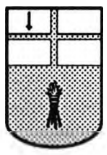
Transputer Configurations for Computer Vision

by

Majid Mirmehdi
Department of Computer Science
City University

September 1991



This thesis is submitted as part of the requirements for a Ph.D. in Computer Science, in the Department of Computer Science of City University, London, England.

تقدیم به پدر و مادر بسیار عزیزم

To my dear parents for providing me with the opportunity.

Contents

List of Figures

List of Plates

List of Tables

Acknowledgements

This thesis is the culmination of my work in a Part-Time PhD course. I would like to express my sincere appreciation to Dr. Geoff Dowling for all his help throughout, and to Dr. Tim Ellis for his advice upon my continuous interruptions of his work over the recent months. I'd like to thank both again for correcting my many solecisms.

I wish to also thank Dr. Geoff West, Beatrice Brillault, and other members of the Machine Vision Group at the City University for all their input. Mr. John Snell must be thanked too for his help towards, and correction of, my vague knowledge of mathematics.

I would like to thank John Hammond for originally setting me on course, and my friend Dr. Farshid Kamali for his encouraging words throughout. I'd also like to acknowledge the support and friendship of Jonathan Prothero, Matthew Pearse, Lynne Taylor, Bahman Motlaq, Hugh Mitchell, my dear brothers Mohsen and Masoud, and my dear sisters Fakhri and Azam. My appreciation also goes to Sybilla de Uray-Ura for her understanding.

Declaration

ABSTRACT

Computer analysis of static images imposes a significant computational burden on the processing hardware. In dynamic vision, the problem is manifold, and the requirement is also to reduce the latency of the processing, in order to allow realistic reaction times to events in the scene. Flexible, massively parallel architectures hold the promise of fulfilling these requirements for low, medium and high level vision tasks, provided that robust algorithms can be implemented in an efficient manner.

In this thesis the role of the transputer as an intelligent processing element for multi-processor parallel architectures will be examined to determine its suitability across the spectrum of vision processing levels. To explore such possibilities, two fields within computer vision will be investigated.

Initially, low and medium level vision tasks will be explored to apply the transputer to the field of label inspection. This investigation will include the introduction and analysis of a data-routing mechanism which will then be compared with one already popular in the world of transputers. Both techniques are suitable for geometric parallelism. In the course of these inquiries, a highly parallelised approach to the solution of the normal parameterisation of the Hough transform will be presented.

Next, to investigate the more demanding aspects of computer vision, bordering around medium to high level vision, the ideas of temporal continuity and motion correspondence of image features in time-varying sequence of images will be examined. A parallel model is described which is designed for use as a basis for implementation of image-feature tracking algorithms on general parallel architectures. The model is independent of feature tracking algorithms. An implementation of the model is outlined using a tracking algorithm founded on features such as the mid-point, orientation and the length of edge segments, and using a modified form of the Kalman filter. The implementation consists of three independent units each of which has been applied in a studied transputer configuration. For example, the tracking unit is based on a tree configuration and displays MIMD characteristics. The edge extraction unit borrows from earlier work in the thesis and further investigates that approach.

Overall, this thesis spans the fields of image processing and parallel processing in the investigation for the applicability of the transputer.

Chapter 1

Introduction

1.1 Background

The original publicity and the subsequent arrival of the INMOS transputer helped mark the 1980's as the decade of parallel processing. Parallel processing may have its roots in the very early days of computing, but it has only been in the last few years, and partly because of the transputer, that it has begun to enjoy popular, widespread recognition and use as a feasible approach to tackling sequential computation. The availability of the INMOS T414 and T800 processors has meant that the averagely funded research department can afford to practice the theoretical concepts of parallel processing in real multi-processor hardware platforms rather than in simulation on an already heavily used, shared, single processor multi-tasking, departmental machine. The transputer's effectiveness and popularity is further corroborated by the impressive ease in which it may be used as a basic building block in constructing different multi-processor configurations.

The building of parallel multi-processor configurations is itself today a requisite in many different fields where the need for more powerful processing is becoming ever more vital as the gap between increasingly complex algorithms and hardware constraints widens. Knowledge-based Expert systems, Artificial Intelligence, and Computer Vision are three major fields which could reap benefits from the fruits of parallel processing.

But these are still early days for parallel processing. For example, for a parallel processing computer vision system to have the diversity and capability of the human visual system in scene recognition and understanding, it must come up with a pair of "eyes" with 250 million

receptors and a "brain" with processing cells numbering around 10^{12} (1 million million) with an average of many thousands of interconnections. The closest machine to this is the Connection Machine from Thinking Machines Corporation containing 65536 processors, dealing with a paltry (e.g. 256x256) image resolution, and costing $3 million.

1.2 Objectives of Thesis

A multi-processor system using a common memory and a shared bus is very efficient when a small number of processors are involved. As the numbers increase, shared memory contention arises as bus bandwidth is reduced and overheads increase. In this research work, the use of transputers for building distributed memory multi-processor configurations are investigated with application to two areas in computer vision,

- low-level and medium-level image processing is examined using SIMD parallel processing on linear chain and array/mesh transputer configurations, with operations such as edge detection and edge segmentation parallelised using specially developed task division techniques and the increasingly popular farming technique. All these are applied collectively but briefly to the field of label inspection,
- the vaguely defined borders of medium-level and high-level image processing are crossed to explore both SIMD and MIMD processing through the stages of edge segmentation, filtering, matching and upkeep of the temporal continuity of scene features, in application to a detailed study of motion correspondence in dynamic scene analysis.

Computer vision is not only concerned with algorithms, but it is also concerned with architectures. Furthermore, it normally requires performance in real-time. Thus, transputer-based parallel processing configurations are evaluated in this thesis with respect to real-time performance in the aforementioned areas of vision.

SIMD processors match well to low-level image processing techniques where the spatial domain may be decomposed for distribution across the available processors. Such matters are investigated with application to transputers and it will be shown how the transputer can serve as a coarse-grain size computer with respect to communication and performance issues. This is a very valid analysis where there is intense com-

petition in the low-level image processing field from single-chip manufactures, and real-time hardware based convolution processors, with one example coming from INMOS itself, the A100 signal processor.

Being a coarse-grain processor, the transputer is a more suitable candidate for MIMD processing where the communicational load is expected to be much less than the computation load. In these circumstances, computation is usually centred on a non-regular pattern or region of interest with some concern in the surrounding or distant regions. However, sometimes algorithms require MIMD processing where the communication load may still be high. Therefore, the really major issue investigated here is whether the transputer can cope with communicational requirements involved in parallelising vision problems across the board. Nevertheless, it must be stated that due to the relatively little work that has been carried out to date on the parallelisation of higher-level vision algorithms, relatively little is also known about the supporting architectures required [DEH89].

Given the resources made to techniques and approaches in computer vision, it is hoped that some of the work will help to further advance the cause in some related vision fields which may benefit from the parallel algorithms, schemes and implementations presented here. For example, a novel approach to the solution of the Hough transform has been implemented which lends itself very favourably to SIMD parallelism. The solution encompasses the introduction and detailed study of a routing mechanism which is compared in performance with the farming approach when implementing the Sobel filter and the parallel sub-image Hough transform. The routing mechanism loses this battle. The work will be applied to the field of label inspection, but will also be of importance to other work described.

Later in the thesis, the problem of correspondence in motion analysis is examined; this is depicted by the middle, shaded box in the 3D scene interpretation cycle depicted in Figure 1.1. A shift of position and velocity can be associated, in each frame sequence, to each of the tokens in an image when a camera is moving relative to a scene. Knowing the physical nature of these tokens, allows the determination of their distance from the camera. By forming groups and structures from these tokens, they can subsequently be employed to aid a unified understanding of a scene and help the navigation of a vehicle about that scene. To achieve that elemental correspondence analysis, a scene flow model is constructed consisting of observed edge tokens, and edge tokens whose spatial positions are optimally estimated using Kalman filtering techniques. These are brought together for every new frame to build an

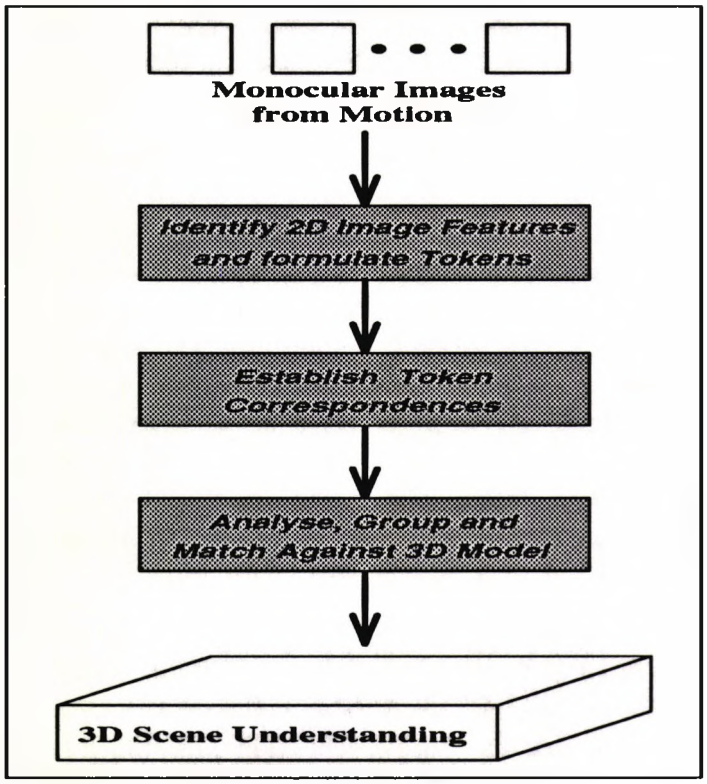


Figure 1.1: An overview of a simplified 3D scene interpretation cycle

up-to-date scene flow model as shown in Figure 1.2.

To establish token correspondence, tokens must be available in the first place (the first shaded box of Figure 1.1). An efficient implementation of this feature extraction stage is attempted in conjunction with, and using, some of the earlier work in SIMD transputing in this thesis. The routing mechanism introduced earlier wins this battle against farm processing. However, this is not really a battle to see who wins and loses, rather it is an investigation of techniques that have been applied for a fair implementation, where suggestions will be provided when there is room for growth and advancement of the approaches introduced or used. All of these will be summarised in the last chapter. In establishing the correspondence between image tokens, a malleable parallel computational model will be introduced as a design basis for a parallel approach, be it distributed memory or shared memory. The parallel model will be independent of particular architectures and token-tracking algorithms. This parallel model will then be used for

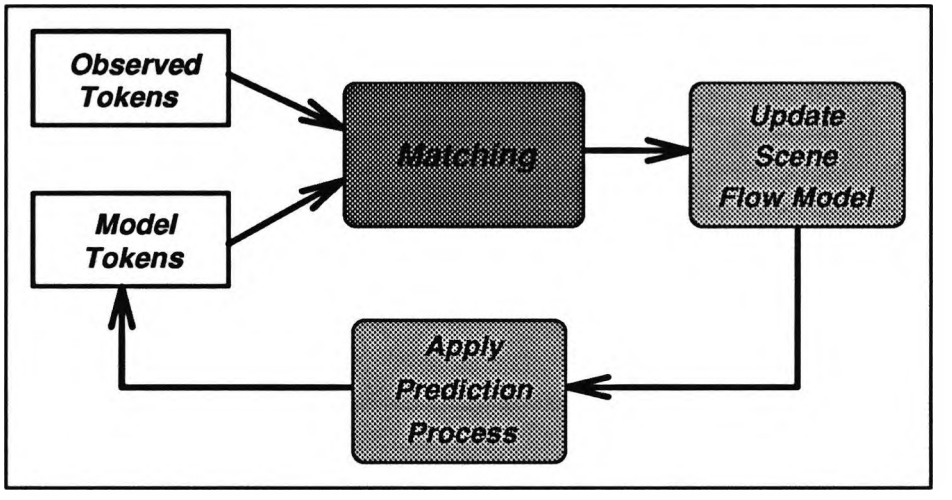


Figure 1.2: Prediction of the state of the flow model

implementing a multi-transputer distributed system, MATCH, which will be studied from many parallel processing points of view, such as processor configuration, load balancing, communications, control, etc. A tree configuration will be used for this implementation. Earlier, the Kalman filtering techniques used in MATCH for filtering and estimating the state of the image tokens will have been derived and discussed.

Overall, this thesis encompasses both parallel processing issues, mainly with respect to the transputer, and many image processing issues which are parallelised, again with respect to implementation on the transputer. The discussion above has been only a brief overview. The aims of the thesis are elaborated further in the next chapter where also some foundations and principles in image processing and parallel processing are laid out. The chapter is followed with four more which together review and discuss the topics mentioned above, and the whole thesis ends with a chapter which assesses the complete work of all the chapters and draws conclusions from them. The conclusions will be at first more immediately related to the actual work presented in the early chapters, followed with overall conclusions on the suitability of the transputer as a powerful tool for computer vision. The possible role of the next generation transputer, the T9000, will also be examined to see what benefits, if any, may be derived from that processor in areas of vision where currently transputers are being investigated, as in the work in this thesis.

1.3 Review of the Chapters

Except for Chapter 2 which is a general review of the foundations necessary for the whole thesis, the work in this thesis is divided such that each chapter is supplemented either by an elementary or extensive review of the associated field depending on the relevance of a topic,

• **Chapter 2** is divided into three primary parts. The first part is further divided into three sections covering those aspects of image processing which are of particular interest in later chapters, such as the Canny edge detection process and the normal parameterisation of the Hough transform. The second part briefly discusses some of the multifarious aspects of the field of parallel processing drawing specifically on examples and architectures for computer vision. The topic of parallel processing is continued into the third part via a compact review of only those aspects of the transputer and OCCAM which are of fundamental importance to this work. This was in preference to a reproduction of available literature on the hardware construction of the transputer and the complete syntax of OCCAM.

Some aims of the thesis are also further elaborated in this chapter.

• **Chapter 3** introduces a communications mechanism for mapping image data across mesh arrays of transputers. The approach is analysed to provide mathematical means of evaluating the distribution and collection costs for differently sized images on differently sized, rectangular transputer arrays. The scheme is then applied and tested within the framework of the communicational requirements of a real-time inspection problem and compared in performance to a more customised and dedicated scheme.

The work in this chapter has been summarised in [Mir90] or [Mir91].

• **Chapter 4** reviews an alternative data distribution and collection mechanism popular in transputer SIMD processing called farm parallelism. The performance of this is compared to the generalised routing mechanism introduced earlier in Chapter 3 by way of introducing and implementing the $\rho\theta sHT$ sub-image Hough transform. The whole process is then also applied to a more sophisticated approach to the problem of label inspection.

The work in this chapter has been summarised in [MWD91].

Please note that the transputer equipment available to the author up to this stage of the work was of a rather mixed nature, and limited in number, as it will become clear when the above chapters are studied. However, more transputers became available prior to the commence-

ment of the work outlined in the next few chapters.

• **Chapter 5** provides both a review of some major features of the field of motion and dynamic scene analysis, and a derivation and step by step study of the correspondence and matching algorithms used for the work later in Chapter 6. These encompass accumulative differencing, Kalman filtering applied to the equations of motion, followed by the α, β filter, and the Mahalanobis distance matching technique.

• **Chapter 6** presents a parallel approach to the problem of motion correspondence by token tracking. Initially, an elementary investigation into the tracking of objects is demonstrated to aid the understanding of general motion issues at hand, and help to prepare for a better and more modular design of a feature tracking system. This work is indirectly used in drawing up a parallel computational model for the establishment of inter-frame correspondence by feature tracking. The model is intended as a guide-line rather than a manual. It is then used for the implementation of a feature tracking system where the features are edge tokens. The mixed SIMD and MIMD system is named MATCH, and consists of three independent sub-units which reflect the general pipeline processing nature of the vision processing cycle. The three sub-units are each associated with a separate transputer configuration, with one unit, the feature extraction engine, drawing heavily from the work presented in Chapter 4. The tracking engine sub-unit implements the filtering and matching algorithms described in Chapter 5. Both sub-units have very modular designs allowing alternative implementations. The full implementation is studied with respect to many image processing and parallel processing issues discussed in earlier chapters.

The work in this chapter has been summarised in [ME91].

• **Chapter 7** provides a final summary of the thesis coupled with conclusions on the particular aspects of the work presented, followed with some more general and overall observations regarding the use of the transputer in the world of vision.

The chapters are followed by an annotated bibliography.

1.4 TIPS: Transputer-based Image Processing System

A breakdown of the hardware and software items used for this work is provided in Appendix A. From very early on in the course of this work, a modular, general, menu-interface system was designed to allow the

integration and connection of the various components used under one controller. This encompasses the following,

- Services provided by the PC, e.g. keyboard, screen, and ports for external devices. These are handled by a server program in C running on the PC,
- User interface and menu system, running in OCCAM on the host transputer and communicating with the PC services when necessary.

The original server program was supplied in basic format as part of the transputer development system [Inm88b]. However, this was extensively expanded by the author to allow the generation of colour graphics, and access to PC ports. The access to the PC ports was necessary for communications with the frame-grabber board (a Data Translation DT2853). Also, since low-level access to the DT2853 is not a standard feature of the board, a set of assembly routines were developed to allow direct input/output of images to and from the board's frame-buffer. This was a non-trivial task, since it involved low-level manipulation of the 80286 descriptor tables to be able to access the memory addresses spanned by the DT2853 board.

The menu-based interface is of extremely modular and adaptable design, and with some easy editing, new menus can be generated very elegantly. A sample menu of the system is shown in Plate 1.1. The system is called **TIPS** (**T**ransputer-based **I**mage **P**rocessing **S**ystem) and was developed and used exclusively for all the work presented in this thesis. A typical early version of TIPS running on a T414 Host with a four-node transputer network is capable of standard image processing operations such as various edge detection operations, smoothing operations, sharpening operations, arithmetic operations, logical operations, histogramming, histogram equalisation, chain-coding and more. Subsequent versions of TIPS have also been endowed with those functions and capabilities discussed by the work in this thesis.

Furthermore, TIPS is independent of the network configurations used since it is a user-interface, and not a communications harness. It will however readily allow the nesting of communications and procedure calls within its structure. This is the means, whereby different configurations and routing mechanisms are investigated in this thesis.

TIPS has also been used a number of times for various under-graduate research projects over the last three years.

Plate 1.1: A typical high-level menu of the TIPS system

Chapter 2

Parallelism and Vision: Some Foundations

2.1 Introduction and Overview

For computer vision to approach a high level of understanding even remotely close to the capabilities of human perception, it must,

- develop better and more efficient algorithms,
- exploit the faster information processing speed of today's most powerful computers.

Many computer vision scientists are engaged in research to ameliorate existing vision algorithms, but the most obvious path lies in a combination of the two principles listed above. Research should be directed towards the creation, refinement and application of vision perception algorithms with respect to parallel processing hardware[1]. This implies any combination of the following,

1. The vision algorithms may be implemented, almost unchanged, on parallel processors, with each processor working on a subset of the data, i.e. *geometric* or *data parallelism* (to be defined later). This is applicable to less elaborate, low-level vision tasks, with cheap, local access to suitably-partitioned data.

[1]Note that use of the words *Research should be directed...* is not intended to imply that the solution to computer vision necessarily lies around the corner by using "parallel vision".

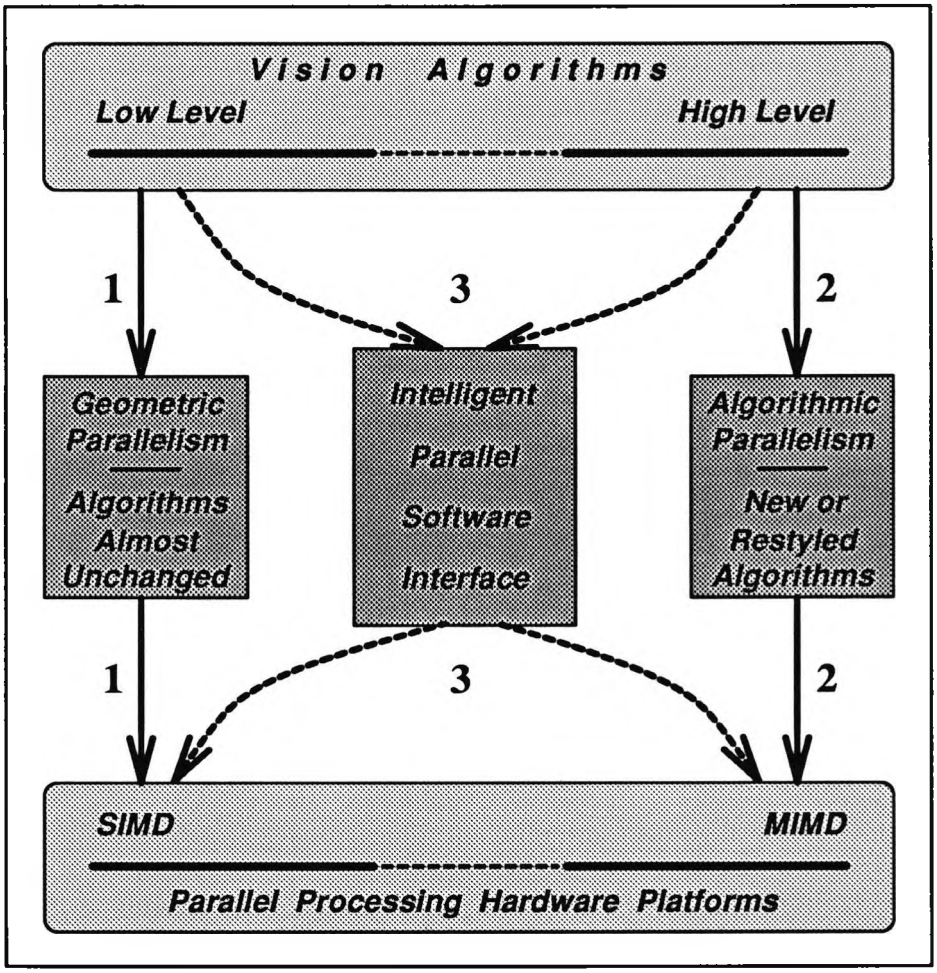


Figure 2.1: Mapping vision algorithms to parallel processing hardware

2. The vision algorithms may first be parallelised and then implemented on parallel processors, with each processor having operational autonomy, and access to any data , e.g. *algorithmic* or *task parallelism*, (also defined later). This is applicable to more complex, high-level vision tasks, and is still a relatively young area of research. Most achievements in this area are rather specific to the proposed problem and not at all general purpose. (The solution of the problem presented in Chapter 6 also falls under this category, despite its concern for generality and flexibility in its own area of application.)

3. All, or at least most, computer vision algorithms may be applied, preferably with little change, on parallel processors, and remain completely transparent to the user. Thus, it would be the task of another programming level, e.g. the operating system, to supervise and exploit any possible parallelism. This would involve automatic mapping of the image data structure and the particular flow of data across the interconnected architecture to achieve the required computation. This would be the ideal case, only it is very difficult to achieve. If implemented efficiently, vision tasks, from low-level to high-level, could be executed without the need for user involvement. But the present naivety regarding the state of high-level vision, and the parallelisation of complex, high-level tasks, automatically eradicates the possibility of having such a general-purpose, intelligent computation system. This is not to mention, the complexities, cost, performance issues, and limits of MIMD programming.

This classification is illustrated with corresponding enumeration in Figure 2.1. The aim of this chapter is to consider and review some of the various areas shown in the diagram, and explain and discuss some of the terminology and some of the points raised within the accompanying text. Furthermore, it is intended to examine if a position within the diagram may be established for the transputer, leading towards the general intention of considering transputer configurations for computer vision. Some conclusions for this will be presented at later stages.

To continue for now in satisfying the aims of this chapter, it has been divided into three distinct parts. The initial part provides a review of some important image processing issues, encompassing principles and techniques which will be of direct relevance to and in support of the work presented later in this thesis. Also, by presenting this review first, the flavour of discussion for the rest of this chapter in Parts 2 and 3 may then be purposely tinged with the topic of image processing and computer vision. In Part 2, a general outline of the techniques and domain of parallel processing will be presented. Finally, a compact outline of some fundamental features of the transputer and its companion-language OCCAM will be discussed in Part 3. In the summary and conclusion section, Figure 2.1 will be returned to, to examine and discuss the three parts of this chapter.

2.2 Part 1: Image Processing for Computer Vision

2.2.1 Background

The field of *image processing* is concerned with the improvement of features in an image. The input would be an image, which following some processing would result in an enhanced output image almost always displaying some salient features. This, in general, completes the processing ready for human interpretation. In *image recognition* or more recently, *computer vision*, the aim is to produce a qualitative and descriptive breakdown of the features in the input image, to aid the analysis and understanding of the scene for autonomous machine perception. Image processing is therefore a pre-requisite stage for computer vision.

Some of the first images ever to undergo image processing techniques were those of digitised newspaper pictures sent by submarine cable between London and New York in 1921. These were coded prior to transmission and decoded by special printing devices at the receiving end. Computer analysis of images became more widespread after the success in correcting the distortion of images sent by a space probe to the Jet Propulsion Laboratory in 1964 [GW87].

Today, the field of image processing and understanding encompasses a plethora of applications, such as medicine, industrial inspection, automatic vehicle guidance, security and surveillance, and TV image coding. Many authors have published the various techniques applied in computer vision for the processing and analysis of images; some of these are books by [Mar80, Nib85, GW87, Sch89, Dav90], and others are numerous conference and journal publications on the subject. The techniques span from low-level pixel-based computation to high-level model-based interpretation through medium-level segmentation and symbolic representation. These will be examined in more detail, exemplified by a breakdown and description of some of the techniques applicable to the work in this thesis. For this, a basic knowledge of the nature of digital images will be assumed. However, as introductory, yet comprehensive texts, much can be gained from [Nib85, GW87], and for more detailed analysis and applications, the reader is referred to [Mar80, GW87, Sch89, Dav90]. Although the topic of image analysis will be outlined separately under the aforementioned levels of complexity, there are techniques that float around the borders of each division, as left undefined in Figure 2.1. Therefore the classification is not always as definitive as sometimes presented.

In general, the following scenario is applied when processing an image. The input image is pre-processed, for example to remove noise, to smooth, or to binarise for a two-tone output image. The resulting image may then be segmented into distinct regions, which are analysed and distinguished. This is sometimes referred to as feature extraction or segmentation. The next stage involves the structural grouping of the features into classes which may then be matched against models with the same properties, thus enabling the identification of the features in the scene. A very simplistic example would be as such: In a road-traffic analysis project, any object segmented as consisting of a rectangular box approximately equal in width and length to pre-determined values, with two circles with centres along the same epipolar line and at pre-determined distances apart from each other, may be regarded as a London Bus viewed from the side.

In a general review of the field, Rosenfeld [Ros88] provides a breakdown of the 2-D image analysis paradigm, and the 3-D scene analysis paradigm (according to Marr [Mar80]).

2.2.2 Low-Level Image Analysis

In low-level image processing schemes, the concern is with the manipulation of images at pixel level to produce an enhanced image, with the image represented as *pixels* held in a 2D array data structure of usually square dimensions, and refered to as the function $F(x, y)$ for $[x, y] \epsilon \{0, ..., n-1\}$ where n usually takes a value formed by a power of 2. The pixels usually take values in the range $\{0, ..., 255\}$, and are referred to as *gray* or *intensity* values. These are internally represented as 8-bit byte values. The techniques employed are referred to as those operating in the *spatial domain.* Some operations performed on pixels are arithmetic and logic operations such as addition or negation, and geometric operations such as translation, scaling and rotation. One popular operation as a pre-processing stage for many image processing applications is *thresholding* as means of transforming a gray-level image, $F(x, y)$, into lower gray-level bands. This is most likely to be a reduction to two-tonal "binary" image, say $B(x, y)$, consisting of black and white pixels only,

$$B(x,y) = \begin{cases} 0 & \text{if } F(x,y) \leq \text{THRESHOLD} \\ 255 & \text{otherwise} \end{cases} \tag{2.1}$$

where THRESHOLD is a pre-determined value. It is the determination

of a suitable THRESHOLD which is of paramount importance for applications. More elaborate and optimal thresholding techniques such as those adaptable to localised area characteristics have been given coverage in many publications, amongst them, [Nib85, GW87, Sch89].

Next, operations combining a pixel and its neighbourhood pixels are used to change or classify the representation of the pixels in the image. These may be for adaptive thresholding, sharpening, blurring, edge detection, histogram equalisation, image restoration, and many other techniques. When in a 2D array structure, each pixel has 8 surrounding neighbours (namely *8-connectivity*), except for the outer border of the array, whose pixel points are usually ignored. Except where stated in this thesis, the operations described will not be applicable to the border pixels. The concept of edge detection will now be concentrated on by describing two relevant techniques.

Edge Detection

Edge enhancement and detection, along with thresholding, is a subclass of general image segmentation techniques. These aid significantly in the determination of features in an image, and later it will be seen how motion can serve as a cue for efficient segmentation. Edge techniques use the discontinuity and similarity of neighbouring gray values to operate. Thus, a sharp transition in intensity between two neighbouring pixels may be regarded as a step change representing an edge. Figure 2.2(a) shows a step edge in 1D.

The magnitude and direction of intensity change at each pixel address (x_i, y_i) in image $F(x, y)$ is obtainable in terms of the directionally oriented spatial derivatives,

$$\nabla F(x,y) = \begin{pmatrix} \nabla_x \\ \\ \nabla_y \end{pmatrix} = \begin{pmatrix} \frac{\partial F(x,y)}{\partial x} \\ \\ \frac{\partial F(x,y)}{\partial y} \end{pmatrix} \approx \begin{pmatrix} \frac{F(x+\Delta x,y)-F(x,y)}{\Delta x} \\ \\ \frac{F(x,y+\Delta y)-F(x,y)}{\Delta y} \end{pmatrix} \qquad (2.2)$$

where $\nabla F(x, y)$ is the gradient vector pointing in the direction of maximum rate of change, with magnitude,

$$| \nabla F(x,y) | = \sqrt{\nabla_x^2 + \nabla_y^2} \qquad (2.3)$$

and with direction measured with respect to the x axis of,

$$\theta = tan^{-1}(\frac{\nabla_y}{\nabla_x}) \tag{2.4}$$

Note, that for Δx and Δy equal to one, then the operators above correspond with operators of the form $(-1\ 1)$ and $(1\ -1)^T$ [Sch89].

These operators are quite sensitive to noise and require high contrast neighbouring edges for efficient detection. Instead, they can be expanded into 3 x 3 masks which provide a higher level of detection at the expense of extra computation. Two popular 3 x 3 edge detection mask are,

$$Sobel \quad \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{2.5}$$

$$Prewitt \quad \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \tag{2.6}$$

For example, the gradient component vectors for the Sobel operator at pixel position (45,7) would be,

$$\nabla_x = (x_{44}, y_8 + 2x_{45}, y_8 + x_{46}, y_8) - (x_{44}, y_6 + 2x_{45}, y_6 + x_{46}, y_6) \tag{2.7}$$

$$\nabla_y = (x_{44}, y_6 + 2x_{44}, y_7 + x_{44}, y_8) - (x_{46}, y_6 + 2x_{46}, y_7 + x_{46}, y_8) \tag{2.8}$$

Lee [Lee83] presented some techniques which help improve the performance of a Sobel operator by eliminating some redundancy at the expense of marginally increased storage capacity. For example, Lee proposes the retainment of the values used for the operations on the neighbours of the current pixel which will also be the neighbours of the next, when the mask window is shifted across the image. (This will be used as a good example for outlining efficient on-chip memory manipulation for the transputer in Part 3). Using these techniques, a tailored version of the Sobel algorithm will be used in the implementation of the label inspection application as presented in Chapter 4.

Another operator which responds to neighbourhood intensity changes is the Laplacian mask. However, this is not usually used on its own

for edge detection due to its high sensitivity to noise. Notice that no separate x and y gradients can be computed,

$$\textit{Laplacian} \quad \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{2.9}$$

The Laplacian is the sum of the second-order spatial derivatives, such that,

$$\nabla^2 F(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.10}$$

It has been used in combined-operation edge detection techniques based on Gaussian smoothing (explained below), such as that proposed by Marr and Hildreth [Mar80]. The Laplacian of the Gaussian ($\nabla^2 G$) of an image results in a set of zero-crossings (a zero-crossing being the spatial location corresponding to the *mid-point* of an intensity change).

Another increasingly popular technique which uses the Gaussian is the Canny [Can86] edge operation. In developing his edge detector, Canny specified three major performance criteria,

1. Good detection: by maximising signal-to-noise ratio, more real edge points and less non-edge points are marked,
2. Good localisation: a selected edge point should be as close as possible to the centre of the real edge,
3. Singular response: only one response per edge point is desired.

The "Mexican Hat" shaped normal distribution is also referred to as the Gaussian distribution. The basic idea for the Canny edge detector is to smooth the image (Figure 2.2(b)) with a 2D Gaussian shaped filter, improving edge connectivity and reducing noise. The 2D Gaussian mask has circular symmetry, so it can be decorrelated and applied as two 1D masks, one in the x direction, and one in the y, thus reducing spatial and computational requirement. The resulting smoothed image can then be searched for local maxima of gradient magnitudes which are subsequently marked as edges. This is conducted by comparing the magnitude of the gradient at each point in the image with those on either side of it in the direction of the original gradient in the centre. This is followed by a hysteresis process, which initially sets an upper and lower threshold limit. Then, any edge points pertaining to

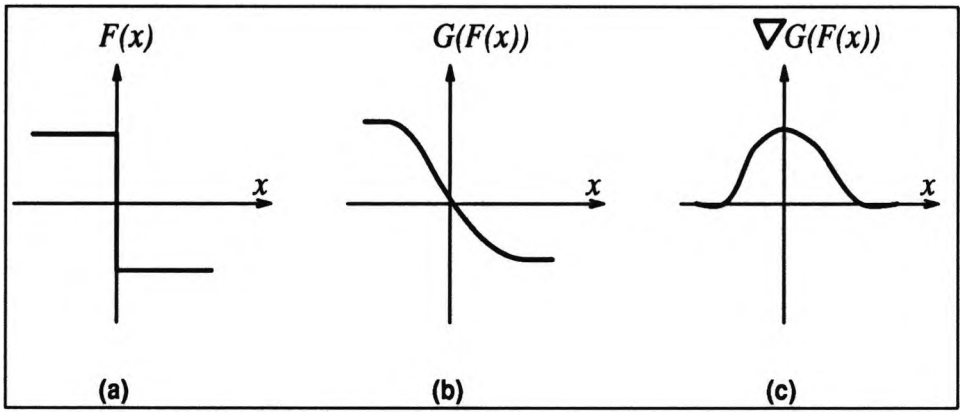


Figure 2.2: (a) A step edge in 1D, (b) After Gaussian smoothing, (c) First derivative of the Gaussian giving the edge maxima

a line segment, with a value above the upper threshold, are immediately accepted, followed with any edges above the lower threshold if they form the remainder of an already established line segment. Thus, broken edge segments may be "grown". The output edge-image is then available.

The value of the standard deviation (σ) for the Gaussian mask determines the size of the mask; this must be selected such that over-smoothing would not lead to the loss of maximum gradients. Davies [Dav87] studies the design and accuracy of optimal Gaussian operators testing for the fidelity of the mask to the Gaussian shape and the isotropy (directionally invariant) nature of the mask. Two optimised masks are then proposed. Also, new techniques for a more efficient application of the Canny edge detector are under way at the Machine Vision Laboratory of the City University with special application to colour images. The Canny edge detector will be used as a more efficient and appropriate edge detection tool than the Sobel operator for establishing spatio-temporal correspondence in Chapter 6, where a performance comparison of the two edge detectors on two different parallel computational models will be presented.

Finally, for other edge detection templates and methods, and the theory of their application and their detection, the reader is referred to more detailed analysis in [Mar80, Sch89, Dav90].

2.2.3 Medium-Level Image Analysis

With medium or intermediate-level image processing techniques, the results are no longer just an output image, and they begin to take different shape and form, where features are segmented and described in usually quite simple representations. Consider edges enhanced through the convolution of an image with the Sobel filter. Except that they exist, nothing is known about them. Their magnitude and gradient could have been recorded along with their spatial position, reducing image-information space requirements. Yet, this would still only be a list of salient pixels. As a continuation of the low-level stage of image processing, there are a number of methods for segmenting and producing feature descriptions with differing complexity and memory requirements. Two of these will be considered now.

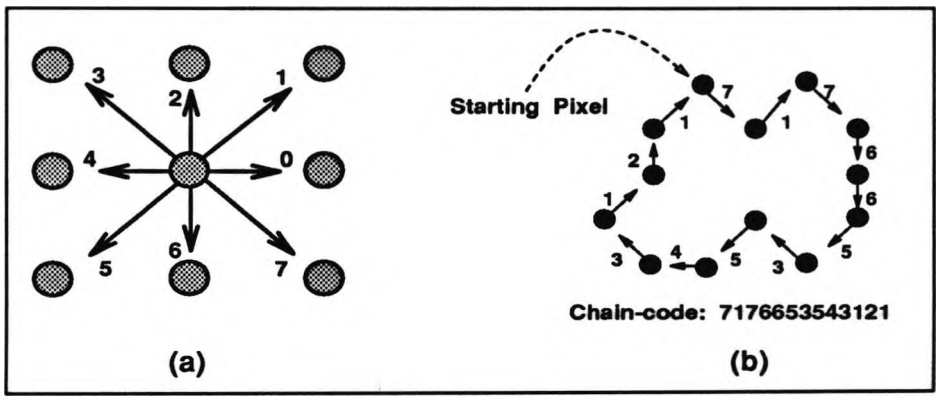


Figure 2.3: (a) 8-connectivity directions chain-code, (b) Chain-code for a closed-curve boundary

Chain-coding [Fre61] is one technique for segmenting edges into open or closed-form curves which in turn will represent edge fragments or distinguishable (areas of image pixels as) objects. Chain-coding, preceded almost always by thresholding, is formed by storing the first pixel address of the start of an edge curve, followed by the position of each subsequent edge point along the edge with respect to the previous point. This is achieved by considering the *4-connectivity* or *8-connectivity* neighbourhood of each pixel, and assigning a direction value for connectivity to each possible neighbour. The chain-code terminates either by the non-existence of further neighbours, or by the ultimate arrival at the starting pixel co-ordinates. These ideas are illustrated in Figure 2.3 by tracing the pixels of the closed-form boundary of an arbitrary object.

Chain-coding can be exploited to yield other information such as the area, perimeter and the centroid of an object. For example, the area A, and the object centroid (x_o, y_o) for an object of C chain points (i.e. the length of the code) are given by,

$$A = \sum_{i=0}^{C-1} \begin{cases} x_i & \text{if } C_i \epsilon \{0,1,7\} \\ -x_i & \text{if } C_i \epsilon \{3,4,5\} \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

$$x_o = \frac{1}{C} \sum_{i=0}^{C-1} x_i \ , \quad y_o = \frac{1}{C} \sum_{i=0}^{C-1} y_i \tag{2.12}$$

More about chain-coding and other representation schemas can be found in [Fre61, GW87]. Chain-coding will be used as an effective object segmentation tool in Section 6.3.

The Hough transform [Hou62] is another technique which may be used for the detection of straight edges and lines in digitised images. Originally, Hough proposed the use of the straight-line formulation,

$$y_i \ = \ mx_i + c \tag{2.13}$$

for transforming all feature points in feature space, (x, y), to the parameter space (m, c), where m and c define the gradient and intercept of the line respectively (Figure 2.4(a)). Since both m and c are unbounded, and can assume a value of infinity, it is better to transform the edge points into sinusoidal curves in the ρ, θ plane defined by,

$$\rho \ = \ x_i cos\theta \ + \ y_i sin\theta \tag{2.14}$$

This is more commonly referred to as the normal parameterisation of a straight line as proposed by Duda and Hart [DH72], and is illustrated in Figure 2.4(c).

Initially, a 2D array space can be assigned to represent the parameter space for ρ, θ. Then, for each edge point, the value of θ is incremented and the corresponding ρ is solved for. This yields a corresponding number of entries in the 2D parameter space accumulator. Thus, for each line in the image there will be a cluster of points that indicate the parameters of the line, as shown in Figure 2.4(b) and (d). (These are idealised diagrams and the non-point sized clusters are there to signify that there is *some* error). The quantisation of the parameter space for ρ, θ is selected depending on the expected accuracy of the edge detection process.

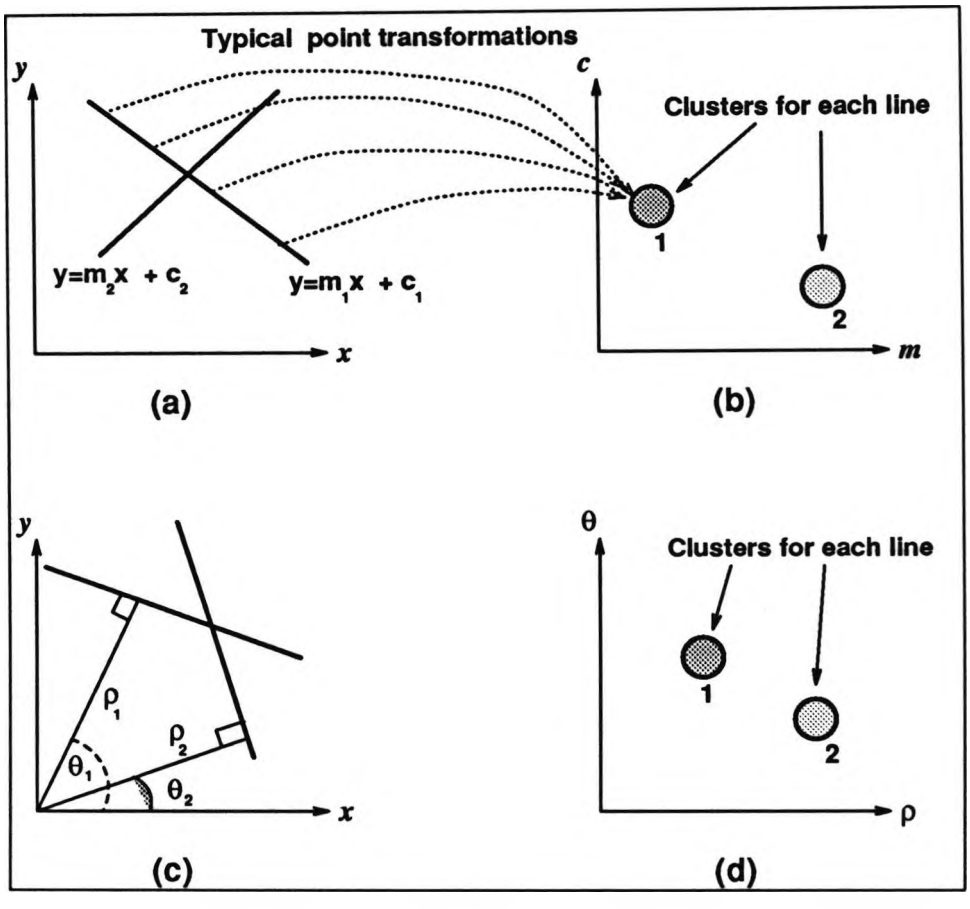


Figure 2.4: (a) m, c formulation of the Hough transform, (b) Clusters for the two lines in m, c space, (c) ρ, θ formulation of the Hough transform, (d) Clusters for the two lines in ρ, θ space

Notice that for edges of non-single pixel width, such as those obtained from Sobel edge detection, unnecessary feature point transformation would result. Thus, a single-width representation of a line segment would be preferred, for example through a thinning operation. Also, it is easy to observe that the general computational load of a Hough transformation is dependent on the number of feature-space points.

The idea of the Hough transform may be expanded to apply to other shapes that can be associated with a parametric equation, such as

$$(x - x_c)^2 + (y - y_c)^2 = c^2 \tag{2.15}$$

for a circle centred on (x_c, y_c) [DH72, BB82, GW87]. This would ne-

cessitate a 3-dimensional parameter space, substantially increasing the computational requirements. Davies [Dav90] presents a comprehensive analysis for an accurate centre location algorithm with reduced computational load. Davies also presents a new formulation of the Hough transform for line segments which eliminates the use of trigonometric functions. This is the foot-of-normal method, which consists of a parameter space congruent with the image space. Ballard and Brown [BB82] generalise the Hough transform for application to arbitrary shapes with no simple analytical form.

The normal and the foot-of-normal parameterisations of the Hough transform will be returned to for further examination in Chapter 4, where an efficient parallel implementation of the (ρ, θ) Hough transform will be presented with application to label inspection. The main idea for parallelisation will be shown to arise through the splitting of the image into smaller sub-images. Other parallel implementations of the Hough transform will also be discussed.

For a general survey of the Hough transform it is worth studying the paper by Illingworth and Kittler [IK88]. This paper discusses recent advances for more efficient application and computation of the transform covering a wide range of topics, from algorithms to hardware implementations.

2.2.4 High-Level Image Analysis

High-level image analysis generally refers to the interpretation of segmented image data, as attained from lower-level stages. Regions of segmented pixels may be *labelled* and *described* by their external characteristics, e.g. their boundary, or by their internal characteristics, i.e. through the pixels comprising the regions [GW87]. Labelled scenes may then be used for matching against model data in order to identify and interpret regions and objects in the scene. Rule-based and predicate logic schemes are examples of techniques used in scene labelling and manipulation [Sch89]. Non-image related knowledge, such as knowledge about world physical constraints influencing imaged entities and their environments may also be used. Some of the techniques employed are multilevel image analysis, 3D modeling and volumetric representations, and image-based knowledge manipulation, including relational graph, and statistical classification schemes.

The area of high-level scene understanding is a wide and varied field with many facets, requiring dedicated analysis and review. For this thesis, the issue of label inspection, to be yet reviewed and discussed,

falls essentially in the domain of low to medium level image processing applications for which edge detection and Hough transformation techniques will be employed. However, the application of image token correspondence described in Chapters 5 and 6, can be used as a cue for high-level image interpretation for understanding motion. The token correspondence scheme provides token descriptions that may be applied to the description of a scene. A higher level system must subsequently use these new tokens, provided for each new frame, for matching against the scene model to formulate an interpretation and complete description of the scene. (The last processing stage of Figure 1.1). The work in this thesis pertaining to the problem of correspondence concentrates on upholding the temporal continuity of token feature sets. Hence, a more detailed review of higher-level scene understanding with particular emphasis on dynamic scene analysis will be presented in Section 5.7, as only a brief outline of some steps to be considered for future use of the correspondence work.

For general coverage of higher-level and 3D analysis of images much may be gained, once again from [BB82, Nib85, GW87, Sch89, Dav90]. These cover topics such as statistical image classification, region and scene labelling, shape from shading, shape from texture, texture, relational descriptors, tree and string grammers (and languages), graph matching (maximal cliques), levels and types of models, and many more.

2.3 Part 2: Parallel Processing (for Computer Vision)

2.3.1 Background

Parallel processing as means of improving computing performance has been firmly established as a major concern across the computing research community for many years. Hockney and Jesshope [HJ88] suggest that this dates back to the days of Charles Babbage's Analytical Engine, when the desire to produce several results at the *same* time, for identical but independent computations, was stated. Many parallel computers have already been offered to the world, and some of these will be mentioned in the forthcoming discussions. Given the high costs normally associated with parallel processing, for the average user the only choice for increased computing performance has usually been in the purchase of a more powerful processor. This is until now, with the impact of VLSI technology advancing the case for parallel technol-

ogy. Chips are becoming faster due to more integrated designs of a system's functional units such as memory, the floating-point or maths co-processors and the communications hardware. Prime examples are the INMOS transputer family of processors, and the 80486 Intel chip. They avoid the delays propagated in accessing devices external to the actual processor.

In today's computing world, mainstream computer usage is becoming more sophisticated moving from the lower level of data and information processing to knowledge processing and finally to intelligence processing [HB84]. For each stage of these progressions, the processing requirements reach unprecedented levels. Certainly the need for faster vision processing via any available technique can hardly need to be emphasised any more than it was mentioned in the introduction to this chapter. For a $P \times Q$ image, a simple edge detector such as a Sobel with a 3x3 kernel, would require millions of operations, given $P = Q = 512$. A Canny operation would necessitate thousands of millions! Thus, the search for more powerful computers has led towards the paths of parallel processing and neural networks. However, for parallel processing, these are early days yet. For example, for a parallel processing computer vision system to have the diversity and capability of the human visual system in scene recognition and understanding, it must come up with a pair of "eyes" with 250 million receptors[2] and a "brain" with processing cells numbering around 10^{12} (1 million million) with an average of many thousands of interconnections. The closest machine to this is the Connection Machine from Thinking Machines Corporation containing 65536 processors and costing $3 million.

Concurrency[3] is achievable both at hardware level and at algorithm level. The related issues will be reviewed next with leaning towards the world of computer vision. Rather than dedicating a section to parallel computers *per se*, the presentation here will cover parallel processing principles, and where known or applicable, will cite machines which have applied the techniques and theories. Initially, attempts at improving uni-processor machines will be outlined. The discussion on parallel processing will start by considering the most common classification scheme for computing architectures. This will be followed by an examination of some of the issues involved and how they relate to the architectures categorised earlier. (Please note that the machine names

[2]Each human eye contains 125 million receptors, called rods and cones, which are nerve cells specialised to emit electrical signals when light hits them [Hub88].

[3]Arguments rage between computer scientists in their attempt to distinguish or claim complete similarity between *concurrency* and *parallelism*. For the purpose of this thesis, they will remain to convey the same general ideas.

mentioned are for general information only, except when as a matter of interest further details are provided. Details of most of the machines may be found in [HB84, Qui87, HJ88].)

2.3.2 Higher Performance in Uni-Processor Systems

Uni-processor machines are based on the von Neumann model which defines a processor and a memory unit connected by a bus. Hwang and Briggs [HB84], Hockney and Jesshope [HJ88], and Quinn [Qui87] provide an excellent outline of various approaches adopted so far for achieving higher performance in uni-processor machines, most of which apply still today with the latest of processors. These may be summarised as,

- multiple functional units, e.g the IBM 360/91 with 2 parallel execution units, one for fixed point and one for floating point operations,
- parallelism and pipelining within the CPU, consisting of parallel adders, carry-lookahead, and instruction pipelining, e.g. Amdahl 470 V/6 where instruction pipelining allows for more than one instruction to be in some stage of execution at the same time, for example during the cycle of fetch, decode, operand fetch, execute, and store,
- balancing of bandwidth between system units, for example by the use of interleaved and cache memory between the CPU and main memory, with the first ever computer to have this being the IBM STRETCH,
- multiprogramming and timesharing, where the system allows for more than one program to be in some state of execution, and the (uni-)processor shares its time between the multitude of user processes,
- plus bit-parallel operations, data pipelining, pipelined functional units, etc.

Pipelined processing is therefore a major technique for achieving higher performance.

Pipelined Computers

The idea of pipelining is to split the tasks involved into a number of independent stages, whereby the output of one stage can be the input of the next. Then, overlapped processing may be achieved while each stage of the pipeline operates on sub-tasks. In a uniform pipeline, each stage of the processing takes similar processing time to execute, given a fair sub-division of tasks. When a fair division is not possible, the slowest stage of the pipeline becomes the bottleneck of the system and processors will remain idle at various stages waiting for communication to and from neighbouring stages.

The ideas of instruction pipelining and functional pipelining in uniprocessor machines have already been mentioned. These ideas have been used in the architecture of most single unit processors, with the processors also being part of larger systems, thus allowing multiple levels of parallelism.

One class of pipelined computers is that of *vector* computers which contain instructions to handle both the data processing and the control sequencing of blocks of data. The CRAY-1 has been described as the most powerful computer of the seventies with supercomputing abilities [HJ88]. The scalar/vector processor part of the machine contains 13 independent pipelined functional units, each performing different tasks such as arithmetic and logical operations.

In multiple processor pipelining (Figure 2.5(a)), cascade of processors handle different partitions of a whole task. Thus, algorithms must be split in an efficient way across processors.

Reeves [Ree84] describes the basic organisation of a pipeline processor for image processing. Each processing stage performs the same operation on every element of data, with a result generated at each clock cycle. An $N \times N$ image would therefore require N^2 clock cycles not including the set-up time. This machine, the Cytocomputer, can also implement near-neighbour operations by means of two shift registers. It has been used for cytology analysis and biomedical image processing, and is suitable for very low-level image processing tasks.

2.3.3 General Structures for Parallel Computers

State-of-the-art parallel processing architectures may be categorised quintessentially into *pipelined computers*, *array processors*, and *multiprocessor systems*. The most popular classification for typical architecture configurations for parallel processing is that of Flynn [Fly66],

whose taxonomy is based on the multiplicity in the instruction and data streams of a computer system. An instruction stream is defined as a sequence of instructions performed by the processor, and the data stream is a sequence of data including input and output results, called for by the instruction stream. Flynn's classes are described as follows.

SISD - Single Instruction stream-Single Data stream: This is typical for most sequential computers. The von Neumann computer model forms the main characteristic of SISD machines such as the CDC 6600, IBM 360/91, Amdahl 470 V/6, and CRAY 1. Some SISD machines may be pipelined and they may have more than one functional unit (which may be pipelined too). The discussion in the previous section applies very much to SISD processors.

SIMD - Single Instruction stream-Multiple Data stream: In this class of architectures, the processors are arranged in an array or mesh format and each processor holds or receives the same set of instructions, but operates on different data sets. There is a control unit, and there may be shared memory or memory only local to each processor. Examples of SIMD machines are the ILLIAC IV, DAP, CLIP7A and the MPP. Variations on the SIMD model are the MSIMD (multiple SIMD), systolic arrays with VLSI processing cells for addition and multiplication, and associative array processors such as the Goodyear STARAN, and the PEPE machines. Systolic arrays are regular arrays of identical finite state machines where each element has a small set of inputs and outputs with simultaneous activity[4]. Associative arrays are built around associative memory which is content addressable, allowing parallel access of memory words, as opposed to conventional RAM which is addressed sequentially. For high processing rates, SIMD machines must be able to maintain a remarkable flow of instructions and data throughout the system. The most popular SIMD format remains the array processor which consists of multiple rows of processors with 4-connectivity to their north, south, east and west processors, and usually wrap-around connections from/to border processors, as shown in Figure 2.5(e). This renders the machine suitable as a parallel processing platform for performing neighbourhood-based image processing algorithms, where regular processing is carried out on highly structured data. However, all of the machines in Figure 2.5 could be programmed for SIMD performance.

MISD - Multiple Instruction stream-Single Data stream: No real embodiment of this class of processors has yet been realised due to its

[4] *Systolic* machines are named after the process of contraction of the heart that rhythmically forces the blood forward.

impractical nature [HB84, Qui87]. It connotes that several instructions are operating on the same data stream simultaneously.

MIMD - Multiple Instruction stream-Multiple Data stream: This specifies a multi-processor system with multiple processing units executing in parallel with multiple instructions on multiple data. Examples of this class of machines may be found in the Connection Machine, the C.MMP (a collection of PDP-11s), CRAY-X MP, and the PC WARP. MIMD machines may be operated via *centralised* or *distributed* control. Memory may be a large common reserve, or distributed localised memory. In general, MIMD machines perform more complex tasks, but need less communications than SIMD machines. Thus, the MIMD architecture can be regarded as a suitable platform for the higher-level end of the image processing spectrum of algorithms. All of the machines in Figure 2.5 could also be programmed for MIMD performance.

Shore [Sho84] classified computer architectures depending on the organisation of the computer's constituent parts. Six schemes were specified ranging from the conventional von Neumann architecture through to associative memory processors which distribute processor logic throughout the memory. In between, word-serial, bit-parallel machines, replicated von Neumann machines, and shared-memory multiple processing element machines are covered. They are mostly sub-divisions of Flynn's SISD and SIMD classes. Hwang and Briggs [HB84] cover other classifications such as Händler's.

Flynn's classification will be adopted in this thesis as it is the most widespread terminology for the specification of parallel processing systems, with the main areas of interest being SIMD and MIMD configurations.

2.3.4 Issues in Multiple Processor Computing

In multiple processor computing some form of arrangement is required for connecting the processors, and co-ordinating their activities. This raises a number of issues which are now discussed briefly; some will be picked up on in more detail in later chapters.

Granularity

The first stage of parallelism is to consider the subdivision of computations into independent tasks. The size of a computation, or grain, is determined by the quantity of communication needed between the different computations. It is preferable to keep this to a minimum.

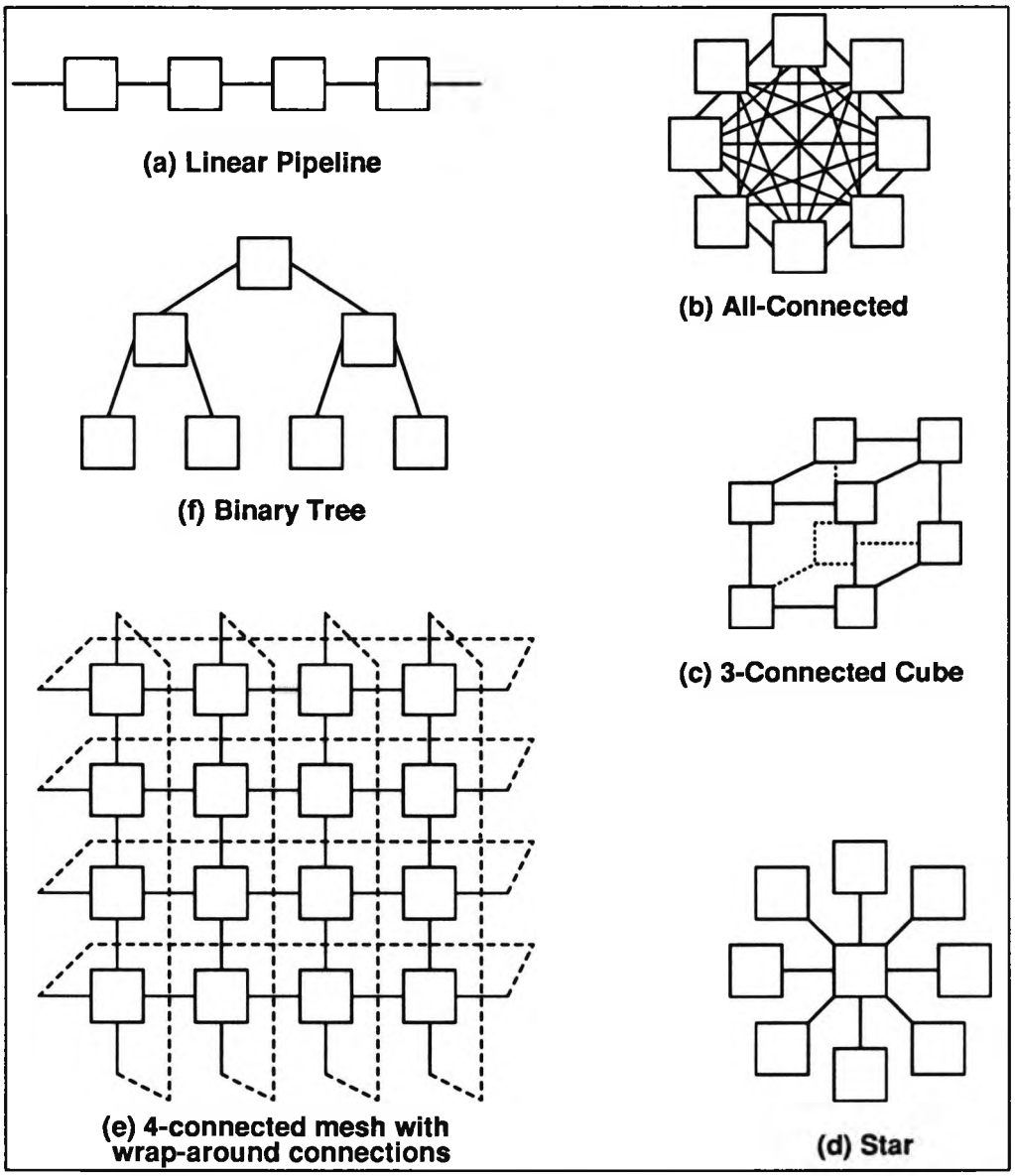


Figure 2.5: Static topologies for pipeline, SIMD, and MIMD computers

In SIMD and MIMD systems the grain size depends on the complexity of the processing nodes. For example, a system with large scale simple processing elements(PEs), such as the CLIP4 image processor [FMM88], is a classic SIMD processor with *fine-grain* parallelism, where each simple PE operates on single-bit data. In contrast, a transputer system, such as the FPS T-Series, can be considered as a *coarse-grain* system, where computation load is expected to be heavier than the communicational requirements.

The performance of all multiple processing systems of all granularities suffers when high rates of communications are involved. Fine-grain machines also have high overheads and administration, and are generally more difficult to formulate software for, which in turn pushes up the costs of parallel processing.

