Parallel Architectures for Image Analysis

Nicholas David Francis

Submitted to the

Department of Computer Science

for the degree of Doctor of Philosophy

at the

University of Warwick

September 1991

Abstract

This thesis is concerned with the problem of designing an architecture specifically for the application of image analysis and object recognition. Image analysis is a complex subject area that remains only partially defined and only partially solved. This makes the task of designing an architecture aimed at efficiently implementing image analysis and recognition algorithms a difficult one.

Within this work a massively parallel heterogeneous architecture, the Warwick Pyramid Machine is described. This architecture consists of SIMD, MIMD and MSIMD modes of parallelism each directed at a different part of the problem. The performance of this architecture is analysed with respect to many tasks drawn from very different areas of the image analysis problem. These tasks include an efficient straight line extraction algorithm and a robust and novel geometric model based recognition system. The straight line extraction method is based on the local extraction of line segments using a Hough style algorithm followed by careful global matching and merging. The recognition system avoids quantising the pose space, hence overcoming many of the problems inherent with this class of methods and includes an analytical verification stage. Results and detailed implementations of both of these tasks are given.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Professor Graham Nudd, my supervisor, for enabling this work to take place and for his guidance throughout it. I would also like to thank John, Rolf, Darren, Dave and Roger for their support along the way. Thanks are also due to my family, especially my wife Lyn, and in addition to those friends mentioned above, to Chris 'n' Jenny and Paul for keeping me sane - or at least for trying.

Thanks also to Dr. Roland Wilson and Professor Mike Paterson for their occasional words of advice.

Declaration

The work described in this thesis has also appeared to some degree in the following published works which I have co-authored, [Francis 91], [Nudd 91], [Francis 90a], [Nudd 90], [Atherton 90a], [Atherton 90b], [Francis 90b], [Vaudin 89], [Nudd and Francis 89a], [Nudd 89b], [Nudd 88a], [Nudd 88b], [Howarth and Francis 88].

The work on the Warwick Pyramid Machine described in chapter 3 was jointly undertaken by myself and others in the architectures section of the VLSI group. Substantive work given in chapters 4, 5 and 6 is entirely my own as is the analysis given in chapters 2 and 7.

1

Introduction

Image analysis involves forming a useful description of a scene given one or more images of that scene. Machines capable of performing image analysis have great potential value to humans as the human world is generally a visual world. If a machine is to be able to interact with that world in a meaningful way, unless the world is constrained, the machine needs to be able to see what is around it, and more importantly to be able to understand what it sees. Enabling a machine to be able to sense the visual world is simply a matter of supplying it with a video camera. However being able to understand the output of that camera is a very different matter. As the Nobel prize winning neurobiologist David Hubel says in "Eye, Brain and Vision" [Hubel 88] :-

> 'The eye has often been compared to a camera. It would be more appropriate to compare it to a TV camera attached to an automatically tracking tripod - a machine that is self-focusing, adjusts automatically for light intensity, has a self-cleaning lens, and feeds into a computer with parallel-processing capabilities so advanced that engineers are only just beginning to consider similar strategies for the hardware they design. The gigantic job of taking the light that falls on the two retinas and translating it into a meaningful visual scene is often curiously ignored, as though all we needed in order to see was an image of the external world perfectly focused on the retina'

This thesis is concerned with what happens to the output from the camera, how to process it and how to design the computer system that performs the processing.

The output from the camera can generally be considered to be a large two dimensional array of numbers which represent light intensities; this is similar to the output from the retina. At the other extreme of the image analysis system there are descriptions of objects. Humans know what objects such as tables, books, houses etc. look like and if a new object is found then it may be learned. If it is not known what an object looks like then that object cannot be recognised. An image analysis system therefore needs to be given these descriptions in order for it to be able to analyse images or recognise objects.

Therefore the recognition system is already beginning to appear more complex. There have been identified two very different forms of data, an array of light intensity values and some abstract descriptions of objects. The next problem is the speed of the system. Humans expect to be able to recognise objects reasonably quickly, probably not in a single frame (1/25th second) but still quickly enough to be able to react to the presence of the object. This is also the case for a machine guided by visual input - for example, there

is little point in being able to recognise that the object is a bus several hours after it has left. A further problem is the cluttered nature of the human world. Objects are unlikely to be always conveniently lying individually on well contrasted uniform backgrounds. On the contrary, they are quite likely to be occluded or partially in shadow. Sometimes even human visual systems need to take a second look at things to be able to recognise them.

It is well beyond the state of the art in image analysis systems to be able to construct and program a machine to be able to perform this proficiently [Rosenfeld 87] [Weems 89]. This development has been limited both by lack of understanding not just of the algorithms but the problem itself, and by the remarkable amount of computation required for even simple image analysis problems. The apparent solution to the scale of the computational requirements is simply to use more processors. The difficulty of the problem itself is often partially eased by changing the world in which the machine is to operate, by, for example, painting the object black and the background white or by other similar techniques. This can work well in controlled circumstances but is unlikely to be of use in real environments.

The subject of this thesis is an attempt to improve the understanding of the architectural requirements of the problem of image analysis given the current understanding of both computers and of the way in which the problem is solved.

The analysis commences in chapter two by attempting to extract the generic requirements of image analysis as described by other researchers. These requirements are at the level of, for example, the amounts of data at each stage of processing, the types of data, the locality of data accesses etc. Next current computer architectural techniques are described. It is then possible to match the requirements of image analysis to the best fitting model of computation. This leads to a prototype computer architecture that attempts to match the requirements of image analysis. The prototype architecture, the Warwick Pyramid Machine (WPM), is described in chapter three. The description includes the motivation for each individual feature of the

architecture and details of how that feature is implemented in practice. This prototype architecture is intended to provide a test-bed for research into matching computer architectures to image analysis. One of the difficulties encountered within this work was wide disagreement upon some of the basic principles of image analysis. Given this disagreement on the problem it is difficult to construct an architecture to assist the solution of that problem.

Chapters four, five and six describe, in some depth, several image analysis algorithms and implementations of those algorithms on the proposed architecture. These include two complete image analysis tasks. The first of these tasks simply involves locating and counting the number of particles of varying sizes in an image. The second task is described in more detail and involves the location of objects in the image based on their shape, given a geometric model of the object. This task is at the leading edge of image analysis research. The principle behind the solution given here is to limit the problem to the recognition of two dimensional objects and then tackling that problem in depth to create a more robust system than previous efforts.

Study of these tasks then allows a reasonable assessment of the performance of the machine. Clearly, tasks can be devised that have different requirements, but the tasks tested do have a reasonable range and offer a significant test for the architecture given in chapter three. Chapter seven analyses how this architecture coped with each task. Every important feature of the architecture is examined and the value of it is tested. This finally leads to a much better understanding of the requirements of image analysis and also a better understanding of how these may be met by computer architectural features.

This thesis contains contributions to these fields in the following specific areas:

- A powerful prototype parallel architecture designed for image analysis applications is described.

- A novel detection and segmentation system for a range of different sized 'blob' like objects is given.

- Several novel implementations of commonly used low level image analysis algorithms are detailed that are also of significant value for other parallel architectures.

- A straight line extraction method is described which eliminates many of the problems often encountered when extracting straight lines from images. An efficient method is given for implementing this on the WPM that gives an order of magnitude improvement in performance in addition to an improvement in the quality of the results over that obtained by straight line extraction algorithms on other architectures.

- A robust and accurate two dimensional object recognition system is given that is an improvement over many existing systems in the following ways:

  Two dimensional objects of any scale may be found given a single model of the object.

  The technique of transformation sampling of Cass [Cass 88] is extended for the case of scale which allows more accurate clustering of the data in the pose space.

  Multiple resolution non-discrete transformation clustering is used that does not involve any pruning of the search space and hence significantly reduces the chances of accidently missing a correct match.

  For the first time an analytical transformation adjustment and consistency verification method for a system involving scale is derived and used. This allows weaker objects to be located as the verification stage is more powerful and also gives a more accurate final result.

An efficient load-balanced implementation is given for the Warwick Pyramid Machine.

A method is suggested which allows object features that do not normally fit into geometric model based recognition techniques to be incorporated into the analysis phase.

- A detailed account of the architectural requirements of many image analysis tasks is given.

2

Architectures for Image Analysis

2.1 Introduction

One of the prime purposes of the work covered by this thesis is to investigate parallel architectures that are well matched to the problem of image analysis. This investigation is based on the assumption that an architecture which provides an efficient match to a problem will be a good architecture to use to solve that problem. The design of such an architecture, aimed at a specific application, should be able to exploit the knowledge of the structure of the data and the behaviour of the algorithms.

This requires an understanding of the problem, the algorithms, the current solutions and their deficiencies. This chapter first describes, in very general

terms, the image analysis problem and how it is conventionally approached, what the classes of algorithms developed attempt to do and what their requirements are. Secondly, parallel computer architectures are examined, and the advantages and limitations of each important architectural model are described. Then the important architectures proposed by other researchers for the image analysis problem are briefly described and some conclusions are made. This chapter is not intended to contain a comprehensive review of all image analysis methods or all parallel architectures. It is aimed more at describing what broad conclusions may be learnt from previous work in this area.

2.2 Image Analysis

2.2.1 Introduction

Image analysis may be defined as the decomposition of an image into meaningful descriptions of the objects present in the actual scene. The area of research into devising algorithms for performing image analysis remains immature, however some general observations may be made.

Generally the first operations that are applied to the image both in the retina [Hubel 88] and in the computer [Weems 91] are simple image to image, iconic or retinotopic operations. These are the best understood and modelled areas of both human and artificial visual systems. However while these operations may highlight features, it is generally true that the information contained in the image is not enough to perform an analysis of the scene. In addition to the image data some knowledge is needed concerning the objects that may occur in the image. For example, an object such as a table cannot be recognised if there is no description of a table available. This knowledge or description may take many forms but in general it cannot be iconic for the table may be orientated in an unknown number of ways, be of any size, or even be occluded. If the information is iconic in form then this restricts the recognition process to a limited set of scenarios. This therefore implies that

there are at least two different forms of data to be processed by an image analysis system, the images or iconic data and the description or symbolic data.

Given these two forms of data an important consideration is how the translation may take place from the iconic to this symbolic form in order to perform recognition. Note that the translation could be performed in reverse or from symbolic to iconic, by, for example, rendering the models and searching for matches at the iconic level. This method is generally not used as it requires a very large number of different scale and orientation models to be tested. Therefore researchers [Hildreth 88] [Nudd 88] [Weems 91] [Tanimoto 86] [Duff 90] break the problem down into three levels of abstraction. These levels are iconic, where the data is image based; intermediate which consists of non-iconic structures which are derived directly from the image; and symbolic, where the data consists of descriptions or abstract structures such as pose or correspondence spaces.

If each of these are considered in turn then it is possible to extract the generic operations and features of each level of processing.

2.2.2 Iconic

As described above, the iconic processing is conventionally considered to be the image to image operations that are applied in the initial stages of the image analysis system, whether that system be animal or machine. This type of processing is perhaps the best understood and modelled and frequently is based on signal processing techniques.

