burden of extracting depth information prior to recognition. Lowe [43] argues that the erroneous conclusions reached by the Gestalt psychologists had caused their valuable insights into perceptual organisation to be ignored. Lowe believes that the perceptual groupings of interest are those which suggest something non-accidental about the image. For example, two edges which meet at a junction in the image are most likely to be caused by two edges of a single object that meet in the same way in three dimensional space. Lowe believes that prior extraction of depth information is not necessary for object recognition; humans are able to recognise objects when no depth information is available.

The traffic monitoring systems described later in this chapter adopt one of two approaches to object recognition; They either perform their search for objects in *parameter space* (pose space) or in *correspondence space*. The following paragraphs give a fairly detailed description of how object recognition is performed both in parameter space and in correspondence space.

Searching for objects in parameter space involves the use of a parameterised model of the object. This typically takes the form of a *wireframe* or *CAD* model of the object. The model

parameters specify the pose of the object in the three dimensional world. If the object pose is unconstrained then six parameters are necessary, three to specify it's position (x, y, z) and three for rotation about each of the axis (α, β, γ) . Given values for the six parameters, it is possible to use computer graphics techniques to project the model into the image. The presence of suitable image features in the vicinity of the projected model features can be used to produce a score. For example, if a projected model line happens to fall exactly over a straight line in the image then a high score would be returned. The model parameters which give rise to the best score represent the object pose which best fits the evidence in the image. As the score is a function of the model parameters, a six dimensional space must be searched to find the maximum. Unfortunately, for realistic problems the search space will contain many local optima and so optimisation is extremely difficult and location of the global optimum can never be guaranteed. The presence of many local optima can be explained by examining the simple case in Figure 2.5.

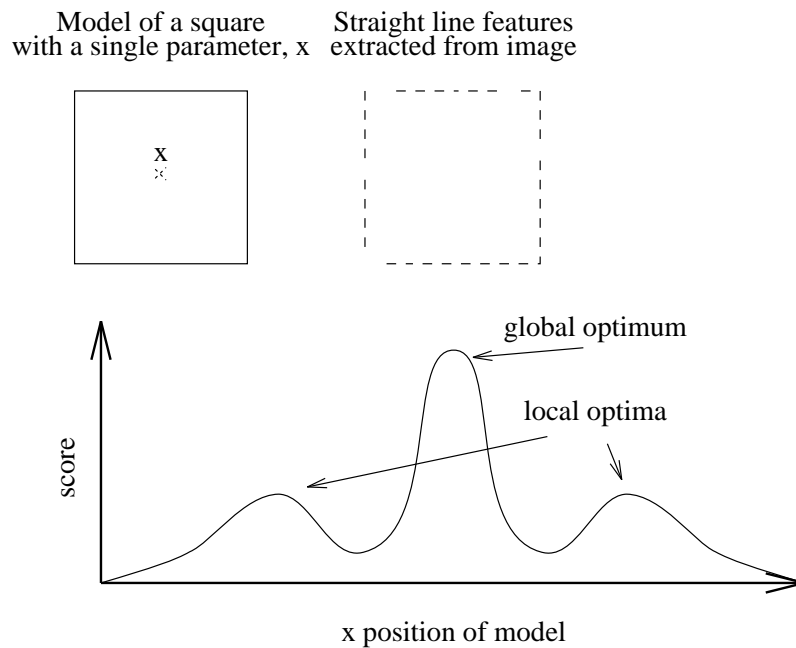


Figure 2.5: The cause of many local optima in parameter space

Considering just one parameter, x , and varying it such that the model traverses across the image from left to right, it is apparent that each model line will register with one and then another image line, producing many local maxima of the score function on the way. It is

obvious that with this happening in all six dimensions, the search space will be littered with many local optima.

The alternative to searching pose space is to perform the search in correspondence space. If the correspondence between image features and model features, i.e. which image feature represents each model feature, is known, then a score function may be derived which reflects how close a projected model feature is to its corresponding image feature. This score function will usually have only a single optima which can be robustly located using any of the techniques from mathematical optimisation. The problem is now one of finding the correct correspondence. An exhaustive search may be performed for small problems by exploring what is known as an *Interpretation Tree*. However, the size of the tree is exponential in the number of image and model features, so real problems must make use of methods to substantially prune the interpretation tree.

There now follows a review of road-traffic monitoring systems which use the techniques of model-based object recognition.

2.4.1 Universat Karlsruhe

Koller's system uses model-based vision to track vehicles through video sequences [44, 45]. The first step identifies clusters of coherently moving image features. The position of each cluster and the direction of its principal axis gives initial values for the vehicle's position, orientation and dimensions. A 3D polyhedral model of the vehicle is projected into the image using these initial values. Straight line edge segments extracted from the image are matched to the 2D projected model edge segments. To improve the quality of the match an illumination model is included which enables matching of shadow edges. The 3D generic vehicle model is parameterised by 12 length parameters which allow the instantiation of different vehicles; hatchback, bus, van. Motion of the vehicle is described with a motion model, the parameters of which are estimated with a recursive maximum a posteriori (MAP) estimator. The system has been demonstrated tracking vehicles through sequences ranging in length from 50 frames to 400 frames. Establishing initial values for model parameters is highlighted as a critical issue. Finding a number of correct correspondences between image

and model features with which to feed the recursive estimator is a severe problem. No indication of execution time is given.

2.4.2 University of Manchester

The group at Manchester has used a 2D statistical shape model which can cope with variability arising both from different designs of car and changes (over a limited range) of 3D viewpoint [46]. By using a set of 2D shape models, each dealing with a different portion of the viewsphere, it should be possible to achieve recognition from any viewpoint. Linear edge segments are extracted from the input image. These segments are labelled using Pairwise Geometric Histograms which encode local geometric contextual information. The parameters of the Active Shape Model are iteratively adjusted until an optimal match is found with the labelled line segments. Tests with synthetic data suggest the method to be resilient to noise while tests with real data have shown the method capable of tracking a vehicle turning a corner.

2.4.3 University of Reading

The group at Reading is indisputably the most prominent in this field of computer vision. Over the years they have investigated several different approaches to tackling the problems of road-traffic monitoring. Some of their earlier work resulted in CARRS (CAR Recognition System) [47, 48, 49]. CARRS is a system for finding and locating vehicles in 2D images. Vehicle recognition occurs when an optimal mapping between model and image features has been found. In other words, if each image feature is labelled according to which part of the vehicle it represent, e.g. front nearside wheel, front windscreen, etc., then the task of vehicle recognition is to find the optimal labelling i.e. a search in *correspondence space*. The first step of CARRS' operation is the extraction of edges from the image using a Canny edge detector. The edges are segmented at curvature maxima to form polygonal approximations which are then searched for occurrences of cue features such as U shapes, S shapes and closed quadrilaterals. A search is then made for consistent labellings of subsets of the cue features. This process yields a number of model hypotheses. Each hypothesis is then verified

by performing viewpoint inversion to identify the viewpoint which is consistent with such a labelling. Knowledge of the viewpoint is then used to project a CAD model of the vehicle back into the image where evidence supporting the existence of the model is aggregated to form a score for the hypothesis. Hypotheses achieving a sufficient score are accepted. Each accepted hypothesis represents an instance of a vehicle in the image and the viewpoint inversion yields the real-world location of the vehicle in the scene.

Further work at Reading yielded Viewpoint Consistency Ascent [50]; a method for iteratively refining an initial labelling in order to find the optimal labelling of image features. At each iteration, additional image features are sought which are consistent with the viewpoint associated with the current labelling.

An alternative approach, which does not necessitate the recovery of viewpoint, makes use of View Independent Relational Models. These represent an object independent of pose (viewpoint). Recognition can therefore be achieved without recovering the viewpoint which is a computationally expensive task. The VIRM consists of relationships between model features such as; the front edge of the roof is parallel to the rear edge of the roof or the front offside window cannot be visible at the same time as the front nearside window. Such a model can be used to search for labellings which are consistent with the VIRM and therefore form a valid hypothesis of the model. The group at Reading have developed a method for automatically generating a VIRM from a CAD model of the vehicle and then using the VIRM to recognise and locate the vehicle in a 2D image [51, 52, 53].

Reading have also developed a traffic understanding system under the CEC ESPRIT VIEWS project (Visual Interpretation and Evaluation of Wide-area Scenes) [54, 55]. The first stage of this system performs frame differencing between the incoming image and a reference frame. The result of this is cleaned up using morphological operators. This yields a number of regions which are tracked from frame to frame. A number of shape, size and velocity characteristics of the tracked regions are then measured to provide an initial indication of the class and pose of each individual vehicle. The initial pose is used to project a CAD model of the vehicle into the image where supporting evidence is sought and aggregated to produce a score reflecting how well the projected model matches the image features. The score is a function of the model parameters (pose) and can therefore be optimised by

searching the parameter space using techniques such as the simplex. If the optimal score is sufficient the vehicle can be considered recognised and its pose recovered. The recovered pose is used as the initial pose in the next frame and in this way the vehicle is tracked from frame to frame. More recent work has investigated the use of kinematic models of vehicles incorporating steering angle and tangential acceleration [56]. Given the recovered pose of the vehicle in previous frames these models can be used to predict the pose of the vehicles in the next frame. This leads to more reliable tracking.

The VIEWS system has been demonstrated on several sequences of traffic scenes including condition of heavy traffic and extensive partial occlusion.

More recently, the vision system has been enhanced to include deformable vehicle models. This allows the CAD model to change shape according to the type of vehicle it is being fitted to; i.e. saloon, estate, hatchback, van, etc. The result of this is that the vehicle can be located more accurately and its dimensions can be recovered and used for classification. In order for the model to be deformable it must have extra parameters which allow it to change shape, e.g. length, width, height, length of bonnet, length of boot, etc. The system uses a six parameter deformable model [57, 58]. This was derived from a twenty-nine parameter model which was fitted to ninety examples of vehicles. Principal Component Analysis revealed that the first six modes accounted for more than 97% of the variation in the data. These six modes were used as the parameters for the deformable model.

2.5 Summary

This chapter has reviewed material relevant to number plate recognition and road-traffic monitoring.

In the case of number plate recognition, the relevant techniques of Optical Character Recognition (template matching, feature based recognition and neural networks) have been introduced. Several number plate recognition systems which have been developed from the early 1980's onwards have been presented. Two of these systems (Elsydel and Newcastle-upon-Tyne) require an external sensor to provide a trigger signal when a vehicle is suitably

positioned in front of the camera. This destroys many of the advantages associated with using video cameras and computer vision for traffic monitoring purposes. The road will have to be closed while a loop detector or tube is installed. The additional sensor not only increases costs but also has the potential to reduce reliability and the system as a whole is now vulnerable to resurfacing and roadworks. It is therefore vital for future systems to incorporate an *automatic trigger* such as the one used in the Safe-T-Cam. This generates a trigger signal by using computer vision techniques to detect when a vehicle is in a suitable part of the video image, thereby avoiding the need for an external sensor.

The system developed at Newcastle-upon-Tyne uses thresholding to locate the number-plate characters. For this approach to be successful, the number-plate must be well lit, perhaps using flashbulbs triggered at the moment of image capture, such that there is good contrast between number-plate characters and number-plate background and the lighting across the plate is reasonably uniform. Also, if thresholding is to be used to identify the boundary of individual characters, then artefacts such as shadows, cracks or dirt on the plate must be absent otherwise adjacent characters may become joined during the thresholding process. Satisfying these constraints in a real environment is impossible and so the use of thresholding can only result in a reduction of potential performance. The Safe-T-Cam uses neural networks to locate the number-plate characters in the image. No details are given but it is likely that the network has been trained to recognise areas of high contrast, a prominent characteristic of number-plate characters. Such an approach promises far greater robustness if suitable methods can be identified.

For non-model based road-traffic monitoring systems, the important concepts of motion detection via frame differencing or feature based motion detection have been introduced. This provided the necessary groundwork for a review of non-model based traffic monitoring systems developed from the early 1980's onwards. These systems displayed no image understanding. They worked by merely tracking groups of image pixels without any understanding of what the pixels represented in the real-world. The drawback of this is that the system will respond to objects even if they are not vehicles. Nonetheless, such systems do have their applications. If the traffic monitoring requirement is only for approximate vehicle counts or approximate vehicle speed then these systems do provide a cheap and cheerful solution. However, computer vision as a tool for traffic monitoring has the potential to

measure a far richer variety of traffic parameters. This potential cannot possibly be realised by the simplistic approaches adopted in the non-model based systems.

The review of model-based traffic monitoring systems was preceded by an introduction to the field of model-based object recognition. Within this field, object recognition techniques based on searching both parameter space and correspondence space were described. Various model-based traffic monitoring systems were then presented, with the work from Reading University highlighted as being the most prominent. The system developed at Karlsruhe and the VIEWS system from Reading both operate by searching parameter space and both claim impressive results. However, as highlighted by Koller at Karlsruhe, the parameter space approach is dependent on accurate initial values for the model parameters. Both systems use position, size, shape and velocity of regions of motion within the image to derive initial values. These cues are not robust indicators of vehicle pose and position even for typical situations but in cases such as skidding vehicles are considered they are of no use whatsoever. If provided with inaccurate initial values, the system will still latch on to a wrong match. Worse still, there is no mechanism for recovery from an incorrect match.

The work from Manchester and the remaining work from Reading is based on searching correspondence space. This approach offers the best performance in terms of recognition accuracy. No initial values are required and a wrong match in one frame can be corrected in subsequent frames.

The model based systems exhibit a degree of image understanding. Using knowledge of the appearance of vehicles, represented in models, they positively identify objects as vehicles, thus making the system more robust and allowing the extraction of extra information such as vehicle dimensions and shape. The extra information allows vehicles to be classified as car, van, bus, etc.

Chapter 3

The Vision Hardware and Parallel Processing

3.1 Introduction

This chapter will describe the general purpose, Transputer based, vision hardware on which both the number-plate recognition and generic vehicle monitoring sensor have been developed. As already mentioned, it is desirable for both of these systems to operate in real time. To achieve this, full use must be made of the parallel processing capabilities offered by the network of Transputers which is created when a number of the vision hardware modules are connected together. After a description of the hardware and its functionality, parallel processing theory will be introduced, followed by a description of the image processing pipeline used in both applications.

3.2 The Hardware

The vision hardware [24, 25] comprises a number of double sized Eurocard boards which can be mounted in a VME crate. Each board is hosted by a single Transputer which controls and configures the hardware on the board and supplies four communication links for connection

to other Transputers/boards. The Transputer links have a bandwidth of 20MHz. To avoid using this bandwidth for image transfer each board is equipped with an interface to a video bus based on the MAXbus standard [59]. Each video bus can transmit 8 bits at 10MHz. For high resolution applications it is necessary to use two 8 bit busses in parallel, each transmitting alternate pixels. The boardset consists of a digitiser/display module, a framestore, a general function board and an input/output/boot module. In addition, a commercially available Transputer Module (TRAM), which provides an interface to a SCSI disk, is described. All boards have an associated set of software driver functions. These provide a user friendly method for configuring the hardware on a particular board. Each board will now be described in detail.

3.2.1 Digitiser/Display Module

A block diagram of this module is contained in Figure 3.1. Video signals input to the board must comply with the CCIR 625 line broadcast standard. An input multiplexor is able to select between three video input channels or a reference image which is generated on-board. The analogue video signal is digitised to 8 bits at a rate of 13.5MHz, producing an image of 768 by 576 pixels which is broadcast down a pair of video busses. Alternatively, a single video bus may be used, in which case 384 by 288 pixel images can be transmitted at field rates. Digitisation occurs between upper and lower voltage levels which can be programmed in software or hardware. Before the digital video stream is transmitted down the video bus it passes through a 256 byte look-up table. The look-up table can be used to compensate for non-linear gamma factors in the camera, or to produce effects such as histogram stretching or thresholding. The board contains both a filter to strip the colour sub-carrier from the video signal and a timebase corrector circuit to compensate for erratic synchronisation signals which often occur when playing back from a video recorder.

The display side of the board will either display directly the image being digitised or the image arriving on one or both of the video busses. Pixels arriving on the video busses first pass through a 256 x 3 x 8 bit colour look-up table. This enables images to be displayed using 256 different pseudo colours chosen from a palette of 16.7 million colours. The board outputs on three RGB video connectors.

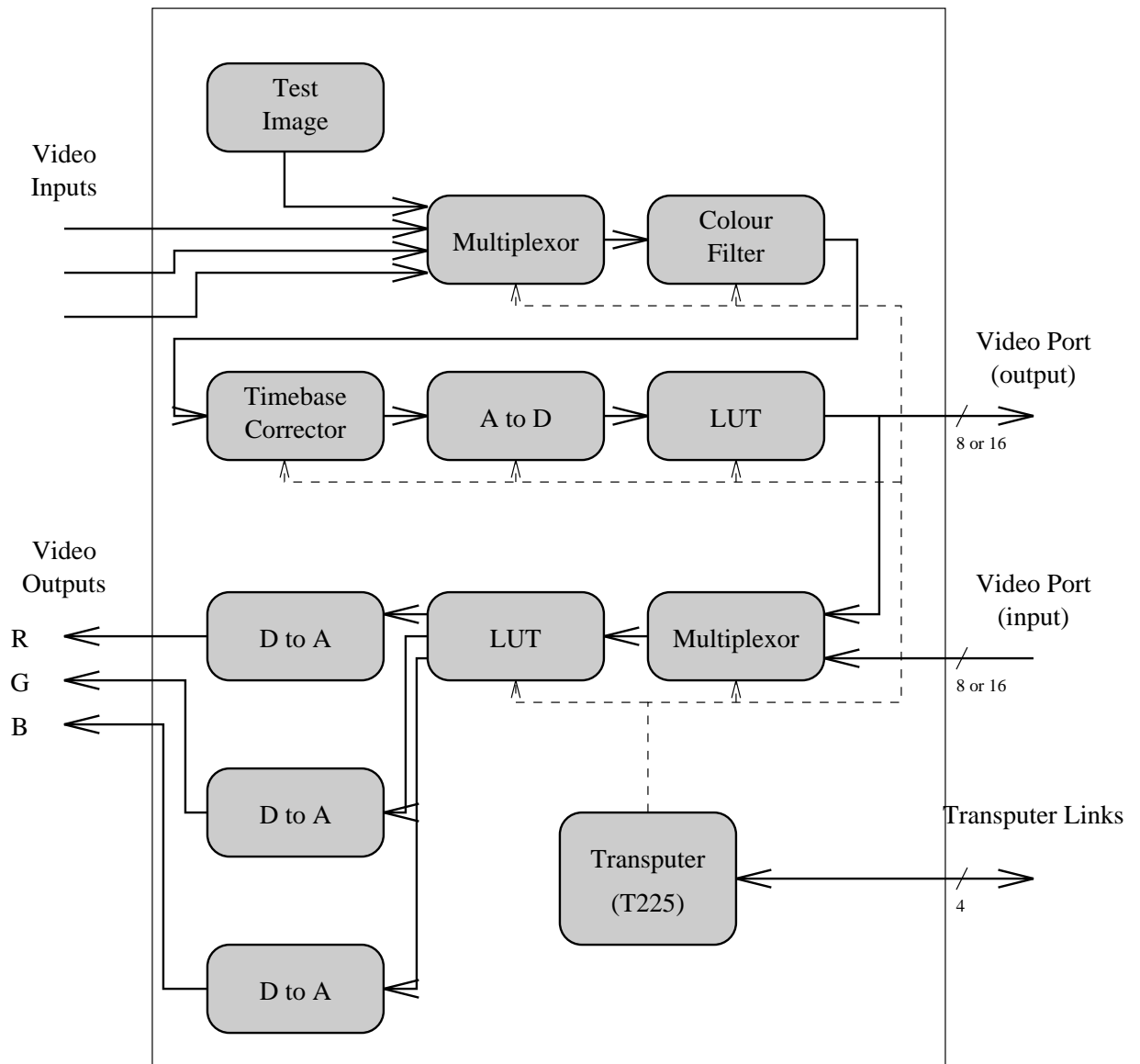


Figure 3.1: The digitiser/display module

3.2.2 Framestore Module

The framestore module (Figure 3.2) has two 1MByte banks of video RAM for storing images, a T805 Transputer and 1M or 4MBytes of program memory(DRAM). Both the video RAM and the DRAM are mapped directly into the Transputer's address space. This provides the applications running on the Transputer with easy access to the image data. The board has two video ports, each able to transfer up to 20M pixels per second via a pair of video busses. Each port can be configured for input or output and may be connected to either bank of video RAM, although both ports may not be connected to the same bank simultaneously. Transfer of images from the video ports to the video RAM is performed automatically in hardware requiring very little intervention from the Transputer. This leaves the Transputer free to perform image processing tasks.

A number of framestores can be configured to input from the same video bus such that they are all able to access global image data. Alternatively, each framestore can be configured to input an image, perform image processing and then output the result. Several such framestore can be cascaded in an image processing pipeline.

3.2.3 General Function Board

The general function board consists of a Transputer, 512KBytes of program memory, a programmable delay line (512KBytes) and two 64KByte look-up tables (Figure 3.3). Inputs A and B are each supplied from an 8 bit video bus. Pixel values from A and B are combined to form a 16 bit address into a look-up table where an output value is stored. In this manner, any pixelwise function of two images can be implemented. The programmable delay line may be used to delay input B by any number of pixels up to a complete frame. The two look-up tables may either be used to implement two independent functions or in a double buffered manner to enable dynamic programming of the look-up tables, i.e. one is used to produce output while the other is loaded with new values, the look-up tables are then switched during frame blanking.

Using the multiplexors to control data flow it is possible for this board to perform operations

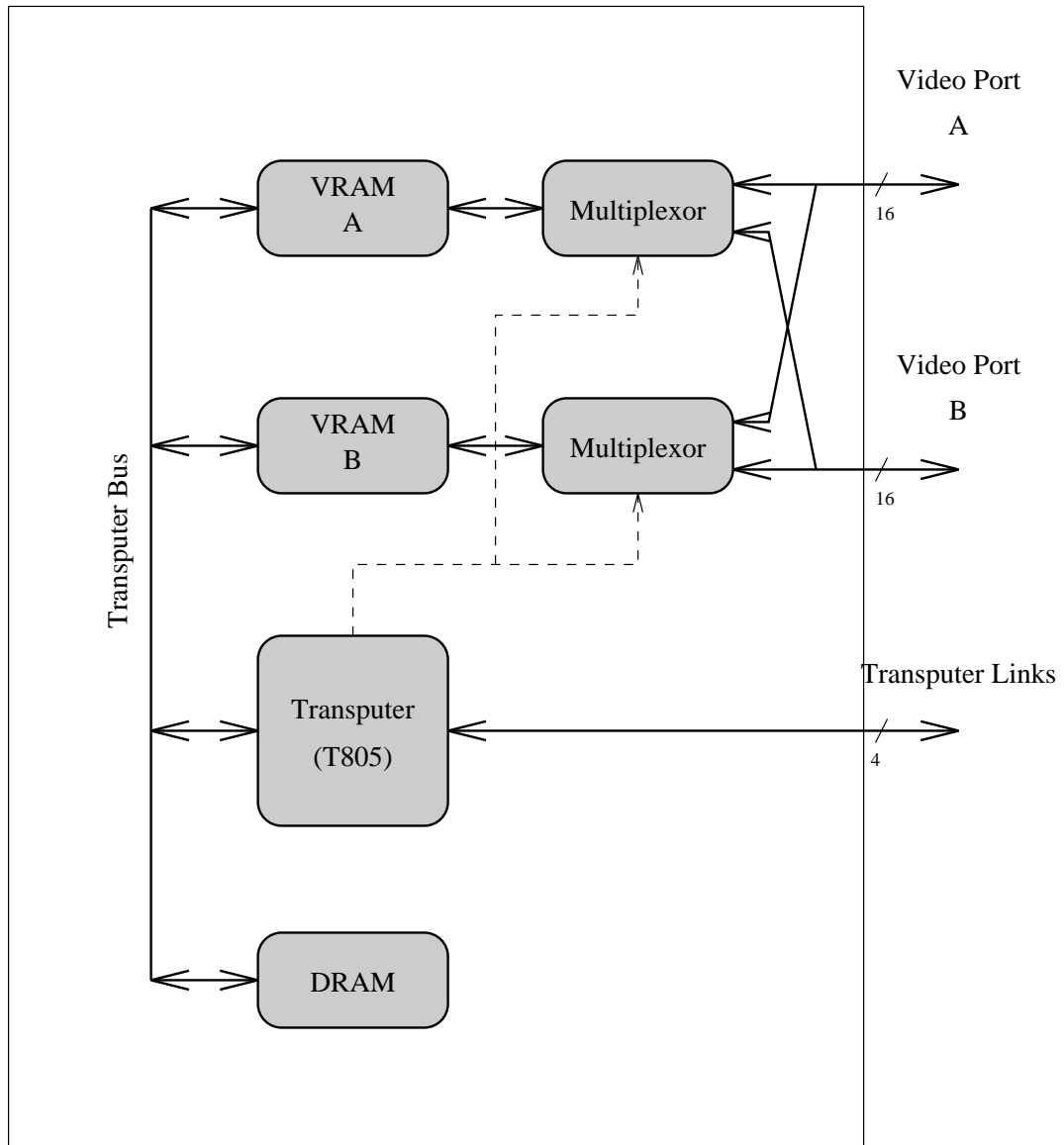


Figure 3.2: The framestore module

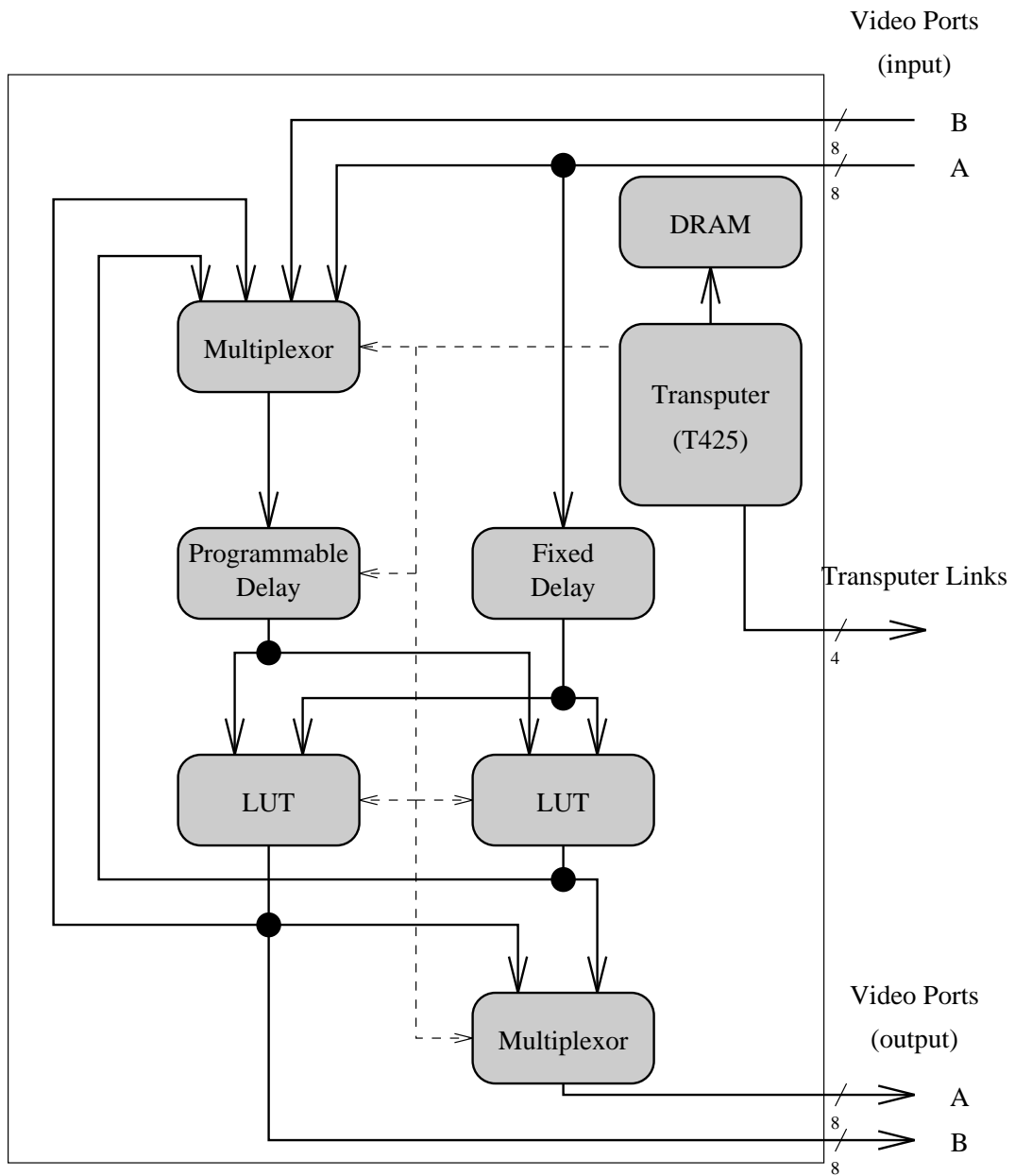


Figure 3.3: The general function board

such as edge detection, frame differencing and temporal smoothing.

3.2.4 Input/Output/Boot Module

The purpose of this module is to make stand-alone vision systems possible. The module is based around a T425 Transputer which control various input/output devices (Figure 3.4). The board has three serial RS232 ports, a parallel input/output port, timer and real-time clock and a battery backed up memory. A mixture of RAM and ROM can be accommodated up to a total of 8MBytes. When building a stand-alone vision system this board is able to boot all other boards by downloading code stored in ROM to the rest of the Transputer network. On power failure an interrupt is generated which triggers the software to back-up critical data values to battery backed-up memory before gracefully shutting down. A watchdog timer is also provided for automatically rebooting the system after program failure.

3.2.5 SCSI Transputer Module

The SCSI TRAM provides an interface between the Transputer network and a SCSI disk. Images can be written to or read from the disk at rates up to five frames per second. The ability to present image processing algorithms with repeatable images is an invaluable debugging aid. Access to a large database of images is of great importance when testing image processing algorithms.

3.3 The Vision Hardware and Parallel Processing

A vision system is created by connecting together a number of the above modules. The programmer will see the boards as a network of Transputers. This network of Transputers may be used to perform tasks in parallel. The following sections will provide an overview of parallel processing theory and a description of the CSP model, around which the Transputer is based.

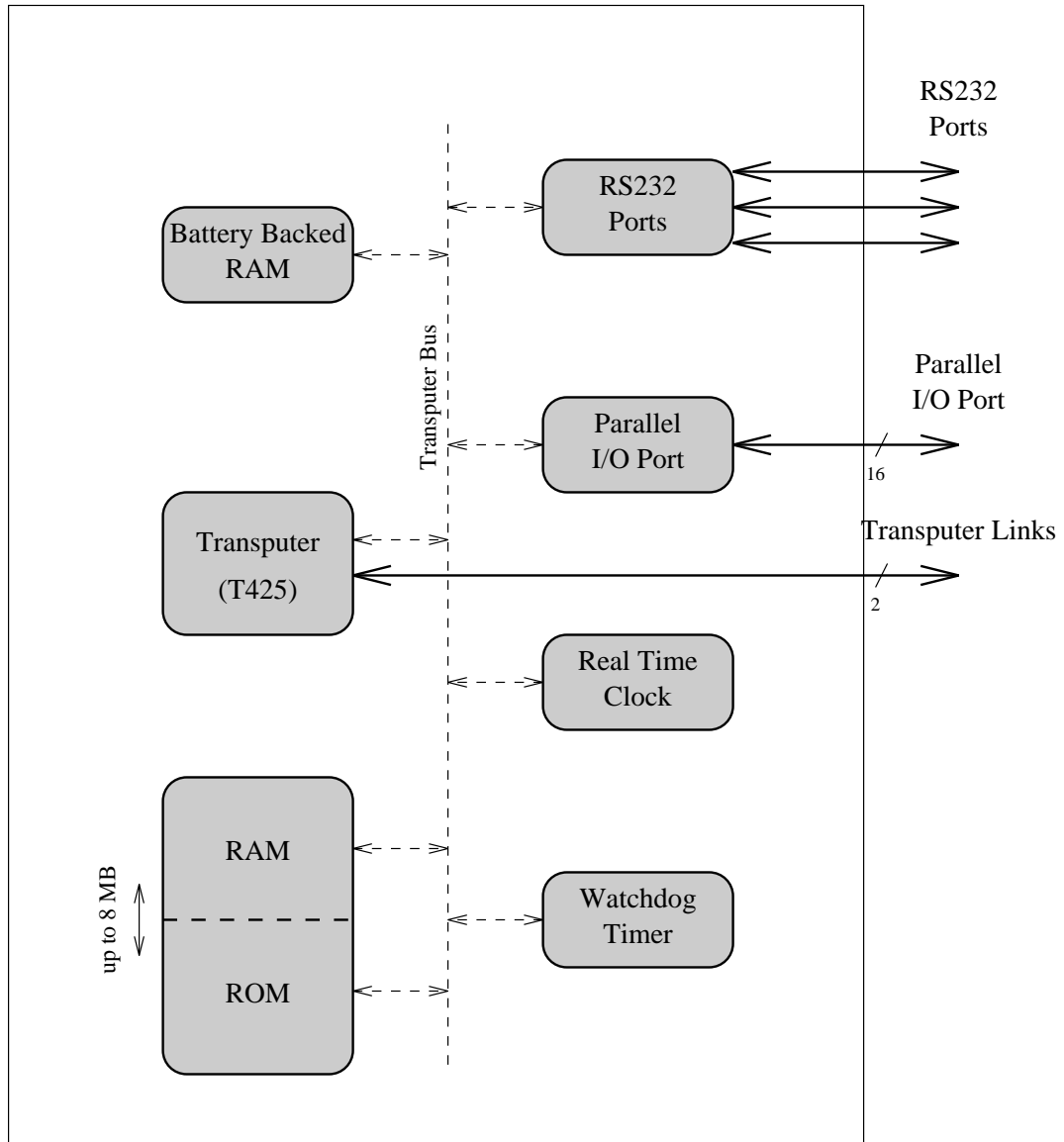


Figure 3.4: The input/output/boot module

3.3.1 Parallel processing

In 1972 Flynn proposed a classification for processors and multi-processor systems which became known as *Flynn's Taxonomy* [60]. Within this taxonomy there are four classifications:

- **Single Instruction Single Data (SISD):**

Sequential processors which adhere to the von Neumann architecture fall into this category. A single processor performs instructions in a sequential fashion, operating on a single stream of data.