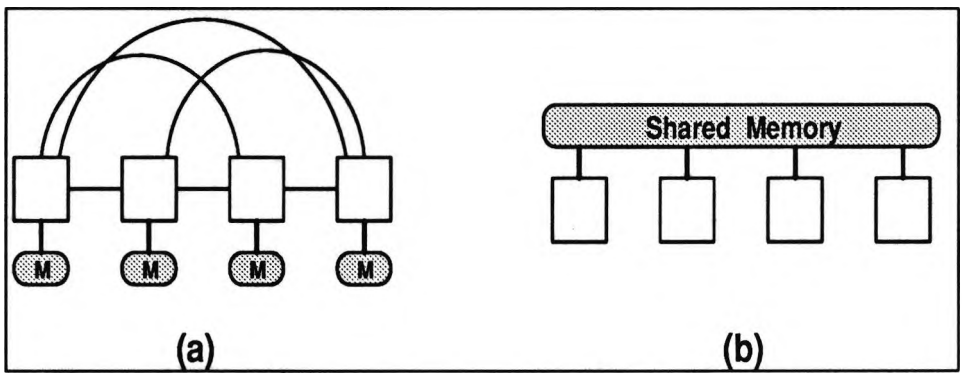


Figure 2.6: (a) (An all-connected) Distributed memory computer, (b) Multi-processor shared-memory computer

Communication and Synchronisation

Parallel processors may be classified into two divisions on the basis of data communications between processors. These are shown in Figure 2.6. In *tightly-coupled* systems, processors share a common main memory, with the whole configuration usually controlled by one operating system. Processors may use a switching mechanism to reach the shared memory. The switching mechanism may take the form of a common bus, a crossbar switch, or a packet-switched network [Qui87]. Examples of tightly-coupled systems are the HEP, C.MMP, and Sequent Balance 8000 multi-processor systems. In *loosely-coupled* systems, rather than a shared memory, each processor has local memory with local control, and communication between processors is necessary

to exchange information. Loosely-coupled systems are also referred to as *distributed* systems, due to the independence and modularity of the individual processors. Communication between distributed processing elements is said to take place via *message-passing.* Example of loosely-coupled systems are the Cm^*, the BBN Butterfly, and the FPS-T series (which is based on transputers).

Two common techniques used in "communicating" in shared-memory systems are *mutual exclusion* and *condition synchronisation. Mutual exclusion* refers to the mutually exclusive execution of a sequence of statements that must appear to be executed as an atomic operation. For example, consider the sum $A := A + B$. If the process operating on the sum did not store the result before another process read the value of A, then the outcome of the entire computation may be erroneous. *Condition synchronisation* refers to the delaying of the execution of a process until some data object it shares with another process is in an appropriate state.

In distributed systems, communications may be *synchronous* or *asynchronous*, with given instances in controlled communications and dynamic communications respectively. Synchronous communication involves the exchange of messages through the mutual agreement of a *sender* and a *receiver* process. Both have to be ready to communicate, and each has to wait if the other is not yet ready. Thus, process synchronisation is implicit in the controlled access to local and global memory by multiple processors through message-passing. Later in Chapter 6, it will be seen how synchronisation is achieved on local processors between multiple local processes. Asynchronous communication, requires that communication can take place at any time between two processes, with the sending process dispatching its message and continuing its work, and the receiving process buffering all received messages and attending to them at a later stage. It is easy to deduce why Ben-Ari [BA90] compares synchronous and asynchronous communication to the telephone system and the postal service respectively. In *send, no wait* asynchronous communications, processing is bounded by the buffer size and the computation load of the destination process, whereas in synchronous communication everything may ground to a halt for either the sender or the receiver, until both are ready to communicate. Otherwise, deadlock will have occurred. An interesting asynchronous message-passing software environment is PISCES [Pra85]. PISCES allows the creation of tasks or processes, and each process is endowed with a queue to which messages are passed via a handler process which in turn will receive all senders' messages. Thus, message passing is always through the handler process. Also, since a task has independent

control on when to pass and when to receive messages, then message passing is entirely asynchronous. PISCES will be encountered again when its use for motion detection will be of interest.

Programming of Parallel Computers

For pipeline computers, the application algorithm is split into N sub-tasks and distributed across the available processors (of which there are N), with each node working on the data set passed to it by its predecessor, which may or may not be a complete image. This is referred to as *task* or *algorithmic parallelism*, and implies that the design and breakdown of the sub-tasks is strongly linked to the number of processors in the pipeline, rendering the portability of the implementation to a longer (or shorter) length pipeline impractical.

In SIMD computers, *data* or *geometric parallelism* partitions the data into N subsets, over the N processors, with each processor performing the whole algorithm on its own data subset. The granularity of the processor nodes will determine the size of the local data partition. This may vary from 1-bit on the DAP, to large array sizes on a SIMD transputer system. *Geometric parallelism* can be further sub-divided into *control-driven* and *demand-driven* data parallelism. These are both other objectives for analysis in this thesis and will be examined later.

In MIMD computers, the situation varies greatly. For fixed implementations, each processor may be booted with a dedicated program which will continue with its own independent data set, however it may be communicated to it. The programmer remains aware of the nature of this communication and has to design his or her tasks accordingly. In general purpose systems, task allocation and load balancing techniques may determine the computation load of each processor, but the task and data allocation of such parallel systems remain to a large extent transparent to the user. These machines are much more likely to be shared-memory systems, where the difficulty of data partitioning and communications need not arise. MIMD computers can effectively be programmed using *data* and *task* parallelism, with the latter being the more applicable. The implementation of MATCH in Chapter 6 is an approach for a problem-dedicated MIMD system.

Major Considerations

The major considerations in acquiring a parallel processing system are *cost*, *performance*, and *reliability*. These issues are orthogonal and the

variation in one will severely affect the others. The main issues pertaining to both SIMD and MIMD systems may be further categorised as,

- *cost*, for cost per unit processor, and for cost in programming,
- *performance*, for throughput and response time,
- *bandwidth*, for a high communication rate per unit of time,
- *partitioning*, for the division of the network into independent subnetworks,
- *accessibility*, for the shortest path connection or route as the most desirable means of inter-processor communication,
- *reconfigurability*, for dynamic reconfiguration as opposed to static configuration,
- *reliability*, for reliable system performance. Two major areas of parallel processing research are *fault diagnosis*, and *fault tolerance* [HB84] for graceful handling of interconnection and processor failures.

Other related factors

There is much that could be covered under the topic of parallel processing! Some issues have been discussed so far, and some others will be discussed throughout the thesis. This will start immediately from Part 3, where some fundamental parallel processing issues with respect to the design and principles of the transputer and the OCCAM programming language will be reviewed, and it will continue throughout Chapters 3, 4, and 6, if not a little in 5. The topics and issues that will be reported on are load balancing and task scheduling, prioritised process scheduling, data routing, deadlock, cross-bar switches, and programming language issues, amongst others. The only parallel language that will be considered is OCCAM. Other languages used for parallel programming are LINDA-C, MODULA 2, Parallel C, and ADA. A comparison of communicating sequential processes in OCCAM and ADA is available in [Mir88].

2.3.5 Performance Measures

It is perpetually desirable to measure, accurately or approximately, the performance of any system. For parallel processing systems this is not always easy.

Hockney and Jesshope [HJ88] introduce two parameters for use in approximate descriptions of the performance of computers. The parameters are more appropriately used in the context of serial and vector computers to describe the performance of computers during a single arithmetic operation on a vector of length n. The parameters are r_∞ and $n_{\frac{1}{2}}$, describing the maximum rate of computation in floating-point operations performed per second, and the vector length at which the performance degrades to half of its maximum. Thus, the faster the machine the higher r_∞. Hockney and Jesshope complement these parameters by others on grain size, communication, and scheduling, for use throughout their book to describe the performance of various machines.

Two general performance parameters usually considered by workers in the field are as follows. The most common is the term *speed-up* which may be defined as the ratio between the time taken to execute an optimised sequential algorithm on a SISD machine, and the execution of the equivalent algorithm as mapped onto a parallel architecture. This conveys a general definition. Thus, given that a SISD processor requires time T_{sisd} to execute an optimised sequential algorithm, and that the time to execute a parallel version of the algorithm on N processors is $T_{||}$, then speed-up is,

$$S = \frac{T_{sisd}}{T_{||}} \tag{2.16}$$

Others define this ratio differently, for example, Quinn [Qui87] defines speed-up as the ratio between a parallel computer executing the fastest serial algorithm and the time taken by the same parallel machine to execute the parallel algorithm using a number of processors.

Another measure is *efficiency* given as a ratio between the speed-up and the number of parallel processors used,

$$E = \frac{S}{N} \tag{2.17}$$

For example, on a pipeline, with the algorithm split across N stages or processors, with time for each stage being $(T_{sisd1}, T_{sisd2}, ..., T_{sisdN})$, the time to process the full algorithm should be,

$$T_{pipe} = MAX(T_{sisd1}, T_{sisd2}, ..., T_{sisdN}) + V \tag{2.18}$$

where V stands for some minimal overhead. Thus the speed-up and efficiency for a pipeline processor is given by,

$$S_{pipe} = \frac{T_{sisd}}{T_{pipe}} \tag{2.19}$$

$$E_{pipe} = \frac{S_{pipe}}{N} \tag{2.20}$$

For all parallel systems the approximate linear speed-up is given by Equation 2.16. However, there are arguments against research towards large scale parallelism based on laws founded on speed-up analysis. Some of these are Grosch's law and Amdahl's law which have been reviewed in [Qui87]. For example, Amdahl's law suggests that a small number of sequential operations can effectively dominate the performance of a parallel machine regardless of the parallel nature of the system and the part of the code that remains parallel. For example, given that λ is a fraction stating the amount of sequential operations from the whole, $0 \leq \lambda \leq 1$, then the maximum speed-up, S_{Amdahl}, achievable by a parallel computer with N processors would be,

$$S_{Amdahl} \leq \frac{1}{\lambda + \frac{1-\lambda}{N}} \tag{2.21}$$

Thus for an algorithm which contains 20% sequential operations, then $MAX(S_{Amdahl}) = 5$, no matter what the value of N may be. As Quinn [Qui87] suggests, there are algorithms with no sequential operations, and therefore Amdahl's law can serve as a way of categorising good candidate algorithms which are suitable for parallelisation.

Hwang and Briggs [HB84] formulate expressions to measure speed-up and efficiency in a SIMD system. Browne and Hodgson [BH89] also present the same for a SIMD transputer array. Speed-up will be used throughout this thesis to indicate performance improvement of the transputer-based systems presented for both the label inspection work and the motion work. Earlier in Chapter 3, the issue will be examined in more detail with respect to SIMD transputer arrays, where previous work on speed-up measurement (i.e. that of Browne and Hodgson) will be reviewed and complemented with new work on data communication strategies and measurements.

2.3.6 Alternative Concepts

A recent parallel processing methodology is the *data-flow* architecture, which embodies a model far different from the von Neumann model, the *control-flow* computer. A data-flow machine enables the execution of an instruction only when its required data is available. Instructions from any part of the program can execute next and therefore there is no program counter. Data flow architectures reflect the graph of the problem where data is passed along the arcs of the graph from one instruction execution point to another. For further analysis of data-flow computers the reader is referred to a detailed description of these architectures in [HB84].

Another major advance in recent years is the development of artificial neural networks, aimed at emulating the way neurons in the human brain are connected. The potential of such networks would be of great value to both fields of artificial intelligence and vision (from low-level to high-level image understanding). Neural networks are based on neural models or connectionist models. In the former, a neuron becomes a "thresholding" unit, which collects signals from its inputs (synapses), and places a signal at the outputs (axons). This leads to massively parallel networks, with special-purpose analog or digital threshold devices as nodes. In the latter, the model is based on self-learning algorithms, thus resulting in nodes which are primitive, programmable processors [Tre88]. Either way, the fine granularity of these networks dictates a high rate of communication and even today's VLSI and WSI technology can still not offer a practical solution. However, simulations of neural networks abound, with hardware platforms such as the Connection Machine [Tre88, HJ88], and the transputer [OHRS90].

2.3.7 Architectures for Vision

The Cytocomputer [Ree84] as a pipelined low-level image processor has already been briefly mentioned. This section will present a compact review of tightly-coupled and loosely-coupled multi-processor architectures which have been applied to, or specifically built for computer vision.

• **DAP** The Distributed Array Processor was first installed at Queen Mary College, London after its development by ICL. Now with AMT Ltd., the DAP510 SIMD computer is the latest version with 1024 processing elements (PEs) all working under the supervision of a master control unit. Each PE is a simple bit-serial processor and may have

access to the rest of the data store by local routing or via an MCU fetch and broadcast scheme [Pag88, HJ88]. The DAP's SIMD architecture is suitable for most low-level image processing problems. Sleigh et. al. [Pag88] report the implementation on DAP of a number of low-level algorithms, such as 3x3 to 9x9 neighbourhood operations, labelling of binary image regions, Marr-Hildreth and zero-crossings operations, and production of image histogramms. They followed their individually successful experiments with an equally successful application for X-ray baggage classification, using thresholding, labelling of distinct binary objects, and finally classifying each object by area size, and rotation and scale invariant moments.

• **CLIP7A** Specifically designed for image processing, this is one of the latest in the CLIP cellular logic image processor family of computers. Whereas the earlier CLIP4 consisted of 96x96x1-bit array of simple PEs, the CLIP7A is proposed as a linear array of 256 elements, with two 16-bit CLIP7 chips at each element, one concerned with data manipulation, the other with local generation of data memory [FMM88]. CLIP7A instructions would be 256x1 vector operations, thus, processing of 256x256 (or larger) images must be emulated transparently to the user, and under the scrutiny of the host. In addition, each processor would have access to 64Kbytes of RAM, and edge storage elements that allow access to all surrounding pixels for 3x3 neighbourhood operations. This is in contrast to the DAP (or the MPP), which can only manage direct access to its 4-connected mesh neighbours.

This machine is proposed [FMM88] as a prototype for introducing a greater degree of autonomy per processor in a SIMD array, and for studying the possibility of performing high-level image processing by implementing rapid data transfer between distant processors. This would be achieved via left or right data exchange from source PE to destination PE across the linear array.

• **The Connection Machine: CM-2** The Connection Machine consists of 65536 bit-serial processors, each with 8K bytes RAM, and 2048 floating-point processors [DEH89]. The machine displays fine-grain parallelism, with each processor capable of accessing data held by any other processor using complex switching networks, hence the name. Comprising of 16 PE chips, each chip contains processors that are connected in nearest-neighbour grid, and due to its n-cube connectivity, the machine can be viewed as a 12-dimensional hypercube with 4x4 single-bit PEs at each node [HJ88]. Used in the main for AI applications, it has also been applied to image synthesis (where the input data is a model to be drawn and the output is a 2D image) applications. However, given

its interconnection map, it surely provides a suitable platform for both SIMD and MIMD vision applications.

Notice that all the machines above have array-like structures, which are most immediately useful for SIMD, low-level, image processing, but are yet programmable as MIMD processors given that irregular PE mapping is possible (but at varying degrees of complexity). However, in massively parallel computers such as the DAP or the CM-2, the difficulty in programming massive numbers of processors to work in coordination, as well the high cost of communication given their grain-size, added to the high cost of execution control, dampens the feasibility of MIMD processing on such large-scale parallel processors. Therefore, due to the relatively complex nature of high-level vision, and general unavailability of MIMD machines, experiences in this area are still very limited. The solution for higher-level computer vision then clearly lies in somewhat more intelligent, unit processors with more local memory, which are capable of both SIMD and MIMD type operations. The IUA and the PC WARP will be reviewed as two such machines. Also, in the summary of this chapter, it will be seen how the transputer, described in Part 3, is envisaged to cater for low-level to high-level vision through SIMD and MIMD architectures.

• **IUA** The Image Understanding Architecture is a multi-layer system of processors with 4096 1-bit processors at the lowest level, 64 16-bit microprocessors at the intermediate level, and a single symbolic processor at the highest level [WRHR91]. Each level of the IUA corresponds with the three levels of abstraction in image analysis as described in Part 1, with the three levels executing segmentation, symbolic representation, and scene interpretation respectively. There is inter-processor communication at each level, but shared-memory between the different levels, giving a tightly-coupled overall system with dual-ported memory. There is also parallel associative communication and control between the low and intermediate levels. In the DARPA Image Understanding Benchmark tests [WRHR91], containing such tasks as the Sobel edge detector, the median filter, the Hough transform, labeling of connected components, and graph matching, the IUA performed its tasks in the order of milliseconds! Weems et. al. [WRHR91] give a detailed analysis and performance evaluation of the IUA and other machines such as the Connection Machine, and the Sequent Symmetry 81, but also emphasise the differences in the number of processors, clock rates, memory configurations, etc. However, the catch is that the IUA is still under construction, and is only available under simulation on a Sequent Symmetry multiprocessor. Unfortunately little more detail of this machine is available, but when it is completed it would serve as a good candidate

for service under group 3 of the classifications in Figure 2.1.

• **The PC WARP** The PC WARP developed at Carnegie Mellon University (CMU) and constructed by General Electric, consists of a systolic linear array of 10 cells with local memory and each cell capable of 10MFlops. The data transfer rate between the cells is 80Mbytes/s. All cells execute the same program, but each performs on different data, and at any given time will be at a different stage of program execution [DEH89]. A classic systolic array is a pipeline, with each cell performing one step of the algorithm.

The PC WARP machine is also a prime example as a candidate for group 3 of the classifications in Figure 2.1, but for different reasons to the IUA. This is due to its *input partitioning* and *output partitioning* schemes. In the former, the image is divided into columns for each cell, such that for a 512x512 image, cell 0 would take columns 0-51, cell 1 would take columns 52-103, etc., and thus each cell has a tenth of the image. This allows neighbourhood operations for low-level image processing. In the latter, the scheme is used for operations requiring access to global image data, and yet can be computed independently. Each cell would have access to the whole image, but would produce output for part of the image only. This is stored locally until all cells have completed processing. At CMU, this approach has been used to implement the Hough transform, chain-coding, graph matching, and connected components labelling algorithms. For example, chain-coding, although quite simple in SISD operation, would create horrendous problems in a SIMD environment with data partitioned across processors. However, in the *output partitioning* mode of PC WARP, the problem would become less complex with the benefit of more arbitrary localised computation.

The next generation machine (iWARP) proposed in [DEH89] is expected to have 72 cells performing at 16MFlops each.

Although no mention of multi-resolution array architectures for image processing will be made in this thesis, the idea of multi-resolution imagery will be examined quite closely at a later stage. As far as the transputer is concerned, perhaps the most well known of the general-purpose, transputer-based, image processing architectures is MARVIN, running the TINA vision software [RPBK90]. This machine will be considered in a little more detail in Chapter 6, with others being mentioned along the way.

2.4 Part 3: The Transputer and OCCAM

The transputer family of processors from INMOS are many in number. The typical transputer is a VLSI device with a processor, local memory, and communication links for connection to other transputers. For example, the T805-25Hz is a 32-bit processor with 33 ns internal cycle time with a peak instruction rate of 30 Mips, a 64-bit floating-point unit (FPU) with peak instruction rate of 4.3 Mflops, 4 Kbytes of on-chip RAM with 1 processor-cycle access time, and bi-directional peak data rate of 2.35 Mbytes/sec per channel on four communication link channels. The CPU and FPU can execute concurrently. It has a micro-coded scheduler for fast context-switching and time-sharing between processes. Transputer board modules typically come with 1 or 2 Mbytes of external RAM. Now programmable in most high level languages, the transputer is most appropriately programmed in OCCAM[5] which was conceived to provide the best mapping of a concurrent system's process architecture on a single or multiple number of processors. The transputer was designed to implement OCCAM's concepts of concurrency and distributed communication.

It was decided for the sake of brevity, that Part 3 would be dedicated to the fundamental issues associated with the transputer, and its programming techniques using OCCAM, since they are the main tools in this thesis. Thus, this section is not about an exposition of the physics of the T414, T800, or T805 transputers, or the syntax of OCCAM. Such material is already in plentiful supply [Inm87, PM87, Inm88a, Inm89]. However, much about the nature of programming transputer configurations in OCCAM will be given coverage throughout this thesis[6].

2.4.1 Building Transputer Networks

Given the four communication links of the transputer, there are a multitude of physical configurations that may be built by the use of processors as standard hardware building blocks. Parallel architectures such as n-cubes, n-linear pipes, stars, arrays, trees, and others (Figure 2.5) can be physically connected via (cumbersome) hardwiring,

[5]The name is derived from the 13th century philosopher/scientist Wilhelm Ockham, (or William of Occam), whose famous OCCAM RAZOR principle states: *Pluralitas non est ponenda sine necessitate* (One must not multiply entities without necessity). This was meant to reflect the original OCCAM design principle of a minimalist approach to avoid unnecessary duplication of language mechanisms.

[6]Please note that when producing sample code, -- represents a comment, and ... represents a *fold* containing OCCAM processes achieving the task described.

or through easy software configuration using the programmable C004 cross-bar switch. This is a 32 way link switch which may be used for static and dynamic configuration of small or large transputer networks to provide the connectivity required by certain applications. Despite the ease of programming, it may be wiser to use hardwiring in critical real-time systems, since the C004 has been observed by Koontz [Koo88] to reduce data transfer rates by up to 25% when tested in connecting T800 processors.

Multi-transputer networks are independent of buses, yet they can be connected to different buses, e.g. as in [RPBK90, Bux91] with the help of some standard hardware expertise. Transputer-based supercomputers include the Meiko Computing Surface, with user facilities for personal-use network configurations and space for large-scale processor scalability depending on requirements. The Meiko machine at the Edinburgh Computing Centre currently contains over 400 transputers, and is a multi-user machine where every user is assigned a number of processors as their own domain. Another supercomputer is the FPS T-Series which is theoretically expandable to $2^{14} = 16384$ nodes, with a 32-node machine capable of 16Mflops at each node. All nodes contain a transputer and two 64-bit WEITEK floating-point chips for multiplication and addition respectively [HJ88].

2.4.2 Communicating Sequential Processes: CSP

Hoare's CSP [Hoa85] was the inspiration for the design of OCCAM. CSP gives a mathematically-based notation for specifying the behaviour of multiple, sequential, processes communicating synchronously. Traces (symbols defining the behaviour of the process), both deterministic and nondeterministic processes (i.e. when processes have or have not a limited range of behaviour, given the influences of their environment), buffered asynchronous communication, deadlock and livelock and many other issues are dealt with in a formal mathematical language to allow the description, design, implementation, and verification of complex computer systems.

2.4.3 Designing OCCAM Programs

OCCAM is based on the CSP model of computation with issues such as parallel process execution, communication, and synchronisation in its very structure, with the same model of concurrency applicable to processes resident on one transputer as on many. This enhances the

prospects for design, portability, and re-assignment on transputer platforms. In comparison with other programming languages, e.g. MODULA 2 or ADA [Mir88] which assume execution of pseudo-independent processes on a shared-memory computer, OCCAM supports a full model of message-passing process concurrency. Data permeates through the system of processes via channels, hiding the details of each process and its data structures from other processes. Thus, each process can be defined in terms of sub-processes similar in structure.

The idea of process design modularity creates a perfect specification mechanism in achieving *geometric* and *algorithmic* parallelism, by mapping data and tasks onto pipeline, SIMD, and MIMD transputer architectures. Various interesting program design issues which will also have an impact on the performance of a system will be discussed next.

2.4.4 Communication Issues

Communications is the essence of distributed parallel programming. Communication channels between processor-local processes are implemented via memory locations, and between processes on different processors via physical, point to point, standard INMOS links. The transputer is capable of initiating communications which are then handled via autonomous link DMA engines, thus freeing the central processor unit to continue processing. Channel communications and processor computation must therefore be always decoupled.

This may be achieved through the use of a simple buffering mechanism such as,

```
PAR       -- Three parallel processes
  ... Receive data from in.channel and buffer in a FIFO in.queue
  ... Process data from top of in.queue, put results on out.queue
  ... Take data from out.queue buffer and place on out.channel
```

Thus computation can always continue, with the buffers smoothing out the flow of data. The code below shows each queue buffer may be implemented as a series of channels created by the use of a replicated PAR statement thus showing how higher level processes may in turn be broken down into a smaller set of processes. For local processors, rather than moving huge data chunks, the data can be stored in global shared memory and only the index to the data passed between channels to reduce the communication bandwidth. Notice that the use of

buffering allows asynchronous communication.

```
[queue.length+1]CHAN OF INT buffer.queue :
PAR
  buffer.queue[0] ! image.data.in
  PAR i = 0 FOR queue.length
    SEQ
      buffer.queue[i] ? image.in.transit
      buffer.queue[i+1] ! image.in.transit
  buffer.queue[queue.length] ? image.data.out
```

Having separated the communication from the computation, it is also helpful to reduce the number of link transfers by communicating larger chunks of data [Atk87]. Each link takes about a microsecond or 20 processor cycles to set up. Once initiated, the transfer is almost autonomous, consuming 4 processor cycles every 4 microseconds. Thus, the piece of code on the right is much more efficient than the code on the left,

```
SEQ i = 0 FOR 256                    |  SEQ i = 0 FOR 256
  SEQ j = 0 FOR 256                  |    image.chan ! image[i]
    image.chan ! image[i][j]         |
```

where each row of the image is transfered (256 link activations) rather than a pixel at a time (65536 link activations). Of course, for larger link transfers the latency increases.

Variant protocols can be used to pass many different data types between processes. During link transfers, the data is broken down into bytes and communicated as such. Each byte must be acknowledged individually. Faster transfers are achieved by overlapping the data and acknowledge messages.

Multiple channel I/O across links, or within a local processor, can be established using *multiplexing*. A multiplexor is a process which collects inputs from a number of channels, and outputs on one. The mirror image of this is a demultiplexor process which receives data on one channel and outputs data on several outgoing channels. Thus, connection can be established between multiple processes on neighbouring processors. The use of identity tags in the communication message ensures correct addressing and communication between source and destination processes in (de)multiplexed communications.

2.4.5 Process Scheduling and Priorities

Processes on the transputer run at two priority levels, *high priority (H)* and *low priority (L)*. For each of these there is a process queue, represented by a linked list in the transputer. The lists contain the workspaces of processes that are ready to execute. New processes are added to the end of the appropriate list, which is also where newly descheduled processes are placed. The transputer always executes an H process if there is one, and only when the H process has to wait for communication, a timer input, or when it simply terminates, can the next H process proceed. Also, only when there are no H processes that can proceed, is an L process allowed to execute. Then, all L processes are time-sliced, unless they are descheduled earlier due to the specific reasons also associated with H processes as given above. They may also be at any time preempted by a ready-to-execute H process.

Two important computation/communication implementation tips are derivable from the explanation above. Firstly, the posit as regards communications, is to run all routing processes at high priority, while the computing-intensive tasks are run at low priority. This ensures the immediate availability of data for computation to go ahead. Secondly, the less the number of processes in the two queues, the less the context-switching overheads, and the quicker the other processes will be attended to. Thus, the right-hand image initialisation task below will execute faster than that on the left.

```
PAR i = 0 FOR 256                  |  SEQ i = 0 FOR 256
  PAR i = 0 FOR 256                |    SEQ j = 0 FOR 256
    image[i][j] := (BYTE 0)        |      image[i][j] := (BYTE 0)
```

A recently developed operating system for the transputer, based on preemptive scheduling, and providing multiple priority levels, is the TRANS-RTXc[VTL90]. This would by-pass the FIFO scheduler scheme where there is no guarantee of the scheduling of a process within a known time-interval. Thus, the TRANS-RTXc introduces the prospect of more efficient real-time systems on the transputer.

2.4.6 Deadlock and Livelock

A common source of *deadlock* in transputer programming occurs when one channel wishes to communicate with another, whilst the other will never reach a stage to be prepared to communicate. This can then propagate throughout the system's set of concurrent processes and cause a

stand-still. Research at Oxford University on deadlock avoidance has proved the possibility of implementing completely deadlock free programs [RD86]. Although the work is based on CSP, it can readily translate into OCCAM.

When concurrent processes communicate with each other continuously without ever communicating with their external world, *livelock* is said to have occurred. Deadlock and livelock can seem identical to the unaware user. One method for detecting them can be the employment of a supervisor or monitor process, running at low priority and completely independently of normal data flow channels, to determine cpu and link usage.

2.4.7 Fair Attendance

The OCCAM language features a powerful *alternation* construct for achieving concurrency and communication between processes. The extent of this property of OCCAM is not always fully understood, and it is the aim of this section to describe the usefulness of this feature through a simple example. The facility has been a major tool in designing the programs written throughout the course of this work.

An ALT process in OCCAM is one which is prepared to communicate with one of a number of processes simultaneously. It will only select one process which is ready to communicate with it. There are two alternation constructs in OCCAM. The symmetric construct provided by the keyword ALT, which guarantees the selection of one of its branches, namely a *guard* from its list of guards, and the asymmetric construct provided by the keyword PRI ALT which gives a guarantee on the selection of one certain guard over the others. The ALT construct is similar to the SELECT statement in ADA, which has ACCEPT statements as alternatives [Mir88].

To illustrate this powerful facility, a simple implementation of the basic functioning of a lift is described. Consider a university with a 5 storey building, plus a basement, where the floors are served by a simple lift which may hold only one person at any one time. The lift will serve its clients on a first-come-first-served basis. It will not accept any further calls for service until it has served its current request. The Vice-Chancellor's office is on the 5th floor, and requests for transport to and from the 5th floor will receive priority service from the lift over any other requests.

If the lift becomes inoperational, then it is assumed that this action

will be concomittant with the depression of the EMERGENCY button. This takes precedence over all lift functions.

The lift services the floors unfairly. It accepts the very first request after it has finished serving the last person. Thus, once the lift is free, the client must be quick on the request button if the lift is to go to that floor next. Notice that this is a very basic implementation and in the real world many extra functions have to be taken into account.

Considering the following definitions,

```
-- Important declarations
[6]CHAN OF BYTE floor  :
CHAN OF BOOL emergency :
BOOL operational :
BYTE service, signal :
```

the code would be as shown in Figure 2.7.

The comments in the figure describe the actions taken once the lift arrives at a floor. These are not shown for all floors.

PRI ALT introduces an element of prioritisation in order of textual encounter. In this example, the emergency channel is always monitored first while the lift remains operational. Next, it is the fifth floor followed by floors 0 to 4. Since the action taken for all floors from the basement to 4th are similar, a replicated ALT may be used.

```
ALT i = 0 FOR 5
  floor[i] ? service.request
```

This example illustrates the *unfairness* of the implementation of the ALT construct, since it always selects from top down. If service is required on all the floors from the basement to 4th, at all times, then the basement request only would *ever* be serviced.

To introduce some fairness into the scheme, a number of approaches could be adopted; two simple possibilities are shown in Figure 2.8.

Notes in [Jon89] provide a detailed analysis of the fair ALT.

All the issues discussed in Part 3 are of extreme importance when designing OCCAM programs. Some of these will be re-visited later in this thesis. INMOS has released a series of technical notes to aid better design of OCCAM programs by transputer users. Some of these,

```
operational := TRUE
WHILE operational
  PRI ALT
    emergency ? signal
      SEQ
        operational := FALSE
    floor[5] ? service.request
      SEQ
        -- Travel to floor 5.
        -- Accept signal to identify destination.
        -- Travel to destination requested.
    ALT
      floor[0] ? service.request
        SEQ
          -- Travel to floor 0, the basement.
          -- Accept signal to identify destination.
          -- Travel to destination requested.
      floor[1] ? service.request
        ...
      floor[2] ? service.request
        ...
      floor[3] ? service.request
        ...
      floor[4] ? service.request
        SEQ
          -- Travel to floor 4.
          -- Accept signal to identify destination.
          -- Travel to destination requested.
```

Figure 2.7: An ALT example for servicing a simple lift

and specially [Atk87] in particular, are essential reading for anyone attempting to produce efficient OCCAM programs requiring maximised performance at low-down code level. Issues such as the manipulation of the compiler's range-checking option, and block moves for vector assignments (which is distinctly useful for shifting image data about the memory), are amongst the many *pourboires* offered.

The issue of more efficient OCCAM programs will be laid to rest after considering one final point, the use of on-chip RAM. The OCCAM programmer can initiate the placement of variables local to a process, on local on-chip memory of the transputer. This will allow a fast memory access, at 1 processor cycle, to the most frequently addressed variables by the CPU, resulting in a much faster execution of the code. For example, when performing the Sobel filter, the neighbourhood gray values

```
— 1 - Rotate the priority of the floors, after each PRI ALT
favourite := 0
WHILE operational
  ... attend to other channels
  PRI ALT i = favourite FOR 5
    VAL shifted IS i MOD 5  :  -  MOD gives remainder
    floor[shifted] ? service.request
      ... processing for floor[shifted]
  favourite := (favourite + 1) \ 5


— 2 - Give lowest priority next time around to floor selected now,
favourite := 0
WHILE operational
  ... attend to other channels
  PRI ALT i = favourite FOR 5
    VAL shifted IS i MOD 5  :  -  MOD gives remainder
    SEQ
      floor[shifted] ? service.request
        ... processing for floor[shifted]
      favourite := (shifted + 1) \ 5
```

Figure 2.8: Two approaches for a fairer servicing of floors.

of a pixel can be read into temporary variables and placed in on-chip RAM. When the operation window is shifted next, those neighbourhood points which are also the neighbours of the next pixel, can be passed to other temporary variables in on-chip RAM. This reduces access to external memory, and speeds Sobel execution by manipulating local memory.

2.5 Summary and Conclusions

The topics discussed in this chapter were intended to serve as a foundation in, and be a general review of, subjects that are of direct importance to the work presented in the rest of this thesis.

In the first part, image processing techniques such as edge detection and edge segmentation were discussed. Emphasis was layed on the levels of complexity in analysing and understanding images. A major tool to aid in symbolic representation of features in digital images is the Hough transform. This was reviewed and will be returned to in Chapter 4.

The discussion in Part 1 was centred on techniques in the *spatial* do-

main; for those in the *frequency* domain, such as the Fourier transform, the reader is referred to [Nib85, GW87, Sch89]. Further image processing techniques and reviews will be presented when the topics of label inspection and motion are presented.

Part 2 reviewed some different aspects of parallel processing, spanning architecture classifications, qualitative issues, programming and performance issues, and parallel vision architectures from different categories. In continuation, the topic extended to Part 3, where the transputer and OCCAM were introduced, and some fundamental aspects like communications, buffering, and more efficient programming issues were examined. OCCAM is chosen as the main programming language in this work not only because it is the only language on the transputer to offer true parallel segmentation of communicating sequential processes, but also because it offers the fastest route to designing parallel programs for the transputer. Its model of concurrency and sequential logic is simple, mathematically consistent, and is built into the logic of producing multi-process networks of processors. Therefore, despite its lack of facilities, such as dynamic memory allocation (which is merely compiler dependent), OCCAM is preferred for use on the transputer as opposed to a selection of other sequential languages which have non-standard and unportable parallel features. Only a little search of the transputer products market is enough to confuse anyone with numerous "Parallel C" compilers.

Now that the case for using OCCAM is laid to rest, the question that has to be asked is, but why use the transputer in the first place? This is simple, and is not an issue of "chip A is better than chip B but chip C has a faster clock cycle than both". Disregarding all advantages and disadvantages of the transputer, most of which will have been examined by the end of this thesis, the transputer remains as the only cheap source of parallel computing, allowing the simple configuration of multi-processor, distributed networks. Whereas most research laboratories can experiment with at least a handful of their own transputers, very few others can probably afford to buy a little time on a general, supercomputer. The cost of programming is therefore the major deciding factor in such matters (Section 2.3.4).

So, what role can the transputer and OCCAM play in the image projected in Figure 2.1? This is examined next.

Certain issues were raised in the introduction to this chapter which need further exploration now in the light of what has been discussed in between. The main focus of reference is Figure 2.1. During the course of this chapter, it was seen that point operations and local neighbourhood

operations in image analysis, can be and have been mapped across simple 1-bit processing elements or higher degree intelligent nodes arranged as regular mesh or array of processors. The arrangement is referred to as SIMD processing. Some machines associated with this approach to low/medium level image processing were examined. The intention of some of the work that follows in this thesis is to show that this can be implemented equally well on a SIMD array of transputer processors. Furthermore, two major approaches in *geometric parallelism* will be employed to aid in the task. This is partly covered in Chapters 3 and 6, and fully projected in Chapter 4.

Global data operations were reviewed as best suitable for MIMD implementation with different processes working on different data. Two fundamentally different machines, yet both capable of general purpose SIMD and MIMD operations were presented. These were the IUA and the PC WARP. In the work in Chapter 6 the application of the transputer as a MIMD tool will be presented. The nature of the system will be partly similar to the IUA, and partly similar to the PC WARP, in that it will have both SIMD and MIMD sub-units, but it will be a system dedicated to tracking motion by correspondence.

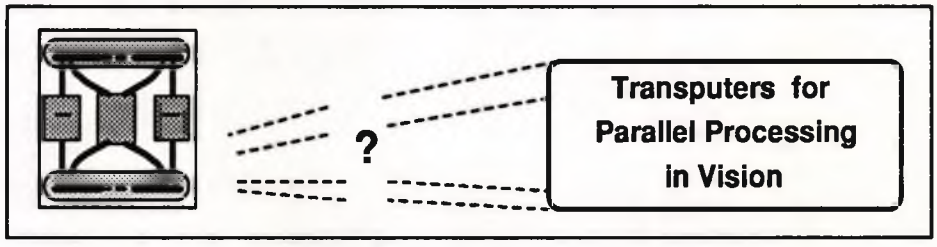


Figure 2.9: SIMD, MIMD, or both?

Finally, it remains to examine route 3 of the enumerations in Figure 2.1. Although this issue will be returned to in Chapter 7, it can be said now that not even a close implementation of such a system stands. Architecturally (i.e. for hardware that can map to the bottom section of Figure 2.1), the resources exist; simply consider fantastic supercomputers such as the IUA (which actually lives in simulation only) and the PC WARP. On the vision side, the algorithms in the range of low to high-level vision prevail to a certain extent, with much that still remains to be developed, and a lot more that are still not suitably developed enough from a parallelisation point of view. The only tools that have brought these together are application-specific interfaces, or simple optimising compilers which are just not intelligent enough to extract more

than some simple SIMD parallelism. Although, no particular solution to this issue can be provided in this thesis, the matter is of interest, and will be re-examined from a transputer/OCCAM perspective in Chapter 7.

In summary, the rest of this thesis is dedicated to the examination of the transputer's role and capability in typical SIMD and MIMD situations. Where will the transputer best fit into in Figure 2.1? Will it be applicable across the board? The idea is illustrated in Figure 2.9. The conclusions for its suitability will be drawn in the final chapter with some brief comments provided in the chapters in between.

Chapter 3

Analysis and Application of a Data-Routing Scheme

3.1 Introduction and Overview

Having considered a SIMD arrangement of hardware as an important platform for low-level image processing, and the necessity of data-partitioning for the implementation of such an idea, these issues will be extended and examined in this chapter with application to transputers.

Geometric parallelism, introduced in the last chapter, may be accomplished via two mechanisms, *control-driven* and *demand-driven*. The demand-driven strategy will be expounded in Chapter 4, whilst the control-driven technique will be closely attended to here through the specification, analysis, and application of a message-routing strategy for an $M \times N$ transputer array network, given a $P \times Q$ image,

- the specification will be the presentation of a controlled data distribution and collection scheme,
- the analysis will encompass the derivation of equations for the approximate projection of the timings required for the distribution and collection of any regular-sized window on the image data,
- the application will be to the field of label inspection, including a short review of the field.

The label inspection application started life as an investigation into proving the possibility of fast, successful image inspection on transputers, which would then grow into a full inspection system, in joint

collaboration with Trivector Systems Ltd., part of the Vinten Group PLC. However, some way through the project, the sponsors withdrew due to financial and take-over problems. Nevertheless, being past the midway post, the evaluation study was conducted to its conclusion. Thus, what will be presented later will be an *ad hoc* approach to a suitably defined label inspection problem, in order to be able to infer the performance of a multi-transputer system for an industrial inspection system.

3.2 Mapping and Communication Costs

The idea of *geometric*, or *data parallelism* was introduced in the last chapter as a method of programming SIMD arrays of processors where each processor performs the same tasks on different data. In low-level image processing the data would preferably be composed of uniform, equal-sized partitions of the image, as depicted in Figure 3.1.

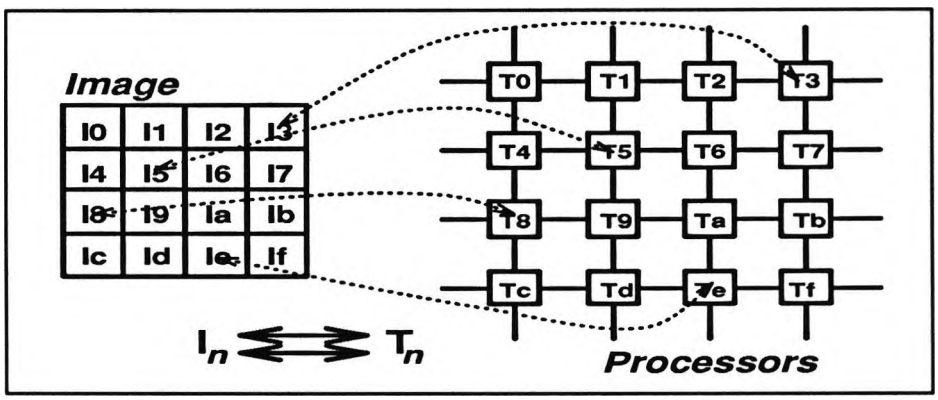


Figure 3.1: Mapping equal-size partitions of an image across a transputer array. Sample mappings are shown with dotted projections.

The granularity of the SIMD processing nodes would determine the granularity of the data. For transputers, in splitting a 512x512 image across a 2x2 array of processors, each of the 4 sub-images would consist of a 256x256 image. Yet, split across an array of 128x128 processors, each sub-image would be 4x4 only. Thus, the data may range across the fine-medium-coarse grain spectrum, depending on the application. Its size is determined in length l, and width w by,

$$l = \frac{P}{M} \ , \qquad w = \frac{Q}{N} \tag{3.1}$$

given that there are $M \times N$ transputers, and the image size is $P \times Q$. Also let it be assumed that the following conditions stand,

$4 \leq P, Q \leq 512, \quad 2 \leq M \leq \frac{P}{2}, \quad 2 \leq N \leq \frac{Q}{2}, \quad$ and $\frac{P}{M}, \frac{Q}{N}$ yield integers.

The process of assignment, distribution, and collection of data takes place under the supervision of a controller processor, and this would usually be a separate transputer to the array. At all other times, the processors exhibit *distributed control*, i.e. they perform their tasks independently of any external supervision, and they only rely on the flow of data as controlled by the supervisor processor. Thus, implicitly, this method of mapping data to processors is referred to as *control-driven*. It may have been noted that inter-processor communication has not been included in the above explanation. This is due to the fact that it will not be necessary, and this is examined now.

Some of the major algorithms in low-level vision need to operate on neighbourhood pixels, and the partitioning shown in Figure 3.1 does not satisfy this requirement at the pixels along the border of each partition. Morrow et. al. [MCK+88] have proposed a method for passing image data which is as follows: the controller communicates with the top-left transputer in the array only, passing the complete image to that processor a *row* at a time. Each transputer in the north row of transputers then retains its own part, and passes the rest of the row to the east until no more data is left. When their own quota is satisfied, each transputer in the north row continues along the same scheme by sending the next segments that it receives partly southwards and partly eastwards until all processors are fed. Following this, an edge swapping operation is initiated, whereby each processor swaps the border information along its four sides with corresponding neighbouring processors. They do not specify their exact handling of the edge transputers, although their predicament is acknowledged. Also no mention of access to pixels from diagonally un-connected neighbours is made. The data is then recalled in a similar but reverse manner. The authors continue by performing the averaging operation, and introducing a more high-level language as a harness for OCCAM in performing image operations.

Clearly, not only is the extra edge-swapping of their distribution phase unnecessary, it is also conducted rather inefficiently at *one pixel* at a time. Assuming that all processors will swap edges exactly in parallel, there will elapse a time while a total of at least $2(l + w)$ extra link set-ups and communications take place before computation can go ahead.

This does not include the time for communicating to diagonal neighbours via other processors. Also, since processes are descheduled at communication points, the context-switching overheads must also be taken into account.

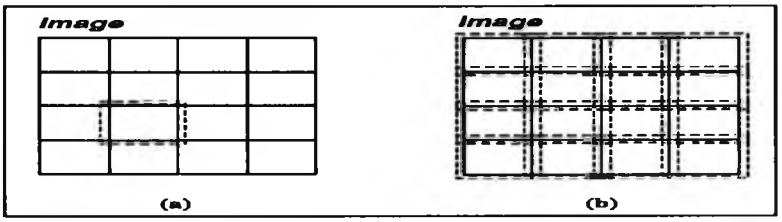


Figure 3.2: For a $P \times Q$ image, (a) shows a sub-image and its extra border area, (b) shows the same for the entire image

A more efficient method is illustrated in Figure 3.2. By including the edge information in the data segments of each original sub-image, the whole edge-swapping phase can be scrapped to reduce the communication stages, with the side-benefit of reducing the overall code size by losing the code for interprocessor control and communication. The segments corresponding to edge transputers may be complemented by data determined in some standard way by the controller and prior to communication. For example, this may be achieved via the use of slightly smaller images, e.g. a 510x510 image allowing borders of width 1. Furthermore, little latency in communications will occur since the increase in row segment sizes routed will be relatively small.

Other efficient routing methods exist (and will be mentioned later), but this work is concerned with the analysis of the above method as a typical message routing strategy. This analysis begins here.

Flatt and Kennedy [FK89] present a timing model for parallel processing, with the initial finding that given a program containing serial and parallel parts, and running on K processors, then the parallel computation time is,

$$T(K) = T_{serial} + \frac{T_{parallel}}{K} + T_o(K) \tag{3.2}$$

with serial execution time of T_{serial}, parallel execution time of $\frac{T_{parallel}}{K}$ if the program may be partitioned into K parallel components of equal running time, and communication and synchronisation overhead of $T_o(K)$. They continue by examining the communication and synchronisation overheads on parallel processors, establishing upper bounds on

the power of parallel processing under ideal conditions (cf. Amdahl's law). They also provide cost, efficiency, and speed-up equations for very generalised measurements. Equation 3.2 is only valid, at least in the case of a SIMD transputer array, if each of the three stages in the equation can be regarded separately. When communications and computations are overlapped, and the transputer is certainly capable of autonomous link transfers while the CPU computes, then $\frac{T_{parallel}}{K}$ and some of $T_o(K)$ will be happening concurrently.

Much more relevant is the work of Browne and Hodgson [BH89] who provide an investigation of the performance of image processing algorithms when executing on data partitioned across transputer arrays. They analyse point, neighbourhood, and global image processing operations, in terms of speed-up and efficiency, by way of simple simulation of communications in terms of memory access cycles, using quoted averages from [Inm89]. In fact, they divide all memory accesses and link communications in terms of a number of different memory reference types of count u, with elapsed time per word of $t_i, i\epsilon\{0, ..., u-1\}$, without differentiating between reading and writing. Then, assuming K processors in total, they use the following general equations for $T(K)$, the time to perform on K transputers, $S(K)$, the speed-up, and $E(K)$, the efficiency, to calculate their estimations,

$$T(K) = \frac{T_p}{K} + \frac{V}{K}\sum_{i=0}^{u-1} p(i)t(i) \tag{3.3}$$

$$S(K) = T(1) - T(K) = T_p(1 - \frac{1}{K}) + Vt_l - \frac{V}{K}\sum_{i=0}^{u-1} p(i)t(i) \tag{3.4}$$

$$E(K) = \frac{T(1)}{KT(K)} = \frac{T_p + Vt_l}{T_p + V\sum p(i)t(i)} \tag{3.5}$$

where, T_p is the total processing time for an image processing algorithm, t_l is the time for a local memory access, i.e. $t(0)$, V is the total volume of data (i.e. by the earlier definition in this section, $V = P \times Q$), and,

$$p(x) = \frac{g(x)}{\int_0^{x_u} g(x)dx} \tag{3.6}$$

where, x is assumed as an address variable that can address the full assumed memory space (including simulated links), and $g(x)$ is the frequency of accessing memory at x, resulting in $p(x)$ as the probability

density of accessing memory at x, with the shape of $g(x)$ determined by the nature of the image processing application.

3.2.1 Evaluating Communication Costs

Given the general difficulty of measuring parallel distributed processors performance, Browne and Hodgson present a very efficient simulation, reporting a maximum difference between estimated and actual performance measurements of $\simeq$ 14%. However, some essential factors that they ignore in their work are, traffic load across the links, which will affect the rate of memory access as defined in their terms, and the whole fundamental issue of image distribution and collection pre and post-processing respectively. No timing for these are assumed or derived, and in fact they assume that the image is already distributed. It is the intention of the work presented here to consider the distribution and collection operation, since it forms an integral and necessary part of the measurement of the total computation time. Thus, Equation 3.3 will be re-stated where T_{dc} is the total communication time for distribution and collection of data, before and after processing, and will be found presently.

$$T(K) = T_{dc} + \frac{T_p}{K} + \frac{V}{K}\sum_{i=0}^{u-1} p(i)t(i) \tag{3.7}$$

with $K = M \times N$. Note that by way of the communications method described earlier, the components of T_{dc} are equal, that is $T_d = T_c$. For the moment, consider T_d only. Initially, let length l, and width w be redefined to encompass the extra border data, b, determined by the image processing operation at hand, (more commonly $b = 2$),

$$l = \frac{P}{M} + b \ , \qquad w = \frac{Q}{N} + b \tag{3.8}$$

Let the total time for distributing a $P \times Q$ image across a $M \times N$ array of transputers, using the row-segment method described above, be,

$$T_d = T_t + T_f \tag{3.9}$$

where T_t is the time it takes the host controller to send the the whole image, row by row, to the top-left (NW) transputer, and T_f is the time it takes for the final package to filter through from the NW transputer to its destination in the bottom-right (SE) transputer in the $M \times N$ array, since that is the furtherest node away. Notice that although T_f

may be significant for large packets in large arrays, nevertheless more usually, $T_t \gg T_f$ and $T_d \simeq T_t$. Therefore, for small transputer arrays almost no marked improvement can be observed if the reverse process to the above were to be implemented. That is, start by sending the data for the SE transputer first, such that T_f will be (almost) non-existent. This will be termed the *reverse-feed distribution* scheme and is the preferred method. For the study here, the first approach, the *forward-feed distribution* will be used as it will help build a simpler picture of what happens.

It is important to notice that computation and communication can be overlapped. However, the concern here is to measure the distribution and collection of data as a separate procedure which can be followed by the separate processing time defined by Browne and Hodgson [BH89]. Also, the overlapping of communications and computation may lead to poorer performance due to the extra code and buffer processes, especially when the ratio of computation to communication is small. Still, the computation on each processor commences once it has its own complement of data, while it may continue to route data through for other processors.

Equation 3.9 describes a time which assumes that during the operation all the other transputers will have received their data packets. This is next shown graphically for a small image on a small array, and can be proved by induction to be true for all cases of M and N.

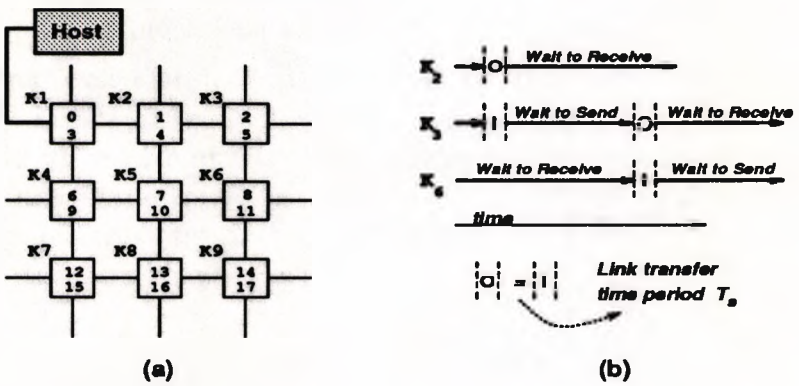


Figure 3.3: (a) Distribution of case data on 3x3 array, (b) Actions of 3 consecutive processors in the data path as the data passes through.

For the sake of clarity, take a simple example including no neighbourhood pixels. Figure 3.3(a) shows the distribution of data on an array

of $K = M \times N$ processors numbered $K1$ to $K9$, and $M = N = 3$, and $P = 6, Q = 3$. Using Equation 3.8 with $b = 0$, then $l = 2$ and $w = 1$. For the general case, there would be lMN packages of data (i.e. row segments) numbered from $\{0, ..., (lMN - 1)\}$, thus for this example these are $\{0, ..., 17\}$, as shown.

Next, define T_a as the period of a link data-transfer which depends on the length of the data-block to be transfered (see Section 2.4), and T_o as the time spent on overheads such as the initiation and set-up of the link transfer. Thus,

$$T_s = T_a + T_o \tag{3.10}$$

giving the total transfer time over a single communication, with $T_a \gg T_o$ and $T_s \simeq T_a$. (T_o is said to take about 1 μsec [Atk87], but is largely dependent on how busy the processor is and can be much greater). Now consider a snapshot of a short period during the distribution stage for any three processors who lie orthogonal to the path of the data flow. Given O for output on any link, and I for corresponding input on receiving link, this snapshot in envisaged for transputers K_2, K_3, and K_6 in Figure 3.3(b) as a data segment passes through, assuming that the period of output for one transputer is equal to the period of input of the transputer it is communicating with. Hence, measurements can be made with respect to the average value of T_s, and thus Figure 3.4 shows the path and number of T_s time-slices necessary for each packet to arrive at its destination.

Note that *Wait to Receive*$\gg$*Wait to send*, with the latter being the period of another link set-up to send the data packet on.

A program was written to simulate this pattern of data flow, and by analysing the results for varying values of M, N, P, and Q, the following formula was deduced to give the exact count, C, of time slices, T_s, of link activity required to distribute any $P \times Q$ image on a mesh-connected array of $M \times N$ transputer, assuming the conditions mentioned earlier are satisfied,

$$C_{lMN} = (M + N - 3) + l(2MN - 1) \tag{3.11}$$

Hence, since $T_d = T_c$, the total data distribution and collection is given by Equation 3.12,

$$T_{dc} = 2T_s C_{lMN} \tag{3.12}$$

The time T_s was measured for the transfer of varying packet sizes. Table 3.1 shows these, along with the measured distribution and collection

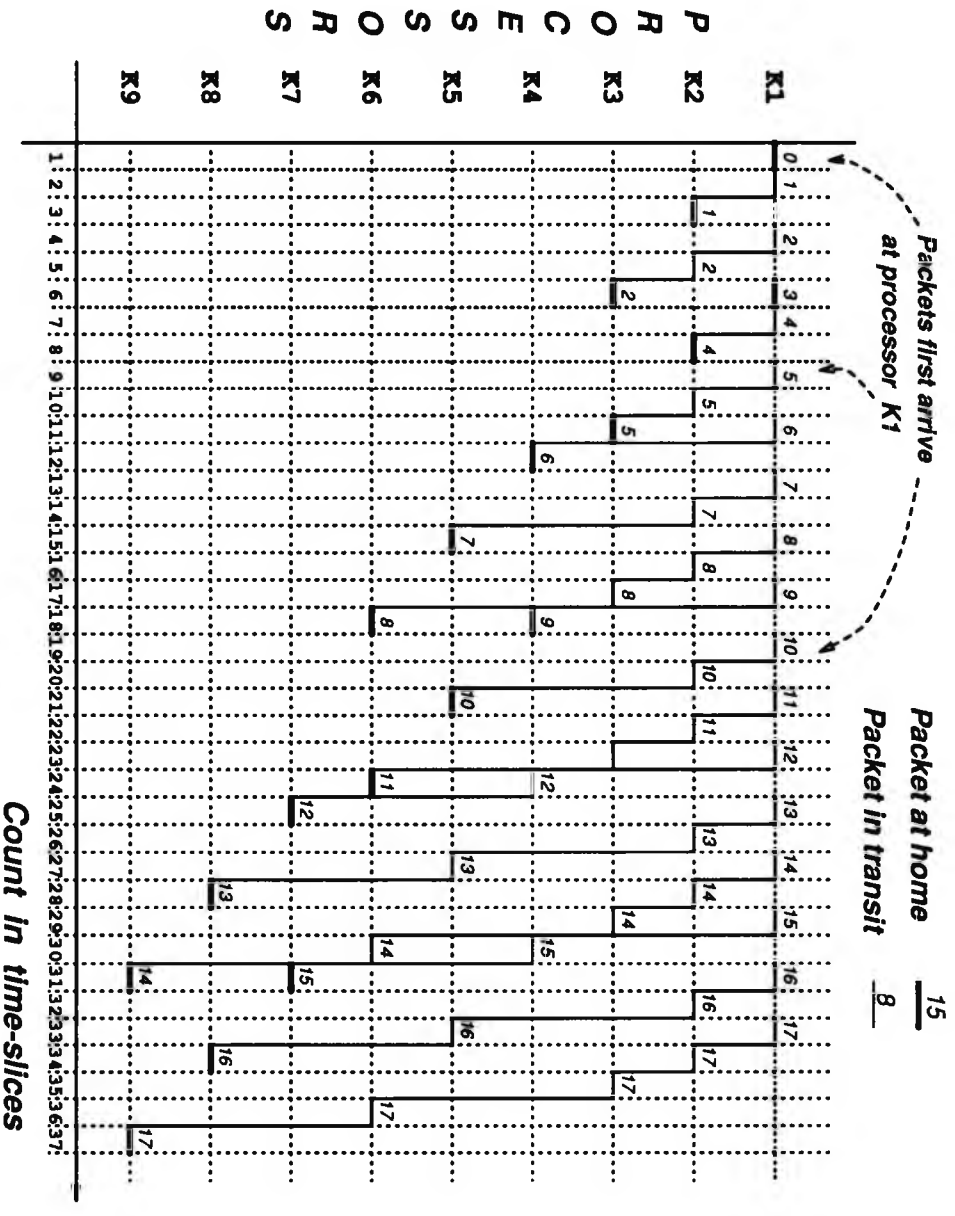


Figure 3.4: The temporal association of the arrival into the system, and travel to final destination, of each data packet for each transputer.

timings for square images, the corresponding estimated timings using Equation 3.12, and the percentage difference between them. The data is for a 2×2 array of 32-bit 20MHz T414 transputers, each with 1Mbyte of local RAM, hosted by a 32-bit 20MHz T800 with links capable of operating at a peak rate of 20Mbits/sec. Also, $b = 2$, such that the distributed partitions would be usable for 3×3 neighbourhood operations if necessary.

Image Packet Size	T_s	Image Array Size	Measured T_{meas}	Estimated T_{dc}	% Difference
512	3.0E-4	512×512	1.034	1.084	4.8%
256	1.7E-4	256×256	0.297	0.310	4.3%
128	8.6E-5	128×128	0.072	0.080	11.1%
64	5.4E-5	64×64	0.021	0.026	23.8%

Table 3.1: Measured and Estimated distribution and collection timings in seconds

Here are two explanations attributed to the differences between the measured and estimated timings. Firstly, the smaller the data packet, the more precision is required in the measurement procedures, since factors such as on/off-chip code and data can influence the performance. Secondly, link communications for the T414 require a receipt acknowledgement for every byte that is communicated; on the T800, this is implemented to overlap with the data transfer, thus allowing continuous transmission with no delays between data bytes. The time for T_s in Table 3.1 is thus a crude average of the data transfer rates in both directions between the host T800 and the NW T414. This means that for this particular case in fact $T_d <> T_c$. These experiments were conducted as such since at the time no alternative equipment was available.

The results presented above further confirm that when possible, longer message lengths and fewer link operations should be used to transfer (image) data. It must also be noted that the results of most image processing applications are rarely themselves an entire image, and amount to a fraction of the original data load. Then most often $T_d \gg T_c$.

3.2.2 More Network Statistics

A number of other measurements can be derived for this scheme. Let it be assumed that T_s is known more precisely and a more homogeneous

network of processors are available, e.g. all T414s or all T800s. Let p_i for $i\epsilon\{0,...,(lMN-1)\}$, represent each packet of data in the system, which are also in the order they are sent to the network. The packet's row and column destination transputer is given by, $m_i = \frac{p_i}{lN}$, $n_i = p_i \bmod N$ for $0 \leq m \leq M-1$ and $0 \leq n \leq N-1$. Then, for each packet p_i, it is possible to find, A_{p_i}, as the time for packet p_i to enter the system, and T_{p_i}, as the time for packet p_i to reach its destination transputer,

if $(p_i < lN)$ and $(n_i = 0)$ then ..... (data on NW transputer)

$$A_{p_i} = T_s((2N-1)\frac{p_i}{N} + 1) \tag{3.13}$$

else if $(p_i < lN)$ and $(n_i <> 0)$ then ..... (data on first row)

$$A_{p_i} = T_s((2N-1)\frac{p_i}{N} + 2n_i) \tag{3.14}$$

else if $(p_i \geq lN)$ then .....(data on other rows)

$$A_{p_i} = T_s(2p_i - l + 1) \tag{3.15}$$

and having found A_{p_i},

$$T_{p_i} = A_{p_i} + T_s(m_i + n_i) \tag{3.16}$$

T_f can now be calculated using Equation 3.16 where $f = p_i = (lMN - 1)$. Another statistic is $T_{E_{mn}}$ which gives the time elapsed before any transputer in position (m,n) has received its own full complement of data,

$$T_{E_{mn}} = T_s(l(2M(m+1)-1)+m+3n-2M-1) \quad \text{(0<m<M, 0<n<N)} \tag{3.17}$$

$$T_{E_{00}} = T_s(l(2M-1)+2(1-M)) \quad \text{(m=0, n=0)} \tag{3.18}$$

More measurements that can also be derived are, $Volume_{Links}$, and $Volume_{data}$, which provide a count of the total number of times all links are set-up for transfer, and the total amount of data moved about the system respectively,

$$Volume_{Links} = 2lMN(M+N-1) \tag{3.19}$$

$$Volume_{data} = 2lw(MN)(M+N-1) \tag{3.20}$$

Many of the above operations occur concurrently and therefore these parameters are cited only for the sake of completeness. They could perhaps best be used for comparison with similar features in other routing strategies.