Examples of the kinds of operations that are applied to images at this stage are enhancements to remove sensor noise such as frequency based filtering and median filters, image transforms, edge detection, thresholding, region growing, connected component labelling, simple shape detection and segmentation.

It is unlikely to be possible to design a single architecture that is optimal for each of these tasks. This is certainly the case if economic factors are taken into account. Therefore there is a need to extract the general requirements of the problem in an attempt to be able to specify an architecture that can meet these requirements. The first and most obvious fact is that the data, both input and output, consists of a two dimensional array of values which is directly derived from the sensor. This gives a large amount of inherent data parallelism.

Secondly, a significant amount of the processing is uniform across the whole image. This tends to be the case, somewhat simplistically, because at these stages it is not possible to treat different parts of the image in a different way as there is no information about what is contained in these different parts of the image. This assumption does break down to a certain degree for those operations where the processing flow is data dependent or for succeeding images in a sequence.

Generally, the precision of the input data to the algorithms listed is low [Weems 91]; most sensors produce around 8 bits of precision. Many algorithms will slightly increase this precision as a result of operations such as additions or multiplications, whereas other algorithms will process binary data resulting from some threshold operation.

Many of the communications patterns tend to involve local communication due to the retinotopic nature of the image. Generally image features are local, and therefore the computation used to extract them requires access to local pixels. At this stage of processing communications tend to become global only when the image is transformed into a non-retinotopic representation such as the frequency domain. The most significant exception to this that occurs while data remains in this configuration is when individual regions or features are extended across the image and properties across the whole feature are required. This tends to blur the distinction between the iconic and intermediate levels of processing.

2.2.3 Intermediate

Tanimoto [Tanimoto 86] defines intermediate level processing as follows: 'A transformation of intermediate-level vision is one that takes as input an image represented as a two-dimensional array, and outputs a structure that is *not* a two-dimensional array'. The intermediate level tends to be less well understood than the iconic layer. Whereas in iconic processing the goals are reasonably well accepted and the algorithms to implement those goals less so, in intermediate level processing there are few generally accepted goals and even fewer algorithms. This makes the task of identifying generic operations somewhat more difficult.

As defined above, the intermediate level is not a processing level as such but more a translation level, converting images to lists of features. The types of operations that fit this definition are the extraction of lines, endpoints, points of high curvature, parallel lines, areas, texture elements, perimeters, dimensions, moments and histograms. The output is typically a list of parameters often with some relationship to coordinate values or dimensions.

The intermediate level has the following requirements. Data should be able to be extracted over an area of the image without having to read in large numbers of pixel values. It needs to be able to read coordinate values from pixels that are in a particular condition and measure areas and lengths. Statistics should be able to be formed, the most important of which is existence, i.e. whether something is present or not. If, for example, endpoints are being extracted from an image and put into a list this should be able to be performed in parallel and therefore there should be multiple points of access to the image.

2.2.4 Symbolic

This layer gives context to the features extracted by the intermediate level, it extracts meaning from the input data by associating it with the information it holds about objects. This approach is somewhat top-down for while the

iconic layer will perform a small number of processes that tend to be independent of the object sought, such as enhancement, the features that are extracted are determined by the objects that are sought by the symbolic layer. However the analysis may not be completely top-down for it is not a feasible approach to attempt to convert the description into iconic data and match at that level.

This layer compares sets of features extracted from the whole image with those stored in its databases. These may be in the form of rigid geometric models, points in multidimensional vector space, nodes in trees or graphs or one or more of many other representations.

This level is not well understood and little undisputed research is available about the requirements of this level beyond specific examples. Generally researchers suggest that algorithms such as graph manipulation or 'logic' [Weems 91] take place at this level and the apparently intuitive belief is that the process requires lower granularity with unstructured control and communications [Arbib and Hanson 87]. There is less agreement on the amount of data to be processed at this level, some believe that it is significantly less at this stage than the iconic level [Cantoni 86] whereas others believe that it is in fact greater than the iconic stages [Hanson 86].

2.3 Architectures

2.3.1 Introduction

Image analysis is a complex and computationally intensive task. Figures such as the following are much quoted [Nudd 89] [Weems 91] concerning the computational requirements. A single medium size convolution filter such as a 7 by 7 filter requires 3 million multiply-accumulate operations when applied to a 256 by 256 pixel image. This is only one of many operations that need to be performed, and this is to a single image - a real time system can expect many images per second. It has been shown [Nudd and Francis 89]

[WSTL 89] [Taylor 85] that single microprocessors are not close to being adequate for this task.

It is very fortunate therefore that the problem does not have to be confined to the use of sequential processors. Image analysis, as described above, has a very large amount of implicit parallelism not only in the data but also in the operations which tend to be able to be performed in a pipelined fashion on successive images. Due to the dual facts that image analysis is too computationally intensive for sequential architectures, and that it is one of a small number of applications from which it is trivial to extract a large amount of parallelism, it has become a leading application driving the development of massively parallel architectures. This has led to a very large number of architectures being developed specifically for this application [Duff 85] [Nudd 88] [Weems 89].

2.3.2 Parallel Processing

Parallelism is a difficult term to define. Does a processor that operates on a number of bits at a time, such as 32 in a 32 bit microprocessor, exhibit parallelism? Modern microprocessors have multiple functional units that may operate simultaneously using instruction pipelining. In both of these cases, while the processors may be considered to contain some degree of concurrency, they are generally not considered to be parallel architectures per se.

Classification of Parallel Architectures

In order to understand this better it is useful to look at how parallel architectures are classified. Many attempts have been made to classify these architectures but only one, that of Flynn [Flynn 66], has obtained widespread use. He divided parallel architectures into the well known 'organisational classes' based on the number of instruction and data streams.

The first classification is 'confluent' (concurrent) Single Instruction Single Data (SISD). In this organisation although several operations may be acting

concurrently the bottleneck is that only a single instruction is being decoded at a time and that there is only a single path to data memory. This organisation is illustrated in figure 2.1 where C is the control or source of the instruction stream, P is the processor and M is the data memory.

Figure 2.1 - SISD Organisation

The second class of parallel architecture is the Single Instruction Multiple Data (SIMD) illustrated in figure 2.2. Again only a single instruction is decoded at once, but in this case that instruction acts concurrently on a number of operands.

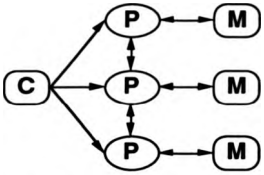


Figure 2.2 - SIMD Organisation

The third class is Multiple Instruction Single Data (MISD). This class of architectures causes much confusion and is not well defined. Flynn suggests that a machine in which each processor has local independent instruction streams but has all of the data stored in a shared memory which each processor may randomly access is MISD (figure 2.3(a)). This organisation has more recently been considered simply as a shared memory MIMD machine as each processor may in fact operate on a different operand. The other architecture suggested by Flynn for this class is a pipelined machine in which each processor, with its independent instruction stream, operates on the derived data from the previous processor (figure 2.3(b)). This is how image

processing systems constructed from sets of dedicated hardware tend to function. This is also now generally considered to be MIMD.

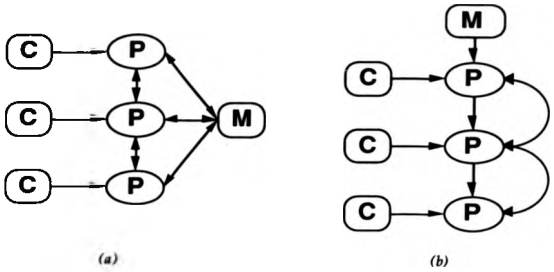


Figure 2.3 - MISD Organisation

In more recent times MISD has been taken more strictly to mean a system that applies multiple instructions to a single operand and has thus been deemed impractical [Duncan 90].

The remaining class is the Multiple Instruction Multiple Data (MIMD) architecture. This consists of a set of processors with independent instruction and data streams. These architectures appear as a set of conventional SISD processors with the addition of communications mechanisms to allow the processors to act together.

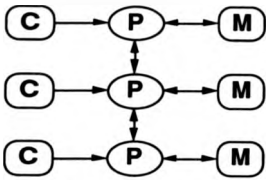


Figure 2.4 - MIMD Organisation

Therefore only two of Flynns' classifications are of significant value in a discussion of parallel processors, these are SIMD and MIMD.

Inter-Processor Communications

In order that an architecture consisting of a number of processors may operate as a single machine the processors must be able to share data with each other. This requires that a form of communications is set up between the processors. In general it is required that any processor should be able to pass data to any other processor and therefore there must either be a path from any processor to all other processors or a system of forwarding communications must be used. The way these processors should be connected together depends on the amount of data that is likely to be passed, the number of processors in the system and the likely nature of the communications patterns which is derived from the algorithms to be used.

This leads to a major classification of parallel architectures based on how data is shared. Processors may share data by having a common memory which each processor may read from and write to thus allowing communication to take place. These architectures are known as shared memory architectures. Alternatively the processors may retain their entire memory locally and communicate through some other mechanism, these are known as distributed memory architectures.

Shared Memory Architectures

Shared memory machines may be broken down further into those machines where each processor's whole memory is shared and those machines where the processors have local non-shared memory for local data and only use the shared memory for communication purposes. This is further complicated by those machines in which the memory which is shared is distributed throughout the processors.

If the only memory a processor has access to is the single shared memory then this will quickly become a bottleneck limiting the number of processors that can be connected in this manner. This limitation may be eased by using local memory for data that does not need to be shared and only putting the data that is of interest to more than one processor in the shared area. This significantly eases the problem of contention as most data accesses are likely to be to local data. A further enhancement is frequently made by copying the shared memory into local caches where the processors may read and write without contention with other processors. However, this causes significant cache coherency problems as checks have to be made that no processor is trying to access the shared memory that has been copied out to another's cache. If instead, the shared memory is distributed then processors may access it simultaneously as long as two are not accessing the same bank. This may increase the performance without causing coherency problems at the cost of a communications network which allows each processor to access each memory bank.

Architectures with a single shared bank of memory generally have the memory connected to the processors by a shared bus or by multiport memory and these are limited to a relatively small number of processors due to contention problems. Those with distributed banks of memory tend to use more complex interconnection networks and consist of larger numbers of processors. This is still limited as these interconnection networks have to interconnect the buses which are generally many bits wide.

Distributed Memory Architectures

Distributed memory processors store their entire memory locally and only they are able to access it, therefore a different communications mechanism must be used to allow processors to share data. This is generally performed by message passing. Message passing requires less data to be transfered across the interconnection network than shared memory machines and therefore they generally use narrower data paths. Contentions do not occur as processors do not have access to another's memory and all communication is under the control of the processors. Therefore distributed memory architectures have fewer restrictions on the number of processors.