- **Single Instruction Multiple Data (SIMD):**

A number of processors all perform the same instruction on different pieces of data. The Distributed Array Processor (DAP), the Massively Parallel Processor (MPP) and the Connection Machine are such processors.

- **Multiple Instruction Single Data (MISD):**

A number of processors, each performing different instructions on a stream of data that has come from a single source. The processors are usually organised into a pipeline such that the first will perform an operation on the incoming data and then pass the result on to the next processor which will perform a different operation and so on.

- **Multiple Instruction Multiple Data (MIMD):**

MIMD describes a number of processors, each performing their own instructions on their own data. The processors may therefore operate asynchronously. For these processors to co-operate in the solution of a single problem they must communicate with each other. Communication may be achieved via *shared memory* in which case controls must be implemented to prevent processors from simultaneously writing to the same memory location. The Sequent Balance and Alliant FX/8 are MIMD processors which use shared memory.

Communication may also be achieved by passing messages along a communication channel. Such systems may have *distributed memory* in which each processor has

its own private memory space. The Intel i860 and Inmos Transputer are distributed memory MIMD processors.

Before a problem may be solved on a multi-processor system it must first be decomposed into a number of processes which may be executed in parallel. This may be a domain decomposition or an algorithmic decomposition [61]:

- **Domain decomposition:** A number of processors concurrently execute the same algorithm but on different data items. If an edge map of an image is required and there are n processors available then the image can be divided into n regions with each processor producing the edge map for one region (Figure 3.5). For fixed size regions this is known as *data driven* domain decomposition. The problem with the data driven approach is that some regions may take longer to compute than other regions. Therefore, some processors will finish their task before others and will be left idle. Such load balancing problems result in reduced efficiency. The alternative to data driven is the *demand driven* approach (Figure 3.6). Here, the data is divided much more finely. As a processor becomes idle it requests another region of the image for processing. In this way all processors are kept busy until no more regions of the image, or tasks, remain.
- **Algorithmic Decomposition:** This approach strives to identify parallelism inherent in the algorithm. Figure 3.7 shows a simple algorithm and how the first two parts may be executed concurrently.

Any problem may now be decomposed into a set of processes which have the ability to communicate with each other and which may execute concurrently. Each individual process is purely sequential. Hoare's study of such processes led to the theory of Communicating Sequential Processes (CSP) [62].

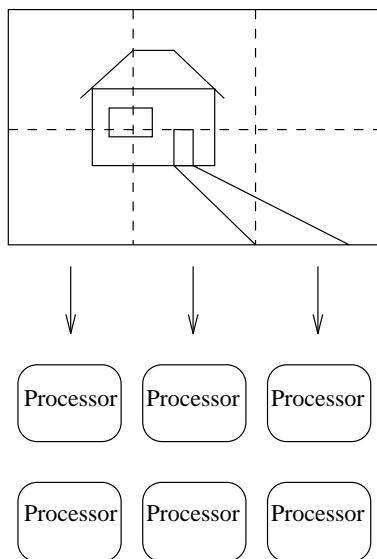


Figure 3.5: Domain decomposition: Data driven. The data is divided into n portions, one for each of the processors

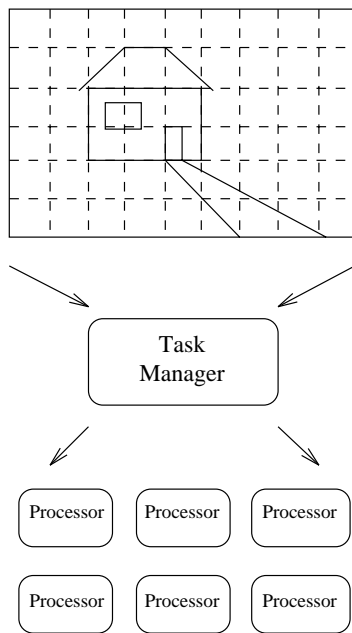


Figure 3.6: Domain decomposition: Demand driven. The data is divided far more finely into a number of tasks. When a processor becomes ready, the task manager provides it with a new task, thus minimising idle time

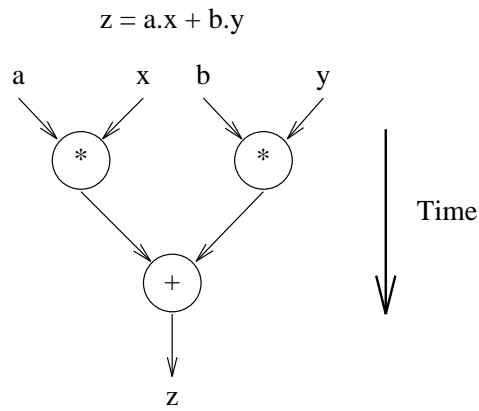


Figure 3.7: Algorithmic decomposition

3.3.2 Communicating Sequential Processes

When designing a system which consists of many communicating sequential processes the designer must be wary of the potential pitfalls. The theory of CSP provides a mathematical tool for specifying and analysing complex systems of interacting processes and so provides the means for avoiding these pitfalls.

Within CSP a process is purely sequential and may execute concurrently with all other processes. Processes synchronise and exchange data by communicating with each other. This communication takes place over *channels*. Each channel is a unidirectional connection between two processes (Figure 3.8). After a process initiates a communication it is unable to continue until the communication is complete.

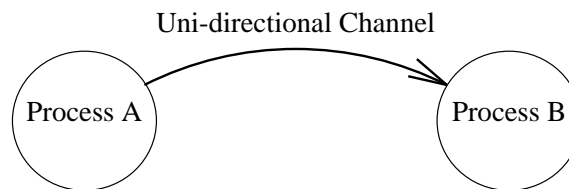


Figure 3.8: Two processes communicating via a channel

One common and serious problem associated with concurrent processes is that of *deadlock*.

If two processes are both waiting for input from each other then neither may proceed and the whole system is in a state of deadlock. Dijkstra's famous 'Dining Philosophers' are an illustration of this problem:

There are five philosophers who are dedicated to thinking but, of course, from time to time they must eat. For this purpose they are provided with a dining room containing a table surrounded by five chairs, one for each philosopher (Figure 3.9). On the table there is a large bowl containing an inexhaustible supply of spaghetti. There are also five plates and five forks. When a philosopher feels the need for food, he enters the room and sits at his chair. He then picks up the fork to his left and then the fork to his right. If a fork is unavailable he waits patiently until it is. When he has acquired both forks he takes some spaghetti from the bowl and begins to eat. Once finished, he replaces his forks on the table and leaves. Now, consider the case when all five philosophers decide to eat at the same time. They sit in their chairs, take the fork to their left and are unable to proceed as the fork to their right is already taken. The philosophers are in a state of deadlock and will eventually starve to death.

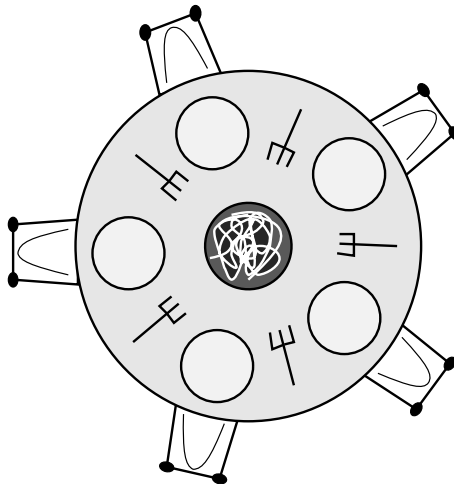


Figure 3.9: Deadlock: The dining philosophers

CSP is capable of identifying such deadlock scenarios and is therefore able to prove that a particular configuration of concurrent processes is deadlock free.

As already mentioned, the Transputer was designed around the concept of CSP. The Trans-

puter contains a scheduler for executing multiple processes on a single processor. It also has four links which enable networks of Transputers to be easily constructed. Communication between processes whether on the same or on different processors is efficiently implemented in hardware.

3.3.3 Parallel C

When the Transputer was released in 1980's, Inmos provided a new language, called OCCAM, for programming it. This language was based on CSP, containing constructs for launching parallel processes and for communication between processes. Inmos have also produced a version of the C programming language which includes extensions for handling parallel processes and communication. Inmos Parallel C, as it is called, has been used to develop both the vision applications described in this thesis. However, Parallel C is not a true implementation of CSP, some of the main differences being that CSP does not allow global data, processes may not be recursive and parameters must be passed by value and not reference (a consequence of there being no global data). The reason for these restrictions in CSP is to avoid conflicts when accessing data. Consider the case of two parallel processes simultaneously writing the same global variable. Obviously such a scenario would cause problems. To avoid these data access conflicts the programmer must adhere to a set of rules [63, 64]. Three possibilities are outlined below:

1. All C functions are implemented as processes which may not be recursive or share global data. All parameters are passed by value through channels.
2. Each parallel operation is written as a self-contained C module containing only one process. All functions called by the process and any global data is private to that program module. Function parameters may be passed by reference but data must be passed by value along channels. Function libraries may be used but each C module must have its own copy of any common libraries.
3. Global data is made private to a single process which will read or write to the data in response to messages from other processes. Functions with local, static data must be implemented as processes. All other functions may receive parameters by reference.

Parameters may be passed by reference over a channel to one other process. In doing so, the sending process relinquishes the right to access the data which is now the property of the receiving process.

The first option is the closest to pure CSP but is very restrictive. The second option requires each C module to have its own copy of any common libraries. This can only be achieved if there is only one C module per processor. The third option allows the use of libraries, providing the functions therein adhere to the rules, and is the option adopted in both the applications in this thesis.

3.4 The Image Processing Pipeline

A parallel solution to a problem may be formulated using the techniques discussed in the previous sections. However, most image processing problems consist not of one problem but of several sub-problems, or stages. Figure 3.10 shows the stages in a typical image processing system. A parallel solution to each of these stages may be obtained using the techniques from above. However, the separate stages may also execute in parallel if they are arranged into a *pipeline*. Figure 3.11 shows such an image processing pipeline. At time t , stage 1 is processing image $i + 2$, stage 2 image $i + 1$ and stage 3 image i . At time $t + 1$ a new image enters the pipeline as input to stage 1 and the output from each stage is passed on to the input of the next such that stage 1 is processing image $i + 3$, stage 2 image $i + 2$ and stage 3 image $i + 1$. The rate at which images pass through the pipeline is governed by the time taken by the slowest stage.

The vision hardware is perfect for the construction of image processing pipelines (Figure 3.12). The digitiser module supplies a stream of images for input to the pipeline. Each stage of the pipeline consists of a general function module or a framestore module. Output from the pipeline is fed back to the digitiser module for display on a monitor and/or to the input/output module as required.

Initially, each stage may consist of a single processor. If it becomes apparent that one stage is forming a bottleneck then it may be decomposed using the techniques above for

implementation on more than one processor.

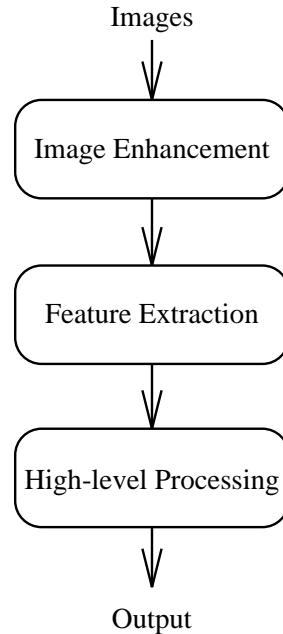


Figure 3.10: The multiple stages of a typical image processing system

3.5 Summary

The general purpose, Transputer based vision hardware provides a modular framework for the implementation of multi-stage image processing applications. The hardware is ideal for the construction of an image processing pipeline which allows each stage of a multi-stage problem to operate concurrently. For real-time operation it may be necessary to implement some stages on more than one processor.

CSP theory provides a mathematical tool for specifying and analysing the behaviour of concurrent systems. The theory may therefore be used for the detection of deadlock. The Transputer is designed around the CSP model and provides an efficient implementation of processes and channels.

Both the number-plate recognition and generic road-traffic monitoring sensor were formulated into multi-stage problems, each stage being implemented as a communicating sequen-

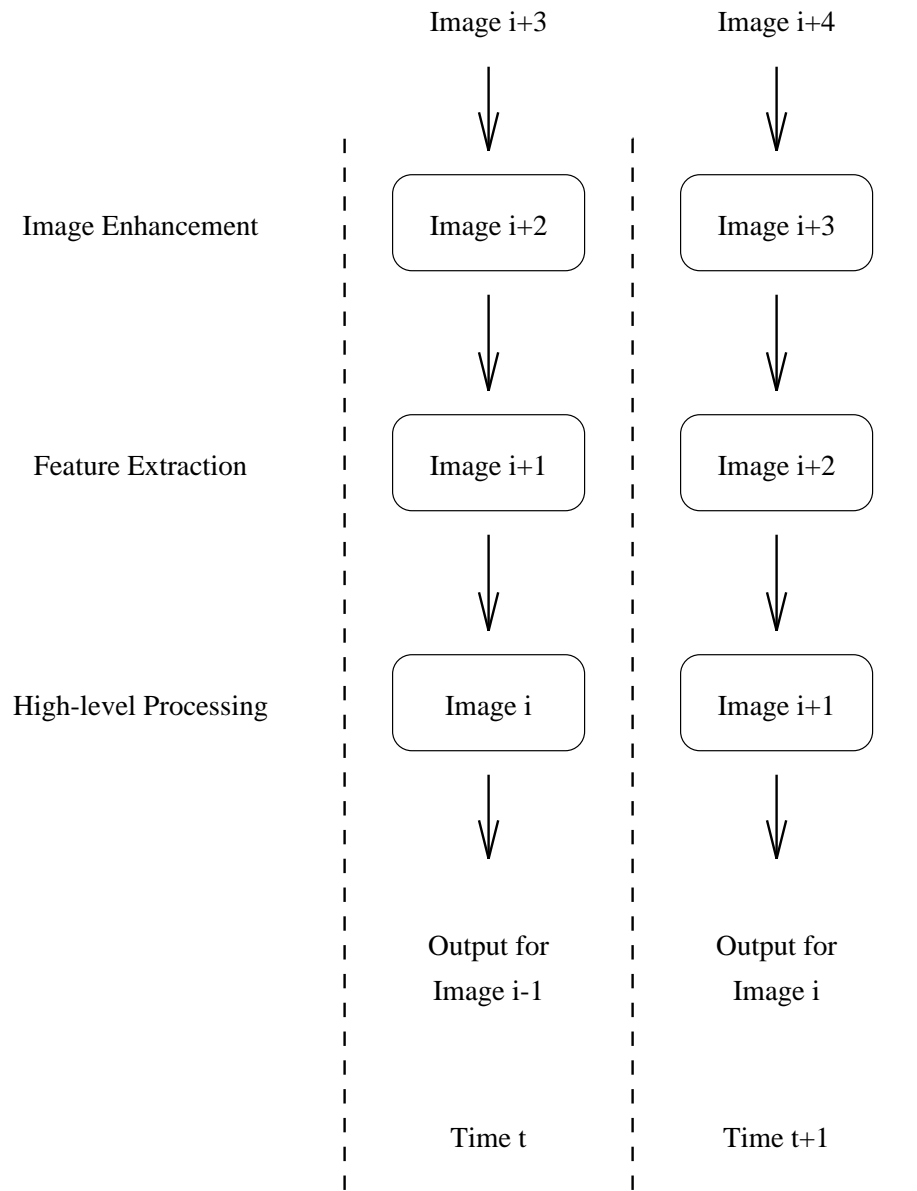


Figure 3.11: The image processing pipeline

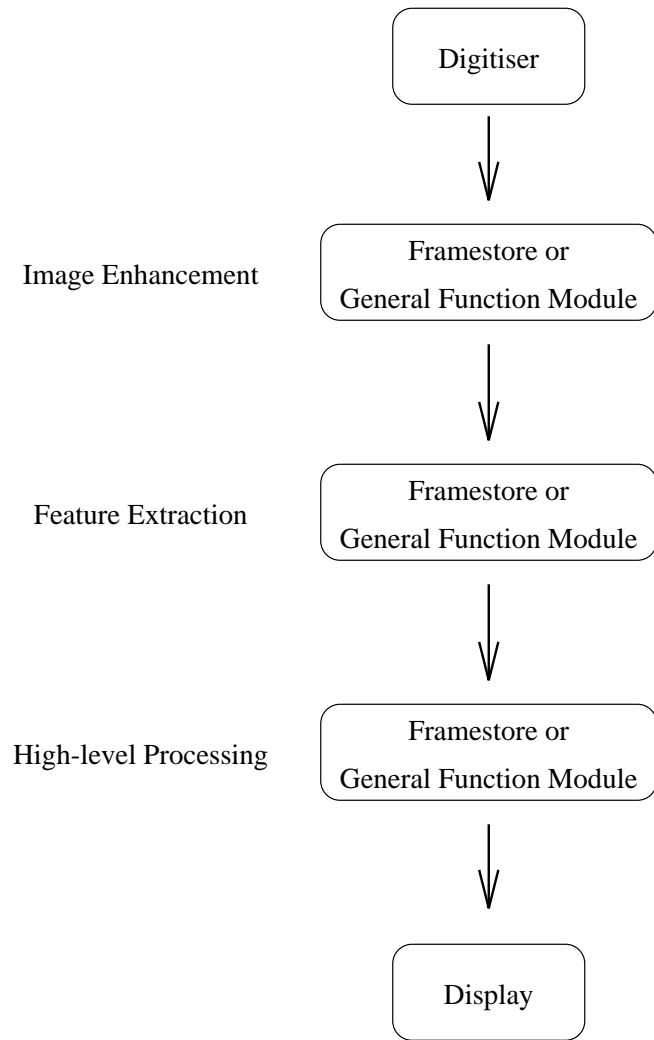


Figure 3.12: The vision hardware configured in an image processing pipeline

tial process written in Inmos Parallel C. For each application, the processes were distributed over a number of processors arranged in an image processing pipeline. Such a pipeline can result in significant latency between an image entering the pipeline and the result emerging but this is of no consequence provided the rate at which images are processed is sufficient. The video bus on the image processing modules was used to transfer image data between processes leaving the transputer links free for the passing of all other data. To avoid conflicts when accessing common data, the rules in alternative 3, Section 3 were adhered to when implementing the systems.

Chapter 4

A Number-plate Recognition System

4.1 Introduction

This chapter presents the number-plate recognition system. The work builds upon a prototype number-plate recognition system that has been developed in the ACRC at Bristol [65, 66, 22, 67]. The prototype performs very well in suitable conditions but when it is presented with more typical images its performance is significantly degraded. This chapter describes the new system, highlighting the completely new algorithms which have been developed to overcome the problems of the prototype.

Decomposition of the system into a number of communicating sequential processes and their mapping onto the Transputer based vision hardware is discussed.

The system components have been subjected to a number of tests. Details of these tests and the results are presented at the end of the chapter.

4.2 The Number-Plate Recognition System

An overview of the number-plate recognition system can be seen in Figure 4.1. The video camera is positioned to survey a section of road. When a vehicle is in a suitable position on the road a trigger is generated which causes the system to grab an image from the camera. This image is scanned by the *finder* to locate the vehicle's number-plate in the image. Once found, the location of the number-plate is passed on to the *reader* which deciphers the number-plate. The deciphered number-plate characters are passed through the *syntax checker* which ensures the decoded plate adheres to user definable syntax rules. The *database* module contains a list of vehicles which are 'wanted', perhaps because they are stolen. If the plate is found in the database the *fax* module will issue a fax which contains an image of the vehicle and a close-up of the deciphered plate. This fax could be sent directly to the local police station alerting them that a stolen vehicle has just been spotted. The purpose of the *buffer* is to cope with the situation of a new trigger signal occurring before the system has finished processing the previous plate. This may occur if two or more vehicles pass in rapid succession.

The *console* handles communication with a terminal, providing a command line interface with a simple set of commands for configuring the system, downloading the database of wanted vehicles and defining syntax rules. The terminal can also be used for displaying a list of decoded plates.

Each of the components which make up the system will now be discussed in detail.

4.2.1 Trigger

The prototype system requires an external trigger signal. This is usually provided by a loop detector buried in the section of road surveyed by the camera. The improved system includes a new feature which enables it to automatically trigger itself without the need for an external signal. The automatic trigger continuously monitors the output of the video camera. When it sees a vehicle traversing a specified point in the image it generates a trigger which causes the system to store the current image and initiate the recognition process.

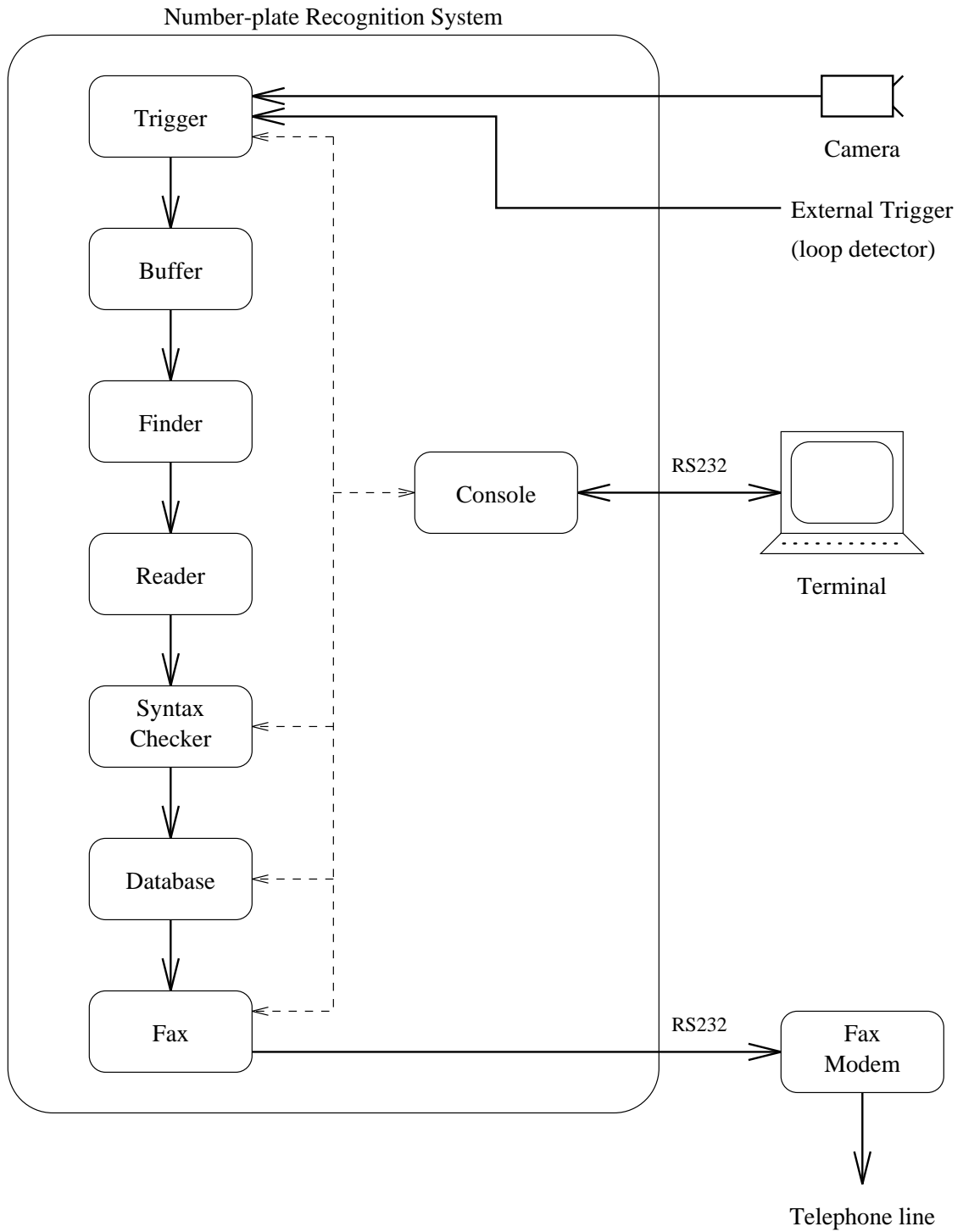


Figure 4.1: An overview of the number-plate recognition system

For this algorithm to generate accurate trigger signals for high speed vehicles it must operate at 25 frames per second. To achieve this the algorithm is very simple and it only operates on a small fraction of the pixels in the image. The automatic trigger employs two virtual *trip-wires* overlaid on the image of the scene (Figure 4.2). Each trip-wire may be in one of two states; broken or complete, depending on whether or not a vehicle is crossing the trip-wire. By using two trip-wires it is possible to distinguish between vehicles travelling towards or away from the camera. This is necessary because, although the trip-wire is positioned over only a single lane of traffic, the vehicles in the opposite lane may create shadows which cause the trip-wire to break. Wet roads are also a problem as the shiny road surface may catch reflections from the headlights of vehicles in the opposite lane, again causing the trip-wire to break. Both of these problems are eliminated by using two trip-wires to ensure the object breaking the trip-wire is travelling in the desired direction.

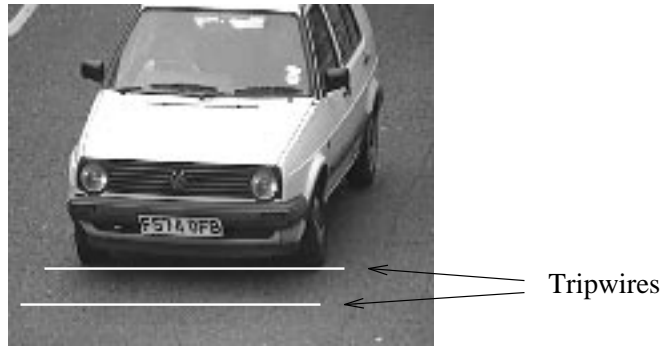
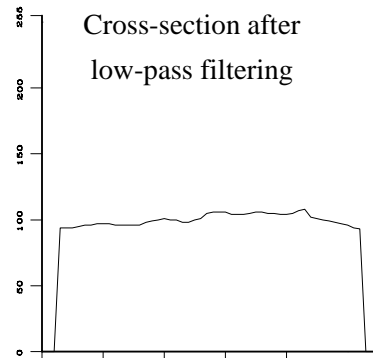
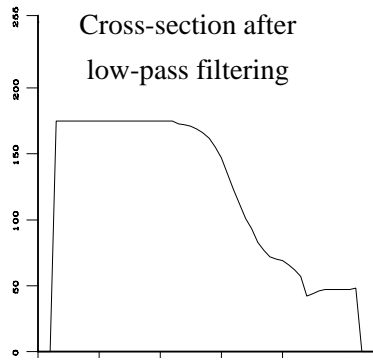
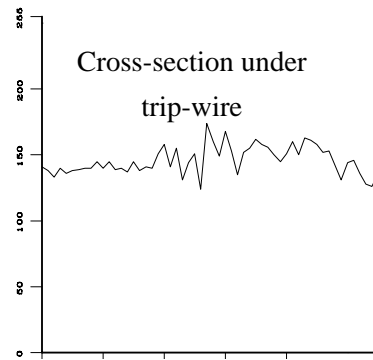
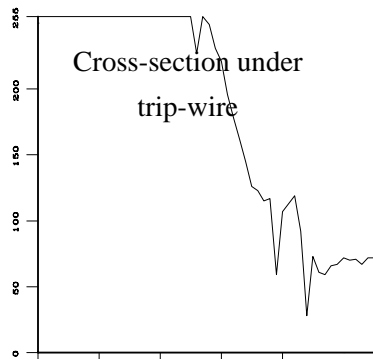
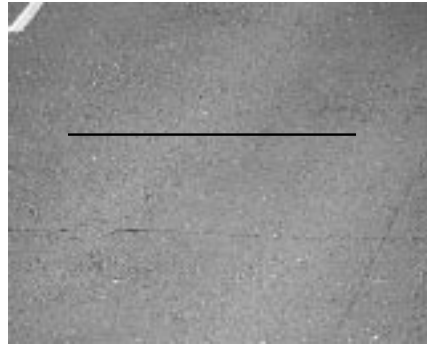


Figure 4.2: Virtual trip-wires overlaid on an image of a traffic scene

The state of each trip wire is determined by looking at the cross-section of the image under the trip-wire. The cross-section is first low-pass filtered to reduce the effects of noise and remove high frequency texture information which is often associated with the road surface. The median deviation of the cross-section is then calculated as $\sum_{i=1}^N |x_i - \bar{x}|$, where $x_i, i = 1, \dots, N$ are the filtered values of the N pixels under the cross-section. Figure 4.3 shows the cross-section of the image, the cross-section after low-pass filtering and the median deviation for both a broken and a complete trip-wire. It can be seen that when a vehicle is crossing the trip-wire the median deviation is considerably higher due to the greater amount of contrast associated with a vehicle compared to that of the road. The median deviation is therefore a suitable cue for deciding the state of the trip-wire.



Median deviation = 35

Median deviation = 4

Figure 4.3: Determining state of trip-wire from median deviation of low-pass filtered cross-section of image

The outputs of the two trip-wires are then used as inputs to a simple state machine (Figure 4.4). Only a vehicle travelling in the correct direction through the trip-wires will result in a trigger signal being generated.

4.2.2 Buffer

When a trigger signal is received, the buffer will place the newly grabbed image on the end of a queue. As soon as the finder is ready to accept a new image it is passed the image from the front of the queue. In this way the system can cope with several triggers in quick succession. The queue may contain a maximum of four images. If a trigger occurs and the queue is full, the newly grabbed image is discarded.

4.2.3 Finder

The prototype system located plates by first thresholding the incoming image to produce a binary image in which the number-plate characters are black and the plate background is white. The binary image is then scanned from bottom to top. When a black region is encountered it is traced around in order to ascertain its height and width. If the dimensions and aspect ratio of the black region are suitable it is labelled as a possible number-plate character. When a number of such possible characters are found in a horizontal row it is assumed the number-plate has been found and its location is passed on to the reader. If no plate has been found after scanning the entire image, a new threshold value is chosen and the process repeated. If several threshold values have been tried and the plate has still not been found then it is assumed that the image does not contain a plate.

This procedure works well provided the following conditions hold:

- The number-plate characters can be successfully segmented by thresholding. This requires the image of the number-plate to be of good contrast and the lighting conditions across the plate to be uniform.
- The characters of the number-plate must not appear to be touching. If they do touch

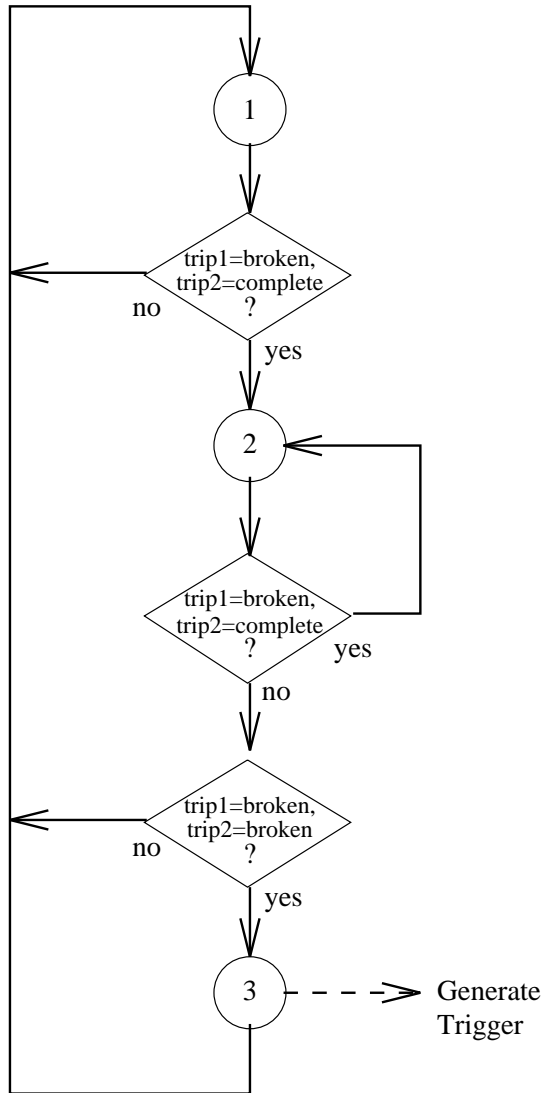


Figure 4.4: State machine used in the automatic trigger

then one or more of the thresholded characters may become merged to its neighbour. This will mean that their dimensions will be incorrect for them to be considered as possible characters and the plate will not be found. Touching characters can only be avoided by ensuring that the number-plate occupies a large enough portion of the image (at least 25% of the image width).

- The number-plate must be free from cracks, dirt, mounting bolts or other imperfections which will cause characters to be deformed or joined when thresholded.

Guaranteeing the satisfaction of these conditions is impossible as the only one which can be controlled is the size of the plate in the image. There is a limit to how big the plate can be made as the more the camera zooms in, the greater the chance of losing the plate off the edge of the image due to variations in vehicle position on the road.

It is the segmentation of characters via thresholding that necessitates the above conditions. If the thresholding process can be avoided then the conditions can be dropped and the system will be able to locate plates more robustly in 'real' images. Therefore, the current number-plate system employs a new algorithm which does not rely on thresholding or any other form of segmentation to extract the individual characters of the plate.