Note that all the above equations have been also derived for the *reverse-feed* distribution mechanism which was stated earlier to be a more efficient distribution mechanism, but will be omitted to preserve space.

The presentation in this section encompassed both an improved version of the data distribution and collection algorithm for a transputer network as proposed originally by Morrow et. al. [MCK+88], and an analysis and breakdown of the algorithm to determine various characteristics. Furthermore, the results may be used to complement the performance measures introduced by Browne and Hodgson [BH89]. Since the method described here is completely general for any image sizes and transputer array sizes, be they square or rectangular, the routing method will hereafter be referred to with the prefix of *general*, or *flexible*. This method will now be applied to a simple real-time task in label inspection, as a static image processing application, to determine its suitability in a practical situation compared with a more *customised* routing method. Hence, the emphasis will be on the application of the message-passing schemes for evaluation for a real-time environment, rather than a precise approach to label inspection as an industrial inspection task. In the following application, little in the classification of faults and defects will be presented. Instead, the concentration will be on the passing of labels that satisfy some simple tests. Nevertheless, a short review of the field of label inspection will be furnished next.

Please note that the discussions that follow, all the way to the end of this chapter, are kept to a minimum, and are detailed enough to provide an overall view of the work carried out only. This has been dictated by lack of space.

3.3 Brief Overview of Label Inspection

The issue of *label inspection* may be regarded as a sub-problem in the domain of *industrial and commercial inspection*, where the general idea is to simply perform rigorous checks on the outward quality of a product before its presentation to the consumer. Figure 3.5 illustrates the type of faults that are of major concern in label inspection. The field of industrial inspection has a wide application area, for example printed circuit-board inspection [CH82], food inspection [Dav90],

specialised packaging inspection [Sha89], or general manufactured-item defect measurement [Bat79, Oka84]. A general review of automated visual inspection techniques and systems can be found in [CH82, CD86]. The application subject of this chapter and the next will be on label inspection only.

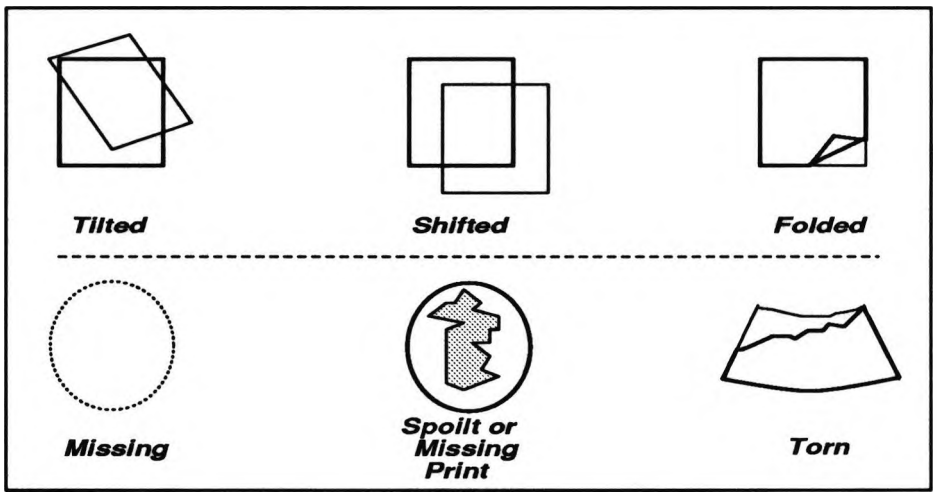


Figure 3.5: Typical defects found through label inspection

Most typical defect detection and inspection techniques are mainly concerned with the extraction of a number of particular features from the image, which are used in the verification of the quality of the product by matching those features against a pre-defined model. Many such techniques are reviewed in [CD86]. This approach is also applicable to label inspection and is adopted for this work. Moreover, in this work a crude approach will be adopted where the matching criteria will produce a simple accept/reject decision. However, prior to analysing the *ad hoc* procedures implemented here, some past, related work in the field is reviewed.

Unfortunately, few industrial sources reveal the nature of their label inspection techniques, and they number amongst the major researchers in the field. One major aspect of label inspection is the process of inspecting the quality of the print on the label. For example, correctness of labels on medicinal containers is an application area where the aesthetics is not as important as integrity.

Commercial systems have been reported in [Hol86, Dic88] with special routines which check for label placement, correct label, and container fill height, with success rates of about 5 to 6 products every second.

Dillman [Dil82], from Object Recognition Systems Inc., uses the sum of row and column pixels of the image of a container as feature values to perform overall label inspection against a prototype. For want of a more robust approach, Dillman applies the following scheme. A feature grid using the features already described is constructed and overlayed onto the prototype. This grid is then shifted in all directions for a pre-determined number of pixels, and a one to one match against the prototype is applied. The first match within pre-determined thresholds is accepted, otherwise the container is rejected. The author achieves an inspection rate of two containers/second using 64x64 images in a fully implemented working system.

Fang et. al. [FKS83] describe their experiment in label inspection consisting of thresholding, registration, and template-matching stages. Each image is converted to a binary image using locally adaptive thresholding. A circular scanning technique is applied (label scale must be known) to points on the label boundary located by a raster scan. Very briefly, this consists of the registration of all points lying on the edge of the label at a certain distance away from the central point located earlier. It is possible that this pattern of points may yield the location and orientation of the label. This can be verified by comparing against stored patterns from a model label. In fact, rather than matching the point patterns, a set of run-coded arc lengths along the circular mask connecting the point patterns is used. Several circular scans of the label are taken to reliably deduce a registration transformation. Using the most acceptable registration, the object is aligned with the model. Using dilation and erosion techniques, local point sets in the object are grown or shrunk respectively, and tested against the model as superset or subset of the model label. If these tests are not satisfied, a '1' is recorded in a "difference" image. When the complete image is processed, it is divided into overlapping 20x20 pixel bins. Should any bin contain more than three error points, defined by pixels with 7 of their neighbouring pixels set to '1', then the label is rejected as defective. No results for this proposed scheme are given, although sample experiments are illustrated.

Shabushnig [Sha89] presents a simple method for testing the presence of a label by finding the distance from the top of a bottle to a region where the pixels representing a label are expected to be found. No attention is paid to the rest of the label, so defects in the lower regions of the label can go un-noticed.

Yamamura of Fuji Electric. Co. describes an automated label inspection system based on direct scanning methods of the image to search

for particular labels in 64x48 binary images [Yam83]. The algorithms scan the image to detect starting and finishing pixels, and they measure width and height of the expected label. Again, a model is used for implementing feature comparison tests. When applied to a whisky bottling line, a processing speed of 5 bottles/second was achieved. The methods used by Yamamura are very straight-forward and *ad hoc*, techniques based on similar principles are used for the work to be described in the next section. Furthermore, the algorithms presented here will be operating on much larger images, which will allow a higher degree of accuracy. Implemented for concurrent performance, they will allow a faster and higher rate of inspection.

Of interest, but out of the scope of this coverage, are also the work of Casasent and Richards [CR88] who inspect health warning messages on cigarette packets using an optical architecture for a Hough transform and 1-D correlation functions.

The points common to all the works described above are the controlled environmental conditions that are either assumed or especially set up to ease the task of inspection. Hence for similar reasons, this research work will also assume constant and controllable illumination, little or no vibration, good contrast between labels and their background, well determined size, and appropriate positioning relative to packaging.

3.4 Ad-Hoc Solutions

The following is a report to show the performance of a multi-transputer imaging system applied to a real-time inspection problem, where some simple tests are conducted on the position of labels on products as well as a crude look at their overall quality. In Section 3.2, a data distribution method was presented which was generalised, flexible, and universally applicable to transputer arrays of any sizes. However, the timings presented for the distribution and collection of data show some limitations for the applicability of that scheme for a real-time situation, for example when a number of labels are to be inspected every second. A label inspection system was initially devised and implemented on the generalised network; this resulted in the design of algorithms that would be general enough to be able to cope with an increase in the size of the system, since flexibility and increase of processing power were regarded as major goals. This meant the standard distribution of the label across any $M \times N$ array, with each transputer ending with any part of the label in the image. Therefore, the algorithms needed to be very thorough to be able to detect a part of a label in each transputer.

Naturally, although very flexible, this proved to be an expensive approach. These issues will be returned to later in this section. Since the work was an industrial assignment and an evaluation study for the implementation of a label inspection system (largely for examining the correct positioning of the label) using a cheap, affordable set-up, then a more dedicated, customised system was designed which used only 5 transputers, including the host, to obtain a better inspection rate.

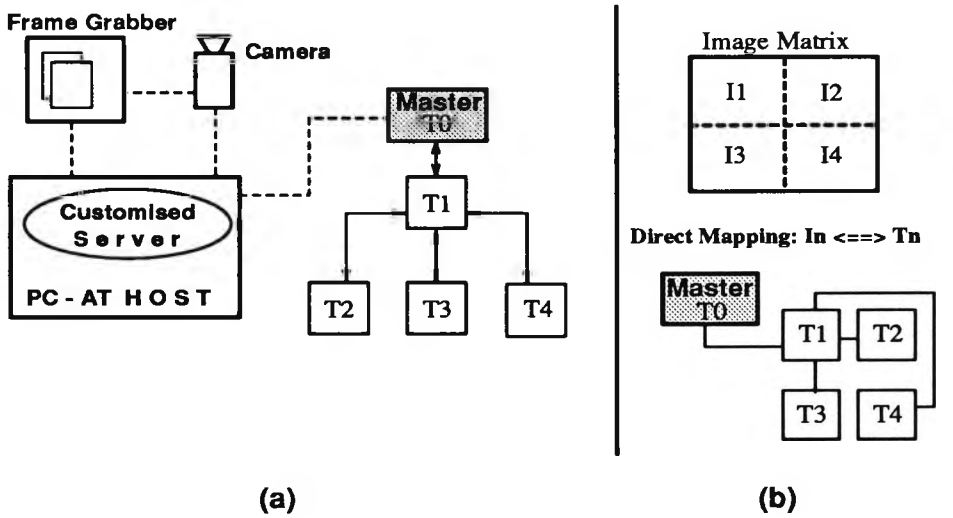


Figure 3.6: (a) A simple depiction of the configuration used for label inspection, (b) A direct mapping of the image to the customised network transputers

The hardware available at the time was configured for the customised network as shown in Figure 3.6 (Please also see Appendix A). The method of distribution and collection of the data across the 4 transputer array is as follows. The 2D image array is spread out into a single 1D vector which is subsequently squirted, in whole, from the main transputer to T1 (Figure 3.6(a)). Transputer T1 extracts I1, I2, I3 and I4 from the vector, retains I1, and passes the rest to T2, T3 and T4 (as vectors) respectively. At this stage, each transputer has its own image section and will be working on the data. Using the OCCAM RETYPE facility, there is no need for actual hard-coded translation of 2D to 1D to 2D vectors, and therefore no time is wasted on those operations. The mapping of the image to the network is also shown in Figure 3.6(b)).

The post-processing resultant image can then be gathered similarly in the opposite direction. Table 3.2 shows the timings for the distribution and collection of image data using both the customised and flexible

Image Size	Flexible Method	Customised Method
bytes	*seconds*	
512x512	1.034	0.757
256x256	0.297	0.192
128x128	0.072	0.049
64x64	0.021	0.013

Table 3.2: Distribution and collection timings for the two routing approaches used

methods, when applied to various image sizes on similar number of transputers. The differences may seem insignificant, but when intending to inspect several labels per second, every microsecond counts. The results shown in Table 3.2 will be less for the label inspection exercise since post-processing results will occupy a few bytes only and there is no necessity for gathering the resultant image.

3.4.1 The Label Inspection Methods

Some approaches for tackling the label inspection problem were reviewed before. However, methods such as template matching [FKS83] and Hough transforms [CR88] are either not fast enough for a real-time environment, unless the bottlenecks involved are hardwired (e.g. cosine/sine evaluation by look-up table) or they require expensive, cantankerous hardware. Here, some simple and efficient algorithms are proposed that will detect some very common defects such as those classified in Figure 3.5. The main principle involved in the following algorithms is the detection of corner points. Again, many expensive corner detection techniques exist such as those to be found in [KR82, ZH83, HS88, Dav90], but here the label is known to be in sharp contrast against its background with the corners expected to exist within certain areas of the image since the magnification can be pre-determined. Their presence is checked for, essential measurements are made and compared with a known perfect model, and if the results are within an acceptable threshold, the label is passed. The system can be setup initially using the menu-system of TIPS to determine the desired thresholds and tolerances.

3.4.2 Rectangular Labels

Initially, some features of a perfect model of a label are specified by the user, such as expected height and width, tolerances on variation in position and orientation of the label etc. This set of measurements, including a histogram of the gray level image of the label, are stored and used in the process of inspection.

On the Dedicated Transputer System

To detect a rectangular label, the master transputer distributes the current image frame over the four transputers in the network, as discussed earlier. Each transputer would then have a part of the image which it will threshold into a binary representation prior to scanning it pixel by pixel. The method and direction of scanning for each transputer is different and is shown in Figure 3.7.

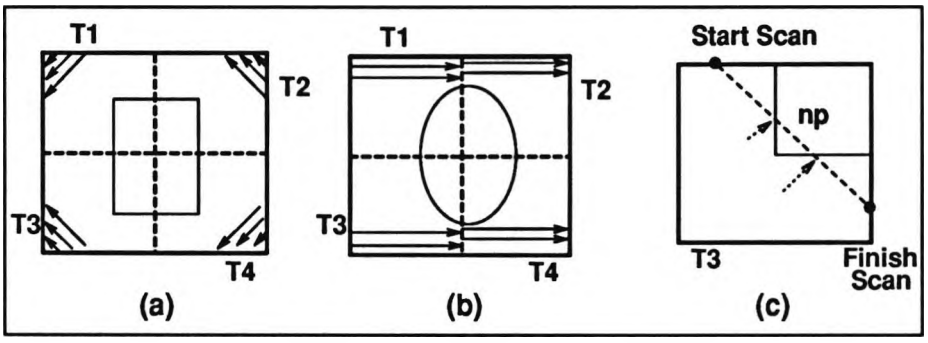


Figure 3.7: (a) Rectangular label scan, (b) Oval label scan, (c) A single scan line on T3 for a rectangular label

The direction of scanning is always perpendicular to the expected corner point, and the first scanned line found to satisfy the following conditions is accepted as the line touching a corner point:

- the start pixel value must be the same as the final pixel value,
- the pixels scanned must show a change of state at least once,
- the number of label pixels *(np)* must satisfy $0 < np <= L$, where L is dependent on the general quality of image. (Figure 3.7(c)),
- the above conditions must also be satisfied by the next two (user adjustable) scan lines (this condition will help deal with noise).

Essentially similar, but nevertheless different programs run on T1, T2, T3 and T4, allowing the system to be loosely termed as MIMD. Each transputer then returns either the address of the pixel which it has found to be the corner point, or a "not found" flag if no corner point was encountered.

On the Flexible Transputer System

In a truly flexible implementation, the system's processing power must be able to grow without resulting in further costs. In the system implemented here, the inspection algorithms are designed to be able to cope with increasing array size, regardless of the number of transputers used. Each transputer will receive a part of the image which may be any one of the possibilities illustrated in Figure 3.8. Since edge information is also passed at the distribution stage, there are no complications if the label falls exactly on the border of two transputers.

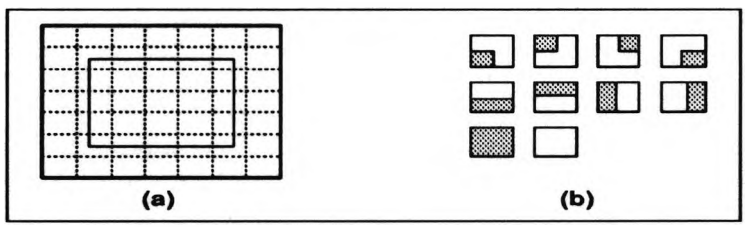


Figure 3.8: (a) Label image across $M \times N$ transputer array, (b) Possible distribution of label segments per transputer

The search strategy is more complex. Each transputer has to scan the whole of its own image section from all directions, stopping only when a possible corner or the end of scan is reached. Hence, as well as the extra distribution time, this method is considerably more costly. Also, the results of only a few of the transputers will be at all significant, and the use of too many transputers would be a waste of resources. (In fact, this spells out why a straight-forward customised approach is a much cheaper, and more appropriate solution).

While the transputers in either method are returning the corner pixel addresses to the master transputer, they are ready and receiving the next image. In turn, once the master transputer has dispatched the next image, it sorts the information returned for the last frame and decides if the inspected label (il) is perfect or faulty. This is determined by using a user-defined label (ul) as a template and checking the following

against it for each corner:

ABS(ul Corner Address - il Corner Address) < Allowance Threshold

This condition will indicate if the label is shifted, tilted, missing, folded, or torn at the corners, with the *Allowance Threshold* allowing for positional changes within the tolerance of the manufacturers. To detect the presence and legibility of the print on the label, the following is computed,

$$MAX(\frac{|H_i - H_i'|}{H_i}) \tag{3.21}$$

where H_i is the linear histogram of the user-defined label and H_i' is the linear histogram of the label under inspection, and both are functions of intensity i, ($i\epsilon\{0...255\}$) [Bat79]. The computed value is then compared to a predefined tolerance parameter, which is re-set heuristically by the system as labels are checked during a run. These calculations are performed only on a predefined area of the label called the *print check area.* Constant illumination is very important for this stage to perform accurately.

Notice that the whole verification process is performed sequentially on a single transputer, but in parallel to the work of the network on the next label image.

3.4.3 Oval Labels

For Oval labels, two "corner points" are searched for. These are the peak point and the base point of the label, and are found by tracing downwards on transputers T1 and T2, and tracing upwards on transputers T3 and T4 (Figure 3.7(b)). Similar comparisons to those used for a rectangular label are used to pass or fail oval labels. This approach does not map very well on the flexible multi-processor array due to the less regular distribution across the different processors; too many noise points are accepted as corner points in each transputer. (This was simulated in software for varying $M \times N$ values on a single transputer). A more sophisticated (and hence more expensive) method will be necessary for a successful implementation, and thus no results for Oval labels on the flexible network will be produced.

3.4.4 Acute-angled Labels

These are treated similarly to rectangular labels, and four distinctive corner points are searched for. The search involves a normal raster scan for transputers T1 and T2 and a vertical left to right scan for transputers T3 and T4. Once the points are returned to the master processor, the defects, if any, are detected using the same comparison methods as those used for a rectangular label.

3.4.5 Other Labels

The same approach may be employed to develop the software to detect most other types of labels.

3.5 Results for the Experiments

The algorithms were tested using 128x128 and 256x256 images of 256 gray levels, with *print check area*s of various sizes. The average image distribution, results collection, and processing times, in seconds, are shown in Tables 3.3, 3.4, and 3.5 for various configurations. Processing efficiency percentages against single transputer implementations are shown in brackets for both flexible and customised systems. Table 3.5 indicates a fast rate of inspection using the customised network configuration. When comparing the average processing times per label against the single transputer implementation, the average efficiency is 72% for 128x128 images and 59% for 256x256 images. In contrast, the flexible 2x2 array of transputers using the general distribution method, and the generalised inspection algorithms, achieves only corresponding efficiency percentages of 40% and 33%. Pictorial results are not shown here, instead they are produced following the work described in Chapter 4.

There are a number of ways for improving the system performance. Firstly, the scanning could be conducted at every other scan-line reducing the search by a factor of two. This will naturally affect the degree of accuracy, and can only be attempted in the "installed" situation by observing the resulting acceptance and rejection rates. Secondly, given a conveyor belt system, the host transputer's *accept* or *reject* decision may not be immediately required, and the transputers could be employed to perform more stringent checks on the labels, such as those that will be described in Chapter 4. Thirdly, should the production-

Label Type	Image Size	Check Area	Average/Label	Labels/Sec
Rectangular	256x256	60x110	0.514	1.9
	128x128	30x60	0.160	6.3
Oval	256x256	95x155	0.487	2.1
	128x128	40x80	0.144	6.9
Acute-angled	256x256	30x125	0.535	1.9
	128x128	30x60	0.170	5.9

Table 3.3: Results for label inspection performed on a single transputer

Label Type	Image Size	Check Area	Average/Label	Labels/Sec
Rectangular	256x256	60x110	0.432 (30%)	2.3
	128x128	30x60	0.111 (36%)	9.0
Oval	n/a	n/a	n/a	n/a
Acute-angled	256x256	30x125	0.382 (35%)	2.6
	128x128	30x60	0.097 (44%)	10.3

Table 3.4: Results for the *flexible* 2 x 2 array of transputers

Label Type	Image Size	Check Area	Average/Label	Labels/Sec
Rectangular	256x256	60x110	0.214 (60%)	4.7
	128x128	30x60	0.054 (74%)	18.5
Oval	256x256	95x155	0.199 (61%)	5.0
	128x128	40x80	0.049 (73%)	20.4
Acute-angled	256x256	30x125	0.239 (56%)	4.2
	128x128	30x60	0.061 (70%)	16.4

Table 3.5: Results for the *customised* configuration of transputers

line speed be able to supply larger numbers of labels past the system than what it is currently able to detect, then a number of customised systems could be implemented in parallel.

3.6 Summary and Conclusions

In this chapter, a message-passing scheme, namely the *forward-feed distribution scheme*, suitable for data distribution and collection in a regular SIMD array of transputers was presented. Furthermore, the distribution and collection mechanism was analysed to provide equations for determining vital performance-related statistics. The *reverse-feed distribution scheme* was also introduced as the favoured method.

Naturally, alternative methods of controlled message-routing exist. This may vary in the manner the data is packaged or passed between source and destination. For example, the data may be passed in alternative formats, e.g. as complete sub-image blocks, which would reduce the number of link communications, but increase the latency. This method has been implemented and used during this research work, for example for MIMD processing of the object detection investigation in Section 6.3. In fact it has also been used for the customised network in this chapter. When using small numbers of transputers, little difference with other efficient methods can be observed. The type of analysis presented in this chapter could simply have been reproduced for this method too. However, the point that the control-driven method is completely analysable, is already made, and a repetition of similar work would have been an unnecessary step and a waste of precious space.

The data may also be routed differently. Two mechanisms of faster source to destination addressing are as follows. Firstly, shortest path addressing methods can be used for routing data across the transputer arrays (by establishing wrap-around communications). This is most efficient when using large numbers of processors, and is also of benefit to complement a system where inter-processor communication is necessary. As such, it is not quite as important (for low-level image processing) on a SIMD array as it is for a MIMD system, where it might even prove obligatory. Secondly, to increase the bandwidth of the message-routing algorithm, it may be possible to use the extra links on the controller to connect to more distribution and collection points on the network. (This could be viewed as a very simple but alternative version of a shortest path method.) The scheme will at best be limited to three connections to the network for a transputer system, and reduce the communication costs by approximately a factor of 3.

The implementation would require special code at the gateway processors in the network. However, an example of where the use of the extra links will not be possible is demonstrated in the MATCH system in Chapter 6, where there simply are not any free links available on the controller processor. Nevertheless, this option will be returned to later as a useful scheme in other configurations.

The routing method presented in this chapter will be used as a communication tool for all control-driven investigations in all future work in this thesis. The use of the scheme was demonstrated and compared against a customised routing method with application to label inspection. It will be employed again in Chapter 4 for the implementation of the Sobel operator and the Hough transform, and in Chapter 6 for the implementation of the Canny operator.

Also presented in this chapter were a brief review of the field of label inspection and a study into the feasibility of the use of a cheap, affordable system with a high processing rate of inspection. Although the customised system described has room for improvement, for example by using more stringent inspection tests, it also has verified that under assumed conditions it can deliver a realistic real-time inspection rate using significantly larger label images than, say, [Dil82, Yam83].

Unfortunately, the investigation has also shown that the communications rate can prove to be a major bottleneck. This is specially so in the generalised network because of its flexible distribution and collection strategy which nevertheless is necessary when the system array size is expected to increase, and software and communications alterations are to be avoided. The issue of communication as a principal problem in the use of transputers will be returned to at a later stage.

In general summary of the latter sections of this chapter, to illustrate the ideas on mapping and communications for low-level image processing operations, a real problem in automatic inspection was considered to allow the evaluation of real-time problems in mapping and communication. The solutions were trivial and crude, and they showed that they limit the expansion of the system since they are not easily divisible across multiple processors. Therefore, more isotropic solutions where every processor would run the same code (geometric parallelism) would be preferable, unless a very dedicated, customised MIMD approach more aggressive than that presented here can be employed. A more extensive solution, with a high potential for performing in real-time in inspecting labels, is presented in Chapter 4 where geometric parallelism is fully exercised.

Chapter 4

Parallel Realisations of the Hough Transform

4.1 Introduction

This chapter continues the work from Chapter 3 by considering an altogether different approach to *geometric parallelism*, namely *demand-driven task farm* or *farm parallelism*. Following a short discussion of the characteristics of this scheme, the method will be compared against the *control-driven* approach in terms of efficiency. The Sobel operation and the sub-image Hough transform will be used as benchmarks. In fact, a section dedicated to the sub-image Hough transform will portray its significance as an important tool for a parallelised edge segmentation technique, and as a demonstration of this, the sub-image Hough transform will be used for a more sophisticated approach to the problem of label inspection. Ways of improving the performance of the farm will be suggested.

4.2 The Task Farm

The demand-driven processor farm model was originally proposed by May and Shepherd [MS87] for a graphical representation of the Mandelbrot set, based on the general theory of the task queue model. The demand-driven model of parallelism is identical to the control-driven model in so far as each processor performs the same sequence of algorithms on the data, and that the (image) data is again sub-divided into equal-sized data segments. However, the data is partitioned into a significantly larger number of (small) data segments than the number

of processors in the network would suggest. This implies that a higher degree of communications is involved in the distribution and allocation of the task data. Described simply, the processor farm model consists of a farmer (master) processor which conducts the proceedings by making available the aforementioned task-data segments to a number of worker (slave) processors which then actually do the work. The farmer processor communicates tasks to the farm as and when necessary and collects results data as and when they are produced. This can happen in many ways, three of which are described here in increasing order of efficiency:

- Each slave processor sends a request to the master processor for a new task, which it returns upon completion of the computation, and follows it with a request for the next task,
- Each slave processor sends a request to the master processor for a new task, but uses a cache mechanism to buffer an extra work packet. When it finishes work on the first work packet, it sends a new request for more work and continues by working on the buffered data packet. This allows an overlap of computation with communication,
- Communications are reduced by scrapping the request access of the slave processors. Instead, each slave processor functions by simply accepting work when its own buffer is empty. Thus, as in the previous case, it always works on one task while it holds another ready in its buffer. While it is busy as such, it passes any incoming data packets to its neighbouring processor via a routing mechanism. The system is kept tightly balanced by the master processor by only supplying enough work to have every processor busy and fully buffered with work. Thus, as soon as it receives some results, it sends another work packet to the network which will be automatically passed around until it reaches an empty buffer.

The Mandelbrot example in [MS87] is CPU intensive and requires limited communications, thus no buffers are used in the implementation. For the tasks considered in this work, the latter of the three methods was found the most efficient, and its actions are depicted in Figure 4.1. The linear topology is the most popular configuration used for farm parallelism, and is suitable for algorithms with a relatively small communications overhead. Suggestions for improvements on this are provided later in this chapter. Figure 4.1 shows a simplified breakdown of the tasks within the master and the nodes. The master processor,

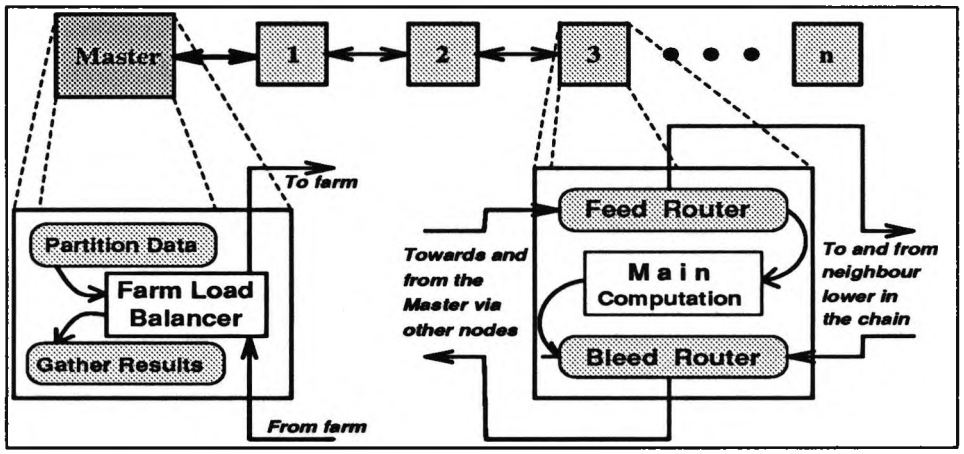


Figure 4.1: A linear processor farm and some of its major processes.

which is both the fountain-head of data packets and the reservoir of processed results, runs three major processes operating in parallel,

- a process which breaks up the data space into the desired partitions and sends them to the farm load-balancer process,
- the load-balancer process which hands out the image-data partitions to fill up the network, and then continues to supply the network with more data partitions when results are returned to it. These results are passed on to the "gatherer" process as they arrive,
- a "gatherer" process which receives results data from the network via the load-balancer, and gathers them into the appropriate format defined by the application. For example if each result packet is an image partition, this process would map it into its spatial position.

Data routing takes place as tasks enter the farm network. The router on each node checks the local buffer to see if it will be able to accept further work. If so, the work packet is buffered and used by the processor when the CPU becomes free, otherwise the router passes the data to the next processor along in the chain. Therefore, a number of concurrent activities take place on a farm processor,

- forward to local buffer or to next processer in the chain, the next work packet depending on the status of the local buffer (feed

router),

- perform user application using the next available data packet. When finished, read the next packet of work from the buffer,
- accept results from the local computation process, and also receive results from processors further down the chain (bleed router). These are input using the PRI ALT construct, which monitors the incoming link channel and the local processor channel with decreasing priority. The results are directed up the chain towards the master processor.

All inter-processor routing is performed at high priority to avoid processor idleness across the farm. Note that all the tasks may be processed and returned from the farm in any order, and that there is no inter-task communication.

The above approach is that used for all the relevant work in this thesis. The nature of the approach, its performance, and the possibilities regarding its improvement will be discussed later in this chapter. For a mathematical modelling of the processor farm the reader is referred to [TD90]. In the next section, the sub-image Hough transform will be discussed. It will then be implemented in Section 4.4 on both the processor farm and the generalised array of transputers from Chapter 3.

4.3 The Sub-Image Hough Transform

The (ρ, θ) Hough transform which was introduced in Section 2.2.3 was said to be better than the (m, c) (gradient, intercept) method because it is bounded: the angle θ can only vary from 0 to 2π and ρ can only vary from 0 to half the diagonal width of the image (assuming the origin is positioned at the centre of the image). The (m, c) transform is unbounded because the gradient can vary from 0 to ∞ and the intercept can be indeterminate. The (ρ, θ) Hough transform is most often implemented using the following sequence of operations,

1. Detect edges using an edge detector such as the Sobel or Canny which provides edge direction data,
2. Threshold edges from the background,
3. For each edge pixel compute the normal and angle, and increment accumulator space at this location,

4. Search accumulator space for all peaks,

5. Back project for each peak into the image and keep portions of straight line segments that correspond to edge pixels.

In most cases, the Hough transform is used to determine all candidate straight lines over the whole image, i.e. all the edge pixels in the image contribute to one accumulator space. Candidate straight lines are detected as peaks in the accumulator space that specify the (ρ, θ) parameters. Back projection is then used to determine the parts of the lines supported by edge pixels in close proximity to the lines in the image. A straight line will be hypothesised as occurring across the whole image as it will be unbounded by end points. However, only part or parts of this line will occur in the image. It is relatively easy to detect long straight lines. Short lines are detected with more difficulty because these produce low amplitude peaks in the accumulator space which can be masked by long lines with similar parameters (and therefore close by in the image). Furthermore, the accumulator space can be large. For example, assuming a 1° resolution for angle and a 1 pixel resolution for the normal, for an image size of 256x256 the accumulator would be 181x360. Whilst this is not a significant amount of memory, it requires much searching to find the peaks as there may be many straight lines in an image. In addition, the Hough transform will not work if the image predominantly contains curved edges, as each curved edge will not generate a dominant peak in accumulator space. The normal solution to this is to use a different formulation for each type of curve, for example, the circle and the ellipse. However these require increased dimensionality of the Hough space, novel parameterisations or multiple stage accumulation.

An alternative formulation of the Hough transform is now described to overcome some of the problems mentioned above. It will be referred to as the $\rho\theta sHT$ transform, where the s signifies sub-images. Let the image be divided up into sub-images, as suggested by [Dav90], e.g. 16x16 pixels. The standard (ρ, θ) Hough transform is then performed on each sub-image and straight lines are detected. The accumulator resolution and size is chosen so as to detect curved edges as small straight line segments. The straight line segments are then processed at a later stage to detect long lines or circular arcs by linking them together. Using sub-images, the processing is simplified and has the following advantages,

- only a small accumulator space is required,
- searching is simplified since only a small number of lines can be present in a sub-image,

- curved edges are detected as approximations to straight lines,
- segmentation occurs between edges at large angles,
- sub-images with very few edge points can be ignored,
- **processing on sub-images can be performed in parallel.**

A small accumulator is feasible, since the number of possible orientations and positions of a line in each sub-image is restricted. Considering orientation, the best case resolution will be 1 pixel over the width (or length) of the sub-image. Therefore, the minimum resolvable angle, ϕ, will be:

$$\phi = tan^{-1}(\frac{1}{length}) \tag{4.1}$$

Hence, for a sub-image of 16x16 pixels, the minimum resolvable angle will be 3.58°, which results in $\frac{360}{3.58}$ intervals. The minimum resolvable radius, ϱ, will be 1 pixel and the longest value of ϱ will be from the centre of the sub-image to any one of the corners. For a 16x16 pixel sub-image this will be 11.31. The accumulator will then be, to the nearest integer, 101x12 bins in size.

Searching the accumulator is simplified since there are few lines present in each sub-image and hence few distinct peaks in the accumulator. The algorithm to find the peaks is shown below:

1. Scan the accumulator for the largest value.
2. Store away parameters.
3. Delete peak point and surrounding area.
4. Repeat until number of peaks detected = *Max_Peaks*

The algorithm repeatedly looks for the largest peak in the accumulator. When this is found, the parameters are stored and the peak is deleted. The surrounding region in the accumulator is also deleted as a peak will normally be part of a small region, so accumulator values nearby are part of the peak (please also see Figure 2.4). *Max_Peaks* is normally empirically set to the expected number of lines per sub-image plus an offset, for example *3 + 7* where *3* is the expected number of lines and *7* is the offset that allows for false peaks (in case the third stage of the algorithm fails to remove the peak completely).

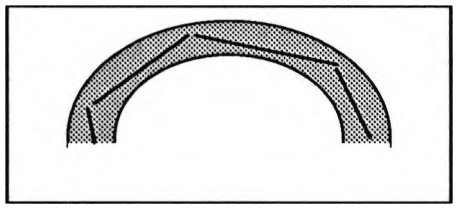


Figure 4.2: Edge region obtained from an image after Sobel edge detection and thresholding. The straight line approximation obtained from the Hough transform is shown in bold.

For the implementation of this algorithm a simple edge detector can be used, for instance the Sobel edge detector from which the orientation and magnitude information for each edge pixel can be determined, and it remains a reasonably inexpensive method. In addition, after thresholding, edges are normally represented by regions more than one pixel thick. This information allows curved edges to be detected more easily, as a straight line can be contained in the region generated by a curved edge as shown in Figure 4.2. These thick edges are often seen as a disadvantage of using the Sobel operator, but this is turned to advantage in this case, and will be seen as an important factor for inspecting oval and circular labels.

The output of the $\rho\theta sHT$ formulation for an image is a set of straight line approximations (represented by their end points), which are examined and grouped for further analysis. Typical groupings are collinear, vertex, curved edge and proximity. Thus higher level features can be extracted from the data.

4.4 Implementation of the Parallel $\rho\theta sHT$

The $\rho\theta sHT$ was implemented using both the control-driven model of computation on an array of transputers, and the demand-driven model on a linear chain of transputers. Principally in both, the processes of edge detection, Hough transformation, and peak detection are performed on each sub-image, and a list of lines local to the sub-image is formed. The master transputer in both computational models analyses all partial lists to form a more unified representation of the image. The determination of the number and size of sub-images will be discussed

shortly.

4.4.1 Control-Driven Model

The image data is the image of the whole label and it is distributed in equal segment sizes, including the border information for each segment, amongst the available processors. This used the *reverse-feed distribution* mechanism of Chapter 3. Each processor, having applied the Sobel filter to its own local image partition, splits it into (further) sub-images and performs the $\rho\theta sHT$, followed by peak-detection. These latter operations are performed on only those sub-images where the number of edge pixels exceeds a certain empirically-determined tolerance. Finally, each processor returns a list of all the lines found locally to the master transputer. This is illustrated in OCCAM pseudo-code in Figure 4.3.

```
PAR
  — Master Transputer
  PAR
    ... Send next image to processors
    SEQ
      ... Receive list of lines found in previous image
      ... Process list to formulate complete lines
      ... Analyse lines and decide to pass or reject label

  — In each processor in array
  SEQ
    ... Receive own local image partition including border edges
    ... Perform Sobel on complete local image partition
    SEQ i = 1 FOR number.of.sub-images
      IF
        (number of edge pixels) ≥ (pre-defined limit)
          SEQ
            ... Perform ρθsHT on sub-image i
            ... Detect peaks, locate lines, and form line list
        (otherwise)
          ... line list is empty
    ... Return list of lines found
```

Figure 4.3: Pseudo-OCCAM code showing the general format of the master and array processors for the inspection of labels.

4.4.2 Demand-Driven Model

In this implementation, the controller splits the entire image into the required sub-images and makes them available to the farm as discussed earlier in this chapter. Each packet is sent, including enough border pixels, to comply with requirements of the Sobel and Hough operations. Initially, farm processors are required to perform the Sobel filter on their packets of data. Again, only if the number of edge pixels found in any packet exceed the aforementioned tolerance level will the $\rho\theta sHT$ operation and peak-detection routines follow. Each processor then returns a list of all lines found in each sub-image. See Figure 4.4 for an OCCAM pseudo-code outline of the overall operations of the master farmer and the worker slaves.

```
PAR
  — For each image on Master transputer
  SEQ
    WHILE more.image.packets.available
      PAR
          ... Send next packet to farm when load allows
          ... Receive list of lines as they arrive from farm
    ... Analyse lines and decide to pass or reject label

  — In each processor in the farm
  PAR  — three main processes.
     ... Pass on line lists from other nodes, and local process
     ... keep or pass incoming work depending on local workload
     SEQ
       ... Perform Sobel and count number of edge pixels
       IF
         (number of edge pixels) ≥ (pre-defined limit)
           SEQ
             ... Perform ρθsHT on sub-image
             ... Detect peaks, locate lines, and form line list
         (otherwise)
           ... line list is empty
       ... Send to local router the latest line-list
```

Figure 4.4: Pseudo-OCCAM code showing the general format of the master and farm slaves for the inspection of labels.

4.4.3 Efficient Calculation of ρ

When calculating the corresponding value of ρ for each value assigned to θ, trigonometric functions can be dispensed with to save computation by employing the following scheme.

Given (g_x, g_y) as the local components of intensity at pixel co-ordinates (x, y) on a feature line in an image, let (x_0, y_0) be the foot of the normal ρ from the origin to that line. Please note, in this case the origin is taken as the centre of the sub-image, although this is not depicted as such in Figure 4.5. Also, the line may need to be produced, and this *is* depicted in the example in Figure 4.5. Then, it can be shown that,

$$\frac{g_y}{g_x} = \frac{y_0}{x_0} \tag{4.2}$$

$$(x - x_0)x_0 + (y - y_0)y_0 = 0 \tag{4.3}$$

$$\rho = \sqrt{x_0^2 + y_0^2} \tag{4.4}$$

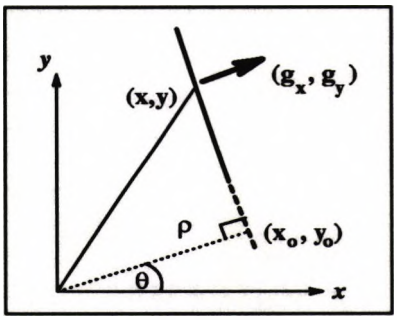


Figure 4.5: Diagram shows parameters used in calculating ρ

Thus, solving for ρ with incurring costs of additions, multiplications, just one division, and a dominating square root,

$$\rho = \frac{xg_x + yg_y}{\sqrt{g_x^2 + g_y^2}} \tag{4.5}$$

This is adopted after the technique introduced by Davies who uses the foot of the normal from the origin as a voting position in the parameter space [Dav90]. This technique in calculating ρ is computationally invaluable for the normal parameterisation as employed in this work. (An alternative approach could have been to allow each processor in the architecture to contain trigonometric look-up tables).

One of the conclusions of Davies's work on error analysis [Dav90] of the foot of the normal method, which is of great relevance to this work, is that the error in calculating ρ which arises due to edge detector errors, and to a greater degree to noise, is dependent on the distance of the line from the origin. Since the centre of the sub-image is the origin, the overall error is minimised, but not eliminated.

4.5 Label Inspection and the $\rho\theta sHT$

The topic of label inspection was briefly considered in Section 3.3, and an *ad hoc* approach was outlined in Section 3.4 for the detection of geometric errors of label shape and position, such as shift from normal position, sticking at a tilted angle, tear and folding (Figure 3.5). In this study, these faults will be examined again using more sophisticated techniques, including the $\rho\theta sHT$ approach to feature segmentation.

Two types of labels are considered: Rectangular and Oval. It was inferred in earlier discussion that the optimum sub-image size will depend on the length of the lines that need to be detected. Since in the search for a rectangular label the aim is to detect long straight lines, a fairly large sub-image size was selected for this inspection. In direct contrast, a much smaller sub-image size was selected for the oval label inspection. For a label of this shape, the aim is to detect short, straight line segments within the curved edges as shown in Figure 4.2.

Consider the processing of each type of label in both the control and demand-driven models. The broken-up short or long line segments in each sub-image pertaining to the label itself or any other features in the image are available at the end of the processing for each image. This is stated to further emphasise the care needed in choosing the optimum sub-image size to ensure that more computation is performed locally, leaving less post-processing for the master transputer. On the other hand, if the processing of each image is so lengthy that the master can afford more line-list processing time before the network is ready for the next image to be sent, then this introduces another feature to consider when implementing the complete system.

Empirically, the test images showed that given a 256x256 image, the following sub-image sizes should be used to achieve the most accurate, rather than the most efficient, results,

Rectangular Label	.....	**64 x 64**
Oval Label	.....	**16 x 16**

However, when alternative image sizes were tried for each label, the difference in efficiency was marginal, whereas the difference in accuracy was much more significant.

4.6 Processing and Inspection Results

The hardware configuration for both of the programming models consisted of a T800-20MHz host on a B004 board interfacing with a PC-AT, and four T414-20MHz transputer modules with 1MB RAM each, as network processors. More details are provided in Appendix A. The results for a single T414 implementation were produced on a 12MHz T414 device. All timings are in *milliseconds* and embrace the time spent on communications where applicable.

4.6.1 Sobel Filtering

Initially, the Sobel filter is considered. For the control-driven model, the best implementation is a straight mapping of the image to the processors, allowing each processor to number-crunch its way across its own image partition. Each partition also includes border edges.

System Configuration	Sobel Filter
Single T414-12MHz	2616
Single T800-20MHz	1132
2x2 CD array	508
DD Farm (16x16)	1273
DD Farm (32x32)	492
DD Farm (64x64)	478

Table 4.1: Results for the Sobel operation on various configurations. (CD=Control-Driven, DD=Demand-Driven)

Alternatively, in the demand-driven model, the image is split into a number of partitions much larger than the number of processors, allowing each processor to perform the Sobel operation on a number of smaller packets. To deduce the nearest to the ideal sub-image size, 16x16, 32x32, and 64x64 sub-image sizes were implemented. Note that the real image sizes communicated to processors needed to include the

border pixels of the partition, resulting in 18x18, 34x34, and 66x66 sub-image sizes. These results are shown in Table 4.1 including the sequential implementation times for a single T414 and a single T800 processor.

The Sobel filter in this implementation uses the response provided by the absolute magnitude of the horizontal and vertical gradients, and the processing includes the time spent on thresholding each resultant pixel. Note that due to the large number of packets in the 16x16 sub-image case, the communication load is greatly increased and four T414s can not match the performance of one T800 or the other network cases. This changes drastically when the sub-image sizes are increased, and therefore much of the communications is overlapped with the computation in the demand-driven model.

4.6.2 $\rho\theta sHT$ Processing

Table 4.2 and Table 4.3 show the total processing time in *milliseconds*, for the inspection of rectangular and oval labels on different transputer configurations using the aforementioned image and sub-image sizes. The totals are for the stages consisting of Sobel edge detection (including thresholding), the $\rho\theta sHT$ transform, and the complete process of peak detection and line-list formation.

System Configuration	Rectangular Label
Single T414-12MHz	18629
Single T800-20MHz	5815
Control-Driven Array	3428
Demand-Driven Farm	2575

Table 4.2: Results for the inspection of the rectangular label test-image using 64x64 sub-images on different configurations.

The pictorial results are shown in Figures 4.6 and 4.7. The images in the top-right of each diagram show for each label the $\rho\theta sHT$ transformation, using the corresponding sub-image sizes.

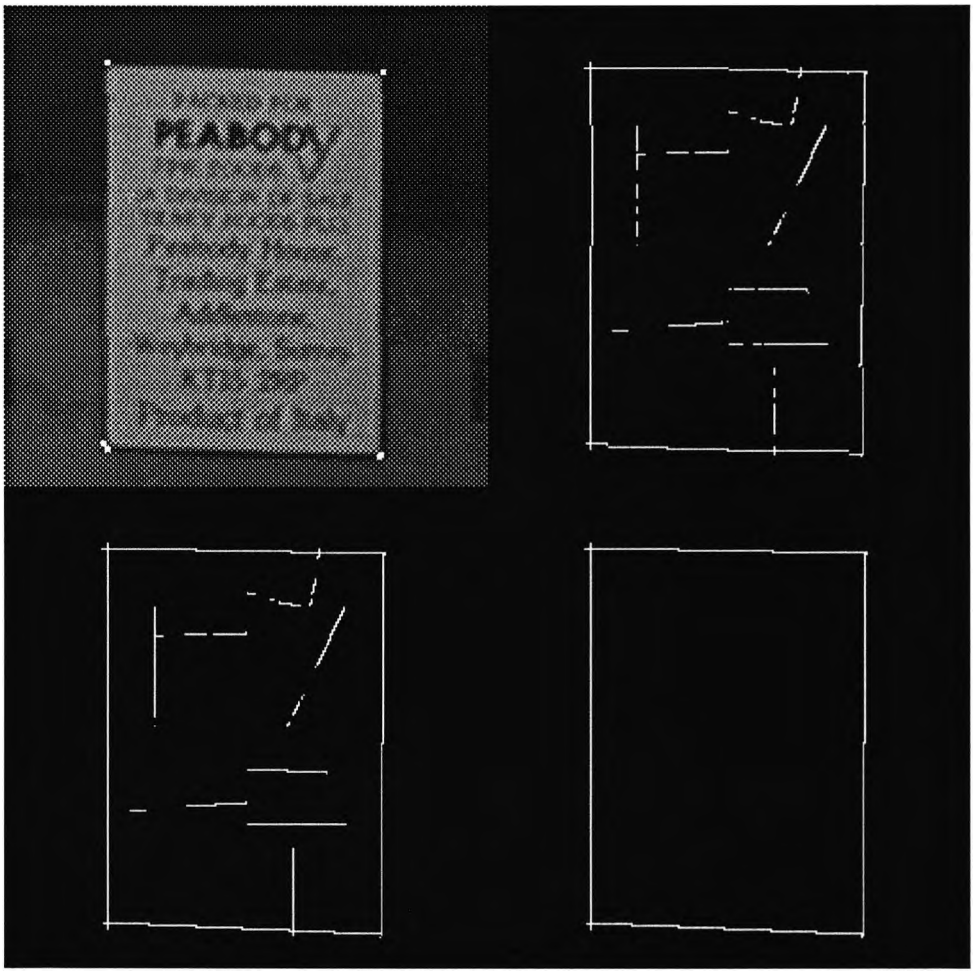


Figure 4.6: (top-left) Original image marked with possible corner points, (top-right) After $\rho\theta sHT$ transformation on network with 64x64 sub-images, (bottom-left) After line proximity analysis, (bottom-right) The final acceptable four boundary lines providing the outline of label and its approximate corner points.

Figure 4.7: (top-left) Original image marked with possible centres, (top-right) After $\rho\theta sHT$ transformation on network with 16x16 sub-images, (bottom-left) After aspect ratio transformation, (bottom-right) Normals of all the lines with peaks at crossing points giving possible centres (Shown with white dots - cf. with top-left)

System Configuration	Oval Label
Single T414-12MHz	24430
Single T800-20MHz	6711
Control-Driven Array	4520
Demand-Driven Farm	2874

Table 4.3: Results for the inspection of the oval label test-image using 16x16 sub-images on different configurations.

4.6.3 Notes on the Execution Times

Except for the sequential implementations, all columns of results include the time spent on the distribution of the image and collection of results, i.e. all conceivable link communications.

The execution time of the $\rho\theta sHT$ is directly dependent on the number of edge features offered after the application of the Sobel edge detector; this is clearly demonstrated by the timing differences for the rectangular and oval test images. These test images were found to produce approximately 7800 and 12300 edge pixels respectively. This is further demonstrated in Figure 4.8 which shows the productive sub-images for the oval test-image on which processing would take place. However, this only reduces the computational load in the system, without the communication load being affected (In fact the communication load is affected marginally since an empty sub-image results in an almost empty line-list data packet containing header information only. But for the relatively small message sizes in this application it can be regarded as insignificant).

The $\rho\theta sHT$ imposes a local Hough space for each sub-image which is released as soon as processing of the sub-image is completed. The ramifications of this are as follows. Since the Hough space is to be released, then the process of peak-detection must be performed immediately. This makes the measurement of the processing time for the separate stages of the $\rho\theta sHT$ rather difficult, hence only totals are provided here. However, it means that almost the complete process has been parallelised and performed locally. There is another clear advantage. The regular Hough transform requires a permanent Hough space. Consider the parallel implementation of the regular Hough transform on a control-driven array. The Hough space could be distributed amongst the processors, with the processors communicating to update

each others Hough spaces. In the demand-driven model, they could communicate with the one and only Hough space as held by the master. Either way, they introduce extra communications and therefore major delays in the processing. Clearly, this problem is non-existent in the $\rho\theta sHT$ implementation and so the execution times presented provide a very reliable comparison of control-driven and demand-driven network models.

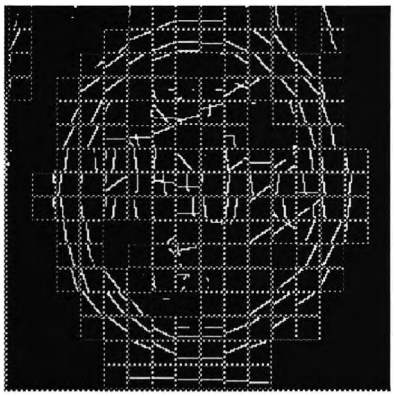


Figure 4.8: Diagram shows the sub-image areas of oval image with salient feature points on which operations take place.

The results display a superior performance by the demand-driven farm model. In this model, the stages involving the distribution, processing and collection of data are concurrent and overlapped, whereas they occur consecutively in the control-driven model thus reducing its efficiency (please also see Section 3.2.1).

Also, the performance of a single T800 should be noted as it manages to put in a remarkable performance, especially through the use of its 64-bit floating-point co-processor.

More discussion on the comparison of the control-driven and demand-driven models will be presented in Chapter 6, where it will be emphasised again that one should determine the use of one scheme or the other depending on the nature of the problem at hand. More immediately and appropriately, the demand-driven network will be re-visited in this chapter to enhance the performance of the label inspection process by employing 25MHz T800 processors as the farm slaves.

4.7 Final Processing and Inspection

Upon completion of the $\rho\theta sHT$ transformation and the return of the results, a list of all the lines in the image, representing their end-points, is available on the master transputer in either network. To follow, the master transputer starts the network with the image of the next label, while it performs, sequentially, the final stage of the inspection on the current label. This final stage requires a limited amount of processing and was found unsuitable for implementation on a network due to the cost/efficiency ratio. It performs fast enough on the floating-point T800 (master) transputer, and leaves the network free for work on the next label. The steps involved in the final processing stage for each label are now considered.

4.7.1 Rectangular Labels

The list of lines is inspected and, using the addresses of the lines' starting and ending co-ordinates, they are linked to form longer, more complete lines. This is in effect a simple run-of-the-mill neighbourhood-proximity phase, which connects together those lines with similar orientation and end-points in close neighbourhood of each other. For a rectangular label the processing is simplified since a good label should have produced lines in certain orientations (with a degree of tolerance) only, and lines with all other orientations can be discarded on first encounter. (In fact they are retained, but unlinked from the line list. This is reflected in Figure 4.6 where no line-linking is apparent on the few non-perpendicular and non-horizontal lines). Then, consider that any line A is under examination for connection to any other line B; then to allow them to be joined, the following criteria have to be met,

$$ABS(x_{ea} - x_{sb}) < T \quad AND \quad ABS(y_{ea} - y_{sb}) < T \tag{4.6}$$

$$ABS(tan^{-1}(\frac{y_{ea} - y_{sa}}{x_{ea} - x_{sa}}) - tan^{-1}(\frac{y_{eb} - y_{sb}}{x_{eb} - x_{sb}})) < G \tag{4.7}$$

where, (x_{sa}, y_{sa}) is starting pixel co-ordinates of line A, (x_{ea}, y_{ea}) is the ending pixel co-ordinates of line A, (x_{sb}, y_{sb}) is the starting pixel co-ordinates of line B, (x_{eb}, y_{eb}) is the ending pixel co-ordinates of line B, G is the gradient proximity tolerance, and T is the pixel neighbourhood tolerance.

That is, the end co-ordinates of line A must be in the close neighbourhood of the starting co-ordinates of line B, and both lines must have a

similar gradient. In effect only the B lines with starting co-ordinates in the pixel neighbourhood area of $x_{ea} \pm T, y_{ea} \pm T$ are inspected. When performing the above equations, provision is made for lines in opposite direction.

These operations provide a new, shorter, list of lines. This new list is further inspected, and the two longest horizontal and the two longest vertical lines falling within respective tolerances are selected. This last phase may need to be adapted for particular labels. These four lines are accepted as the boundary outline of the label if their starting and ending co-ordinates are found to satisfy the pre-determined thresholds (Figure 4.6). In addition, other features such as the vertices may be obtained. The boundary and the position of the label are then matched against a known model label, resulting in a fail or pass decision.

4.7.2 Oval Labels

Oval labels are approached in a slightly different manner. Most significantly, the line list is considerably larger due to the smaller sub-image size (which breaks up the curved lines into numerous small straight lines).

Initially, all horizontal and vertical lines are removed since, in principle, curved lines are being sought and no horizontal or vertical lines need be present. This also reduces the cost of the computation that follows. Next, using *a priori* knowledge, an aspect ratio is applied to the line list resulting in the image shown in the bottom-left of Figure 4.7. (However, for this example the horizontal and vertical lines are kept for a more aesthetic image.)

A neighbourhood-proximity inspection similar to that described above is performed to eliminate from the list any lines which have no other line in their neighbourhood. This is then followed by mapping the path of the normal to the centre of each of the lines (arc segments) remaining in the list, as shown in Figure 4.7 (Bottom Right). The image holding the normal paths is smoothed using a low pass filter and the three highest crossing-point peaks are considered as possible centres, as marked in Figure 4.7 (Bottom-Right and Top-Left). Please note that these centres are all three heavily concentrated about the centre and not quite resolved in the diagrams. Any one of these that, within a pre-set threshold, comes closest to the position of the centre of the model label, is accepted as the real centre, and thus the label may be passed or failed. Although this is not attempted in this implementation, one approach to tackle this problem could consist of fitting a model

template on each inspected label and shifting it about to find the best possible match against the extracted centre points. No faulty labels would give a close enough centre position following these calculations. The technique described is a Hough-type approach in image space.

Another method to locate the centre of any set of lines forming a circle is that suggested by Thomas [TC89]. The method estimates the centre and radius of a circular arc by minimising the least mean-square errors between the given set of data points and the curve. However, this method is very sensitive and gives a centre for any good length of an arc, which in the case of a partly torn label would be disastrous. Nevertheless, since it is a fast process, it could be used to initially estimate a centre for the label. Then, the address of the centre can be employed to eliminate all the lines in its local and outer neighbourhood before performing, on the reduced line list set, the adopted process of peak crossing points calculation as described above. Davies [Dav90] also describes enhancements to the Hough transform for faster and more accurate circle centre location.

The final inspection processing for both types of labels involves numerous floating-point operations especially for the centre finding algorithm on the oval label. This explains the reason for the lengthy execution time for the T414 host which, unlike the T800, does not have an on-chip floating-point unit. The timings are shown in *milliseconds* in Table 4.4.

Configuration	Rect. Label	Oval Label
T414-12MHz HOST	99	841
T800-20MHz HOST	33	91

Table 4.4: Results for final stage processing of both label types

The figures shown in the various tables are representative of tests carried out on three different test label images for each type of label. The inspection rate was found reliable when applied to a number of perfect and grossly faulty labels. The sensitivity of the fault detection has not been pursued. The advantages of the more sophisticated techniques used in this chapter for the inspection of the labels are that the positioning of the label in the image is now less critical, and also that the integrity of the whole label is examined. For example, by finding the lines describing the rectangular label, the whole borderline of the complete label has been checked for rips or folds.

4.8 Summary and Conclusions

In this chapter the model of the demand-driven farm as an approach to *geometric parallelism* was introduced and briefly investigated. It is because a potentially higher degree of communications is involved in the distribution, allocation, and collection of tasks that the processor farm model will only be beneficial as a solution to the class of vision problems with a high computation to communication ratio. Yet, since no inter-processor communication exists, the scheme is unlikely to be suitable for high-level vision. The approach is also highly unlikely to be implementable on a fine-grain SIMD array, such as the DAP or the CLIP4. Multi-processor computers with nodes capable of intensive local computations are the most likely candidates for this method, such as an appropriate configuration of transputers performing parallel tasks on different partitions of data.

Although adequate for the purposes of this thesis, the farming techniques presented can be transformed into more sophisticated schemes. For example, in this implementation, a data packet travels until it finds an idle processor (or in fact an empty buffer on the next processor). This means that during the initial load where every processor is completely idle, the processors further down the chain may not even receive any data packets if the computation requirements per packet are very small. It would be more efficient to distribute the load evenly by specifically addressing data packets to all the network processors during the initialisation stage. Then, during system run, the load would be balanced as before with the hungriest workers taking their pick as and when they are ready.

The use of farm parallelism is very much dependent on the nature of the problem at hand. This was examined in comparison to the control-driven network by considering a highly parallel approach to the (ρ, θ) Hough transform in which the total image is decomposed into smaller sub-images with local Hough spaces. Thus the problem is suitable for parallel implementation on most parallel architectures, be they the IUA, the PC WARP or transputer-based. This approach was named the $\rho\theta sHT$ transform, for which favourable results were obtained on T414-based array and farm networks.

Recently, through the availability of more T800-25MHz transputers, the performances of both the Sobel and the $\rho\theta sHT$ were re-examined for a longer, more powerful linear farm chain. For all the results that will follow, percentages showing the efficiency of the increasing farm size against a single processor farm will be shown.

No. of Processors	Sobel 16x16	Sobel 32x32	Sobel 64x64
1	910 (100%)	884 (100%)	888 (100%)
2	467 (97%)	452 (98%)	473 (94%)
3	335 (91%)	317 (93%)	366 (81%)
4	315 (72%)	271 (82%)	305 (73%)
5	315 (57%)	265 (67%)	285 (62%)
6	314 (48%)	268 (55%)	285 (52%)
7	314 (41%)	270 (47%)	286 (44%)
8	314 (36%)	270 (41%)	286 (39%)

Table 4.5: Sobel results using the Demand-Driven model on T800 processor network with 256x256 images

The results for the Sobel operator for varying work packet sizes in Table 4.5 indicate that the no real gains in efficiency can be expected after about the fourth or fifth processor in the farm. This may be attributed to the computation to communication ratio, such that the processors nearer the master are ready to accept more work, and they leave trailing transputers idle. Note that the 32x32 data packet gives the most cost-effective computation to communication ratio. These performances will be extended further in Chapter 6.

Table 4.6 and Figure 4.9 show the performance of the $\rho\theta sHT$ on T800-25MHz processors for the rectangular and oval label images using corresponding sub-image sizes of 64x64 and 16x16. Performance efficiency percentages are also shown in Table 4.6 which denote the speed-up against a single transputer farm implementation.