Inter-Connection Networks

In an ideal interconnection network each node would have a direct link to every other node, but this would, for medium or large numbers of nodes, be too costly. In this discussion a node is considered to be either a processor or a memory bank, processor to processor communication is required for distributed memory machines and processor to memory for shared memory machines. As exhaustive networks are generally not realistic, compromises have to be made and reduced interconnection networks are devised for connecting nodes. These networks break down into two major categories, static networks where each node can only communicate directly with a fixed set of other nodes and dynamic networks where the set of nodes any node may communicate with is not fixed.

Dynamic Networks

Dynamic networks form a path between two nodes via a set of routing switches. Hence a point to point communication is set up between any two nodes dynamically. If the path between any pair of nodes is via a single switch then the network is known as a single stage network, otherwise it is known as a multiple stage network. Examples of dynamic networks are crossbar switch networks, Clos, omega and delta networks. Differences

between these networks include the number of stages between a pair of nodes, whether a communication between two nodes may be blocked by the interconnection network being used by other nodes and the number and size of switches used by the interconnection network.

Static Networks

Static networks only allow nodes to communicate directly with those that they are directly connected to and therefore if a node is to communicate with any other node it is necessary that the intermediate nodes forward the message for it. These are, therefore, only really suitable for distributed memory architectures.

The critical factors of static networks are the length of the path between two nodes and the number of interconnects emanating from each node. In an exhaustive network of n processors the length of any path is 1 but the number of interconnects from each node is n-1. Generally the designer of an interconnection network attempts to trade-off one of these parameters against the other. This trade-off is bounded by engineering and economic constraints and also the problem for which the system is to be used. Ideally the network should be chosen to mimic the communication pattern of the data. A certain amount of flexibility can be obtained by the use of dynamically reconfigurable networks which may be reconfigured to some degree to suit the requirements of the algorithm.

One of the simplest and most common interconnection networks is the nearest neighbour. In this network the nodes are organised in a one or two dimensional array and connected to their immediate neighbours. Hence in a one dimensional array of n nodes two links are required per node and if the ends are wrapped around to form a ring the maximum path length is $n/2$. In a North, East, South, West (NESW) connected two dimensional array of dimensions $\sqrt{n}$ by $\sqrt{n}$ with the edges wrapped around to form a torus each node needs four links and the maximum path length is $O(\sqrt{n})$.

This may be extended to k dimensions with a hypercube. In a k dimensional hypercube each node has k links and there are 2^k nodes giving a maximum path length of k. The hypercube is a popular network as it offers a good balance of the maximum path length with the number of links emanating from each node. In addition, a very simple algorithm may be used to route messages to the correct destination. To do this the hypercube is organised such that the address of each node connected to a given node differs from it by one bit. Then to route a packet of data the node inspects the destination address of the packet and sends it to one of its neighbouring nodes that increases the match between the destination address and the address of that node.

Other static networks such as star and tree networks are occasionally used but they offer little over those mentioned above unless the data communication pattern happens to take that form.

Processor Granularity

The granularity of a parallel machine refers to the smallest level of synchronistation within that machine. Therefore a single bit processor that operates on one bit at a time is generally referred to as fine grain and a processor that operates on a large number of bits at a time is referred to as coarse grain. This classification is important when considering parallel processors because, in general, there are limits to the size of machine which is economically feasible to construct. Therefore as coarse grain processors are larger than fine grain processors, in general, an architecture consisting of fine grain processors can contain more processors than one consisting of coarse grain. This has meant that most massively parallel architectures have been constructed with fine grain processors.

There are various arguments for and against each level of granularity. Those most quoted are that the finer grain the processor is, the shorter is the theoretical cycle time that it may achieve, due to such things as carry propagation. Other arguments are based around efficiency as many

operations do not make use of the full width of the word in coarse grain machines and therefore a portion of the hardware is wasted for these operations. Counter arguments to these are often technology based. As vastly more research and development takes place on coarse grain machines then the state of the art always favours these architectures. Additionally as chip density increases this allows more processing functionality to be placed on a chip. Fine grain processors take advantage of this by an increase in the numbers of processors on each chip. At some point this ceases to become limited by the area of the chip and instead is limited by the number of pins on that chip, which increases only as the square root of the chip area. Coarse grain processors may take advantage of the increase in chip area by adding functionality such as adding a floating point unit. This may take advantage of the extra chip area without a large increase in the requirement for pins. Another argument is that coarse grain machines will always be fundamentally faster for those problems that have low limits to the amount of parallelism it it possible to extract.

2.3.3 Parallel Architectures

SIMD Architectures

The issues described in the previous sections have implications on the design of SIMD architectures. These implications give rise to constraints on the design of SIMD machines which may only be stretched to a limited degree by computer architects if wasted chip area is to be minimised.

In general, the hardware gain of SIMD architectures is achieved by having a single controller instead of many, thus saving the space used by the additional controllers. This gain clearly becomes more substantial if the number of processors per controller is large. Therefore SIMD architectures tend to consist of large numbers of fine grain processors. Fine grain processors need to be simple in order to justify their use, therefore there is a need to keep the communications mechanism as simple as possible. Shared memory cannot really be used for SIMD processor communication as the individual

processors do not generate their own addresses and hence all address the same location at once in different memory banks. Therefore for architectural efficiency reasons SIMD architectures generally consist of a large number of simple distributed memory processing elements connected with a simple interconnection scheme.

MIMD Architectures

MIMD architectures tend to have somewhat opposite constraints to the SIMD machines. As each processor has its own controller then making it coarse grained does not add significantly to its complexity. In addition, as already pointed out, MIMD machines appear as if they consist of SISD processors with the addition of communications mechanisms. This has not been missed by the designers of MIMD machines and they commonly make use of the state of the art in sequential processors and add interconnection networks to these to produce MIMD machines.

Associative Architectures

Associative or content addressable memories have long been used in situations where the memory is to be addressed by its contents rather than by its address. In associative memories a field is broadcast to the memory, each cell compares this field with what is stored in that cell and if it matches it sets a flag to note the match. In some cases there will be no match and this has to be recognised, in other cases there will be multiple matches and again some circuitry needs to be added to deal with this case. It is common for content addressable memories to have mask bits to enable a three level logic such as 'this bit should be set', 'this bit should be clear', and 'don't care'. Associative memories tend to be somewhat larger than conventional memories and hence only get used in some specific circumstances such as caches.

It has occurred to computer designers that the bit serial, fine grain, SIMD processors are not very far removed from the concept of associative memories. SIMD machines have the broadcast capability and operate on the same instruction, therefore they may all simultaneously compare the

broadcast value with the stored value and set a flag bit if they match. The processors may implement any depth word by iterating through the word comparing each bit in turn. Therefore it is common for massively parallel SIMD architectures to be enhanced in minor ways to improve this facility.

Other Architectures

In addition to those architectures proposed by Flynn there have evolved several that do not fit very well into his organisation. These include heterogeneous and hybrid machines, dataflow, pipeline and systolic machines and even neural networks. Of these only hybrid and heterogeneous will be considered here as the others are either rather poorly defined or of limited generality. Heterogeneous machines allow several different types of processors to be used in a machine recognising the fact that some types of processors are better for some tasks than others. The hybrid machine is an extension of that philosophy to recognising that some parallel architectures are better at some tasks than others and incorporating several different modes of parallelism in a single design. An example of these ideas might be using a SIMD machine to process the vector component of a problem and a powerful sequential processor to process the scalar component; this method is often used by supercomputer manufacturers. Another is to use dedicated signal processing hardware to perform the enhancement of an image and more general purpose hardware to analyse it.

2.4 Image Analysis Architectures

Having briefly described the general requirements of image analysis and the principle features directing the design of parallel architectures this section will describe particular architectures that have interesting or relevant features for this application. This is by no means intended to be a complete review of the intricate details of every parallel architecture. Instead certain general features that have been applied to aspects of the image analysis problem are

described. For a more detailed review see [Howarth 91] [Nudd and Francis 89] [Francis and Nudd 90].

2.4.1 SIMD

SIMD processor arrays were seen for a long time as the only way to be able to achieve massive parallelism at affordable prices. Using fine grain processors it has been possible for many years to build machines which contain many thousands of processors, this has only recently become possible with coarse grain processors and that is still at great cost. Three machines using this model stand out amongst those developed in the seventies and early eighties, these are the Cellular Logic Image Processor (CLIP) [Duff 85] developed by University College London, the Distributed Array Processor (DAP) [Reddaway 79] [Hunt 89] developed by ICL and now marketed by Active Memory Technology and the Massively Parallel Processor (MPP) [Batcher 80-83] built by Goodyear for NASA. These machines are all remarkably similar, being mesh connected, bit serial SIMD machines. Two of these machines (CLIP and MPP) were designed specifically for image processing and the other has also been used extensively in this area [Reddaway 83a,b] [Oldfield 85].

The CLIP has been the centre of concerted research resulting in a number of different architectures with more complex ALUs, hexagonal connectivity and more recently an increase in word size. The Blitzen machine [Blevins 88] is a development of the MPP using higher technology allowing 128 processors on a chip each with 1Kbits of on-chip memory. Work is also taking place on developing a version of the DAP in state of the art technology and an increase in the granularity of the processors has also taken place with the addition of a multiplier to each processor.

The generic operations of iconic processing given in section 2.2.2 were that the operations tend to be uniform, local, massively parallel and fine grain. These requirements map very well onto the constraints of a general SIMD processor, therefore it is not very surprising to note that SIMD array architectures have been applied extensively to this problem.

A more recent development is the Connection Machine (CM) [Hillis 85] [TMC 89] of Thinking Machines Corporation of Cambridge, Massachusetts. This is an ambitious SIMD architecture that breaks away from the conventional, inspired by the concept of having 'intelligent memory' and each processor being able to communicate equally easily with any other processor regardless of where it is. Therefore in addition to a mesh the 16, 32 or 64K bit serial processors are connected with a hypercube. The CM was designed for artificial intelligence applications and therefore has proved to be fairly successful at symbolic processing [Tucker 88] [Little 87] [Thompson 88]. However the extra overhead of communication has meant that the iconic performance has been significantly degraded [Rosenfeld 87] [Weems 89]. More recent versions of the CM have also increased the granularity of the processors with the addition of a multiple bit arithmetic unit. One of the significant contributions of the CM is in the area of programming the machine. Traditional SIMD array architectures tend to be programmed with simple data-parallel extensions to existing languages. The CM provides a very supportive programming environment and languages such as Apply, *Lisp and C*.

2.4.2 Associative Architectures

As described above the concept of associative memories has been extended into the design of SIMD processor arrays. This has been developed in two somewhat different directions, one where the comparator remains single bit and the processor array is still very much like a conventional SIMD machine [Weems 89] [Nudd 88] and the other where the processor has a wider comparator and is more reminiscent of a real content addressable memory [Lea 91] [Duller 89]. Associativity in either form has much to offer as it combines the excellent match between the SIMD array and the requirements of iconic processing with an increased ability to extract non-iconic data from the array which is what so limits conventional SIMD array architectures.