The new algorithm is based on the fact that the image of a number-plate, even if degraded by the problematic artefacts above, will contain a significant number of vertical edges. Figure 4.5 shows a poor quality image of a number-plate together with a horizontal cross-section through the plate. Vertical edges are characterised by a large difference in the grey-level of adjacent pixels and are clearly visible as steps in the cross-section of the plate. This provides the basis for the first stage of a number-plate location algorithm. This stage identifies all parts of the image which might possibly represent a number-plate and works as follows:

1. Take a horizontal cross-section through the image at every n^{th} line of the image. n is chosen to be less than the height of the smallest character that is expected to be encountered. This ensures that any number-plate in the image is 'hit' by at least one of the cross-sections.

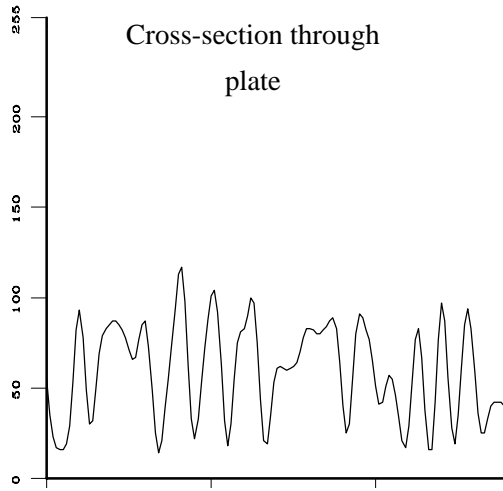
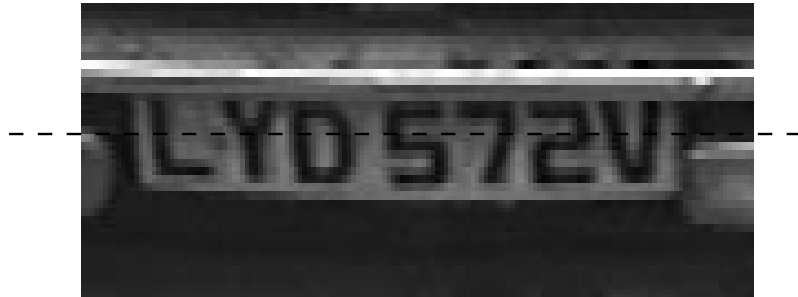


Figure 4.5: Image of a number-plate together with a cross-section through the plate

2. For every pixel in the current cross-section, if the difference in grey-level between the pixel and its neighbour is greater than some threshold T , then mark the pixel as a vertical edge.
3. Cluster all vertical edge points on the current cross-section into groups such that the horizontal distance between vertical edge points in the same group is less than the width of the largest character expected to be encountered. This works on the assumption that every character in the plate will produce at least one vertical edge and that the distance between any two characters in the plate will not be more than the width of a character. So, if the distance between two vertical edge points is more than the width of a character it is assumed that the points must be from different possible plates.
4. Merge clusters which are vertically adjacent into a single cluster.

Executing this algorithm will yield a number of clusters, where each cluster represents a possible number-plate. The positions of the edge points within each cluster determine the horizontal position and left and right extremes of the possible plate. A list of these possible plates is passed on to the next stage of the algorithm.

The second stage of the algorithm attempts to find the top, bottom and angle of tilt of the possible plate. In the process it is able to reject a large proportion of the possible plates. This stage of the algorithm works as follows:

1. Take the horizontal line between the left and right extremes of the possible plate and divide it into a number of intervals. For each interval, find an estimate for the position of the top and bottom of the plate. This is done by placing a horizontal bar over the current interval of the horizontal line and counting how many vertical edges there are in the image under the bar. The bar is then slid vertically upwards a pixel at a time and the number of edges counted at each point. If the bar is sliding over a number-plate character, then when the bar passes beyond the top of the character the number of vertical edges will dramatically drop (Figure 4.6). The bar is slid no further when the number of edges reaches zero or the distance between the bar and the original horizontal line is equal to the height of the largest character expected to

be encountered. So, as the bar is slid upwards, the point which gave the least number of edges is used as the estimate for the top of the plate. The same procedure is used to estimate the bottom of the plate by sliding the bar downwards. Figure 4.7 shows an image of a number-plate together with the estimates for the top and bottom of the plate at each interval between its left and right extreme.

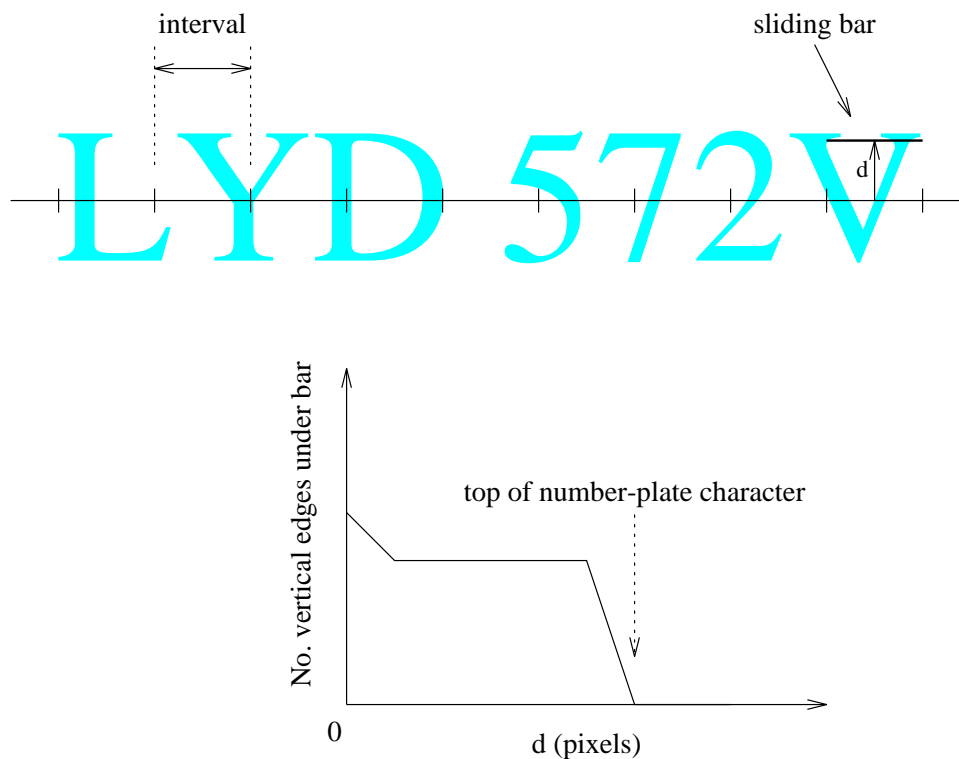


Figure 4.6: Estimating top of plate by counting the number of vertical edges under the sliding bar

2. Some of the estimates for the top and bottom of the plate will be wrong due to noise or image features other than number-plate characters. To find the plate's height and angle of tilt a simple model of a number-plate is fitted to the estimates for the top and bottom. The simple model consists of two parallel lines having parameters a_1 , a_2 and m (Figure 4.8). The normal procedure would be to fit the model to the data points using a least squares error:

$$e(a_1, a_2, m) = \sum_{i=1}^N \left((t_i^y - mt_i^x - a_1)^2 + (b_i^y - mb_i^x - a_2)^2 \right) \quad (4.1)$$

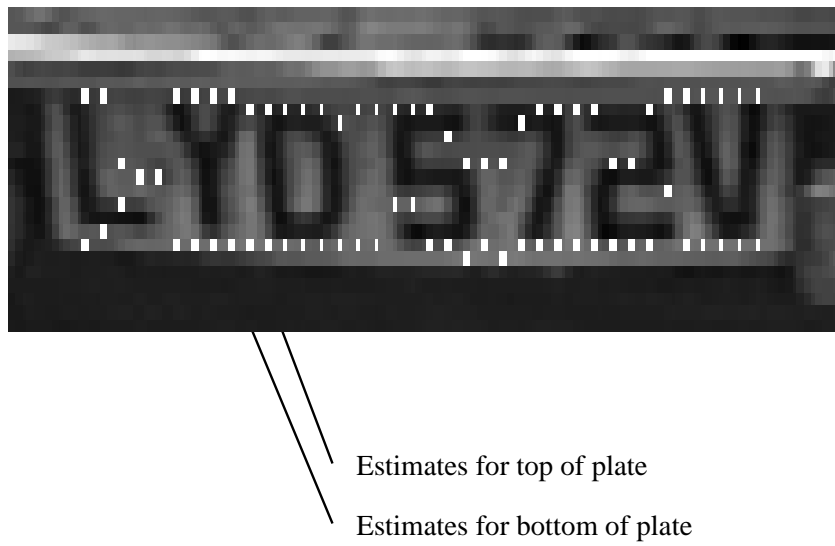


Figure 4.7: Estimates of the top and bottom of a number-plate

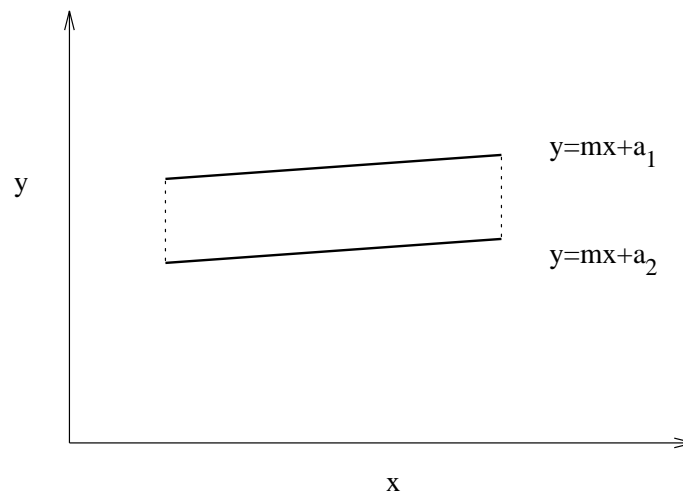


Figure 4.8: Simple model of a number-plate

Where, t_i^x, t_i^y are the x, y coordinates of the estimates for the top of the plate and b_i^x, b_i^y are the estimates for the bottom of the plate.

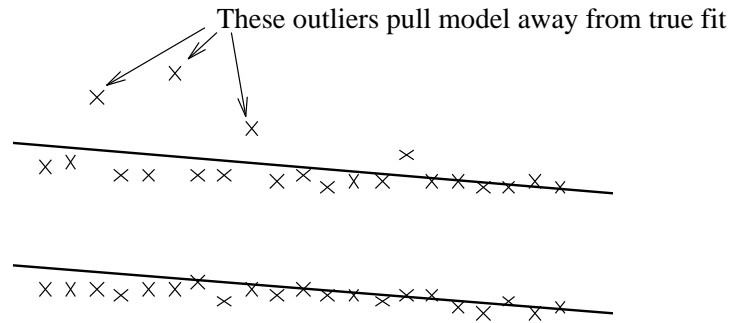


Figure 4.9: LSE: Poor fit of model due to outliers

The advantage of least squares is that the parameters of the model may be found via simple differentiation and solving the resulting set of linear equations. However, this simple error measure is not ideal as it only takes a few outliers in the data to cause a rather poor fit (Figure 4.9). To overcome this problem, the following error function was used:

$$s(a_1, a_2, m) = \sum_{i=1}^N \left(\frac{1}{1 + (t_i^y - mt_i^x - a_1)^2} + \frac{1}{1 + (b_i^y - mb_i^x - a_2)^2} \right) \quad (4.2)$$

As a high value of this function indicates a good fit it will be referred to as a score function rather than an error function. The important characteristic of this score function is that the further a point is away from the model, the less contribution it makes, therefore, outliers are unable to influence the fit of the model. Unfortunately, the nature of the function means that it may not be solved via simple differentiation as it may have several local maxima. The approach adopted for finding the global maximum was as follows:

For a particular value of m , try a range of values for a_1 and remember the one which gives the maximum value for s . Then try a range of values for a_2 , again remembering which gives the greatest value of s . This works because a_1 and a_2 are the parameters for two separate lines, therefore, they may be fitted independently. This combination of a_1, a_2 represents the greatest value for s which can be attained for this particular m . The value of m yielding the maximum for s is then located using a *Golden Section*

Search (Appendix D). These values of a_1 , a_2 and m give the best fit of the model of the number-plate to the estimates of the top and bottom of the possible plate.

3. A number of criteria can now be used to filter out possible plates that do not adequately fit the model. If the best fit of the model yields a value for s which is not sufficiently high, the plate is rejected. The difference between a_1 and a_2 gives the vertical height of the plate. If this does not fall within certain bounds, the plate is rejected. If m , the angle of tilt of the plate, does not fall within certain bounds the plate is rejected. Typically, only a couple of possible plates survive this filtering and are passed on to the reader.

Figure 4.10 shows several possible plates, the estimates for their top and bottom and the best fit of the model to these estimates. For each possible plate that satisfies the criteria, a bounding box specified by the coordinates of the top-left, top-right, bottom-left and bottom-right of the fitted number-plate model are passed on to the reader. Obviously, the bounding box may be inclined at an angle.

4.2.4 Reader

The prototype system deciphered characters with a template matching scheme. The bounding box of each possible character is scaled to be the same size as the stored templates. The possible character is identified by finding the template which has the smallest error when compared to the scaled character using a euclidean error function. The weak link in this method is the positioning of the bounding box. If the character was poorly segmented, perhaps due to a lack of contrast or dirt on the plate, then the bounding box may be incorrectly positioned either omitting part of the character or including some feature which is not a part of the character. If this happens, template matching has little chance of correctly identifying the character.

The new system overcomes this problem by avoiding the use of a bounding box for each character. In fact, it is unable to use a bounding box because the finder in the new system does not segment individual characters and therefore is unable to supply details of their bounding boxes. The new finder does, however, provide a bounding box for the whole plate.



Figure 4.10: Several possible plates (a), the estimates for their tops and bottoms (b) and the best fits of the number-plate model (c)

As the bounding box may be at an angle it is first skewed to make it horizontal. Skewing is a satisfactory approximation to rotation in this case as the permitted angle of rotation is very small, only a few degrees. This box is then scaled equally in each direction such that its height is the same as the height of the stored templates and the aspect ratio of the plate, and therefore the characters within the plate is maintained. Appendix A contains a description of the scaling process. Three different approaches were then tried for recognising the characters within the scaled plate:

- Template matching
- A large neural network having n outputs, one for each of the n possible characters
- n small networks, each having one output. One network is trained to recognise each of the n possible characters

Template matching is performed by sliding each template over the scaled plate, this time using a normalised correlation function for matching. Whenever a maximum in the correlation function is found, that position is noted as a possible position for that particular character, the value of the maximum giving the confidence in the match. The possible character is entered into a table. After matching all templates, the table has the form of Table 4.1. This table is passed on to the syntax checker which must choose letters and digits from the table such that the syntax rules are adhered to and the maximum overall confidence is achieved.

Neural networks are an alternative to template matching for character recognition. As previously mentioned, two different architectures have been investigated. Figure 4.11 shows the single, large network, while Figure 4.12 shows the multiple, small networks. Both types of network are used in exactly the same way. A window is slid over the scaled plate with the elements of the window providing the input values to the neural networks. The number of nodes in the input layer is therefore governed by the size of the window. In this case, an 8 by 8 window was used and so all the networks had 64 nodes in the input layer. The number of nodes in the hidden layer, h , must be determined by experimentation. For the large network, values of h between 5 and 36 were investigated, while for the small networks h was between 5 and 20.

		Character and associated confidence							
5 best matching characters	E	9	A	9	A	N	M	W	
	0.8	0.7	0.9	0.9	0.7	0.8	0.9	0.8	
	F		4	5	H	A	H	V	
	0.7		0.8	0.5	0.4	0.3	0.5	0.6	
	C				4	W		M	
0.5				0.3	0.1		0.2		
L									
0.4									
0.3									
0.2									
0.1									
x position of characters	120	128	136	144	152	160	168	169	

Table 4.1: Format of table produced by reader

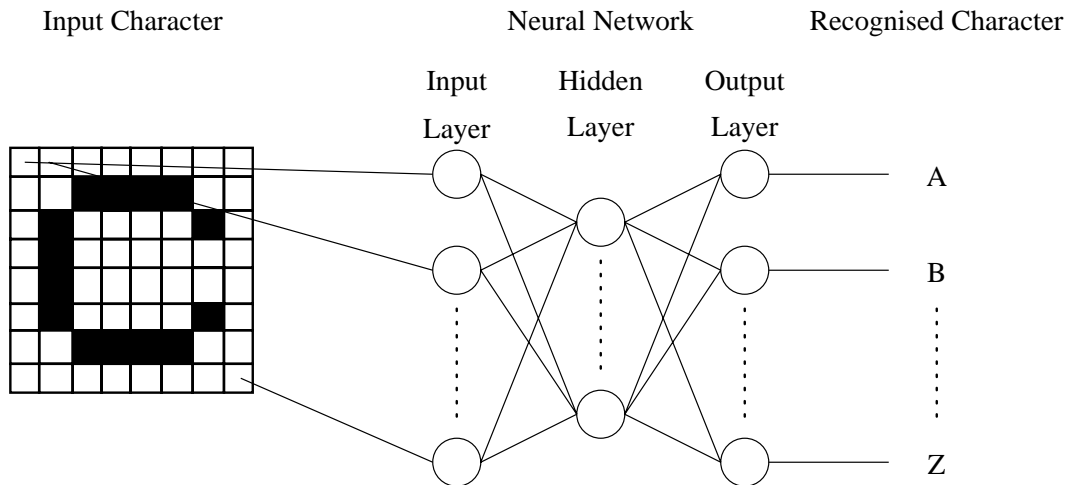


Figure 4.11: Large neural network for character recognition

Before the networks can be used for recognition, they must first be trained. The SNNS Neural Network Simulator Package was used for this [68]. The process requires a large number of training examples, each of which must have associated with it the desired output when the example is presented as input to the network. The examples were gathered by adapting the prototype number-plate system to find and classify all number-plate characters in an image and then storing a sub-image of each character to disk. In this way, 5000 classified images of number-plate characters were collected. Each of these had to be inspected manually to correct misclassifications and to remove examples that were not valid number-plate characters. The examples were then scaled to be 8 by 8 pixels using simple linear interpolation.

Pairs of characters were then randomly chosen from the 5000 examples. The pair of characters were placed next to each other and an 8 by 8 window randomly positioned within the 16 by 8 pixels of the pair. The contents of the window were then stored as a new example. This was done as the input to the networks will come from a window sliding over the scaled plate. Therefore, the network will frequently be presented with a portion of a pair of characters and the training examples must reflect this. This process was used to create a total of 10000 examples.

Normalisation was then performed by examining the grey-level histogram of each example. Figure 4.13 shows a typical example together with its histogram. The value of min is chosen such that one sixth of the example's pixels have a value less than min while max is chosen such that one sixth of the pixels have a value greater than max . The normalised pixel values are then obtained using the transfer function in Figure 4.14. The 10000 examples were then divided into a *training set* for training the network and a *test set* for evaluating the network's performance.

The training process, which is performed by the SNNS package, begins by randomising the weights within each node of the network. Each example is then presented to the network and the difference between the actual network output and the desired output for each example is summed and used as the network error. Training the network amounts to adjusting the weights in order to minimise the error. Several iterative methods are available for doing this such as back propagation and scaled conjugate gradient descent, as used in this case.

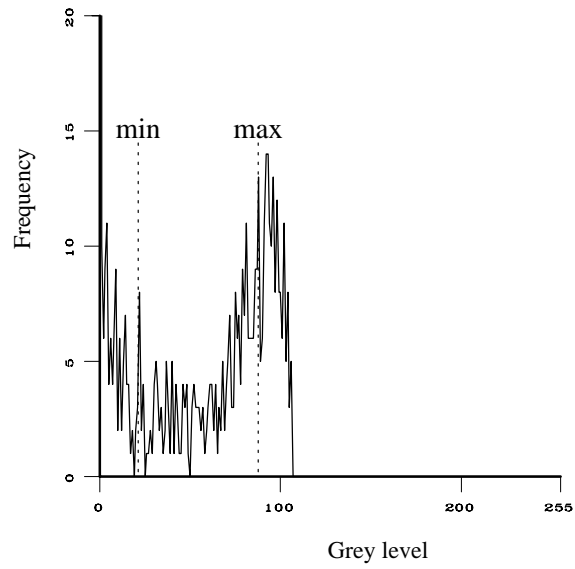
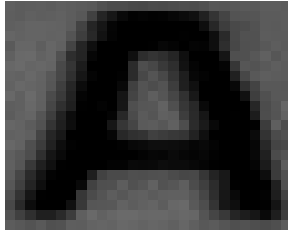


Figure 4.13: Typical number-plate character and its histogram

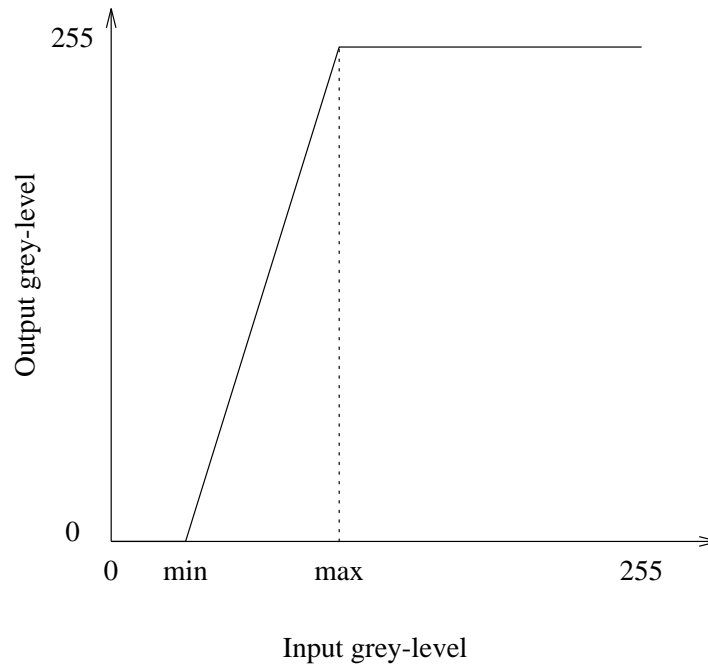


Figure 4.14: Transfer function used to normalise grey-levels of number-plate characters

Network training usually ceases when the error can be minimised no further. However, such a stopping criteria may result in the network becoming over-trained. This means that it has learnt to classify the training examples extremely accurately but it has lost its ability to generalise and is nowhere near as good at classifying previously unseen examples. The solution to this problem is to divide the 10000 examples into three sets; a training set, a test set and a *validation* set. The training set is used to train the network but after each iteration the network is shown the validation set and the network error calculated. When the network begins to overlearn, the error for the validation set will begin to rise (Figure 4.15). At this point training is stopped.

Training the network is essentially an optimisation problem, the function to be optimised being the network error as a function of the network weights. As this function will typically have many local minima it is impossible to guarantee that the trained network has found the global minimum. To increase the chances of finding the global minimum it is usual to train the network several times, each time starting with a different set of random weights.

The trained networks produced by trying each of a range of nodes in the hidden layer from

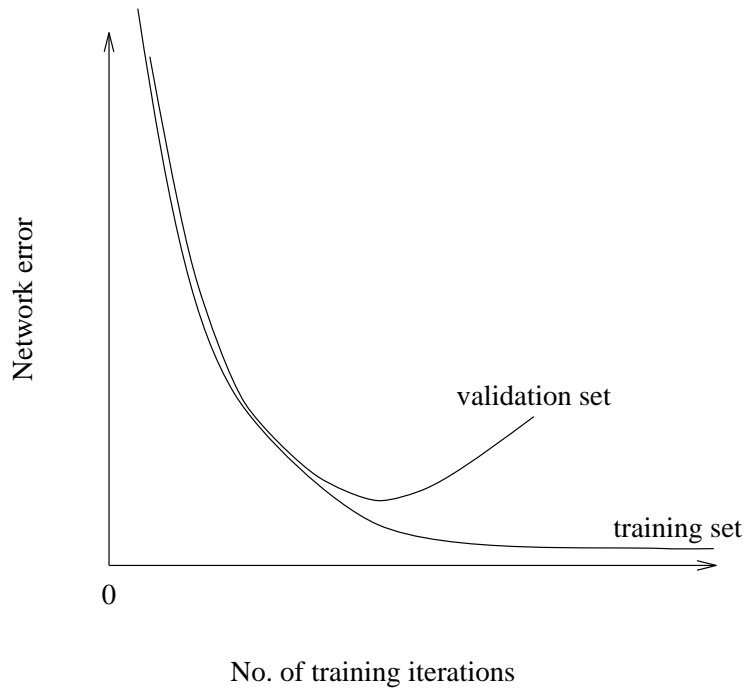


Figure 4.15: Network error for training and validation sets against number of iterations

several different random starting positions are evaluated using the test set. Each example in the test set is presented to the network and the network output is compared to the desired output. If the difference is less than some threshold, the network is considered to have classified that example correctly. The percentage of correctly classified examples can then be calculated for each network. The network which performs the best is selected for use in the number-plate recognition system.

As the window is slid over the scaled plate, the n network outputs (whether from the large network or the n small networks) represent a confidence in the match for the n possible characters (Figure 4.16). When a maximum in the confidence of a character occurs, the character is entered into the table, as in Table 4.1. As before, the table is passed on to the syntax checker to extract the deciphered plate.

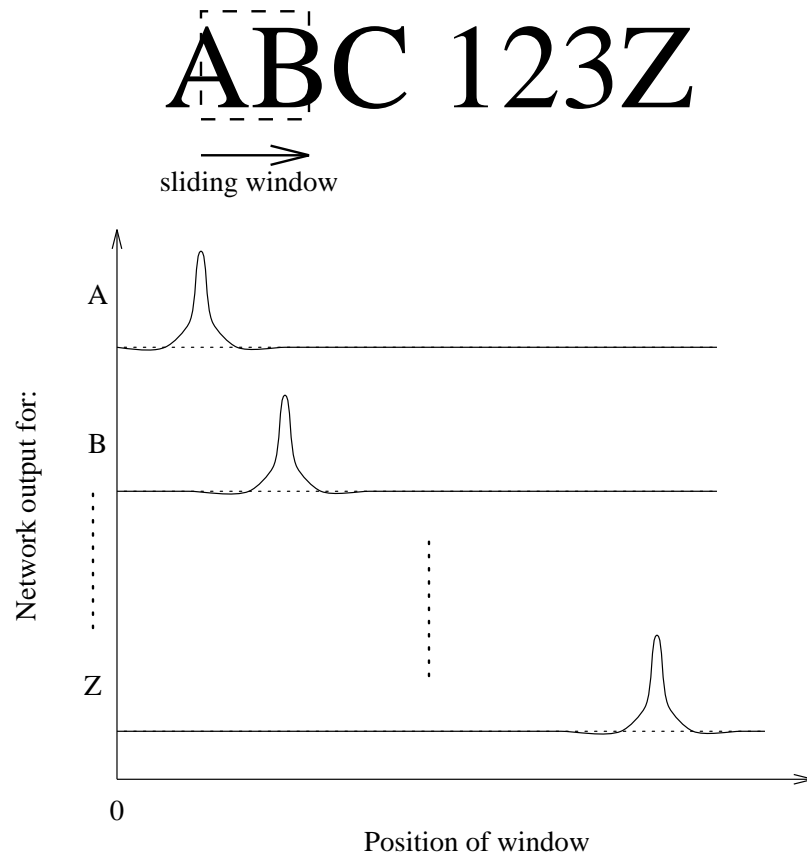


Figure 4.16: n network outputs giving confidence in match of n possible characters

4.2.5 Syntax Checker

An extremely crude syntax forcing algorithm was included in the prototype system but its design and implementation was so flawed that it often did more harm than good. A new algorithm was designed and implemented for the current system. It has the following features:

- User definable syntax rules.
- The output from the reader is searched for all possible combinations which satisfy the syntax rules. Each plate which satisfies the rules is assigned a confidence based on the confidence of each character, the uniformity of spacing between characters and the number of characters. The plate with the highest confidence is selected as the recognised plate.

Syntax checking is usually performed as part of the compilation process for computer programming languages. In such applications, the syntax checker is usually implemented as an *automaton* or a *finite state machine*. Figure 4.17 shows such a setup. The input string is scanned once and the output indicates whether or not it satisfies the syntax rules (or grammar). With such an implementation, the grammar is usually *hard-wired*. This means that a change to the grammar requires a new implementation. As the number-plate system requires user definable syntax rules such an approach is not suitable and the following *tree search* approach was adopted.

Firstly, a notation for defining the syntax rules was defined:

Set of valid symbols = $\{A, B, C, D, E, M, N, -, [,], \{, |, \}\}$, where,

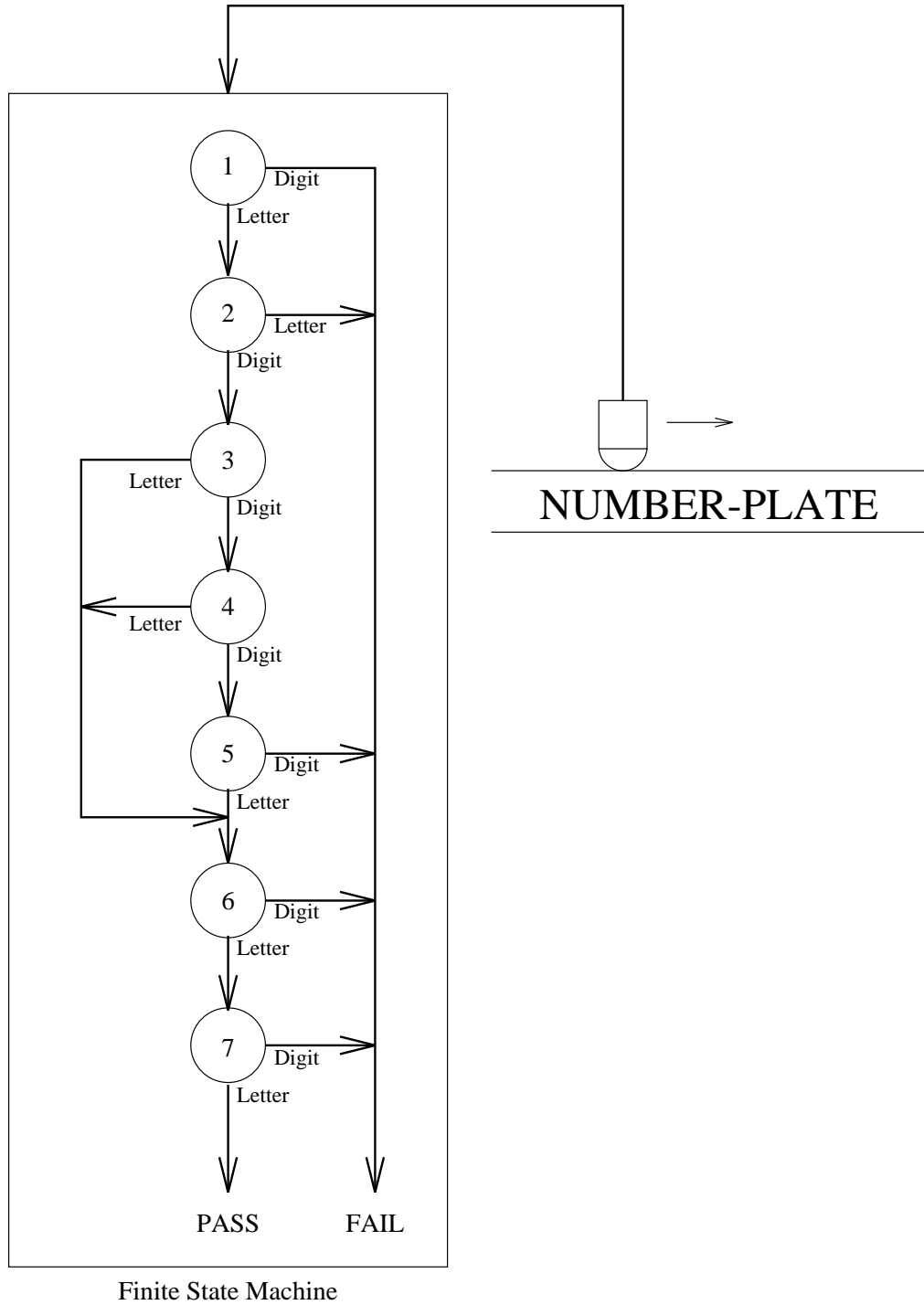


Figure 4.17: A possible implementation of a syntax checker using a finite state machine

- A = All 26 alphabetic characters, A, . . . , Z.
 B = All alphabetic characters, except I.
 C = All alphabetic characters, except I and Z.
 D = All alphabetic characters, except I,Z and Q.
 E = All alphabetic characters, except I,Z and O.
 N = All 10 numeric characters, 0, . . . , 9.
 M = Numeric characters, 1, . . . , 9.
 $_$ = An extra large gap may occur between the adjacent characters.
 $[. . .]$ = Contents of square brackets are optional.
 $\{. . . | . . . | . . .\}$ = Curly braces contain a list of at least 2 mutually exclusive options separated by vertical bars.

The majority of British number-plates may then be represented by two syntax rules:

1. $E\{M|MN|MNN\}_DDD$.
2. $DDD_{\{M|MN|MNN\}}E$.

These rules, together with the table produced by the reader (Table 4.1) comprise the input to the syntax checker. The syntax checker is implemented as a recursive algorithm which begins by expanding the syntax rules into all possible combinations containing no square or curly braces:

$$\begin{aligned}
 E\{M|MN|MNN\}_DDD &\Rightarrow \\
 &EM_DDD \\
 &EMN_DDD \\
 &EMNN_DDD \\
 DDD_{\{M|MN|MNN\}}E &\Rightarrow \\
 &DDD_ME \\
 &DDD_MNE \\
 &DDD_MNNE
 \end{aligned}$$

Each of the expanded syntax rules is then used to create a tree which represents a search through the possible characters in the table provided by the reader. In essence, the first

element of the syntax rule is examined. This defines a set of possible characters which it can match. The first column in the table from the reader is then searched for the character with the greatest confidence which is a member of that set. Next, the second element in the syntax rule is matched against the characters in the second column in the table, and so on until all elements of the syntax rule have been matched to a possible character in the table. At this point, the overall confidence of the matched characters is calculated and used as the plate confidence. When all expanded syntax rules have been examined the plate with the greatest overall confidence is chosen as the deciphered plate.