The higher number of feature points in the oval label image result in a higher computational load than in the rectangular label image. This combines with the extra communications due to the larger number of sub-images to cause a slower performance rate for the oval label when there are up to two processors in the farm network. As more processors are added, the computational load of the oval label spreads out more evenly, and a slightly better performance is achieved.

Although the results in Table 4.6 indicate that given more transputers an inspection rate of 1-2 labels per second could be achieved, a more efficient utilisation of larger numbers of processors for a demand-driven farm network would be to configure them in the forms shown in Figure 4.10. The bi-linear and tri-linear farms can be employed to increase

No. of Processors	Rectangular (64x64)		Oval (16x16)	
1	3989	(100%)	4121	(100%)
2	2058	(97%)	2117	(97%)
3	1510	(88%)	1480	(93%)
4	1191	(84%)	1162	(89%)
5	1123	(71%)	1068	(77%)
6	1017	(65%)	940	(73%)
7	927	(61%)	862	(68%)
8	862	(58%)	798	(65%)

Table 4.6: $\rho\theta sHT$ results using the Demand-Driven model on T800 processor networks for rectangular and oval images

throughput by a factor of two and three respectively. They will be used later to examine possible improvements to the implementation of the Canny edge detection process as part of the motion analysis system presented in Chapter 6.

Also, by employing two modules of the systems described here, the throughput of an overall label inspection system of 4-5 labels per second could be achieved which is an adequate rate for performance in a real-time situation.

One other factor directly influencing the $\rho\theta sHT$ execution time is the number of edge points, and a less busy image would greatly reduce the amount of processing involved as depicted in Figure 4.8. This is corroborated by [ES89], who have implemented the standard Hough transform on a customised-pipeline transputer network with eight T800 processors. For 1000 edge points in a 256x256 test image, they achieve a total processing time of 159 milliseconds for eight transputers. If the execution timings for the complete processing of rectangular and oval labels were to be scaled down, an average comparable timing for 1000 edge points of 87 milliseconds would be obtained. This shows an approximate improvement by 50% for this implementation. In their implementation the parameter space is divided amongst the transputers, and communications are necessary to update the appropriate area of the Hough space held by other transputers. This results in considerable programming headaches in comparison to the easily programmed $\rho\theta sHT$ implementation.

Most other parallel implementations of the Hough transform are on

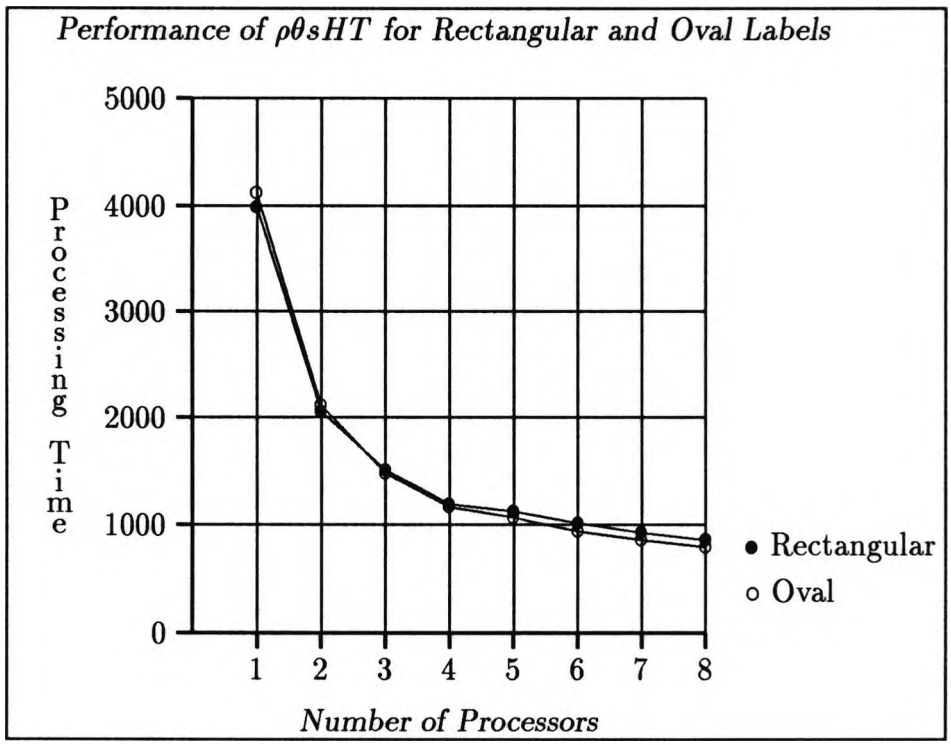


Figure 4.9: Processing times in milliseconds for corresponding number of processors.

SIMD arrays of processors. Rosenfeld et. al. [ROH88] compare several of many alternative implementations for simple 1-bit PE meshes, with techniques such as the assignment of each different θ to each PE to speed-up the processing. All the techniques naturally involve massive communication loads for such fine-grain machines. For just over 1600 pixels, Hough transformation timings of approximately 130, 920, and 180 milliseconds are reported for different algorithms on the GAPP and MPP processors. Scaling these down to 1000 edge points, approximate timings of 81, 575, and 113 milliseconds are obtained respectively. Bearing in mind that the programming of these algorithms is again very complex, only the best of their results is just slightly faster than the implementation presented here.

On the PC WARP systolic array (described in Section 2.3.7), each of the ten cells receives one-tenth of the Hough space, partitioned by θ [DEH89]. The whole image flows through the PC WARP cells, and each cell increments its own partition of the Hough space only. At the end, the partitions are returned to the external host. A time of 340 milliseconds is reported for a 180x512 image with almost 1000 edge

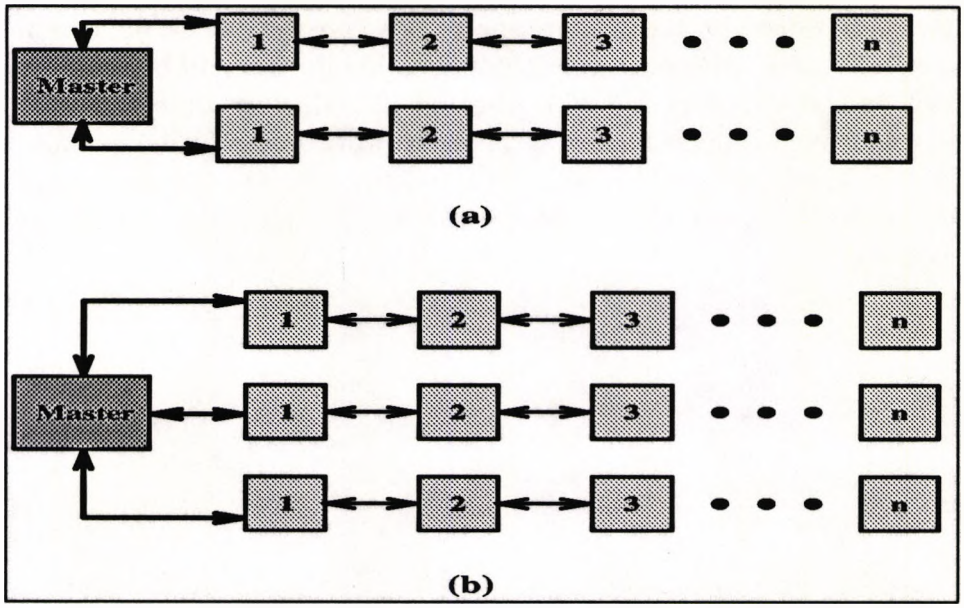


Figure 4.10: (a) Bi-linear farm network, (b) Tri-linear farm network

pixels, which partitions as 18x512 segments for each PC WARP cell. This again is a far cry from the 87 millisecond timing achieved here albeit that they use a slightly larger image.

Also presented in this chapter were some robust techniques in handling the results of the $\rho\theta sHT$ processing to inspect the integrity of rectangular and oval labels.

In summary, to find image line segments using the $\rho\theta sHT$ on transputers, the following factors (stated almost in order of contemplation) must be considered to determine the best optimised configuration:

- the size of the input image (I x I),
- the most commonly occuring length of lines under inspection,
- the size of the sub-image (S x S),
- the number of processors to be used,
- the type of *geometric parallelism* to be used.

The overall conclusion for this chapter is that in implementing a medium-level vision technique in different ways on two typical, SIMD transputer networks, the whole process has been pre-occupied with different

aspects of communication, as would be the case in any distributed system. However, for a transputer network to be a viable bet in a critically real-time situation, faster communication links would be required if not too much effort is to be directed at reducing communication costs and overheads (which may in itself prove to be futile and also result in unnecessary real costs). Still, for very computationally bound medium-level vision tasks, the transputer has the potential of real-time performance.

Chapter 5

Dynamic Scene Analysis

5.1 Introduction

Dynamic scene analysis or Computer Analysis of Time Varying Images (CATVI) is the process of analysing a sequence of images captured at various frame times with the purpose of making inferences about the structure and movement of the observed world. Major uses have been found in diverse applications such as,

- **Security Surveillance:** For example, [ERG91] present a robust system for correctly classifying the genuine and false detection of intruders, be they human or rabbit, in an outdoor scene.
- **Meteorology:** The application of cloud tracking, pollution and fire detection have been investigated from aerial and satellite image sequences [MA78, BSI90].
- **Transport:** Two application areas are traffic monitoring [AD90] and autonomously guided vehicles (AGV) [THKS88, SH88].
- **Medical:** Experimental studies have been carried out on cell motion and heart motion [YIT80].
- **Civil Engineering:** Study of displacement in large suspended structures, such as the Humber Bridge can be found in [STD90].
- **Industrial:** Dynamic robot vision [EWM87] and dynamic monitoring of industrial processes [Nag83] are increasingly popular areas of research.
- **Behavioural Studies:** Study of animal feeding patterns and training of athletes have been reported [Nag83].

Two principle uses of dynamic scene analysis are in the tracking and identification of discrete objects moving through the scene, and in estimating the ego-motion of the camera. Tracking discrete objects within a scene is more commonly performed with a stationary sensor, and this constraint can considerably simplify the task [ERG91]. The research described here forms part of the front end (low and intermediate level processing) to a vision system for use with an autonomous guided vehicle (AGV) that would be fitted with a camera. Thus, the aims of this investigation are twofold. The first is to provide visual input to the vehicle control system to aid location and navigation. This would place the rest of the vision system in a position to consider depth, ego-motion, and further analysis towards 3D scene understanding. The complete system is targeted towards operating in man-made environments, matching primitive image features with a geometric model-based representation of the world [EWM87, BO91]. The second aim is to continue the investigations of the earlier chapters on static image processing, and consider new approaches in dealing with a dynamic image processing problem using transputer-based configurations. Although the investigation here is not a fully-fledged high-level vision problem, nevertheless the solutions will be seen to tread the paths of MIMD processing.

Whilst the data rates from most image sensors, e.g. TV cameras, is very high (25 frames per second), it is not always necessary to process every frame, and a more important characteristic is the overall latency of the processing, which determines the time lag between acquiring the image and being able to act on the information it contains. Part of the requirement for vision comes from the need to augment other location-based information, e.g. odometry, which tends to accumulate errors over time, and results in increasing uncertainty in the position of the vehicle. Periodically, say once every few seconds, updating this information by using visual landmarks can minimise this uncertainty. By extracting a number of features from a sequence of images, correspondence between the features may be established to provide magnitude and direction of the flow of the detail in the scene. From this information, the ego-motion parameters of the camera motion, depth and information about scene structure, can be deduced at a later stage in the information processing cycle (Figure 1.1).

The rest of this chapter is divided as follows. Initially, a review of the different techniques used in the detection and measurement of motion is outlined. Later on in the chapter, a breakdown of the motion analysis approach adopted (in this research) for tackling the problem of correspondence will be presented. This will include a review and analysis of

related work. In the next chapter, the motion analysis approach will be used in the definition of a parallel computational model for tracking image primitives. This will be followed by a parallel implementation on a network of transputers. The emphasis throughout this chapter, and the next, will be on the algorithmically independent nature of the parallel computational model.

5.1.1 Some Definitions

Visual motion perception is a complex perceptual process. Much research has been directed towards understanding the psychology and reasoning used by biological vision systems in the analysis of motion to help establish specific goals for CATVI. For example, a housefly can track moving objects over a background identical in texture to the moving target, which thus renders the object indistinguishable in the absence of relative motion [RP80].

Ullman [Ull79] and Marr [Mar80] both have provided a historical analysis and new insight in the field. In fact, together they have introduced, discussed, and at least touched upon, many topics that have since become the subject of research by various scientists, and this review will repeatedly refer to their work. Ullman divides the problem of interpreting visual motion into two parts: the *correspondence problem* and the *3-D interpretation* problem. He defines *correspondence* as the process that identifies elements in different views as representing the same object at different times. Once the correspondence process is completed, the 2-D transformations in the object's appearance may be interpreted to achieve a decomposition of the changing scene into objects. Hence, their 3D structure and motion may be recovered. This is termed as *Structure from Motion.* (As explained earlier, the area of research covered here is a parallel approach to the correspondence problem. The topic of *Structure from Motion* will only be briefly reviewed later on in this chapter and returned to in passing in the concluding chapter of this thesis). One major derivative of the theory of structure from motion is that of the *rigidity assumption.* This states that, given the general truth that most things in the world are locally rigid, then,

> any set of elements undergoing a two-dimensional transformation which has a unique interpretation as a rigid body moving in space should be interpreted as such a body in motion.

In his subsequent work, Ullman [Ull81] points out that the study of motion perception in biological systems, and computational studies on

the interpretation of time-varying imagery, can complement each other by providing further insight into the principles and discoveries attained in each field.

Marr [Mar80] also viewed the psychological aspects of visual perception and conducted experiments in the analysis of the correspondence problem and structure from motion. The majority of his work was on the basis that the visual system uses information about direction alone to help analyse the visual field. He then outlined an algorithm for quickly detecting the sign of movement direction at the level of local edge segments. He applied this algorithm to segment independently moving surfaces. Marr also employed the term *optical flow* which he defined as,

> the use of the retinal velocity field induced by motion of the observer to infer the three-dimensional structure of the visible surfaces around him.

Some other definitions will be encountered later in this chapter.

5.2 Motion Detection and Measurement

The essential factor in motion analysis is time. It is by deducing the changes which occurred in two instants in time that one can begin to interpret the motion in a scene. Therefore, given two consecutive image arrays $F_1(x, y)$ and $F_2(x, y)$, motion may be described in terms of a displacement vector field $V(x, y, t)$ which is derived by establishing *correspondence* between points in the two image frames. The direction and magnitude of a vector in the vector field are the direction and physical displacement between corresponding points in the images, in other words Marr's *optical flow*. The computation of this vector is a measurement of visual motion. There will be more on optical flow later.

The detection and measurement of motion will take on a varying degree of complexity if the scene is subject to one or more of the following conditions,

- there is a moving background,
- there is camera movement,
- there are random illumination changes
- there are multiple objects,

- objects move at different orientations and at different directions,
- objects occlude each other.

In a real-world scene, all the above are applicable, thus making the task of machine vision a multi-faceted challenge. In considering the problem of ego-motion of a camera mounted on an AGV, all the above conditions will need to be taken into consideration in one way or another and will be discussed when applicable.

With some cross-over, most motion detection and measurement techniques can be divided into two major schemes, intensity-based schemes using low level image processing techniques, and token-based schemes using medium to high level image processing techniques [Ull79, Mar80, ADM81, TB81, Ros83].

5.2.1 Intensity-based Schemes

Some motion measurement techniques are based directly on the local intensity changes in the image. These may be subdivided into *differencing*, *correlation*, and *gradient* schemes.

Differencing Technique This technique is more commonly applied to a positionally static camera. Subtraction of corresponding pixels in two frames from a sequence will result in a point-by-point determination of changes in intensity. A subsequent threshold of the resultant image will give rise to significantly differing regions which may be classified as,

- regions composed of object pixels in the first frame and background pixels in the second,
- regions composed of background pixels in the first frame and object pixels in the second.
- regions composed of pixels from the same moving object but at substantially varying degree of intensity.

and defined as,

$$DIFF_{t,t+\delta t}(x,y) = \begin{cases} 255 & \text{if } |F(x,y,t) - F(x,y,t+\delta t)| > \Delta \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

given two images $F(x, y, t)$ and $F(x, y, t + \delta t)$ selected from a sequence at times t and $t + \delta t$, and Δ as a predetermined threshold, dependent on the noise and complexity of the scene.

For example, consider an object in a scene with a contrasting background, which moves from left to right from one frame to another. The difference picture will contain a region on the left due to the uncovering of background, and a region to the right due to the covering of background. These effects are demonstrated in Figure 5.1, where two images in a sequence are shown along with their thresholded difference image. There has been a voluminous amount of research and practical work using differencing techniques [JMA79, Jai81, HJ83, AD90], perhaps due to their computationally simple and fast nature which also allows for an easy introduction to problems in motion analysis. For example, Jain [Jai81] applied differencing techniques to identify and segment changing regions and to classify them using a decision tree.

Differencing techniques are highly sensitive, but they are not robust enough to be solely used for inferring occlusion or 3D-structure. Nevertheless, as part of more sophisticated systems, effective and practical use has been found in controlled environments, for example in security surveillance. Ellis et. al. [ERG91] use differencing techniques to observe changes within a fixed perimeter area around a prison, and Bernat and Rupel [BR90] have implemented a similar system, based on transputers, for tracking human motion across international borders. In the next chapter, a simple implementation of an object tracking system based on differencing will be described; this was conducted as part of an investigation and experimentation for laying the groundwork for a more comprehensive, parallel, motion analysis system.

Correlation Technique This is a direct matching technique carried out at the level of small image segments. A small region containing an object in the first image is matched with a sub-image in the second image of a sequence. Consider a scenario where a sequence of images of size $m \times n$ are available, and a sub-image S for a frame acquired at time t is known to contain an object. It is intended to determine the occurrence of the object in a later frame F captured at time $t + \delta t$, where,

$$S_t = s(x, y, t) \quad x_1 \leq x \leq x_2 \,, \;\; y_1 \leq y \leq y_2 \tag{5.2}$$

$$F_{t+\delta t} = f(x, y, t + \delta t) \quad 0 \leq x \leq m \,, \;\; 0 \leq y \leq n \tag{5.3}$$

The cross-correlation technique will then provide a measure of match, $C(x, y)$, between the initial sub-image and every sub-image formed at point (x, y) by applying,

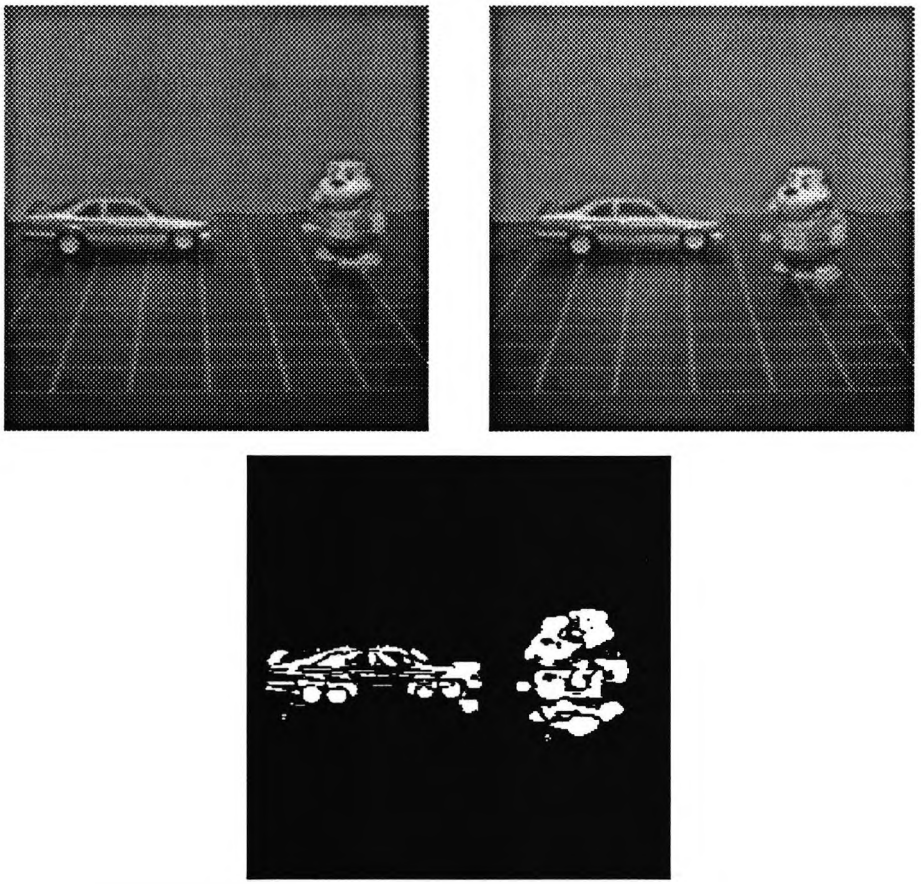


Figure 5.1: An example of Differencing: Two frames from a sequence and their difference image.

$$C(x,y) = \sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} S_t(i,j)\, F_{t+\delta t}(x+i, y+j) \tag{5.4}$$

Selective measurement is usually carried out within a reasonable area of (x_1, y_1) since the object may be assumed not to have moved by a large amount especially if δt is small. Information on the object from preceding frames could be used in determining this area. The values of $C(x,y)$ can be *normalised* to eliminate its sensitivity to areas of average high intensity, since such areas can result in false matches. Correlation techniques are described and applied in [ADM81, Ull81, STD90]. For example, Aggarwal et. al. [ADM81] use normalised cross-correlation as a similarity measure in establishing correspondence between image templates. An example of such a process is illustrated in Figure 5.2.

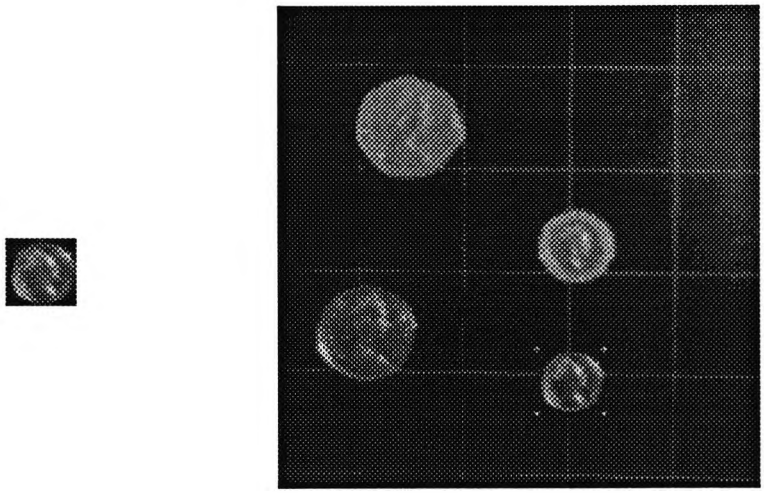


Figure 5.2: An example of Correlation: The template and the match in the later frame.

Note, that the area of image where the template has been found is marked.

Independently-moving multiple objects, changes in scale and illumination levels, and changing object viewpoint are some of the major factors that highlight the restricted capability of correlation techniques in performing matches from frame to frame. Also, correlation techniques are computationally expensive, but perhaps with the present availability of parallel processors, may begin to enjoy more attention.

Spatio-Temporal Gradient Techniques Gradient based schemes use the change in intensity at an image point over both time and space to estimate the rate of translation (i.e. movement) of the underlying surface. Given an image $F(x,y)$, then the temporal intensity change $\frac{dF}{dt}$ at position (x,y) may be deduced from,

$$\frac{dF}{dt} = \frac{dx}{dt}\frac{ds_x}{dt} + \frac{dy}{dt}\frac{ds_y}{dt} \tag{5.5}$$

where $\frac{dx}{dt}$ and $\frac{dy}{dt}$ are the components of the gradient, and $\frac{ds_x}{dt}$ and $\frac{ds_y}{dt}$ are the components of velocity in the x and y directions, associated with image point (x,y). This approach is the already encountered *optical flow* [Mar80, HS81], and is illustrated in Figure 5.3. Horn and Schunk [HS81] present an iterative, gradient-based technique to determine the optical flow in a coarsely quantised sequence of images. Caffario and Rocca [CR83] use gradient techniques to achieve TV inter-

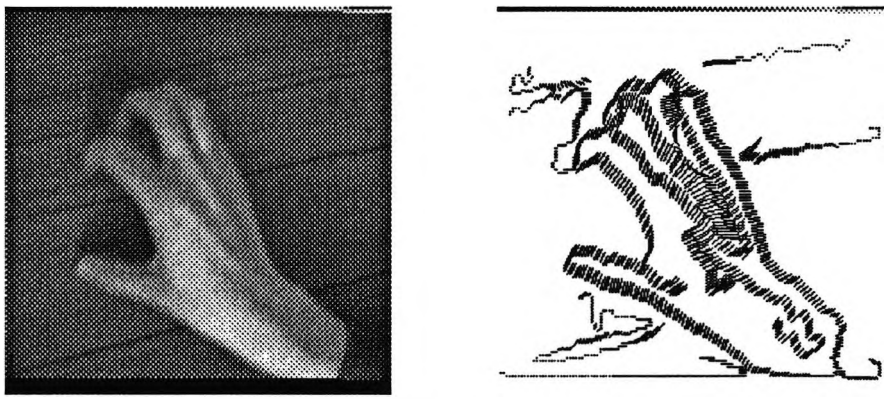


Figure 5.3: An example of Optical Flow (Courtsey of Dr. S. Gong of QMWC, London)

frame coding for eliminating redundancy and remote guidance. They also introduce into Equation 5.5 a noise error term with related covariance to account for various luminance differences. Fenema and Thompson [FT79] detect motion by using this technique to segment an image into regions corresponding to surfaces with distinct velocities. Optical flow has been used extensively to infer depth, 3D motion and structure from image sequences [Ull79, Mar80, Ros83, AN88]. For example, in estimating depth, two images of a scene captured fairly closely in time from a moving, monocular camera system could be used in place of two images from a stereo pair.

5.2.2 Token-based Schemes

In token-based schemes for measuring motion, elements in the image are identified, located and matched over time. These elements are generally referred to as features, primitives or tokens (all of these terms will be used interchangeably in this discourse). To allow a continuous perception of motion between a succession of tokens, the visual system has to establish a *correspondence* between them. In intensity-based schemes, while no matching is required in determining motion vectors, differencing and correlation provide a means for matching between two regions of two frames. The task of matching is somewhat more complicated between tokens.

Both biological and machine motion analysis systems share two general problems in token matching [Ull81]. The first is the degree of prepro-

cessing and the complexity of the participating tokens, which determine the level at which correspondence can be established. Matching can be achieved between simple tokens such as points or edge segments. Alternatively, complex tokens such as structured forms, or complete descriptions of recognised objects may be used, with *rigidity assumption* playing a large role in the analysis. Since a complex token would usually have a unique counterpart in the next image, their use can simplify the matching process. (However, a complete description of all the objects recognisable by the human visual system for machine vision is an impossible task for tackling everyday, real scenes.) In contrast, a simple token such as a small edge segment, has a number of candidate matches but carries two distinct advantages,

- It reduces the preprocessing requirements which are essential for faster machine motion perception,
- It allows the detection and tracking of arbitrary objects in (simple or) complex scenes. This is possible because correspondence between complex objects is established by matching the elementary tokens from which the objects are constructed.

It is relevant to note Marr's [Mar80] notion of the *primal sketch* which he defines as a set of basic units that are the first to be formed in the course of visual analysis, and serve as building-blocks for higher order constructs. Thus the immediate use of a set of primitives after the matching process, is their role in determining structure from motion, and to keep track of complex objects.

The second general problem concerns the role of token-based schemes in relation to intensity-based schemes in an integrated visual motion analysis system. Although fast, intensity-based schemes are highly sensitive to noise and result in a high degree of inaccuracy, e.g. in recovering the velocity field. Also, intensity-based schemes are ineffective when occlusion occurs. On the other hand, token-based schemes are more localised and accurate, and can track sharply localised tokens over long distances, but all at a higher processing cost in solving the correspondence problem.

Tracking of objects pre-supposes some earlier recognition stage, and is associated with discrete objects moving through the scene. For an AGV, this approach is inappropriate, since in addition to the disadvantages already stated, most of the elements of the scene are fixed, and only exhibit ego motion.

In this thesis, the interest lies in tracking primitive image features, such as edge segments, in the scene as a precursor to model matching. A token tracking scheme provides a method of maintaining frame-to-frame correspondence between features, accelerating the process of matching the image to the scene model and the detection of unexpected obstacles. The parallel computational model presented in the next chapter will be based on the tracking of tokens, irrespective of the nature of the token, i.e. be it point, corner, edge segment or whatever. This latter issue will be dealt with in the following section.

There follows a survey of past research on token-based tracking, including edge tracking. The idea of the flow model will be introduced as a precursor to a general edge tracking algorithm, which can be implemented in many ways. Two implementation approaches will be examined, both based on Kalman filtering which is a recursive filtering technique for estimating the state of a linear system. Standard Kalman Filtering is one approach and is discussed in Section 5.5. The discussion includes a brief outline of the second approach which is derived from the Kalman filter and is called the α, β filter. The process for matching corresponding tokens from one frame to another is usually an expensive process especially if a number of competing candidate matches are involved. This issue, and the selected approach for this research are reviewed in Section 5.6. Before the summary, the penultimate section briefly reviews the fundamental research issues in 3D (structure from) motion, and considers the role of token tracking in that field.

5.3 Feature Tracking for Motion Analysis

In Section 5.2 various issues of dynamic scene analysis were reviewed. This section concentrates on the most recent work in motion analysis which utilise tokens as features of interest for tracking.

The use of elementary image tokens in a feature-based approach to the solution of correspondence and tracking has been suggested by numerous authors, for example [Ull79, SJ87, THKS88, CSD88, HS88], and various image features such as points, corners, vertices and edges have been tracked to help in formulating structure from motion. It must be emphasised that these features are being viewed here as independent entities and not as a sub-description of an object. As mentioned previously, the interest lies in exercising the tracking on the feature specifically, and it is incidental, but presently irrelevant, that a certain

group of corners or edges may belong to one object. A higher level description and classification of the features would occur at a later stage in the processing for formulating structure from motion, and this is described briefly in Section 5.7.

Sethi and Jain [SJ87] use the idea of *smoothness of motion* to identify the same physical point in a sequence of frames as opposed to just two frames. They argue for this approach as a logical step in handling the correspondence problem since the motion of the object is not expected to change instantaneously, as inertia would prohibit it. They then assume a complete object to be a point and apply their proposed algorithm to the point. This is now briefly reviewed. Given *path coherence*, the problem is a qualitative decision on the best of a set of possible trajectories given m points in a sequence of n frames. This is illustrated in Figure 5.4.

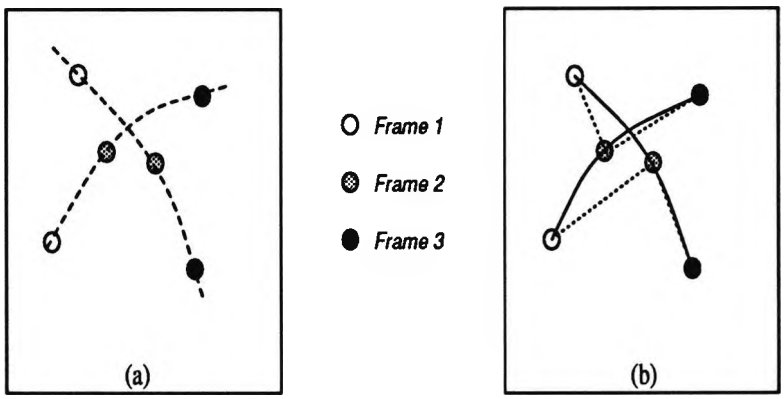


Figure 5.4: (a) shows the actual trajectories of two points over three frames, and (b) shows two possible trajectories for correspondence of points

For m points in n frames there exists a set of m^n trajectories. The authors present an algorithm to determine point trajectories over several frames, and continue by applying it to a real sequence from a world-saving scene in *Superman* where points on the head and belt of three running soldiers are tracked. The points were manually selected from the reel. In another example, they use differencing to perform a rough segmentation of their objects. The centroid of the object is then extracted as a coarse point representation of the object for feature point tracking. The ideas used here are rooted in the notion of *affinity* introduced by [Ull79], which defines that the correspondence between isolated token pairs is governed by a certain *built-in* similarity metric

(termed *affinity*). Griffin and Messimer [GM90] also examine the problem of tracking feature points, assuming constant velocity and rigid motion, using two imaging geometries: orthographic projection, as in an X-ray system, and perspective projection as in a camera system. The goal in each case is to find point paths by using a heuristic method to determine the best of a number of feasible paths, and is thus similar in nature to the work in [SJ87]. Possible applications of point tracking are traffic analysis and cell motion analysis.

Corners may be treated as points but are generally assigned more attributes such as the local grey level and edge strengths, which are used for matching against other corners in later frames. Shah and Jain [SJ84] describe a time-varying corner detector based on the AND operation between the cornerness and the temporal derivative. (Similar work applied to edges is described later). They compare different corner detectors such as the Kitchen-Rosenfeld [KR82] and the Zuniga-Haralick [ZH83], and use the latter to form their corner detector, which finds the cornerness at a point and uses the absolute value of difference at that point to approximate the temporal derivative. Harris and Stephens [HS88] have reported a combined corner and edge detector in their specific search for consistent feature extraction in noisy and natural imagery, such as one with trees or bushes. Such image regions usually yield different and fragmented edge segments on each image of a sequence. However, by using their combined detector, they obtained thin, continuous edges that terminate in corner regions. The edge terminators were then connected to corner pixels residing in corner regions to form connected edge-vertex graphs. This satisfied their aim in producing features that consist of edges meeting at corners, i.e. junctions or vertices. They also found many unconnected corners located in textural regions such as a bush. Given an image of grey values $F(x,y)$, their corner detector performed by searching for the corner response $R(x,y)$ for 8-connected local maxima and thresholding the resultant image, where,

$$R(x,y) = Det[M(x,y)] - c\,Tr[M(x,y)] \tag{5.6}$$

and,

$$M(x,y) = \begin{pmatrix} A(x,y) & C(x,y) \\ C(x,y) & B(x,y) \end{pmatrix} \tag{5.7}$$

$$A(x,y) = \sum_{uv} G_{uv} \frac{\delta F}{\delta x_{x+u,y+v}} \tag{5.8}$$

$$B(x,y) = \sum_{uv} G_{uv} \frac{\delta F}{\delta y_{x+u,y+v}} \tag{5.9}$$

$$C(x,y) = \sum_{uv} G_{uv} \frac{\delta F}{\delta x_{x+u,y+v}} \frac{\delta F}{\delta y_{x+u,y+v}} \tag{5.10}$$

G_{uv} is a Gaussian weighting function, and c is a constant to provide discrimination against high contrast pixel step edges.

This corner detector is an improvement on Moravec's corner detector [HS88],

- it is isotropic and determines the average changes in intensity that result from shifting the window in all directions by a small amount.
- it shifts a smooth, circular Gaussian window across the image rather than a noisy, binary and rectangular window.
- it reformulates the corner measure to make use of the variation in $R(x,y)$ with the direction of shift, to improve its response to edge detection.

The corner detector, for which they showed improvement over the Kitchen-Rosenfeld corner detector [KR82], was developed specifically to allow a more robust detection of features in essentially textural regions. Later, it was used by Stephens and Harris [SH88] as part of an edge-vertex feature extractor for use in tracking edge-vertices in the DROID vision system. The features are initially extracted for the first two images and matched using prior estimates of camera motion, and image-plane attributes of feature points. The visual matches are then used to estimate camera motion via an iterative ego-motion algorithm [HP87]. The features are then refined and tracked by performing an independent Kalman filter (described in Section 5.5) on each one.

A relatively early article on detection of motion in dynamic scenes by using edges is that of Haynes and Jain [HJ83]. They present an edge operator which includes both change and edge detection,

$$TimeVaryingEdginess = DIFF_{t+\delta t}(x,y) * S(x,y,t+\delta t) \tag{5.11}$$

where S is the Sobel 3x3 edge detector and $DIFF$ is the result of the subtraction of corresponding pixels of the centre point of the Sobel

mask (See Equation 5.1). The operator takes the static edginess over one frame and at each point multiplies this edginess by the difference between corresponding pixel gray values of the current frame and another frame in the sequence. Given a sufficient difference, this operator will detect low-contrast moving edges, and if the gradient strength is sufficient it will also detect low difference points that would arise, for example from slow motion. They obtain from this information a very approximate direction of motion by considering the sign of the gradient and of the difference for each edge point. Limitations of their approach are in the difficulty in selecting a suitable threshold, in the lack of response to horizontal edge movement, and most importantly in the detection of false points which are time-varying (e.g. due to a change in illumination or reflectance) but not necessarily moving.

Thomas et. al. [TLM+90] have presented a transputer-based implementation of a vision-guided vehicle which navigates by tracking road edges. The edge extraction and tracking techniques employed are similar to the special methods used in previous work in road-edge following. In the bootstrap mode, the strongest edges on the left and right of the image space are accepted as road edges by assuming that the vehicle is safely positioned in the expected direction of traffic flow. Having an accurate knowledge of camera position and orientation, the edges are projected onto the real-world horizontal plane using a perspective transformation. A model of the two road boundaries represented as a pair of concentric circular arcs lying in the ground plane is then built. Next, the model is used to guide the search for edges from frame to frame by predicting the position of the edge in the next frame on the basis of the known velocity of the vehicle and by extrapolating the model along the circular arcs from the most distant points previously detected. Details of the transputer implementation are unavailable. Another approach to road-edge following was presented by Thorpe et al [THKS88] which uses multi-class adaptive colour classification of pixels of the image to determine on-road and off-road regions. Each road and non-road class is given a confidence value from colour and texture measures, and a voting system using a 2D parameter space, similar to a Hough transform, is employed to select the best next road position. The parameter space is made up of parameter P as the column of the road's vanishing point and parameter θ as the road's angle from the vertical in the image.

The concept of edge tracking will now be continued by introducing the idea of a scene flow model.

The Scene Flow Model

Movement of a mobile robot or an AGV will introduce a considerable amount of vibration and noise into the measurement of the flow of motion, thus affecting the process of inferring depth and structure from motion in the subsequent stages. By adopting a *flow model*, consisting of a set of active tokens, a technique for minimising the degradation of measurement can be employed. Each model token can be represented by a vector of feature parameters composed of geometric and dynamic attributes. The geometric attributes may be position, neighbourhood characteristics, length, gradient, etc., depending on the type of token employed. Dynamic attributes associated with each token include velocity estimates. For each token extracted from an image in a sequence, referred to as an *observation*, a match in the flow model is found, and the flow model is updated with the active token. If no match is found, the token may be added as new to the flow model. Those tokens already in the flow model for which no match is found may either be deleted immediately, or tied over for a number of frames. Deletion may then take place if still no match is found. For observation tokens, only their geometric attributes would be known which are determined following the process that extracts them from each new image.

This is illustrated in Figure 5.5 which represents a more complete instantiation of Figure 1.2. It may be seen that by introducing a method for predicting and estimating the geometric and dynamic attributes of the flow model's active tokens, and employing a robust matching technique, taking into account measurement uncertainties, a continuously updated model of image flow can be maintained, which provides a suitable platform for 3D motion analysis. The information describing the causative tokens is made available to a 3D scene-model matcher, which can relate the tokens to those already in its database describing the scene. The efficacy of a token-based approach may be further emphasised here by pointing out that tokens from a scene are uncorrelated and estimation and measurement error in one will not affect or propagate in others.

The flow model can be regarded as a pre-processing stage for inferring depth and structure from camera ego-motion. In an attempt to provide a robust and accurate scene flow model, increasing use of optimal estimation techniques such as Kalman filtering is being made [CSD88, SH88, HS90, CSDP89, DF90].

Harris and Stennett [HS90] track known three-dimensional objects in simplified scenes using the α, β filter on control points on high contrast edges. These control points may be surface markings or profile edges.

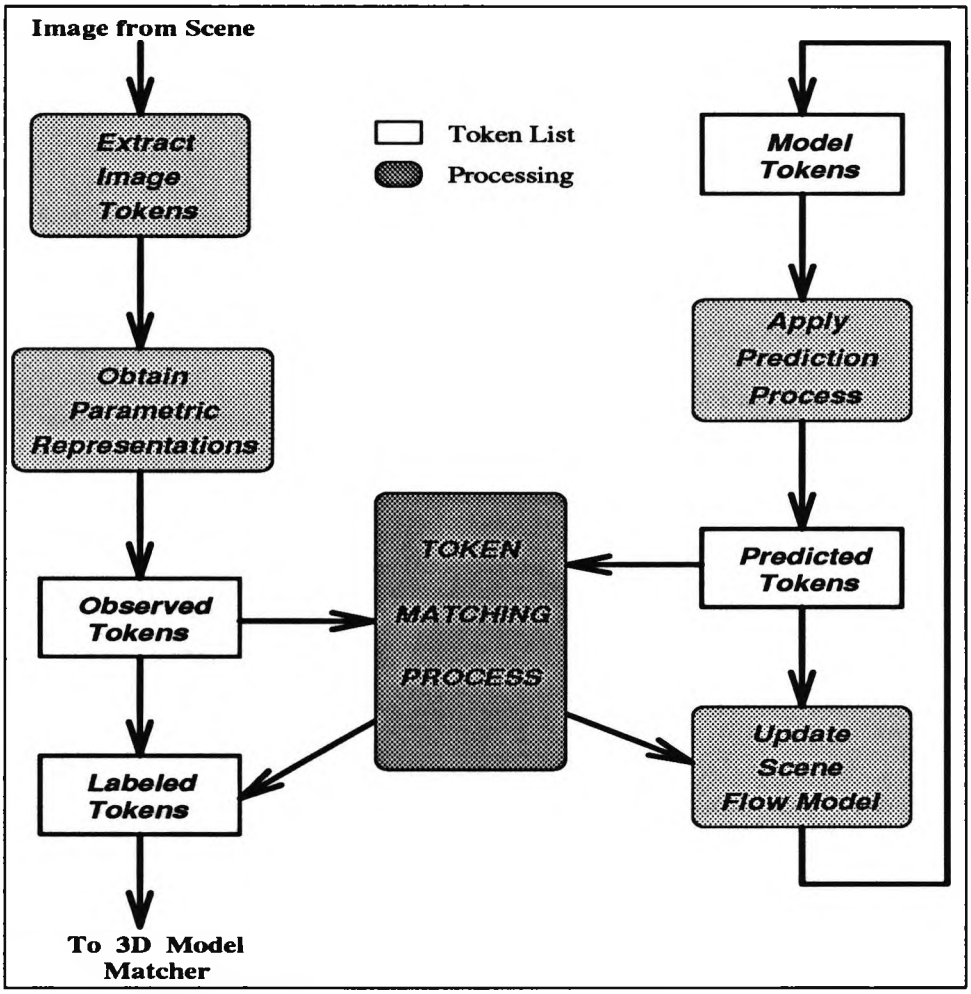


Figure 5.5: Prediction of the state of the flow model

Crowley et. al. [CSD88] and Chehikian et. al. [CSDP89] show that by assuming a flow model such as that above, it is possible to provide an elegant and reliable solution to the problem of image flow measurement and correspondence. They continue by applying a simplified form of the Kalman filter for tracking edges in software and hardware.

Also using edges as tokens, Deriche and Faugeras [DF90] experimented in edge tracking, using the the α, β filter. The α, β filter, which is an extended Kalman filter, will be employed as the main implementation algorithm for the parallel computational model presented in Chapter 6 of this thesis. In this implementation, edges will be used as tokens. This decision is fairly arbitrary, and the nature of the token is quite irrelevant to the parallel approach which is the major feature of this study. The

approach is designed to be applicable whatever the representation of the token, and use of edges as tokens allows its simplified demonstration. For example, edges and vertices together can carry more details for token matching. Each vertex can be classified by its degree, which is the number of edges attached to it. Its attributes would then consist of the directions of the attached edges, the length of the edges and the local vector image gradient [SH88]. It will be seen in the next chapter that grouping of such information into data structures related to each edge-vertex description will be a simple extension of the edge-only implementation. Other image primitives could also be added on, for example, corners. The parallel computational model will in fact only refer to tokens, where tokens can be any type of feature as necessary.

In the next section, two geometric representations of edge tokens will be examined. Use of the Kalman filter has already been mentioned but without much clarification. This will be remedied with a detailed examination and derivation of the Kalman filter equations for the motion model. The matching process will then be described, completing the various stages of the algorithm. The overall picture will be pieced together in the summary for this chapter.

5.4 Token Parameter Representation

It was shown that the dynamic scene will be described by a set of parametric primitives, namely edges, which will be based on observations about the scene. In selecting an edge token representation, the geometric parameters for line segments must be examined. Some of these are illustrated in Figure 5.6 and Table 5.1.

Edges in real 3D structures are expected to remain fairly consistent in frame sequences, providing good feature parameters as well as future connectivity information. Most simply, edge tokens may be described by their end-points. However, on their own, the end-points of a line are an inadequate representation, since digitisation effects[1] and the extraction process can result in fragmented edge segments in each image of a sequence, and both these factors deteriorate the chances of successful matching and Kalman filtering. It is therefore necessary to observe some measure of uncertainty and tolerance when considering any representation of a (digitised) line.

[1]Pixels are affected by noise both at sensing and amplification of the initial electronic signal, and by quantisation noise when registered in a frame-store. One major source of error is variation in illumination.

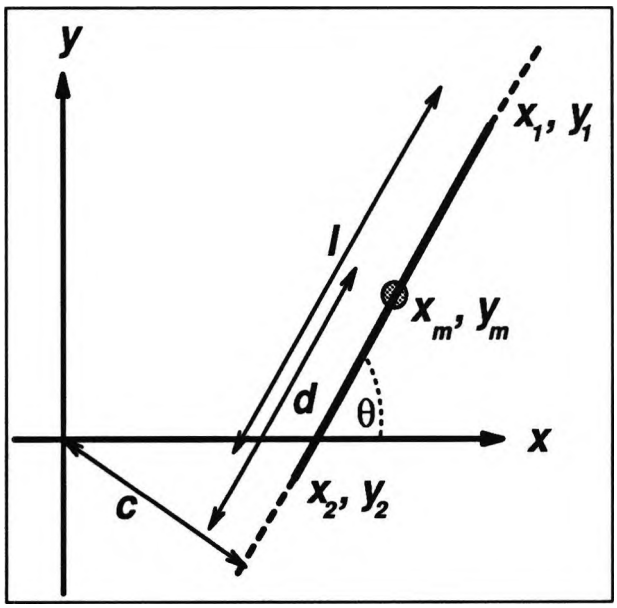


Figure 5.6: Parametric features of a line segment

Crowley et. al. [CSD88] and Chehikian et. al. [CSDP89] use feature vector $V = [c, d, \theta, \frac{l}{2}]$ to represent line segments. Considering the uncertainties associated with the edge extraction process, the precision of the perpendicular distance c from the origin will be to the order of very few pixels. However, the shorter the edge segment length, the less reliable is the orientation θ. Thus θ is highly dependent on l.

Deriche and Faugeras [DF90] compare this representation with the mid-point representation feature vector $M = [x_m, y_m, \theta, l]$ and following an analysis of the $[c, d, \theta, \frac{l}{2}]$ and $[x_m, y_m, \theta, l]$ covariances show that the $[x_m, y_m, \theta, l]$ representation is more appropriate by noting that,

- $[c, d, \theta, \frac{l}{2}]$ feature vector representation leads to a covariance matrix that depends strongly on the position of the associated line segments in the image through the parameters c and d. Thus, the uncertainty on the $[c, d]$ parameters for two line segments with the same length and orientation will be completely different depending on their position in the image. This situation will not arise for the mid-point representation, since the uncertainty of the mid-point (x_m, y_m) depends only on the uncertainty associated with the endpoints of the line segment,

- $[x_m, y_m, \theta, l]$ feature vector parameters are decorrelated allowing for more efficient post-processing and tracking.

Line Endpoints	x_1, y_1 and x_2, y_2
Midpoint	x_m, y_m where, $x_m = \frac{x_1+x_2}{2}$ and $y_m = \frac{y_1+y_2}{2}$
Orientation	$\theta = tan^{-1}(\frac{y_2-y_1}{x_2-x_1})$
Length of line	$l = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$
Perpendicular distance from line segment to the origin	$c = (\frac{x_2*y_1-x_1*y_2}{l})$
Distance along the line from perpendicular intercept of the origin to midpoint of the segment	$d = \frac{(x_2-x_1)*(x_2+x_1)+(y_2-y_1)*(y_2+y_1)}{2*l}$

Table 5.1: Description of parameters used in two representations of a line segment

By adopting Deriche and Faugeras's conclusions, each edge segment will be tracked by implementing four Kalman filters for it, one for each of its x_m, y_m, θ and l parameters.

5.5 Optimal Estimation

The problem of estimation may be defined as the process of extracting information from measured data. To estimate the minimum error state of a system, an optimal estimator may be applied which takes into account [Gel74],

- information describing the initial condition of the system,
- a knowledge of system model and measurement dynamics,
- assumed statistics of system noise,
- assumed statistics of measurement errors.

5.5. OPTIMAL ESTIMATION

Problems such as determining the path of earth satellites, or the state of the *current* in a circuit as the voltage is affected by instrument noise, can be cited as examples where optimal estimation techniques may be applied. The problem of estimation may be categorised into three areas of interest: consideration of the past state of a system is *smoothing*, the estimation of its present state is *filtering*, and the estimation of its future is *predicting*. The primary concern here is with the issues of *filtering*.

Modern elementary estimation methods have undergone a drastic simplification in the last thirty years. To aid the examination of modern estimation theory, mathematical tools such as matrix algebra, basic probability theory, and calculus are considered adequate[Lie67]. A notable contributor to the fusion of the various topics involved was R. E. Kalman who produced an optimal filtering technique for estimating the state of a linear system with results in the time domain rather than the frequency domain[Kal60]. Figure 5.7 represents an example where the capabilities and limitations of optimal estimators may be observed.

Thus, Kalman filtering [Kal60] can be defined as a statistical approach to modeling and estimating a time-varying state vector from noisy measurements, and within this context, may be used as a recursive estimation scheme designed to match the dynamic system model of the moving token, the statistics of the error between the model and reality, and the uncertainty associated with the measurements.

The requirement here is to consider the application of Kalman filters to the tracking of edge segments in order to maintain a frame-to-frame correspondence between features and aid the process of matching the image to the scene model. During tracking, there is uncertainty of measurement as well as inaccuracy of the model, and these may be represented as process noise. Hence, the state of the token in the flow model may be predicted using the Kalman filter. The general equations for the model of the system dynamics and the measurement model are as follows,

$$X_{t+1} = \Phi_{t+1,t} X_t + \mu_t \tag{5.12}$$

$$Y_t = C_t X_t + \omega_t \tag{5.13}$$

where,

X is the vector of state variables,
Y is the vector of measurements,

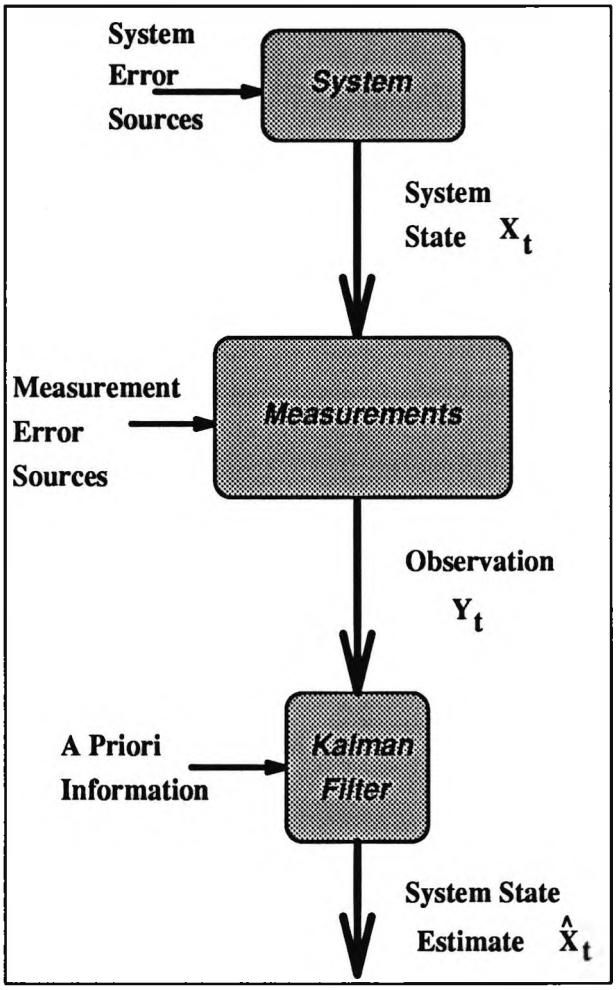


Figure 5.7: Estimating the state of a linear system

Φ is the state transition matrix,
C is the output matrix,
μ is the zero-mean Gaussian noise sequence of covariance Ψ,
ω is the zero-mean Gaussian noise sequence of covariance Ω.

Applying to Equations of Motion

When a given line segment moves, each feature parameter representing the line follows a trajectory in the one dimensional space. The kinematics of the motion of the line segment consists of the trajectory, the velocity and the acceleration of the feature parameters. Therefore, four

independent Kalman filters are applied to each parameter in feature vector $M = [x_m, y_m, \theta, l]$, thus necessitating the introduction of the following state vectors,

$$M_1 = \begin{pmatrix} x_m \\ \dot{x}_m \\ \ddot{x}_m \end{pmatrix}, M_2 = \begin{pmatrix} y_m \\ \dot{y}_m \\ \ddot{y}_m \end{pmatrix}, M_3 = \begin{pmatrix} \theta \\ \dot{\theta} \\ \ddot{\theta} \end{pmatrix}, M_4 = \begin{pmatrix} l \\ \dot{l} \\ \ddot{l} \end{pmatrix}$$

For each edge segment feature, the state vectors represent the position of the parameter, its velocity and its acceleration.

Given that the derivative of position is velocity and that of velocity is acceleration, then the equations of motion for uniformly accelerated movement are,

$$\ddot{x}_t = a \tag{5.14}$$

$$\dot{x}_t = at + v_0 \tag{5.15}$$

$$x_t = \frac{1}{2}at^2 + v_0 t + c \tag{5.16}$$

Thus at time $t + 1$,

$$\begin{aligned} x_{t+1} &= \frac{1}{2}a(t+1)^2 + v_0(t+1) + c \\ &= x_t + \dot{x}_t + \frac{\ddot{x}_t}{2} + c \qquad (5.17) \\ \dot{x}_{t+1} &= a(t+1) + v_0 \\ &= \dot{x}_t + \ddot{x}_t \qquad (5.18) \end{aligned}$$

Therefore, the equations of motion describing the system dynamics and measurement model can be equated as follows,

$$X_{t+1} = \begin{pmatrix} x_{t+1} \\ \dot{x}_{t+1} \\ \ddot{x}_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \frac{1}{2} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_t \\ \dot{x}_t \\ \ddot{x}_t \end{pmatrix} + \mu_t \tag{5.19}$$

and

$$Y_t = \begin{pmatrix} y_t \\ \dot{y}_t \\ \ddot{y}_t \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_t \\ \dot{x}_t \\ \ddot{x}_t \end{pmatrix} + \omega_t \tag{5.20}$$

where,

$\Phi_t = \begin{pmatrix} 1 & 1 & \frac{1}{2} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ is the state transition matrix evolving the position, velocity and acceleration from one time sample to another,

$C_t = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$ is the output matrix in this application,

$\Psi_t = E(\mu_t \mu_t^T) = \begin{pmatrix} \sigma_p^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_a^2 \end{pmatrix}$ is the zero-mean covariance matrix.

In this application, the value Y_t of the measurement model is a combination of the measured position x from the matching process, and ω_t which is the measurement uncertainty error σ_m^2 on position with zero-mean Gaussian noise sequence of covariance Ω_t.

The model derived above is a classic example of uncorrelated system and measurement noise processes. A typical source of noise common to all aspects of the model is the vibration of the camera mounted on an AGV. In their study, Deriche and Faugeras [DF90] simply ignore this due to their lack of knowledge about it, and assume it to be zero. Here, this lack of exact knowledge is felt too, but it is proposed that the presence of this noise is inherent in the model and measurement noise processes, and thus is already dealt with, only not in an independent sense.

5.5.1 Prediction Algorithm (or Solution of Kalman Equations)

The prediction algorithm is *bootstrapped* into action by setting the estimated position of the state vector and its associated covariance matrix using initially measured and estimated data,

$$\hat{X}_0 = E(X_0) = Y_0 \quad \textit{and} \quad P_0 = Var(X_0) = \Psi_0 \tag{5.21}$$

In the ensuing *run* stage, the following steps are dealt with sequentially,

- Compute the Kalman gain matrix, G_t, which indicates the weight to be attached to each new measurement,

$$G_t = \Phi_t \, P_t \, C_t^T \, (C_t \, P_t \, C_t^T \, + \, \Omega_t)^{-1} \tag{5.22}$$

  It is easily seen that the influence of the measurement in determining the state estimate may be altered by using a small or large

uncertainty Ω_t to achieve a large or small weighting G_t respectively.

- Now compute the prediction estimates for the state vector using the Kalman gain,

$$\hat{X}_{t+1} = \Phi_t \hat{X}_t + G_t (Y_t - C_t \hat{X}_t) \tag{5.23}$$

- Compute the covariance prediction for the state vector,

$$P_{t+1} = (\Phi_t - G_t C_t) P_t \Phi_t^T + \Psi_t \tag{5.24}$$

Note that this is a simplified form which disregards the correlation between the system and measurement noise processes as previously mentioned above.

- Compute the covariance prediction for the measurement vector,

$$S_{t+1} = (C_t P_{t+1} C_t^T) + \Omega_t \tag{5.25}$$

This covariance is then used to determine the *search area* for the matching process which follows.

5.5.2 Error Modelling

Due to the inaccuracy of the system model, *divergence* [Gel74] or *growing uncertainty* will occur in the Kalman filter. The assumed constant velocity and acceleration model is essentially correct only *locally*, e.g. when considered in small time steps (which is why in this work the velocity is continuously tracked, but uniform acceleration is assumed). Through appropriate error modelling, the system model constraints may be made more flexible. Hence, two techniques are employed for reducing the divergence problem.

Firstly, divergence is minimised through the addition of process noise as already shown in Equation 5.13 using the term μ_t with covariance Ψ_t, where,

$$\Psi_t = E(\mu_t \mu_t^T) = \begin{pmatrix} \sigma_p^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_a^2 \end{pmatrix} \tag{5.26}$$

The choice for the values of σ_p^2, σ_v^2 and σ_a^2 is largely heuristic and for this work roughly estimated values were used and refined by practical

experiments. The choice on the variance on the position σ_m^2 is determined from the amount of noise expected from the digitisation and edge extraction stages. For example, this can be determined by repeated edge segmentation of a static scene and measurement of the positional variance under normal conditions.

Secondly, old data can be regarded as irrelevant (since the model is claimed to be local in time and not in space) and thus eliminated from the system. This may be achieved by weighting old data according to the time of their occurrence [Gel74]. This can be accomplished by increasing the covariance matrix for past measurements by using an scalar factor λ greater than or equal to 1 in Equation 5.24 to give,

$$P_{t+1} = \lambda(\Phi_t - G_t C_t) P_t \Phi_t^T + \Psi_t \tag{5.27}$$

The discussion above outlines the first algorithm that is used in this study. In addition, a second algorithm which is an extension to the Kalman filter but of a simpler design, will also be discussed briefly now. This is intended to show the flexibility of the parallel computational model, proposed later, in embracing different algorithms for easy implementation.

The second algorithm uses the limiting Kalman filtering equations, better known as the near optimal α, β tracker, again based on an assumed model of the trajectory with constant velocity and acceleration. Following the work by Deriche and Faugeras [DF90] and Gelb [Gel74] the Ricatti differential equation is derived from the standard Kalman filter by adding process noise to the covariance prediction for the state vector; this steady state equation is then solved and the Kalman gain, composed of two positive scalars less than one, is deduced. By assigning constant values to the Kalman gain, $G = (\alpha\ \beta)^T$, the application dealing with a constant velocity model will yield the following decoupled equations,

$$x_t = -(\alpha+\beta-2)*x_{t-1} - (1-\alpha)*x_{t-2} + \alpha*v_t + (-\alpha+\beta)*v_{t-1} \tag{5.28}$$

$$\dot{x}_t = -(\alpha+\beta-2)*\dot{x}_{t-1} - (1-\alpha)*\dot{x}_{t-2} + \beta*(v_t - v_{t-1}) \tag{5.29}$$

and the covariance matrix P,

$$P = \frac{\sigma_m^2}{1-\alpha} \begin{pmatrix} \alpha & \beta \\ \beta & \beta*(\alpha+\beta) \end{pmatrix} \tag{5.30}$$

The α, β filter has the following advantages,

- it is simple to implement
- the coefficients of the recursive equations are determined once only at compile time
- the position and velocity components are completely decoupled
- only the state vectors from 2 previous time frames need be stored

This decoupled implementation allows for the recovery of the components of the state vector, i.e. position and velocity, *in a completely separate way* using the two Kalman filters in Equations 5.28 and 5.29. The reader is reminded that this is applicable to each of the four geometric feature parameters of the token. Also to be noted is that the α, β algorithm and the standard Kalman filter solutions are expected to yield the same results, only they use approaches with different computational and algorithmic complexity.