The Content Addressable Array Parallel Processor (CAAPP) [Weems 84] from Hughes Aircraft and the University of Massachusetts is a mesh connected

SIMD array that has been enhanced for iconic and associative operations. The associativity is added in the form of a some/none and count response. Each processor performs the comparison bit serially then the some/none response is used to inform the controlling processor of the presence of matches and the count gives the number of processors matching. The array is augmented with a 'coterie' or gated connection network that allows processors to disconnect themselves from neighbouring processors, thus allowing data dependent regions to be set up.

SCAPE, WASP and GLiTCH from the Universities of Brunel and Bristol [Lea 91] [Duller 89] offer slightly different developments on the same principle. Each machine is based around having a processor for every word of content addressable memory to process any matches in place. Due to limitations on the number of pins and the belief that the way to maximise performance is to maximise the number of processors at the expense of the communications network, the processors are organised in a linear array. A barrel shifter is added to attempt to reduce the problems that would otherwise occur in communication with processors holding neighbouring pixels in the other dimensions. An interesting result of following strict associative principles is that some degree of fault tolerance is gained (as the processors are addressed by contents and not address) enabling ambitious wafer scale designs to be proposed.

2.4.3 MIMD Architectures

Many MIMD machines tend to be constructed from conventional microprocessors, but in recent years several processors have been developed specifically for use in such parallel architectures. A key feature of MIMD machines is how the communications is handled, there are several aspects to this. For distributed memory machines the first is whether the communication has to be performed by the processors themselves, for example by having to copy the message from memory to an output port. The second is whether there is any hardware support for the routing of messages to the destination node.

The Inmos Transputer [INMOS 85] offers several features that make it suitable for use in coarse grained parallel machines. These include hardware process scheduling and four on-chip communications engines which remove the load of transmitting and receiving data from the processor. Hardware message routing is to be performed by the next generation Transputer. Intel and Carnegie Mellon University have developed the iWARP processor [Borker 88] [Intel 89] that offers sufficiently similar facilities that it may be considered in the same way as a fast Transputer.

There are many examples of the use of more conventional microprocessors for constructing parallel machines. Three varied examples, using either the Intel 80386 or the Motorola 68020 processors are the BBN butterfly, the Sequent balance and the Intel iPSC/2. The BBN butterfly uses an omega network to allow each of the processors to access each other processor's memory - thus it has distributed shared memory. The Sequent machine has a single bank of shared memory. The contention is low for the machine is very coarse grained, the unit of parallelism being a UNIX process. The Intel iPSC/2 is a distributed memory machine with the processors being nodes on a hypercube.

These MIMD machines are generally marketed at much wider areas than the SIMD or associative architectures - for example the Sequent is aimed at general purpose multiuser UNIX systems. All of these machines have been benchmarked for image analysis tasks and have been shown to be inadequate for iconic processing. This may be seen from the DARPA image understanding benchmarks reported in [Rosenfeld 87] [Weems 89]. The BBN butterfly took from 3 to 18 seconds to perform an edge detection operation dependent upon the number of processors used. All three of these machines took several seconds to perform a connected component labelling operation. However their structure is well matched to the estimated requirements of symbolic processing.

2.4.4 Hybrid Architectures

Of the large scale parallel architectures that do not fit well into Flynn's organisation several stand out; these are Purdue Universities partitionable SIMD/MIMD machine (PASM) and the Hughes Aircraft and the University of Massachusetts Image Understanding Machine (IUA). PASM [Siegel 86a,b] consists of an array of conventional processors split into subarrays each with a control processor controlling their operation as several MIMD machines. This control processor also has the ability to issue actual machine instructions to its sub-processors enabling them act as SIMD processors. This allows the machine to be programmed as either an SIMD or MIMD machine. However this architecture does not offer any significant advances over MIMD architectures apart from this interesting approach to programming.

The Image Understanding Architecture (IUA) [Weems 84,89] [Shu 88] is a large scale heterogeneous hybrid machine that incorporates an SIMD array and several arrays of different MIMD processors. The SIMD array consists of the CAAP processors described earlier. The MIMD arrays are a numeric array of signal processing chips and a parallel Lisp machine. The arrays are organised so that they are topologically on top of each other, forming a pipeline machine with the image entering the SIMD array, then data from that passing through the MIMD arrays, which are intended to perform Lisp based AI processing.

2.5 Conclusions

It has been shown that a complete image analysis architecture must be able to process both iconic data and symbolic data and also to be able to convert between these forms of data. These three requirements make conflicting demands on an architecture and are therefore difficult to satisfy with a single architectural model. It has also been indicated above that SIMD architectures provide an efficient implementation for iconic processing and that MIMD machines fit the model given for symbolic processing. Therefore a hybrid

architecture, which consists of each of these different processing models, seems to fit the problem well. This however leaves a significant problem of translation between the two. Associative methods appear to satisfy this translational problem and hence a machine consisting of all of these architectural models appears to be worth investigation.

3

The Warwick Pyramid Machine

3.1 Introduction

This chapter describes the design and development of an architecture called the Warwick Pyramid Machine (WPM) [Nudd 88-91] that has been devised for image analysis applications. WPM has been developed and constructed by the VLSI group at the University of Warwick in the period 1987 to 1991. This architecture is intended to be a research platform from which to study the requirements and hardware implications of image analysis. The basic principle of WPM is to attempt to match the problem of image analysis as closely as possible.

The derivation of the architecture from the basic principles of image analysis is first described. This is followed by a more detailed description of the specific aspects of each of the elements of the architecture. Then some simulation and programming details are given followed by the actual implementation that has led to the partial construction of the machine.

3.2 Derivation of WPM

The previous chapter described the problem of image analysis at the level of generic operations and showed how it may be broken down into an iconic problem, a symbolic problem and an intermediate translational problem. It also showed that mesh connected bit serial SIMD processor arrays are well suited to the general problem of iconic processing and have been successfully used for this task for many years. Symbolic processing, however, is less well understood, and it was pointed out that a coarse grained MIMD array might be a reasonable initial compromise. Moreover, this appears to offer the most flexible option for an experimental research architecture. Therefore WPM consists of both an SIMD array and an MIMD array. The presence of these two arrays generates an additional problem as some mechanism to take data from the iconic array and pass it to the symbolic layer is required. This concept is illustrated in figure 3.1.

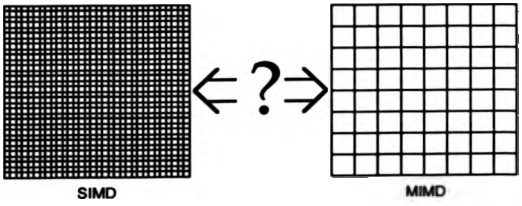


Figure 3.1 - Linking a fine grain SIMD array with a coarse grain MIMD

On a conventional SIMD array there exist several methods of passing data into and out of the array. The most obvious way of passing data into the array is the single controller that issues instructions to the whole array, which gives a limited broadcast facility. In addition there may be a facility such as an edge register, as for example on the DAP, that allows access to the edge of the array for passing data in and out. On conventional SIMD arrays there is usually little need to communicate non-iconic data, certainly not in large quantities, as there is little apart from a host processor to pass this data to. As the proposed architecture suggests that the symbolic processing does not take place on the SIMD array, substantial amounts of non-iconic data will need to be extracted. Therefore some additional mechanisms, that are not required on conventional SIMD arrays, need to be devised to perform this function. As the data to be extracted is not iconic then a method that simply increases the bandwidth for image data is not entirely suitable. The most promising design suggested for this task is one that is tightly coupled to the SIMD array and may extract information from it in an associative manner, as this allows data to be extracted that is related to features in the data such as size or number rather than large collections of pixel values.

The Image Understanding Architecture (IUA) mentioned in the previous chapter overcomes this problem by making use of the dual facts that the large (up to 512 by 512) SIMD array does not fit on a single chip and that there is access to memory that is off-chip. This allows each sub-array of processing elements to have their external memory dual ported with one of the processors that makes up the MIMD array and to allow that processor access to the local associative response before the global responses are accumulated at the controller. This provides a very interesting model to investigate further. For each group of SIMD PEs there is a direct communications path to a processor of the MIMD array in addition to the path to the global controller. This effectively joins the two arrays topologically one on top of the other instead of side by side as would be the case if the SIMD model were directly extended. This appears to offer significant advantages to the bandwidth of the communication between the two arrays.

However, there are several objections to this model. The first is the controller. It is proposed that the large SIMD array is to be controlled by a single controller (if this may be achieved in practice) and therefore the entire array is to be executing the same instruction at once. This, while perfectly adequate for normal iconic operations, may be very inefficient in those situations where the number of data elements that a particular operation applies to is very small. This is the case, for example, when the SIMD array is assisting in the extraction of features as in these situations the data is more sparse. Having only a single controller is a common cause of inefficiency in conventional SIMD machines. This inefficiency is somewhat overcome by smaller arrays such as the DAP where the image will normally be larger than the array and hence will be tiled onto the array, with each processor receiving several pixels to process. In this situation a local operation may not have to be applied to the complete image; instead it may be restricted to a subset of the tiles.

In addition to this, it does not appear that an architecture without local control may make good use of the associative capabilities of the machine. Associative responses such as 'some/none' or 'count' are often used to affect the control flow of the algorithm, therefore the value of having access to these locally is reduced as this local information may not affect the control flow of the patch of image from which it comes. For example, take the case where the machine were to iterate through a set of features performing some data dependent operation on each and extracting some value as a result. In this case the machine with a single controller would have to iterate through each feature in turn using the global associative responses to control the operation. This does not make use of the potentially increased vertical bandwidth as only a single value is passed upwards at a time across the whole image. If the control were available locally then each local patch could iterate over the number of features that happen to be within its patch and make full use of the distributed vertical communications.

An attractive option is therefore to follow the example of the IUA by splitting the SIMD array into patches and mapping each patch directly underneath an MIMD processor whilst allowing each patch of the SIMD array a local controller to which these associative responses could be passed and affect the local control flow. The justification for having a single controller is the simplicity of the hardware, however given a complex machine with coupling from each patch of the SIMD array to an MIMD processor the addition of extra controllers does not appear to add too much overhead. In addition to this it reduces the length of the control and response communications paths.

The second aspect of the IUA involving the coupling of the two arrays is the dual ported memory between the SIMD processors and the MIMD array. This does not fit in very well with the model of image analysis given in the previous chapter as it implies the extraction of iconic data to the MIMD array. This is because the only purpose this memory could possibly serve is to either extract or insert iconic data as there is no way of discriminating the features from the image data using this method.

Therefore from this discussion a much simpler and uncluttered picture of an architecture begins to form. This is shown in figure 3.2. The combination of an SIMD patch, its local controller and the MIMD processor is known as a cluster as shown in figure 3.3. These clusters are used as building blocks from which the complete machine is constructed.

Also shown on this diagram are the communications within each component layer. Clearly if the machine is to function effectively as an iconic processor then the SIMD array needs to be able to act as a single unit. This requires data communication at the SIMD level and synchronisation at the controller level. The previous chapter described many different interconnection networks that are appropriate for connecting the MIMD processors. At this point a two dimensional nearest neighbour network is used as this best fits the cluster model given. If later this is shown not to provide sufficient communications bandwidth then this can easily be

changed. At this stage little is defined, the architecture is conceptual yet derived directly from basic principles.