In fact, the process is more complicated than this as the table may well contain columns of possible characters which have arisen due to the reader incorrectly identifying characters. During the exploration of the search tree, the syntax checker must therefore try omitting columns from the table.

At each node of the tree, the x positions of the matched characters are checked. If two characters are found to be overlapping or if the distance between them exceeds a predefined limit, the sub-tree below that node is pruned. This ensures that ridiculous plates are not accepted while at the same time, greatly speeding up the search.

The final task of the syntax checker is, if necessary, to insert *wild cards* in the deciphered plate. Some characters, such as 'D' and 'O', '8' and 'B', can appear very similar and are therefore difficult for the reader to discriminate between. In most cases the syntax checker is able to resolve these ambiguities as the context of the character dictates whether it must be a letter or a number. The exception to this is the case of 'D' and 'O'. To cope with this the special symbol '%' is used as a wildcard to represent either 'D' or 'O'. For example, if the syntax checker produces 'F752HYD' as the deciphered plate, special attention must be paid to the 'D'. The syntax checker examines the output from the reader and looks at the column containing the 'D'. If the column also contains an 'O' and the difference between the confidence of the 'D' and the confidence of the 'O' is below a threshold, then the 'D' is substituted with the wild-card symbol, '%'.

4.2.6 Number-plate Database

The database may hold up to 5000 number-plates. Whenever a number-plate is deciphered it is searched for in the database. The search takes into account wild-cards, so the deciphered plate 'F725HY%' would match both 'F725HYD' and 'F725HYO'. If a match is found and fax sending is enabled, a fax is issued.

4.2.7 Fax Generation

The destination of the fax is programmable by the user. The fax contains details of the site name and location of the installation, the time and date, the deciphered plate and a list of up to four plates from the database which match the deciphered plate. The fax also contains an image of the vehicle, together with a close of the number-plate. An example of a typical fax can be seen in Figure 4.18. As the majority of fax machines are only capable of producing binary images, the grey-scale image of the vehicle and the number-plate must first be dithered.

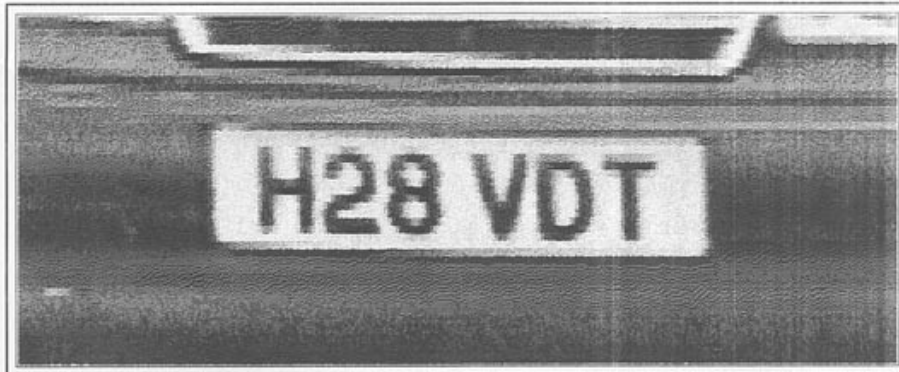
4.2.8 Console

The number-plate system may be connected to a laptop computer or terminal via one of the RS232 ports. The console provides a simple command-line interface with such a terminal. The command set provides the following functions:

- Setting of time, date, site name and location
- Uploading database of number-plates and list of syntax rules
- Configuring RS232 ports; baud rate, parity, etc.
- Setting system parameters; enable/disable fax, destination of fax, enable/disable on-site printout (OSP), external/automatic trigger and position of trip-wires for automatic trigger

site:Park Row location:Bristol +21,9545206 time:17:45:25 date:97- 8- 5
NUMBER PLATE ALERT **H28VDT**

number of matches:0



GOLDEN RIVER
TRAFFIC
World Leaders in Traffic Instrumentation

for service and maintenance
phone: 0869 240400
fax: 0869 246858

Figure 4.18: A typical fax generated by the number-plate recognition system

When the OSP is enabled the console will display on the terminal a list of all deciphered plates. An asterisk adjacent to a deciphered plate indicates that it was matched to one or more of the plates in the database.

4.3 Implementation on Vision Hardware

The number-plate system comprises three of the vision hardware modules; a digitiser, a framestore and an input/output/boot module. When connected together, these three modules provide three Transputers for implementing the system.

The number-plate recognition system is clearly a multi-stage problem and is thus ideally suited for formulation as an image processing pipeline. The buffer provides a stream of images to the finder at the head of the pipeline. The list of possible plates from the finder is passed to the reader in the next stage of the pipeline. The table of possible characters produced by the reader is then passed to the syntax checker. The final stages of the pipeline implement the database and perform fax generation if necessary. The software is therefore structured such that each of the system components is implemented as a separate CSP, written in Inmos C. By placing the finder, reader, syntax checker, database and fax processes on separate processors an image processing pipeline is easily constructed. In fact, sufficient performance is achieved with the finder, reader and syntax checker processes all placed on a single Transputer. Figure 4.19 shows the distribution of processes over the available processors. It may be noted that none of the processes are placed on the digitiser module. This is because the Transputer on the digitiser module only has 4K of memory which is completely used up by the low-level functions for controlling the hardware on the board. The digitiser is controlled by sending it messages along one of its Transputer links.

The finder, reader and syntax checker are currently all placed on the same Transputer. The structure of the software means that if greater performance is required it is merely a matter of plugging in more framestore modules and re-distributing the processes over the extra processors. By placing the finder, reader, syntax checker, database and fax processes on separate processors an extended image processing pipeline is created, yielding a significant increase in performance.

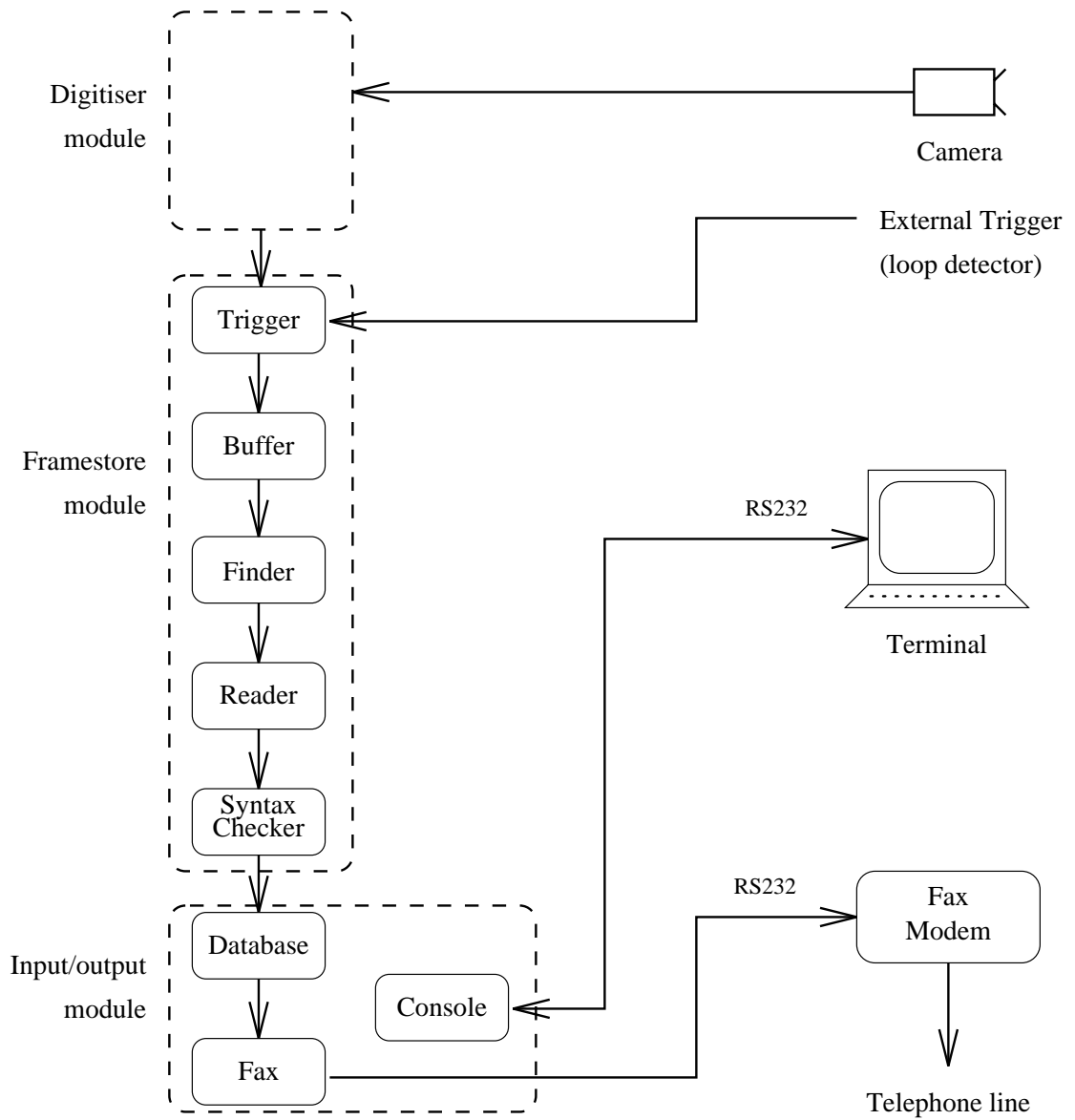


Figure 4.19: Distribution of number-plate system processes onto vision hardware

4.4 Results

When assessing the system's performance there are three aspects to consider:

- Performance of the Automatic Trigger
- Performance of the Finder
- Performance of the Reader and Syntax Checker

In order to measure the system performance within these three areas a camera was setup at a height of about 10 meters above the ground, looking towards the front of vehicles as they approached the camera along a city centre road. The vehicles were travelling at speeds up to 40mph. The weather conditions were dry and lighting varied from strong sunlight to overcast skies. The camera was equipped with an autoiris and its shutter speed set at $\frac{1}{1000}$ second. A total of six hours of video was recorded at different times in the day from mid-morning to early evening. From this video, still-images of three thousand vehicles were digitised and stored on a SCSI disk. A selection of these three thousand images are shown in Figure 4.20. It can be seen that the number-plates typically occupy about 15% of the image width. Some of the plates are mounted at a significant angle, some are dirty, cracked or broken and some contain strong shadows. For each of these still images the vehicle's number-plate was manually deciphered and stored in a database.

4.4.1 Performance of Automatic Trigger

To test the operation of the trigger, the six hours of video were played through the number-plate recognition system. A human observer then monitored the video sequence together with the output from the system keeping a tally of the following:

- The number of correct triggers, i.e. the system generated a trigger when a vehicle had crossed the virtual trip-wires in the correct direction.
- The number of false positives, i.e. the system generated an erroneous trigger when no vehicle had crossed the virtual trip-wires.



Figure 4.20: A selection of the three thousand number-plate images

- The number of false negatives, i.e. the system had not generated a trigger even though a vehicle had crossed the virtual trip-wires.

The tape actually contained a total of 4357 vehicles (although only 3000 were stored on the SCSI disk) and the system generated a total of 4195 triggers. In Table 4.2 the number of correct triggers and the number of false negatives are also expressed as a percentage of the number of vehicles on the video, while the number of false positives is also expressed as a percentage of the total number of triggers generated by the system during the test.

Correct Triggers	4124	94.7%
False Negatives	233	5.3%
False Positives	71	1.7%

Table 4.2: Performance of automatic trigger

4.4.2 Performance of Finder

The system was programmed to highlight the position of all possible plates located by the finder. The three thousand images on the SCSI disk were then processed by the system while a human observed whether the actual vehicle number-plate was included in the possible plates located by the finder. On average, the system located 1.10 possible plates per image and the actual plate was included in the possible plates in 99.17% of the three thousand examples.

4.4.3 Performance of Reader

In this experiment the three thousand images on the SCSI disk were all presented to the number-plate recognition system. For each image the plate, as deciphered by the system, was compared to the manually deciphered plate and the number of differing characters counted. In cases where the syntax checker was unable to find a valid plate within the possible characters provided by the reader, the system indicates that the plate has not been

read. This test was performed using template matching, n small neural networks and the single, large neural network. Results for these tests are shown in Table 4.3.

	Template Matching	n small neural networks	Single, large neural network
Plate completely correct	59.5%	69.4%	85.4%
1 character wrong	27.2%	8.6%	5.8%
2 characters wrong	7.1%	3.8%	1.8%
3 characters wrong	1.7%	2.6%	1.0%
4 characters wrong	0.9%	4.4%	1.2%
5 characters wrong	1.0%	1.4%	0.7%
6 characters wrong	0.3%	1.6%	0.5%
7 characters wrong	0.1%	1.5%	0.1%
Plate not read	2.3%	6.8%	3.5%

Table 4.3: Performance of reader

4.4.4 Execution times

The system was presented with the three thousand images on the SCSI disk. For each image the time taken to find the plate, read it and perform syntax checking was recorded. The total time to find, read and syntax check each plate (the pipeline latency) was also recorded. Over the three thousand images, the maximum, minimum and average values for these times were calculated. Table 4.4 shows these results for template matching while Table 4.5 and Table 4.6 contain the results for the n small neural networks and the single large network respectively. As the finder was identical for all three cases, the times for the finder are nigh on the same in all three tables.

The difference between the maximum and minimum for all these times is quite considerable so it is interesting to look at the actual distributions. Figure 4.21 shows the distribution for the finder. Figures 4.22, 4.23 and 4.24 show the distributions of the reader and syntax checker for the template matching, n small neural networks and single large neural network

respectively.

	Maximum	Minimum	Average
Finder	4.98	1.36	2.29
Reader	10.80	1.19	2.09
Syntax Checker	7.91	0.02	0.96
Pipeline latency	21.52	3.22	5.34

Table 4.4: Execution times for template matching (seconds)

	Maximum	Minimum	Average
Finder	4.96	1.36	2.28
Reader	20.37	2.56	3.64
Syntax Checker	912.49	0.02	2.75
Pipeline latency	926.54	4.37	8.67

Table 4.5: Execution times for n small neural networks (seconds)

	Maximum	Minimum	Average
Finder	4.98	1.36	2.29
Reader	3.93	0.51	0.71
Syntax Checker	18.74	0.02	0.18
Pipeline latency	23.12	2.03	3.18

Table 4.6: Execution times for single, large neural network (seconds)

4.5 Conclusions

The tests have shown the automatic trigger performs very satisfactorily, coping well with the varying lighting conditions. The small number of false positives are of little consequence as the time taken by the system to process an image containing no number-plate is very small and so the image is rapidly discarded. It would be difficult to reduce the number of false negatives as they were all caused by two or more vehicles driving very close behind

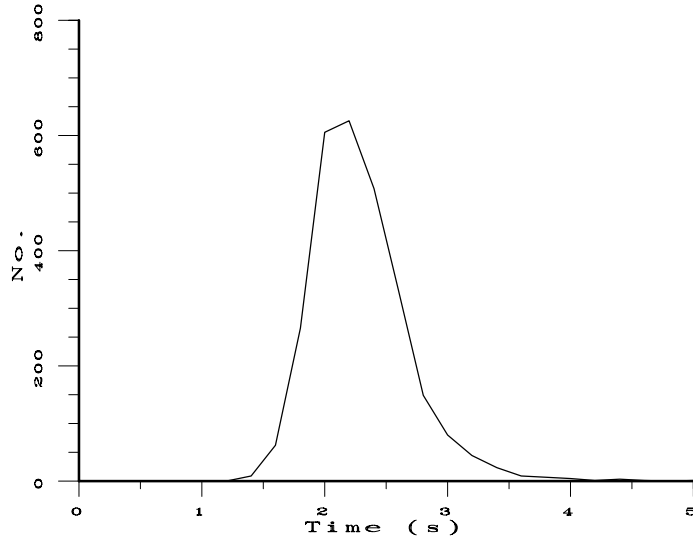


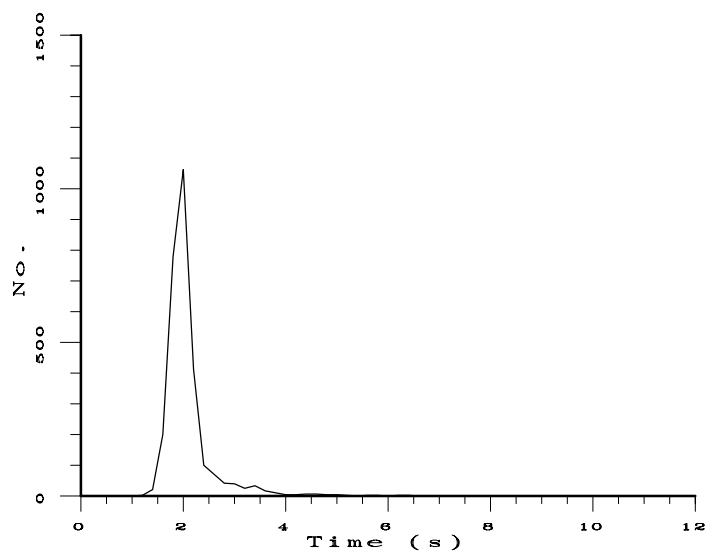
Figure 4.21: Distribution of times for the finder

each other. This results in the vehicles appearing to be joined and so a trigger is only generated for the first vehicle.

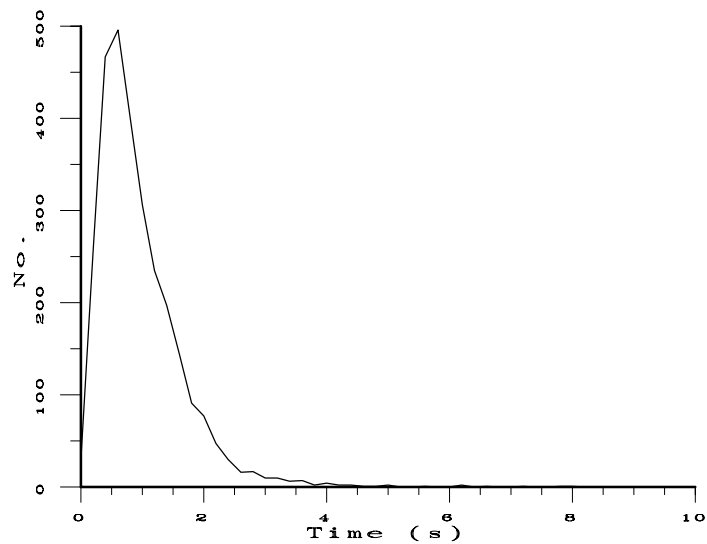
The performance of the finder was excellent, finding 99.17% of the number-plates in the test images. It achieved this despite many of the plates being at an angle, dirty, cracked, broken or partially/completely in shadow. I think there are few improvements to be made here.

With regard to the reader, the above results make it clear that the single large neural network gives the best performance, both in terms of recognition accuracy and execution times.

The times taken by the finder, reader and syntax checker are all closely related and highly dependent on the proportion of the image containing strong vertical edges (possible plates). The explanation for this is as follows: the time taken by the finder is governed by the number of possible plates it locates. For each region that is identified as a possible plate, estimates of the top and bottom of the possible plate have to be found and the number-plate model fitted to these estimates. So, the more possible plates in the image, the more work

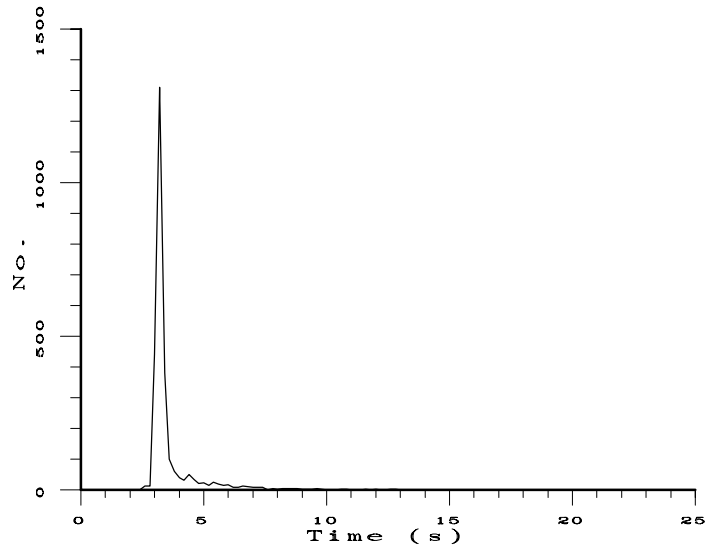


(a)

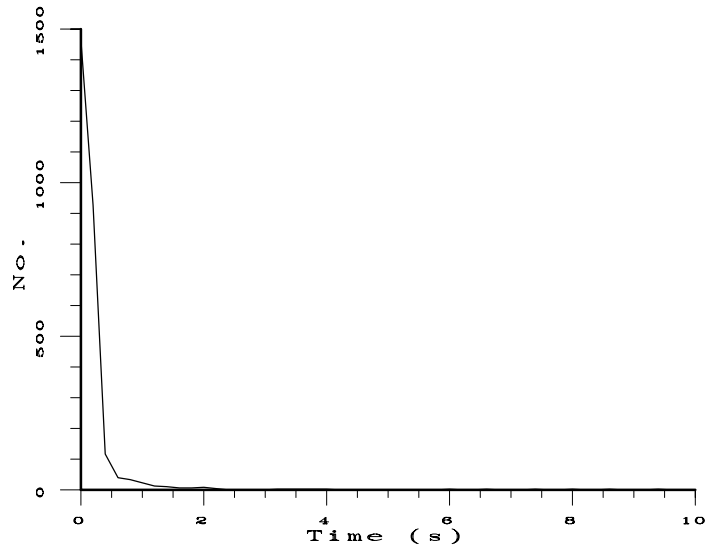


(b)

Figure 4.22: Distribution of times of the reader (a) and syntax checker (b) for template matching

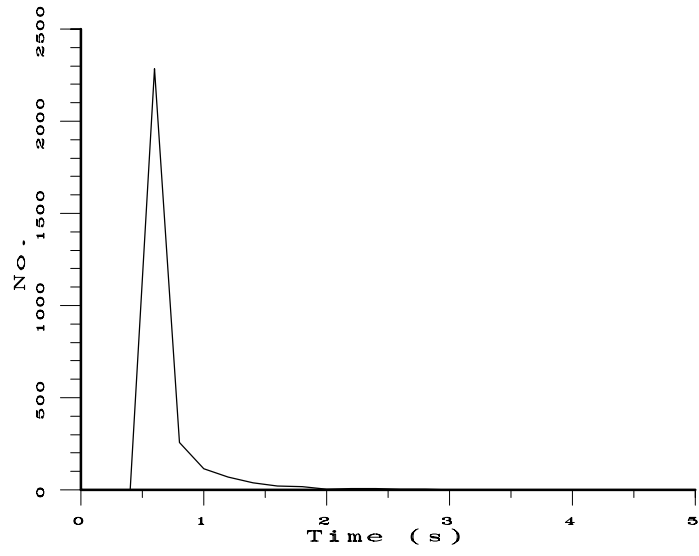


(a)

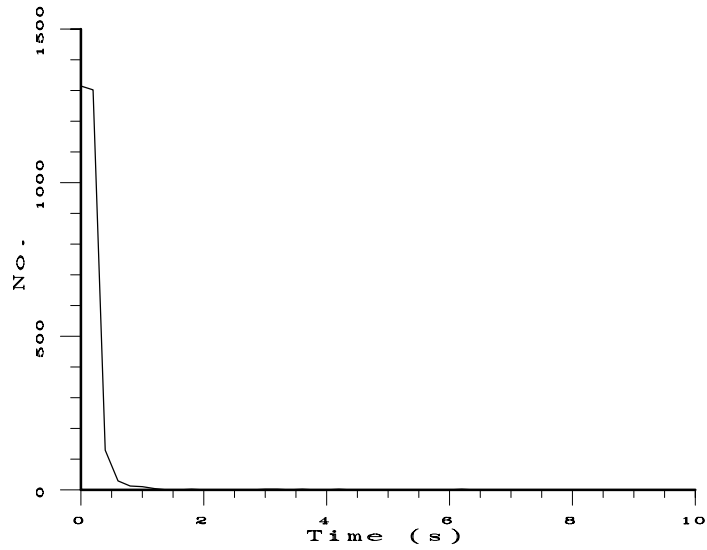


(b)

Figure 4.23: Distribution of times of the reader (a) and syntax checker (b) for n small neural networks



(a)



(b)

Figure 4.24: Distribution of times of the reader (a) and syntax checker (b) for the large neural network

the finder has to do and the longer its execution time. As the number of possible plates located by the finder increases so does the work load of the reader as it has to try and decipher each of the possible plates passed to it by the finder and so the reader's execution time also increases. The reader produces, for each possible plate, a table of the possible characters which it passes to the syntax checker to search for a combination which satisfies the syntax rules. So, the more possible plates, the longer the execution time of the syntax checker.

The distributions in Figures 4.21, 4.22, 4.23 and 4.24 prove that the large difference between the maximum and minimum times are insignificant as they are caused by only a few plates. However, the large maximum value of the syntax checker for the small neural network case is of particular interest. The image which is responsible for the large maximum value contains a vehicle with a large area containing strong vertical edges. The finder successfully locates this area and passes it on to the reader as a possible plate. The possible plate actually contains the "VOLVO" emblem and a large portion of the radiator grille from the front of a lorry. The small neural networks are particularly vulnerable to this type of image data, generating a far larger number of erroneous possible characters than either the template matching or the large neural network. The large number of possible characters cause the syntax checker to take the exceptionally long time of 912.49 seconds.

The average total processing time of 3.18 seconds per image is perhaps a little slow, 2 seconds being more desirable. This could easily be achieved by spreading the system processes over more processors. If the finder (which is responsible for nearly three quarters of the 3.18 seconds) was placed on one processor and the reader and syntax checker placed on another, then images would emerge from the pipeline at an average of every 2.29 seconds.

4.6 Summary

This chapter has described the development of the prototype number-plate recognition system into a fully operational, high performance product. The new feature of the Automatic Trigger together with the new and highly successful algorithms for number-plate location have been described in detail. Three methods for character recognition have been investi-

gated, with the large neural network architecture proving to be the most successful.

Together, these developments amount to a robust system, capable of reading vehicle number-plates in a *realistic* environment.

Chapter 5

A Generic Road-traffic Monitoring Sensor

5.1 Introduction

This chapter states the aims of the generic road-traffic monitoring sensor and how they can be accomplished with a suitable object recognition algorithm. The alternative approaches of performing object recognition via searching parameter space or correspondence space are discussed. The development of a new, efficient algorithm for performing object recognition via searching correspondence space is presented. A pipelined implementation of the system on the transputer based vision hardware is then described. Finally, results from trials of the sensor are presented.

5.2 A Generic Road-traffic Monitoring Sensor

The aim of this sensor is to be able to process a sequence of images from a road-side camera and for each image, recognise and locate all vehicles in the image. The sensor must extract the following information for each vehicle:

- Realworld position of vehicle (x, y, z) .
- Realworld orientation of vehicle (α, β, γ) .
- Principal dimensions of vehicle (length, width, height).

The system is intended to be only the first component in an *intelligent* traffic-monitoring system. It can be regarded as a highly sophisticated sensor, the output of which requires further processing in order to extract the desired information. However, the output of the sensor is so rich that it will satisfy the requirements of nearly any conceivable traffic-monitoring system. With suitable post-processing of the sensor output, the following traffic parameters are easily measured:

- Vehicle counts, speed and headway
- Flow rates (vehicles/minute)
- Vehicle class (car, van, bus, etc.)
- Vehicle density (vehicles per meter)
- Entry/exit statistics for complex junctions
- Congestion detection
- Queue length
- Incident detection

5.2.1 Model Based Object Recognition

To achieve the aims specified in the previous section, the sensor needs to be able to identify and locate vehicles in images. The process of identifying and locating objects in images is known as *object recognition*. In order to perform object recognition, the sensor will make use of knowledge about the appearance of vehicles and the geometry of the camera with respect to the road. This knowledge is represented in models.

As discussed in Chapter 2, model based object recognition may be performed via searching in parameter space or in correspondence space. Searching parameter space requires the global optimum of a score function to be found. Unfortunately, the score function will typically have many local optima and so location of the global optimum may not be guaranteed. This leaves two paths open:

1. Use optimisation methods such as simulated annealing or genetic algorithms. These methods attempt (but do not guarantee) to find the global, or at least near global optimum.
2. Use cues such as position in image and direction of motion to establish approximate values for the model parameters. If the approximation is good enough (close to the global optimum), then optimisation methods such as the simplex may be used to precisely locate the global optimum.

Both of the above approaches have associated problems. With the first approach, the problem is that simulated annealing and genetic algorithms cannot guarantee the location of the global optimum and may well converge on one of the local optima. With the second approach, the problem is with the use of cues such as direction of motion to infer the pose of the vehicle. Such inferences imply assumptions about the motion of vehicles which may well not hold under the very conditions in which the sensor is supposed to operate. Consider an incident in which a vehicle is skidding sideways along the road, how valid are the motion cues now?

The downfall of searching parameter space is the necessity to optimise a complex function with many local optima.

Alternatively, object recognition may be performed via searching correspondence space. This involves finding the optimal correspondence between image and model features. If this correspondence is known and the score function is carefully designed, the score function will only have a single optimum which can be robustly and relatively quickly located using any of a number of optimisation methods.

For the above reasons it was decided to search correspondence space when performing object

recognition within the traffic-monitoring sensor. Unfortunately, the correspondence based approach rapidly becomes intractable when dealing with *real* problems. The remainder of this chapter describes a new algorithm which turns the concepts of searching correspondence space into an efficient, tractable implementation. The implementation requires two types of model, a geometric model and a relational model. These contain knowledge about vehicles, the camera and the road-scene.

5.2.2 The Geometric Model

The object recognition procedure requires knowledge of the objects to be located. Additionally, if the objects are to be located in real world coordinates, then the position of the camera relative to the coordinate system and camera parameters, such as focal length, need to be known. All of this knowledge is represented in the model shown in Figure 5.1.

The camera parameters are fixed and need only be derived once during setup either from specifications, direct measurement or some sort of calibration procedure. The optical axis of the camera lies in the $y - z$ plane of the coordinate system, inclined at an angle α_{cam} below the z axis.

The model parameters are unknown. It is the task of the object recognition algorithm to find values for the model parameters such that when the three dimensional model is projected into the image, the model coincides with the evidence in the image. The model only has three pose parameters rather than the expected six. This simplification is based on the assumption that a vehicle will always be flat on the road which is approximated by the plane $y = 0$. This assumption is necessary to resolve the scale ambiguity in which a large, distant object appears identical to a small, near object. Assuming the vehicle is on a known plane enables recovery of absolute values for distance and size rather than just the ratio between them.

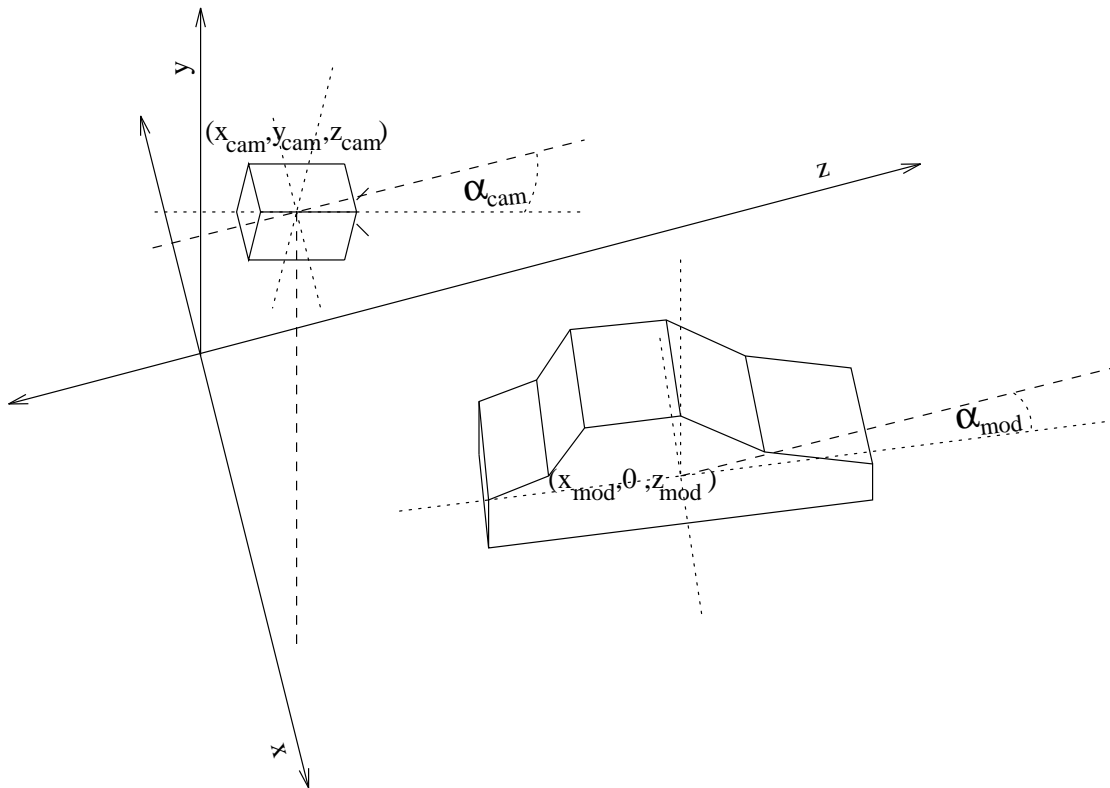


Figure 5.1: Model containing knowledge of vehicle appearance and camera position within coordinate system

5.2.3 Choice of Image Feature

Object recognition will be performed by searching correspondence space for the optimal mapping (correspondence) between image and model features. Therefore, the choice of which type of feature to use must first be made. Work at Reading University (Section 2.4.3) used complex features such as 'U' and 'S' shapes. However, for this system it was decided to use simple features such as edges or regions because the simpler the feature the more robustly and quickly it can be extracted from the image.