Finally, the α, β equations show that the representation of active tokens in the flow model will be continuously refined by their geometric and dynamic feature attributes as discussed earlier in the comments accompanying Figure 5.5.

5.6 Token Matching

A common operation in perception is the comparison of primitives at different points in time, to determine if they represent the same physical quantity. In this section, the process of matching tokens (or primitives) between the observed tokens and flow model tokens is discussed (Figure 5.5). Indeed, this is a verification of the measure of affinity as stated in Ullman's [Ull79] general correspondence principles. The results of the Kalman filtering are used to predict an image region within which the next expected instance of the edge being tracked will lie (Figure 5.8). Given that a certain number of observed tokens will be found to exist in such an area, a matching process must be employed to establish which token best corresponds with the edge being tracked.

Now, perceived quantities are never exact measurements and constant from one scene to another, and thus some measure of tolerance and uncertainty must be introduced. Therefore, the token matching process employs the Mahalanobis distance which is essentially a feature distance squared, normalised by variance to represent the uncertainty,

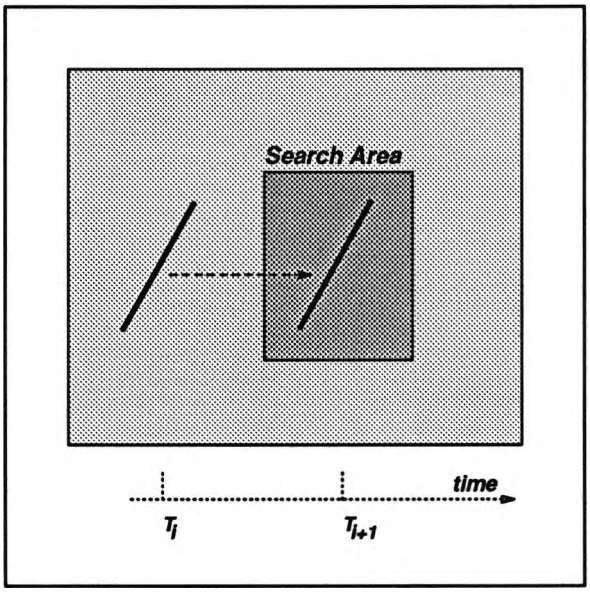


Figure 5.8: An example showing the estimated search area for the next instance of an edge segment

$$d_{\chi^2} = (F_f - F_e)^T(\Pi + \Gamma)^{-1}(F_f - F_e) \tag{5.31}$$

where F_f is the feature vector of N components with a covariance matrix Π and F_e is the feature vector found in the estimated search area with a covariance matrix Γ. Note that where no correlation exists between vectors F_f and F_e then the covariance matrix of the vector difference, $F_f - F_e$, is the sum of Π and Γ. The distance associated with each component has a χ^2 distribution with 1 degree of freedom in this application.

Hence, the Mahalanobis distance can be used to determine a similarity measure for the comparison of edge tokens represented by feature vector $[x_m, y_m, \theta, l]$. For each component of each observed feature vector in the search area, the Mahalanobis distance is calculated using the components of the flow model feature vector that is under scrutiny. Given that the distance for each component is less than the Mahalanobis threshold (3.84 for a successful matching probability of 95%), the observed token can be regarded as a candidate match and a sum of the distances of the components is determined as the cost. The most promising match is then selected to be the token with the least overall cost. A pseudo-OCCAM outline of this process is provided in Figure 5.9. The code is ideally parallelised: for the matching procedure

for a token executing on a single transputer, it is more efficient to run the algorithm sequentially, since less processes exist for the scheduler to cope with. However, ideally, the independent parts could be run in parallel.

```
—Comments:
—Each parameter will require its own covariance representation.
—e and f stand for estimated and flow model respectively.
Match.Found := -1
MINIMUM.COST.SO.FAR = VERY.EXPENSIVE
PAR i = 0 FOR (Every.Token.in.Search.Region)
  SEQ
    PAR
      cost[i][0] := MAHALANOBIS(x_me[i], Γ_ex, x_mf, Π_fx)
      cost[i][1] := MAHALANOBIS(y_me[i], Γ_ey, y_mf, Π_fy)
      cost[i][2] := MAHALANOBIS(θ_e[i], Γ_eθ, θ_f, Π_fθ)
      cost[i][3] := MAHALANOBIS(l_e[i], Γ_el, l_f, Π_fl)
    Total.Cost[i] := cost[i][0]+cost[i][1]+cost[i][2]+cost[i][3]
SEQ i = 0 FOR (Every.Token.in.Search.Region)
  IF
    Total.Cost[i] < MINIMUM.COST.SO.FAR
      SEQ
        Match.Found := i
        MINIMUM.COST.SO.FAR := Total.Cost[i]
    TRUE
      SKIP
```

Figure 5.9: Pseudo-OCCAM code outlining an ideal implementation for the token matching process using the Mahalanobis distance function (not shown).

Another implementation could be similar to the pseudo-code illustrated in Figure 5.10. This algorithm allows the token under inspection to be abandoned immediately, upon encountering the first (of the four) Mahalanobis distances that falls foul of the threshold test. (Note, this avoids extra computation for most cases but leads to a worst case for those matches that satisfy all the conditions and for which the cost function has to be calculated effectively twice. A more elaborate and sequential code can overcome this problem by calculating, saving and then examining each component cost before going ahead with the next component cost.) For a very busy scene with many tokens, this algorithm would perform best.

The following are the equations used for calculating the Mahalanobis

```
—Comments:
—Each parameter will require its own covariance representation.
—e and m stand for estimated and flow model respectively.
VAL MT IS 3.84 :
Match.Found := -1
MINIMUM.COST.SO.FAR = VERY.EXPENSIVE
PAR i = 0 FOR (Every.Token.in.Search.Region)
  SEQ
    IF
      (MAHALANOBIS(x_me[i], Γ_ex, x_mf, Π_fx) ≤ MT) AND
      (MAHALANOBIS(y_me[i], Γ_ey, y_mf, Π_fy) ≤ MT) AND
      (MAHALANOBIS(θ_e[i], Γ_eθ, θ_f, Π_fθ) ≤ MT) AND
      (MAHALANOBIS(l_e[i], Γ_el, l_f, Π_fl) ≤ MT)
        SEQ
          PAR
            cost[i][0] := MAHALANOBIS(x_me[i], Γ_ex, x_mf, Π_fx)
            cost[i][1] := MAHALANOBIS(y_me[i], Γ_ey, y_mf, Π_fy)
            cost[i][2] := MAHALANOBIS(θ_e[i], Γ_eθ, θ_f, Π_fθ)
            cost[i][3] := MAHALANOBIS(l_e[i], Γ_el, l_f, Π_fl)
          Total.Cost[i] := cost[i][0] +
                           cost[i][1] + cost[i][2]i + cost[i][3]
      TRUE
        SKIP
SEQ i = 0 FOR (Every.Token.in.Search.Region)
  IF
    Total.Cost[i] < MINIMUM.COST.SO.FAR
      SEQ
        Match.Found := i
        MINIMUM.COST.SO.FAR := Total.Cost[i]
    TRUE
      SKIP
```

Figure 5.10: Pseudo-OCCAM code outlining a more efficient, but still ideal, implementation of the token matching process using the Mahalanobis distance function (not shown).

distance for each observed token parameter in the projected search area,

$$d_{\chi^2}(x_m) = \frac{(x_{me} - x_{mf})^2}{P_{11} + \frac{\sigma^2}{2}} \tag{5.32}$$

$$d_{\chi^2}(y_m) = \frac{(y_{me} - y_{mf})^2}{P_{11} + \frac{\sigma^2}{2}} \tag{5.33}$$

$$d_{\chi^2}(\theta) = \frac{(\theta_e - \theta_f)^2}{P_{11} + \frac{2\sigma^2}{l^2}} \tag{5.34}$$

$$d_{\chi^2}(l) = \frac{(l_e - l_f)^2}{P_{11} + 2\sigma^2} \tag{5.35}$$

given that the covariance matrix for feature vector $[x_m, y_m, \theta, l]$ is,

$$\Lambda_{[x_m,y_m,\theta,l]} = \begin{pmatrix} \frac{\sigma^2}{2} & 0 & 0 & 0 \\ 0 & \frac{\sigma^2}{2} & 0 & 0 \\ 0 & 0 & \frac{2\sigma^2}{l^2} & 0 \\ 0 & 0 & 0 & 2\sigma^2 \end{pmatrix} \tag{5.36}$$

and P_{11} is obtained from P in Equation 5.30. These distances, bounded by the threshold, define the search area (although in practice, only those associated with the mid-point are chosen in this implementation using a large uncertainty).

This section concludes the final topic related to the token matching algorithm. The full algorithm will be summarised in Section 5.8 after some words about the general use of token tracking in 3D motion analysis.

5.7 Towards Structure from Motion

Analysis of corresponding image features have led to a considerable and varied amount of research on *structure from motion* [Ros83, Low87, HP87, SH88, AN88, THKS88, SBC+89, HS90, WH90]. 3D perception is a prerequisite for autonomous vehicle navigation. Road following systems based on 2D image-plane techniques have been successful [THKS88], but in less constrained and structured environments, tasks of greater complexity, such as obstacle avoidance, are necessary and require a 3D approach.

Many techniques for understanding 3D scene geometry have been applied in the past. The trade-offs have been complexity of implementation and detail of scene representation. With respect to the latter,

token tracking provides a robust and compact technique which accrues easy-to-update information as further knowledge is extracted from subsequent images. This section is intended to highlight the uses of token tracking in its own aftermath. The discussion will be brief since the nature of this topic is beyond the scope of this thesis.

When a camera is moving relative to a three-dimensional scene, a shift of position and velocity can be associated, in each frame sequence, to each token using the information provided from the token tracking stage. Knowing the physical nature of these tokens, allows the determination of their distance from the camera. By forming structures from these tokens, a higher level of understanding about the scene is attained which can subsequently be employed to aid the navigation of a vehicle about the scene.

Let it be assumed then, that matching has been performed and correspondences are available for images obtained either from a stereo pair of cameras or through successive frames in a sequence. The initial objective in 3D scene interpretation is then to obtain a *depth* map of the scene. Human perception of depth is remarkably efficient without the need for high-level information, e.g. depth data can be derived from random dot stereograms [Ull81]. In computer vision, depth may be recovered by examining the relative disparity of corresponding features (i.e. tokens in this particular case), which arises due to the shift between the viewer and the feature being viewed. This is best illustrated by holding a finger a short distance in front of one's face and observing the shift when the finger is viewed by each eye alternately. The further away the finger is held, the less the shift in position. Therefore, to achieve the same perception in computer vision, stereo images, or images obtained from camera motion, can be analysed to determine depth and structure from motion, given that the motion of the sensor is approximately known.

Camera ego-motion is the major cue in 3D scene interpretation for motion of an AGV and has been determined in various ways by deriving information from corresponding tokens in serial images. An example of such an approach is the work on the DROID project at Plessey Roke Manor Research. Matched point information from a sequence of images has been manipulated by Harris and Pike [HP87] to estimate the motion of a camera moving through a static environment. A bootstrap process determines a 3D instantiation of a scene using the first two frames of the sequence to estimate the depth of these points. In the run mode, point features are matched using image-plane proximity and attribute similarity, followed by an estimation of the relative

camera motion using these matches. The motion is represented by a 6-dimensional quantity, describing both vector translations and rotations of the camera. Kalman filtering procedures are then applied to the camera ego-motion estimates, to project the next search region in which candidate points for matching must lie. The points are then matched using an alternative implementation of the Mahalanobis distance concept. This work is later followed by Stephens et. al. [SBC+89] who used both stereo and motion to determine both feature position and camera ego-motion. They describe a prototype 3D vision system in which surfaces are constructed from the 3D representation of matched feature points and are used to segment the scene into navigable and non-navigable regions. The basis of surface extraction is in applying triangulation techniques to Kalman filter tracked corner points [HS88].

Two other recent techniques applied to 3D scene understanding are now briefly reviewed.

Walker and Herman [WH90] use geometric reasoning as a cross-over step, from formulating 3D structures to maintaining a 3D model, via model matching and object completion. Geometric reasoning is used to determine the type, position, and orientation of structures necessary to complete an object, and to hypothesise additional structures about known objects in the real world. For example, in the domain of airplanes, they all must have two symmetric wings. In effect, this approach divides the latter stage of processing of the vision system shown in Figure 1.1 into that shown in Figure 5.11.

Initially, the 3D structures (which in turn correspond to 2D features, such as corners and edges, obtained from stereo or camera motion) are developed as sparse 3D wireframe descriptions of the scene. The scene model is represented as a graph of planar surfaces, edges, and their topology and geometry. With each new wireframe developed from the next processed image, the resulting wireframes are matched with the current scene model, and the model is updated as necessary. Geometric reasoning is then used to exploit strong constraints within the domain of knowledge applicable to the 3D objects under study and derive information from 3D structures to complete the scene model. The authors apply this system to an original model designed to describe partially complete polyhedral objects, with a number of constraints. They use it in recovering 3D models of urban buildings with the constraint that all surfaces and edges are limited to be either horizontal or vertical. They then apply geometric reasoning to complete faces since buildings often have parallelogram faces, and as they do not normally float in mid-air, supporting edges can be hypothesised for floating faces. The

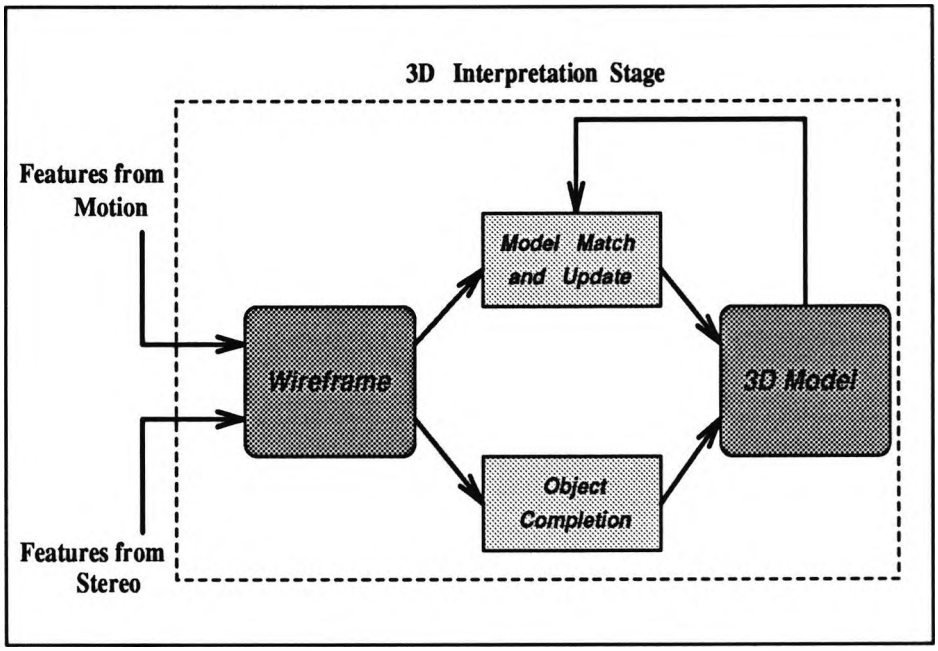


Figure 5.11: Stages in 3D scene understanding

authors also suggest the use of complete CAD models in an attempt to deal with ambiguities propagated from earlier stages. This idea has also been suggested by Ellis et. al. [EWM87].

Finally, an alternative approach to 3D scene interpretation will be considered that accomplishes recognition of a 3D object without resource to depth information. Lowe [Low87] presented a three-stage method to recognise objects from 2D images which makes use of prior knowledge of objects:

- **Perceptual Organisation** This groups straight line segments in the image, on the basis of proximity, parallelism, and collinearity. The intention is to achieve a grouping of features that tend to be invariant over a wide range of viewpoints.
- **Search Space Reduction** This locates and groups structures to reduce the number of primitives that will need to be matched at the next stage. To quote:

  > For example, trapezoid shapes are detected by examining each pair of parallel segments for proximity relations to other segments which have both endpoints in close proximity to the endpoints of the two parallel segments.

- **Model Matching** This process consists of an optimal and ranking procedure for selective matching of the perceptual groupings against the structures of the object model which are likely to give rise to that form of grouping. The author's approach included view-point solving and verification for a more accurate interpretation of model matches.

The role of token tracking is clear as a necessary pre-processing stage in all the above examples. The examples were specifically on the actual interpretation of corresponding tokens in a 3D scene, but using different approaches in accomplishing similar stages in the complete process cycle, from tokens through scene structure description to scene model matching.

In the next section, a summary of the points raised in this chapter will be provided.

5.8 Summary

In this chapter, a mixed combination of related work was presented:

- a review of some general approaches for tackling the early stages of motion understanding and scene analysis,
- a section by section breakdown of the motion tracking algorithm to be employed later in the next chapter.

After a brief introduction to dynamic scene analysis, two major approaches in the subject were outlined: *intensity-based schemes* and *token-based schemes.* Following a general review of the former, the latter topic was further presented in detail. This included a study of the state of research in token-based algorithms, and the presentation of the scene flow model as a robust technique for measuring image plane motion via token tracking. The scene flow model has been adopted in this research work for use in tracking edge tokens in particular. The flow model may be used for providing token correspondences, token velocity information, and it can aid in the recovery of depth. All of these can contribute to the composition of a 3D model of a scene. Next, geometric representation of edges as line segments were compared, with the $[x_m, y_m, \theta, l]$ representation selected as the more appropriate. Then, Kalman filtering techniques applied to motion equations were derived to aid general token-based tracking, by applying the filter to each representative feature of the causative token. This was followed by an

examination of the Mahalanobis distance for the matching of observed tokens to that of the flow model. Finally, a review of some possible approaches for employing tracked and identified tokens in 3D motion analysis were discussed.

A sequential, step by step breakdown of the overall token tracking algorithm programmed for this work is now presented. To start with, when the bootstrap stage is initiated, no knowledge of the motion of the line segments in the model is available. Therefore, the following steps are necessary for the bootstrap mode.

- The Bootstrap Mode

    1. Capture the first image frame.
    2. Perform feature extraction.
    3. Construct list of tokens.
    4. Assume zero velocity and acceleration for tokens and initialise associated uncertainties with large values to indicate a low confidence in the assumption.
    5. Assign tokens to the flow model and initialise estimated values of each token parameter with observed measurements.

The system is now prepared for the run mode which will follow immediately after the bootstrap process, and continue repeatedly for each new image frame.

- The Run Mode

    1. Use the flow model to predict the position of each token parameter in the next image frame.
    2. Use estimated values and their uncertainty to determine the search area for each token in the next image frame.
    3. Capture the next image frame.
    4. Perform feature extraction.
    5. Construct list of newly observed tokens.
    6. Use the Mahalanobis distance to match each flow model token to those observed tokens found in the search area.
    7. Update the flow model accordingly depending on the outcome of the matches.
    8. Continue from Step 1

This algorithm is inherently generalised and can be applied to any representation of tokens, be they corners, edges or vertices, and in a high level sense, it can be mapped to those algorithms employed, for example, in [HP87, CSD88, SH88, DF90, WH90] which track tokens in 2D images. Therefore, Figure 5.5 really represents a symbolic flowchart of the overall algorithm.

In the next chapter, a parallel computational model for generalised token-based algorithms will be presented followed by an implementation of the algorithm presented above using the computational model as the guide. By selecting a general tracking algorithm such as that above, the algorithm-independence of the parallel model approach is hoped to be emphasised in the next chapter. This will be demonstrated to a certain extent, by implementing step 1 of the algorithm (in run mode) using two algorithmically disparate approaches to Kalman filtering, which is construed as the heart of the algorithm. In fact, this section of the overall algorithm will be dedicated to a whole separate parallel network of processors as a separate unit, thus also displaying the independence and flexibility of the implementation.

Chapter 6

A Parallel Approach to Token Tracking

6.1 Introduction

In Section 2.1, by way of Figure 2.1, a general classification of vision algorithms for parallel architectures was presented.

The most natural approach to exploiting parallelism in motion can be determined by regarding the steps outlined in Figure 1.1. At the highest level these display an inherent pipeline effect. Pipelining, as a temporal approach to parallelism, would allow the division of the steps, in between image capture to 3D scene understanding, into independent units executing as black boxes in parallel, each supplying its outside world with its results. Further parallelism may then be investigated by delving into the domain of each of the black boxes.

By momentarily casting the reader's mind back to Section 5.7, it can be stated that whichever approach to 3D scene interpretation is to be employed, the provision of features extracted from 2D visual input is an unavoidable and universal necessity. Therefore, a major contribution of the work presented in this chapter, is the provision of a continuously updated scene flow model as part of the front end processing to an AGV vision system. To achieve this, the first two black boxes of Figure 1.1 will be considered in some detail, with major concentration on the second stage, i.e. the establishment of token correspondence. The unique aspect of this work in comparison to other related work is the parallel computational model designed as an approach to efficient implementation of the scene flow model on parallel processing hardware. Also, the implementation itself is maintained as a general algorithm, so

that it may be adapted easily if changing conditions and requirements so dictate. Moreover, the approach proposed in this chapter, decouples the stages involved in the tracking of tokens into independent units, providing a chance for absolute optimisation of each unit. Thus, the system as a whole may be fine-tuned for a better performance. These points will be elaborated on later in this chapter.

With respect to algorithmic and geometric parallelism, a hybrid of these two parallel programming models will provide for a suitable implementation of the token tracking algorithm. This issue will be discussed in Section 6.5.4, where it will be shown that in using a hybrid approach, system performance can deteriorate due to certain characteristics of the transputer.

The work in this chapter is divided as follows. Initially, a review of work in motion is presented where parallel processing has played an active role. In Section 6.3, an initial investigation into motion analysis is outlined using differencing and chain-coding techniques to aid the tracking of objects in a simple scene. Next, in Section 6.4, a skeleton parallel computation model for tracking tokens is proposed. The model will be independent of the tracking algorithm and the host architecture. In the subsequent section, the model will be used efficaciously to realise a transputer-based MIMD implementation of the tracking algorithm as described in the previous chapter. This section will encompass many features, such as a performance evaluation of the Canny edge operator. In Section 6.6 results of the implementation are produced, and general summary and conclusions are presented in the final section.

6.2 Use of Parallelism in Motion

In this section, a brief review of the use of parallelism in dynamic scene analysis will be presented, with special leaning towards the use of transputers as the principal parallel architecture. The discussion will also concentrate on the fundamental issue of achieving correspondence, since that topic is the basis of this work.

Until recently, the concentration of research in motion has been on the improvement and accuracy of motion algorithms only, a few examples are [ADM81, HJ83, SJ87, HP87, THKS88, SBC+89]. When the need for efficiency has arisen, many have resorted to implementing their algorithm in hardware [CSD88, SBC+89, DF90]. Thus, the use of parallel processing has been largely neglected, probably due to the lack of availability or access to parallel architectures. The arrival of the transputer

as a cheap, available source of parallel computing is about to change this, and some recent work which has been trickling through will be reviewed a little later.

6.2.1 Some Notions

The pipeline approach discussed earlier, is a logical structure for perception, and has been employed by most of those who have attempted to view the motion problem from a parallel perspective, (regardless of the relation in their approach to human perception). Pipelining will be at least inherent in all the works reviewed here.

Ullman [Ull79] addresses three assumptions in considering the computational feasibility of correspondence: *Parallelism, Locality,* and *Simplicity. Parallelism* is deemed necessary since there are a large number of elements, and Ullman states that the pairing of corresponding elements can be accomplished, *"to a large extent"*, in parallel. In this thesis, it will be shown that due to the independence of the tokens, and by way of the parallel computation model and the consequent implementation, correspondence of the flow model tokens can take place in full parallelism. *Locality*, states that there are only local connections between processors. By this, Ullman has in mind the issue of sharing and exchanging of correspondence information. This issue will be discussed and respected by the model but regarded in implementation as an unnecessary step in a parallel approach to determining correspondence between independent tokens, i.e. local communications will not be necessary. *Simplicity* states that the individual processors are rather simple computing devices. This classification is more befitting of machines such as the CLIP4 [FMM88] and the AMT DAP [HJ88] (see also Section 2.3.7) and is certainly not applicable to transputers. But a number of simple processors could be simulated in software on transputers to achieve the same effect if necessary. In fact, in some areas of research into neural networks, exactly such a notion has been considered [OHRS90]. Ullman continues by defining a simple network to perform a minimum-cost mapping function to perform correspondence. This in principle is the root of the approach described in Section 5.6, when using the Mahalanobis distance for determining the closest match.

Martin and Aggarwal [MA78] have suggested three phases of motion perception as existing in parallel in the human visual system. (The reader is reminded to notice the inherent pipelining principle).

- **Peripheral** processes that scan the field of view and detect features within the field such as colour, shape, texture, but in this

case mainly motion,

- **Attentive** processes which focus their attention on interesting areas of the field of view, and must be able to track the movement and understand the associated details of the object in motion,
- **Cognitive** processes that perform higher-level abstractions by relating observations from the preceding two phases to knowledge about the real world.

Thus, while peripheral processes extract image features, attentive processes could be expected to perform token correspondence, and cognitive processes could be active in the interpretation of motion. Ullman [Ull81] has suggested that a combination of intensity-based schemes for peripheral processes, and token-based schemes for attentive processes, could be engaged as the feeding processes to cognitive activities.

6.2.2 Some Examples

Tan and Martin [TM86] have applied peripheral and attentive processing to the problem of rigid object tracking, and have implemented their approach using the simulated MIMD parallel environment called PISCES [Pra85](please also see Section 2.3.4).

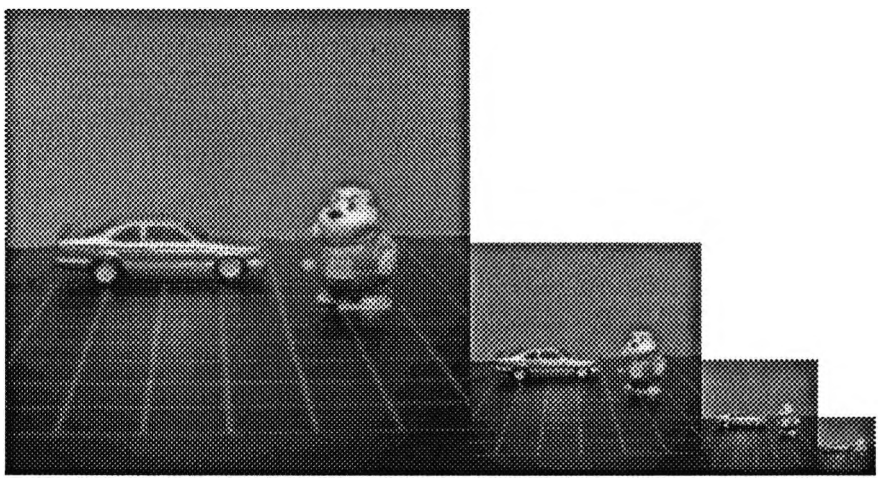


Figure 6.1: Example of a multi-resolution pyramid image representation

They create a set of parallel, peripheral processes to extract promising areas representing possible motion. These areas are subsequently di-

rected to other processes for attentive processing. The system is made up of a pipelined image pyramid structure[1] (Figure 6.1), with peripheral processes active at the coarsest level, and attentive processes cooperating at the finer resolution levels. Attentive processes move to a finer resolution level on successful matching. Loss of objects forces the processes to coarser resolution levels. Each level of the pyramid maintains the image in a different time frame from that of its neighbour level, with the finest level holding the most recent, as shown in Table 6.1.

Tracking is achieved by using trajectory of motion to project a search window in the next level of the pyramid, to be attended to by the associated process.

Level	Resolution	Time
0	64x64	$t-3$
1	128x128	$t-2$
2	256x256	$t-1$
3	512x512	t

Table 6.1: A four-level, temporal, pipelined pyramid, with t as the present time

Their implementation, running on a VAX 11/780 under UNIX 4.2, uses many novel ideas, such as bounded windows on the image for more restrictive searches, and a central information exchange repository called the scene description model (sdm) through which *agent* processes can communicate. Unfortunately, the implementation suffers from a lack of generality, in which only rigid objects, described by their area, and major and minor axes, can be tracked, with the simulation performed on paper cut-outs. For example, their object detection approach would be inadequate for a real scene. Occlusion becomes a specifically acute problem since much of the scene information is lost at the coarsest resolution level. Another cause for the deterioration in performance is the idea of the four-level, temporal pyramid, and its asynchronous nature with respect to the agent processes. (Note, the following comments would apply even more acutely for a more realistic scene.) The pyramid is continuously updated as new frames arrive, while attentive processes traverse down the pyramid. This flow-through of images in the pyramid structure means that an agent process would not necessarily have

[1] An image pyramid structure may be constructed by partitioning the image into 2x2 pixel sets, and using the average value of each set to form the image layer of the level above. Please also see Equation 6.11.

access to consecutive images as it progresses to a finer resolution level. Moreover, images flow through regardless of the processing speed of the agents. Thus, due to computational delays, agents can miss frames and lose track, and objects may be picked up by new agents thus causing duplication, all leading to further inefficiency in performance. However, as a basis for the consideration of the problems involved in motion from a parallel perspective, Tan and Martin's work provides an important analysis and implementation.

For a more recent point of view, two major transputer-based systems will now be discussed which have a direct bearing to AGVs. These two are part of four systems whose associated research activities are being pooled together within one consortium known as the VOILA project [Bux91]. The intention of the VOILA project is to develop several dynamic vision systems for the control of robot vehicles operating in different environments such as, indoors in industrial and commercial areas, or outdoors in stockyards and car parks etc. Thus the software and hardware knowledge of some eight separate European institutions are being integrated to produce efficient platforms for diverse applications. All the information regarding the systems about to be noted are cited from [Bux91], a report which only became available at the time of writing this thesis, thus much of the information is entirely new.

The DROID project has already been mentioned and some related work has been covered [HP87, HS88, SH88, SBC+89] in the previous chapter. In summary, it consists of a corner detection scheme for token matching, followed by 3D estimation and analysis. Although [SBC+89] reports a hardwired schematic of the corner detector [HS88], a new transputer based implementation is reported in [Bux91]. This shows an approach similar to that presented later on in this chapter, where a dedicated network of transputers is proposed and assigned to the process of feature extraction as the first step in the process of token correspondence. The corner detection is performed on transputers, arranged in multiple one dimensional arrays and employing a farming approach. The collection of a SUN4 host, the network controller and slave processors is known as PARADOX. A Datacube MaxScan framegrabber board is accessed in PARADOX by a special interface board with an on-board transputer. The interface board allows image data to be passed at video rate to an array of 32 transputers on a Transtech MCP 1000 card. Images are partitioned into 4x7 segments (no explanations are provided for this) and transmitted in parallel to the multiple farm networks. Following the corner detection, the data is passed to the Sun4 host workstation which as well as acting as the overall controller, carries out the correspondence and 3D computations required

by the DROID system. On a serial machine, the corner detector is said to account for 90% of the entire computation of DROID's 3D scene interpretation capabilities. PARADOX performs its full analysis in 0.87 seconds per 256x256 frame with comparable performance between the separate stages assigned to the transputer network and the SUN4. The VOILA report suggest many improvements as currently under consideration for DROID, such as more accurate ego-motion measurement, and a transputer implementation of the correspondence and the 3D scene analysis stages.

The second system is a combination of the TINA vision software and MARVIN multi-transputer architecture. Although these two systems are fairly well-known in the vision community, their development into 3D motion analysis is quite recent and new features have been revealed in some detail only in the VOILA report. For example, Rygol et. al. [RPBK90] discuss the recovery of 3D scene geometry and control of a robot arm for picking up (stationary) parts, but conclude only with an announcement of their intention to implement a feature tracker. Hence, as with DROID, it is interesting to note for now the authors' approach to the problem of correspondence in particular.

The GEC HRC MARVIN (as defined in [Bux91]) consists of special purpose TMAX cards each containing 1 Mbyte of dual-ported video RAM, and a T800 transputer that provide a wide bandwidth, multinode interface between data on a Datacube frame-grabber, industry standard MAXbus, video-bus and a transputer array. Initially, it is reported that full-frame 3D stereo analysis, including Canny edge detection, stereo fusion, line fitting, object recognition and location is performed in 12-15 seconds on the 18 transputer MARVIN system. This is declared as an order of magnitude too slow for a machine to maintain an on-going description of its environment. Next, in order to exploit an alternative avenue, the VOILA report presents an example of tracking a known object on MARVIN by tracking the parallel lines of the object, followed by the use of the successfully tracked segments in the estimation of the position and orientation of the object as a whole. Their approach is based on the exploitation of the spatio-temporal coherence of the world. The full-stereo system described above is initially used to recognise and locate the object, and to *boot* a *run-mode* in which the predicted position of an object is used to reduce the computation load. This predictive feed-forward algorithm is capable of returning the 3D position and orientation of the moving object in 300ms. The interesting factor is the combination of the edge extraction and tracking procedures implemented on MARVIN using the processor farm model. The farm controller is known as a virtual tracker, and the nodes as

feature trackers. Each feature tracker, which is resident on one TMAX card to ensure the most rapid access to the stereo data, performs 1D edge detection in the vicinity of the predicted position of the line segment to find the new actual position of the object's edge feature. New line segment descriptions of the edge are then computed for both stereo images[2] and projected out into 3D. The master virtual tracker conducts the distribution and collection of features and results respectively. Following the return of all the features, the latest position of a moving object is calculated.

The above description is of a prototype which seems to perform well for a known object in a well-contrasting scene, and where not all the segments need to be tracked successfully to determine the position and orientation of the object. Similar constraints applicable to object tracking systems in general are prevalent here too, e.g. severe occlusion. The combined edge extraction and tracking procedure is a fair approach which compared to a separately staged setup, reduces data communication rates (at the cost of design modularity). Moreover, this combination is a more practical proposition for their application, since the predicted search area where the image should be edge filtered is more accurately determined through the familiarity with the object geometry, e.g. the knowledge of the length of the expected edge segment.

This prototype MARVIN system is involved in a project with many software and hardware facets and with a strong manpower base, and it is expected that future research will lead to a system more readily applicable to general scenes as defined by the aims of the VOILA project. The examples will now continue with two more capsular descriptions of transputer-based motion detection systems using intensity-based schemes.

Stephen et. al. [STD90] have used a simple transputer set-up to determine the displacement of Civil Engineering structures, such as the behaviour of the Humber bridge at the centre of its span, using a correlation/template matching technique, applied to a single target object only. The tracking and predicting procedures are as follows. A 12x12 pixel template of a user selected feature is searched for in subsequent images. The matching is determined by evaluating a least square error similarity measure over a 32x32 search window. The prediction algorithm also uses a least mean square parabolic approximation applied to the object's motion trajectory. This is not as optimum a technique for

[2]This author believes that by using calibration information from the geometry of the stereo cameras, the matching of the edge segments in the stereo pair of images is carried out through a one-dimensional search along the connecting epipolar line.

linear prediction as Kalman filtering, but results in a simple equation for fast computation and still provides some measure of immunity to noise. The implementation consists of a single processor which tracks, and a pipeline which is used to execute the prediction process, in stages, on its three nodes. A real time response is achieved due to the small size of the images dealt with, and the application to a single target object.

Bernat and Rupel [BR90] have presented a cellular, transputer-based system to detect and track human motion across the border between the United States and Mexico. The tracking and prediction technique is based on the differencing technique, with median filtering applied to reduce the effects of camera noise or wind shake. The image is divided into geopixels (a rectangle of contiguous pixels in the image), each of which is associated with a cell which determines the cause of motion in the geopixel. The cells base their decision by consulting with neighbouring cells. Various thresholds are used to distinguish between noise and real motion, and motion is tracked by noting movement from "cell to cell". The parallel implementation treats cells as processes on a single transputer which communicate over soft (internal) channels, and the geopixels are treated as data packets farmed out to slave processors. The authors itemise a number of shortcomings of which two of the most interesting will be cited here, selecting one from a motion point of view, and one from a parallel implementation point of view. Each cell bases its decision on the existence of motion, using a pre-defined equation relating the input from the change detectors between two frames, and the cell's decision from the previous frame. Accordingly, decisions undergo a relaxation during communication, and if they are not continually supported by additional cells detecting a change, then prediction continuity will suffer. The implementation is of the one-dimensional case for the pre-defined equation. Objects in real-life do not move in one dimension, and to implement motion in two dimensions, the cells need to communicate with their diagonal neighbours too. Hence, the authors state their preference for increased number of links on the transputer. The solution necessitates multiplexing and re-routing.

Note that the last two items of motion work on transputers have been based on intensity-based schemes, and have been successful due to the nature of the applications. Intensity-based schemes can also find an inherently suitable platform in neural networks. However, this author has found no particular references to support this idea.

Other transputer-based motion applications of note are [AD90] and [TLM+90], both of which provide little discussion on the actual trans-

puter implementation. (The motion algorithm used in [TLM+90] has been cited in Section 5.3).

6.3 The Initial Investigation

The aim of this section is to outline some initial work on object tracking carried out in the attempt,

- to have a more thorough understanding of intensity-based motion techniques by regarding one in detail,
- to understand the basic, but fundamental, issues in dealing with full, rigid objects in a sequence of images, and as a consequence, use the knowledge for a better approach towards tackling object primitives, namely tokens,
- to help lay out the basic requirements for the parallel computation model, particularly with regard to process communication and synchronisation,
- to design a modular software structure that will be capable of embracing various tracking algorithms, intensity-based or otherwise, with little change, and still based on the parallel model.

Stated in order of importance within the context of this section, the latter two were by far the most important issues. Hence, by attempting the implementation described here, which will be regarded compendiously, much work in the later application of the token tracking algorithm was saved. This work on object tracking was based on differencing and chain-coding techniques (please see Sections 5.2.1 and 2.2.3, respectively). Yet, only the motion issues will be concentrated on at this stage, saving the parallel processing issues for the later sections to come. The aim is therefore to merely share some notions that were either applicable or inappropriate for use in the approach to token tracking. It will however be said that the system was fronted by TIPS, expanded to run on both a PC-Host B004 board, and a Harlequin frame-grabber board with a T800, 20MHz processor and 1Mbyte memory on board. The network was arranged as a tree network. All these issues and more, such as the system communications, process load balancing etc., will be discussed in detail for the token tracking application, since the underlying system architecture is applicable to both object and token tracking techniques.

The technique used here is similar to the *accumulative differencing* technique, which allows the analysis of frames $F_2, ..., F_n$ against the reference frame F_1 in the sequence [Jai81]. This allows a partial history of the movement of an object in the scene to be observed. From the outset, it was decided that as a first step towards parallelism, each object must be assigned to a separate, monitoring process. To keep matters simple, the idea of *smoothness of motion* and *path coherence* [SJ87] for a rigid object will be assumed. With these points in mind, the implementation will now be described through the consideration of its main stages: bootstrap and run.

6.3.1 Bootstrap Stage: Continuous Object Detector

The bootstrap process should have more appropriately been named the *continuously-executing object detector* process. This process remains active throughout the run of the system. Initially, by user interaction, it saves an image of the scene which becomes the *reference* image. Then, through its direct access to the Harlequin frame buffer, it samples and accepts images each time it has finished processing the previous image. This processing of the image incorporates the following steps,

1. Capture the next frame.

2. Perform differencing, against the reference image, on a "hot space" border area around the image. Keep an account of the number of pixels that satisfy the threshold Δ, in CP (Changed Pixels),

$$CP_t = CP_t + \begin{cases} 1 & \text{if } |F(x,y,0) - F(x,y,t)| > \Delta \\ 0 & \text{otherwise} \end{cases} \qquad for\ t = 1, 2, ... \tag{6.1}$$

   where Δ was taken as 25, to represent (approximately) a 10% change in the state of a pixel from the possible range of values associated with a pixel, which is $[0, ..., 255]$. The "hot space" area is shown in Figure 6.2.

3. If the total count exceeded a predetermined threshold, in this case 10, then the hot space area is searched by a chain-coding process, which produces an *object list* of all the objects in the area (The objects must have at least their starting pixel co-ordinates in the

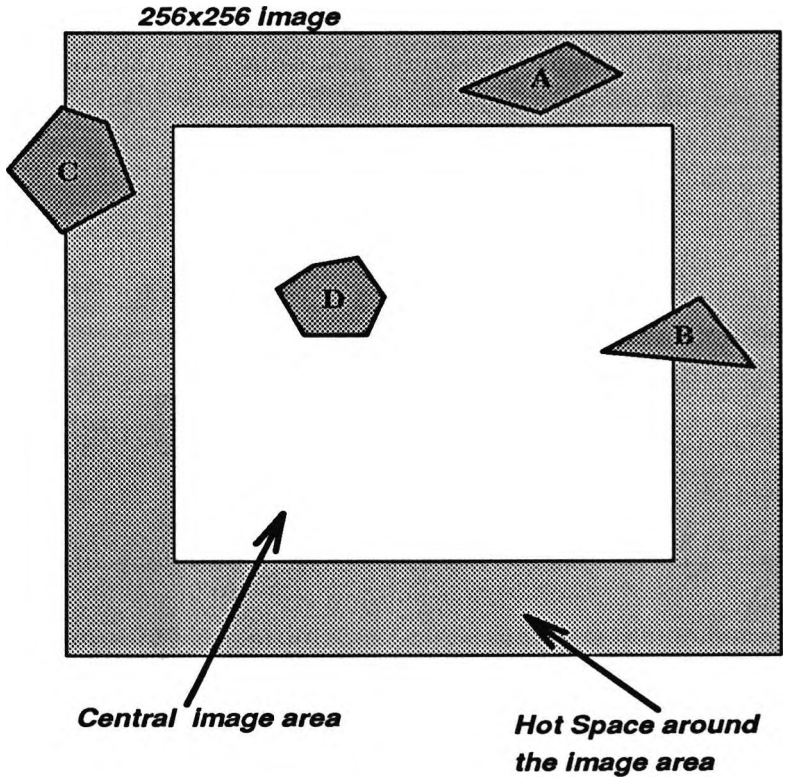


Figure 6.2: "Hot Space" area of image and some spatially classified objects

"hot space" area, therefore both objects A and B in Figure 6.2 are detected). To avoid confusion, objects that are found to be still in the course of entering the image are struck off the *object list*, since their description will be incomplete, for example object C in Figure 6.2. It is assumed that the object will be captured in a future frame. Should the object never enter the image completely, no unnecessary action will have been undertaken.

4. Send the list of objects found to the network controller, and continue processing from Step 1 again. The list contains information describing the chain-code, the area, and the centre point for each object. Addition of further characteristics would be a simple process of adding to the data structure describing each object.

Thus, the bootstrapping process attempts to minimise its amount of computation by only searching a limited area in the image (if it was to process the full image, a heavy processing bottleneck would have

built up at this early stage). Furthermore, the bootstrapping process's function remains independent of the subsequent tracking stage. Also, note that the direction of motion for objects A, B and C is irrelevant to the detection process. The objects are dutifully detected and reported, but it remains the task of the tracking controller process to establish if an object, such as object C, is just entering or just leaving the scene.

Note, other processes run concurrently with the detector process to handle the data exchange between it and the tracking controller process.

6.3.2 Run Stage: Continuous Object Tracker

The run stage consists of a controller process to manage and organise the tracking and a number of tracker processes to carry out the actual task.

Controller Process

Whenever the detection process communicates with the controller process, it passes the *object list* and the full associated image. The controller itself consists of a number of parallel processes which are simplified and itemised below.

- Buffer processes which queue incoming lists and images.
- A process which accepts an *object list* for the latest frame plus its corresponding image. It scans the list and matches the attributes of the objects within, to those currently being tracked. If no match is found, the object is issued as new to a waiting process. (Matching is a straight comparison of the object attributes all of which must satisfy corresponding thresholds).
- Buffer processes which queue incoming processor requests.
- A process which accepts requests from the tracking processes for sub-images which are expected to contain the next instant of an object.
- Buffer processes which output results to the outside world, i.e. TIPS.

Each tracker process executes the following (compacted) algorithm as enumerated below, given the object state vector S_f at time t_f in frame

f to be $S_f = [S_{xf}, S_{yf}, S_{\theta f}]$, describing the object's current position co-ordinates and direction of travel. $MISSING$ is a variable used for occlusion analysis.

1. Accept assignment for tracking a new object, and receive object attributes. Thus at frame $f = 0$, time is t_0, state vector is S_0, and $MISSING = 0$.

2. Request a sub-image of the next frame, where the size of the sub-image is determined using a large uncertainty.

3. Receive new sub-image ($f = f + 1$). Perform chain-coding and form list of objects found in sub-image. Match each object to that under investigation. If successful, update state vector with observed measurements, set $MISSING = 0$ and continue, otherwise carry on from Step 8.

4. Using the previous and new position attributes, estimate the velocity V_f, and predicted displacement D_{f+1}, and thus predict the state vector S_{f+1} of the object in frame $f + 1$,

$$V_f = \frac{\sqrt{(S_{xf} - S_{xf-1})^2 + (S_{yf} - S_{yf-1})^2}}{t_f - t_{f-1}} \tag{6.2}$$

$$D_{f+1} = V_f * (t_{f+1} - t_f) \tag{6.3}$$

$$S_{\theta f} = tan^{-1}(\frac{S_{yf} - S_{yf-1}}{S_{xf} - S_{xf-1}}) \tag{6.4}$$

giving new position co-ordinates for S_{f+1} of,

$$S_{xf+1} = S_{xf} + D_{f+1} * COS(S_{\theta f}) \tag{6.5}$$

$$S_{yf+1} = S_{yf} + D_{f+1} * SIN(S_{\theta f}) \tag{6.6}$$

5. Use the new position to determine the next search area. This is centred on the position with a radius determined from the maximum and minimum pixel position co-ordinates of the object, obtained when chain-coding the object. If the sub-image lies fully within image limits, then continue from Step 6 (This would be applicable to object D in Figure 6.2). If the sub-image is found to lie partly outside the viewing limits of the camera, but only by a margin less than a pre-determined tolerance, then continue from Step 7, otherwise give up and continue from Step 1.

6. Send request for search area sub-image to controller and continue from Step 3.

7. Send signal to report that object is leaving the field of view, followed by an adjusted search window. Continue from Step 3.

8. Let $MISSING = MISSING + 1$. If $MISSING$ (occluded) is less than a pre-determined tolerance, then use previous associated kinematics to estimate new search area and continue from Step 3, otherwise report to controller that the object has been lost and continue from Step 1.

The uncertainty measure is achieved by over-estimating the size of the search area, and thus some protection against variance in actual motion is achieved.

6.3.3 Some Remarks on the Investigation

This implementation was tested on uncluttered, simple scenes. Plate 6.1 shows four (numbered) frames from a motion sequence of a single toy object representing the object's history as it moves from scene-left to scene-top-right. The rectangular areas denote the search area predicted by the tracking algorithm. Notice how the very initial appearance, i.e. in spatial position, of the object resembles that of object C in Figure 6.2. The object's next two full appearances in the image is then used to make the first prediction for the third occurrence. (This has not been reflected in the algorithm described above for simplicity).

Similarly, Plate 6.2 shows the progress of two objects. One object is travelling from left to right and the other is travelling in the opposite direction. The objects are travelling at a slower speed than that in Plate 6.1, thus a continuous deposit is left in the image.

Occlusion is tested and shown in Plate 6.3, where the area between the dotted lines is an object which has disappeared through image differencing, since it remained stationary. The moving object is lost after *frame 32* and regained at *frame 38*. A new prediction for *frame 39* is shown, followed by the new position in *frame 40*, which due to the slow motion of the object is still within the search area of *frame 39*. When occlusion occurs, the tracking process requests the full image in its attempt to relocate the object, thus drastically increasing its computational load.

Objects, regarded as homogeneous regions in the image, are putative tokens. The main advantage of this investigation has been the provision of a springboard for designing an efficient parallel model and distributed processing environment for inter-frame correspondence of

tokens. The implementation is employing a combined intensity-based and token-based approach to detect objects and to establish token correspondence, even though the matching is a simple process of object attribute comparison. The "hot space" detection mechanism works well and reduces computation, although it may miss objects if they travel through it fast enough. Also, it is only applicable if the scene is known to be devoid of moving objects to start with. It's use is therefore completely redundant for considering camera ego-motion, where instantaneous image-global movement may be observed. The width of the "hot space" area could be optimised to suit a particular application where the maximum speed of objects would be known. Examples of application areas are security surveillance, car parking-lot and aircraft taxiing monitoring, all of which are quite controlled environments and can be managed with less sophisticated techniques. Flexibility for administering a more elaborate tracking algorithm, using better object descriptors for more precise segmentation, also exists due to the modularity of the design. The system's tracking capability could be enhanced by employing a more reliable estimation technique, such as Kalman filtering. For example, a Kalman filter would be more precise in estimating a smaller search areas for an occluded object. For occlusion, the uncertainty associated with the estimation would grow frame by frame while the object is missing, increasing the possibility of a re-encounter.

Some shortcomings of this simplified approach are its pre-requisites for the use of a static camera, and the need for almost constant illumination. More detailed analysis of object tracking issues may be found in various published articles such as [FT79, ADM81, TB81, Nag83, TM86, SJ87, HS90].

There are many questions that are raised by this section. How modular is the design, how are the processes distributed amongst processors, how do they communicate with the controller, how does the system achieve a balanced load, and how does the controller deal with incoming messages? All these issues and more will be clarified through the tackling of the problem of token tracking in the next few sections.

Plate 6.1: Frames 11, 18, 24 and 33 of a sequence showing the tracking a single object

Plate 6.2: Frames 14, 30, 56 and 76 of a sequence showing the tracking of multiple objects

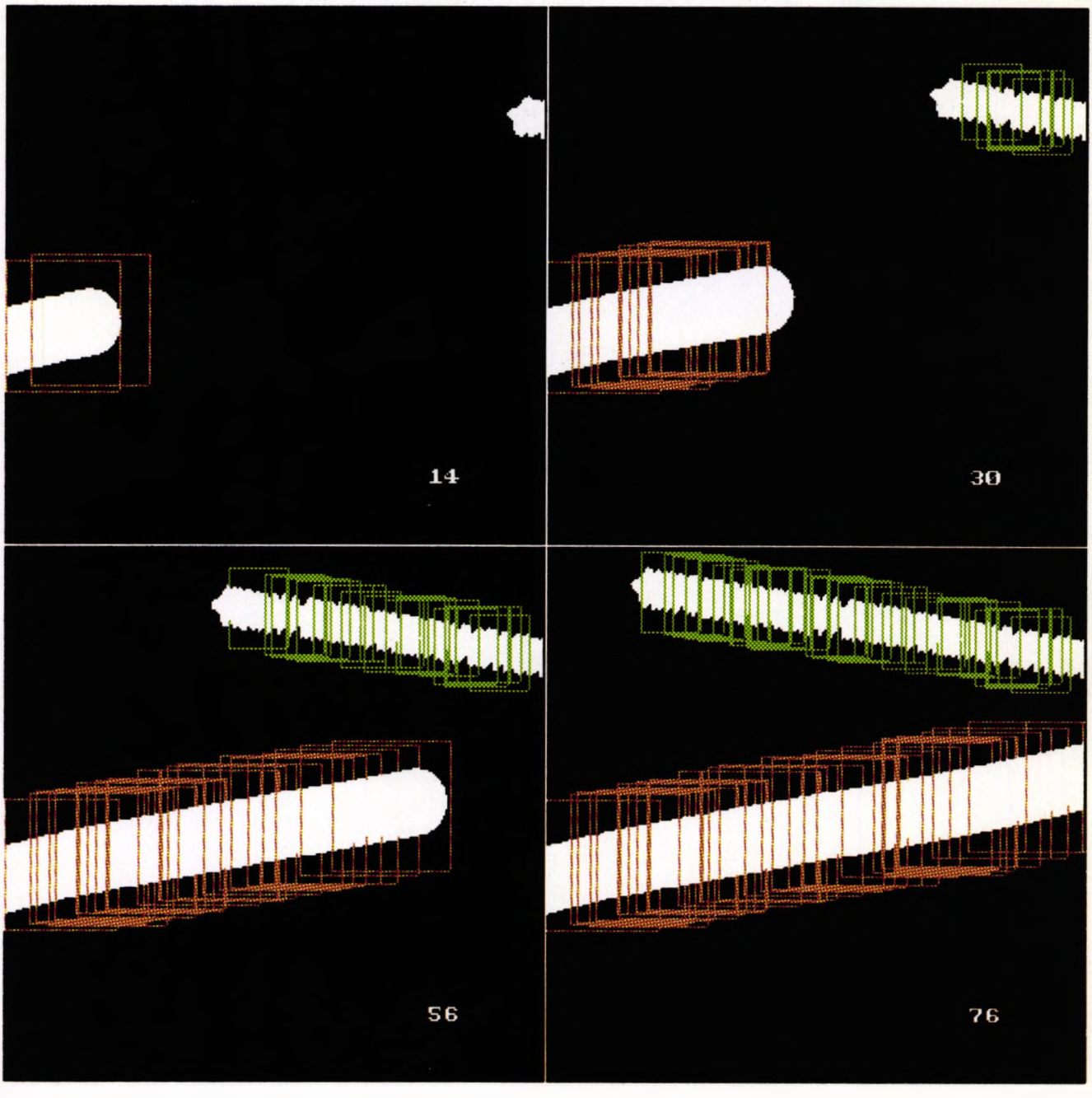
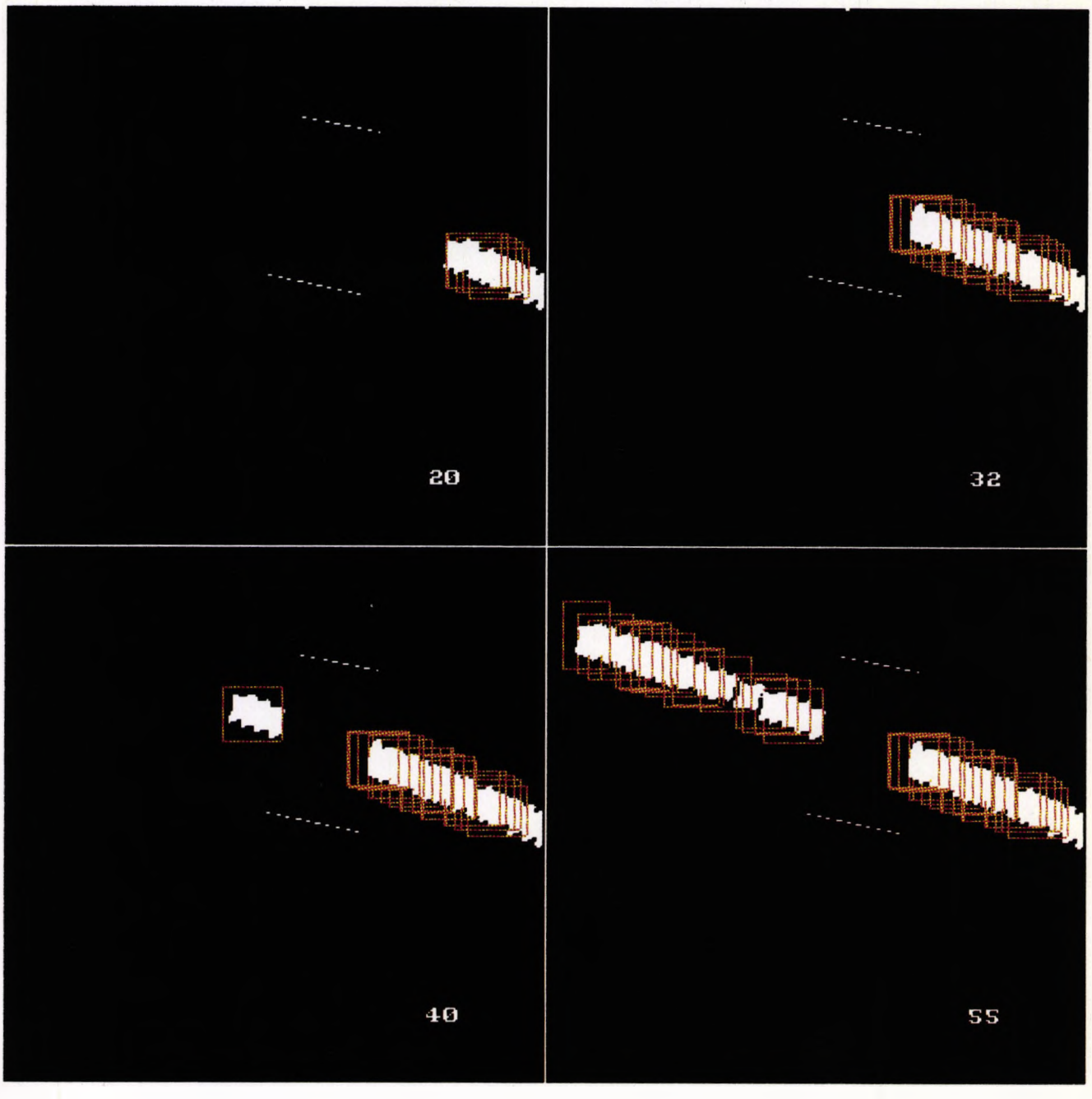


Plate 6.3: Frames 20, 32, 40 and 55 of a sequence showing continued tracking despite interruption by occlusion



6.4 A Parallel Computational Model

A skeleton, multiprocessing model is now presented for parallel computation of inter-frame correspondence. The model is kept concise and general to allow it to remain applicable to both shared-memory and distributed processing parallel platforms. The model could have been described more specifically to represent a "handbook" for a loosely-coupled transputer implementation. Perhaps this may be said to have been achieved in the next section, where the implementation specifications can be grouped together and regarded as a matured model. But in this section, for the sake of clarity, portability, and applicability, a non-specific, high-level approach will prevail.

6.4.1 Assumptions, Requirements and Preliminaries

The model assumes that tokens are continuously available from a sequence of frames through a token extraction phase. No waiting is expected in accessing a new set of tokens for the next frame. To attain a reasonable, general-purpose viability, the following requirements should be claimed by the model,

- as many tokens as are necessary must be tracked,
- the nature of a token (corner, edge, any connected region) must be arbitrary, so long as it can be described as a feature vector or matrix, i.e. as a standard data structure,
- to each token one tracking process must be assigned,
- the model is to be independent of the tracking algorithm, and thus, the structure of the model tracking processes must remain independent of the tracking algorithm,
- addition of further processors or processes must not necessitate alteration of the model, or the tracking algorithm,
- in a high-level descriptive sense the model must remain independent of any concurrent architecture,
- the model should be implementable on any concurrent architecture, although one may not be as efficient as another.

(Those points in the above itemisation not previously covered, will be referred to as the features of the model are unravelled.)

The model does not intend to exploit parallelism at the algorithm level, since it is not expected to have any knowledge of the algorithm. The parallelism lies in the handling of the tokens. Any process-local parallelism is left as the responsibility of the tracking process to capitalise on.

Before continuing with the model definition, the variables displayed in Table 6.2 are defined for use throughout this section.

P	total number of processes
T	total number of tokens
N	total number of processors
Q_i	total number of processes on processor i (for $i = 0, 1, ..., N-1$)

Table 6.2: Some definitions used in the model

The model defines that all tokens must be processed concurrently. This is commensurate with the assignment of at least one process to each token to be processed in parallel with other tokens on the selected (parallel) architecture. There may be occasions where two or more processes are tracking the same token due to inaccurate predictions or noisy data. This forms a possible many-to-one mapping between processes and tokens:

$$P >= T \tag{6.7}$$

In the real world, there may not be as many processors available as there are processes, or tokens, therefore the model makes provisions for a number of processes, i.e. token trackers, to be executed in parallel on the same processor. It can therefore be expected that,

$$P >> N \quad and \quad T >> N \tag{6.8}$$

In the following discourse, it will be assumed that $P = T$ for simplicity, except when stated otherwise, and that processes and trackers are one and the same and may be referred to interchangeably.

Given the issues discussed so far, the model assumes that a parallel architecture involving a number of processors is available, although it is unimportant how these processors are interconnected or even how the processes executing on this network of processors share or have access to information. What is important, is that they DO share and

have access to information. The necessity for a systematic control of the tracking processes, and the coordination of the laws of information sharing as applied to them, combine to dictate the existence of an overseer or moderator or controller process. This tracking-network controller-process is reviewed next, to be followed by the other facets of the model.

6.4.2 System Controller

The task of the system controller is reflected in its name. It creates processes, synchronises their actions, accepts their termination, and controls the sharing of information, the balance and the process life of the network. It conducts its tasks through a *Blackboard* mechanism similar in concept to that of the *sdm* used by Tan and Martin [TM86]. However, unlike the *sdm* which forms the actual scene description data, the *Blackboard* in this model is the vehicle for coordinating the low level operations of the multitude of processes whose results may be used at a higher level (external to the model) for a unified interpretation of the image scene. It acts as a repository of information for sharing knowledge about the status of the whole system. It is only incidental that as a sub-function, the *Blackboard* also holds the scene flow model. The *Blackboard* may be local to the system controller, or dedicated to a separate controlling process. The former option will be assumed henceforth.

It is the system controller's task to detect the presence of replicated tracks (i.e. when $P > T$) via the knowledge represented through the *Blackboard*, and to issue the actions necessary to continue with only one track by forcing the termination of others. This necessitates some form of information-sharing between the tracking processes via the *Blackboard*, whether the implementation is via a shared-memory or a message passing system. Thus read/writes, messages or requests for new information which will all go via the *Blackboard* must be queued and serviced by the system controller on a first-come first-served basis.