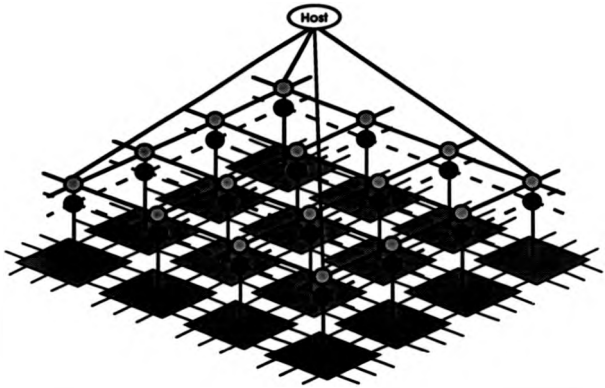


Figure 3.2 - The Concept of the Warwick Pyramid Machine

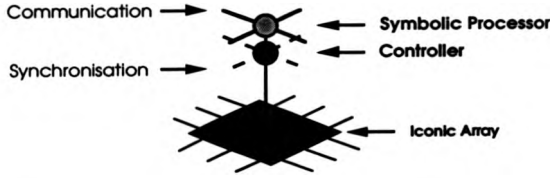


Figure 3.3 - A Cluster of The Warwick Pyramid Machine

Another key advantage of this strategy of breaking the machine into clusters is that it builds modularity into the architecture. The construction of a 512 by 512 SIMD array is an ambitious target and when the rest of the machine is also considered then the project may seem unrealistic or be too expensive to be affordable. Having broken the machine up into clusters it may be noted that this makes it possible to very easily choose the size of a machine, varying the size from a single cluster - which is effectively an associative SIMD array, to very large numbers of clusters forming a complete machine with one processor per pixel.

3.3 Architecture Description

This section describes, in more detail, the architecture that was designed based on the principles outlined in the previous section. This gives the full design of the architecture before describing the actual implementation in a later section. Note again that the design is based on assumptions about the problem that are not yet proven and it is one of the purposes of the machine to find out if these assumptions hold.

3.3.1 SIMD array

The SIMD array is to be used for the iconic processing of images. Therefore the first concern is the input of the images from the sensor. There are two ways of performing this. The first is to download the images from the top via the symbolic processors and the second is to load them in from the base. As the communication paths from the top are not designed for iconic data it would not be efficient or sensible to do it this way in a completed machine. Therefore the data should be loaded in from the base. This may be implemented using the processor to processor communications or by the use of an additional hardware mechanism.

The next decisions to be made concern the processor design. The first is the granularity of the processors. An initial examination of the processing

requirements suggests that a single bit processor provides a reasonable implementation, allowing a large number of processors to be used at low cost without too large a penalty on performance. This is significantly harder to justify if there are a large proportion of multiplications performed on the array as an n bit multiplication requires $O(n^2)$ steps for a single bit processor. If this is the case then some additional mechanisms may be needed, similar to those that have been added to the DAP and CM to improve their performance on scientific processing.

It is important for each processor to have a small amount of independence for certain operations, such as bit serial multiplications, to be performed. This is generally done by having an activity flag which controls whether a processor is to perform that operation or not. This is in turn often simplified so that the activity bit simply controls the writing of the data to memory as this is easier to implement. A design such as that given in [Nudd 88] which has internal memory uses the flag to mask the write strobe. Designs such as the DAP that have external memory cannot efficiently do this so they use a read/write/modify operation which reads the memory, modifies it conditionally upon the activity bit and then writes it back. This however will generally take an extra cycle as two memory references are made.

Due to the nature of the data a mesh connected array appears to be a good choice for the communications network, this also simplifies the layout of the processors both on the chip and the board. A seamless connection between the patches that are controlled by different controllers is desirable so that a processor may communicate in the same way with all of its neighbours regardless of where it is in the patch.

Some SIMD architectures have all of the memory apart from a few flags external to the SIMD chip [Reddaway 79], others mix it [Weems 84] and some fit it all on the chip [Blevins 88]. It would be desirable to get all of the memory that a PE can access on the chip as it would reduce the number of pins required and it would allow more efficient activity masking. However this has several disadvantages, firstly it restricts the amount of memory available

to each processing element to a small fixed amount and secondly it is likely that the packing densities available from doing this are much less than those which are manageable by memory manufacturers. Therefore it is highly likely that some external memory will be needed and hence there are two options. The first is to have it all, apart from a few flag bits, off chip, the second is to have an arrangement with some on chip memory used as a form of faster cache and have the bulk off chip. However this second arrangement increases the complexity of the design significantly which is not desirable for this simple bit serial processor. Of course it is possible that with extensive use it will be found that the amount of memory required is in fact low, in which case it might be possible to fit it all on chip, however at this stage the simplest and most flexible option is to use all external memory.

The size of the patch is a trade-off between the overhead of the controller and the overhead of the patch size on the associative response and efficiency of operation. If the patch is too small then there will be more controllers and the use of a fine grain SIMD processor will be harder to justify for efficiency reasons. If however the patches are large then the arguments given above about inefficiency of large arrays are valid. Therefore a compromise size is chosen at 16 by 16. This figure is also useful as it matches expectations of the number of processors that may be fitted on a chip in the near future.

3.3.2 Intermediate Level

The intermediate level has been specified to consist of a controller which controls a patch of SIMD processors, supplying the patch with instructions and data and extracting data back by the use of associative responses such as 'some/none' and 'count' which inform the controller of the existence of a response and the number of responders.

The initial design of the architecture [Howarth 87] [Nudd 88] had a processor which in addition to performing this control function acted as an intermediate processor taking the data from the SIMD array, performing some indeterminate actions on that and then passing a transformed set of

data to the symbolic processor. However it was found in practice that this processor had only a very small role to play except in the control of the SIMD array. It was found that this increase in complexity by having three levels that were programmed by the user made the machine far more complicated to model. In addition while it was performing this computation the SIMD patch that it controlled was idle. Therefore this processor was simplified to being a dedicated controller whose role in processing was restricted to simple translational operations such as decoding.

It was decided that the SIMD array was a valuable asset and that it should be kept active as much as possible, and hence that the controller should have a very small overhead. It was found that one of the most frequent operations was tight loops for providing multiple bit operations. For example, to supply an operation to the SIMD array that adds two n bit numbers together producing an $n+1$ bit result requires a tight loop around the generation of the three addresses (the two sources and the destination). Each of these addresses must be incremented on each iteration of the loop and the loop counter itself must be incremented and tested. A design decision made was to attempt to perform this operation such that the SIMD array was kept fully active throughout the tight loop. This decision directed much of the design of the controller.

In order to allow the SIMD patches to join together to act as a single array which is necessary for many of the iconic operations there needs to be some form of synchronisation between the controllers of adjacent patches. In addition it was decided that it would be necessary to allow adjacent patches to synchronise together without enforcing the synchronisation of the whole machine. This would allow patches which note that a region they are processing is shared between them to synchronise together to process it uniformly without requiring the whole machine to synchronise.

3.3.3 Symbolic Layer

The symbolic processing level has to interface with the cluster controllers in order to be able to accept data from them. In this case, where the communication is to be asynchronous, and it is important to minimise the overhead to the controller, a shared memory design is appropriate. As only a single processor from the MIMD array is attached to a single controller the design of shared memory is both simple and efficient.

In addition to this communication of data, it is envisaged that the symbolic layer provides instructions to the controllers to initiate processing based on results. The symbolic processor must be able to give these directions without degrading its performance significantly. Therefore these directions take the form of instructing the controller to perform an operation in a similar manner to a function call. The symbolic processor may then wait for the result or not, depending on what else it has to do. This may be done by the symbolic processor providing the controller with a start address of a routine stored in its microcode store and a return address for the result to be stored in.

The amount of computation and size of memory required by the symbolic layer is open to much debate. As described in the previous chapter there is not even consensus on the amount of data to be processed at this level. Therefore it is difficult to make a clear decision on the details of the hardware to be used to implement this level. A compromise has to be made which may then be adjusted at a later point.

Finally the symbolic layer itself is to consist of an array of distributed memory processors connected by a message passing network as this model fits well into the manner in which it is to be programmed and allows maximum flexibility for expanding the machine and varying the number of clusters.

3.4 Simulation and Programming

Having specified the architecture at this level it is possible to begin to simulate it. This was done using an interpreter of a language called Cluster Programming Language (CPL) [Howarth and Francis 88]. This basic language matches the structure of the SIMD patch and controller and allows programs to be interpreted at the level of arithmetic operations. CPL gives simple control structures and variable allocation. This language allowed simple porting of algorithms to the architectural model and enabled some of the initial design ideas to be rejected and the rest of the model to be supported. However this did not give any detailed knowledge of the execution of the algorithms as there was no concept of how long an operation was taking.

Having performed this simulation the next stage was to attempt an implementation of a cluster and from that a number of clusters in order to test all of the ideas presented in practice. Before this a further stage of simulation was necessary using a simulator of a cluster that operated at a much lower level than the interpreter of CPL by actually simulating the hardware components and taking in the same raw microcode as the actual machine [Francis 89]. This simulator, as it accepts the output of the WPM assembler, executes the identical programs to the actual hardware. Therefore programs could be developed for the machine well in advance of its construction. In addition to this certain hardware modifications could be tested to analyse their value. For example, it was noted that a 'count' response such as proposed initially [Nudd 88] that took 125 cycles to operate was sufficiently slow that it dramatically affected the way in which the response was used and in fact it meant that the 'count' was not of any significant value. This led to the design of a much faster responder.

The early method of programming the machine was in microcode for the bottom two levels of the machine and Occam or Parallel C for the symbolic level. This, while producing very efficient code, made the machine very difficult to program. Therefore an integrated method of programming the machine was developed that involved a single object orientated model that

encompassed the whole machine [Vaudin 91]. This led to a significant easing of the problem of programming and enabled the good fit of the architectural model to the task to be fully exploited.

3.5 Implementation

3.5.1 Introduction

The Warwick Pyramid Machine has been implemented as far as possible using off the shelf components as the idea behind it is to develop understanding of parallel image analysis architectures and not to spend many years developing VLSI components which may then be made obsolete by the knowledge obtained from the use of the machine. Therefore the architecture does not on the whole make use of state of the art components and hence the performance obtained may in fact be somewhat less than could be gained from the use of the technology used in other image analysis architectures.

3.5.2 SIMD Array

The iconic array is currently implemented using Distributed Array Processor (DAP) chips manufactured by Active Memory Technology Ltd (AMT) as used in the DAP SIMD array machine [Hunt 89]. Each of these chips consist of a square array of 8 by 8 very simple processing elements (PE) connected on a 4-way nearest neighbour mesh. Therefore four of these chips are required to make a 16 by 16 SIMD patch. The DAP chips were chosen for the implementation as they match the requirements reasonably well and they allow the architecture to be constructed without the potentially very long process of designing custom VLSI circuitry.