The choice between edges (or straight line approximations of edges) and regions, then had to be made. Now, when searching correspondence space, an exhaustive search would examine $\frac{i!}{(i-m)!}$ combinations, assuming a one to one mapping between model and image features and where i is the number of image features and m the number of model features. Figure 5.2 shows a simple model of a cuboid, together with the number of features extracted for both straight lines and regions. The table in Figure 5.2 shows the number of combinations to be explored in both cases.

It is clear that using regions significantly reduces the number of combinations to be explored and will provide a corresponding improvement in speed.

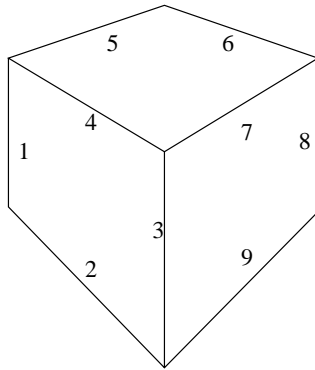
5.2.4 Extraction of Image Features

The segmentation of an image into meaningful regions is, in itself a vast area of research. The aim is to divide the image into a number of regions, each of which represents a different object or part of an object in the image. This may be achieved in one of two ways [69]:

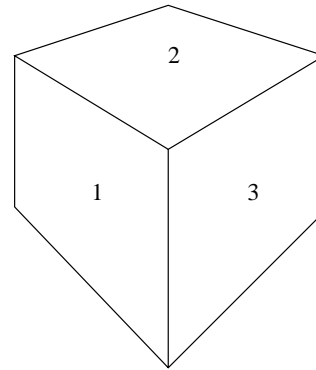
- **Local techniques:** Pixels are grouped into a region if they have similar characteristics to their neighbours.
- **Global techniques:** Pixels are grouped into a region if they have similar characteristics to large numbers of pixels which may be distributed throughout the image.

Type of feature:

straight line approximation
of edges



regions



Number of
features: 9

3

	Straight lines	Regions
Model	9	3
Image	9	3
No. combinations	3.63×10^5	6

Figure 5.2: Simple model of a cuboid together with number of features extracted for both straight lines and regions. The table shows the number of combinations to be explored in each case

Similar characteristic usually means similar grey-level although alternatives include colour and texture.

Local techniques are often based on a *region growing* algorithm. Each region starts life as a single seed pixel. Each of its neighbours is examined and if the neighbour has similar characteristics to that of the growing region, it is merged with the region. This process is iterated until no more regions will merge. Some techniques have attempted to integrate edge information such that pixels cannot be in the same region if they have a strong edge between them.

Global techniques tend to be based on the image histogram. Consider a simple image with a light object on a dark background. The image histogram would contain two peaks, one due to the object and one due to the background. The image could easily be segmented into two regions, one for object and one for background, by examining the histogram and placing a threshold at the minimum between the two peaks. Pixels below the threshold belong to the background region while pixels above belong to the object (Figure 5.3).

Unfortunately, the iterative nature of region growing algorithms renders them useless for this application as road-traffic monitoring systems require real-time operation. The global, histogram based methods tend to be much faster, a well known algorithm being *K means*. This is also an iterative algorithm but when only using grey-level information it requires few iterations and can be performed very quickly. The algorithm is as follows:

1. Calculate a grey-level histogram for the image.
2. Arbitrarily position the K means within the range of the grey-levels.
3. Place a boundary at the mid-point between each of the means.
4. For each mean, take all the pixels between the boundary to its left and the boundary to its right and calculate their average value. Move the mean to this average value.
5. Iterate steps 1-4 until the means converge to a stable state.
6. Map each pixel in the image to the number of the mean which it is closest to. Figure 5.4 shows the histogram of an image, the final positions of the k means and the resulting

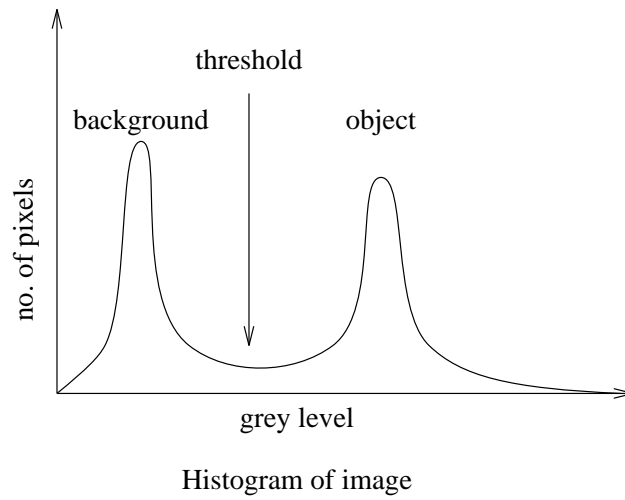
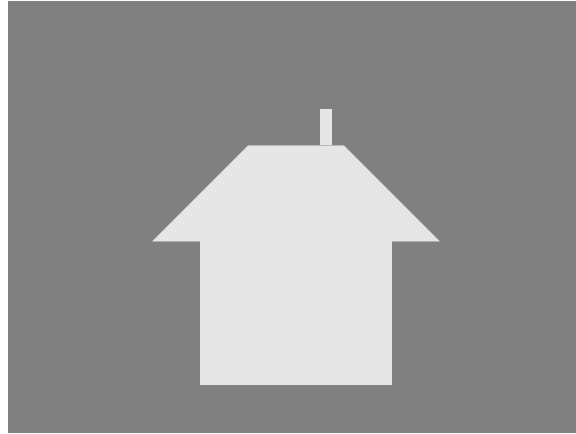


Figure 5.3: Region segmentation based on image histogram

mapping.

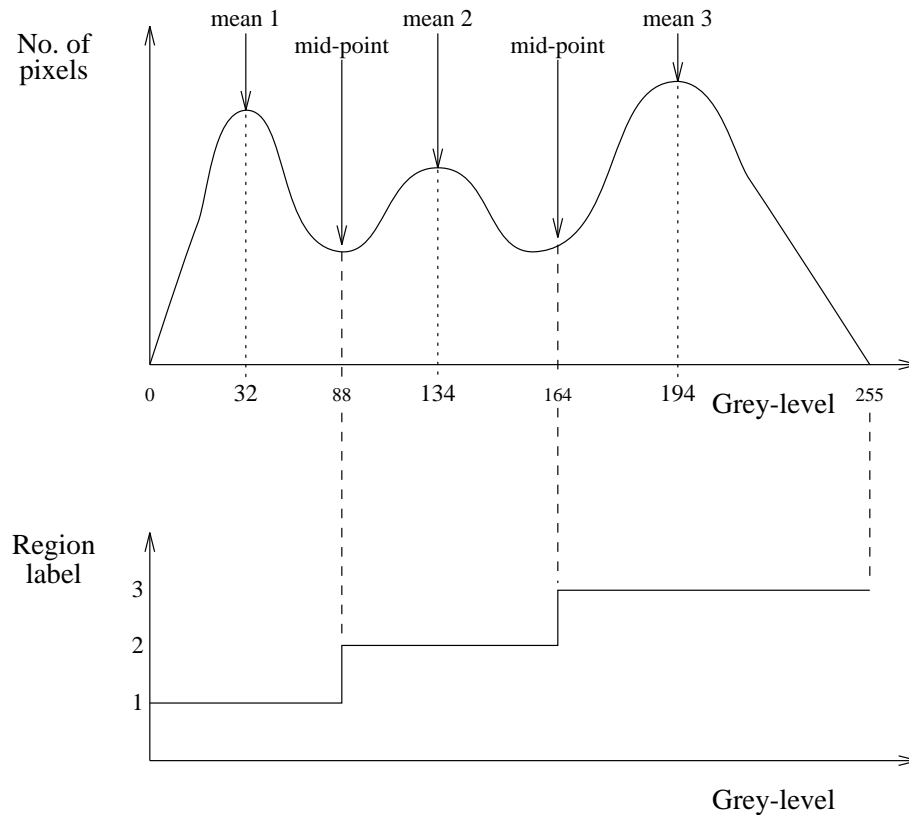


Figure 5.4: Image histogram, positions of k means and corresponding mapping

The number of means, K , will effect the result of the segmentation process. If K is too low, an under-segmentation will occur in which objects or parts of objects which should have been represented by different regions are merged into a single region. On the other hand, if K is too high, an over-segmentation will occur where objects or parts of objects will be represented by more regions than necessary.

After performing an initial segmentation it is common to perform some post-processing during which an attempt is made to correct for under or over-segmentation by splitting or merging regions respectively. The process of splitting an under-segmented region is extremely difficult as the decision of *where* to split the region must be made. This decision is by no means trivial. Merging of over-segmented regions has no such counterpart and the merge is very simple to perform. For this reason it is desirable for K to be biased

toward an over-segmentation and a post-processing operation to perform region merging where necessary. However, the larger the value of K , the more regions produced by the segmentation. More regions means more processing for subsequent stages. Therefore, K must be chosen as low as possible while still providing an over-segmentation.

5.2.5 Merging of Image Features

For an over-segmented image, the optimal mapping between image and model features must be a many to one mapping. Section 5.2.3 showed that the number of combinations to be explored during an exhaustive search for the optimal mapping is $\frac{i!}{(i-m)!}$, where i and m are the number of image and model features respectively. This result only applies to a one to one mapping. In the case of a many to one mapping, the number of combinations is approximately:

$$\text{number of combinations} \approx \frac{(\sum_{k=1}^{max} \frac{i!}{(i-k)!k!})!}{(\sum_{k=1}^{max} \frac{i!}{(i-k)!k!} - m)!} \quad (5.1)$$

Where, max is the maximum number of image features that may be mapped to a single model feature, i is the number of image features and m is the number of model features.

Table 5.1 shows that even for a simple case with $i = 8$ and $m = 6$, the number of combinations grows explosively as max increases.

max	Number of combinations
1	2.0×10^4
2	1.4×10^9
3	5.1×10^{11}
4	1.7×10^{13}

Table 5.1: Number of combinations against max

It is clear that keeping max as low as possible will significantly reduce the number of combinations to be explored when searching for the optimal mapping. The aim is, therefore, to merge the regions produced by the K means such that the segmentation is ideal and only

a one to one mapping need be considered.

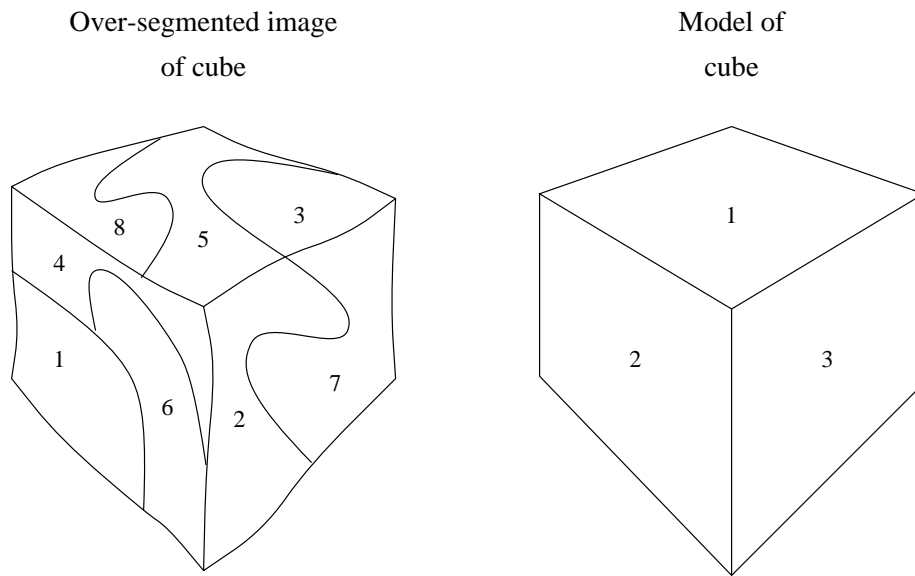
Many of the regions produced by the segmentation are only a few pixels in size. These regions tend to be caused by noise or texture in the image. The first step in the merging process is therefore to merge these small regions into larger ones.

The second stage of the merging process recursively generates all possible combinations of merged regions containing between 1 and max_{merged} connected regions. It is these merged features which will be mapped to model features in the search for the optimal correspondence. However, as the majority of the merged features could not possibly represent any of the model features due to their completely different shape, it would be foolhardy to include them in the search for the optimal mapping. A filtering operation is therefore performed which rejects all merged features which do not look like any of the model features. The approach is as follows:

1. Each model feature has associated with it a list of merged image features which look like it. These lists are initialised as empty.
2. Recursively generate all possible combinations of merged regions, where each merged feature contains between 1 and max_{merged} , connected regions.
3. Check each merged feature against each model feature. If it looks like a model feature, add it to the list for that model feature.

Figure 5.5 shows an over-segmented cube, a list of merged image features which look like model features and, for each model feature, a list of the merged image features which look like it. Only the merged features which look like model features will be considered when searching for the optimal mapping which will now be one to one.

The difficulty with the above procedure is in deducing whether a merged image feature looks like a model feature. The appearance of each model feature changes as the model parameters vary. Deciding whether the merged image and model features look alike is another object recognition problem. Given a merged image feature and a model feature, the model parameters must be found such that when the model feature is projected into



Merged image feature	Regions
a	1,4,6
b	3,5,8
c	2,7

Model feature	Possible merged image features
1	b
2	a, c
3	a, c

Figure 5.5: An over-segmented cube, a list of merged image features and, for each model feature, a list of the merged image features which look like it

the image it lines up as best as possible with the merged image feature. The task is tackled in four stages:

The first stage finds approximate values for x_{mod}, z_{mod} . This is achieved by calculating the centroid of the merged image feature. The image coordinates of the centroid are then inversely projected to a line in the real world which is intersected with the road ($y = 0$), yielding approximate values for x_{mod}, z_{mod} . The inverse projection is described in Appendix C.

The second stage uses a shape descriptor based on a histogram of angles. This is constructed by examining each point on the external boundary of the merged feature and calculating the angle of the tangent at that point. After doing this for all boundary points a histogram of the angle of the tangents may be produced. Figure 5.6 shows a rectangular feature and its histogram of angles. Using average values for $l_{mod}, w_{mod}, h_{mod}$, the x_{mod}, z_{mod} values calculated earlier, and a range of values for α_{mod} , the model feature is projected into the image. The projection of the model into the image is described in Appendix B. The angle of the projected lines of the model feature are used to construct a second histogram. The two histograms are correlated to produce a measure of their similarity. The model feature is aligned with the merged image feature by selecting the value of α_{mod} which leads to the most similar histograms. If the similarity measure is not high enough, the merged image feature is deemed not to look like the model feature, otherwise the next stage is applied.

The third stage involves a shape descriptor based on the aspect ratio of the features. The position of the largest peak in the histogram of the merged image feature defines the major axis of the feature. All of the external boundary points of the merged image feature are projected onto the major axis. The distance between the two projected points which are furthest apart defines the length of the feature along the major axis. The width of the feature is calculated by similarly projecting the boundary points onto the minor axis which is perpendicular to the major axis. The aspect ratio is simply the length divided by the width.

The model feature is then projected into the image using average values for $l_{mod}, w_{mod}, h_{mod}$, and the values for $x_{mod}, z_{mod}, \alpha_{mod}$ derived earlier. Using the same major and minor axis,

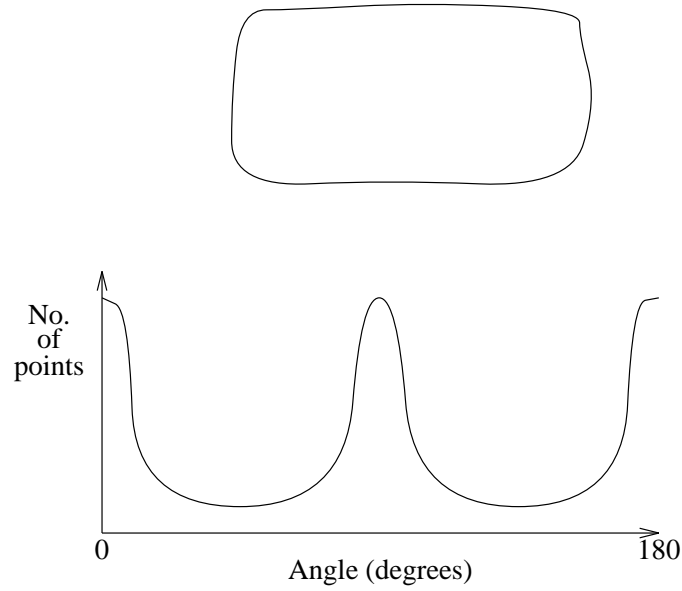


Figure 5.6: A rectangular feature and its shape descriptor based on a histogram of angles

the length and width of the projected model feature is calculated. If the aspect ratio of the projected model feature and the merged image feature differ by more than a specified amount, the merged image feature is rejected, otherwise the final stage is applied.

The final stage attempts to match the model feature to the merged image feature by minimising an error function which reflects how well the projected model feature lines up with the merged image feature. The error is dependent on the model parameters and is defined as:

$$e = e_1 + e_2 + e_3 \quad (5.2)$$

and,

$$\begin{aligned} e_1 &= \left(\frac{l_{mod} - l_{av}}{l_{sd}} \right)^2 + \left(\frac{w_{mod} - w_{av}}{w_{sd}} \right)^2 + \left(\frac{h_{mod} - h_{av}}{h_{sd}} \right)^2 \\ e_2 &= (i^{ic} - i^{mc})^2 + (j^{ic} - j^{mc})^2 \\ e_3 &= (w^i - w^m)^2 + (l^i - l^m)^2 \end{aligned}$$

Where, (i^{ic}, j^{ic}) is the image coordinate of the centroid of the merged image feature, (i^{mc}, j^{mc}) is the image coordinate of the centroid of the projected model feature, l^i, w^i

are the length and width along the major and minor axis of the merged image feature, l^m, w^m are the length and width along the major and minor axis of the projected model feature, l_{av}, w_{av}, h_{av} are the average values of the model dimensions, and l_{sd}, w_{sd}, h_{sd} are the standard deviations of the model dimensions.

The purpose of e_1 is to ensure that if the model had to be contorted to a highly non car-like shape in order to line up the merged image and projected model feature, a large error is returned. e_2 serves to line up the centre of the merged image feature with the centre of the projected model feature, while e_3 causes the model to change shape until the lengths and widths of the projected model and merged image features are the same.

Average values for $l_{mod}, w_{mod}, h_{mod}$, and the values for $x_{mod}, z_{mod}, \alpha_{mod}$ derived earlier are used as a starting point within the six dimensional parameter space. There is a good chance that the closest optimum to the starting point will be the global optimum. *Powell's direction set* method (Appendix D) is used to minimise the error from the starting point and hopefully locate the global minimum.

After optimisation, if the error is below a threshold the merged image feature is accepted, otherwise it is rejected. If accepted, the merged image feature is added to the model feature's list.

When all combinations of merged image regions have been matched to all model features, each model will have a list of the merged image features which look like it (as in Figure 5.5). The one to one optimal mapping between these merged image features and the model features must now be found.

5.2.6 The Interpretation Tree

The *interpretation tree* is a diagrammatic representation of a search for the optimal mapping. Each node of the tree represents a different possible mapping. In the root node, each model feature is mapped to nothing. Each of the root's siblings is created by mapping the first model feature to each of the merged image features in the model feature's list. In the next generation of siblings, the second model feature is mapped to each of the merged image

features in its list, and so on. At the leaf nodes of the tree, each model feature will be mapped to an image feature. Figure 5.7 shows the interpretation tree for the simple cube example in Figure 5.5. In this case the size of the tree will be $\geq \frac{i!}{(i-m)!}$ because each model feature may also be mapped to nothing. For even modest values of i and m the size of the tree becomes so large that an exhaustive search will not be possible in real time. If $i = 20$ and $m = 10$, a computer capable of evaluating 10^6 nodes per second would take nearly 8 days to explore the complete tree! For this reason it is necessary to reduce the size, or *prune* the tree. Knowledge of the appearance of vehicles will be used to prune branches of the tree which are leading to an unfeasible mapping. It is important to note that it is the effectiveness of the pruning strategy which governs the success or failure of the correspondence based approach.

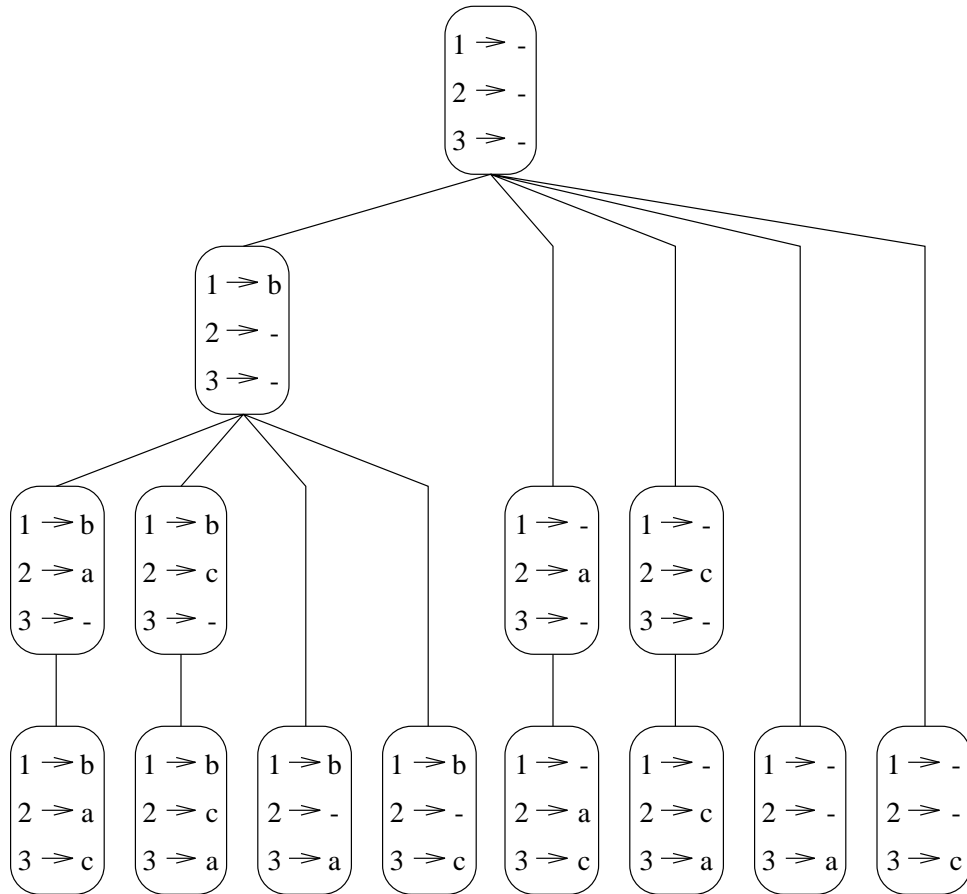


Figure 5.7: Interpretation tree for recognising a cube

5.2.7 Tree Pruning with the View Independent Relational Model

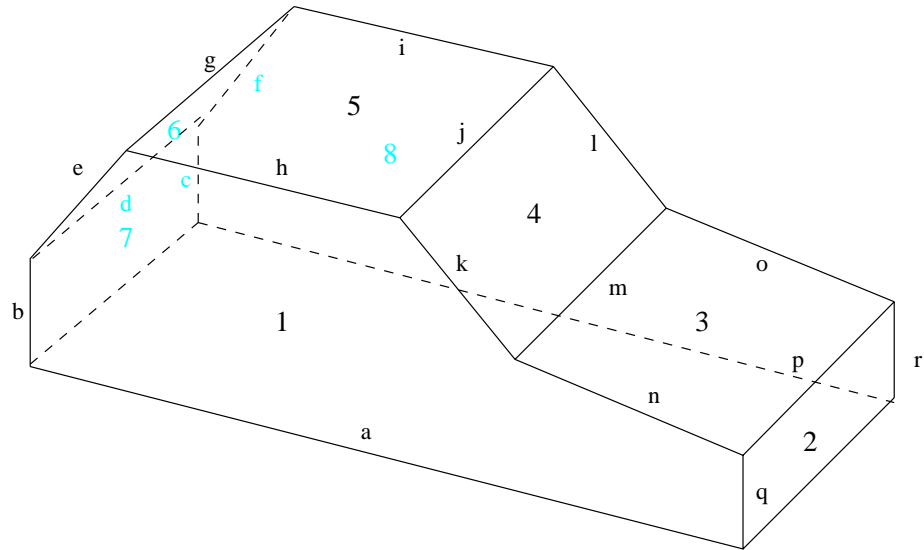
A *view independent relational model* is a way of describing the appearance of an object in the image via a set of relationships which are view independent. The relationships used in this case are:

- **Connectivity:** The front windscreen will always be touching the bonnet whichever angle the vehicle is viewed from, assuming the windscreen and the bonnet are both visible.
- **Co-visibility:** If you can see one side of the car, then you can't see the other.
- **Straight lines:** A straight line will always appear straight, whatever the view angle.
- **Parallel lines:** If the distance between the camera and the object is significantly greater than the size of the object, perspective effects will be negligible and it is then true to say that lines which are parallel will always appear parallel from any viewpoint.
- **Vertex:** Two lines which meet at a vertex will do so whatever the viewpoint.

Figure 5.8 shows the view independent relational model of a vehicle. The regions are labelled numerically, while the lines between regions are labelled alphabetically. The fact that the front edge of the roof is parallel to the rear edge of the roof is easily represented by the relationship *parallel(g, j)*.

The knowledge embedded in the View Independent Relational Model (VIRM) can be used to prune the interpretation tree. When a node of the tree is generated, a check is made to ensure that the mapped image features conform to the relationships in the VIRM. If the mapping is not consistent with the VIRM, the node is pruned and none of its siblings are explored. For each node of the tree, the following checks are made on the mapping.

Firstly, the connectivity relations are checked. During the region merging process, a look-up table is generated containing details of which regions connect to which. It is therefore a simple matter of looking in the table to check if mapped image features are connected. If the mapped image features do not satisfy the connectivity relations, the node is pruned.



Connected	Non co-visible	Straight	Parallel	Vertex
1, 2	1, 8	a	d, g	b, d, e
1, 3	2, 7	b	g, j	c, d, f
1, 4		c	j, m	e, g, h
1, 5		d	m, p	f, g, i
2, 3		e		h, j, k
2, 8		f		i, j, l
3, 4		g		k, m, n
3, 8		h		l, m, o
4, 5		i		n, p, q
4, 8		j		o, p, r
5, 6		k		
5, 8		l		
6, 7		m		
6, 8		n		
		o		
		p		
		q		
		r		

Figure 5.8: View independent relational model of a vehicle

The co-visibility relations are next to be checked. If a model feature has been mapped to an image feature it implies that that model feature is visible. So, if two model features are defined in the VIRM to be non co-visible but both model features are mapped to image features, then the mapping is not consistent with the VIRM and the node must be pruned.

In order to check the final three types of relation, the points along the common boundary of each pair of connected image features in the mapping are fitted to a straight line using a *least squares error* method. If the minimised error is sufficiently small, the boundary is labelled as a straight line. The angle of the line is calculated and its end points are found by projecting all boundary points onto the line and taking the two projected points which are furthest apart. If a line is defined by the VIRM to be straight but the above process has not labelled it as straight then the mapping is inconsistent and the node will be pruned.

The parallel relations are simply checked by calculating the difference in angle between the two lines. If the difference is above a threshold then the relation has not been satisfied and the node will be pruned.

The vertex relations are checked by taking each pair of lines defined in the VIRM to meet at a vertex and calculating their intersection point. If, for both of the lines, the distance of one of its end points from the intersection point is sufficiently small, the relation is satisfied.

5.2.8 Tree Generation with the View Independent Relational Model

A better approach is to use the VIRM not only to prune the tree but to control its generation in the first place. Consider Figure 5.9 which shows the features extracted from an image of a cube next to a pyramid. In an early stage of generating the interpretation tree, feature 4 has been mapped to the top of the cube. Following the tree pruning method from the previous section, the next step is to map the front of the cube (for arguments sake) to each of the remaining image features, thus creating the siblings of the current node in the tree. Each of the siblings would then be checked for consistency with the VIRM and pruned if necessary. This requires five siblings to be generated and then checked with the VIRM. However, as the front of the cube must be connected to the top of the cube, a sibling will only be valid

if the image feature mapped to the front of the cube is connected to the feature mapped to the top. Looking at Figure 5.9, it is clear that the only features connected to feature 4 are 1, 5 and 6. This information is easily extracted from the connectivity table generated during the region merging process. Therefore, only three siblings need be generated and checked rather than five.

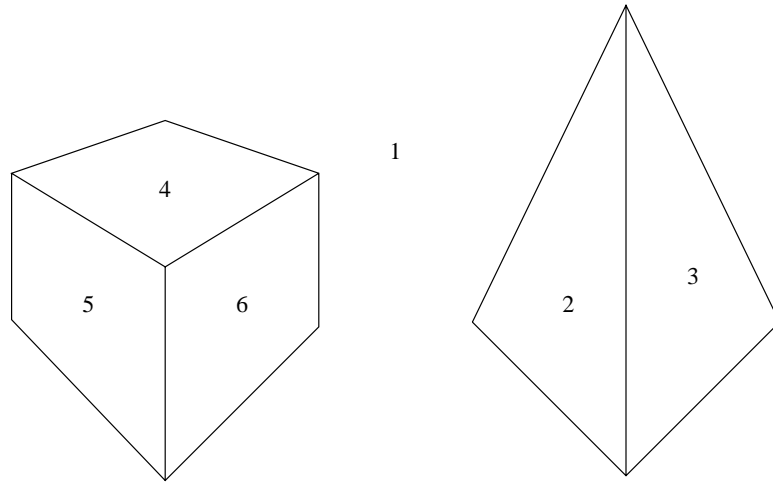


Figure 5.9: Tree generation with the VIRM

Before a node's siblings may be generated the choice of which model feature to add to the mapping must first be made, i.e. in the above example it was decided to add the front of the cube to the mapping. For the simple cube model it does not make any difference what order the model features are added to the mapping. However, for more complex models such as the vehicle model, the order effects the size of the tree. The previous paragraph described how the connectivity relationship in the VIRM can be used to control the generation and therefore reduce the size of the tree. This only works if the model feature being added to the mapping is connected to one of the model features already in the mapping. The model features must therefore be added to the mapping in a suitable order. To achieve this a new algorithm has been developed for tree generation:

1. The tree's top level nodes are created by mapping each model feature to each merged image feature in its list.

2. Generate each node's siblings. Firstly, identify all model features which are connected to at least one of the model features already in the mapping. Each of these model features is called a *child*. A child's *parent* is the model feature, already in the mapping, to which it is connected. The node's siblings are generated by mapping each child to each merged image feature in the child's list that is also connected to the image feature to which the parent is mapped. As it is possible for several parents to have the same child, a flag is set once a child has been used to generate siblings so that it is not used again.
3. Recursively apply step 2 in order to generate the whole tree.

5.2.9 Tree Pruning and Pose Recovery with the Geometric Model

Given a mapping between image features and features in the geometric model it is possible to line up the model with the image features and thus recover the model parameters. This is pose recovery. The process is based on minimising an error function where the error function reflects how well the model lines up with the image features. If the minimised error is not sufficiently small it infers that the mapped image features do not look like the model. In this case the mapping may be rejected and the branch of the tree below this mapping may be pruned. In this system a new strategy has been developed for performing pose recovery. It operates in several stages, each of which progressively refines the model pose:

1. $l_{mod}, w_{mod}, h_{mod}$ are initialised to average values.
2. The x_{mod} and z_{mod} parameters of the model are estimated by lining up the center of the model with the center of all the mapped image features. The center of the mapped image features is derived by averaging the image coordinates of the external boundary of the mapped features (Figure 5.10). The image coordinate of the center of the mapped features is then inversely projected to a line in the real world. The intersection of this line with the road ($y = 0$) yields approximate x_{mod} and z_{mod} values for the model's pose. Appendix C describes the inverse projection from image to real world coordinates.

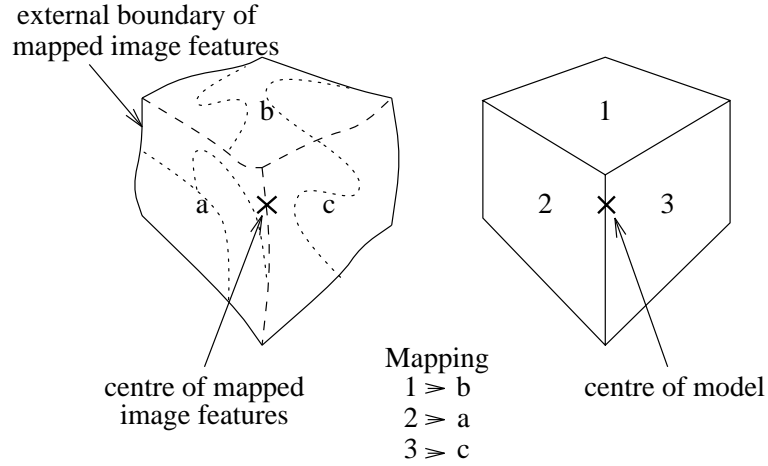


Figure 5.10: Centre of mapped image features is found by calculating the average coordinates of all points on the external boundary

The remaining stages make use of four different error functions, each of which progressively refines the match between the mapped image features and the projected model features.