In implementation, the system controller will preferably execute on a separate processor with a large memory reserve.

6.4.3 Communications

The communications of the model is largely responsible for the behaviour of the system. It has to be flexible enough to allow expansion without imposing major alterations. Communication can be achieved

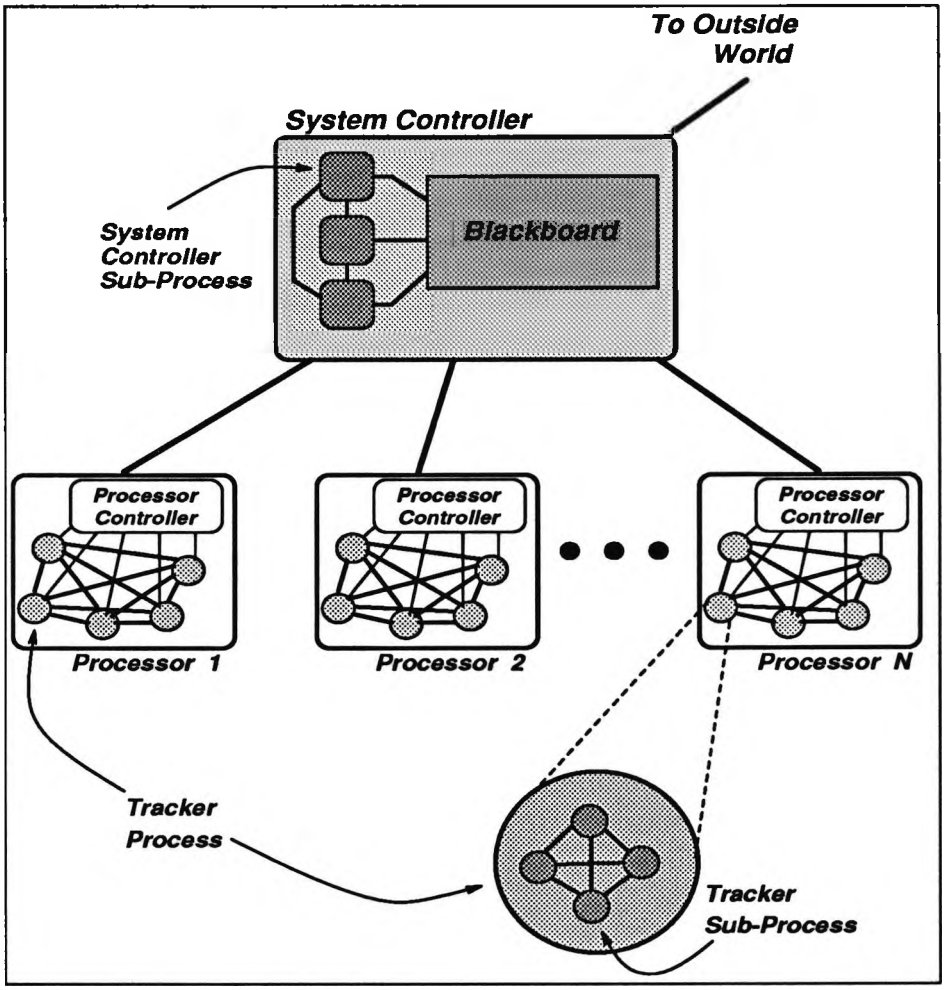


Figure 6.3: Overview of model communications

in shared-memory systems via the use of semaphores in reading and writing of data, and in a distributed memory system via the passing of messages (e.g. in tightly-coupled and loosely-coupled systems respectively). However, this remains as an implementation aspect outside of the model definitions, and any reference to *communication* must be regarded as applicable to both shared-memory and distributed memory systems or even a combination of both.

The model defines communications to exist between,

- system controller and outside world,
- system controller and processor controllers,

- system controller and tracking processes,
- system controller, its sub-processes, and the *Blackboard*,
- tracking processes on the same processor,
- tracking processes on different processors,
- sub-processes of each tracking process.

These have been visualised in Figure 6.3. Although the diagram more closely resembles a message passing system, it can also apply to a multi-processor shared-memory system where the connections show the read/write access of data which would take place via a shared bus. The number of processes shown are notional.

More details on communication will be encountered in the following sections.

6.4.4 Data Structures

Computer vision requires the handling of a wide range of data structures. For example, the pixel domain is typically represented by square arrays, but following feature segmentation, say by chain-coding, the data structure more appropriately representing the semantics of connected pixel co-ordinates would be a list of nodes for each independently chain-coded segment. Therefore, the model must ideally be able to handle the semantics of a wide-ranging assemblage of data structures. Moreover, the structures may be dynamic, and even asymmetrical. In fact, it is factors such as these that to some extent have determined the implementation of particular vision tasks on particular software and hardware environments. This is more explicitly shown in Figure 2.1.

Given the problem of token correspondence, the model can have a more definitive approach to narrowing the domain of its requirements. Regarding the communications between the system controller and outside world as implementation dependent, the rest of the communications can be specified to take place via dynamic, floating-form data structures, where the structures must be globally acceptable by all processes. Notice a globally-acceptable data structure does not necessarily pertain to globally-defined data. This would depend on a message passing or a shared-memory architecture.

For example, a process wishing to communicate its results to the system controller may pass a globally-acceptable data structure containing the information shown in Table 6.3.

0	Data Structure control id
1	Sending Process id
2	Token Characteristics
3	Token history (past f frames)
4	Predicted Position for next frame
5	Image Area Required for next frame
.	. . .
.	free space
.	. . .

Table 6.3: Example of a typical, globally-acceptable data structure, containing data pertaining to a token feature

The model is defined to be independent of any line tracking algorithm. Different tracking algorithms may require different items of information for a token. This idea of floating-form data structures ensures that the carriage of information across the model remains unaffected, and that it remains for the tracking algorithm employed to be designed to input to and extract from the data structures, the information it requires. Thus an algorithm-independent communications platform may be established.

Some other types of communication may be regarded as follows. The controller will communicate with processors to create new processes, and with processes to supply them with their required information, to accept their predictions, or register their termination. In turn, processes need to request and receive information about regions of the image and they supply the controller with their predictions. They may need to communicate with processes on the same processor or other processors to exchange tracking information, e.g. for occlusion, or to off-load the state of a track during load-balancing stages. These aspects are by nature communicationally expensive, performance degrading and to some degree difficult to implement. However when implemented, these must be limited to take place under instruction from the system controller, or at least with the approval of the system controller. Either way, the model dictates that the system controller know what is going on. It will be shown how this problem is dealt with in Section 6.5.

6.4.5 Load Balancing

By determining a suitable strategy for the allocation of tasks, a similar computation load can be achieved for each processor. Ideally, it must be recognised that different tracking algorithms are available for tracking different token representations, where the tokens may be of various description and degrees of complexity. For the moment assume the application of the model to one type of token using the same algorithm. Hence, the model requires the same tracking algorithm to be applied to each token and therefore the complexity of computation per process is approximately of the same degree. Thus, computation load approximates to the number of tracker processes on a processor and the ideal balanced state for the model is defined to be a state where the difference between the number of tracker processes executing on any two processors is never greater than 1. Given N processors, the load must be distributed at the initial *bootstrap* stage such that each processor is committed to Q processes where,

$$Q_i = \begin{cases} \frac{T}{N} + 1 & \text{if i} < \text{(T MOD N)} \\ \frac{T}{N} & \text{otherwise} \end{cases} \qquad (6.9)$$

$$\text{for i} = 0,1,...,\text{N-1}$$

During the *run* mode the controller repeatedly updates that part of the *Blackboard* concerned with the welfare of the processes. When new processes are created they are assigned to processor i where,

$$Q_i = MIN(Q_0, Q_1, ..., Q_{(N-1)}) \qquad (6.10)$$

and where MIN returns the *first* minimum value if more than one were encountered.

When tracker processes terminate, they simply return a signal so that the *Blackboard* may be updated by the controller. While monitoring the *Blackboard*, the controller may find that a processor is running fewer processes than another, i.e $|Q_i - Q_j| > 1$, for some i and j. The controller may then redistribute the computation by issuing instructions for processor i to pass some of its load to processor j or vice versa.

6.4.6 Processes and Their Behaviour

The model is defined for achieving parallelism at token or feature level rather than at the (tracking) algorithm level. Thus, by nature, a MIMD

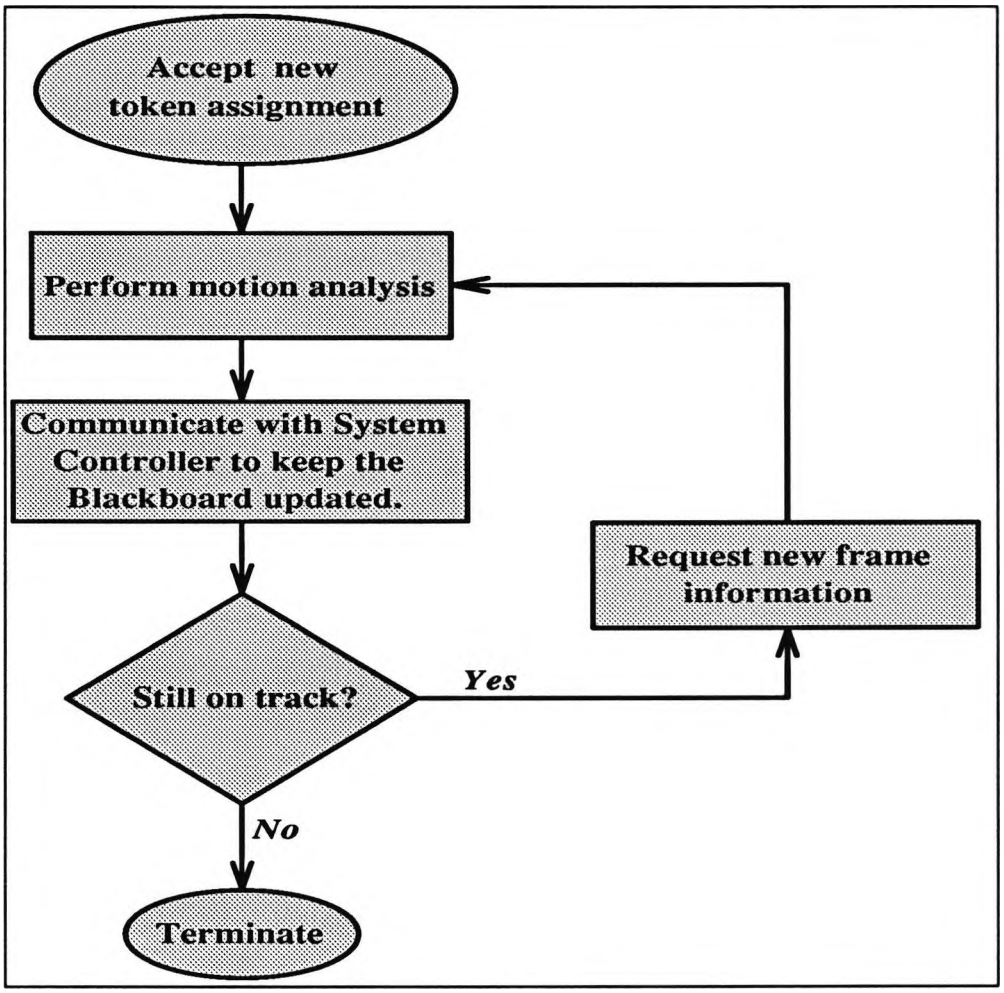


Figure 6.4: A flow diagram showing the basic algorithm of a tracker process

model is being proposed where, numerous processes running the same or different tracking algorithms, work on different data sets from different or (partially) the same parts of the image, which may or may not be overlapped. Each process is an intelligent entity capable of controlling its own activities and performing motion analysis, and given a snapshot of the status of the model's processes, they may be expected to be at various phases of computation. Each process must have access to means to apply the full tracking algorithm to its assigned token, preferably through shared libraries. For a distributed processing platform, the libraries should be replicated only once per processor. All tracking processes on each processor should have a particular status and priority

depending on the nature of the token being tracked and the importance attached to it. It may be that all processes should have equal priority within each frame regardless of their nature. A process must have access to any part of the image of any size it requires.

A flow diagram showing an abstraction of the function of a tracker process is presented in Figure 6.4. The model requires that ideally there would be no bounds on the number of processes active at any one time in the system. Naturally, one limiting factor will be the amount of memory available to each process (or processor as a whole).

New processes are created when new tokens appear in the image. Working processes simply go on tracking until they lose the track or the corresponding edge segment exits the field of view. What happens next at such situations is dependent on the particular tracking algorithm employed. Terminating processes must return their status to the controller for the last time and cease execution.

Processors remain responsible for their set of processes through a local controller. For example, they create a new process when prompted by the system controller. Some other tasks may include filtering process data in, out and through.

Synchronisation

Figure 6.3 boasts a number of different processes, local and remote, which may communicate with each other. For these to work together, some form of synchronisation is necessary. Process synchronisation will be largely dependent on the implementation platform. Synchronisation issues were described in Section 2.3.4.

The discussion above reviewed the main features of the model. Many aspects of the model will be formulated at implementation level and in the next section the model is implemented as part of the full system design.

6.5 MATCH: A Multi-Processor Token Tracker

6.5.1 Overview

The model described in the previous section will now be extended to achieve a distributed memory, message-passing, multiprocessor implementation of the token tracking algorithm outlined in Chapter 5. In addition, a section is dedicated to the edge extraction procedure and its most efficient fulfilment.

The complete implementation is named **MATCH**: The **M**ultiprocessor (or **M**ixed SIMD/MIMD) **A**pproach to **T**racking-by-Correspondence **H**ardware. The details of the hardware used are provided briefly in Section 6.6, and in more detail in Appendix A. In the meantime it will suffice to say that, firstly, the same overall equipment as that described for the work described late in Chapter 4 was available only now with more transputer TRAM modules, and secondly, that unlike MARVIN [RPBK90, Bux91] or PARADOX [Bux91], all transputer boards and modules used were industry-standard hardware boards. Therefore, the emphasis laid more heavily on pinpointing efficient implementation routes via software rather than hardware. This means that there are not necessarily any fixed configurations for the sub-units of MATCH, and the efficiency of algorithm execution will be allowed to define the hardware connections required. This helps towards the achievement of one of the parallel model's goals: the ability to extend and add processors as required (depending on the adeptness of the software design; a subject that also will be covered later).

At the highest level, the implementation is made up of three independent transputer configurations executing in parallel, as shown in Figure 6.5. These are the *HI* (Host Interface - running under the auspices of TIPS), the *FEE* (Feature Extraction Engine), and the *TE* (Tracking Engine). This arrangement is proposed as an efficient decoupling of the inter-frame correspondence problem, allowing the optimisation of each stage towards a practical whole. The *FEE* will accept images from the *HI* network, and following a token detection process, passes them in a *token list* to the *TE* phase, where they will be analysed for correspondence. From the *HI* to the *TE* through the *FEE* network, these stages could be viewed as a three-stage pipeline, which can be further complemented when the stage pertaining to 3D model and scene analysis is introduced. Please compare this pipeline configuration with that of the stages in general vision scene understanding processing as depicted in

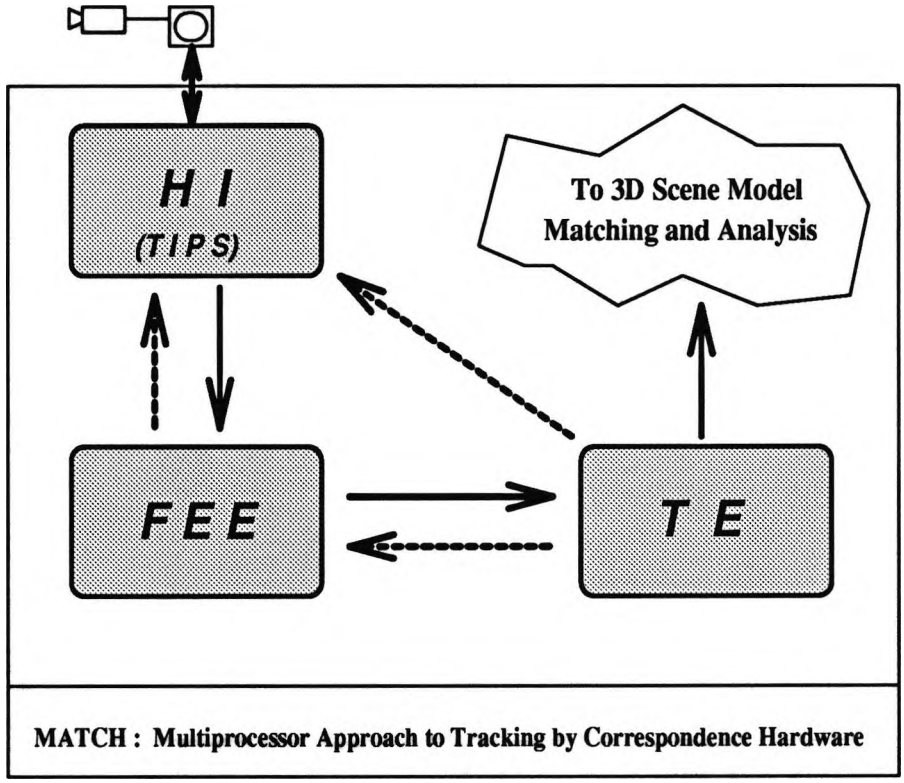


Figure 6.5: The three decoupled, parallel and communicating units of the implementation are shown in a pipeline format represented by thick lines. The dashed lines represent results and diagnostics communications.

Figure 1.1. (At present, the existing configuration is transformed into a *ring* network by assigning the *HI* to mediate between the engines and the external world.) Further parallelism is exploited in each constituent part, where the *FEE* will serve as a SIMD model of computation, and the *TE* as MIMD. The term engine is used since, ideally, the network concerned should display improved performance by the addition of further processors, as required by the model.

A network of transputers can be configured to achieve various parallel models. We examine and discuss the suitability of different models for both the edge extraction and the tracking stages. The discussion on the host interface, which is the server and the viewer of the proceedings, follows last.

Finally, an important note is brought to the reader's attention. Often, one encounters the title of a piece of research work claiming *real-time*

operation, say in motion analysis. By looking deep into the research, one may find that the real-time system operates on a single rigid object in a contrasting scene with an appropriate, well-defined model. Furthermore, what constitutes as real-time in one circumstance may not necessarily be so in another. In fact, in AGV motion analysis, motion displacement relative to camera input sampling rate is assumed slow, particularly for a complex, indoor scene, and processing on each frame may be allowed to be completed in the order of seconds. So, it may have been noticed that no extravagant claims regarding real-time analysis have been made here, since MATCH may or may not be *real-time* given the circumstances. The system's performance will be discussed in the closing sections of this chapter, outlining the factors that can affect performance, such as the complexity of the scene and therefore the number of tokens tracked.

6.5.2 Some Practical Issues

During the implementation of this work, a further set of transputers became available such that the full transputer equipment then stood at, the PC HOST B004, the Harlequin Frame Grabber, and a total of eighteen 20 and 25MHz TRAM transputer modules each endowed with 1Mb of RAM, and with a memory access speed of 3 or 4 cycles. The extra transputers particularly allowed for a better evaluation of the feature extraction stage of the implementation, the results of which are presented in Section 6.5.3.

Throughout this work image sizes of 256x256 are used. This was chosen in contrast to 512x512 images, purely due to the lack of memory. The system is designed at every stage to cope with larger images. In fact, each time an image is captured during test runs of the system, it is a 512x512 image, which is subsequently reduced in resolution into a 256x256 and then fed into the system for analysis. The process of reducing the image resolution works by assigning the average of the sum of every group of 2x2 pixel sets in a 512x512 image, $F(x, y)$, to the pixel value for a corresponding position in a 256x256 image, $f(x, y)$, using the following equation,

$$f(x,y) = \frac{1}{4}\sum_{i=0}^{1}\sum_{j=0}^{1} F(2x+i, 2y+j) \tag{6.11}$$

for $[x, y] = 0, ..., 255$.

6.5.3 The Feature Extraction Engine

WARNING!: This section makes many references back to earlier work described in this thesis.

Let it be assumed that the next frame in a sequence of images is available. The *FEE* is then the first stage in the manipulation of the tokens. The selected features in this application are edge segments. Under perfect conditions, edge segments may be continuously extracted and considered as possible recurrences of the same edge features from the most recent frame in a sequence. However, edges are spatially extended features, and factors such as illumination, digitisation, and camera vibration are amongst many that can lead to inconsistencies in extracting them from real images. (Edge drop-out has been considered as a major problem by [CSD88, HS88, DF90].) The function of the *FEE* is therefore of paramount importance, since a poor provision of tokens will lead to a poor tracking rate, causing a high rate of track loss. The effort in ascertaining a satisfactory level of consistency at the least computational cost will now be investigated.

The *FEE* consists of a master controller transputer, and its network engine of transputer processors. The tasks of the *FEE* span across two distinct phases: a parallel detection of edges, and a sequential segmentation of the edge pixels into line segments.

Detecting the Edge Pixels

The edge detection filter used by the *FEE* is the Canny filter which was described briefly in Section 2.2.2. A sequential OCCAM translation of the Canny operation from a C source program was already available and only some minimal optimisation of this code was attempted by the author. The major effort was concentrated on the management of its parallel implementation; for this, the *geometric parallelism* approach was selected, since the edge detection operation is a low-level task which must be applied equally to the whole image. These ideas which were covered earlier in this thesis, are once again valid here, and in fact the work on parallel Sobel edge detection for the Label Inspection problem will be used here too, to provide a similar analysis for the Canny edge detector. Note there is a complete contrast in the amount of computation the Canny detector requires when compared with the Sobel filter. Both the array (control-driven model) and farm (demand-driven model) networks will now be re-visited to determine a fast execution of the Canny filter. However, this will be brief since a full analysis of the networks was provided in Chapters 3 and 4. It would

also be desirable to provide comparisons with the Sobel filter results of Table 4.5, to show how the complexity of computation and size of data packets affect performance. In fact, since some more transputers became available for this part of the work, and to aid quicker analogy of Sobel/Canny results, the old Sobel filter results will be re-measured and produced for the increased size networks. Furthermore, the Sobel filter implemented here will use the square root of the sum of the squares of the horizontal and vertical gradients as the value of the gradient magnitude. Why though use the Canny edge filter when the Sobel has already been implemented and tested? The importance of the *FEE* as a robust precursor to the tracking stage has already been indicated. This necessitates the employment of a more precise edge filter such as the Canny, in preference to the Sobel. Another importance of this choice will be reported when the discussion turns to segmentation of edges into lines.

Array Size	Sobel Data Routing	Sobel	Canny Sub-Image	Canny Data Routing	Canny Total
1x1	-	1.649	268x268	-	13.740
2x2	0.296	0.462	140x140	0.361	3.303
2x4	0.335	0.378	140x76	0.457	2.098
4x4	0.373	0.375	76x76	0.550	1.374

Table 6.4: Execution times for Control-Driven Model on 256x256 images with corresponding Canny sub-image sizes for shown transputer array sizes

In the control-driven model, the *reverse-feed distribution* scheme is used, and the image is spread across the array and each sub-image contains an area overlapping with its immediate neighbours, as in Figure 3.2. In the Sobel operation, it was shown previously that the size of this overlapping border area is 1 pixel. For Canny, the area size is determined through the value of σ which is used to form the Gaussian mask. A value for σ of 1.6 leads to a 13x13 mask. Thus, the length and the width of the sub-image must be increased by 12, using 6 pixels on each side of the area. The Canny process will then produce results only for the required area of the real sub-image, as shown by solid lines in Figure 3.2, and uses the pixel information from the overlapped border when it requires to look in that area. The results of the control-driven model are shown in Table 6.4 and Figure 6.6, for single transputer and 4, 8, and 16 processor array configurations. The corresponding sub-

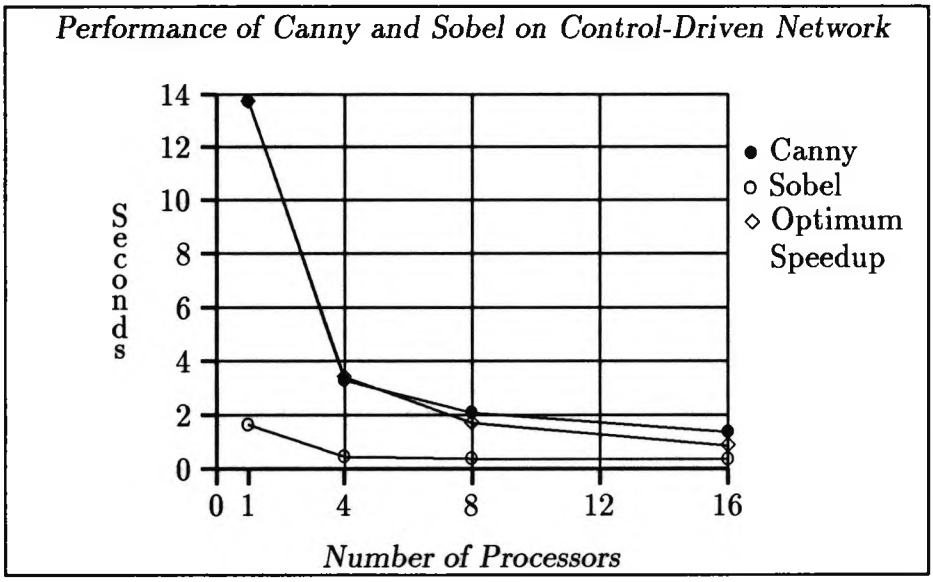


Figure 6.6:

image sizes are also shown for the Canny. The execution times are for images of 256x256 pixels and they include the time spent on the distribution and collection of data. The timing for this communication of data is also provided separately in the Data Routing columns. The performance of the Sobel operator, and the optimum linear decrease in processing time for Canny, are also shown for comparison. Speed-up factors for the control-driven model are shown in Table 6.5, with efficiency percentages provided in brackets. Please note that these show the speed-up of T800-25MHz processors over a single T800-20MHz implementation.

Array Size	Ideal Speed-up	Sobel Speed-up	Canny Speed-up
1x1	1	1.00 (100%)	1.00 (100%)
2x2	4	3.57 (89%)	4.16 (104%)
2x4	8	4.36 (55%)	6.55 (82%)
4x4	16	4.40 (28%)	10.0 (63%)

Table 6.5: Speed-up table for Control-Driven Model on 256x256 images

For the Sobel operator the amount of computation is very small and most of the processing time is spent on the communication alone. Notice, the heavy communications bottleneck built at the top-left trans-

puter in the array, disguises the processing time of the Sobel operation as the size of the array is increased. In contrast, the processing speed of the Canny reduces satisfactorily by the addition of extra processors.

No. of Farm Processors	Sobel 32x32(34x34)	Canny 32x32(44x44)	Canny 64x64(76x76)
1	1.157	21.065	15.149
2	0.582	10.633	7.893
3	0.404	7.172	5.477
4	0.316	5.468	4.266
5	0.279	4.403	3.307
6	0.276	3.689	3.058
7	0.276	3.176	2.648
8	0.275	2.823	2.453
9	0.275	2.553	2.102
10	0.275	2.316	2.063
11	0.275	2.148	1.937
12	0.275	1.987	1.849
13	0.275	1.891	1.715
14	0.275	1.784	1.715
15	0.275	1.656	1.715
16	0.275	1.632	1.715

Table 6.6: Execution times for Demand-Driven Model on 256x256 images

The performance of the demand-driven model was examined for different task packet sizes of 32x32 and 64x64 extracted from a 256x256 image. To allow for the necessary edge border data, the actual data packet sizes were 44x44 and 76x76 respectively. Still, these resulted in 64 and 16 packets in total, again respectively. The results for the increasing number of farm processors (up to 16) are shown in Table 6.6, and in Figure 6.7. Although deteriorating, some improvement is still observed right up to 16 processors for the 32x32(44x44) sub-image implementation. This occurs since there are a large number of data packets and even after the processors nearest to the master transputer have stocked themselves with work packets (i.e. one to work on and one waiting in their buffer), some work packets still remain to reach the processors furthest away. This is unfortunately not true for the fewer work packets that exist in the 64x64(76x76) implementation, so after the addition of the 13th processor no benefits in execution time can

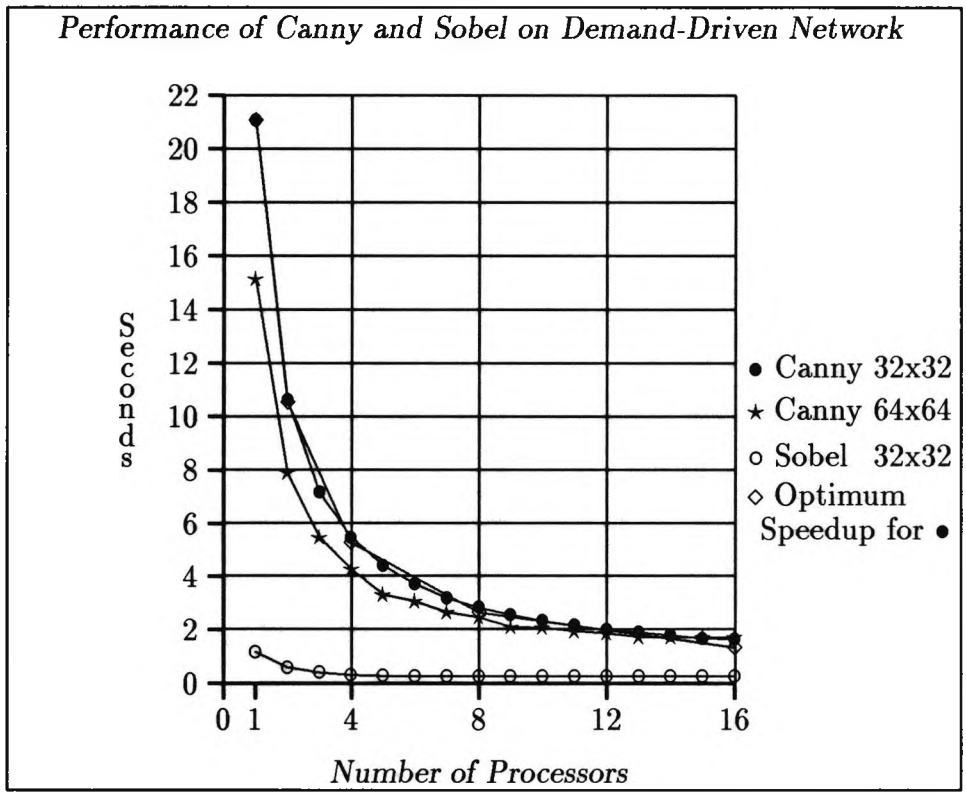


Figure 6.7:

be achieved. Nevertheless, up to the 13th processor, the 64x64 implementation puts in a better performance since the computational and communicational loads are better spread. In comparison, for the Sobel operator, for which the 32x32 sub-image is used due to its better performance compared with other sub-image sizes as shown in Table 4.5, there are as many work packets as the 32x32 Canny. However, since the algorithm is considerably less elaborate, the time to process a packet is much less than the time required to transmit a new packet to a slave processor and the addition of extra processors is ineffective since the shortest path processors become free quickly and eventually manage to consume all the packets. The speed-up and efficiency results for the demand-driven network are shown in Table 6.7. These show the performance matched against a single transputer farm implementation. The efficiency of adding more processors for a computationally intensive algorithm such as Canny in comparison to a Sobel is evident from the results. Also, the difference in the efficiency rates for the 32x32 and the 64x64 Canny show that as more processors are added the 32x32 implementation can spread its load at a very high rate, whereas the 64x64 is already fairly well balanced.

No. of Processors	Ideal Speed-up	Sobel Speed-up	Canny 32x32 Speed-up	Canny 64x64 Speed-up
1	1	1.00 (100%)	1.00 (100%)	1.00 (100%)
2	2	1.99 (99%)	1.98 (99%)	1.92 (96%)
3	3	2.86 (95%)	2.93 (98%)	2.77 (92%)
4	4	3.66 (92%)	3.85 (96%)	3.55 (89%)
5	5	4.15 (83%)	4.78 (96%)	4.58 (92%)
6	6	4.19 (70%)	5.71 (95%)	4.95 (83%)
7	7	4.19 (60%)	6.63 (95%)	5.72 (82%)
8	8	4.20 (53%)	7.46 (93%)	6.18 (77%)
9	9	4.20 (47%)	8.25 (92%)	7.21 (80%)
10	10	4.20 (42%)	9.10 (91%)	7.34 (73%)
11	11	4.20 (38%)	9.81 (89%)	7.82 (71%)
12	12	4.20 (35%)	10.60 (88%)	8.19 (68%)
13	13	4.20 (32%)	11.14 (86%)	8.83 (68%)
14	14	4.20 (30%)	11.81 (84%)	8.83 (63%)
15	15	4.20 (28%)	12.72 (85%)	8.83 (59%)
16	16	4.20 (26%)	12.91 (81%)	8.83 (55%)

Table 6.7: Speed-up and efficiency table for Control-Driven Model on 256x256 images

By comparing the implementation of the two models it can be stated that although some improvement is observed in additionally extending the demand-driven Canny (32x32) network, the cost/efficiency ratio dictates that given this many number of processors it would be more economical to employ the control-driven model (as depicted in Figure 6.22(B)). The control-driven model performs better at all 4,8 and 16 processor configurations, with the farm model capable of performing better than an 8 processor array configuration at 12 processors for the 32x32 Canny implementation, and at 10 processors for the 64x64 Canny implementation.

Some further analysis and discussion will be presented in the concluding chapter of this thesis.

In general, the results presented here compare well with other Canny realisations. Bottalico et. al. [BSI90] report 39.3 seconds, and 6.0 seconds for execution on a single T800, and a network of 16 transputers respectively, but they do not specify their test image size. Rygol et al [RPBK90] report 5.5 seconds for a 512x512 image on 24 transputers.

Grouping the Edge Pixels into Edge Tokens

This is the second phase of the tasks of the *FEE* and itself consists of a number of stages. These tasks are presently executed sequentially, and on a single T800 only (which is also the *FEE* controller and interface to its outside world.)

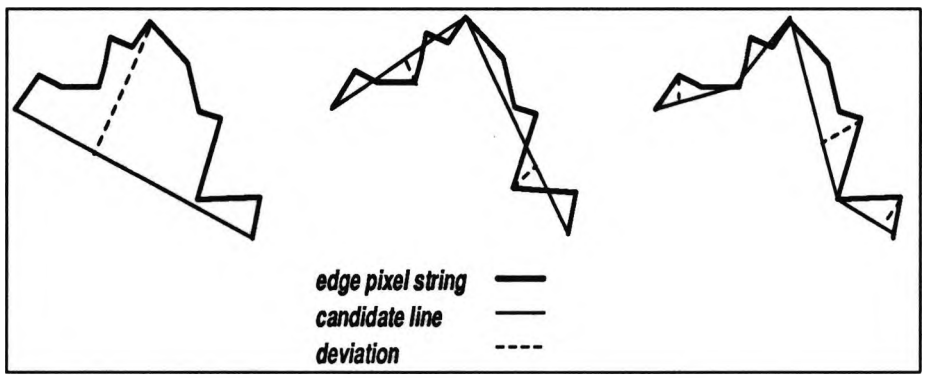


Figure 6.8: Three levels in polygonal approximation of an edge list

The initial stage consists of the segmentation of edges into lines after a technique introduced by Lowe [Low87], and extended by Rosin and West [RW89]. There now follows the operation which has been implemented in this study. Once the resultant Canny image has been returned from the network, the edge pixels are scanned and grouped together in strings, producing a a number of lists of connected edge pixels. (Notice, that had Sobel filtered edges been used for this stage, a thinning operation would have been necessary, with the processing introducing an extra level of inaccuracy. Thus, the use of the Canny proves its benefit at this stage. However, it may be that a Sobel operation followed by a thinning application, take less processing time than a Canny operation. At the end, it is the accuracy of the results that wins over other considerations). Let the connected edge pixels be called space curves. These are either open or closed curves. Open curves start and end with only one neighbouring pixel, whereas closed curves have two neighbours at all points and are detected by following the curve until the start pixel is re-encountered. The whole process is similar to chain-coding in an image of single-width edges.

Following the formation of the edge lists, a polygonal approximation of the lines must be found. Lowe's work is followed, whence each list of edge pixels is hypothesised as being a straight line passing through its

end-points. A list is then segmented into two by determining its point of maximum deviation The same process is then applied recursively on each of the two lists. The recursive process is discontinued on the satisfaction of one of two conditions through predetermined thresholds. The conditions are when a line segment is less than 4 pixels[3] in length and when the deviation is less than a certain number of pixels, which for this implementation it is set to 4 pixels, and has been determined empirically.

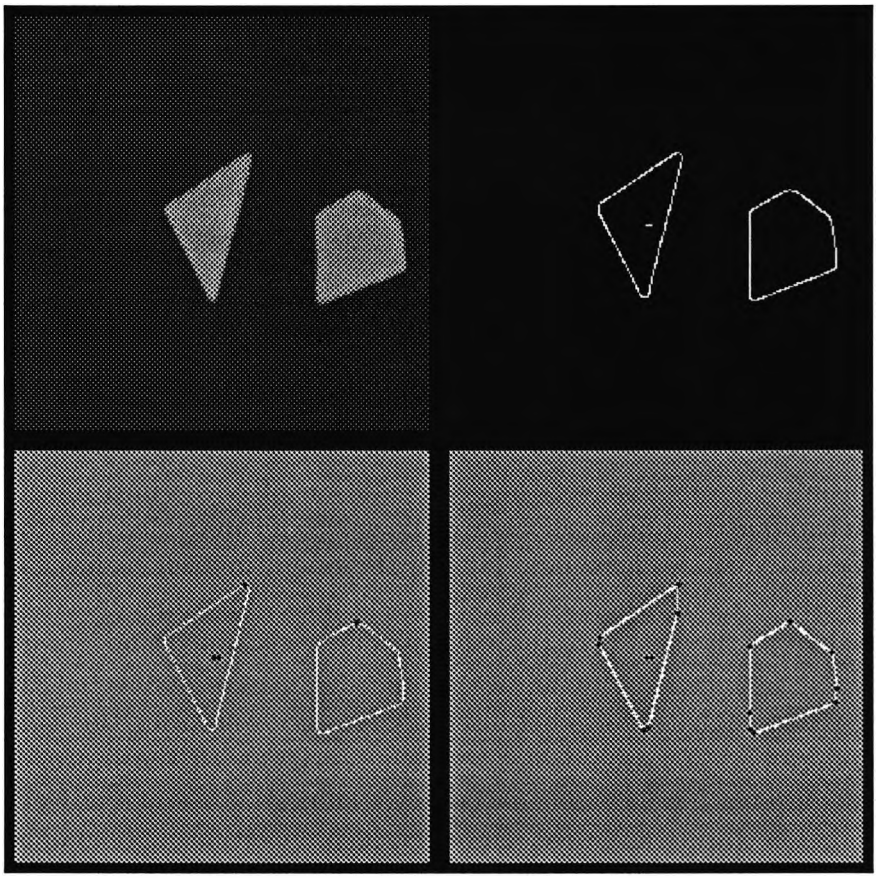


Figure 6.9: *FEE* analysis on a simple scene: (top-left) Original scene, (top-right) Canny filtering, (bottom-left) Grouping of pixels into connected strings, (bottom-right) Segmentation into lines through recursive algorithm.

The recursive procedure produces a multi-level tree where each level describes a finer approximation of the list of edge pixels than that

[3]The smallest line with non-zero deviations is a line of 3 pixels in length. It is undesirable to retain a three pixel line if the deviation is zero.

Detected pixels	Detected strings	Detected Line Segments	Line Linking	Total
470	3 (1.548)	15 (0.053)	0.008	1.609

Table 6.8: *FEE* **processing results for simple scene**

above. As the recursion unwinds, if any of the line segments handed up to the higher level (closer to the root) are more significant than the line segment on the current level, then they are retained and passed to the next level up , otherwise the line segment at the current level is returned. The measure of significance is determined after Rosin and West's report which is defined as the ratio of the maximum deviation divided by the length of the line segment. Figure 6.8 shows an example of line fitting at different levels of recursion.

Detected pixels	Detected strings	Detected Line Segments	Line Linking	Total
4434	171 (1.833)	360 (0.851)	0.812	3.496

Table 6.9: *FEE* **processing results for busy scene**

It has already been mentioned that OCCAM does not support dynamic memory allocation and therefore does not allow recursion. Instead, the problem must be handled using iterative techniques. The problem of coding the segmentation algorithm in OCCAM was overcome by implementing a stack, to store and retrieve memory variables at each level of the multi-level tree in the line segmentation algorithm.

Results of *FEE* processing are shown in Figure 6.9 and Table 6.8 for a simple scene, and Figure 6.10 and Table 6.9 for a busy laboratory scene. The following description applies to both. The top-left corner picture is the original scene, followed to its right by the image obtained after the Canny filter operation. A count of the number of pixels are provided in the first column of each table. Bottom-left image represents the list or strings of connected pixels. These are marked by a "+" pattern at their start and end points. Column two of each table represents the number of strings found in each image, with the processing time in brackets. The final image in the bottom-right represents the edge segments as determined by the recursion procedure described above. Again start and end points of the line segments are marked. For example, notice

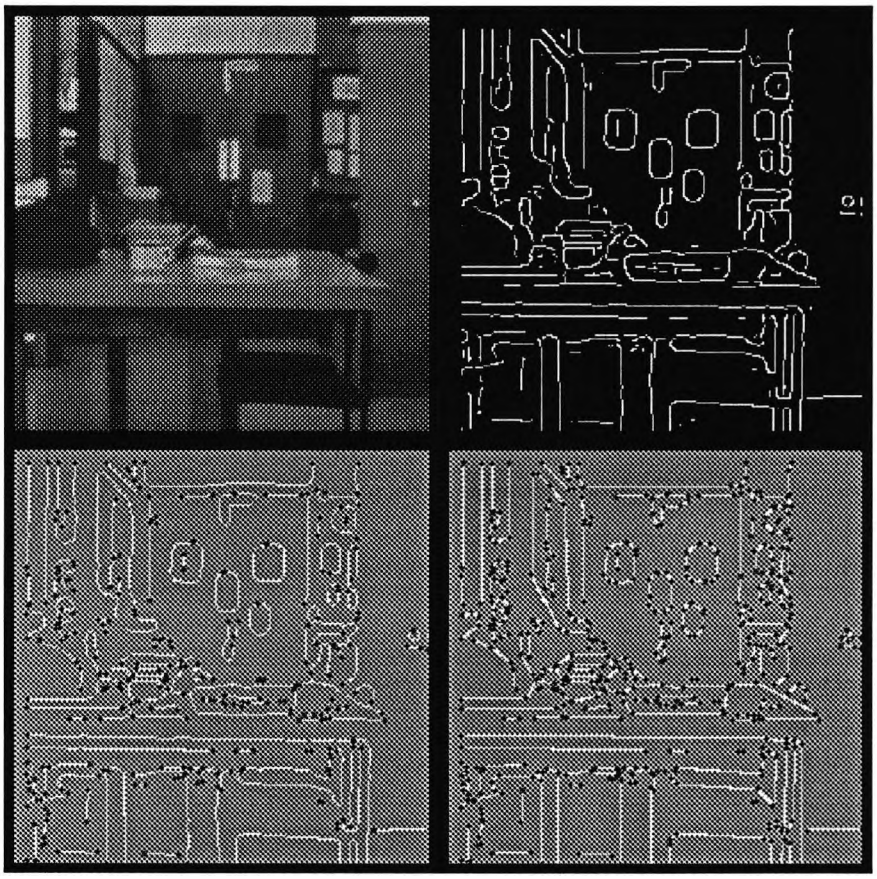


Figure 6.10: *FEE* **analysis on a busy scene: (top-left) Original scene, (top-right) Canny filtering, (bottom-left) Grouping of pixels into connnected strings, (bottom-right) Segmentation into lines through recursive algorithm.**

the window-panel on the laboratory door, whose curvature leads to a multitude of small line segments. The number of line segments detected is provided in the third column of the table, followed by its processing time.

The final stage of processing consists of the line linking phase which is a more elaborate version of the algorithm developed for the label inspection process mentioned in Section 4.7. For this implementation, this comprehensive neighbourhood-proximity process is used to bridge possible short breaks along edge length. The execution timings in the fourth column of the tables refer to this process, followed finally by the total processing time spent on the post-Canny processing requirements.

The grouping of the edge pixels into strings is the most costly process, since each pixel in the image needs to be examined at least twice, once searching for open lists and once searching for closed lists. To reduce the total processing time a number of methods could be attempted. The emphasis would be to use the processing power of the *FEE* network processors to achieve a better speed of computation. The task of grouping the pixels into lists could be split across the transputers, by allowing each transputer to follow its Canny process by applying the string formation algorithm while it still holds its sub-image data-packet. Then it would be the onerous task of the controller to piece all strings together into a set of continuous lists by reference to the results of each sub-image. The recursive process of line segmentation could be implemented using a technique for applying recursive algorithms on pipelined transputer networks [Red88], however, this is also a non-trivial problem, and may result in longer computation time due to the extra communication load. The use of the line linking process has both its advantages and disadvantages. On the one hand, it aids to reduce the number of lines involved and may help to overcome some cases of drop-out from one frame to another. On the other hand, due to its dependence on a number of thresholds, it can lead to erroneously joined-up lines. Its use has therefore been implemented on a switch which can be selected at system run time. Naturally, the *FEE* processing time is reduced when line linking is switched off.

Two methods for an immediate reduction in the complete processing time of the *FEE* have been introduced. Firstly, in order to consider the most salient of the edges in the scene, those pixel strings which are fewer in length than a pre-determined number of pixels, are ignored. This reduces the task of line detection and line linking, and also drastically lightens the load for the tracking engine. In contrast, it can also lead to inaccuracies in continuously segmenting robust edges for the tracking engine. Results due to the enforcement of this process are presented later in the next section. Secondly, it is noticeable that the initial, parallel Canny detection, and the subsequent post-processing are decoupled tasks. Thus, the two stages can be overlapped, such that in its steady state the *FEE* system will require as long as the computation time of its longest stage to provide the *TE* with new tokens.

This is more clearly visible by considering the parallel tasks of the *FEE* controller as simplified and summarised in Figure 6.11. Not all buffer processes are shown to maintain clarity. One important buffering process which allows the overlapping of the edge detection and edge segmentation processes resides between those processes, as shown. The engine transputers run processes similar to those shown in Figure 4.1.

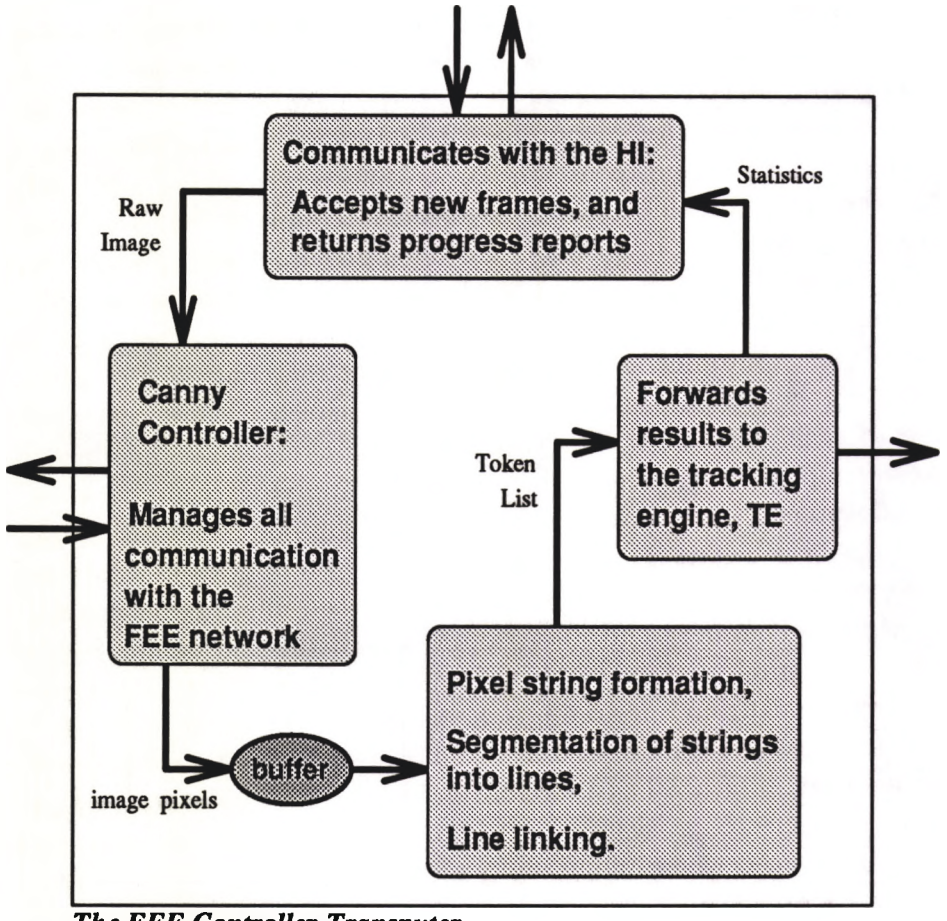


Figure 6.11: Overview of the parallel tasks of the *FEE* controller

Another detail not shown in Figure 6.11 is the fact that the *FEE* controller is linked to each of the *HI* processors separately, such that it can report graphical results to one (the Harlequin), and textual reports to another (the B004).

Of course, other methods, such as the Hough transform, could be used for edge segmentation. However, a Hough transform based on a simpler edge detection process, for example the Sobel operator, provides much poorer edge tokens than the procedure described above. Using a more sophisticated technique, such as the Canny, would lead to an even greater computation load and processing time. Also, in the initial stages of this implementation the $\rho\theta sHT$ was employed, but it provided inconsistent results, thus incapacitating the tracking stage. This

was predictable; the errors in the registration of an edge, associated with the distance of the edge from the centre of the $\rho\theta sHT$ sub-image are quite considerable, and lead to inconsistent description of the edge from frame to frame. Added to this the fact that there will be some disparity caused due to camera motion, it is deducible that the $\rho\theta sHT$ would not suit this application. Another shortcoming of the $\rho\theta sHT$ for this application is its inherent feature that different sub-images sizes suit the detection of different lengths of lines (Section 4.4, [Dav90]). Clearly, in a typical motion detection scene, lines of arbitrary lengths would be encountered. The *GHT* [BB82] was found to require extensive processing and the performance further suffers for the high communications rate necessary for the global Hough space. Still, MATCH has a completely modular design, and other techniques for evaluation and implementation could be considered for the *FEE* as and when necessary.

The final result of the *FEE* processing is made available to the *TE* via the link connection between the *FEE* and the *TE* master controllers as shown in Figure 6.22.

6.5.4 The Tracking Engine

The tracking engine as a sub-unit of MATCH, provides the opportunity for applying the model described in Section 6.4. Again, the decoupling of the problem is emphasised, since the *TE* may at this stage be viewed as a black box whose only expectation is to receive a *token list* from the outside world, in return for which it will supply another *token list* as part of the current state of the structures in the scene, which must be interpreted as so by another stage of processing.

The transputer has been cited as an ideal tool for MIMD processing by many authors [Pag88, WP89, DEH89]. This is now examined and evaluated with application to the problem of token correspondence.

In determining the type of multiprocessor configuration to use, the n-linear pipe, the array, the hypercube, and the tree networks were narrowed down as the most likely candidates to fit this purpose. The intention was to keep matters fairly simple and straightforward, but most of all to choose the most appropriate configuration given the limited number of processors available.

The processes in the parallel model follow their predictions by requests to receive all the edge segments found in a certain area of the image in order to perform their matching process. The requests are spatio-temporally independent, but process and processor dependent. Since

the criteria that any processor must be able to execute any task cannot be met, the processor farm computational model (described in Chapter 4) was dismissed as unsuitable and thus so was a linearly arranged configuration.

The array processor configuration or a model of higher dimension is appropriate for implementation if inter-processor communication is to exist as defined in the tracking model. For a transputer-based system, such configurations can provide the most efficient basis for shortest path addressing from any processor to another (Section 3.6). It will be shown later in section 6.5.4 that inter-processor communication is not necessary, due to the choice and nature of token-tracking requirements. Therefore, 2D mesh, cubic, or other similar configurations of even higher order are unnecessary, rather than unsuitable, for this implementation. However, it must be brought to notice that when inter-processor communication is necessary, the network transputers would need more complex message handlers and consume extra processor cycles in routing messages to other processors since there are only a limited number of link connections. Moreover, there simply were not enough transputers available to implement a suitably-sized array or cube network. To add to this further, a desire for investigating an alternative configuration also played a major role.

In fact, the use of the configuration adopted for this implementation can be justified as the most practical in any case. As in any application, the minimum amount of communication along with the possibility of taking the shortest route are the most desirable factors in a message-passing, multiprocessor system. Given that the communications in the tracking model have fixed target addresses, a network topology where the path between the source and the target is unique and short would be ideal. A tree configuration was selected to achieve this where there is a unique and direct path between the tree root (or controller) and the target process(or), and each node in the network need only hold a small table of addresses for its own children. The bandwidth of the system is also increased since a message can be propagated from root node to target using $O(log_b N)$ communications in a network of N processors with b branches at each node. This compares with $O(N)$ propagations in a linear topology. For clarity in understanding and presentation, the implementation discussed here only uses two branches at each node, although three are available given the number of links on each transputer. However, a step up to three branches would be a simple case of reconfiguration of the system and alteration of each processor's local address-table. The correspondence analysis remains unaffected, and given the very small number of processors used in this implementa-

tion, it is guess-estimated that a substantial performance improvement would not be observed by using a tertiary tree.

Various elements of the tracking engine are now considered using direct reference to the issues discussed in the model. To start with, problems faced at the lowest level, i.e handling processes, will be discussed, where it will be seen how they affect the path of the implementation. But before continuing, the reader will be reminded by providing a simplistic overview of the task at hand. In short, there will be a number of processes, running on a smaller number of processors, each of which will be responsible for a token in the image. The processes are expected to provide a progress report on the token, through communication with the central controller and *Blackboard* system.

Process Manipulation

It was mentioned early on in this thesis that the OCCAM language is sometimes described as the assembly language of the transputer, and that its support for concurrency (process invocation, communication and synchronisation) provides for simpler concurrent program design. However, the OCCAM compiler for the transputer does not support dynamic memory management, thus memory and processes must be declared at compile time. One of the effects of this restriction is the inability to create processes to track new tokens. This limitation is solved by declaring for each processor, a number Q of inactive, static processes at compile time where each process q,

$$q \in (0, 1, 2, ..., Q-1) \tag{6.12}$$

is *activated and deactivated* instead of *created and terminated* during a system run.

All processes start at a deactivated state, and a deactivated process initialises its environment and waits to be activated as a completely fresh tracker process. An alternative method would have been to load a node processor with new processes from the host processor [Inm88b], but this method was considered too costly in management and communication terms for this application. Also dynamic creation of processes is valid when an "unlimited" number of tracking processes are desired. For this work, the system wishes to select those tokens that are specifically salient, and thus a static allocation of a limited (but still large) number of tokens is desirable and intentional, and it allows for a much faster and more efficient implementation with a higher likelihood of

keeping nearer to real-time. These ideas are further emphasised in the next paragraph.

New processes are activated when new tokens appear in the image, and working processes simply continue tracking until they lose their track or the corresponding edge segment disappears from view. A process may lose the track because of noise or inconsistent edge data from the edge extraction stage. An edge segment may disappear because of *occlusion* or simply because it leaves the camera's field of view. In such situations the process may continue tracking by using one of numerous techniques in its attempt to re-establish contact with the "lost" edge segment. These may be continuous imaginary tracking, active search of neighbourhood areas, simple wait for reappearance and more other expensive methods. The approach that has been adopted, and it proves the most efficient for this application, is described as follows. An image of a real scene will generally consist of many edges. The intention of this particular application is to provide a basis for a unified interpretation of all the constituent features of a scene. Loss of tracking information on some parts of a scene will not affect the continuity of information regarding the overall nature of the scene. Therefore, it is chosen for a process to terminate immediately upon losing an edge segment. Should this happen when the edge segment leaves the field of view, then termination has not been in vain. If occlusion has occurred, the edge may either never reappear or it will show itself at a later frame when it can be issued to a new process for a fresh track. Finally, if the edge segment is simply lost by its tracker, regardless of the cause, the actual line segment can be issued to a new process in the next frame for a fresh track. For both previous cases, the new information will be incorporated immediately into the scene interpretation process working elsewhere. Loss of track is a more common occurrence due to inconsistent data from the edge extraction stage. Thus creation and termination of processes is a common occurrence and so *activation and deactivation* of processes a cheaper, more appropriate implementation of the model requirements.

Nevertheless, the issue of occlusion will not be left as such, since the *TE* can still deal with it, only it is proposed here that it is more efficient to deal with it in the manner described above. When occlusion does occur, Kalman filtering results of the tracking procedure will be exploited to continue the tracking of a lost token. This can be simply achieved by replacing the unobtainable measurement parameters of a token with the estimated parameters. Thus tracking is said to have continued through an imaginary phase. If new measurements become available after a predefined number of frames, the token is said to have reappeared and the tracking can continue for real. Otherwise, the process is allowed to

give up and deactivate . This occlusion analysis and tolerance process has been programmed into the system as an optional requirement which may be switched off or back on. Results to support this are provided in Section 6.6. It should be expected that in a very simple scene with only few edges,

$\Longrightarrow$ consistent tracking will occur,
$\Longrightarrow$ the tracking processes would have a long life,
$\Longrightarrow$ there would be a very low rate of activation and deactivation,
$\Longrightarrow$ and the system would, ideally, remain balanced.

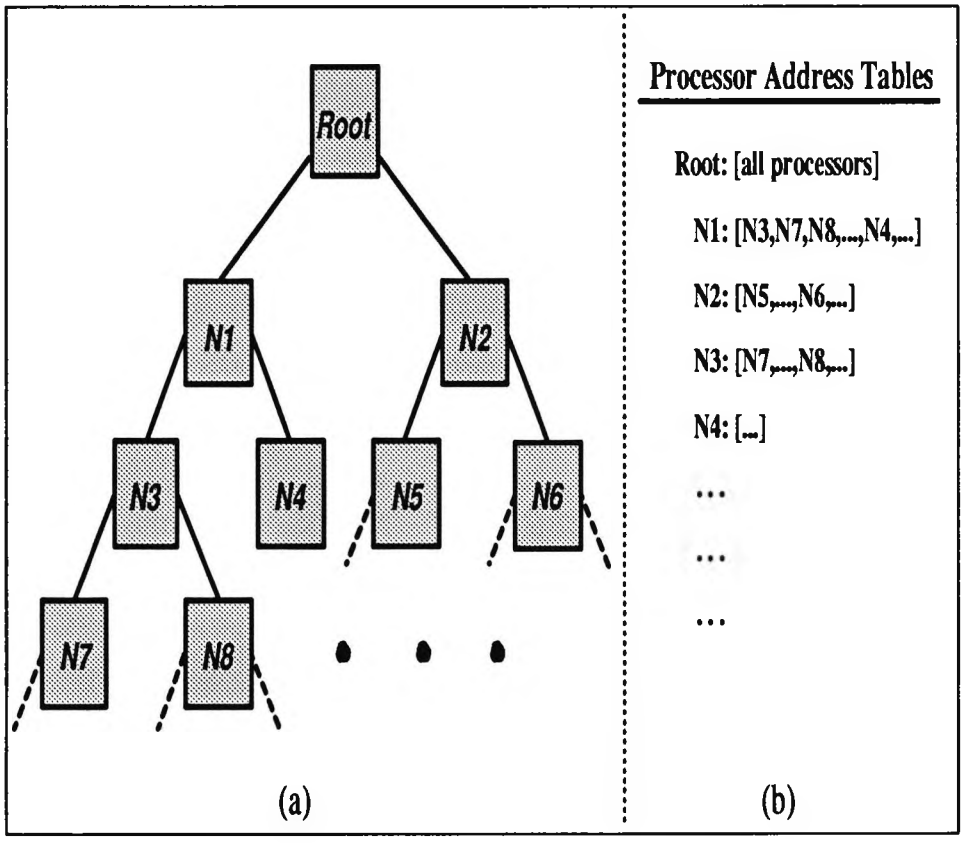


Figure 6.12: (a) A typical tree network, (b) address table for each tree node

However, in a real, noisy scene, a token's track life may not last very long. This could arise through inconsistent edge extraction and/or significant acceleration in the scene. Short track life would lead to a fairly high rate of activation and deactivation in this implementation resulting in inter-frame instances when system load is unbalanced. But this is soon overcome when new tracking processes are activated to either

continue with lost tracks or start on fresh edge segments. This approach works well in practice and eliminates the need for inter-processor communication to off-load tracking processes onto other processes. This too is further illustrated in Section 6.6.

The system is implemented on a binary tree network where the root processor of the tree network becomes the system controller and the nodes are the slave processors (Figure 6.12). Each node is said to be the parent of the lower level nodes branching from itself, which are in turn referred to as its children. Each processor node in the tracking engine contains a number of parallel processes which are the process tasks, the buffers, the multiplexor and demultiplexor processes, and the message routing tasks of the processor. These and the communication channels are shown in Figure 6.13. Figure 6.14 provides a pseudo-OCCAM outline of the parallel processes. When a data packet is received, the feed-router process of the processor determines which processor it belongs to. By looking up its own private address-table, the packet is then forwarded down the appropriate branch towards the child processor. This is illustrated in a sample tree network in Figure 6.12. The diagram also shows a sample address table. Note that each table is further sub-divided to contain child processor addresses relevant to each branch.