A DAP PE [AMT 88] consists of a 3-input single-bit adder with two multiplexers to select input from one of 3 on chip flag registers, external memory, a data bus internal to the cluster or the output of a neighbouring PE. This is shown in figure 3.4. One of the flag registers can act as an activity mask bit, restricting a subset of operations to those PEs satisfying a certain

condition. Bidirectional row and column lines run between the PEs in the array to facilitate the reading and writing of data from the array. Each PE can perform operations which require only a single memory reference in a single cycle. Operations which require two memory references take two cycles, this includes those that use the activity mask bit. All of the DAP's memory is external to the chip. The processing elements are currently clocked at 10MHz. A mechanism is supplied that enhances loading the image data from a sensor by the use of an additional register that can be clocked independently of the processor at twice the rate. This allows data to be loaded serially into each array patch without affecting the processing taking place on the array.

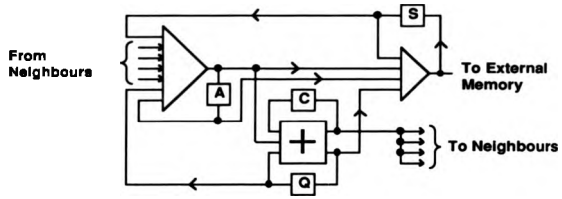


Figure 3.4 - Simplified Schematic of a DAP Processing Element

The DAP chips provide a good match to the design suggested earlier, however the DAP does not possess any of the associative capabilities that are required. The row and column lines that run through the array may be used to provide a some/none response as the result obtained at the edge register is the logical AND of the bits in each row/column. The some/none is a logical OR which way be achieved with the AND operation by inverting the data and the result. However the count may not be performed so easily. It is a requirement of the specification that the count response is obtained within a small number of cycles in order for it to be effective. Therefore some additional hardware was necessary for this. Several methods were devised [Russell 90] but the only

one that met the requirement involved the construction of a gate array [Walton 89].

This gate array is attached to each of the 64 memory lines coming from the DAP chip. Every time the processor writes to its external memory, the gate array counts the number of processors that were asserting their output. As four DAP chips make up a cluster, there have to be four gate arrays. These can be daisy-chained together to give a count and some/none response for the whole cluster. This result is available to the controller two cycles after the processing elements output the value. This is sufficiently fast that it presents little overhead to the controller.

3.5.3 Controller

The controller is composed of two major functional units, a microcode sequencer and a scalar ALU. Both of these have been implemented using parts available from Advanced Micro Devices. In addition to these parts the controller has a microcode store, interface to the iconic array and interface to the symbolic processor. It is designed to maximise the rate at which instructions can be issued to the SIMD array and thus to maximise its utilisation. A schematic of the complete cluster showing the details of the cluster controller is given in figure 3.5.

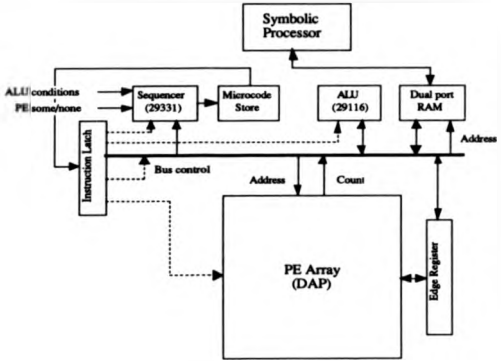


Figure 3.5 - Block diagram of a single WPM cluster

The instruction stream from the microcode store has four components: instructions for the sequencer, ALU, iconic array, and control of the data bus. The three functional units operate in parallel to allow address calculation and microcode loops to function with no cost to the micro-instruction rate. A horizontal microcode of 61-bits wide has been used to minimise instruction decode time. The microcode word is shown in figure 3.6.

Bit Position	0 5	6 21	22 34	35 44	45 60
Use	Sequencer instruction	ALU instruction	SIMD instruction	Data bus control	Immediate Operand

Figure 3.6 - Microcode Word Format

Dynamic addressing of the PEs memory is provided by the controller. In a typical low level operation two source images may be combined and the result written out to a third. Three memory pointers are therefore required to iterate over the depth of the image, one for each source and one for the result. The controller can generate addresses in parallel with the array operation, so that no extra cycles are required for the address calculations. This is particularly useful for implementing multiple bit instructions.

This may be illustrated with the example given earlier of a multiple bit add. This requires three memory references which takes the DAP three cycles, therefore the minimum number of cycles that the loop may take is also three cycles. The design of the controller makes it possible to increment and output the three addresses in parallel with their use by the array. Also the sequencer may operate in parallel with the other functional units and the result is that there is no overhead in the loop at all and simply a few cycles at the start to set up the addresses.

One method by which data is passed into and out of the iconic layer is the edge register. This may pass a 16-bit value to all, or to a selected horizontal or vertical line of the PEs. The edge register is connected to the horizontal and vertical buses that run through the iconic array. It is possible to select either the horizontal or vertical bus or to mask individual lines in the bus. This makes it useful for loading certain types of data into the array and extracting data from specific processor. However, while this is satisfactory for loading small amounts of data, it is not a particularly efficient method of loading an image. As described earlier the DAP processing elements provide a fast input path that can continue independently of the processing elements at a faster clock rate which is ideal for the input of an image from a sensor.

The clusters need the capability to synchronise with each other in order to allow communication at the SIMD level. This is achieved by the use of an open collector synchronisation channel that passes through the clusters. Several of these channels may be used to allow several different sets of

clusters to synchronise. In addition to this, communications between the SIMD patches are set up so that neighbouring clusters may communicate at the SIMD level. The communication links between neighbouring clusters have two states. The first is cluster mode where the edges of the patches wrap around to form individual tori. This mode makes it significantly easier to simulate a larger machine using multiple tiles of image on each cluster. The second state is where the neighbouring clusters are connected to each other, which is known as array mode.

3.5.4 Symbolic Array

The processor used for numeric and symbolic processing is the Inmos Transputer [INMOS 85]. These devices are suitable for tasks involving both computation and communication. They were originally chosen because of their internal support for multiple process execution and the ease with which they can be connected to form MIMD networks. A T800 Transputer contains an internal floating point unit, 4K bytes memory cache and 4 DMA engines enabling data to be transferred to a neighbouring processor at the rate of 20 Mbits per second per channel. The Transputer passes instructions and receives data through RAM that is dual-ported with the controller.

The ratio of symbolic processors to iconic arrays can be altered. If the application requires more iconic than symbolic processing, then a single Transputer could be associated with four controllers and their respective arrays. If the converse is true, four Transputers could be associated with one controller and array. Flexibility and modularity have been designed into the cluster to allow for all of these permutations.

3.6 Conclusions

This chapter has described the Warwick Pyramid Machine which is an architecture aimed at addressing the requirements of image analysis. The architecture consists of clusters. A number of clusters are connected to each other at each level to form a machine which may be a size appropriate for the

application. A cluster consists of a 16 by 16 patch of fine grain SIMD processors currently implemented using DAP PEs, a controller for the patch and a single symbolic processor which has been implemented with an Inmos Transputer.

This architecture has been simulated in depth and a test system has been constructed from off the shelf components. The ensuing chapters describe the mapping of a number of algorithms onto this machine.

4

Image Processing on WPM

4.1 Introduction

The initial stages in most image analysis tasks are generally concerned with iconic processing, therefore a computer architecture designed for image analysis must be able to perform this type of processing in addition to the processing of features and symbols. Iconic processing generally involves fairly simple operations applied to very large amounts of data and while all processors are capable of performing the individual operations adequately very few architectures are suitable for either the flow of data or the number of operations that need to be applied. Studies have shown [WSTL 90] [Francis et al 91] that moderate sized arrays of state of the art microprocessors are not

adequate for near real-time performance. Often this type of processing is either ignored or off-loaded to application specific hardware. While dedicated hardware is adequate for many purposes it offers little programmability. Therefore while this may be used in some specific fixed circumstances it is not suitable in situations where the problem is less well defined.

In addition to describing the way in which many commonly used standard algorithms may be implemented on the architecture, several novel algorithms are also described in this chapter. These are used to construct a processing flow that may be used for the detection and segmentation of a class of objects in a certain type of image. This processing flow has proved to be successful in processing an application based on infrared imagery in the inter-frame period [WSTL 90]. The study of a complete low level processing flow provides a check that nothing of central importance is overlooked by the study of individual algorithms. This includes the form taken by the data input to and output from an algorithm. It would, for example, be possible to devise efficient new algorithms or implementations of algorithms that had very different requirements for input or output data than the neighbouring algorithms supplying or accepting data from it. It may be that the disadvantage of remapping the data outweighs any advantages gained by the improvement in the new algorithm or implementation. Therefore great care has to be taken when inspecting algorithms to check that the data patterns are not out of context with what it is reasonable to expect.

4.2 Iconic Tasks

4.2.1 Introduction

As described above the iconic processing is generally performed on the bottom layer of the WPM which can act as a conventional SIMD array. This is generally because the operations involve images both as input and output, and that iconic operations tend to involve local communications and require little local control.

4.2.2 Sobel Edge Detection

One of the most common operations in image processing is the detection of edges. It is believed for many applications that the changes in contrast or edges of objects provide sufficient information to initiate the analysis of the image, therefore significant effort has been devoted to the edge detection problem. The common feature of all of the techniques devised for edge detection is that they process the image locally and uniformly. This means that the SIMD array provides an ideal architecture for this application.

The basis of most edge detectors involves a filter that computes the difference between pixel values - clearly if pixels that are near each other have a large difference in their values then there is a likelihood that there exists an edge between the pixels. However a very simplistic difference filter based on differencing the values of neighbouring pixels tends to be very prone to high frequency noise, and therefore a common approach taken by many methods is to smooth the image before applying the difference filter thus reducing the high frequency element of the image. This has to be performed very carefully as the edge itself is a feature containing high frequencies and care has to be taken to avoid removing the edge which is the subject of the search.

Probably the most commonly used edge detector is the Sobel. This is a very simple edge detector that involves only a 3 by 3 neighbourhood, yet the results produced are adequate for most applications. Filters with larger mask areas can produce better results for noisy imagery with large features as the evidence may be accumulated over a large area making them less prone to high frequency noise, but they tend to miss small objects even in clean imagery due to the smoothing effect of the large filter. The conventional Sobel edge detector is generally described by the set of masks shown in figure 4.1. As the area of the mask is small it may detect small features accurately but is not very resistant to noise.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Figure 4.1 - Sobel masks

The two masks shown in figure 4.1 are both applied to the image, one of the masks giving the horizontal component of the edge and the other giving the vertical component. In order to understand the behaviour of these masks they may be broken down into the set of 2 by 2 masks shown in figure 4.2. The process breaks down into a convolution with the simple 2 by 2 smoothing mask shown in figure 4.2(a) followed by a convolution with both of the difference filters shown in 4.2(b) and 4.2(c). To obtain the horizontal component from these the image is first convolved with the mask in 4.2(a) and then the result of that is convolved with the mask in 4.2(b), the vertical component is obtained by convolving the result of 4.2(a) with 4.2(c). In addition to clarifying the action of these masks, considering the edge detector in this way can lead to a more efficient implementation.