3. The error function in this stage is the average distance between the centre of each mapped image feature and the centre of the projected model feature to which it is mapped (Figure 5.11).

$$e = \frac{1}{N} \sum_{n=1}^N ((i_n^{ic} - i_n^{mc})^2 + (j_n^{ic} - j_n^{mc})^2) \quad (5.3)$$

Where, $(i_n^{ic}, j_n^{ic}), n = 1, \dots, N$ are the coordinates of the centroids of the N mapped image features, and $(i_n^{mc}, j_n^{mc}), n = 1, \dots, N$ are the coordinates of the centroids of the N projected model features.

Minimising this function will line up the centre of each image feature with the centre of its corresponding projected model feature. Using average values for $l_{mod}, w_{mod}, h_{mod}$ and the $x_{mod}, z_{mod}, \alpha_{mod}$ values derived previously as a starting point, the minimum of this function is located with Powell's optimisation method. If the minimised error is not sufficiently small the current mapping is rejected otherwise the next stage is applied.

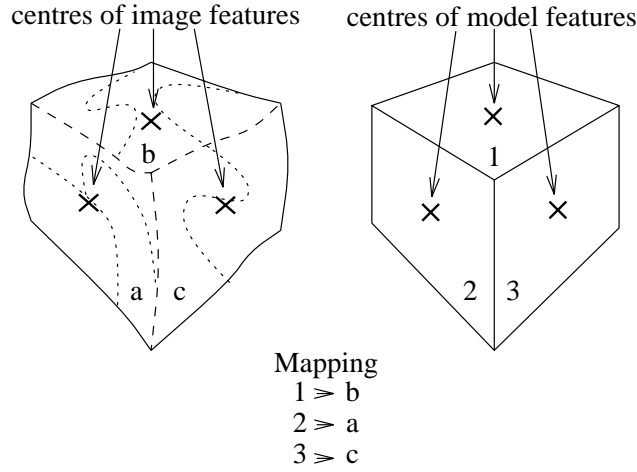


Figure 5.11: Centres of mapped image features and corresponding projected model features

4. The error function used in this stage is the same as used in the previous stage but with an extra term which compares the size of each model feature with that of the mapped image feature:

$$\begin{aligned}
e = & \frac{1}{N} \sum_{n=1}^N ((i_n^{ic} - i_n^{mc})^2 + (j_n^{ic} - j_n^{mc})^2) + \\
& \frac{1}{N} \sum_{n=1}^N ((w_n^i - w_n^m)^2 + (l_n^i - l_n^m)^2)
\end{aligned} \tag{5.4}$$

Where, $w_n^i, l_n^i, n = 1, \dots, N$ are the widths and lengths of the N mapped image features, and $w_n^m, l_n^m, n = 1, \dots, N$ are the widths and lengths of the N projected model features.

The length and width of each feature is measured along the major and minor axis respectively (Figure 5.12). Minimising this error function therefore causes the model to change shape until each model feature is approximately the same size as its mapped image feature. Again, Powell's optimisation method is used to find the minimum, using as a starting point the position of the minimum located in the previous stage. If the minimised error is not sufficiently small the mapping is rejected, otherwise the next stage is applied.

The error functions in the previous stages were designed to have only a single optimum such that they could be robustly and quickly optimised at the expense of the recovered

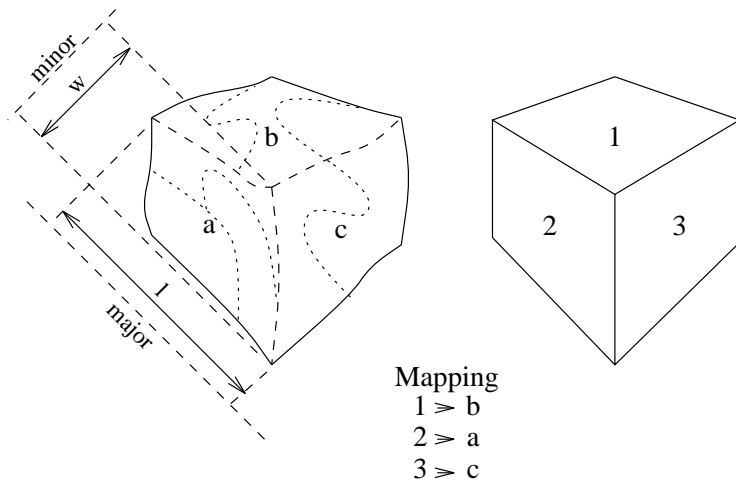


Figure 5.12: Length and width of features are measured along major and minor axis respectively. Dimensions of image feature a are shown here.

parameters being only approximate. The error functions in the final two stages will line up the projected model features very much more precisely with the image features. It is also likely that these error functions will have many local optima. However, the approximate values for the model parameters which have been derived during the previous stages will be within the catchment area of the global optimum and so Powell's method, starting at the values derived previously, will find the global optimum.

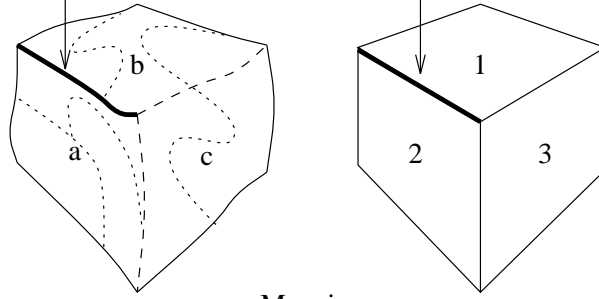
5. The internal boundary between two image features corresponds to the line between the two model features to which the image features are mapped (Figure 5.13). This error function reflects how close each point of the internal boundaries is to the corresponding projected model line.

Given a projected model line defined by the image coordinates its two end points $(i^{m0}, j^{m0}), (i^{m1}, j^{m1})$, the error, p , for a particular point (i, j) is:

$$p(\dots) = \begin{cases} (i - i^{m0})^2 + (j - j^{m0})^2 & \text{if } \beta < 0 \\ (i - i^{m1})^2 + (j - j^{m1})^2 & \text{if } \beta > 1 \\ (i - i^{m0} - \beta(i^{m1} + i^{m0}))^2 + \\ (j - j^{m0} - \beta(j^{m1} + j^{m0}))^2 & \text{otherwise} \end{cases} \quad (5.5)$$

where,

points on this internal boundary correspond to this internal model line



Mapping

1 \ni b

2 \ni a

3 \ni c

Figure 5.13: Internal boundary between two image features and corresponding projected model line

$$\beta = \frac{(i^{m1} - i^{m0})(i - i^{m0}) + (j^{m1} - j^{m0})(j - j^{m0})}{(i^{m1} - i^{m0})^2 + (j^{m1} - j^{m0})^2}$$

For a particular model line, p is a function of i, j and defines a surface as in Figure 5.14.

The closer the point (i, j) is to the line, the smaller the error.

The error function for this stage is defined as:

$$\begin{aligned} e = & \frac{1}{N} \sum_{n=1}^N ((i_n^{ic} - i_n^{mc})^2 + (j_n^{ic} - j_n^{mc})^2) + \\ & \frac{1}{N} \sum_{n=1}^N ((w_n^i - w_n^m)^2 + (h_n^i - h_n^m)^2) + \\ & \frac{1}{P^i} \sum_{n=1}^{P^i} p(i_n^{ip}, j_n^{ip}, i_i^{m0}, j_i^{m0}, i_i^{m1}, j_i^{m1}) \end{aligned} \quad (5.6)$$

Where, $(i_n^{ip}, j_n^{ip}), n = 1, \dots, P^i$ are the coordinates of the points on the internal boundaries of the mapped image features, and $(i_i^{m0}, j_i^{m0}), (i_i^{m1}, j_i^{m1})$ are the endpoints of the model line which corresponds to the internal boundary on which point (i_n^{ip}, j_n^{ip}) lies.

Again Powell's method is used for optimisation, starting from the position of the minimum located in the previous stage. If the optimised error is not small enough the mapping is rejected, otherwise the final stage is applied.

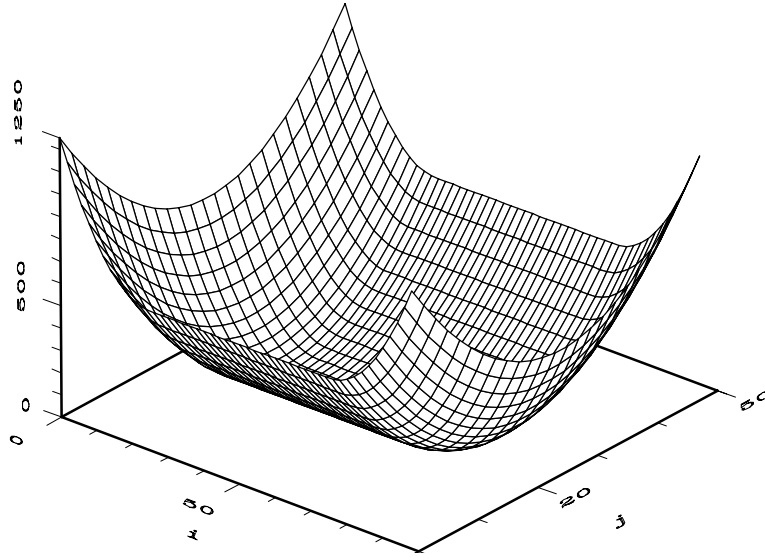


Figure 5.14: Error surface for matching a point to the line: (25,25),(75,25)

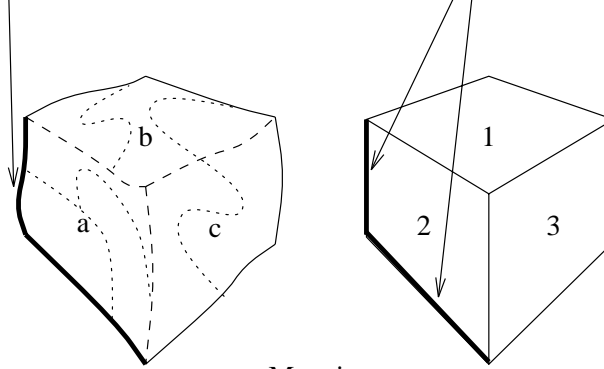
6. Points on the external boundary of an image feature could correspond to any of the external lines of the model feature to which the image feature is mapped (Figure 5.15). This error function reflects how close each point on the external boundaries is to the closest of the relevant model lines.

The error function for this stage is defined as:

$$\begin{aligned}
 e = & \frac{1}{N} \sum_{n=1}^N ((i_n^{ic} - i_n^{mc})^2 + (j_n^{ic} - j_n^{mc})^2) + \\
 & \frac{1}{N} \sum_{n=1}^N ((w_n^i - w_n^m)^2 + (h_n^i - h_n^m)^2) + \\
 & \frac{1}{P^i} \sum_{n=1}^{P^i} p(i_n^{ip}, j_n^{ip}, i_i^{m0}, j_i^{m0}, i_i^{m1}, j_i^{m1}) + \\
 & \frac{1}{P^e} \sum_{n=1}^{P^e} p(i_n^{ep}, j_n^{ep}, i_e^{m0}, j_e^{m0}, i_e^{m1}, j_e^{m1}) \quad (5.7)
 \end{aligned}$$

Where, $(i_n^{ep}, j_n^{ep}), n = 1, \dots, P^e$ are the coordinates of the points on the external boundaries of the mapped image features, and $(i_e^{m0}, j_e^{m0}), (i_e^{m1}, j_e^{m1})$ are the endpoints of the external model line which is closest to the point (i_n^{ep}, j_n^{ep}) .

points on this external boundary correspond to either of these external model lines



Mapping
1 \supseteq b
2 \supseteq a
3 \supseteq c

Figure 5.15: External boundary of an image feature may correspond to any of the external lines of the model feature to which it is mapped

If, after optimising with Powell's method using the position of the minimum in the previous stage as a starting point, the error is too large, the mapping is rejected. Otherwise, the mapping is accepted and the parameter values which gave the minimum error represent the model pose such that the projected model lines up as best as possible with the mapped image features.

5.2.10 Optimal Mapping Selection

After traversing the interpretation tree the best, or optimal, mapping must be selected. As the pruned interpretation tree is explored, each mapping which satisfies the relations in the VIRM and adequately matches the geometric model is assigned two scores, a VIRM score and a GEOM score. The VIRM score is simply the number of mapped image features in the mapping. The GEOM score is the final minimised error after fitting the geometric model to the mapped image features. Prior to searching the tree, a best mapping is initialised with a VIRM score of minus one. Then, for each node of the tree, if the VIRM score of the current mapping is greater than the VIRM score of the best mapping, the current mapping becomes the best mapping. If the VIRM score of the current mapping is equal to the VIRM

score of the best mapping and the GEOM score of the current mapping is less than the GEOM score of the best mapping, then the current mapping becomes the best mapping. Also stored with the best mapping are the model parameters recovered by matching the geometric model to the image features in the mapping.

So, after exploring the tree, the best mapping represents the optimal correspondence between image and model features and the associated vehicle pose has been recovered. This pose is the pose of the recognised vehicle in the image being processed and forms the output of the generic road-traffic monitoring sensor.

In some cases, perhaps due to an under-segmentation of the original image, it is not possible to find a good correspondence between image and model features. This will lead to the best mapping having a low VIRM score. In these cases, the recovered pose should not be relied on as it is based on very little information, perhaps only a single image feature. Therefore, if the VIRM score of the best mapping is below a threshold, the sensor does not output the recovered vehicle pose.

5.3 Implementation on Vision Hardware

The generic road-traffic monitoring sensor consists of a digitiser module, a general function module and three framestore modules. These modules are arranged in an image processing pipeline. The software is implemented as three communicating sequential processes. Figure 5.16 shows the hardware configuration and the mapping of the software processes onto the available processors. The operation of each module together with its associated processes will now be discussed in more detail.

5.3.1 Digitiser Module

The digitiser performs two tasks:

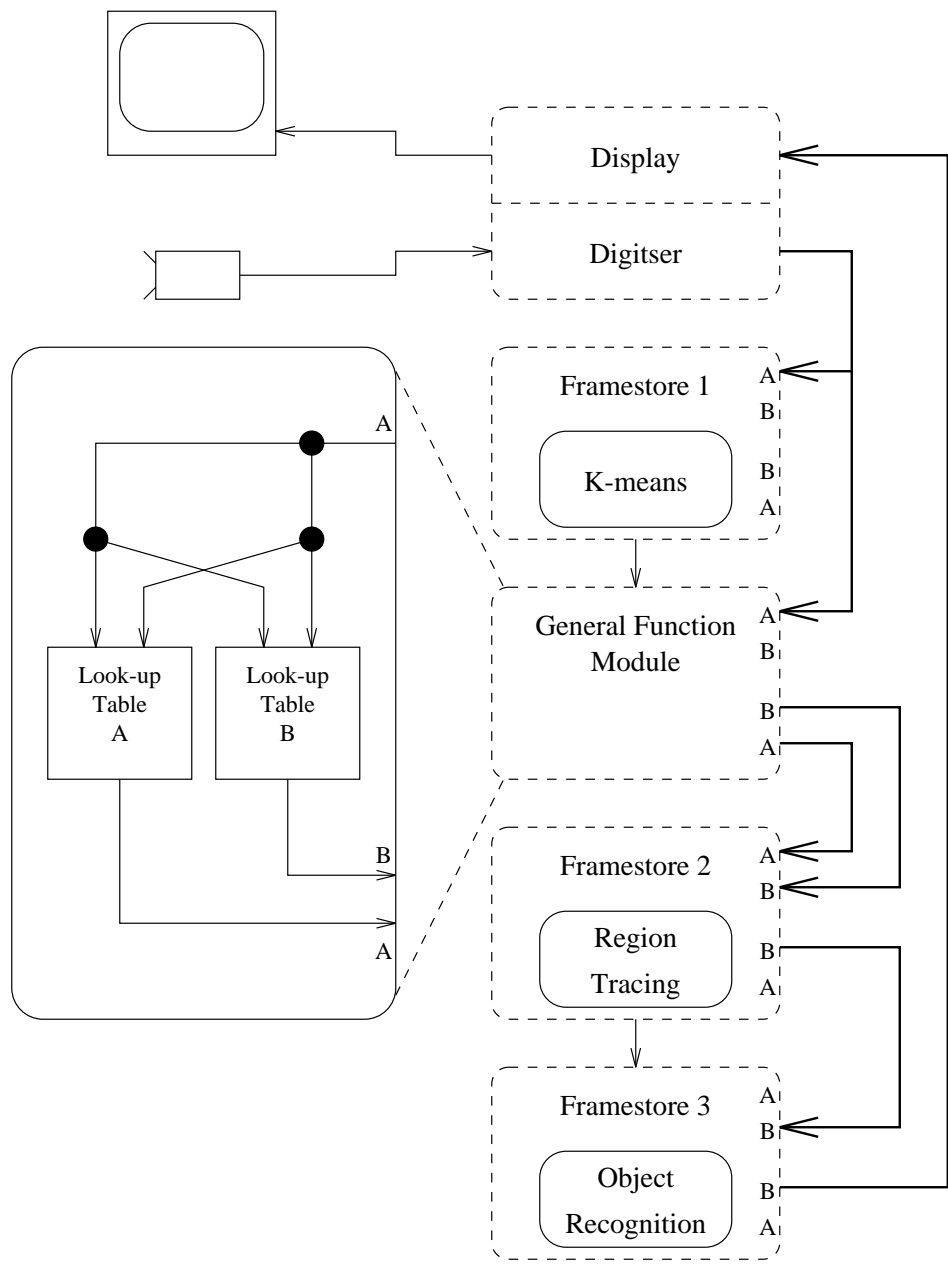


Figure 5.16: Configuration of hardware and mapping of software processes onto hardware for generic road-traffic monitoring sensor

1. Digitise the incoming signals from the video camera and provide the head of the pipeline with a continuous stream of digitised images.
2. Receive digital images from the tail of the pipeline and output them in analogue form suitable for display on a monitor. The display consists of the matched model of a vehicle superimposed on the original image from the camera.

5.3.2 Framestore Module 1

The Transputer in this module has the *K-means* process running on it. It takes images from the digitiser and calculates a grey-level histogram. After the K-means algorithm has been applied to this histogram, the positions of the k means are used to produce a mapping between *pixel value* and *region label*, where *region label* is between 1 and K , as described in Section 5.2.4. The mapping is then transmitted down the transputer link to the general function module.

5.3.3 General Function Module

This module is configured such that images from the digitiser arriving at input port A are passed directly to both look-up tables. Look-up table A is programmed according to the mapping received from framestore 1, and outputs via output port A. Images emanating from output port A are thus the result of performing the entire K-means segmentation on the input images.

Look-up table B is programmed to output the incoming pixel values unchanged, via output port B. These images are therefore identical to the input images. The purpose of passing them through the look-up table is merely to keep them in sync with the K-means images.

5.3.4 Framestore Module 2

Input port A receives the K-means images from the general function module. The data content of these images has been drastically reduced but needs to be converted to a more

concise format for future processing. This conversion is performed by the *region tracing* process running on this module. The *region tracing* process traces around the boundary of each region in the K-means image and splits the boundary into a number of traces (Figure 5.17). Each region can then be represented as a list of traces, where each trace is a list of the boundary points within the trace. Also stored with each trace is the identity of the region which is adjacent to it.

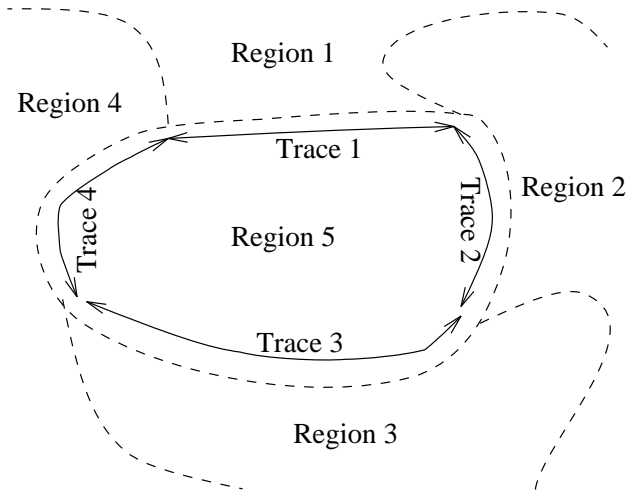


Figure 5.17: Region boundary made up of a number of traces

Other information stored is the number of points in each trace and the total number of points which make up the boundary of each region. The entire data structure representing the K-means image is shown in Figure 5.18

It is possible to trace the boundary of every region in the segmented image. However, only a small portion of the image actually contains a vehicle. So, it is sensible to only trace the regions within *regions of interest*, thereby keeping the total number of regions passed on to the next stage to a minimum. Within this implementation the regions of interest must be defined manually although a method based on motion detection via frame differencing should be able to identify the regions of interest fully automatically.

The region tracing process also performs the low level region merging which eradicates the small regions resulting from noise and texture in the image.

```

typedef struct {
    int x, y;
} Point;

typedef struct {
    Point *points; /* list of points in trace */
    int len;       /* number of points in trace */
    int neighbour; /* identity of neighbouring region */
} Trace;

typedef struct {
    Trace *traces; /* list of traces in region */
    int len;       /* total number of points in boundary of region */
} Region;

Region *regions; /* list of all regions in K-means image */

```

Figure 5.18: Data structure representing the K-means image

This representation of the K-means image not only reduces the space required to store it but, more importantly enables rapid retrieval of region description and adjacency information which is required in later processing. Once constructed, the entire data structure is passed down the Transputer link to framestore 3.

This module also inputs the original camera images from the general function module and outputs them via output port B at the same time the data structure is sent down the transputer link. Again, this is for synchronisation purposes and ensures that framestore 3 receives the original image and the corresponding concise representation of the segmented image at the same time.

5.3.5 Framestore Module 3

The *object recognition* process runs on the Transputer in this module. Using the data structure received from framestore 2 it performs region merging and then object recognition, yielding the position, pose and dimensions of the vehicle in the image. The recovered vehicle parameters are then used to project the wireframe model of the vehicle onto the original image. This image, with the vehicle model superimposed, is then sent back via output port B to the digitiser module for display on a monitor.

5.4 Results

In order to test the system a camera was setup to survey cars negotiating a T junction. The camera was at a height of 8m above the road surface, inclined at an angle of 27.5 degrees from the horizontal, where horizontal means parallel to the road plane. The camera parameters were therefore as follows (all distances measured in mm):

$$x_{cam} = 0$$

$$y_{cam} = 8000$$

$$z_{cam} = 0$$

$$\alpha_{cam} = -27.5$$

$f = 6.2$ from lens specification

$C_i = 68, C_j = 68$ from camera specification

Two cars negotiating the junction were then recorded on video and subsequently digitised at a rate of 3.77 frames/second (the maximum speed the images could be written to disk), providing one sequence of 36 frames (sequence A) and one of 46 frames (sequence B). The digitised images were stored on SCSI disk and the system reconfigured to input images from the SCSI disk rather than the digitiser. An investigation of various aspects of the system performance is presented in the following sections. As the system is striving to control the exponential size of the interpretation tree, the key aspects are factors such as the number of regions produced by the region tracing process, the number of merged image features generated, the number accepted and the number of nodes explored in the interpretation tree.

5.4.1 K-means Segmentation

The K-means segmentation is required to produce an over-segmentation for the reasons explained in Section 5.2.4. To ascertain whether this is achieved, the images from the two sequences were processed by the system and the result of the K-means segmentation of each image inspected manually. The two sequences contained a total of 82 images. Figure 5.19 shows a selection of the images together with their segmentation. A tally was kept of the number of images which were successfully over-segmented, i.e. did not contain under-segmented regions which would hinder the subsequent recognition process.

This procedure was repeated with several values of K . Table 5.2 contains the results from these tests. In Section 5.2.4 it was also stated that K must be as low as possible in order to keep the number of regions produced to a minimum. Table 5.2 shows that $K = 8$ gives a good compromise between an over-segmentation and a low number of regions and is therefore the value used in the remainder of the tests.

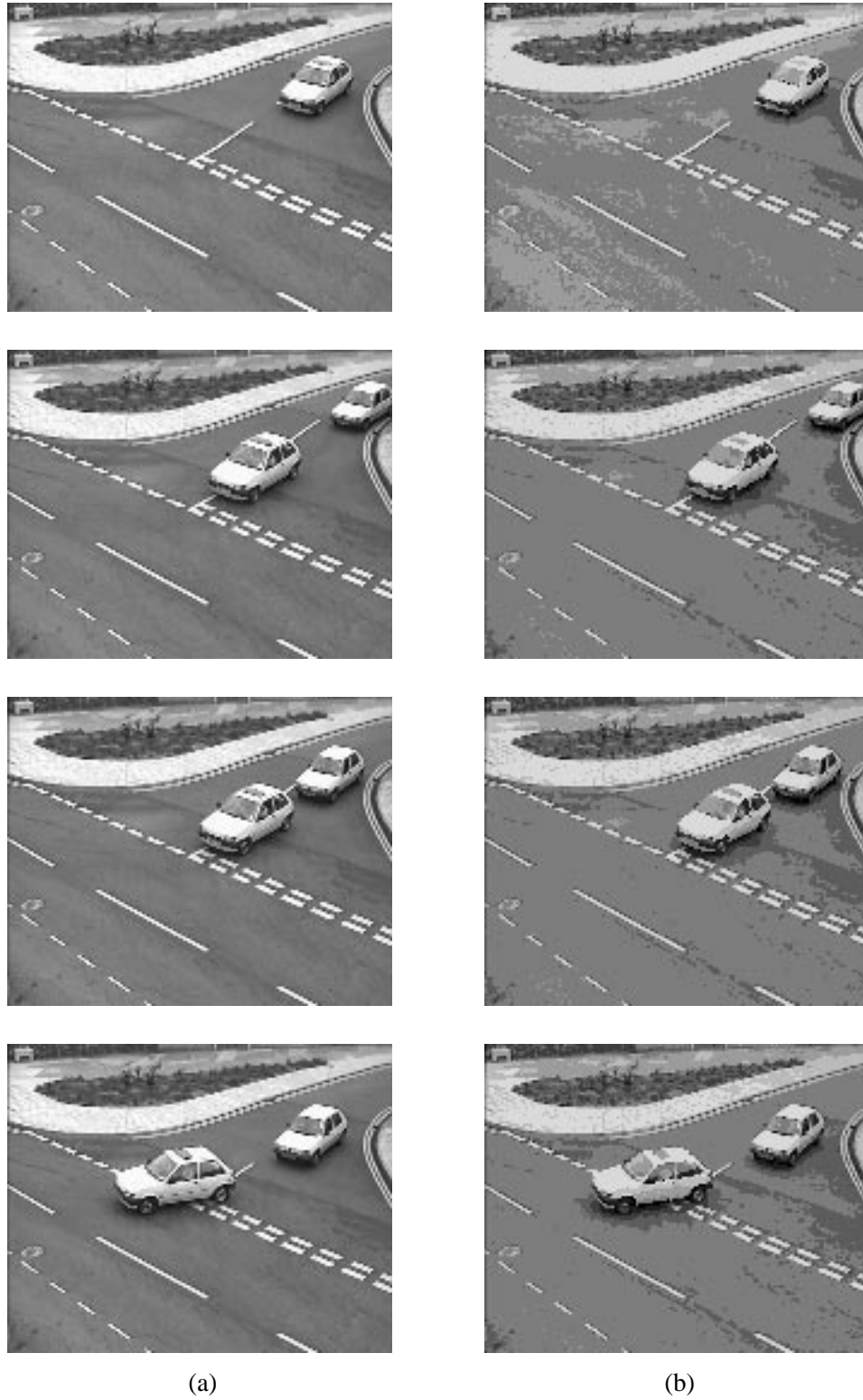


Figure 5.19: A selection of images (a) together with their K-means segmentation (b)

K	% of images over-segmented	number of regions
3	0	13.6
4	31.7	19.2
5	41.4	21.0
6	41.5	25.3
7	45.1	27.9
8	82.9	30.1
9	86.6	32.5
10	89.0	32.5

Table 5.2: Percentage of images for which over-segmentation occurs together with average number of regions produced for several values of K

5.4.2 Region Tracing

Figure 5.20 shows several images together with the traced region boundaries within the region of interest. Only regions which have a boundary length below a specified limit are passed on to the region merging process.

For the 82 images in the two sequences examined, an average of 30.1 regions per image (within the region of interest) were passed on to the region merging process.

5.4.3 Region Merging

The region merging algorithm recursively generates all possible combinations of merged features containing between 1 and max_{merged} image regions. If the merged feature looks like any of the model features, it is accepted, otherwise it is rejected. The number of accepted merged features is highly dependent on the value of max_{merged} . Table 5.3 shows the average number of merged features generated and the average number accepted per image, for several values of max_{merged} .

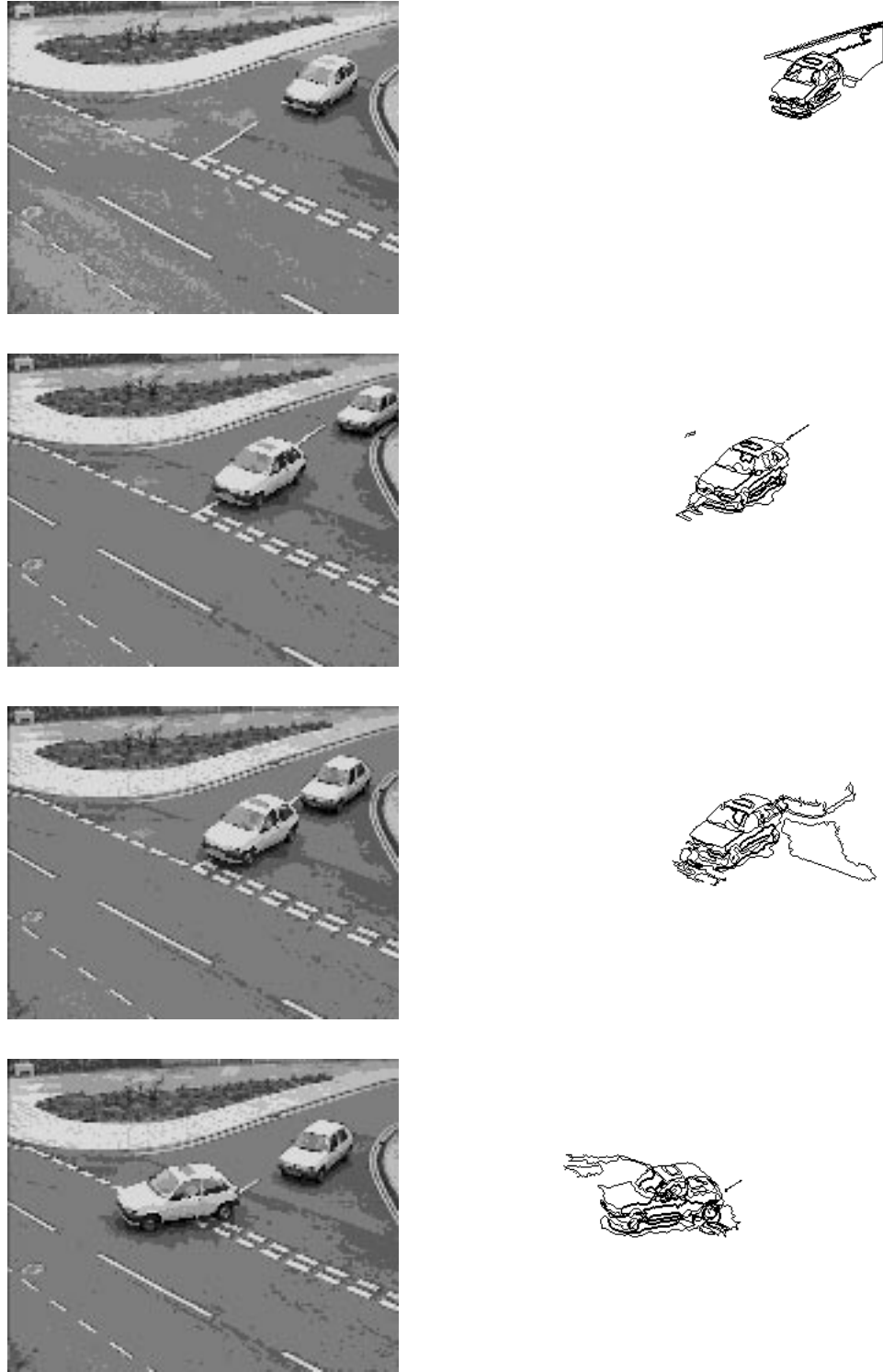


Figure 5.20: Traced region boundaries within region of interest

max_{merged}	Number generated	Number accepted
1	28.7	7.2
2	79.5	34.4
3	190.9	87.61

Table 5.3: Average number of generated and average number of accepted merged features per image for several values of max_{merged}

5.4.4 Vehicle Recognition

For vehicle recognition to occur the K-means process must have provided an over-segmentation and max_{merged} must be large enough to enable suitable merged image features to be generated. Table 5.4 shows the percentage of images in which the vehicle was recognised for several values of max_{merged} . For $max_{merged} = 1$, the recognition accuracy is very poor. There is a drastic improvement when max_{merged} is increased to 2 and a further, slight improvement when $max_{merged} = 3$. It is likely that $max_{merged} = 4$ would lead to another slight improvement although the recognition accuracy appears to be approaching a ceiling with $max_{merged} = 2$ and 3. As will be seen in Section 5.4.6 the execution time prohibits running the tests with $max_{merged} = 4$. Table 5.4 also shows the average VIRM score for each image. A higher VIRM score means that more image features were successfully matched to the model which in turn means a more accurate recovery of the model parameters. Therefore, a value of $max_{merged} = 3$ was used for the remainder of the tests.

max_{merged}	% of images with vehicle recognised	Average VIRM score
1	15	1.7
2	83	2.9
3	84	3.1

Table 5.4: Percentage of images in which the vehicle was correctly recognised together with average VIRM score per image for several values of max_{merged}

Figure 5.21 shows both the theoretical maximum and the actual size of the interpretation

tree plotted against the number of accepted merged image regions.