When a router process finds that a message has arrived "home", it passes it to a buffer/demultiplexer process which acts as a distributor of data to the collection of tracker processes currently active on the processor. Similarly, a buffer/multiplexer process is always active and ready to collect matching and tracking messages from the tracker processes. It passes the results back to a router process which points them towards the processor's parent processor along with other processors results arriving from all immediate child nodes. The data coming into the buffer/demultiplexor is by nature queued. However, for data going out of the processor, they are received from the multiple tracker processes by the buffer/multiplexor through a fair ALT arrangement to ensure an equal service to the processes. They are then buffered (queued) for output, via a bleed-router process, to a higher level processor in the tree. The fair ALT schemes follow the principles described earlier in Section 2.4.7. The bleed-router also has the task of accepting the messages and data from the child processors of the processor it is executing on.

Each process has access to processor-local libraries which contain the tracking algorithm routines. Each can execute the full tracking algorithm, the standard Kalman filter or the α, β Kalman tracker, as

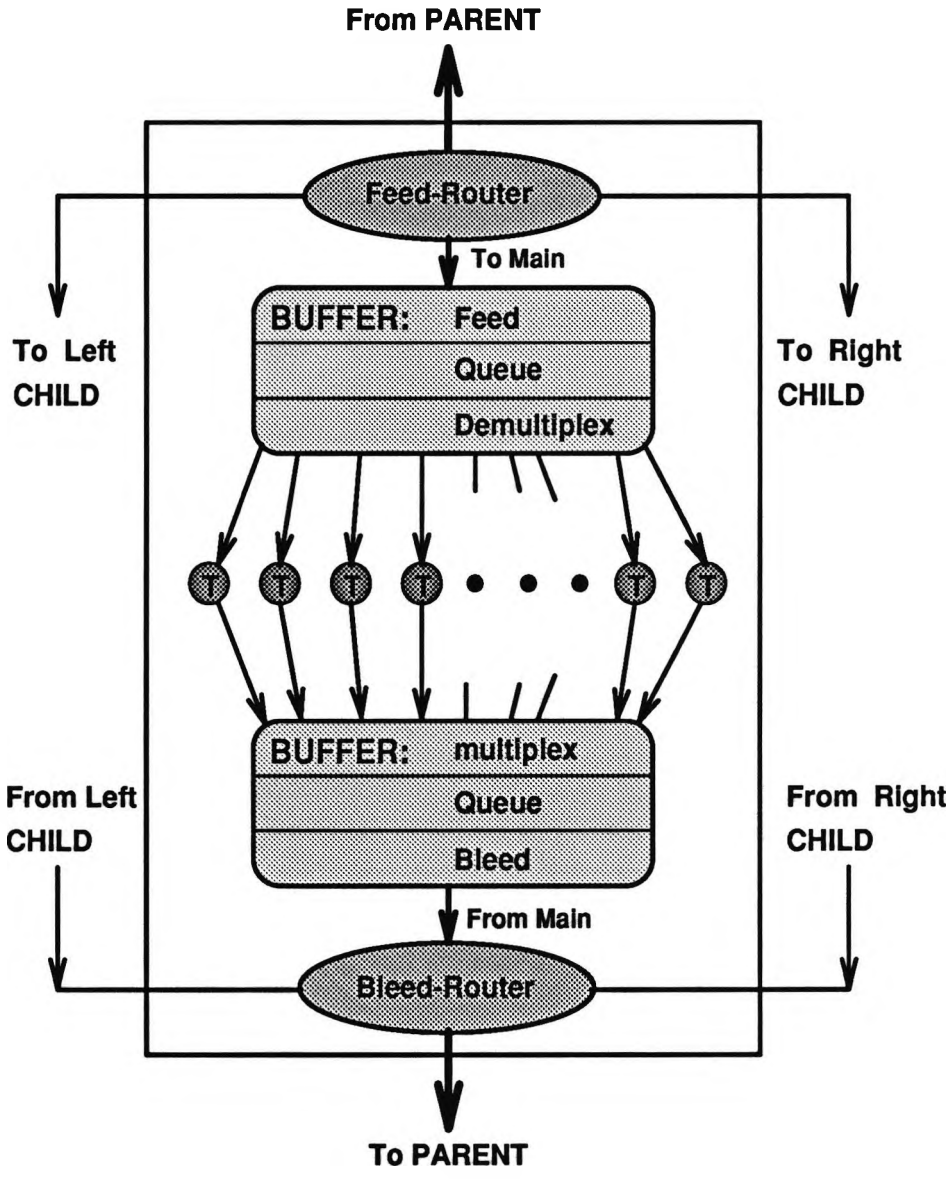


Figure 6.13: Parallel processes, including tracker processes, executing on each processor of the tracking engine

```
— Router processes are run in parallel, but at a higher priority
— level to that of the main task process.
— The channels of communication for the routers are shown
— in brackets.
PRI PAR
  PAR       — High priority processes for main communications
    ... feed.router(From.PARENT,
                    To.CHILD.Right,To.CHILD.Left,To.MAIN)
    ... bleed.router(To.PARENT,
                    From.CHILD.Right,From.CHILD.Left,From.MAIN)
  PAR       — Low priority processes of the main process
    ... buffer messages and data as they arrive on TO.MAIN channel
    ... demultiplex for consumption: send out on process.in[q] channels
    PAR q = 0 FOR Number.of.PreSpecified.Processes   — i.e. Q
      tracker.process(q,process.in[q],process.out[q])
    ... multiplex for collection: receive on process.out[q] channels
    ... buffer messages and output on FROM.MAIN
```

Figure 6.14: Parallel processes of a TE processor (the low priority processes can be read through as though depicting the *sequence of events*)

appointed. (The overall algorithm for each process was presented in Section 5.8.) Each process could also exploit further parallelism, since it has to process four independent, uncorrelated parameters per token, as shown in Figure 6.15. This provides an opportunity for mixing in some *geometric parallelism* into the MIMD model presented here.

However, let it be assumed that Q tracker processes are currently active on any single processor in the system. There is then a potential increase by $4Q$ in the number of schedulable processes. It takes the transputer 13 processor cycles to start a new process and approximately 20 cycles[4] at best to re-schedule it (as a low priority process) each time its turn comes up. This is clearly inefficient for the local-processor performance, and processor cycles could be spent more wisely by executing the filters in sequence. The alternative would be to communicate these tasks to other transputers, but that would be a phenomenally expensive approach. However, this opportunity could be exploited in say a finer-grain, but tightly-coupled multiprocessing platform, such as the IUA, where less communication would be necessary.

Even further parallelism is attainable per process. The decoupled equa-

[4]Transputer instructions: ldl I; ldl W; stnl-1; ldl W; ldpri; or; runp

```
— Each process assigned to a single token can exploit further
— parallelism by invoking additional parallel processes for independent
— Kalman filter calculations on token parameters.
PAR
  ... Kalman filter for x_m
  ... Kalman filter for y_m
  ... Kalman filter for θ
  ... Kalman filter for l
```

Figure 6.15: Possible parallel sub-processes of a token tracking process

tions 5.28 and 5.29 of the α, β Kalman tracker can be evaluated separately and concurrently. Another level of parallelism arises at the time of computation of the Mahalanobis distance. This issue was discussed in Section 5.6.

The constitution of process messages and data will be discussed in a following section. However, the general semantics of their action is reviewed, partly here, and partly later in the comments on the system controller. Processes send and receive various types of messages or requests back and forth to the system controller. For brevity, only those sent from the processors are touched on here. These are itemised below,

- **request**: a request for the provision of a list of tokens to be found in a particular (search) area of the image following some tracking analysis,
- **request**: a request for the latest frame's time-stamp,
- **message**: the results of the current frame where either a match is found and indicated so that the system controller can update the scene flow model, or where no match is found. Depending on the occlusion analysis switch discussed earlier, this could mean the termination or the provisional continuation of the process,
- **message**: a short message for the verification of deactivation.

It is emphasised that the general framework of the messages implemented here are by no means intended to be rigid and unchangeable. These may be increased or decreased in number or context depending on the algorithm in mind. The system's overall modular design should be able to handle other variations. This is specially achievable through

the use of the OCCAM CASE statement on variant-protocol channel communications. For example, facilities exist to allow a process to request the time-stamp of the next available frame. This is particularly useful in the case of the object detection investigation presented in Section 6.3 where the time-stamp was necessary in determining the next position of the object, assuming constant velocity, since the previous time-stamp. This facility is merely by-passed and ignored by the current Kalman filter-based implementation. In the next section on the system controller, it will be seen how various processes are geared to handle and process varying tracker process requests.

System Controller (SC)

In its implementation, the system controller behaves almost as specified in the model. It is the organiser and the conductor of the tracking engine.

The *SC* is constituted of three parallel processes at the highest level of its process family. These are enumerated and demonstrated in Figure 6.16, along with their communication channels.

The *HI* handler is used to return general status and functioning information about the *TE* to the interface network. The *FEE* handler continuously receives extracted edges from the *FEE* for time-stamped frames which are subsequently buffered and fed to the *TE* handler. Its task is therefore to execute an elementary buffering procedure.

The tasks of the *TE* handler are now reviewed. This process consists of a number of parallel sub-processes which collectively execute the main functions of the *SC* (Figure 6.16). One of the main tasks of the *SC* is to ensure an equal amount of computation for each processor in the *TE* network. In this work both *static* and *dynamic* load balancing are achieved. Static load balancing is accomplished at the bootstrap stage of the MATCH system when a specific number of tasks are allocated to each processor. At this stage, the *SC* receives the very first edge segments and distributes them amongst the network processors according to Equation 6.9. The actual assignment may take place in a number of ways, for example in a round-robin fashion or by assigning groups of tokens at a time. The former is the method implemented here and a simple pseudo-OCCAM simulation of the code is shown in Figure 6.17. These assignments are registered in the *Blackboard* using data structures named the *process.table* and the *balance.table* as described in the aforementioned figure. Dynamic load balancing is normally achieved by redistributing processes between heavily loaded and lightly loaded

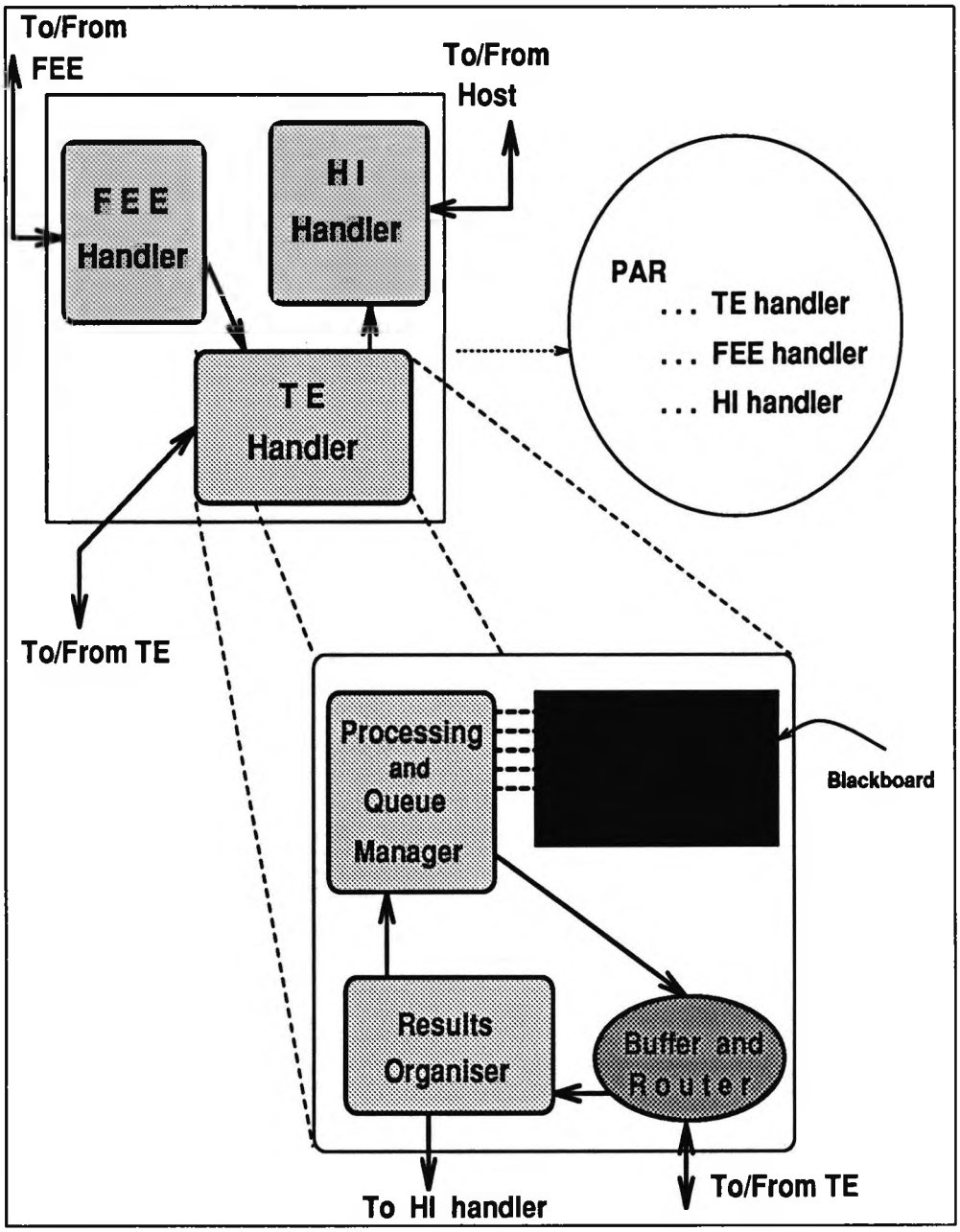


Figure 6.16: Highest level of processes in the System Controller, and detail of the TE handler

```
— Each object is assigned to a process on each processor.
—
— Processes are identified in the process table as follows:
—  process.table[i][0] := transputer address          (starts at 0)
—  process.table[i][1] := process.no on transputer   (starts at 0)
—  process.table[i][2] := activation status           (0,1 or 2)
— These form the address of token no. i, where
—   PROCESS.ACTIVE       is 0
—   PROCESS.TRANSIENT    is 1  (during occlusion)
—   PROCESS.INACTIVE     is 2
—
— System is kept balanced by use of the balance.table
— which ensures that no processor runs more processes
— than another (except by 1). Typically,
—  balance.table[transputer address] := no. of active processes
— The balance.table is initialised to zeros at start.
—
— Let tree.nodes be the number of transputers in tree. Let T
— be the transputer address.(Cyclic between 0 and tree.nodes-1)
—
T := 0
SEQ i = 0 FOR total.no.of.edges
  [header.details]INT header :
  SEQ
    — set up process
    process.table[i][0] := T
    process.table[i][1] := balance.table[T]
    process.table[i][2] := PROCESS.ACTIVE
    balance.table[T] := balance.table[T] + 1
    — header contains edge segment details and token ID., which
    — is also the colour code used for visual results display.
    ... formulate header
    To.TREE ! Boot.New.Token;process.table[i];header
    IF
      T = (tree.nodes-1)
        T := 0
      TRUE
        T := T + 1
```

Figure 6.17: A round-robin assignment of tokens to processors

processors. An alteration on this theme is managed here which will be explained soon, but first let some of the stages involved be examined which happen once the bootstrap phase has been completed.

So, consider a time during the *run* stage some frames later. Processes return their predictions and requests for the next frame via messages. Some processes will profess via their predictions that they have lost the track and that they have deactivated. Other processes will state that they intend to continue tracking and require a new set of observed tokens. These messages are initially parsed and then organised for further action by the *Results Organiser* process, which incidentally, contains buffer sub-processes to queue the incoming requests. The *Results Organiser* process initially analyses the incoming messages and requests, sending a report (for one or a group of tokens) to the *HI* process. Both deactivation and continuation requests are then sent on along the appropriate channels for other processes to take care of them. Continuing processes are of two groups as itemised in the review of the processes earlier on. Some may be reporting their find, and some may be requesting a new search area. So, these, along with the deactivation requests, are reported through separate channels. However, since the *Blackboard* is not implemented as a shared data object (and rightly so, given the OCCAM message passing philosophy), all deactivation and continuation messages and requests are accepted by the *same* process which is the only process with read and write access to the *Blackboard.* Yet, these requests are sent along separate channels which provides a groundwork for a shared-access *Blackboard* implementation, if necessary.

The requests are received on separate channels by the *Processing-and-Queue Manager* under an ALT control. This is the main process of the *SC*, and here is a brief summary of its tasks,

- Deactivating processes are marked on the *Blackboard*, where they will be used for reassignment when new tokens are discovered,
- Processes requesting further data on observed tokens are queued,
- Results from those processes reporting their correspondence match are accepted and used to update the scene flow model,
- When free of the above actions, the *Processing-and-Queue Manager* continuously monitors the *process-service-queue* and services the requests, in a FIFO manner. The service comprises of the gathering, the grouping, and the dispatch of a *token (sub-)list*, consisting of all those edge segments whose end-points fall in the

predicted region of the image as specified in the body of the request message.

These actions are depicted in Figure 6.18. Other channels are also serviced by this process, for example, a channel for providing the next time-stamp to a requesting process.

```
— The process monitors incoming messages and requests on
— a high priority ALT branch, itself consisting of a
— multi-branched ALT construct. Thus, when nothing is
— incoming, the servicing of the requests is continued.
—
PRI ALT
  ALT
    receive.token.update ? signal; message
      ... update scene model
    request.new.data.set ? signal; message
      ... queue process for attention
    request.deactivation ? signal; message
      ... set process status to PROCESS.INACTIVE
    request.new.time     ? signal; message
      ... send new time-stamp
  TRUE & SKIP      - Nothing else to do.
    ... attend to the process-service-queue.
```

Figure 6.18: An overview of the *Processing-and-Queue Manager* process

When all the processes have finally provided their match results pertaining to a particular frame (identified universally by its time-stamp), any remaining edge segments that are unaccounted for, from this frame, will be issued to new or deactivated processes. The task is triggered as a sub-function of the *Processing-and-Queue Manager* and carried out by reference to details held on the blackboard. Thus the *SC* controls the load balance of the processors using the principles defined in the model (see section 6.4.5). The state of affairs at the *boot-up* stage was discussed earlier through Figure 6.17 by applying Equation 6.9. During system *run*, when the necessity for issuing a new process arises, the *balance.table* details are checked to find the processor N_i with the minimum number of active processes according to Equation 6.10. Using the processor's ID, it is referenced in the *process.table* to find the first member q of its process set that is in a state of deactivation. This process is then charged with the responsibility of tracking the new token.

In the case where no minimum is found, a new (never activated before) process on the first and nearest processor is activated. Therefore, dynamic load balancing is achieved by redressing the discrepancy between heavily and lightly loaded processes during system *run*. This approach is preferable to redistributing processes between transputers, since it avoids the necessity of inter-processor computation, and is more practical as a result of the fact that tracking assignments occur commonly due to both new tokens in the scene, and old tokens with failed tracks which must be redesignated as new.

The *SC* uses a simple rule to reduce the possibility of duplicated tracks. It is a fact that no duplicated tracks will exist at the *bootstrap* stage. Later in the *run* mode, since the *SC* has control over which edge segments are to be delivered to requesting processes for their matching stage, it can simply refrain from supplying to a requesting process those edge segments already identified by another process as a match for its own edge segment, thus reducing duplication risk. However, if some edges are already supplied as possible candidates to more than a few processes then the possibility of duplication is increased. It is arguable that given duplication on a low scale, it is harmless when large numbers of tokens are being tracked. A way of tackling duplication would be for extra processor-to-*Blackboard* bi-directional communication, or even independent interprocessor communication to share knowledge about the status of the system. Although this would allow for more intelligent processes, the extra load on the system is unfeasible, when a low duplication rate is acceptable. Also, duplicated processes are harmless in so far as they would ideally present the same information to the scene flow model. Furthermore, duplication is only more likely to occur in a complex scene with densely packed tokens in the image, and when edge segments with extremely similar features are so close to each other, so as to be able to fool the accuracy of the complete Kalman filtering and Mahalanobis distance matching process.

Some run-time results showing an extremely low-rate for duplicated tracks are presented in the results section later in this chapter.

The Blackboard (as part of the SC)

The *Blackboard* is the *TE* system's *knowledge base*, as (loosely) defined in Section 6.4.2. There are no strict concepts associated with blackboards; for example the one presented in this work and those in [TM86, THKS88] all differ from each other considerably. Thorpe et. al. [THKS88] *propose* a distributed process architecture using a hier-

archical blackboard structure, which is "scattered" amongst different modules in the system. The modules then need to synchronise to exchange information via a central database. (Tan and Martin's [TM86] blackboard was briefly mentioned in 6.4.2.) The *Blackboard* presented here, is in effect a collection of various data structures organised, accessed and updated by one major process. All access to the *Blackboard* by other processes must take place via hard-channel and soft-channel communications, but ultimately through the *Processing-and-Queue Manager* process.

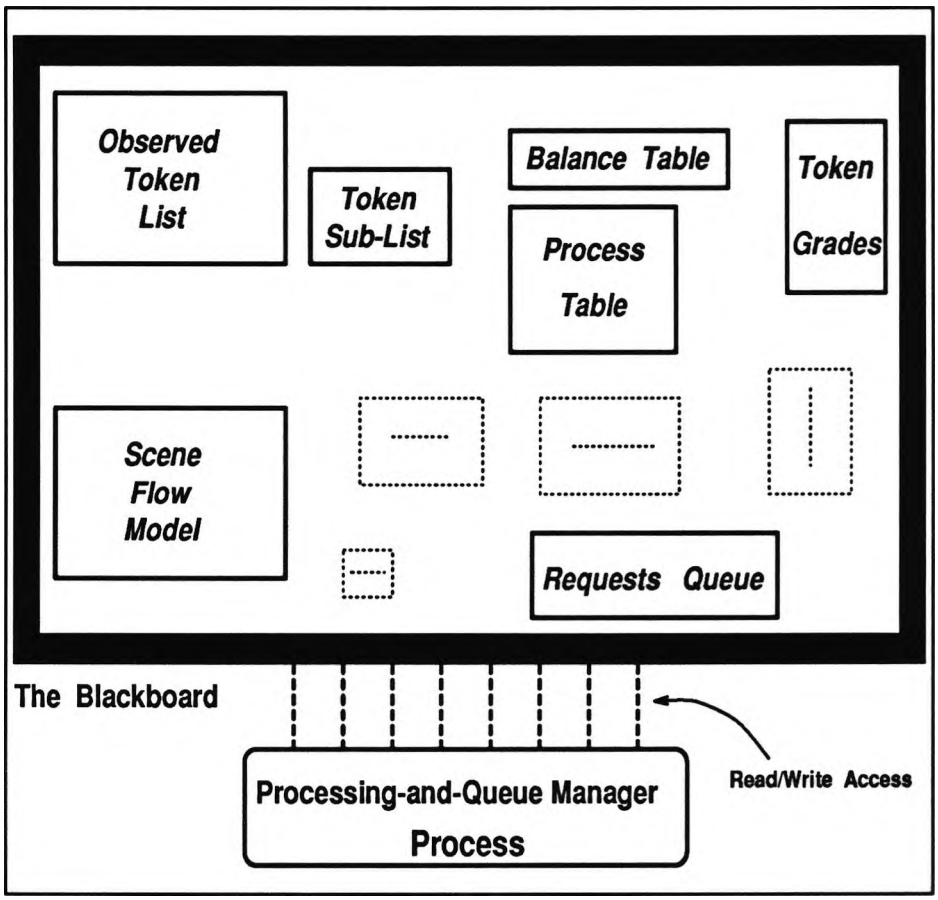


Figure 6.19: Some data representations on the *Blackboard*

The *Blackboard* has already been involved in some of the discussion so far. Here, the aim is the presentation of Figure 6.19, exemplifying a sample of its major contents. Some of these will now be described in short.

The **token.grades** data structure, assigns an *INITIALISE* flag to each

token as they are supplied by the *FEE* process. As the tokens in the frame are treated and the match results are returned, the *token.grade* structure is progressively updated with those for which a correspondence has been found. These are re-graded to *MATCHED*. This information aids the avoidance of duplication whereby the *Processing-and-Queue Manager* can disallow the supply of those tokens already *MATCHED* to tracker processes.

The **scene.flow.model** holds the latest information on the currently active tokens, consisting of lists of $[x_m, y_m, \theta, l]$ information.

The **observed.token.list** holds the latest set of tokens as extracted from the latest frame by the *FEE* network. These are grouped into sub-lists and dispatched to tracker processes when their search area requests have been processed. Each group will contain a list of those tokens whose both end-points fall inside the search area; this can easily be changed such that all edges with a minimum of one end-point in the search area are elected into the token sub-list.

The **process.table** holds an inventory of the address and identification of all the processes on all the system transputers, be they ACTIVE, INACTIVE or TRANSIENT.

The **balance.table** holds the number of processes ACTIVE or TRANSIENT on each of the transputers in the *TE* network. It provides an indexing mechanism into the **process.table** when the *SC* needs to balance the *TE* load.

TE Communications

(Some facets of the *TE* communications have also been already considered in the earlier parts of this discussion, such as those in Figure 6.13 and its accompanying comments.)

The model requirements regarding the communications are followed in full, but with respect to the nature of the implementation. The major loads in communication are between the *SC* and the outside world, which comprises of,

- import of extracted edge data from the *FEE* network,
- the export of *TE* results to the *HI* processors,
- the exchanges between the *SC* and the numerous tracking processes, which takes place via *en route* processors.

There is no communication between tracking processes.

Floating-form communications is achieved by using 1D integer or byte arrays (the fixed size of which is determined by the implementation algorithm) where a message of any length may be communicated as demonstrated in the skeletonised example in Figure 6.20.

The PROTOCOL statement defines a message consisting of an integer, followed by two pairs of values with each pair consisting of a size value succeeded by that number of components. The size value can be zero for one or both of the arrays. In the case when both sizes are zero, only the initial INT in the message is transfered, usually a tag value. A general description of the PROTOCOL statement may be found in [PM87]. The first integer array is always used as a way of passing essential housekeeping information, such as source process address or destination process address, consisting of a processor number and a process number. This manner of communication is adhered to at all stages of the *TE* network, where the principle part of the message (the second array) can be formed into an array of INT's. It is hoped that its generality can be applied for use with different token tracking algorithms. For example, one requirement of any tracking and prediction algorithm is the capability of examining any region of the image. In this implementation, image regions of *any* size may be passed by utilising the local processor power to pack the region from a 2D array into a 1D array which is then unpacked by the remote processor locally. This is specially important in transputer link communications where link activity can take place independently of CPU processing once the CPU has initiated it. In the object detection system described in Section 6.3, this facility was used to pass sub-images from the *SC* to the *TE* processes. By careful use of the RETYPE facility in OCCAM other types of data could also be passed through INT structures [PM87].

Figure 6.21 demonstrates the format of *some* of the messages used by the *SC* and the *TE* processes. These all have the *tag;message.size1; message1;message.size2;message2* format, where a *messageX* may either be a message or a request, and which may be made up of different items of information packed into an array of INT's. For some message types the contents of the message are also provided in Figure 6.21.

A *header* message contains the information necessary for the booting of a process with a new token's data, following an *activation.signal* sent earlier to that process. Requests are made by returning the proposed search area. Requests are satisfied by returning the list of tokens found in the search area, and so on.

Use of buffering in *TE* communications is of paramount importance to

```
— Variable length channel protocal definition
PROTOCOL TE.Messages IS INT; INT::[]INT; INT::[]INT
—
— Define channel iolink
CHAN OF TE.Messages iolink :
—
— A process using the iolink channel would follow this pattern:
— Define array. ID.TAG would be globally known.
[100]INT message  :
SEQ
  ... message array is set up here.
  iolink ! ID.TAG; process.address; (SIZE message)::message
— Each message would be recognised and dealt with by establishing
— the value of its identification tag.
```

Figure 6.20: Variable length channel communications

ensure that system performance is not degraded by waiting-to-communicate channels. For example the router of any *TE* processor must not wait to communicate with a certain tracker process while messages for the child processors are arriving and expecting to be routed through (Figure 6.13). So a buffer on any processor accepts and queues messages for the processor, allowing the router to route. It follows logically that data-routing processes must be run at a higher priority than other processes, a principle already emphasised in this thesis.

The issue of deadlock is always a detailed and important factor in the design of any distributed multiprocessing system. Buffering as a solution is not always enough on its own and fairly strict protocols may have to be applied to avoid the potential of deadlock. This can take the form of an acknowledgement or *handshaking* signal established between any two communicating processors. As an example, the *TE* uses a disguised form of this procedure in allowing processes to request the time-stamp of the next frame. This can act as both an acknowledgement and a useful piece of information for many tracking algorithms.

6.5.5 The Host Interface

The interface consists of two transputers both acting as hosts. They are T800-20MHz processors mounted on a B004 and a Quintek Harlequin frame grabber/buffer board respectively, with both boards free standing in the PC (For more details please see Appendix A). Figure 6.22 shows

the connections of the host processors to the *FEE* and *TE* networks.

The Harlequin is used to receive images from a camera in the real-time live mode. The images are then reduced in resolution from 512x512 to 256x256 using Equation 6.11, time stamped, and passed to the *FEE* for edge extraction. The Harlequin processor is also responsible for receiving the results of both the feature extraction stage, and tracking stage from the *FEE* and *TE* respectively. These are subsequently displayed on a separate monitor allowing a visual observation of the system performance. The processes of supplying images, receiving *FEE* images, and receiving *TE* token tracks, run in parallel and independently of each other. Only, the frame grabber's display memory is dual-ported such that extraction and tracking results may be written to the same area of shared memory.

The B004 host processor acts as the interface to the outside world by having access to the PC ports via the server program part of TIPS. It is also used to drive the user interface as before. As with the Harlequin processor, the B004 processor is also connected to receive progress reports and results from both the *FEE* and *TE* networks. Independent processes monitoring the inputs from these two networks pass the networks messages to a display process, which formats them and passes them to the PC server program for displaying on the EGA monitor. A typical display screen is shown later in Section 6.6.

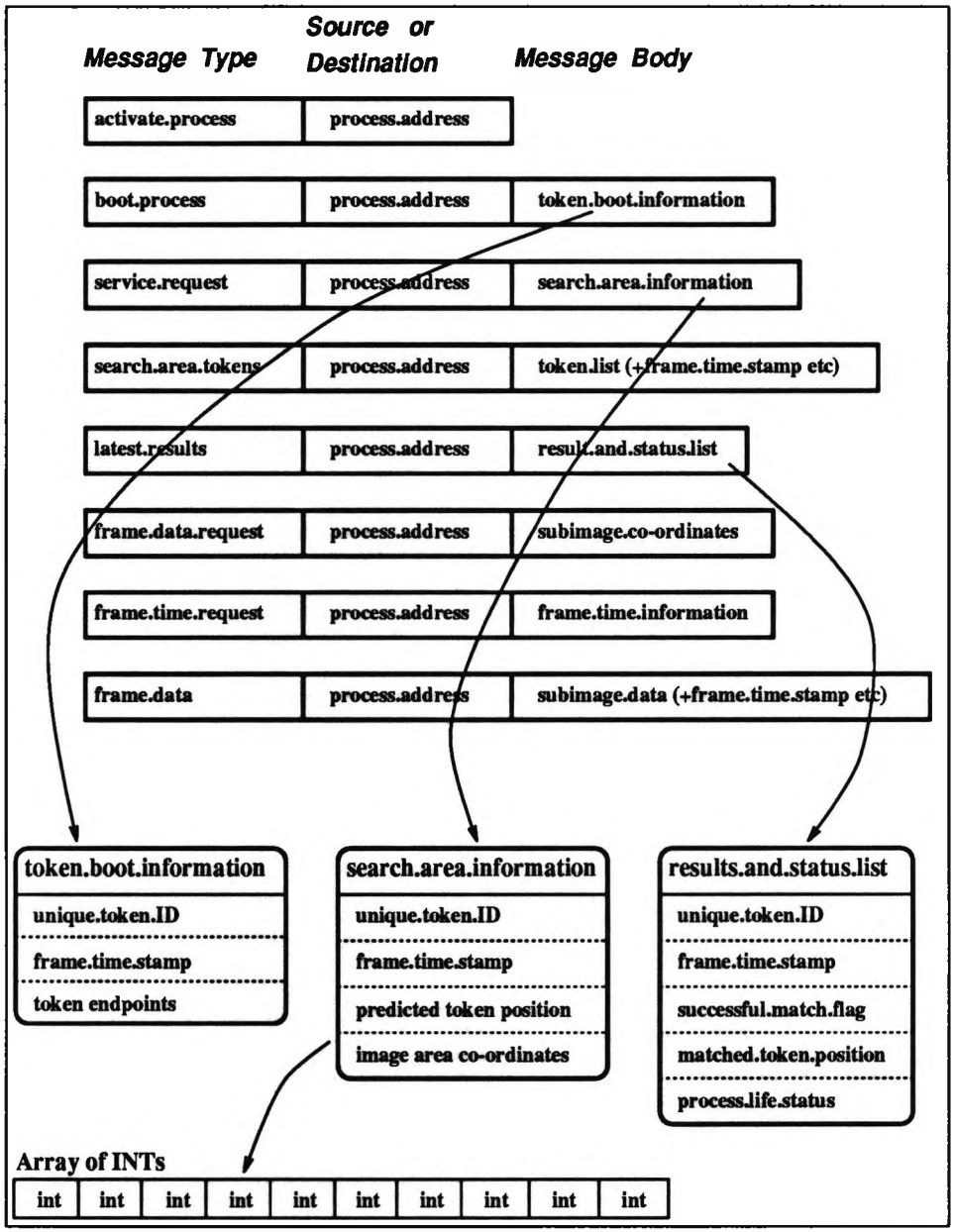


Figure 6.21: A hierarchical breakdown of some TE system communications

6.6 Tracking Results and Analysis

The results presented in this section were obtained using the set-up shown for MATCH in Figure 6.22 and Table 6.10, depicting the configuration and number of transputers used, and displaying the mix of SIMD and MIMD approaches employed by the system.

Network	No. of Transputers	Network Type
HI	2	singular connection
FEE	8	regular array
TE	10	binary tree

Table 6.10: MATCH: network configuration table

All the transputers in the system were T800s at 20 or 25MHz with standard 64 bit FPU and 4K on chip SRAM, and at least 1MB of RAM each. Thus, the system currently consists of 8 and 10 transputers for the *FEE* and the *TE* networks respectively. This provides a fairly balanced state between the two networks for a busy scene of over 100 edge segments, as the results will verify later. Since the number of edge segments supplied to the TE may be controlled, a balanced state may always be attained and the *TE* kept busy while the *FEE* processes the next image frame.

A series of experiments were conducted to distinguish the various characteristics of the system. Motion of the camera was attained by mounting a camera on a MAXTASCAN inspection and measurement platform. This machine contains a carriage which moves along a bridge mounted on the main frame. This allows an accurate and smooth motion with speeds of up to 12cm/s, controllable manually and dynamically. Pictorial and corresponding statistical results of the tracking implementation are provided in Figures 6.23, 6.24, 6.25, 6.26, Plates 6.4, 6.5, 6.6, 6.7, 6.8, and Tables 6.11, 6.12, 6.13, and 6.14. All execution times are in seconds, except when stated otherwise. It was mentioned previously that a unique ID is assigned to each edge token. This ID is in fact a colour value, selected from a circular scale with the *SC* as the implementor and manager, and is mapped against a Harlequin colour LUT when drawing the edge segment on the display monitor. The results in each figure are presented as four frames sampled at arbitrary intervals, with the frame number printed in the bottom right-hand corner of each frame in each quarter-image. The frames are in this sequence: top-left, top-right, bottom-left, and bottom-right.

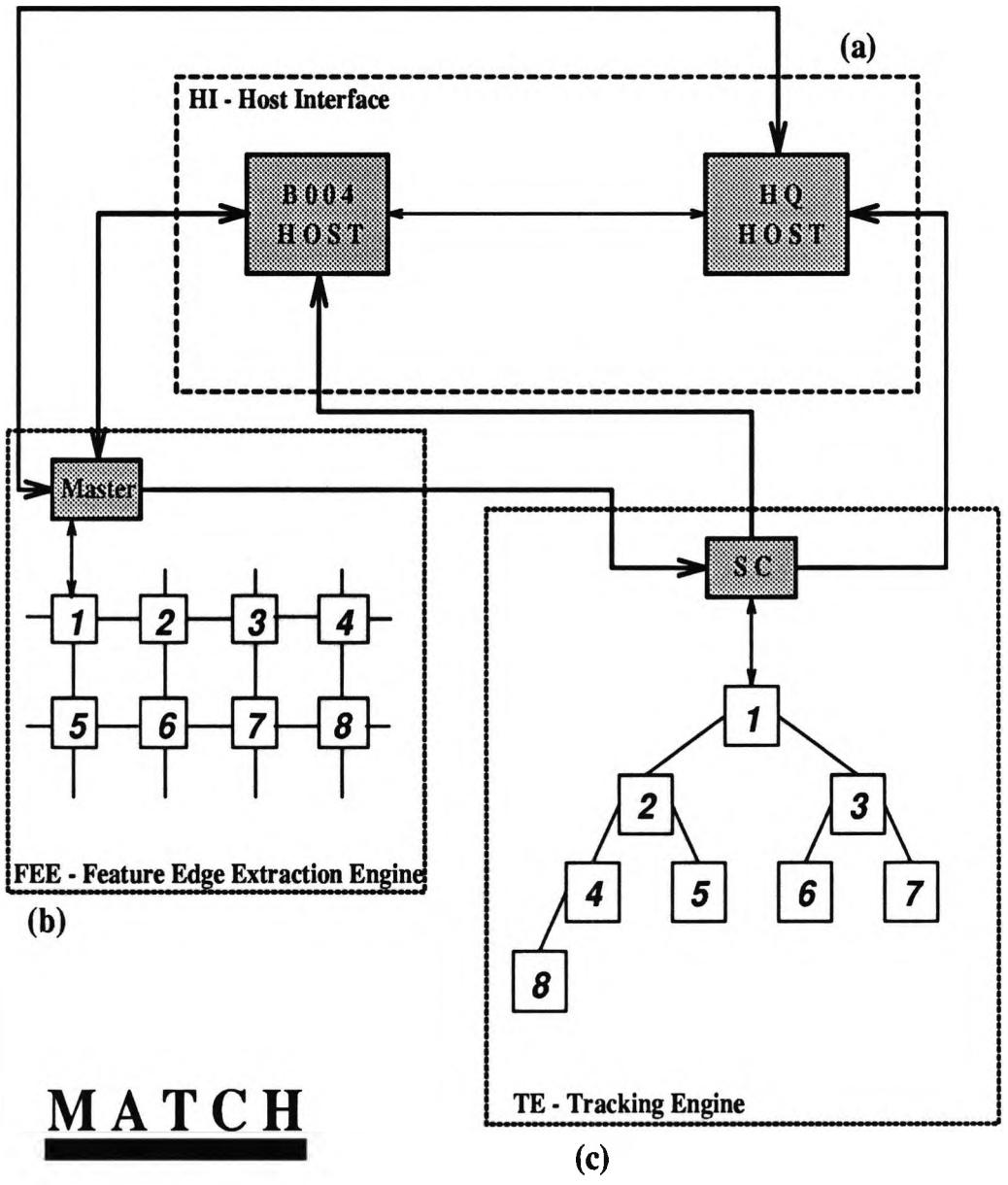


Figure 6.22: MATCH: Diagramatic outline of current system configuration. To keep the diagram simple, the C004 link switch and associated connections are not shown.

Frame Number	Number of Tokens	Matched Tokens	Failed Matches	Duplicated Tracks	Success Rate (%)
7	8	8	0	0	100
14	8	8	0	0	100
20	8	8	0	0	100
29	8	6	2	0	75

Table 6.11: Frame results for simple scene

Avg. No. of Tokens	Avg. time for SC	$Min - Max$ time/process	
		filtering	matching
8	0.27 sec	$136 - 281$ μsec	$138 - 585$ μsec

Table 6.12: Average frame processing time for simple scene obtained over a lengthy run

Initially, a simple, artificially constructed scene is presented to easily and comfortably convey the robustness of the tracking. Figure 6.23 shows the original image of the starting frame from a sequence of 29 frames. Plate 6.4 displays the result of the tracking sampled at frames 7, 14, 20, and 29. By studying them, it can be observed that following an unstable start, which is expected since the constant velocity is not yet attained, the tracking proceeds in a solid manner (with the camera moving from left to right). A loss of tracks for the red and light-blue edges is noticeable in frame 29. These were artificially induced by altering the speed of the camera motion, yet this only affected a comparatively few set of tokens. Perhaps rather expectedly, no duplicated tracks were observed given the clarity and simplicity of the scene. Table 6.11 shows the statistics obtained for the simple scene.

The average processing time for the SC in completely servicing a frame is displayed in Table 6.12. Note that on average, only the first eight processors of the TE would have been active with one tracker process each, compared with numerous processes for the two real scenes presented next. The minimum and maximum times spent by a process in Kalman-filtering and matching a token are also provided. This was obtained by monitoring the performance of every tracker process in the system. Note that the matching involves the Mahalanobis tests between the token being tracked and each token in the requested search

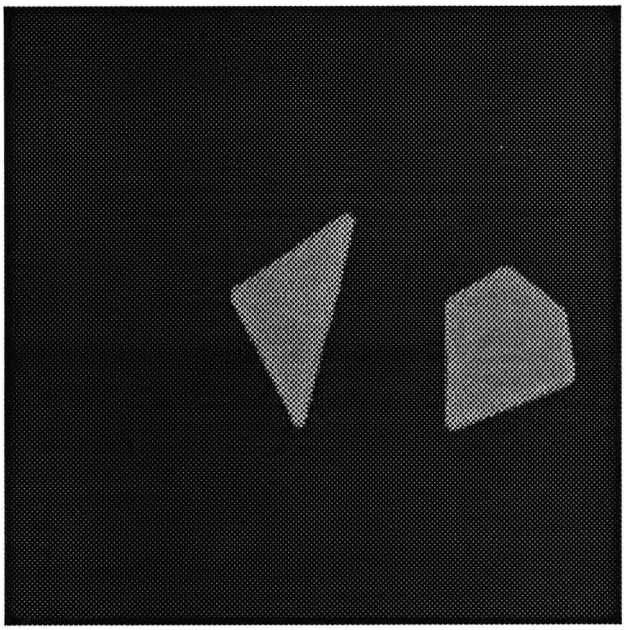


Figure 6.23: First frame of the sequence for simple scene

window (Please also see Figure 5.10). From Table 6.12, it is evident that for the worst case, when the processing time for an isolated token is taken into consideration, the complete process of filtering and matching is under 1 millisecond. Ideally speaking, near one thousand tokens could be tracked every second. However, the queue-processing and the communication load of the *SC* produces overheads that are unavoidable in a centralised control of a distributed message-passing system. More about the behaviour of the *SC* will be said a little later.

Plate 6.5 shows a case where the occlusion analysis option is switched on. Initially, notice that the object on the right is tracked smoothly through the whole sequence. The object on the left has a split edge on the right which by frame 9 has become one edge (the red colour) and is being tracked as one. Tracking goes smoothly up to frame 16, but between frames 16 and 26 all 3 main edges are lost by the *FEE* at least once. When they are lost, their associated processes still continue to track, but also new tokens are assigned to the *new* edges in the image. A little later, either these *new* edges are lost in turn, or the edge is reclaimed by the original Kalman filter, in which case the tracking is then continued by the old process using the old colour. This sequence of events may be clearly observed in frame 26 of Plate 6.5 for the light-blue edge which has reclaimed its token.

Frame Number	Number of Tokens	Matched Tokens	Failed Matches	Duplicated Tracks	Success Rate (%)
First Busy Scene (Plate 6.6)					
7	104	75	29	1	72.1
16	108	78	30	0	72.2
45	108	69	29	2	63.9
62	105	70	28	1	66.7
Second Busy Scene (Plate 6.7)					
5	85	65	20	0	76.5
23	82	63	19	1	76.8
39	84	66	18	2	78.6
56	84	66	18	1	78.6

Table 6.13: Frame results for real two example real scenes

In the two example real scenes coming up, some important points must be observed when viewing the results (which are obtained in both cases with the camera moving from right to left). Firstly, only *selected edges* are tracked; for example, those below a certain length are ignored. Therefore a higher rate of track loss is possible due to edge drop-out at the extraction stage. Secondly, the Kalman filter works "independently" of feature speed, though the examples here show only a small motion in order to allow performance over a large number of image frames. Thirdly, as the camera moves, new tracks will overwrite previously plotted tracks as coloured edges are traced on the display monitor. Thus, some tracks may look as if they were terminated, whereas they are merely *hidden*. Finally, the camera moves horizontally, therefore the reader should expect to notice vertical edges tracked better and more visible.

The first frame in the sequence of 62 frames for the first real scene example is displayed in Figure 6.25, followed with the tracking results sampled at frames 7, 16, 45 and 62 in Plate 6.6. The prominent orange and gray areas in the top right corner of frame 62 illustrate the idea of immediate reassignment of an edge segment to a new tracker process (hence a new ID and a new colour from orange to gray) when a track loss occurs. This may be observed elsewhere in Plate 6.6, but must not be confused with edge tracks overlaid by others. For example, the aforementioned gray area has overwritten the orange tracks visible in frame 45. The second example is represented via Figure 6.26 and

Plate 6.4: Frames 7, 14, 20 and 29 of a sequence for a simple scene, with colour-coded line segments displaying spatio-temporal continuity

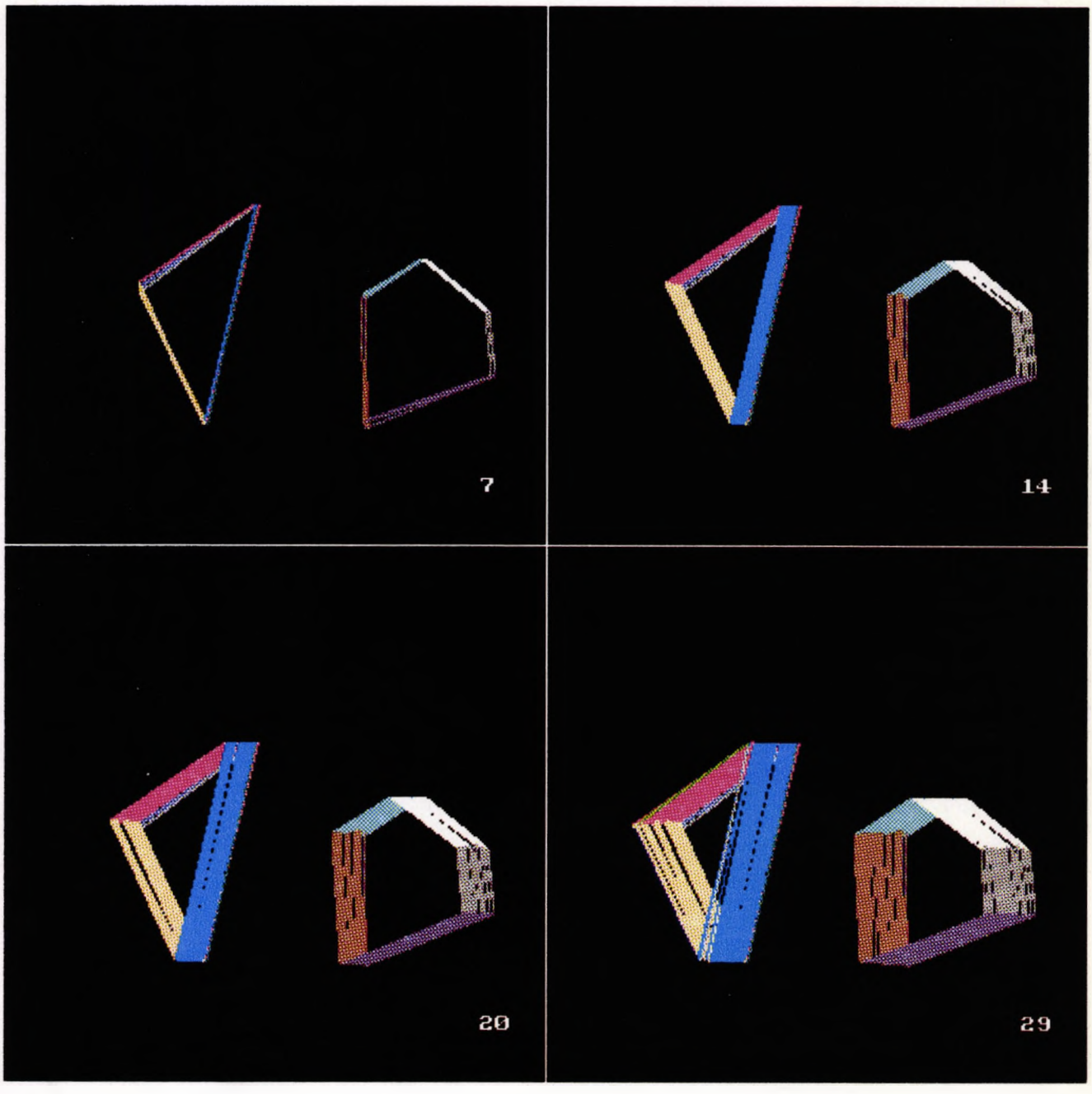
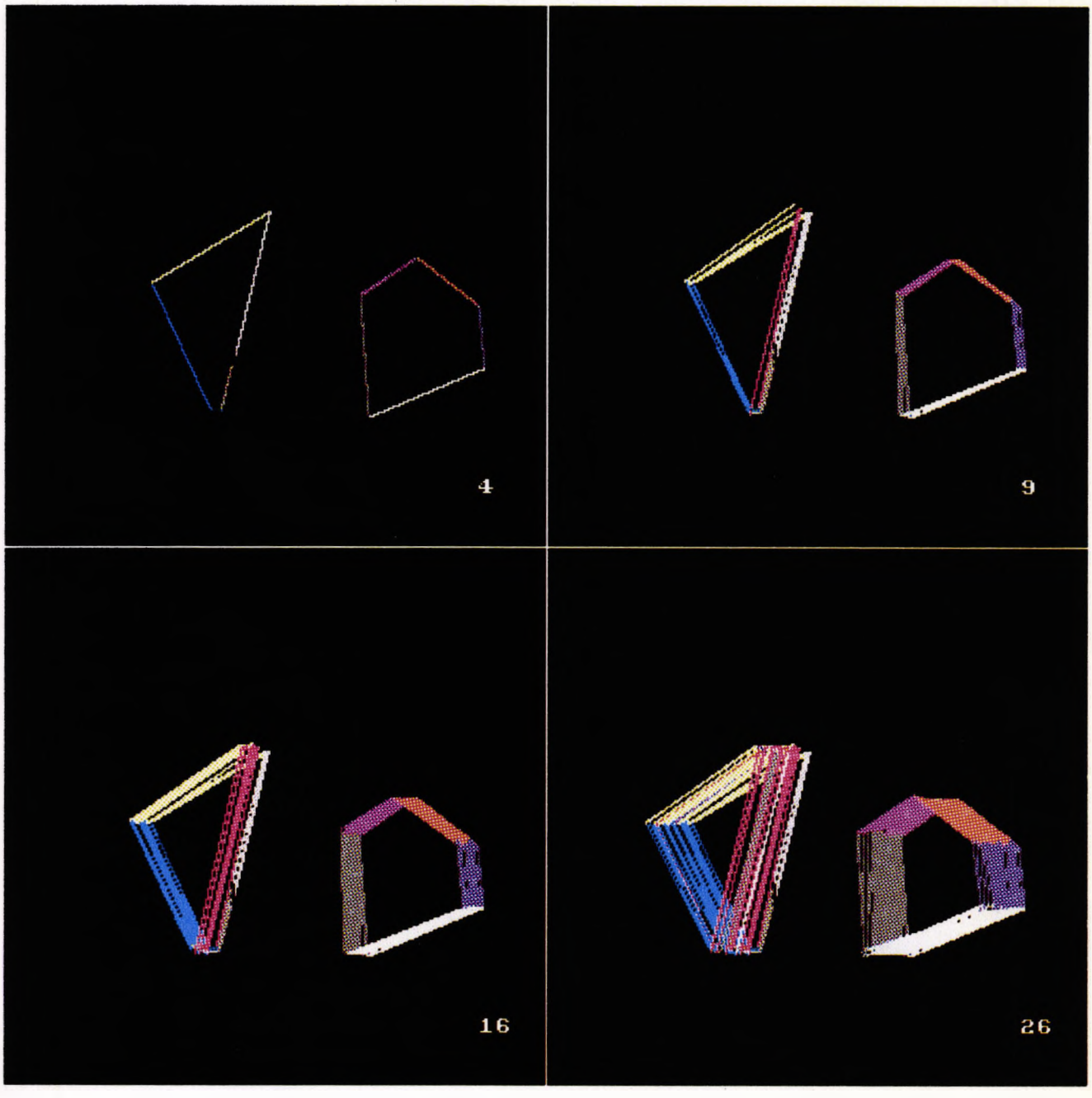


Plate 6.5: Frames 4, 9, 16, and 26 of a sequence for a simple scene with occlusion analysis, with colour-coded line segments displaying spatio-temporal continuity



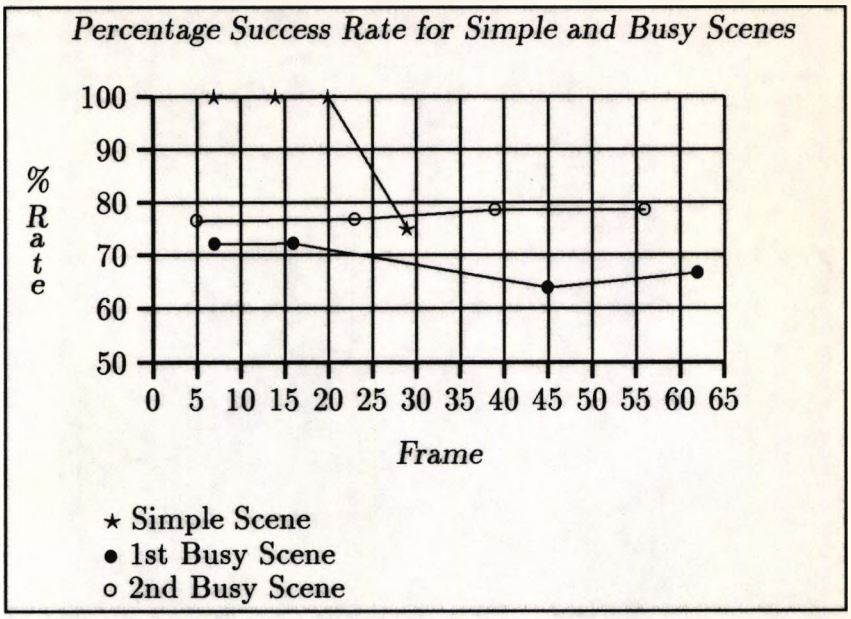


Figure 6.24:

Plate 6.7. This example contains many horizontal lines but some excellent tracking, with examples where little has been overwritten. These may be observed in the middle-right and the bottom-left of the scene (especially the light-gray tracks). The statistics for the two busy scenes are shown in Table 6.13.

The percentage success rate for each of the examples shown are plotted in Figure 6.24. A steady rate is observed in all cases. The large percentage drop for the simple scene is in fact an artificially induced loss of two tokens out of eight. The busy scene with less tokens has a higher success rate due to the higher chances of finding a match. The number of duplicated tracks was found to be very low in all the experiments conducted. In fact, to such a limit that their effect on the overall performance may be regarded as negligible.

Occlusion analysis was not switched on for the real scene examples.

Table 6.14 displays the *SC* and minimum to maximum average *TE* process execution times for the filtering and matching stages of a process. The execution times include some time spent on a bare-minimum amount of extra communication for reporting system progress. Again, the complete ideal processing time for a token is about the order of 1 millisecond, such that almost 1000 tokens could ideally be tracked every second. Furthermore, that would be on each transputer! However,

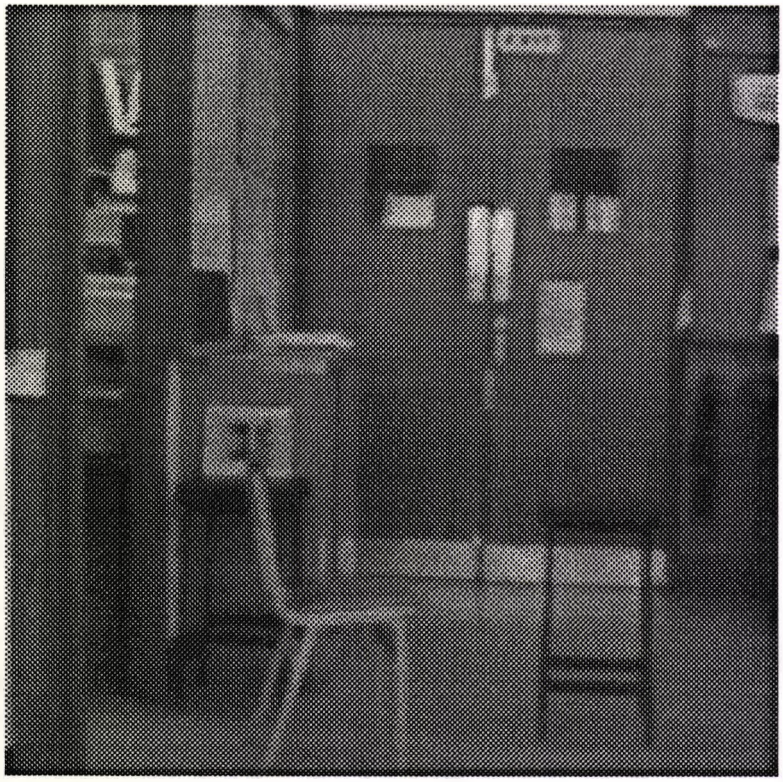


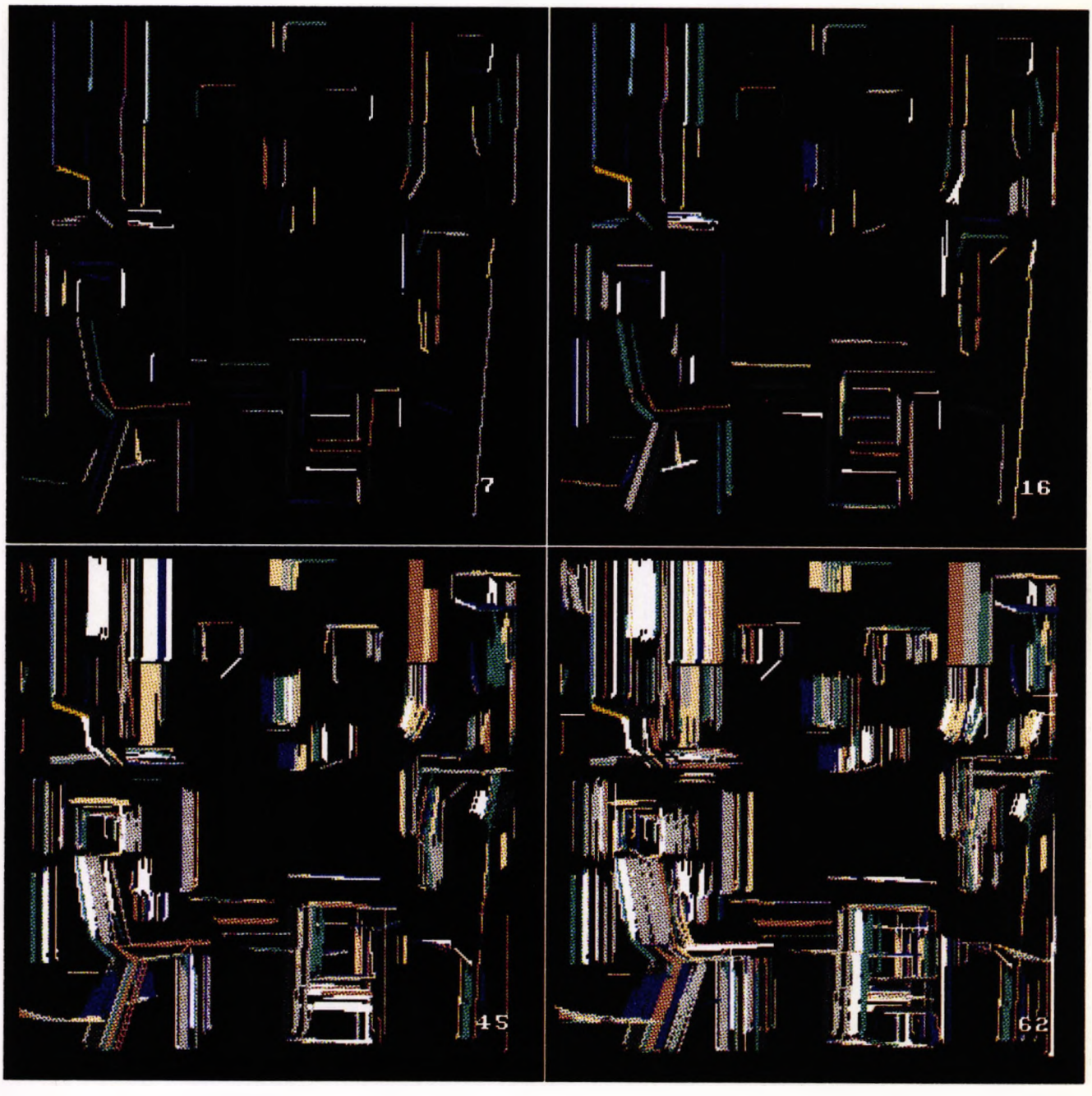
Figure 6.25: First frame of the sequence for real scene 1

in reality, memory constraints and context-switching times would not allow such an achievement, even after the communication requirements are totally ignored. Thus the load must be spread across a number of processors. The following points must be observed when examining the process timings,

- The minimum processing time for matching is the same in all the example scenes. This may be due to those processes which do not find any tokens in their search area to perform any matching with. The overheads are approximately just some non-starter loop statements. However, the point below will also apply to this case.