1	1	
1	1	

(a)

1	1	
-1	-1	

(b)

-1	1	
-1	1	

(c)

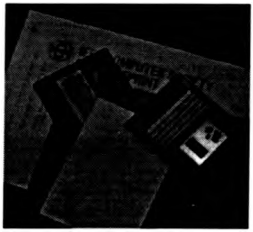
Figure 4.2 - The division of the Sobel Masks

Figure 4.3 shows the Pyramid C++ code that implements the computation of this operator on WPM. The Pyramid C++ language is described in [Vaudin 91]. The positions 1, 2 and 3 marked in the code show where the three masks given in figure 4.2 are computed. Position 4 shows where the magnitude is computed using the Manhattan magnitude metric.

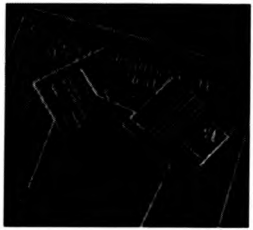
```
EdgeImage
sobel(ShortImage& image)
{
    ShortImage  smoothed,tmp,horiz,vert;
    EdgeImage   edge;

    smoothed = image + image.east() ;
    smoothed = smoothed + smoothed.south() ;    // 1

    tmp = smoothed + smoothed.west() ;
    horiz = tmp - tmp.north() ;                 // 2

    tmp = smoothed + smoothed.north() ;
    vert = tmp - tmp.west() ;                   // 3

    edge.mag = vert.abs() + horiz.abs() ;       // 4

    return(edge);
}
```

Figure 4.3 - Pyramid C++ code for Sobel

As shown in figure 4.3 the convolution with the smoothing mask is separable and on a mesh connected SIMD array it simply consists of two additions of neighbouring pixel values. If the depth in bits of the image is b then on WPM this takes $4b+1$ cycles for the first addition and $4(b+1)+1$ cycles for the second addition - taking into consideration the increase in the number of bits following the first addition. This gives a total of 70 cycles for conventional 8 bit data. To convolve the result of the smoothing with the first difference filter takes one $(b+2)$ bit add with a neighbouring pixel and one $(b+3)$ bit subtract from another neighbour. This takes 87 cycles on WPM. Convolving with the other difference filter takes exactly the same time. Therefore to compute the horizontal and vertical components of the Sobel takes 70+87+87 cycles = 24.4 μs. This very rapid performance is achieved because of the nature of the operation and the fact that no multiplications are required. This operation also illustrates the way in which simple algorithms may be optimised for use on a bit serial SIMD array. The results of performing this operation are shown in figure 4.4.



(a) Original image



(b) Vertical edge image

(c) Horizontal edge image

(d) Magnitude

Figure 4.4 - Results of Sobel Edge Detection

Having obtained the two components, generally a magnitude and sometimes the orientation of the edge at each pixel is required. These may be obtained in several ways usually depending upon the requirements of the following algorithm. The most common way of performing this operation is to compute the magnitude using root sum square of the two components and the orientation using an arc tangent. Both of these operations are well suited to an SIMD array in that the same operation is applied across the whole image and no communication is required. However neither operation is well suited

to bit serial processing due to the nature of multiplication, square roots and arc-tangents. This is often overcome by computing the magnitude using the Manhattan distance metric rather than the Euclidean and by quantising the orientation component and computing it using a number of comparisons. Quantising the orientation is often an acceptable approximation as frequently the process following does not make use of the complete range of orientations. It is also the case that any orientation computed over a three by three neighbourhood is unlikely to be of a high enough accuracy to justify a high resolution orientation being computed using an arc-tangent. Pyramid C++ code to perform the orientation computation to a quantisation of 8 different angles is given in figure 4.5. This code makes use of the '*where(bit_image)*' function that restricts the operation to those processors that have their pixel of the binary image *bit_image* set.

```
// calc direction
edge.dir = ByteImage(1) ;
where (horiz.abs() > (vert.abs() << 1))
 edge.dir = ByteImage(0) ;
where (vert.abs() > (horiz.abs() << 1))
 edge.dir = ByteImage(2) ;
where (horiz < ShortImage(0))
 edge.dir = ByteImage(4) - edge.dir ;
where (vert < ShortImage(0))
 edge.dir = ByteImage(8) - edge.dir ;
edge.dir &= ByteImage(0x07) ;
```

Figure 4.5 - Pyramid C++ code to compute quantised orientation

4.2.3 Convolution

Convolution has very similar requirements to the Sobel edge detector. The operation is also a function of the pixel value and the values within the local neighbourhood, however, unlike the Sobel, in the general case it requires multiplications to be performed. The action of a 5x5 convolution is :-

$$R_{x,y} = \sum_{j=-2}^{2} \sum_{i=-2}^{2} I_{x+j,y+i} \; M_{j,i}$$

Where $R_{x,y}$ is pixel (x,y) of the the resultant image, $I_{x,y}$ is pixel (x,y) of the original image and $M_{j,i}$ is the value (j,i) of the array of filter coefficients.

The multiplications that are within this inner loop somewhat impair the effectiveness of a bit serial architecture on this problem, particularly if the accuracy of the coefficients is high. The multiplications required are uniform across the whole image as the filter coefficient is constant. This enables the controller to adjust the instruction stream for each constant, incorporating the bit pattern of the coefficient into the instruction, meaning that the multiplications required are significantly faster than for a general multiplication.

The operation is implemented by shifting the image so that each pixel within the neighbourhood of the same size as the mask passes over each processor once. In order that there are no unnecessary time-wasting shifts the data is shifted in a spiral fashion as shown in figure 4.6. As each new data element passes over a given processor it is multiplied by the filter mask value and accumulated with the current total.

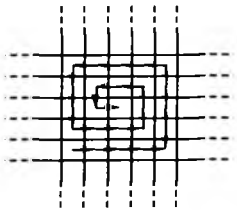
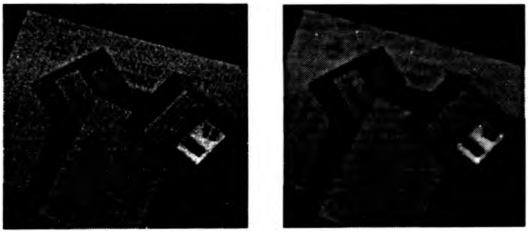


Figure 4.6 - Data communications pattern for a 5 x 5 convolution operation.

On WPM a non separable three by three convolution takes approximately 110 μs depending somewhat upon the coefficients in the mask; a separable seven by seven filter takes nearly 200 μs.

4.2.4 Median Filtering

Median filtering takes the median value of the pixels in the neighbourhood of each pixel as the new value for that pixel. This operation is generally used to remove high frequency 'spot' noise while retaining edges, as unlike a smoothing convolution which is also often used to remove high frequency noise, the median filter does not smooth larger areas. An example is that an ideal step edge is left completely intact by a median filter whereas a smoothing filter will turn the step into a ramp the steepness of which depends upon the size of the convolution filter used. An example of the use of a median filter is shown in figure 4.7 where a noisy image (figure 4.7(a)) is filtered using a 5 by 5 separable median filter to produce the result shown in figure 4.7(b).



(a) Noisy Image *(b) Result of median filter*

Figure 4.7 - Example of a Median Filter used to remove high frequency noise

However, median filtering on conventional SIMD arrays is not as simple an operation as convolution despite it being an iconic pixel operation with data accesses generally from the same neighbourhood area as a convolution. This

is because, unlike convolution, the control flow of the median filter has some data dependency. In order to compute a median value a sort operation has to be performed, and as each processor sorts a set of numbers then many comparisons take place. At each comparison there are two alternative instruction paths to follow, first where one number is smaller and the second where the other number is smaller. This does not well match a single instruction stream computer. When sorting, usually each path will follow an identical instruction stream but will apply the operations to different memory locations, on SIMD processor arrays without the ability to form local addresses each of these paths will need to be executed serially by each processor. Attempts can be made to minimise this but it does create a potentially large overhead. An additional problem is the amount of memory consumed by the operation. A convolution uses very little local memory, however most implementations of a median filter involve copying the entire neighbourhood into each processor - this may consume a large amount of memory, particularly for large filters.

Copying all of the neighbouring pixel values into each PE works in parallel across the array and takes $O(bm)$ steps, where b is the number of bits in the image being filtered and m is the area of the filter. The next step involves sorting the elements within each PE. If a conventional sorting operation is used then a selection sort is a reasonable choice as it reduces the number of times data needs to be moved between memory locations, which as outlined above is the operation that slows down an SIMD array. The selection sort works by finding the smallest element followed by the next smallest etc. For a median filter only $m/2+1$ iterations need to be applied as the upper set of values do not need to be sorted. Careful use of flags minimises the number of data dependent copies that need to be applied, however this algorithm takes $O(m^2 \log m)$ time to execute.

Using this algorithm the exact time the complete median filter takes is $(3mb+3) + (\text{int}(m/2)+1)(m + m(2+3b +3b +3\log m+9) + 17m)$ cycles on WPM. For a 3 by 3 filter with m therefore equal to 9 applied to an 8 bit image this

equals 4224 cycles or 422 μs. For a 7 by 7 filter with m equal to 49 and b equal to 8 this gives 117554 cycles or 11.8 ms.

An alternative method is suggested in Danielsson [Danielsson 81]. This algorithm uses a partial lexicographic sort to find the median value without sorting all of the elements. As before each element in the neighbourhood is copied into the processor. Then the number of elements with its top bit clear is compared with $m/2 + 1$, if it is less than this then the top bit of the result is 1 otherwise the top bit is 0. This process continues in a similar manner for each bit of the operands therefore taking $O(mb\log m)$ time. Therefore if b is less than m then this operation is quicker than the selection sort method - even in the case where b is similar to m then the constant is lower as the data dependency of the operation is reduced somewhat. Anyway in most cases generally b will be lower than m.

Using this algorithm the median takes $(3mb+3)+2\log m+m(5\log m) + (b-1)(2+\log m+1+3m+m(6\log m)+2(4\log m))+3\log m)$ cycles. For m equalling 9 and b equalling 8, $\log(m)$ is 4 giving a total of 2465 cycles or 246.5 μs. For m equal to 49 and b equal to 8, $\log(m)$ is 6 and the time taken is 1.66 ms. Both of these times provide a significant improvement over the selection sort method, in addition this algorithm gives a clear speed advantage for larger medians. Table 4.1 summarises this performance.

	Conventional	Danielsson
3x3	422μs	247μs
7x7	11.8 ms	1.66 ms

Table 4.1 - Summary of performance of Conventional and Danielsson median implementation on WPM

4.2.5 Connected Component Labelling

Connected component labelling consists of associating a single label with each connected set of pixels in an image. Two pixels are connected if there is a complete path between them consisting of pixels with the same label. An 8-connected component is one where the pixels may be connected by diagonal links whereas a 4-connected component has to be connected by links in the NESW directions. The algorithms discussed here are for the more general 8-connected case. The maximum length path between any two pixels in the component is known as the diameter of the component. Connected component labelling may be performed on binary images in which case all connected regions of set pixels are given the same label. Alternately images with multiple bits of accuracy may be labelled in which case some condition must be devised for set membership. This may be that the pixels have the same grey level or it may be that they are within a given range of grey levels.