Figure 5.22 shows the recognised vehicle overlaid on a selection of the original images from the two sequences.

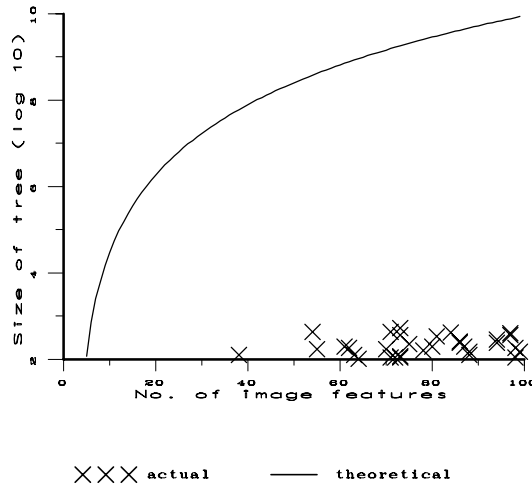
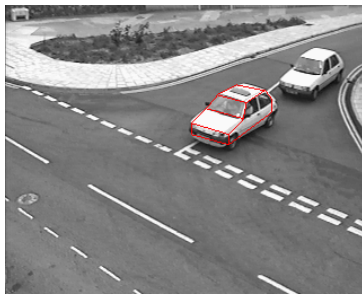
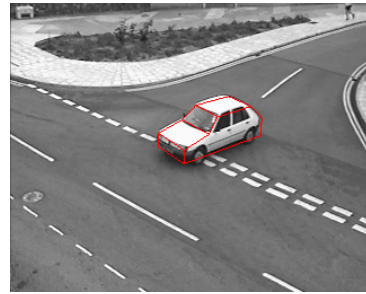
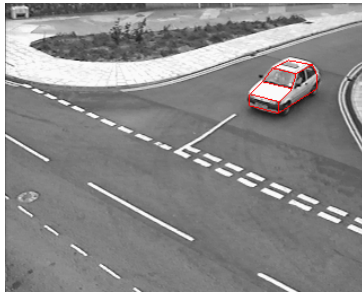


Figure 5.21: Theoretical maximum and actual size of interpretation tree against number of image regions

5.4.5 Vehicle Parameters

The recovered vehicle parameters may be used to extract among other things, the vehicle path, direction and speed. Figures 5.23 and 5.24 show the vehicle path overlaid on a map of the T-junction for the vehicle in sequences A and B respectively. Figures 5.25 and 5.26 show plots of speed both raw and after filtering with a running mean (window length 3) for the vehicle in sequences A and B respectively. In these plots the raw speed values contain significant noise but the running mean does a good job of filtering this out. The noise is due to errors in the recovered vehicle parameters. An indication of the size of these errors can be derived by calculating the average difference between the raw values and the filtered values. This gives an average error of 1.53 km/h over the two plots. This represents an average error of 11 cm in the recovered position of the vehicle per sample. Things appear



Sequence A

Sequence B

Figure 5.22: Recognised vehicle overlaid on a selection of images from sequences A and B

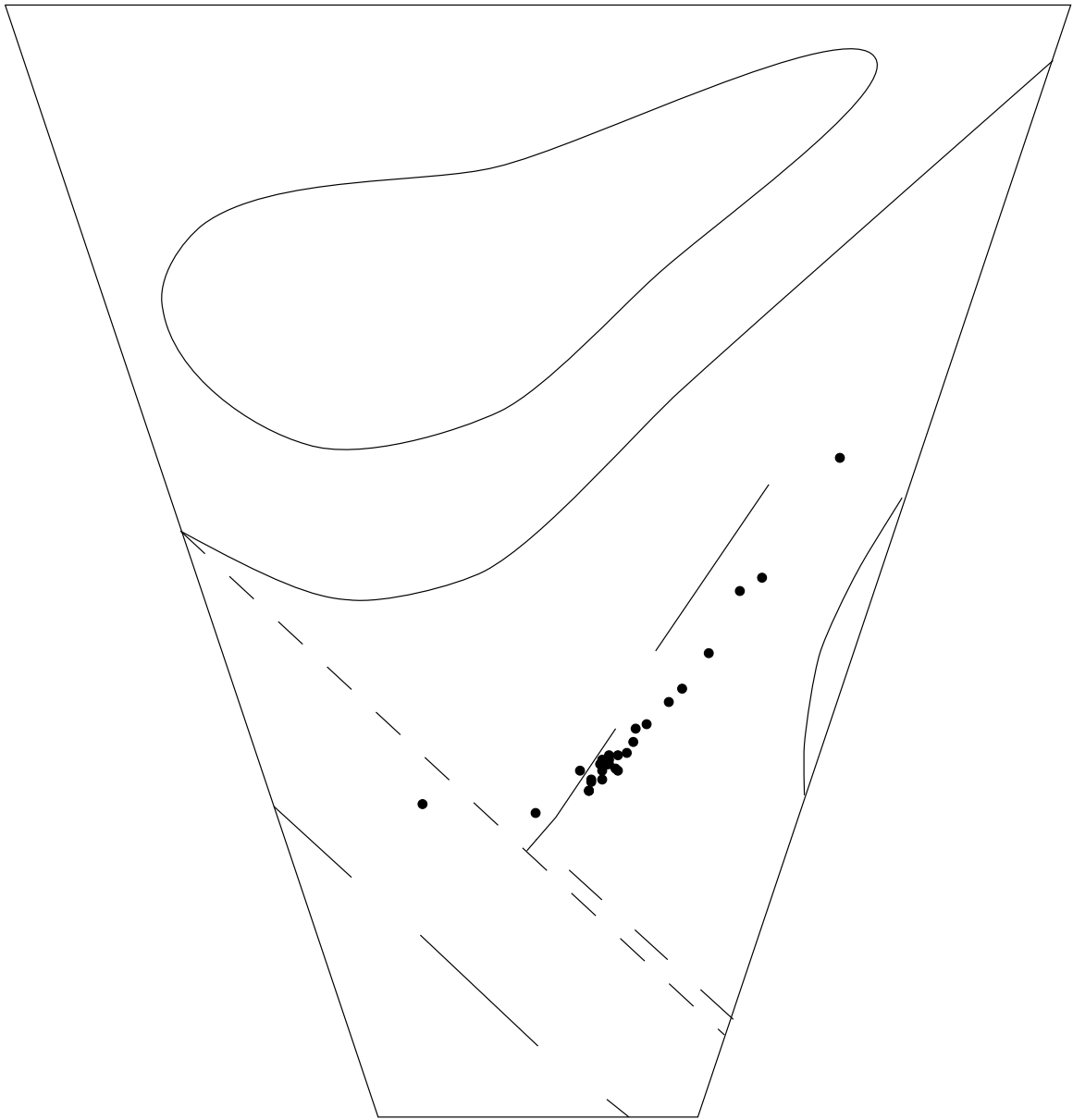


Figure 5.23: Vehicle path overlaid on map of T-junction for vehicle in sequence A

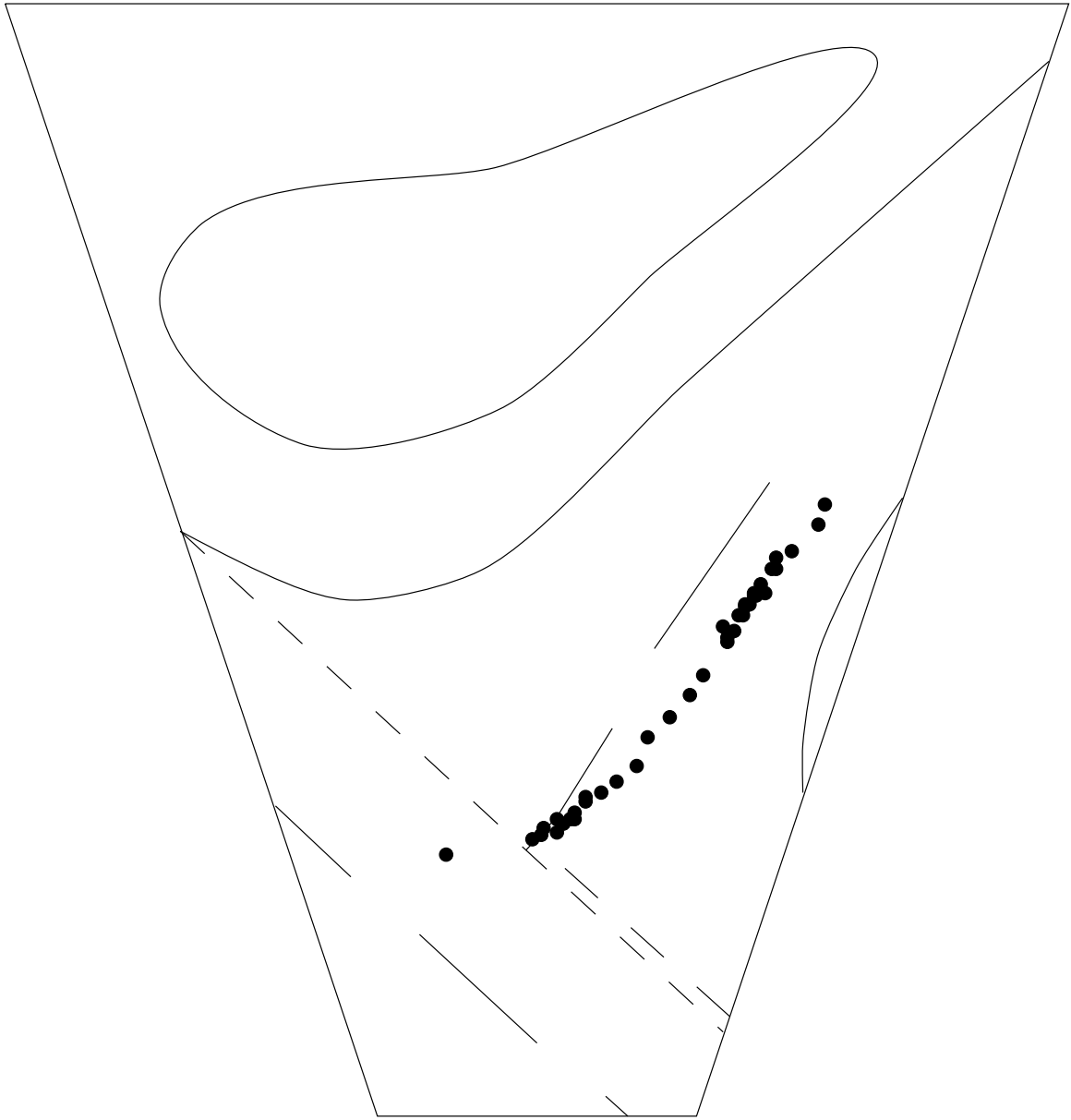


Figure 5.24: Vehicle path overlaid on map of T-junction for vehicle in sequence B

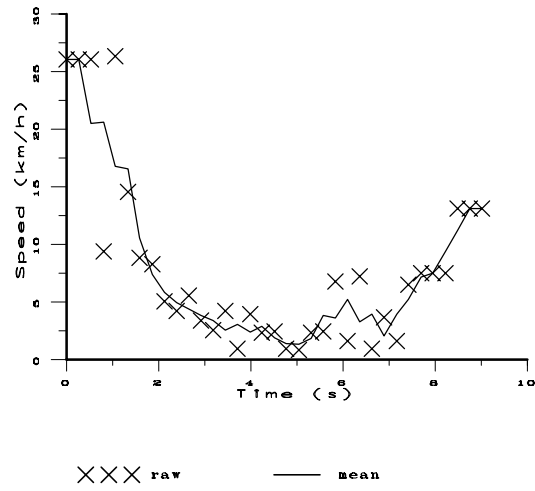


Figure 5.25: Vehicle speed against time for vehicle in sequence A

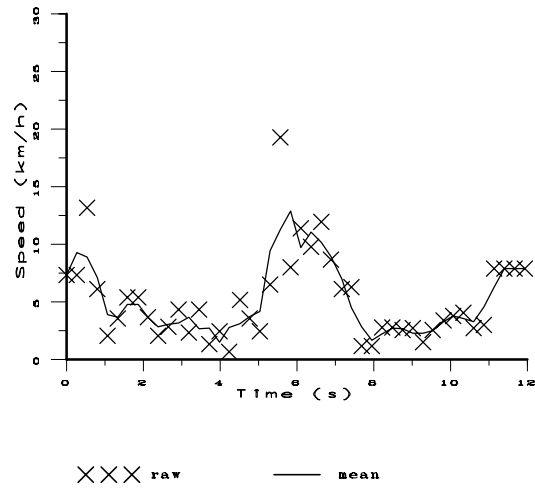


Figure 5.26: Vehicle speed against time for vehicle in sequence B

worse than they are due to a combination of the high sampling rate (3.77 frames/s) and the relatively low vehicle speed which combine to give small vehicle movement between samples, typically less than 1.0 metre. The error in the recovered vehicle parameters therefore gives a significant percentage error in the speed. However, as the size of the error is independent of speed, if the vehicle was travelling more quickly or the sampling rate was lower such that the vehicle movement was greater between samples, the percentage speed error would be smaller.

5.4.6 Execution Times

Table 5.5 shows the average execution time per image for each stage of the system. The average times were measured over 82 images from the two sequences.

	Time (s)
K-means segmentation	0.04
Region tracing	3.72
Region merging	1519.54
Vehicle recognition	9934.71

Table 5.5: Average execution time per image for each stage of the traffic monitoring sensor

Although the K-means segmentation provides a new image every 0.04 seconds (25 frames/s), the K means are only updated every 0.09 seconds. This is because the K-means process runs independently on framestore 1 from the mapping operation carried out by the general function module. The general function module is able to perform the mapping from pixel values to region labels at frame rates even though the mapping function is only updated every 0.09 seconds.

The time between images emerging from the pipeline is governed by the slowest stage of the pipeline which is clearly the region merging and vehicle recognition stage, the time being 3 hours, 10.9 minutes.

5.5 Conclusions

The segmentation quality of the K-means process is adequate but not exceptional. It is likely that region growing techniques would better the 83% of images in which over-segmentation occurs. The redeeming factor of the K-means algorithm is that it may be neatly implemented on the vision hardware. The low-level, but slow, operations are performed in hardware by the look-up tables on the general function module enabling the segmentation to be performed at frame rates.

Figure 5.21 illustrates the effectiveness of the pruning strategy employed in the recognition algorithm. Even for cases where up to 99 merged features were searched for the optimal mapping, the maximum size of interpretation tree generated was 529 nodes. Had the complete tree been generated it would have contained 7.6×10^9 nodes.

The best vehicle recognition rate was 84%, achieved with $max_{merged} = 3$. This recognition rate seems to be approaching the limit dictated by the success of the segmentation. There is a close correlation between the images in which the vehicle was not recognised and the images in which under-segmentation occurred. If the under-segmented images are omitted from the test, the recognition accuracy increases to 94%.

The recovered vehicle parameters were used to plot both vehicle path and vehicle speed. After filtering the raw speed values with a running mean, the speed plot and path plot for each vehicle agree closely. The vehicle in sequence A enters the scene relatively quickly, slows down and stops at the white line before accelerating away from the junction. This behaviour is indicated by the speed plot and the path plot which shows a greater density of points where the vehicle is travelling slowly. The vehicle in sequence B enters the scene quickly and then slows down as it approaches a second car already stopped at the junction. When the second car moves out of the way it accelerates again before slowing and stopping at the junction. Finally it accelerates away from the junction. Again this ties up with the speed plot and the path plot which shows two areas of dense points where the vehicle has been travelling slowly or stopped.

The difference between the raw and filtered speed indicated that the average error in the

recovered vehicle position was about 11 cm. This is quite impressive considering the average distance between the vehicle and the camera was about 20 m.

5.6 Summary

This chapter has presented the design and implementation of a generic road-traffic monitoring sensor. It is based on an object recognition algorithm which locates vehicles in images of road scenes by searching correspondence space. The algorithm for searching correspondence space rapidly becomes intractable as the number of image and model features increases. The problem becomes worse by an additional factorial factor if the case of a many to one mapping is considered. In real problems the many to one mapping *must* be considered as the ideal segmentation required for a one to one mapping is impossible to achieve.

The novel algorithm presented here overcomes these problems via two routes. Firstly, knowledge of the vehicle's appearance is used to intelligently merge an over-segmented image, thus providing an ideal segmentation. This allows the computationally less complex case of a one to one mapping to be considered. The merging process is also able to filter out the majority of merged features on the basis that they could not possibly represent any of the model features. This results in a drastic reduction in the number of features which must be searched for the optimal mapping.

Secondly, the search for the one to one mapping is made tractable by the use of a new algorithm for generating the interpretation tree together with a pruning strategy which utilises two different models to effectively prune the tree enabling the optimal mapping to be found.

At various stages in the recognition process, the pose of the geometric model must be recovered so as to register the projected model with the mapped image features. A new, multi-stage method for pose recovery has been developed which progressively refines the model pose by optimising a number of carefully designed error functions.

The implementation has achieved excellent results, recognising 84% of vehicles in two sequences of test images. For each recognised vehicle the recovered pose has been used to plot both the vehicle's path on a plan view of the road junction and its speed against time. It is interesting to note that the recovered model parameters may be used to reconstruct the road scene not merely as a plan view but as a full three dimensional *virtual* traffic scene, viewable from any position, even the driving seat of one of the vehicles. Unfortunately the implementation is far from real time. However, the advances to the object recognition algorithm which have been presented here have turned it from an intractable problem to a tractable one for a *real life problem*. This is a significant result.

Chapter 6

Conclusions

This thesis has presented two road-traffic monitoring systems, implemented using computer vision techniques on a parallel vision system developed at the University of Bristol.

The number-plate recognition system features an automatic trigger. This enables the system to operate in a completely stand-alone manner, requiring no external sensors. This helps to minimise both costs and disruption during installation. Avoiding the use of external sensors such as tubes or loop detectors means the system is not vulnerable to roadworks or resurfacing, while fewer system components means increased reliability. The automatic trigger has been shown to correctly generate a trigger for nearly 95% of the 4357 vehicles on which it was tested.

The new algorithm for locating the number-plate does not rely on any form of thresholding. This enables it to operate under a far greater range of lighting conditions including cases where parts of the plate are in strong shadow. Dirt or cracks on the plate cause far fewer problems than for the threshold approach. The benefits of the non-threshold approach are illustrated by the new algorithm which successfully locates the number-plate in 99.2% of the 3000 test images. It is notable that the test images had not been filtered to remove *unfair* examples in which the plate was dirty, broken, cracked, hanging at an angle or in strong shadow.

The neural network in combination with a new syntax checking algorithm provides a robust method for deciphering the plates. The large neural network architecture achieved the best performance in terms of both recognition accuracy and speed compared to the small neural network or template matching. The new syntax checking algorithm offers great flexibility as it allows user-definable syntax rules. This means the system should be capable of operation abroad where the number-plates have a different format (although the neural network may need retraining if the number-plate fonts are significantly different).

The system as a whole was tested on 3000 images for which the number-plates had been manually deciphered to provide a ground truth. Of these plates, 85.4% were read completely correctly by the system. The average execution time of 3.18 seconds per plate is sufficient for many applications such as automatic tolls, car parks or monitoring of less densely populated roads. For more demanding applications, the design of the software as a set of communicating sequential processes enables easy extension of the image processing pipeline by simply adding an extra framestore module and redistributing the processes. This would result in the average execution time reducing to 2.29 seconds.

The generic road-traffic monitoring sensor has been shown capable of tracking vehicles from two test sequences containing a total of 82 images. The model based approach is able to recognise vehicles as oppose to the non-model based approach which merely tracks groups of image pixels without understanding what they actually represent. The vehicle was correctly recognised in 84% of the test images and its position and pose in the real-world recovered. The recovered parameters were used to plot the vehicles' paths on a plan view of the road junction and their speeds against time. The new object recognition algorithm is based on searching correspondence space rather than parameter space. This avoids the need of the parameter space approach to generate accurate initial values for the vehicle parameters in order to latch on to a correct match. Unfortunately, the correspondence space approach is inherently an intractable problem as it involves searching an exponentially sized interpretation tree. The great success of the object recognition algorithm developed here is in its pruning capability. In cases where the complete interpretation tree had 7.6×10^9 nodes, a maximum of 529 nodes were actually explored. This enables the system to achieve good recognition results with complex images from a real traffic scene.

6.1 Future Work

The number-plate recognition system would benefit from a further investigation to find the optimum neural network architecture and training strategy for character recognition. Perhaps a more up to date training set including real negative examples rather than the artificial ones generated for this work would improve the recognition accuracy. Redistribution of the software processes onto more processors would improve speed if this was necessary.

For the generic road-traffic monitoring sensor, a more highly parameterised vehicle model would be beneficial. The extra parameters would provide increased flexibility in the shape of the model allowing it to fit more precisely to various different types of vehicle.

Although the algorithm presented here enables an efficient exploration of the interpretation tree, there is scope for further improvements. The interpretation tree contains many common sub-trees. As these are explored, calculations are effectively being repeated. In order to reduce the occurrences of such common sub-trees, the following modified algorithm is proposed:

1. Generate merged image features using the geometric model to accept or reject each one. Pass the accepted, merged features on to the next stage.
2. Generate all possible mappings containing a pair of merged features. Use the view independent relational model and the geometric model to accept or reject each mapping. Pass the accepted mappings on to the next stage.
3. Generate all possible mappings which are a union of two mappings accepted in the previous stage. Use the view independent relational model and the geometric model to accept or reject each mapping.
4. Repeat stage 3 until no further mappings may be generated.

6.2 Contribution of this Work

This work has made the following contributions to the computer vision and traffic monitoring communities:

- An automatic trigger based on an extremely fast algorithm for detecting when vehicles pass a specified point on the road.
- A highly robust and fast algorithm for finding number-plates in images even if the plate is dirty, broken, cracked or hanging at an angle.
- A syntax checker which when coupled with the output of a neural network, deciphers number-plates. The syntax checker is programmable such that an operator may define the format of allowable number-plates.
- An algorithm for recognising vehicles in images of traffic scenes. The algorithm efficiently searches the pruned interpretation tree to find the optimal mapping between image and model features.
- A method for recovering a vehicle's position and pose parameters from a set of image features mapped to model features. The method utilises a number of carefully designed error functions together with optimisation techniques in order to recover the vehicle model's parameters.

Bibliography

- [1] V K Govindan and A P Shivaprasad. Character recognition - a review. *Pattern Recognition*, 23(7):671–683, 1990.
- [2] Richard Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.
- [3] Dan Hammerstrom. Neural networks at work. *IEEE Spectrum*, pages 26–32, June 1993.
- [4] Dan Hammerstrom. Working with neural networks. *IEEE Spectrum*, pages 46–53, June 1993.
- [5] Elsydel Ltd., Elsydel Head Office, 63 Boulevard Bessiers, 75017 Paris, France. *Information Leaflet*, 1988.
- [6] P G Williams, H R Kirby, F O Montgomery, and R D Boyle. Evaluation of video-recognition equipment for number-plate matching. In *2nd. International Conference on Road Traffic Monitoring*. Institute of Electrical Engineers, February 1989.
- [7] M M M Fahmy. Computer vision application to automatic number-plate recognition. In *Proceedings of 26th. International Symposium on Automotive Technology and Automation*, pages 625–633, Aachen, Germany, 1993.
- [8] Glen Auty, Peter Corke, Paul Dunn, Murray Jensen, Ian Macintyre, Dennis Mills, Hao Nguyen, and Ben Simons. An image aquisition system for traffic monitoring appilcations. *SPIE: Cameras and Systems for Electronic Photography and Scientific Imaging*, 2416:118–133, February 1995.

- [9] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis and Machine Vision*, pages 510–512. Chapman & Hall, 1993.
- [10] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis and Machine Vision*, pages 524–533. Chapman & Hall, 1993.
- [11] Rafael Inigo. Traffic monitoring and control using machine vision: A survey. *IEEE Transactions on Industrial Electronics*, IE-32(3):177–185, August 1985.
- [12] K W Dickinson and C L Wan. Road traffic monitoring using the Trip II system. In *2nd. Int. Conf. Road Traffic Monitoring*, pages 56–60, London, February 1989. IEE.
- [13] Neil Hoose. *Computer Image Processing in Traffic Engineering*. Research Studies Press, 1991.
- [14] S Takaba et al. Measurement of traffic flow using real-time processing of moving pictures. In *32nd. Conf. on Vehicular Technology*, pages 488–494, San Diego, CA, 1982.
- [15] T Abramczuk. A microcomputer based TV detector for road traffic. In *Symposium on Road Research Program*, Tokyo, Japan, October 1984.
- [16] Rafael M Inigo. Application of machine vision to traffic monitoring and control. *IEEE Transactions on Vehicular Technology*, 38(3):112–122, August 1989.
- [17] A T Ali and E L Dagless. Computer vision for automatic road traffic analysis. In *Int. Conf. on Automation, Robotics and Computer Vision*, September 1990.
- [18] A T Ali and E L Dagless. Automatic traffic monitoring using a transputer image processing system. In *2nd. Int. Conf. on Transputer Applications*, pages 1–8, July 1990.
- [19] A T Ali and E L Dagless. Computer vision-aided road traffic monitoring. In *24th. Int. Conf. on Road Transport Informatics & Intelligent Vehicle-Highway Systems*. ISATA, May 1991.
- [20] A T Ali and E L Dagless. A parallel processing model for real-time computer vision-aided road traffic monitoring. *Parallel Processing Letters*, 2(2):257–264, 1992.

- [21] J Bulas-Cruz, A T Ali, and E L Dagless. Real-time motion detection and tracking. In *8th. Scandanavian Conference on Image Analysis*, May 1993.
- [22] A T Ali, J Bulas-Cruz, and E L Dagless. Vision based road traffic data collection. In *Proc. ISATA 26th. International Conference*, pages 609–616, September 1993.
- [23] J Bulas-Cruz, A T Ali, and E L Dagless. A temporal smoothing technique for real-time motion detection. In *CAIP*, September 1993.
- [24] E L Dagless, D Milford, J Bulas-Cruz, M Priestley, and D Markham. Image processing hardware and algorithms. In *Proceedings of the 5th. School of Computer Vision and Graphics*, pages 9–34, Zakopane, Poland, February 1994.
- [25] M Pout, M Priestley, and D Markham. Vision processing hardware: User guide. Technical report, University of Bristol, UK, 1993.
- [26] J Versavel, F Lemaire, and D Van der Stede. Camera and computer-aided traffic sensor. In *2nd. Int. Conf. Road Traffic Monitoring*, pages 66–70, London, February 1989. IEE.
- [27] P Briquet. Video processing applied to road and urban traffic monitoring. In *6th. Int. Conf. Road Traffic Monitoring*, pages 153–157, London, April 1992. IEE.
- [28] N Hoose. Queue detection using computer image processing. In *2nd. Int. Conf. Road Traffic Monitoring*, pages 94–98, London, February 1989. IEE.
- [29] D M Kelly. Results of field trial of the IMPACTS image processing system for traffic monitoring. In *6th. Int. Conf. Road Traffic Monitoring*, pages 137–142, London, April 1992. IEE.
- [30] M Fathy and M Y Siyal. A real-time image processing approach to measure traffic queue parameters. *IEE proceedings. Vision, image and signal processing*, 142(5):297–303, October 1995.
- [31] J Malik, J Weber, Q T Luong, and D Koller. Smart cars and smart roads. In *Proceedings 6th. British Machine Vision Conference*, pages 367–381, Birmingham, England, 1995.
- [32] Dieter Koller, Joseph Weber, and Jitendra Malik. Robust multiple car tracking with occlusion reasoning. In *European Conf. Computer Vision*, 1994.

- [33] D Koller, J Weber, T Huang, J Malik, G Ogasawara, B Rao, and S Russel. Towards robust automatic traffic scene analysis in real-time. In *Int. Conf. Pattern Recognition*, pages 126–131, Jerusalem, Israel, October 1994.
- [34] Dieter Koller, Joseph Weber, and Jitendra Malik. Towards real-time visual based tracking in cluttered traffic scenes. In *Intelligent Vehicles Symposium*, pages 201–206, Paris, October 1994.
- [35] B Beymer, P McLauchlan, B Coifman, and J Malik. A real-time computer vision system for measuring traffic parameters. In *CVPR, 1997*. Submitted to.
- [36] Paul L Rosin and Tim Ellis. Image difference threshold strategies and shadow detection. In *6th British Machine Vision Conference*, pages 347–356, Birmingham, England, 1995.
- [37] Matthew T Cornish and Jonathon P Wakefield. Automatically locating an area of interest and maintaining a reference image to aid the real-time tracking of objects. In *Proceedings of British Machine Vision Conference*, pages 475–484, September 1996.
- [38] Panos G Michalopoulos. Vehicle detection through image processing: The autoscope system. *IEEE Transactions on Vehicular Technology*, 40(1):21–29, 1991.
- [39] Brian Carlson. Clearing the congestion: Vision makes traffic control intelligent. *Advanced Imaging*, 12(2):54–56, February 1997.
- [40] Chris G Perrott and Leonard G C Hamey. Object recognition, a survey of the literature. Technical Report 91-0065C, School of MPCE, Macquarie University, NSW 2109 Australia, January 1991.
- [41] D Marr. *Vision*. W H Freeman, 1982.
- [42] C Goad. Special purpose automatic programming for 3D model-based vision. In *Image Understanding Workshop*, pages 94–104, 1983.
- [43] D Lowe. *Perceptual Organisation and Visual Recognition*. Kluwer Academic Publishers, 1985.
- [44] D Koller, K Daniilidis, T Thorhallson, and H Nagel. Model-based object tracking in traffic scenes. In *European Conf. Computer Vision*, pages 437–452, S. Margherita, Ligure, Italy, May 1992. Springer-Verlag.

- [45] D Koller, K Daniilidis, and H H Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10(3):257–281, 1993.
- [46] E C Di Mauro, T F Cootes, C J Taylor, and A Lanitis. Active shape model search using pairwise geometric histograms. In *Proceedings of British Machine Vision Conference*, pages 353–362, Edinburgh, UK, September 1996.
- [47] R M Bodington, G D Sullivan, and K D Baker. Experiments on the use of the ATMS to label features for object recognition. In *European Conf. Computer Vision*, pages 642–551, Springer-Verlag, Berlin, 1990.
- [48] R M Bodington, G D Sullivan, and K D Baker. The consistent labelling of image features using an ATMS. *Image and Vision Computing*, 7(1):7–12, February 1989.
- [49] S Zhang, L Du, G D Sullivan, and K D Baker. Model based 3D grouping by using 2D cues. In *Proceedings of British Machine Vision Conference*, pages 217–222, Oxford, UK, September 1990.
- [50] L Du, G D Sullivan, and K D Baker. 3D grouping by viewpoint consistency ascent. *Image and Vision Computing*, 10(5):301–307, June 1992.
- [51] S Zhang, G D Sullivan, and K D Baker. Relational model construction and 3D object recognition from single 2D monochromatic image. *Image and Vision Computing*, 10(5):313–317, June 1992.
- [52] S Zhang, Geoff D Sullivan, and Keith D Baker. Using automatically constructed view-independent relational model in 3D object recognition. In *European Conf. Computer Vision*, pages 778–786, Santa Margherita, Ligure, Italy, May 1992. Springer-Verlag.
- [53] S Zhang, G D Sullivan, and K D Baker. The automatic construction of a view-independent relational model for 3-d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):531–544, June 1993.
- [54] G D Sullivan and K D Baker. Model based vision for road traffic understanding. In *26th. Int. Symposium on Automotive Technology and Automation. Advanced Transport Telematics / Intelligent Vehicle Highway Systems*, pages 559–567, Aachen, Germany, September 1993. ISATA.

- [55] A D Worrall, G D Sullivan, and K D Baker. Pose refinement for active models using forces in 3D. In *Proceedings of the Third European Conference on Computer Vision*, pages 341–350, May 1994.
- [56] S J Maybank, A D Worrall, and G D Sullivan. Filter for car tracking based on acceleration and steering angle. In *Proceedings of British Machine Vision Conference*, pages 615–624, September 1996.
- [57] G D Sullivan, A D Worrall, and K D Baker. Visual object recognition using deformable models of vehicles. In *Workshop on Context-based Vision*, pages 75–86, Boston, USA, July 1995. IEEE.
- [58] J M Ferryman, A D Worrall, G D Sullivan, and K D Baker. A generic deformable model for vehicle recognition. In *Proceedings of British Machine Vision Conference*, pages 127–136, Birmingham, September 1995.
- [59] Datacube Inc. *MAXbus Specification*, December 1987. Doc No SP00-4.
- [60] M J Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, 1972.
- [61] Alan Chalmers and Jonathon Tidmus. *Practical Parallel Processing*. Thomson Computer Press, 1996.
- [62] C A R Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [63] Jose Bulas Cruz, Richard Storer, Dave J Milford, and Erik L Dagless. Developing embedded applications in an array of specialised transputer modules. In *WOTUG 17*, April 1994.
- [64] Jose Bulas Cruz. *Image Processing Applications using a Transputer-based System*. PhD thesis, Bristol University, Bristol, UK, July 1995.
- [65] R A Lotufo, A D Morgan, A S Johnson, and B T Thomas. A transputer based automatic number-plate recognition system. In *Proceedings 2nd. International Conference on Applications of Transputers*, July 1990.
- [66] A D Morgan and R A Lotufo. A description of the number-plate recognition algorithms. Technical report, Department of Computer Science, University of Bristol, April 1990.

- [67] Pavel Zemcik and Erik L Dagless. Printing grey scale images on a fax machine. *Microprocessors and Microsystems*, 18(5):271–279, June 1994.
- [68] *SNNS, Stuttgart Neural Network Simulator, User Manual, Version 4.0*. Breitwiesenstrasse 20-22, 70565 Stuttgart, Fed. Rep. of Germany, 1995.
- [69] Dana H Ballard and Christopher M Brown. *Computer Vision*, chapter 5. Prentice-Hall, 1982.
- [70] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, chapter 10. Cambridge University Press, 1992.