- It is not clear *exactly* why the processing of the Kalman filters varies between the values shown in the appropriate tables, albeit a very slight variation. Two explanations are as follows. Firstly, it may be that in cases where spatial movement and velocity are zero, there is less computation involved, and secondly, each processor is at different stages of processing and may have different numbers of active processes on the scheduled list, thus context-switching of these low-priority time-sliced

Plate 6.6: Frames 7, 16, 45 and 62 of a sequence for real scene 1, with colour-coded line segments displaying spatio-temporal continuity



Busy Scene	Avg. No. of Tokens	Avg. time for *SC*	*Min* − *Max* time/process filtering	matching
1st	108	1.65 *sec*	136 − 320 μsec	136 − 905 μsec
2nd	85	1.05 *sec*	136 − 279 μsec	140 − 792 μsec

Table 6.14: Average frame processing time for busy scenes obtained over a lengthy run

processes leads to differing execution timings.

• Due to the complexity of the scene, the matching takes slightly longer than in a simple scene, since more tokens are likely to exist in the search area. However, an upper limit on the number of tokens that can appear in a search area can be set to be able to control the matching load. In this implementation this is set to 9 after experimental observations. If the execution time for a successful match for a search area with only one token in it may be taken as approximately only slightly more than the minimum processing time given in Table 6.14, then in fact the timings show that a maximum of about 5 or 6 matches are being conducted within a maximum matching-process time of about 792-900 μsec.

• The *TE* processors are idle while they await the arrival of the data from the next frame. This idle time could be spent in a number of ways. One would be to perform tracking on combinations of features, such as whole objects. Also, the tracking could employ a more complex filtering approach to model acceleration, as well as other features associated with each token. Furthermore, work is currently in hand at the Machine Vision Laboratory of the City University in trying to use this idle time for 3D scene matching of the tracked tokens locally. This involves the parallelisation of the approach and the satisfaction of the communicational requirements while the idle time lasts.

The execution times for filtering and matching compare well with other work such as that of [STD90] who use a very simple least mean square method for predictive object tracking which takes 64μsec to execute.

The processing, processors and processes are all monitored by TIPS to keep the user informed of the goings-on in the various parts of the MATCH system. An instant of this is shown in Plate 6.8. The window on the left shows general processing results for each frame. The middle window shows the load-balance of the system by displaying the number of active processes on each processor. The last window demonstrates the address of each active tracker process. For example, note that for 86

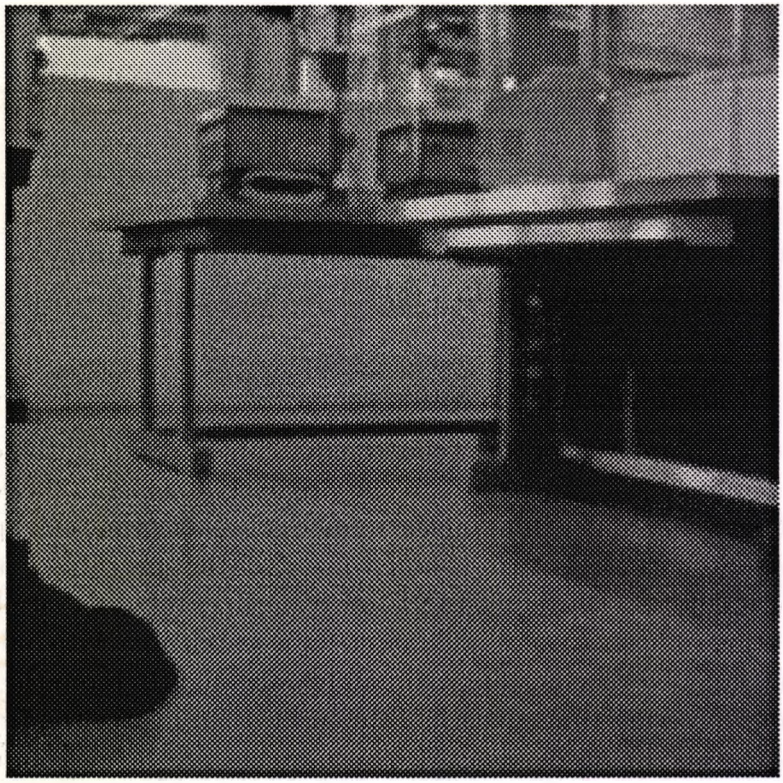


Figure 6.26: First frame of the sequence for real scene 2

edge lines (from the previous frame), the first 5 transputers in the *TE* were running 10 processes each, and the last four were running 9 each, demonstrating the most efficient load balance. The two small windows below right, are meters displaying the address of the last requesting process, and the address of the last terminating process, respectively. The timings show the full processing time inclusive of all extra progress reporting communications (which can be disabled).

6.6.1 Some Efficiency Issues

The *SC* is undoubtedly a major communications bottleneck for the *TE* network. The computation time of the *TE* system is dictated by the administration tasks of the *SC*, which include queue house-keeping, system load monitoring, communications per token, and other functions related to the general upkeep of the *Blackboard*.

The intricate set of actions handled by the *SC* are interwoven in a complex, and inter-dependent web of tasks which are difficult to decouple

Plate 6.7: Frames 5, 23, 39 and 56 of a sequence for real scene 2, with colour-coded line segments displaying spatio-temporal continuity

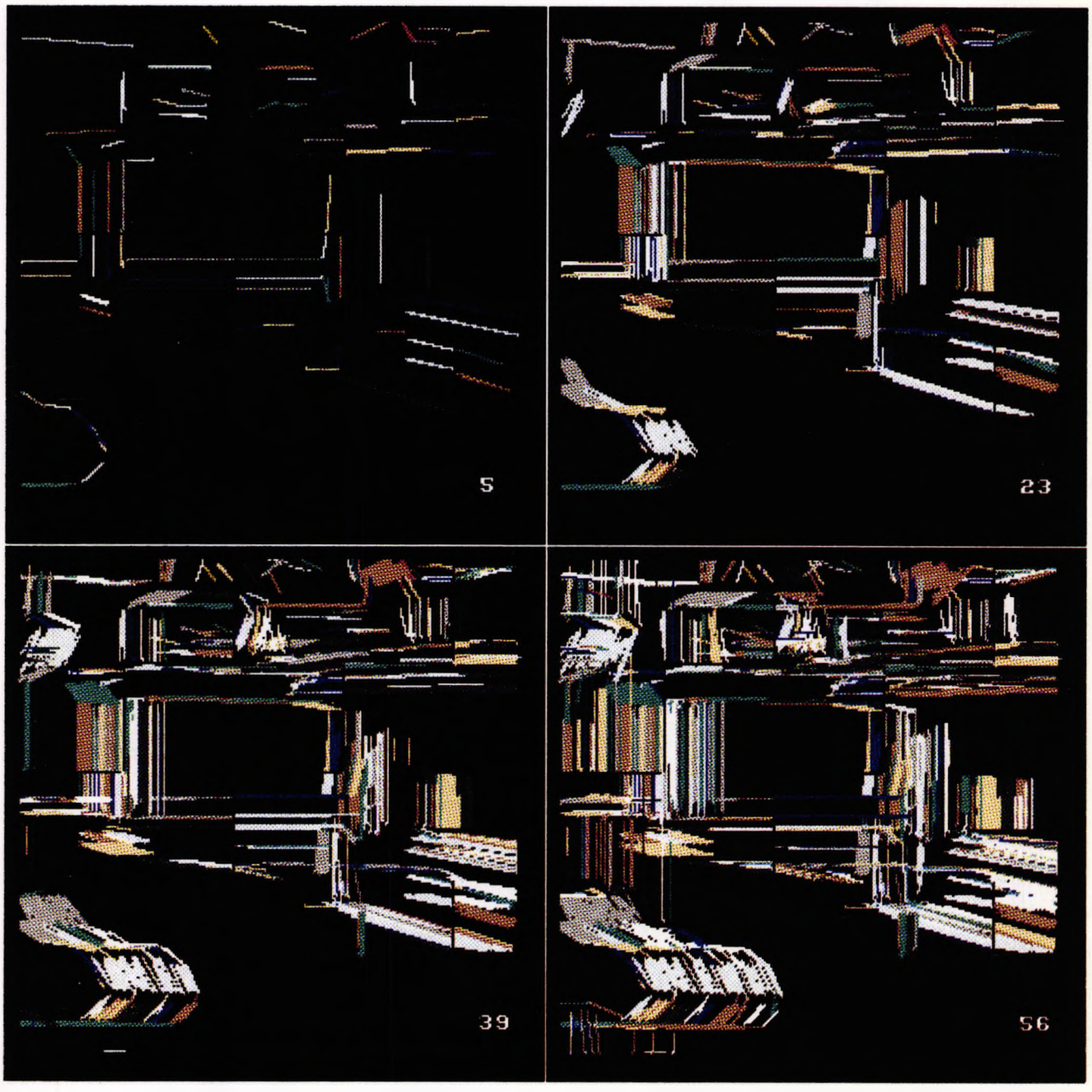


Plate 6.8: The monitoring of the MATCH system

and analyse separately. For example, its actions are dependent on the speed of the processes in performing their tracking, which are in turn dependent on the efficiency of the supply of information for performing the tracking. Naturally, the number of tokens involved has a direct affect on the performance of the system. Thus, the timings presented in Table 6.12 and Table 6.14, albeit very small, are directly related to the conditions surrounding each particular test case. Also, the greater the number of tokens in the system, the longer for the *SC* to get round to paying attention to the process requests. Using a selection filter, the system must therefore be employed to monitor just enough tokens to ensure a maximum response time. It is interesting to note the following as a contrast. For higher resolution images (such as a 512x512) the execution time for the Canny is expected to increase by a power of two, whereas the *SC* will still only need to deal with approximately the same number of edge segments (which may now be longer lines, but are still represented by their end-points).

Improvements could be achieved by employing multiple *SC*s with independent tracking engines. Also, from Figure 6.22 it is noticeable that there is only one branch from the *SC* to the *TE* processors. This is due to the lack of available links after administrative duties have been taken care of. By re-distributing these duties (e.g. combining the progress reports to the HOST B004 to pass through the Harlequin HOST) and freeing a link, it would be possible to double the bandwidth of the communications. Another approach to increase the performance of the *SC* would be the use of the soon to be available T9000 processor. This topic will be covered in further detail in Chapter 7. The general performance of the *TE* network, and the *SC* component in particular, further suffers by an approximated 25% slow-down in link communications when the C004 link-switch chip is used for connecting the transputers (please see Section 2.4.1). Perhaps this will diminish when INMOS Ltd. introduces the C104, the next generation communications chip, due out around the same time as the T9000.

The weak-point in the current realisation of MATCH is not a lack of transputers as it may be thought at first. It is true that in order to track more tokens, more transputers were assigned to the *TE* network. In return, this left fewer for the *FEE* network, causing a slow-down in the engine's performance on the Canny edge detection. However, the major problem lies elsewhere as briefly mentioned earlier: The total processing time of (the overlapped stages of) the *FEE* network of 8 transputers (2x4 array), taking into account the extra overheads, is around 4.1 seconds. This includes around 2.1 seconds for Canny, overlapped by around 3.5 seconds for the edge segmentation stage. For an average of

108 tokens, the *TE* network is ready for its next set of tokens in about 1.9 seconds, again including overheads. Thus, frame-processing rate is dictated by the *FEE* processing, for which some improvements were suggested late in Section 6.5.3. The most immediate step would be to parallelise the edge segmentation phases.

The results presented were obtained using the α, β Kalman filter algorithm. The general algorithm-independence of the match system was also put to the test by using the standard Kalman filter approach. However, since the α, β and the standard Kalman filters are in nature the same technique (only varying in algorithm), the results for both systems are very similar, leaving no grounds for a comparison study, except from the point of view of computational performance. This was found to be marginally slower than the figures presented in Table 6.12 and 6.14. Still, the standard Kalman filter implementation was a step in reaching the α, β tracker (as derived in Section 5.5), and a useful approach in testing MATCH with an alternative algorithm.

Of course, the system was also tested with the basic object tracking algorithm as presented earlier.

6.7 Summary and Conclusions

This chapter discussed the edge-tracking implementation of a parallel model for tracking tokens on a mixed SIMD/MIMD parallel processing platform. Initially, a few notions on the use of parallelism were introduced, followed by some past and present work in motion using concurrency, especially the work in hand on the VOILA project. Next, an initial investigation into the application of some basic principles of motion, using parallel processing, was presented. This investigation of object-tracking enormously helped in the design and implementation of the work that was yet to come.

The parallel model of computation was proposed next. It spanned issues such as communications, data structures, a blackboard mechanism, load balancing and much more. The model was purposely kept free of any association with particular tracking algorithms or *multi-process* architectures. Thus, it may be applied, as a guideline, across many token tracking applications where parallel processing is sought.

The model was continued into the next section where the MATCH system was presented as a multi-processor tracking system. Initially, for the feature extraction stage, two different applications of the *data parallelism* model were compared on two SIMD networks for the Canny

edge detection operation. Results for increasing number of transputers per network were weighed against each other. The array network was found to be more efficient over the farm implementation, but there is easy scope for improving the farm network's performance, given the few transputers available. The main idea is to *double* the processing bandwidth by adding a new farm to another link on the master processor (Please see Figure 4.10).

Processors		Sobel		Canny		Canny	
Farm1	Farm2	32x32		32x32		64x64	
1	1	0.577	(100%)	10.626	(99%)	7.886	(96%)
2	2	0.302	(96%)	5.413	(98%)	4.257	(93%)
3	3	0.250	(80%)	3.675	(98%)	3.049	(89%)
4	4	0.247	(64%)	2.806	(97%)	2.441	(87%)
5	5	0.247	(56%)	2.284	(96%)	2.001	(83%)
6	6	0.247	(56%)	1.937	(95%)	1.840	(83%)
7	7	0.247	(56%)	1.717	(92%)	1.527	(86%)
8	8	0.247	(56%)	1.526	(92%)	1.527	(80%)

Table 6.15: Execution times for bi-linear farm, Demand-Driven Model on 256x256 images.

Table 6.15 shows the results for the implementation of the Sobel and Canny operations on a bi-linear farm network (with the corresponding percentage improvement over the linear farm in brackets), with the master controller supplying packets of data to whichever farm that is ready to accept next. An alternative approach to this method of supply would be to decouple the farms by splitting the image into two halves and supplying each farm with only the appropriate half-image. Splitting the number of transputers in the network into two branches also reduces the communication bandwidth for those packets of data which would otherwise have had a longer path to travel before reaching their destination. From the table it can be observed that once again it is only efficient to add more processors for an application with a high computational load which reduces the communication latency. Thus the performance soon saturates for the Sobel, whereas excellent efficiency for the heavy computation load of the Canny is observed as the farm slaves increase in number. (If these results are compared with those of Table 6.6 for equivalent number of transputers (e.g. two farms of 4 for a linear chain of 8 processors), then not much improvement is observed, since in both cases the controller is still supplying work to

similar number of processors, only for this case it supplies through two links which are monitored via a standard ALT construct).

The efficiency idea could be continued ultimately to a third branch leading to a tri-linear farm. Also, multiple arrays could be used for the control-driven approach, with each array processing part of the image. However, the *FEE* system's real bottleneck was found to be the non-localised processing of the string transformations (into lines) which renders the job somewhat abstracted from easy parallelisation. Since the concentration of this work has been on the correspondence analysis stage, the requirements for further improvement of the *FEE* system's bottlenecks were considered out of the scope of the most immediate work, and were not pursued any further.

Next, the parallel model was extended for implementation on the transputer as a distributed memory model, using the α, β Kalman filter for the tracking algorithm. The MIMD implementation used a tree network whose suitability and functional aspects were considered along with other factors such as the *SC*'s *Blackboard* and process management issues, and *TE* processor functions and process manipulations. Results pertaining to the general analysis of the processes, the matching, and the tracking were presented next, succeeded by some efficiency issues. These showed a frame processing rate suitable for use in AGV motion analysis for providing visual input to the vehicle central system to aid location and navigation. The *TE* network's overall throughput may be increased by employing three branches at each node of the tree rather than the present binary approach. However, the *TE* still performs at an acceptably fast rate especially in relation to the *FEE* network. Furthermore, the modularity of the design of MATCH allows for the integration of a different, more efficient *FEE* network, whatever its configuration and overall nature may be, without interfering with the structure or the implementation of the *TE*, as long as the *TE* receives its token lists via a link connection.

The applicability of the system was tested by using an object tracking algorithm (described in the earlier investigation phase), and two different approaches of the Kalman filter to linear estimation. This vindicated the effectiveness of the model and its subsequent implementation as a sound proposition towards an independent communications and hardware platform for the analysis and implementation of general token tracking algorithms based on transputers.

Chapter 7

Summary and Conclusions

7.1 Brief Summary and Initial Conclusions

The aim of this thesis was to present and evaluate the use of the transputer in appropriate configurations with respect to the enhancement and parallelisation of vision algorithms, as pertaining to distinct image processing applications and problem areas.

In Chapter 2 some foundations in the fields of image processing and parallel processing were laid, and many important and relevant topics to this thesis were reviewed. In addition, some fundamental aspects of the transputer and OCCAM were studied. Also, issues in more efficient OCCAM programming on the transputer were discussed enhanced by the "lift" example.

Chapter 3 introduced the *forward-feed* or *reverse-feed distribution* mechanisms for distributing images across arrays or meshs of transputers. The scheme was analysed rigorously to provide mathematical means of evaluating the distribution and collection costs of images on transputers prior to main image processing. Since communication factors are of primary importance when evaluating or simulating system performance, the issues tackled by this work can be of great value. Estimated results were obtained which matched closely those measured (Table 3.1). The scheme was then applied to a real-time situation where the general positioning of labels stuck on products were examined.

Some initial thoughts following this work were that transputers are extremely easy to work with, and one need simply connect them together to establish multi-processor configurations without resource to

external logic or other specific hardware. The *reverse-feed distribution* scheme was intended as a general scheme to map any $P \times Q$ image onto a $M \times N$ transputer array. This flexibility also meant that the method was too costly for the real-time label inspection problem, added to which was the cost of performing totally generalised, parallel algorithms for inspection. To this end, a customised system of transputers was set up, and this investigation showed a much better performance efficiency. From the results in Tables 3.3 and 3.5, it could be concluded initially that a customised set-up of transputers could provide a cheap, affordable, and efficient system for label inspection. More generally, it can be concluded that the transputer is very likely to perform well as a processing unit in an embedded system.

In Chapter 4, the demand-driven approach of farm parallelism was examined and compared with the control-driven mechanism of the *reverse-feed distribution* scheme through the implementation of the Sobel edge filter and the Hough transform as preliminary, but independent, stages of a more sophisticated approach to label inspection. In addition, the Hough transform was shown to parallelise well when considered in sub-images. The resulting $\rho\theta sHT$ transform was also found to be suitable for detection of arcs as small line segments, which greatly simplified the detection of oval (or circular) labels. Inspection results were provided in Tables 4.2, 4.3, and Figures 4.6 and 4.7. Sobel and $\rho\theta sHT$ processing times were further measured for a larger, eight transputer configuration as presented in Tables 4.5 and 4.6 in Section 4.8.

Despite some of the very efficient speed-ups achieved, and despite the fact that some computationally complex algorithms are being executed extremely fast, the transputer is still rather slow to provide a fully real-time performance in this application, although the potential is there. In fact it is not so much that the transputer itself is slow, but that the communication demands are very heavy when distributing data and receiving results. This can be concluded to be generally true for SIMD parallel processing of images on transputers.

Chapter 5 started as the vehicle for introduction to the field of CATVI or dynamic scene analysis. The opportunity was then used to introduce the optimal estimation technique of Kalman filtering, which was subsequently applied to derive the state and measurement vectors for the equations of motion. Token matching via the Mahalanobis distance was studied, and this was followed by the possible uses of tokens in 3D structure from motion understanding. The overall feature tracking algorithm used for the bootstrap and run stages of a feature tracking system employing a scene flow model was outlined in Section 5.8.

Finally in Chapter 6 much of the earlier work in this thesis on geometric parallelism was updated and used in the *FEE* engine in conjunction with a MIMD approach to feature tracking via the *TE* engine of the MATCH system. The MATCH system was shown to reflect the pipelined processing phases in typical vision processing through comments accompanying Figures 1.1 and 6.5. The system was described in detail through its varying sub-units which represented the steps involved in the process of achieving motion correspondence by tokens. Earlier, an investigation into object tracking had also been outlined. This had been followed by the parallel computational model which was the guideline to the implementation of MATCH. Extensive results for the different stages of MATCH were presented in Sections 6.3.3, 6.5.3, 6.6, and 6.7.

Conclusions specific to the role of the transputer within the concepts and results of Chapter 6 must be that again the SIMD parallelism of the *FEE* unit is rather slow due to the communications latency. The *TE* fares rather better, since the overall processing of the tokens is of a lighter nature, and so the *TE* is always ready for the *FEE* network, given that an appropriate number of tokens are involved. Nevertheless, the communicational load of the *TE* is extremely heavy, since the system control is centralised, and the *SC* is completely burdened with work.

There were some advantages in using the transputer too. It was most straight-forward to work with and parallel processing techniques were implemented on transputer configurations with great ease. Crowley et. al. [CSD88] and Deriche and Faugeras [DF90] have discussed the implementation in hardware of Kalman filters for token tracking. The transputer has been shown capable of achieving filtering and matching rates that justify its serious consideration in place of a dedicated hardware system. In addition, it provides a flexibility for alterations and improvements that is not reflected in a hardware-based system. Furthermore, MATCH is intended as the front-end processor for a vision-based vehicle navigation system. Subsequent to token tracking, the next stage matches the labelled tokens to the geometric model of the environment - this stage is currently under development, but will operate in parallel with a "bootstrapping" model recognition and vehicle location algorithm also currently under development [BO91].

7.2 Some General Comments

The general conclusion of this thesis must be that, for computer vision, communications are the nemesis of the transputer. Communi-

cations have caused such problems for the transputer that most have resorted to enhancing their transputer systems with special hardware-based buses or shared memory. Examples of these may be found in [Pag88, DEH89, RPBK90, Bux91]. What then becomes of the transputer with respect to the SIMD and MIMD spectrum of hardware for low-level to high-level computer vision as depicted in Figure 2.1? It was seen how the PC WARP and the IUA architectures are built to cater for different vision algorithms. They are successful to a certain degree to satisfy their intended purposes, and are probably as general-purpose as vision architectures come. (Note that no real instant of the IUA has yet been reported). It is envisaged that the transputer may be able to play a role as an autonomous processor, either as part of a larger machine, or as the building block of a completely transputer-based machine. However, without special hardware, as employed for such machines as MARVIN and PARADOX, it is doubtful that it could compete with other vision architectures which invariably include dedicated communications mechanisms or shared-memory systems. Dedicated hardware would be at least a necessity when concerned with SIMD processing. However, for MIMD processing, the transputer could prove the ideal tool if faster and more efficient communications were provided. MIMD processing is computationally bound, and that is where the transputer shines. But processors then need to communicate, be it with a control processor or with other processors. Multiplexing and de-multiplexing of messages across links is tedious, and it seriously limits the communications bandwidth. A prime example of this is the MATCH system. Some solutions that spring to mind are as follows,

- make the communication links faster,
- achieve direct processor to processor connection using link connection reconfiguration techniques during program activity,
- have a communication processor in hardware which will also handle the multiplexing and de-multiplexing.

Faster links are limited only by what the latest technology can offer. Reconfiguration is not so much non-trivial than down-right cumbersome, where each algorithm will have different reconfiguration requirements, once again leaving the burden on the user, and probably leading to non-portable code. However, in the absence of any other means to speed up processing, reconfiguration is a valuable approach [Pag88]. A hardware communications processor is a very attractive idea which could greatly reduce processing/communications latency. It would also

be a more sensible idea than an arbitrary increase in the number of links, since that would still not be enough for many (besides VLSI limitations on the number of links possible), and it would merely create more administrative problems for the end-user for inter-connection and programming of the devices.

Overall, as with the PC WARP and the IUA, the transputer may find itself in a suitable home in a hybrid architecture, with perhaps extra communications hardware, that supports both SIMD and MIMD. In such a case it might even fit and perform well as an elementary research machine within the framework of classification 3 of Figure 2.1. However, as stated by Prior et. al. [PNRC90]:

> ...for many applications the performance of a system in an irregular configuration can be expected to exceed by far that of systems in standard configurations...

(Note the pseudo-MIMD customised approach to label inspection described in Chapter 3 which was more successful than a regular grid approach with a similar number of transputers). This further emphasises the necessity and the difficulty in achieving an ideal system within vision which is capable of the task depicted in classification 3 of Figure 2.1, since the hardware burden should be taken away from the scientists for them to be able to concentrate on the task of pushing the boundaries of computer vision towards those of human vision for scene perception.

OCCAM as a parallel processing tool, allows for a very user-friendly sub-division of a problem into parallel sub-processes. For that it is applauded and in the author's opinion, OCCAM is as good as any language for exploiting vision algorithms that must be implemented in parallel. It could even form the underlying structure of the intelligent interface of Figure 2.1's classification 3. However, at the moment OCCAM performs best only on the transputer, and it is almost unavailable, except in interpreted format, on any other machine. Therefore, before some improvements to the transputer are seen, it is unlikely that OCCAM will gain the popularity that it deserves.

7.3 Major Contributions

The major contributions of the work presented in this thesis are briefly itemised as follows,

- Proposal and implementation of a decoupled approach to the problem of correspondence,
- Presentation of a parallel computation model for the implementation of token tracking algorithms for different image tokens (for achieving motion correspondence),
- The introduction and implementation of the $\rho\theta sHT$ line segmentation technique,
- The presentation of an image distribution and collection technique with performance evaluation tools for transputer arrays,
- Fresh approaches towards more real-time inspection of product labels,
- And finally, more generally, the evaluation of the position of the transputer within the spectrum of vision processing levels.

7.4 The T9000

The new generation INMOS transputer has only recently been announced [Inm91] and it is to be available around the middle of 1992. The T9000, as it is named, is projected to feature the following,

- Peak performance rate of 200Mips for its integer processor,
- Peak performance rate of 25Mflops for its floating-point processor,
- Communication links that provide a peak total of 80Mbytes/second bidirectional bandwidth,
- Pipelined superscalar micro-architecture allowing multiple instructions to be issued and executed per processor cycle. The T9000 contains hardware which assembles instructions from the instruction stream into groups and then sends them through the pipeline,
- 16K instruction and data cache,
- Workspace cache of 32 words holding the most frequently used data,
- Virtual channel multiplexing and communications hardware capable of mapping logical links onto physical links.

It is said that the T9000 running at 50MHz will be capable of executing compiled code for a 20MHz T805 typically 10 times faster. The T9000 will be complemented with the C104 packet routing switch which is an improved and updated version of the C004. The C104 is expected to introduce only sub-microsecond latency. Together, they will allow faster reconfiguration for direct connections to other T9000s. Could it be that a lot of the communicational problems associated with the existing family of transputers will be eliminated with the introduction of this new proposed transputer? This could happen most immediately in two ways,

- Fewer processors would be required since each T9000 will be capable of putting in the performance of several T800 or T805 transputers. Fewer processors would need less in the way of communications,
- Faster link speeds, and the virtual channel router, will provide a much higher communicational bandwidth giving a better overall computation to communication ratio.

Not only will the T9000 processor be a massive boost to SIMD or MIMD transputer-based systems in general, it may finally herald the transputer's entry into the super-league of microprocessors with the prospect of much more widespread use. It will certainly be a strong contender for vision-based architectures, capable of coping with even the task of low-level image processing at rates comparable to present special-purpose processors.

More immediately for the work in this thesis, the T9000 could be used as a super *SC* in the *TE* network of MATCH. If its projected performance improvement over a T805 could be assumed to be true here too, the *SC* could perform its task for a scene of approximately 100 tokens in an average of 135 milliseconds, giving a projected processing rate of about 7 frames a second which would, by present standards, be more than adequate for AGV motion. This is not to mention the benefits that would be observed by employing T9000 processors for other parts of the MATCH system.

In summary and conclusion, it must be said that the transputer can be used for real-time processing in vision. Only, there are still constraints in low-level image processing, which are likely to be alleviated by the T9000. However, for medium and high-level vision operations when computation load is quite demanding, the transputer performs well

enough to warrant its use for real-time vision work. This was amply justified through the investigations on the $\rho\theta sHT$ and Canny implementations, and also through the token tracking application. Furthermore, the flexibility offered by the transputer in building and programming multi-processor architectures means that it is a more cost-effective approach than constructing special-purpose hardware for many vision tasks. Finally, not only have transputers made it possible for more scientists to perform research at lower cost, they have further emphasised the problems computer vision faces in reaching the capabilities of human vision.

Annotated Bibliography

[AD90] A.T. Ali and E.L. Dagless. Automatic Traffic Monitoring using Transputer-Image Processing System. *Proc. of the Second Int. Conf. on Transputer Applications*, pages 209–210, 1990. The authors formulate moving object detectors by combining differencing with edge detection, and apply them to traffic scene analysis. They report dissatisfaction with the communication speeds of the transputer for their implementation.

[ADM81] J.K. Aggarwal, L.S. Davis, and W.N. Martin. Correspondence Processes in Dynamic Scene Analysis. *Proceedings of the IEEE*, 69(5):562–572, 1981. The authors use two general approaches for establishing correspondence between features, pixel-based and token-based techniques. Icons and structures are used for matching between frames in each approach, respectively.

[AN88] J.K. Aggarwal and N. Nandhakumar. On the Computation of Motion from Sequences of Images-A Review. *Proceedings of the IEEE*, 76(8):917–935, 1988. The authors present a review paper on the developements in the computation of motion and structure of objects in a scene from a sequence of images, highlighting and comparing token-based and optical flow-based approaches.

[Atk87] P. Atkin. Performance Maximisation. Technical Report 72-TCH-017-00, INMOS Ltd., 1987. This is a very useful technical note for new OCCAM programmers. It provides many tips for more efficient coding.

[BA90] M. Ben-Ari. *Principle of Concurrent and Distributed Programming.* Prentice Hall, 1990.

[Bat79] B.G. Batchelor. Interactive Image Analysis as a Prototyping Tool for Industrial Inspection. *Computers and Digital Techniques*, 2(2):61–70, 1979. A variety of tasks, including texture analysis of metal surfaces, measurement of female screw threads, and print legibility inspection are considered by the author.

[BB82] D.H. Ballard and C.M. Brown. *Computer Vision.* Prentice Hall, 1982.

[BH89] R.F. Browne and R.M. Hodgson. Mapping image processing operations onto transputer networks. *Microprocessors and Microsystems*, 13(3):203–211, 1989. This paper examines the performance of SIMD transputer networks with application to image processing. However, it considers computational speedup without explicitly taking into account communication costs.

[BO91] B. Brillault-O'Mahoney. *(In Preparation).* PhD thesis, City University, London, 1991.

[BR90] A.P. Bernat and J. Rupel. A Transputer-Based Motion Detection and Tracking Algorithm. In *Proc. Third Int. Conf. of NATUG*, pages 295–305. IOS Press, 1990. A differencing technique is used in conjunction with a cellular decision making process to track motion across international borders. The transputer is found as an elegant approach for implementation.

[BSI90] S. Bottalico, F. De. Stefani, and F. Imelio. Real-Time Processing Architectures in Ground Surveillance Systems. In *Proc. of 5th Int. Conf. on Image Analysis and Processing*, pages 699–705, 1990. Transputers are used for the implementation of low-level image processing algorithms for remote sensing and surveillance systems. Results for Sobel, Canny and Marr-Hildreth edge detectors are produced, amongst other techniques.

[Bux91] B. F. Buxton. P2502 VOILA - Vision Research Pilot Project. 2nd annual report, VOILA Consortium, May 1991. This is a report on the state of research on the vision systems involved in the VOILA consortium, whose aim is to develop dynamic vision systems for the control of robot vehicles.

[Can86] J. Canny. A Computational Approach to Edge Detection. *Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986. The author presents a mathematical analysis for an edge detector based on a function accurately approximated by the derivative of a Gaussian.

[CD86] R.T. Chin and C.R. Dyer. Model-based Recognition In Robot Vision. *Computing Surveys*, 18(1):67–108, 1986. The authors present a comparative study and survey of model-based object-recognition algorithms for robot vision including applications to industrial inspection.

[CH82] R.T. Chin and C.A. Harlow. Automated Visual Inspection: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(6):557–573, 1982. A wide-ranging overview of automated visual inspection techniques is provided.

[CR83] C. Cafforio and F. Rocca. The Differential Method for Image Motion Estimation. In *Image Sequence Processing and Dynamic Scene Analysis, ed. T.S.Huang*, pages 104–124, 1983. Using differencing and cross-correlation techniques, the authors present a method for esti-

mating motion with application to television images where coding for redundancy reduction and remote guidance is necessary.

[CR88] D. Casasent and J. Richards. Industrial Use of a Real-time Optical Inspection System. *Applied Optics*, 27(22):4653–4659, 1988. An industrial inspection system is described including a label inspection phase using the Hough transform and 1D correlation.

[CSD88] J.L. Crowley, P. Stelmaszyk, and C. Discours. Measuring Image Flow by Tracking Edge Lines. In *IEEE Proceedings of the 2nd Int. Conf. on Computer Vision*, pages 658–664. Computer Society Press, December 1988. This paper proposes the idea of tracking edges for use in updating a model of image flow, using a very simplified form of the Kalman filter which can subsequently be implemented in hardware.

[CSDP89] A. Chehikian, P. Stelmaszyk, and S. De Paoli. Hardware Evaluation Process for Tracking Edge Lines. In *IEEE Proceedings of the Int. Workshop om Industrial Applications of Machine Intelligence and Vision*, pages 332–336, April 1989. A 68000 processor and an ADSP 2100 are proposed for use in a hardware implementation of a simplified Kalman filter for edge tracking.

[Dav87] E.R. Davies. Design of Optimal Gaussian Operators in Small Neighbourhoods. *Image and Vision Computing*, 5(3):199–205, 1987. This paper presents two optimised Gaussian masks which are more accurate in reducing image noise and can maximise isotropy.

[Dav90] E.R. Davies. *Machine Vision: Theory, Algorithms, Practicalities.* Academic Press, 1990.

[DEH89] P.M. Dew, R.A. Earnshaw, and T.R. Heywood, editors. *Parallel Processing for Computer Vision and Display.* Addison-Wesley, 1989.

[DF90] R. Deriche and O.A. Faugeras. Tracking Line Segments. *Image and Vision Computing*, 8(4):261–270, 1990. This paper analyses two different representations for a line segment and selects the mid-point representation to use in its implementation of the α, β tracker. It also performs some detailed error analysis.

[DH72] R.O. Duda and P.E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM*, 15(1):11–15, 1972. The authors present the Hough transformation technique based on the normal (ρ, θ) parameterisation of a line.

[Dic88] M. Dickey. Success Demonstrates Inspection Reliability. *Robotics World*, pages 37–39, 1988. This article outlines case studies of label inspection applications.

[Dil82] E.G. Dillman. Vision System for Quality Control of Label Inspection. *Proc. SPIE Int. Soc. Opt. Eng.*, 336:168–172, 1982. A feature comparison approach based on image row and column sums is described for inspecting labels on containers.

[ERG91] T.J. Ellis, P. Rosin, and P. Golton. Model-Based Vision for Automatic Alarm Interpretation. *IEEE Aerospace and Electronic Systems*, 6(3):14–20, 1991. A surveillance system is presented, implemented as a knowledge-based recognition system. Differencing and median filtering techniques are used to detect motion, followed by classification and model matching to identify the causes of alarms.

[ES89] S. Eghtesadi and M. Sandler. Implementation of the Hough Transform for Intermediate-Level Vision on a Transputer Network. *Microprocessors and Microsystems*, 13(3):212–218, 1989. A parallel implementation of the Hough transform on an eight transputer network is presented. The authors partition the Hough space across the processors which are updated via link communications.

[EWM87] T.J. Ellis, G.A.W. West, and P. Moukas. Complete Object Inspection using CAD Models and Object Inspection. In *Proceedings of the Alvey Vision Club Conference, AVC87*, pages 117–124, 1987. A robot arm is employed to provide different views of an object in its stable orientations. The authors then use CAD models of the object to check for manufacturing defects.

[FK89] H.P. Flatt and K. Kennedy. Performance of parallel processors. *Parallel Computing*, 12(1):1–20, 1989. The authors study the upper bounds of performance of parallel processors under ideal conditions through the impact of communication and synchronisation overheads.

[FKS83] T.J. Fang, L.N. Kanal, and G.C. Stockman. Experiment in Label Inspection Using Template Matching. *Proc. of Int. Conf. on Systems, Man and Cybernetics*, pages 192–196, 1983. A three-stage experimental method for label inspection consisting of adaptive thresholding, registration, and template matching is proposed.

[Fly66] M. J. Flynn. Very High-Speed Computing Systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966. The author categorises machines on instruction and data streams. His classification is widely in use today.

[FMM88] T.J. Fountain, K.N. Matthews, and Duff. M.J.B. The CLIP7A Image Processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-10(3):310–319, 1988. This paper describes the CLIP7A processor which is a 256 element linear array processor with two CLIP7 processors at each element.

[Fre61] H. Freeman. On the Encoding of Arbitrary Geometric Configurations. *IRE Trans. on Electronic Computers*, EC-10:260–268,

1961. This paper describes the popular method of chain-coding using re ctangular and hexagonal array representations of geometric shapes.

[FT79] C.L. Fenema and W.B. Thompson. Velocity Determination in Scenes Containing Several Moving Objects. *Computer Graphics and Image Processing*, 9:301–315, 1979. This paper describes a non-matching procedure which is a gradient intensity method. It detects and quantifies velocities of several moving objects in an image which may be thus segmented.

[Gel74] A. Gelb. *Applied Optimal Estimation.* MIT Press, 1974. This book on optimal estimation, covers the subject from a theoretical, mathematical and engineering point of view. Linear and non-linear estimation techniques are covered amongst other topics.

[GM90] P.M. Griffin and S.L. Messimer. Feature Point Tracking in Time-Varying Images. *Pattern Recognition Letters*, 11(12):843–848, December 1990. Feature point tracking is described for non-rigid motion of constant velocity points using minimum number of frames.

[GW87] R. C. Gonzalez and P. Wintz. *Digital Image Processing.* Addison-Wesley, 1987.

[HB84] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1984.

[HJ83] S.M. Haynes and R. Jain. Detection of Moving Edges. *Computer Vision and Image Processing*, 21:345–367, 1983. A time-varying edge detector is presented which helps in estimating the direction of motion by combining edge detection and frame differencing.

[HJ88] R.W. Hockney and C.R. Jesshope. *Parallel Computers 2.* Adam Hilger, 1988.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[Hol86] J. Hollingum. Window on Japanese Vision Sensing. *Sensor Review*, pages 80–82, April 1986. The label inspection application of the MW-2000 system built by Fuji Electric Co. is reviewed.

[Hou62] P.V.C. Hough. Method and Means for Recognising Complex Patterns. *U.S. Patent 3,069,654*, 1962. The author introduces the straight-line transformation for detecting complex patterns of points in binary image data.

[HP87] C. Harris and J. M. Pike. 3D Positional Integration from Image Sequences. In *Proceedings of the 3rd Alvey Vision Conference*, pages 233–236, 1987. Kalman filtering is applied to image point features to determine camera ego-motion.

[HS81] B.K.P. Horn and B.G. Schunck. Determining Optical Flow. *Artificial Intelligence*, 17:185–203, 1981. An iterative algorithm for determining velocity vectors based on the spatial temporal gradient method is presented. The motion considered here is assumed to be simple, continuous and varying almost everywhere across the image.

[HS88] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988. This paper presents a combined corner and edge operator to cater for images regions containing texture and isolated features such as a bush.

[HS90] C. Harris and C. Stennett. RAPID - A Video Rate Object Tracker. In *Proc. 1990 British Machine Vision Conference*, pages 73–77, 1990. The authors present a real-time model-based tracking algorithm for a known three-dimensional object. Control points on the edges of the known object are tracked using the Kalman filter.

[IK88] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 44:87–116, 1988. This paper lists the wide coverage given to the Hough transform in dealing with different parameterisations, applications, and architectures.

[Inm87] Inmos. *Transputer Instruction Set: A Compiler Writer's Guide.* Prentice-Hall, 1987.

[Inm88a] Inmos. *OCCAM 2 Reference Manual.* Prentice-Hall, 1988.

[Inm88b] Inmos. *Transputer Development System User Manual.* Prentice Hall, 1988.

[Inm89] Inmos. *The Transputer Databook.* Prentice-Hall, 1989.

[Inm91] Inmos. *The T9000 Transputer: Products Overview and Manual.* Prentice-Hall, 1991.

[Jai81] R. Jain. Dynamic Scene Analysis Using Pixel-based Processes. *IEEE Computer*, pages 12–18, August 1981. Differencing and accumulative differencing techniques are explained, and applied to classify changing regions using a decision tree.

[JMA79] R. Jain, W.N. Martin, and J.K. Aggarwal. Segmentation through the Detection of Changes Due to Motion. *Computer Graphics and Image Processing*, 11:13–34, 1979. Differencing techniques are used to correspond changing regions of images to moving objects. Refinement processes, such as gap filling, are used to aid better segmentation of (regions into) objects.

[Jon89] G. Jones. Carefully Scheduled Selection with ALT. *OCCAM User Group Newsletter*, 10:17–23, January 1989. An analysis and discussion on the implementation of fair ALT selection in OCCAM is presented.

[Kal60] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, pages 35–45, March 1960. Using sophisticated mathematical terminology, this is the original paper in which Kalman re-examines the classical filtering and prediction problems. It shows that the optimal estimate is the orthogonal projection of the true state upon the linear space spanned by the measurements.

[Koo88] K. Koontz. C004 Transfer Rate Problems. This was an electronic mail posting on the occam-request@uk.ac.oxford.prg newsgroup set up for the OCCAM and transputer community. It showed results for routing delays caused by the use of C004s., 1988.

[KR82] L. Kitchen and A. Rosenfeld. Gray-Level Corner Detection. *Pattern Recognition Letters*, 1(2):95–102, 1982. A corner detector is proposed for measuring cornerness in gray level images without prior segmentation.

[Lee83] C. C. Lee. Elimination of Redundant Operations for a Fast Sobel Operator. *IEEE Trans. on Systems, Man and Cyb.*, SMC-13(3):242–245, 1983. A useful report on the reduction of computation time in performing the Sobel operator is presented.

[Lie67] P.B. Liebelt. *An Introduction to Optimal Estimation.* Addison-Wesley, 1967. This book provides a very elementry , but first-rate, introduction to the topic of optimal estimation. It contains dedicated sections on the mathematical and probability theories used in the book.

[Low87] D.G. Lowe. Three-Dimensional Object Recognition from Single Two-Dimensional Images. *Artificial Intelligence*, 31:355–395, 1987. A process for transforming 2D image features to knowledge about the 3D scene is presented. This involves perceptual grouping of edge segments. Also presented is a technique on segmentation of linked points into straight line segments.

[MA78] W.N. Martin and J.K. Aggarwal. Survey: Dynamic Scene Analysis. *Computer Graphics and Image Processing*, 7:356–374, 1978. An early survey of the field which also introduced the ideas of peripheral and attentive processing.

[Mar80] D. Marr. *Vision.* Freeman, 1980. To aid visual perception, the author proposes three levels of representation for visual information: the primal sketch, the $2\frac{1}{2}D$ sketch and the $3D$ model representation.

[MCK+88] P. Morrow, D. Crookes, P. Kilpatrick, P. Milligan, and N. Scott. A Comparison of Two Notations for Programming Image Processing Applications on Transputers. In *Proceedings of the OCCAM User Group*, volume 7, pages 1–9, 1988. This paper describes a logical approach to routing data across an array of transputers. It also introduces a harness language called LATIN for performing imaging operations.

[ME91] M. Mirmehdi and T.J. Ellis. A Parallel Approach to Tracking Edge Segments in Dynamic Scenes. *Submitted for Publication*, 1991. This paper is currently at the IEE under consideration for publication.

[Mir88] M. Mirmehdi. Comparison of Parallel Processes in ADA and OCCAM. *Computing*, June 1988. This magazine article, published in two parts, consists of a review of the differences in ADA and OCCAM in handling communication and synchronisation between processes.

[Mir90] M. Mirmehdi. Product Label Inspection Using Transputers. In *Proc. of Second Int. Conf. on Transputer Applications 90, Southampton (TA90)*, pages 408–416. IOS Press, 1990. This paper summarises the work described in Chapter 3 of this thesis.

[Mir91] M. Mirmehdi. Product Label Inspection Using Transputers. *Concurrency: Practice and Experience*, 3(4):265–273, 1991. This is the same as the paper above which was selected to appear in an special issue of this journal on TA90.

[MS87] M.D. May and R. Shepherd. Communicating Process Computers. Technical Report 72-TCH-022-00, INMOS Ltd., 1987. The processor farm in its basic format is presented and applied to a ray-tracing example. Farming is also compared to pipelining.

[MWD91] M. Mirmehdi, G.A.W. West, and G.R. Dowling. Label Inspection using the Hough Transform on Transputer Networks. *Microprocessors and Microsystems*, 15(3):167–173, April 1991. Using two approaches in geometric parallelism, the problem of industrial inspection is examined by using a variation of the Hough transform specially suitable for parallelisation.

[Nag83] H.H. Nagel. Overview on Image Sequence Analysis. In *Image Sequence Processing and Dynamic Scene Analysis, ed. T.S.Huang*, pages 2–39, 1983. A review paper outlining some motion analysis techniques and some attempted applications up to 1983.

[Nib85] W. Niblack. *An Introduction to Digital Image Processing.* Prentice Hall, 1985.

[OHRS90] K. Obermayer, H. Heller, H. Ritter, and K. Schulten. Simulation of Self-Organising Neural Nets: A Comparison between a Transputer Ring and a Connection Machine CM-2. In *Proc. Third Int. Conf. of NATUG*, pages 95–106, 1990. Benchmark studies are presented to compare the performance of parallelised *Self-Organising Feature Maps* on two parallel systems. Of interest is the mapping of neuron nodes to transputer nodes.

[Oka84] Y. Okawa. Automated Inspection Of The Surface Defects Of Cast Metals. *Computer Vision, Graphics, and Image Processing*,

25:89–112, 1984. The author presents a method of detecting surface defects of cast metals.

[Pag88] I. Page, editor. *Parallel Architectures and Computer Vision.* Clarendon Press, 1988.

[PM87] D. Pountain and D. May. *A Tutorial Introduction to OCCAM Programming.* Inmos Document 72 OCC 046 00, 1987.

[PNRC90] D. Prior, M. Norman, N. Radcliffe, and L. Clarke. What Price Regularity? *Concurrency: Practice and Experience*, 2(1):55–78, 1990. The authors review various properties of irregular graphs and suggest their use as useful maps for connecting processors together in distributed memory machines.

[Pra85] T.W. Pratt. Pisces: An Environment for Parallel Scientific Computation. *IEEE Software*, 2(4):7–20, 1985. A virtual machine which provides a simulated MIMD environment for parallel computation using processes that can communicate asynchronously.

[Qui87] M.J. Quinn. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill, 1987.

[RD86] A.W. Roscoe and M. Dathi. The Pursuit of Deadlock Freedom. Technical Report Monograph PRG-57, Oxford University Computing Laboratory, 1986. This booklet describes a CSP approach to proving programs free of deadlock.

[Red88] S. Redfern. Implementing Data Structures and Recursion in OCCAM. Technical Report 72-TCH-038-00, INMOS Ltd., 1988. A brief technical report with tips on handling queues, stacks and lists, including methods of coping with recursion.

[Ree84] A.P. Reeves. Survey: Parallel computer architectures for image processing. *Computer Vision, Graphics, and Image Processing*, 25:68–88, 1984. Pipeline, SIMD and MIMD structures for image analysis are surveyed, and a simple model for the comparison of such systems is proposed.

[ROH88] A. Rosenfeld, Ornelas.J., and Y. Hung. Hough Transform Algorithms for Mesh-Connected SIMD Parallel Processors. *Computer Vision, Graphics and Image Processing*, 41:293–305, 1988. Different techniques in improving the performance of the Hough transform on SIMD arrays are presented and tested on the GAPP and MPP.

[Ros83] A. Rosenfeld. Motion: Analysis of Time-Varying Images. *Fundamentals in Computer Vision, ed O D Faugeras*, pages 173–183, 1983. A short review paper providing an outline of pixel-based and region-based techniques in estimating motion, followed by some remarks on the inference of structure from motion.

[Ros88] A. Rosenfeld. Computer Vision: Basic Principles. *Proceedings of the IEEE*, 76(8):863–868, 1988. This *invited paper* provides a general review of the techniques, operations, and architectures used in computer vision.

[RP80] W. Reichardt and T. Poggio. Figure-Ground Discrimination by Relative Movement in the Visual System of the Fly, Part I: Experimental Results. *Biology and Cybernetics*, 35:81–100, 1980. The motion detection and measurement in the visual system of the fly is based on non-linear multiplication-like interactions between adjacent pairs and groups of photoreceptors. A fly can detect and track a figure which moves relative to a ground of similar texture.

[RPBK90] M. Rygol, S. Pollard, C. Brown, and J. Kay. MARVIN & TINA: A Multiprocessor Vision System. In *Proc. of Second Int. Conf. on Transputer Applications 90, Southampton (TA90)*, pages 218–225. IOS Press, July 1990. The authors present a multi-transputer vision system (using specially developed hardware) for recovery of 3D scene geometry, and for control of a robot arm.

[RW89] P.L. Rosin and G.A.W. West. Segmentation of Edges into Lines and Arcs. *Image and Vision Computing*, 7(2):109–114, 1989. This paper summarises line and arc segmentation algorithms based on Lowe's work on perceptual organisation.

[SBC+89] M. Stephens, R. Blissett, D. Charnley, E. Sparks, and J. Pike. Outdoor Vehicle Navigation using Passive 3D Vision. In *Proc. 1989 IEEE Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 556–562, June 1989. The ego-motion of a camera mounted on an outdoor vehicle is determined by tracking feature points such as corners. Also, triangulation techniques are used to form 3D surfaces to determine navigable areas in images.

[Sch89] R.J. Schalkoff. *Digital Image Processing and Computer Vision.* John Wiley & Sons, 1989.

[SH88] M. Stephens and C. Harris. 3D Wire-Frame Integration from Image Sequences. In *Proceedings of the 4th Alvey Vision Conference*, pages 159–165, 1988. Edge-vertices are used as features in the DROID system as a preparation for determining structure from motion.

[Sha89] J. G. Shabushnig. Inspection of Pharmaceutical Packaging with Linear-Array Video Sensors. In *Society of Manufacturing Engineers Vision 89 Conference*, pages 13–23, 1989. The inspection of label position on a cylindrical bottle and verification of proper child-resistant closure on the bottle are examined.

[Sho84] J. E. Shore. Second Thoughts on Parallel Processors. In *Computing and Electrical Engineering*, volume 1, pages 95–109, 1984.

This paper present another classification of parallel processors by dividing them into six classes depending on the amount of processing and memory hardware.

[SJ84] M.A. Shah and R. Jain. Detecting Time-Varying Corners. In *7th Int. Conf. on Pattern Recognition*, volume 1, pages 2–5, 1984. A corner detector is developed combining the Zuniga-Haralick Corner Detector [ZH83], and the differencing of two images in a sequence.

[SJ87] I.K. Sethi and R. Jain. Finding Trajectories of Feature Points in a Monocular Image Sequence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(1):56–73, 1987. This is an important paper addressing the problem of correspondence. It examines point tracking by taking the ideas of smoothness of motion and path coherence into consideration. It selects the most smooth trajectory from the possible set as the most appropriate track for a point.

[STD90] G. A. Stephen, C. A. Taylor, and E. L. Dagless. Real Time Image Analysis for Dynamic Displacement Measurement. In *Proc. of Second Int. Conf. on Transputer Applications 90, Southampton (TA 90)*, pages 211–217. IOS Press, July 1990. A correlation technique is used by the authors to find the match for a template in a search area. A least squares method is then used for tracking.

[TB81] W.B. Thompson and S.T. Barnard. Lower-Level Estimation and Interpretation of Visual Motion. *IEEE Computer*, pages 20–28, August 1981. Thi paper reviews intensity-based and token-based techniques.

[TC89] S.M. Thomas and Y.T. Chan. A Simple Approach for the Estimation of Circular Arc Center and its Radius. *Computer Vision, Graphics, and Image Processing*, 45:362–370, 1989. This is a short note for the estimation of the center and radius of a circular arc by minimising the least-mean-square errors between the given set of data points and the curve.

[TD90] R. W. Tregidgo and A. C. Downton. Processor Farm Analysis and Simulation for Embedded Parallel Processing Systems. *Proc. of 12th OCCAM User Group Technical Meeting*, pages 179–189, 1990. The authors provide a mathematical analysis of processor farms and attempt to show that general-purpose multi-processor systems can be analytically designed.

[THKS88] C. Thorpe, M.H. Hebert, T. Kanade, and S.A. Shafer. Vision and Navigation for the Carnegie-Mellon Navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-10(3):362–373, 1988. Using low resolution images, this work describes the implementation of a colour classification algorithm for road-edge following and 3D vision techniques for obstacle detection and avoidance.

[TLM+90] B.T. Thomas, R.A. Lotufo, A.D. Morgan, D.J. Milford, and E.L. Dagless. Real-Time Image Analysis for Vision Guided Control.

In *IEE Proceedings of UK IT Conference*, pages 66–70. IEE, March 1990. This paper implements an edge tracking algorithm and a surface segmentation technique, both specifically for road-edge following, on a multi-transputer architecture.

[TM86] C.L. Tan and W.N. Martin. A Distributed System for Analyzing Time-Varying Multiresolution Imagery. *Computer Vision, Graphics and Image Processing*, 36:162–174, 1986. The authors present a hierarchical, pipelined, multi-resolution object tracking system implemented on a VAX machine, using the PISCES MIMD message-passing simulation package.

[Tre88] P. C. Treleaven. Parallel Architecture Overview. *Parallel Computing*, 8:59–70, 1988. This paper provides an overview of some recent parallel architectures, and discusses their likely commercial impact.

[Ull79] S. Ullman. *The Interpretation of Visual Motion.* The MIT Press, 1979. Using artificial intelligence methodology to investigate the phenomena of visual motion perception, the author attends to the problems of correspondence and structure from motion.

[Ull81] S. Ullman. Analysis of Visual Motion by Biological and Computer Systems. *Computer*, 14:57–69, August 1981. This paper divides motion measurement techniques into intensity-based and token-based schemes. It concludes by an update on the state of the recovery of structure from motion.

[VTL90] E. Verhulst, H. Thielemans, and K. Leuven. Preemptive Process Scheduling and Meeting Hard Real-Time Constraints with TRANS-RTXc on the Transputer. In *Proc. of Second Int. Conf. on Transputer Applications 90, Southampton (TA90)*, pages 288–295. IOS Press, 1990. The authors discuss several design features of a transputer operating system capable of running processes at multiple priority levels.

[WH90] E.L. Walker and M. Herman. Geometric Reasoning for Constructing 3D Scene Descriptions from Images. *Artificial Intelligence*, 34:275–290, 1990. This paper describes a method for completing object descriptions and performing model matching, using model domain knowledge and geometric reasoning.

[WP89] J. Wexler and D. Prior. Solving Problems with Transputers: Background and Experience. *Microprocessors and Microsystems*, 13(2):67–78, 1989. A general survey of distinctive characteristics of transputer-based concurrent system's design using OCCAM is presented.

[WRHR91] C. Weems, E. Riseman, A. Hanson, and A. Rosenfeld. The DARPA Image Understanding Benchmark for Parallel Computers. *Journal of Parallel and Distributed Computing*, 11(1):1–24,

1991. A detailed study and comparison of a set of image analysis tasks on various parallel computers is presented. The benchmark itself is also examined.

[Yam83] T. Yamamura. Automated Label Inspection Apparatus. In *Information Processing 83*, pages 169–172, 1983. A linear scan approach for the inspection of labels on Whisky bottles is presented.

[YIT80] M. Yachida, M. Ikeda, and S. Tsuji. A Plan-Guided Analysis of Cineangiograms for Measurement of Dynamic Behaviour of Heart Wall. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(6):537–543, 1980. Movement of heart wall boundaries are detected by applying edge filters to differenced images, and using correspondence between portions of the wall between successive frames.

[ZH83] O. A. Zuniga and R. M. Haralick. Corner Detection Using the Facet Model. In *Proc. 1985 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 30–37, 1983. The pixel values in a given neighbourhood are considered to be discrete quantised noisy samples from an underlying continuous gray tone intensity surface. Using this assumption, the cornerness of the pixel neighbourhood is measured.

Appendix A

Hardware and Software Lists

The following provides a listing of the hardware and software systems used at various stages of this work. The list in **A.1** corresponds to the work in Chapters 3 and 4, and the list in **A.2** corresponds to the work in Chapter 6.

A.1 Label Inspection

The transputer modules in the list below are early, non-standard items from Transtech Ltd.

- Standard IBM-PC compatible computer with an 8MHz 80286 processor, EGA colour monitor, and 20Mbyte hard disk.
- An INMOS B004-compatible board designed to slot into an IBM-PC compatible computer. The board contains a T800-20MHz transputer which is connected to a PC port via its link 0. It has 2Mbytes of RAM accessable at a rate of 4 processor cycles. Earlier, the board had been fitted with both a T414-12MHz chip and a T800-20MHz chip, and both of these have also been used for quoting some figures in the thesis.
- Transtech TSMB-16 Module motherboard mounted on a customised Transtech metal box with a power unit. The board is capable of housing 16 vertically mounted transputer modules, and contains a C004 crossbar chip which is accompanied with basic configuration software.
- Four TSM42 20MHz modules, each with 1 Mbytes of RAM with an access rate of 4 processor cycles fitted onto the TSMB-16 motherboard. The module links are capable of bi-directional data transfer at rates of up to 2.35Mbytes/sec.
- Data Translation DT2853 512x512x8-bit frame-grabber board fitted into the PC. It has two on-board memory buffers, and features a

hardware cursor, LUT processor, and 1:1 pixel aspect ratio. This frame-grabber was interfaced using specially written software to communicate with the B004 transputer via the PC server program (all as part of TIPS).

- Standard CCD camera with 12.5-75mm F1.8 6X zoom lens.
- An EIZO 3010 12" monochrome monitor, used for viewing images.
- The TDS development environment version D700B, later D700C.
- Development and use of TIPS (version 1.0 and later 2.0).

A.2 Correspondence Analysis and Token Tracking

The transputer hardware in this section is completely industry-standard, except for the Harlequin frame-grabber board.

- Standard IBM-PC compatible computer with an 8MHz 80286 processor, EGA colour monitor, and 20Mbyte hard disk.
- An INMOS B004-compatible board designed to slot into an IBM-PC compatible computer. The board is connected to a PC port via one of its links. It has 2Mbytes of RAM with an access rate of 4 processor cycles.
- A Harlequin transputer-based, 512x512x8-bit, frame-grabber board designed to slot into an IBM-PC compatible computer. It has a T800-20MHz transputer with 1 Mbyte of RAM with an access rate of 4 processor cycles, in addition to its two dual-ported image buffers. This board is compatible with the INMOS B007 graphics board.
- A B012 Eurocard TRAM motherboard with slots for up to 16 TRAMs where TRAMs are TRAnsputer Module boards, housed in customised Transtech metal box with power unit.
- Eight TRAMs each with a T800-25MHz transputer and 1 Mbyte of RAM with an access rate of 3 processor cycles. These are 32-bit processors with a 64-bit floating-point unit, 4K on-chip SRAM, and 4 links capable of bi-directional data transfer at rates of up to 2.35Mbytes/sec per link.

- The Harlequin's graphic display generates its output via a raster scan through a block of memory. The bytes accessed as such are then converted into an 18-bit colour combination using the board's Red, Green and Blue output channels via a programmable LUT. This allows the use of up to 256 colour combinations which were used for the display of the results shown in Chapter 6.
- Eight further TRAMS, each with a T805-25MHz transputer and 1 Mbyte of RAM each with an access rate of 4 processor cycles. The T805 is essentially a T800 with extra processor instructions used for debugging purposes.
- A Packard Bell colour monitor used to view the colour image results.
- Standard CCD camera with 12.5-75mm F1.8 6X zoom lens.
- MMS2: Standard INMOS network configuration software.
- The TDS development environment version D700D.
- Further development and use of TIPS (version 3.0).

APPENDIX A. HARDWARE AND SOFTWARE LISTS