Appendix A

Scaling by Linear Interpolation

The number-plate recognition system requires an image of a number-plate to be scaled equally in each direction such that the new height of the number-plate is the same as the height of the stored templates, while maintaining the aspect ratio of the characters within the plate.

Given T_h is the height of the stored template in pixels, P_w, P_h are the width and height of the number-plate and S_w, S_h are the width and height of the scaled plate. The new dimensions of the scaled plate, S_w, S_h , may be calculated as follows:

$$S_h = T_h \tag{A.1}$$

and,

$$\frac{P_h}{P_w} = \frac{S_h}{S_w}$$
$$S_w = S_h \frac{P_w}{P_h}$$
$$S_w = T_h \frac{P_w}{P_h} \tag{A.2}$$

The relationship between a point in the scaled plate, (i, j) and the corresponding point in the original plate, (x, y) is:

$$x = i \frac{P_w}{S_w} \tag{A.3}$$

$$y = j \frac{P_h}{S_h} \tag{A.4}$$

It is clear that integer values of i, j will more often than not correspond to non-integer values of x, y . As the pixel data in the original plate is not continuous, linear interpolation must be performed between the nearest pixels (Figure A.1).

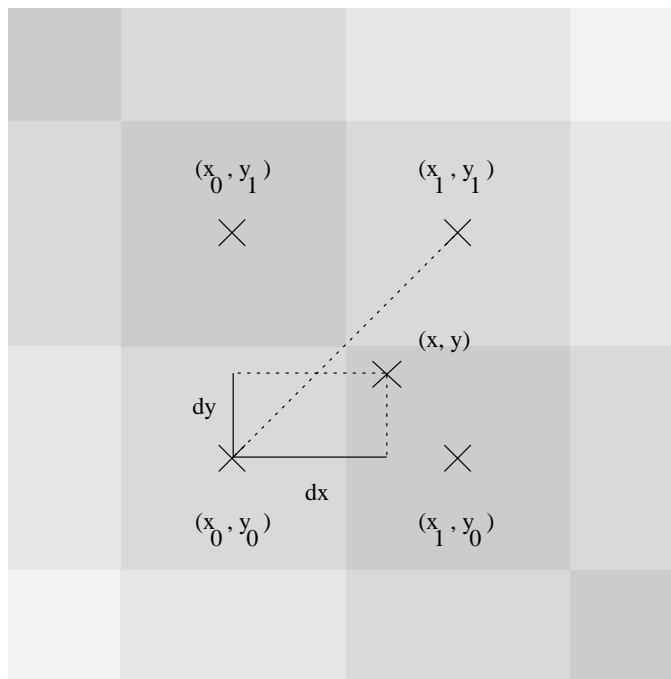


Figure A.1: Intensity at (x, y) must be derived by interpolating between nearest pixels

Given (x, y) ,

$$x_0 = x \text{ rounded down to the nearest integer} \quad (\text{A.5})$$

$$x_1 = x_0 + 1 \quad (\text{A.6})$$

$$y_0 = y \text{ rounded down to the nearest integer} \quad (\text{A.7})$$

$$y_1 = y_0 + 1 \quad (\text{A.8})$$

if $p(x, y)$ is the intensity value of the plate at (x, y) ,

if $dx \geq dy$ then,

$$p(x, y) = p(x_0, y_0) + dx(p(x_1, y_0) - p(x_0, y_0)) + dy(p(x_1, y_1) - p(x_1, y_0)) \quad (\text{A.9})$$

otherwise,

$$p(x, y) = p(x_0, y_0) + dx(p(x_1, y_1) - p(x_0, y_1)) + dy(p(x_0, y_1) - p(x_0, y_0)) \quad (\text{A.10})$$

The following algorithm may then be used to calculate the pixel values, $s(i, j)$, in the scaled plate:

```
Sh = Th
Sw = Th * Pw/Ph
for i=0 to Sw {
  for j=0 to Sh {
    x=i * Pw/Sw
    y=j * Ph/Sh
    x0 = x rounded down to the nearest integer
    x1=x0+1
    y0 = y rounded down to the nearest integer
    y1=y0+1
    dx=x-x0
    dy=y-y0
    if dx >= dy then
      s(i,j)=p(x0,y0) + dx * (p(x1,y0)-p(x0,y0)) + dy * (p(x1,y1)-p(x1,y0))
    else
      s(i,j)=p(x0,y0) + dx * (p(x1,y1)-p(x0,y1)) + dy * (p(x0,y1)-p(x0,y0))
  }
}
```

Appendix B

Projection of a 3D Model into the Image

To project the wireframe model of a vehicle into the image, the model must first be instantiated with the given model parameters in the real world coordinate system. The vehicle model is stored as a list of lines defined by the coordinates of the vertices at their ends. The coordinates of the vertices are relative to the vehicle's coordinate system (Figure B.1). The length, width and height (l, w, h) parameters of the model are used to instantiate the model within its own coordinate system.

The position and pose ($x_{mod}, z_{mod}, \alpha_{mod}$) parameters of the model are then used to project the vertices from the vehicle's coordinate system into the real world coordinate system. A point in the vehicle's coordinate system (x_v, y_v, z_v) is related to a point in the real world (x_r, y_r, z_r) as follows:

$$\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} = \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} \begin{pmatrix} \cos \alpha_{mod} & 0 & \sin \alpha_{mod} \\ 0 & 1 & 0 \\ -\sin \alpha_{mod} & 0 & \cos \alpha_{mod} \end{pmatrix} + \begin{pmatrix} x_{mod} \\ y_{mod} \\ z_{mod} \end{pmatrix} \quad (\text{B.1})$$

Each vertex in the real world must then be projected into the camera's coordinate system (Figure B.2). The relationship between a point (x_r, y_r, z_r) in the real world and a point (x_c, y_c, z_c) in the camera's coordinate system is as follows:

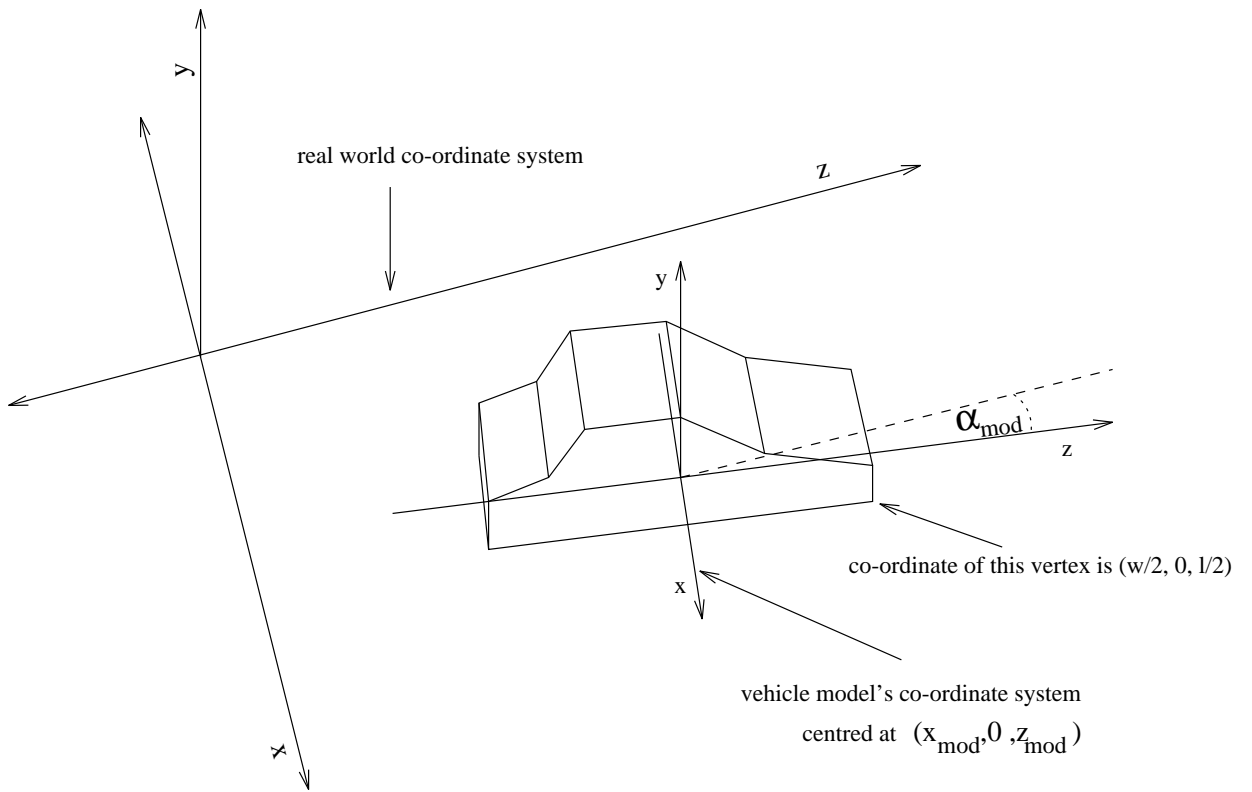


Figure B.1: Vehicle model's coordinate system

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \left[\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} - \begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{pmatrix} \right] \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha_{cam} & -\sin \alpha_{cam} \\ 0 & \sin \alpha_{cam} & \cos \alpha_{cam} \end{pmatrix} \quad (\text{B.2})$$

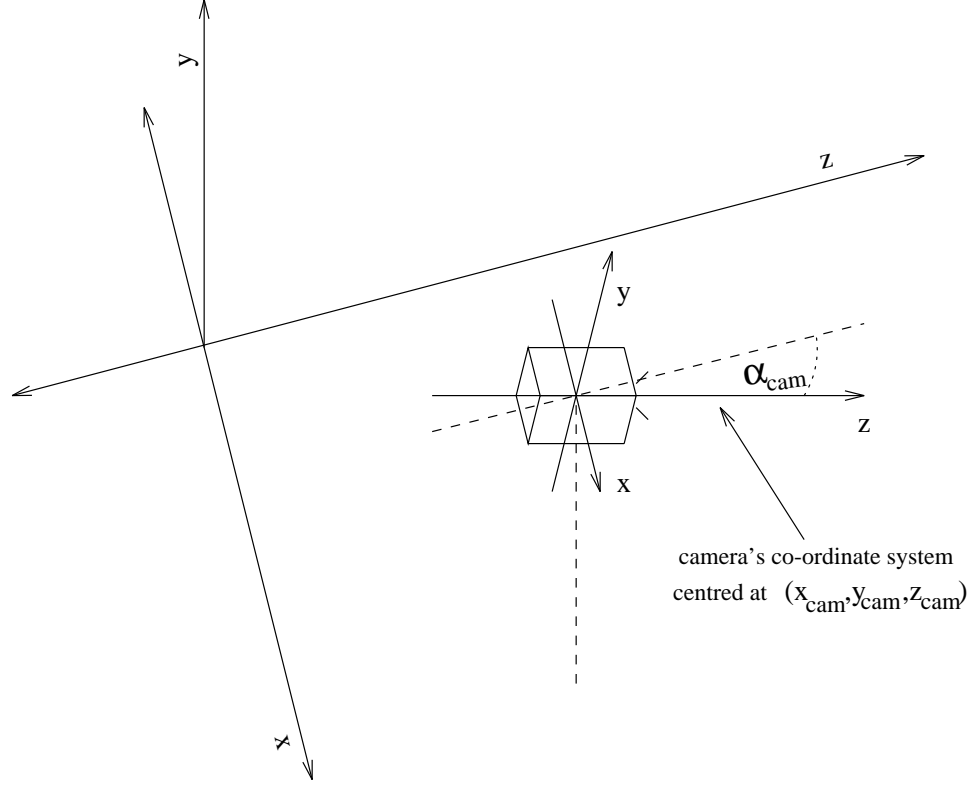


Figure B.2: The camera's coordinate system

Each vertex in the camera's coordinate system must then be projected into the image plane. Using the notation in Figure B.3, the relationship between a point (x_c, y_c, z_c) in the camera's coordinate system and a point (i, j) in the image is as follows:

$$i = \frac{x_c}{z_c} f C_i \quad (\text{B.3})$$

$$j = \frac{y_c}{z_c} f C_j \quad (\text{B.4})$$

The above equations are used to project all model vertices to points in the image.

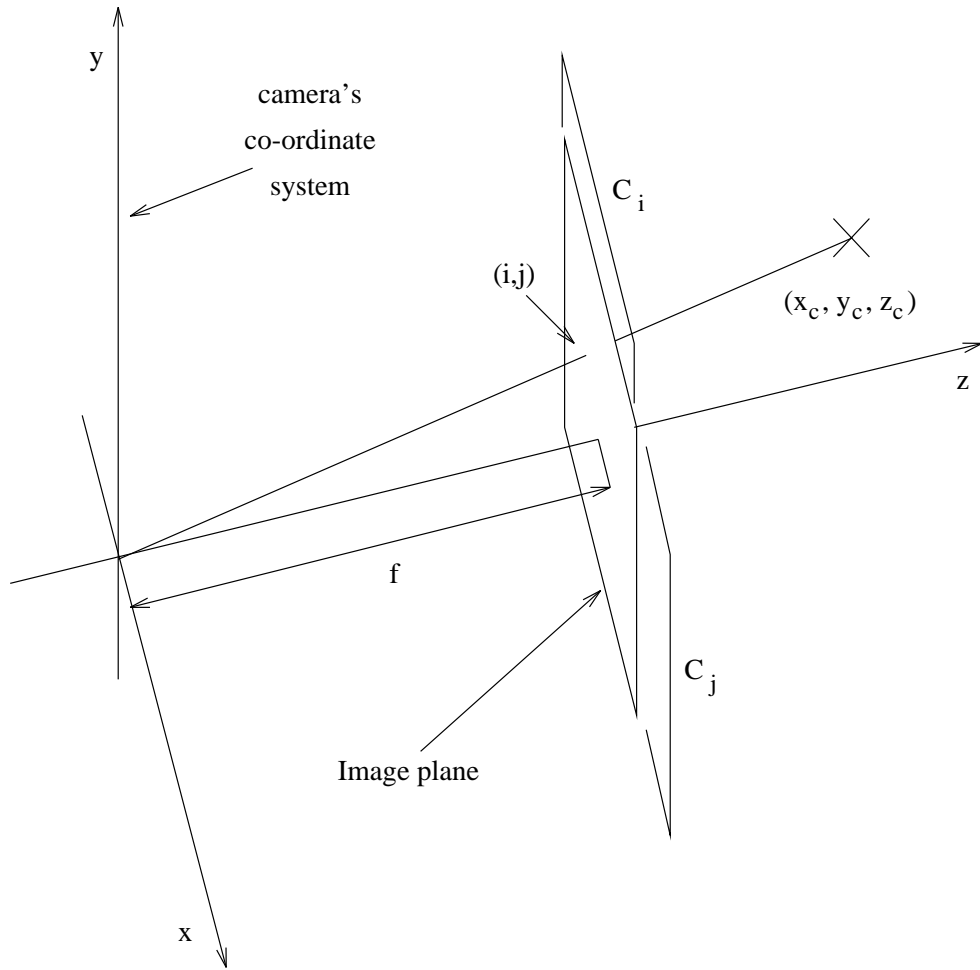


Figure B.3: Projection of a point (x_c, y_c, z_c) in the camera's coordinate system to a point (i, j) in the image, where, f is the focal length of the camera, and C_i, C_j are the number of pixels per millimetre on the image sensor in the i and j directions respectively

Appendix C

Inverse Projection of a Point in the Image to a Point in the Real World

Given a point in the image (i, j) , the corresponding point on the road surface in the real world must be found. In fact, as the image contains no depth information, it is only possible to project the point in the image to a line in the real world. Using the notation defined in Figure C.1, the point (i, j) projects to the following line in the real world:

$$\begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{pmatrix} + \lambda \left[\begin{pmatrix} \frac{i}{C_i} \\ \frac{j}{C_j} \\ f \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha_{cam} & -\sin \alpha_{cam} \\ 0 & \sin \alpha_{cam} & \cos \alpha_{cam} \end{pmatrix} \right] \quad (C.1)$$

To find the corresponding point on the road surface $(x, 0, z)$, this line is intersected with the plane $y = 0$:

$$\begin{pmatrix} x \\ 0 \\ z \end{pmatrix} = \begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{pmatrix} + \lambda \left[\begin{pmatrix} \frac{i}{C_i} \\ \frac{j}{C_j} \\ f \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha_{cam} & -\sin \alpha_{cam} \\ 0 & \sin \alpha_{cam} & \cos \alpha_{cam} \end{pmatrix} \right] \quad (C.2)$$

Solving simultaneously gives:

$$x = x_{cam} - \frac{y_{cam} \frac{j}{C_i}}{\frac{j}{C_j} \cos \alpha_{cam} - f \sin \alpha_{cam}} \quad (\text{C.3})$$

$$z = z_{cam} - \frac{y_{cam} (\frac{j}{C_j} \sin \alpha_{cam} + f \cos \alpha_{cam})}{\frac{j}{C_j} \cos \alpha_{cam} - f \sin \alpha_{cam}} \quad (\text{C.4})$$

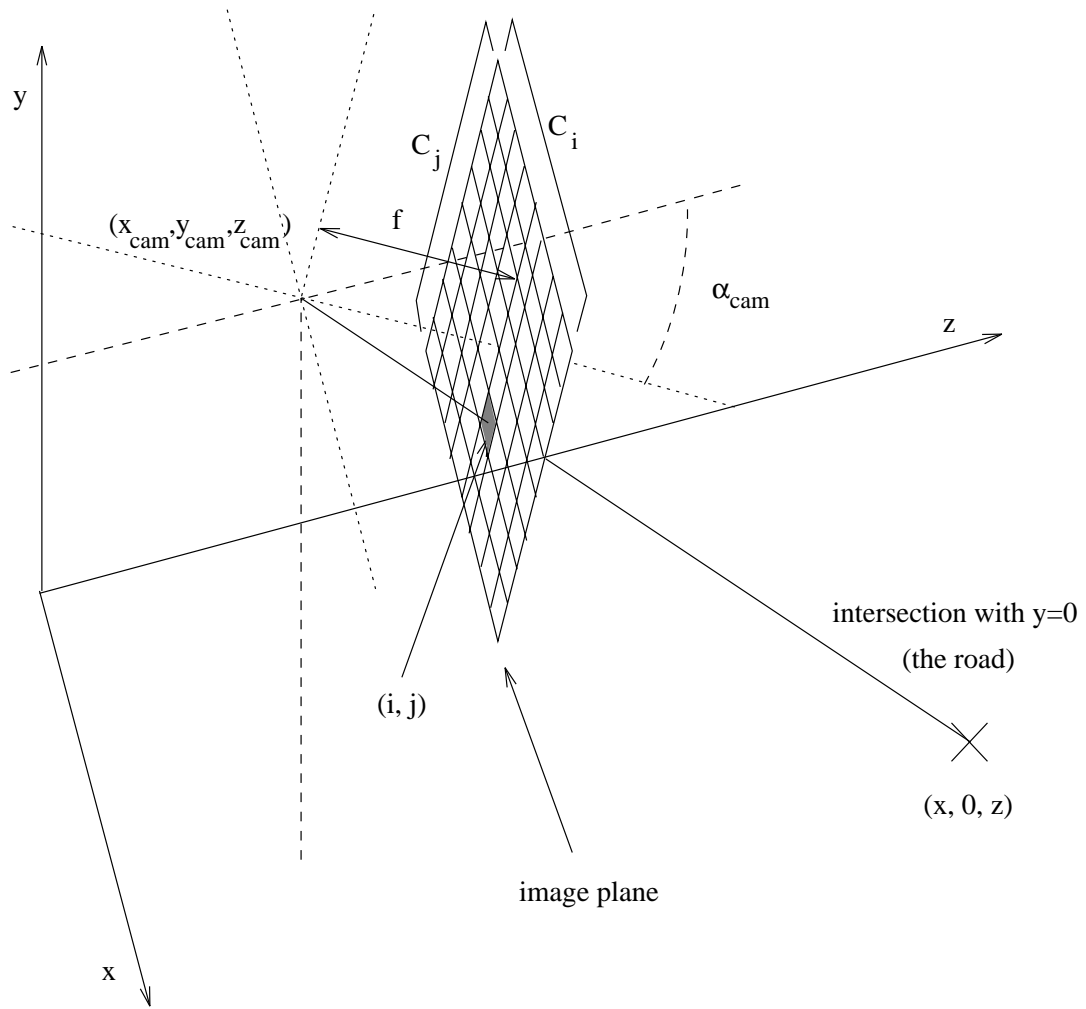


Figure C.1: Inverse projection of a point in the image to a line in the real world, where, $(x_{cam}, y_{cam}, z_{cam})$ is the camera position, α_{cam} is the incline of the camera from horizontal, C_i, C_j are the number of pixels per millimetre on the image sensor in the i and j directions respectively, f is the focal length of the camera, and (i, j) is the point in the image to be projected to a point on the road

Appendix D

Optimisation

Optimisation is the task of locating the *optimum* (maximum or minimum) of a function of one or more variables. Typically, a function will have more than one *local optimum*. A local optimum is the optimum within a local neighbourhood as oppose to the *global optimum* which is the highest or lowest value that the function may ever take on. Location of the global optimum can never be guaranteed unless an exhaustive search of the function's parameter space is undertaken or the nature of the function is well understood, i.e. a parabolic surface in two dimensions only has a single optimum which must therefore be global.

Optimisation methods may be placed in one of two categories; those which optimise functions of one variable and those which optimise functions of more than one variable. The former are referred to as one dimensional and the latter as multi-dimensional.

D.1 One Dimensional Optimisation

D.1.1 Golden Section Search

To find the minimum of a one dimensional function this method requires three points, a, b, c , such that $a < b < c$ and $f(b) < f(a)$, $f(b) < f(c)$. The points a, b, c are then said to *bracket*

the minimum (Figure D.1). The following, iterative algorithm is then applied:

1. Choose a new point, x , either between a and b or between b and c . Suppose in this case x is between b and c .
2. If $f(x) < f(b)$ then the new bracketing points are b, x, c . Alternatively, if $f(b) < f(x)$ then the new bracketing points are a, b, x .
3. Repeat steps 1 and 2 until $c - a$ has reduced to a sufficiently small interval.

The way to choose a new point x , given a, b, c is simply to place x a fractional distance of 0.38197 into the larger of the two intervals (a, b or b, c), measuring from b . This particular fraction gives rise to the name *golden section search* and its derivation may be found in [70].

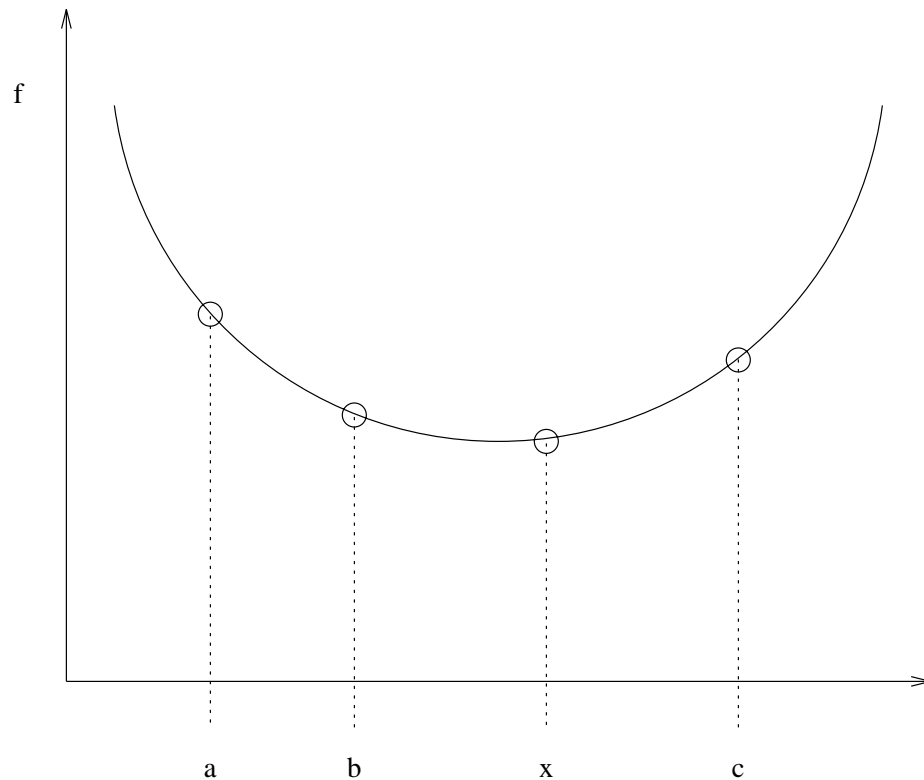


Figure D.1: Bracketing of a minimum for golden section search

D.1.2 Brent's Method

The golden section search makes no assumptions about the shape of the function. However, many functions, especially as their minimum is approached, begin to look parabolic. The three points bracketing the minimum can be fitted to a parabola, the minimum of which will be close to the minimum of our function and is therefore a good choice for the new bracketing point. Given three points, a, b, c , and the function values at these points, $f(a), f(b), f(c)$, the abscissa, x , of the minimum of the parabola fitted through the points is:

$$x = b - \frac{1}{2} \frac{(b-a)^2(f(b)-f(c)) - (b-c)^2(f(b)-f(a))}{(b-a)(f(b)-f(c)) - (b-c)(f(b)-f(a))} \quad (\text{D.1})$$

If our function is sufficiently well behaved, parabolic interpolation will locate its minimum far more rapidly than the golden section search.

Brent's algorithm initially uses the slow but sure method of the golden section search and switches to parabolic interpolation when the bracketed part of the function becomes sufficiently well behaved.

D.2 Multi-Dimensional Optimisation

D.2.1 Downhill Simplex

The *downhill simplex* is suitable for optimising multi-dimensional functions. It will locate the nearest local minimum from a particular starting point.

The simplex is a geometrical object having $N + 1$ vertices in an N -dimensional space. So, in 2D space, the simplex is a triangle. The simplex requires a starting position, consisting of $N + 1$ points somewhere within the multi-dimensional space, from where to begin it's trek downhill. The simplex may undergo a number of transformation, namely *reflection*, *reflection and expansion*, *contraction* and *multiple contraction*. These transformation are illustrated in Figure D.2. The simplex will usually undergo reflections, where the vertex with the highest function value is reflected across the opposite face to a lower value. Repeatedly performing such reflections will cause the simplex to topple downhill, finally arriving at

the nearest local minimum. If possible, the simplex will expand so as to take larger steps. When necessary, it will contract in order to pass through a small hole or along a narrow valley.

D.2.2 Powell's Direction Set Method

Given an N -dimensional function and a vector \mathbf{n} , any of the one dimensional minimisation methods from the previous section may be used to minimise the function along the direction of \mathbf{n} . If a set of vectors are defined, $\mathbf{n}_0, \mathbf{n}_1, \dots, \mathbf{n}_N$, which span the function's space, then repeated minimisation along each vector's direction will eventually locate the function's minimum. However, if a set of vectors can be found such that minimisation along one will not spoil minimisation along the others, then only N one dimensional minimisations will be required to find the function's minimum. Such a set is known as a set of *conjugate directions*.

Powell's method attempts to find such a set of conjugate directions while at the same time trying to align the directions along any long, narrow valleys which may be evident in the function.

D.2.3 Simulated Annealing

Simulated annealing may be used to perform either combinatorial or continuous optimisation of a multi-dimensional function. It's name and operation are based on an analogy with nature. Consider a quantity of molten metal. At high temperatures, the molecules within the metal are able to move around freely but as the temperature reduces so does the mobility of the atoms. If the metal is cooled slowly, the atoms arrange themselves into a crystal lattice which extends across billions of atoms. Such a crystalline state represents the minimum energy state for the system of atoms. It is important to note that *slow* cooling is essential. If the metal is cooled quickly, perhaps by quenching, the minimum energy, crystalline state will not be achieved.

Applying these ideas to numerical optimisation requires four items:

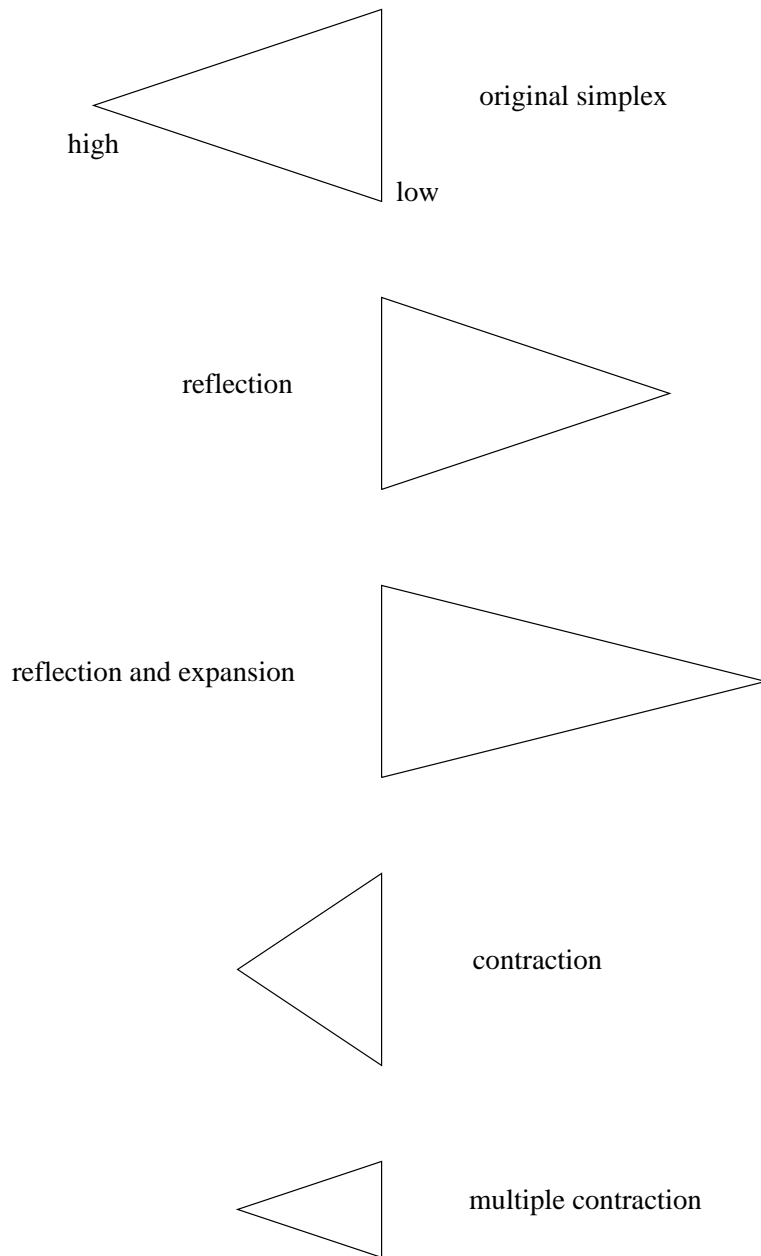


Figure D.2: Transformations undergone by the downhill simplex

1. A representation of the system's state.
2. A method for generating random changes to the system state.
3. The function to be minimised. This represents the *energy* of the system and is dependent on the system state.
4. A parameter, T , which represents the temperature of the system and an *annealing schedule* which governs the rate at which T is reduced.

Representation of the system state is entirely problem specific. For a combinatorial problem it will merely be the current combination, whilst for a continuous problem it might be a point in multi-dimensional space.

Random changes to the system state can be produced by *shuffling* parts of the state for combinatorial problems or simply adding a random vector (which may be biased downhill) in the continuous case. Each proposed random change is inspected and if it results in a reduction in energy it is accepted and becomes the new system state. However, if it results in an increase in energy then the probability of acceptance, p , is $p = \exp^{(-E_2 - E_1)/kT}$, where, E_1 is the energy of the current state, E_2 is the energy of the proposed state, k is *Boltzmann's* constant which relates energy to temperature, and T is the temperature of the system. This equation has its roots in the theory of thermodynamics. The policy of always accepting a downhill move and sometimes accepting an uphill one is known as the *Metropolis* algorithm.

The energy function to be minimised and the system temperature, T , are self explanatory. However, the annealing schedule is the tricky bit. Deciding on a particular annealing schedule is the *black art* of simulated annealing. If T is reduced too rapidly the global minimum will not be found and the system will become stuck in a local minimum. Decreasing T more slowly necessitates more iterations of the algorithm and so takes longer. This trade-off must be resolved experimentally.

D.2.4 Genetic Algorithms

Genetic algorithms are based on Charles Darwin's theory of *survival of the fittest*. Any

member of a species who is in some way better than the others will have a higher chance of surviving and therefore mating and so passing on his genes to future generations. In this manner, the species will gradually advance from one generation to the next.

Genetic algorithms are suitable for optimising both combinatorial and continuous problems. Three things are required in order to formulate a problem as a genetic algorithm:

1. A representation of the problem as a number of *genes* forming a *chromosome*.
2. A method for mating two chromosomes in order to produce children.
3. A score function which returns the fitness of a particular chromosome.

Consider a six dimensional function for which the maximum must be found. Each parameter of the function may be coded as a binary number. The six binary numbers may then be concatenated or spliced together to form a chromosome representing a particular point in the search space. Each binary digit within the chromosome is known as a gene.

A population of randomly generated chromosomes is then created and reproduction between chromosomes begins. Two chromosomes are randomly selected, where the probability of a particular chromosome being chosen is proportional to its fitness. This means that the fitter chromosomes are more able to spread their genes. Each chromosome is split in two at the same, randomly chosen *crossover* point. The first part of the first chromosome is then joined to the second part of the second chromosome and vice versa thus generating two children.

In order to maintain diversity within the population, mutation occasionally occurs during reproduction and one or more of the childrens' genes are randomly altered.

After each generation the fitness of the population will improve and hopefully, after many generations, the maximum of the fitness function will be found.