The most obvious implementation of the algorithm on a mesh connected computer produces a reasonable average case running time. However in the worst case using this method, connected component labelling can potentially run no faster on a mesh connected parallel machine than on a sequential uni-processor machine of the same cycle time. In this implementation each processor is initially given a unique label, this is generally achieved by using its address in the array as the label. Consider initially a binary image, for a grey scale image simply involves extensions to the set membership criteria. Mark each processor that contains a set pixel as active. Each active processor then passes its label to its 8 neighbours and receives in return their labels. It then keeps the smaller of the new labels passed to it and its currently stored label. This process is then repeated until the smallest token in each region has propagated throughout the whole region. At that stage each connected region has a unique label.

In the case of a multiple bit image the processors have to compare their pixel value with those of their neighbours. This can be achieved by passing their

pixel value with the label, the neighbour can then perform the comparison before deciding which label to keep. The process then proceeds in a similar manner to that outlined above.

The propagation of the labels requires in the worst case a number of iterations equal to the diameter of the blob. With convex 'blob' like regions this diameter is $O(n)$ giving a total time which is $O(n \log n)$ - as the label is of size $O(\log n)$ and takes $O(\log n)$ time to propagate and compare per iteration However, in the worst case region - that of a line winding down the image in a snake-like shape with the minimum label point at one of the ends (figure 4.8) - then the operation takes $O(n^2)$ iterations and hence a time $O(n^2 \log n)$.

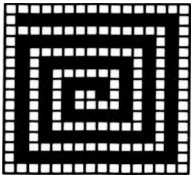


Figure 4.8 - Worst case image for Labelling Operation

An important addition to the algorithm is to ensure that it does not have to run to the maximum pathological extent for every average case. In practice it is very unlikely that any line will be longer than $O(n)$ so a way of determining when the array is fully labelled needs to be used in order to terminate the algorithm as soon as possible. This detection may be performed by using either the count or some/none facility which can check very quickly whether any relabelling has taken place within that iteration of the algorithm, if not it terminates. This makes the algorithm always take $O(d)$ iterations, where d is the diameter, which is as it should be. This algorithm takes 17 ms for the case where the diameter is 256 and several seconds for the worst case.

While the above algorithm is both simple and intuitive and performs well in the average case, it exhibits very poor performance in the worst case. Any

image analysis system that is expected to perform in a real world environment cannot function well with this amount of uncertainty in the performance. Cypher et al in [Cypher 90] describe an algorithm that represents a breakthrough as it is capable of labelling any image in $O(n)$ iterations. This is achieved by the use of a shrinking algorithm [Levialdi 72] that iteratively shrinks each region to a single pixel located in one of the corners of the bounding box of the region. Each iteration is stored creating a three dimensional image which contains the history of the shrinking. The time this process takes is $O(m)$ where m is the sum of the width and the height of the bounding box of the region. Clearly for an n by n image the maximum value m may take is $2n$. Having performed the shrinking operation the final remaining pixel in each region is given a label related to the position of the pixel and the iteration number. Once the some/none response has reported that all regions have been shrunk out of existence then the algorithm backtracks through the three dimensional output propagating the labels back through the regions until the regions are back, completely restored, uniquely labelled. This return operation takes the same number of iterations as the shrinking operation. On the WPM this algorithm takes 7 ms for an image with largest diameter region of 256 and 14 ms in the worst case. It may be seen that the difference between the average case and the worst case is much less than the other algorithm. Cypher's algorithm provides better performance and a more consistent time at the cost of a largely increased memory consumption.

A method analogous to that used on conventional sequential architectures is to build equivalence tables and resolve them and then relabel the image. Each processor notes whether it and its neighbour should be connected, if they should the equivalence of their two labels is passed to their controller. The symbolic processors can then maintain a global equivalence table to store the equivalence of the labels and then determine a minimal set from that. However, this would require the transmission of a very large amount of information to the controller and the symbolic processor to store and work on

a very large equivalence table. Several enhancements can be made to this to make it more feasible.

The first of these is to compute the connected components over each cluster independently of all of the other clusters. As noted above, the time taken to label an image is proportional to the size of the image multiplied by the log of that size and therefore if connected component labelling is computed over a cluster the time for this operation is much reduced from that required for a complete image. However many regions may have been split by cluster boundaries so these equivalences need to be noted and this much reduced set of equivalences is passed to the symbolic processors to be matched.

The equivalences are extracted by each cluster comparing two of its edges with the corresponding edges on the neighbouring clusters. Having extracted the equivalences they are resolved in a very simple $O(n)$ operation on the symbolic processor, the only difficulty being that the operation is serial and has to take place in a single processor and hence the equivalences need to be communicated to that processor. Relabelling takes place by the use of a global broadcast and compare where each SIMD processor compares its label with the broadcast label and conditionally alters it to the new label.

The second method is to compute connected components along each row independently of the other rows. Then the labels from each row are matched with neighbouring rows and the equivalences are again passed to the symbolic processor. The maximum diameter of the first stage is the length of the row or n. However Cypher's method may not be used to compute these much simpler connected regions and therefore the somewhat slower algorithm has to be used. The advantage is obtained from the fact that the labels are propagated in a single dimension which makes each iteration a simpler step than propagating the label horizontally, vertically and in both diagonals.

The performance of each of these methods on two test images is shown in table 4.2 and table 4.3. The first table shows the results on an artificial image,

the second table on a real image. The propagation algorithm is the algorithm described initially. The shrinking algorithm is that given by Cypher. The cluster plus propagation is the propagation algorithm working within each cluster independently followed by a resolution of equivalences. The cluster plus shrinking is the Cypher algorithm working on each cluster and finally the row algorithm is the algorithm where regions are connected along the rows first and then equivalences resolved.

If these tables are compared then the only advantage of the shrinking algorithm over the propagation algorithm is the time each iteration takes to perform. This is due to the simpler nature of the propagation of the labels. This misses the significant advantage of reasonably bounded execution times because in practice few if any images lead to such pathological regions. As shown a small performance gain is achieved by making use of independent clusters with both the cluster plus propagation and the cluster plus shrinking algorithms. The row algorithm does achieve a low count of cycles per iteration but this does not overcome the increase in iterations caused by it. The reason that these methods do not achieve even greater performance gains is the time taken to resolve the equivalences at the symbolic level.

	Propagation	Shrinking	Cluster + Propagation	Cluster + Shrinking	Row
Max # iterations	230	232	18	27	116
Cycles per iteration	672	285	672	285	168
Time 1 (μs)	15456	6612	1210	770	1950
# Equivalences	0	0	167	167	295
Time 2 (μs)	0	0	1789	1789	2173
Total Time (ms)	15.5 ms	6.6 ms	3.0 ms	2.6 ms	4.1 ms

Table 4.2 - Comparison of Connected Component Algorithms - Artificial Image

	Propagation	Shrinking	Cluster + Propagation	Cluster + Shrinking	Row
Max # iterations	106	169	31	31	65
Cycles per iteration	672	285	672	285	168
Time 1 (μs)	7123	4817	2083	884	1092
# Equivalences	0	0	269	269	680
Time 2 (μs)	0	0	2095	2095	3328
Total Time (ms)	7.1 ms	4.8 ms	4.2 ms	3.0 ms	4.4 ms

Table 4.3 - Comparison of Connected Component Algorithms - Natural Image

The above description has been of algorithms suitable for implementation on either WPM or conventional SIMD mesh connected architectures, however Shu and Nash [Shu 90] show how having additional hardware in the form of a gated connection network enables very fast computation of connected component labels. The gated connection network allows processors to disconnect themselves from their neighbours. In their algorithm the processors that form a part of the region disconnect themselves from the processors that do not. Their architecture [Shu 88] [Weems 89] then has the hardware capability to perform a some/none operation over each of these independently connected regions - with each processor within the region obtaining the results - within a small number of cycles. This allows a minimum to be computed very simply within each region independently of the shape and size of the region. Each region can then be labelled with this minimum processor address within $O(\log n)$ time regardless of the make-up of the image.

4.2.6 Edge Thinning

The result of many edge detection operations is an edge that is several pixels wide. This is often not desirable for many applications. For example, in the case of edge following it makes the edge somewhat more difficult to follow or in the case of the use of the Hough transform to extract straight lines it introduces an uncertainty in the angle of the line. Many solutions have been proposed to thin the output from the edge detector. The majority of these involve a binary mask which is iteratively applied to the edge pixels to erode them leaving a central 'skeleton' set. These are all somewhat heuristic and tend to be erratic in their performance, an implementation of the operator given by Elliman and Mahmood [Elliman 88] takes approximately 63μs on WPM.

An alternative which applies well to edge detected images that have not been thresholded is based on the edge orientation information. The algorithm

simply compares each pixel with the neighbouring pixels perpendicular to the edge direction. If the pixel is smaller than these neighbouring pixels then it is suppressed. This is a very simple, effective and quick operation given a quantised edge orientation image taking 54 μs.

4.2.7 Thresholding and Edge Following

Simple thresholding is a straightforward operation that maps very well onto SIMD arrays. Each processor simply compares the pixel value it stores with the broadcast constant and if it is greater than or equal to it, it sets the output bit else it clears the output bit. However thresholding an image, especially an edge image, with a single value is rarely of any great use. An enhanced thresholding algorithm for better thresholding of edge images known as hysteresis thresholding is given by Canny [Canny 86].

Hysteresis thresholding allows two threshold values to be given, a low threshold and a higher threshold. Initially the edge image is thresholded using the upper threshold, the idea being that at least one pixel on every significant edge will have a value above this. But it is recognised that this threshold is unlikely to capture the whole edge. Therefore the image is thresholded using the lower threshold and this is also stored; ideally this second threshold would be set at a level to include all of the pixels in an edge. Having performed these thresholds, the pixels in the first thresholded image are considered seed pixels for an edge following operation that takes place using the second thresholded image. Therefore, the final result is that any edges above the lower threshold are included that also have a pixel above the upper threshold somewhere along their length. This allows a low threshold to be set to extract weak parts of edges yet also allows completely insignificant edges to be ignored.

This edge following may take place in a very similar manner to the connected component labelling operation described above except it is somewhat simpler as the label does not need to be propagated. The operation begins with each

seed pixel and works outwards from it within each region marked by the second threshold image. This is very simple to perform by using the first algorithm described in the connected component section (4.2.5). However, in this case the data is of the form that will result in very long chains as this is exactly what the hysteresis thresholding algorithm is attempting to achieve. This, as noted before, results in very poor performance of the propagation algorithm. Wysocki [Wysocki 89] when mapping the Canny edge detector [Canny 86] onto the DAP used this method and limited the number of iterations to 5. This is not adequate as very few edges will have this small a distance between seed points unless the upper threshold has been set too low. Hysteresis thresholding, as it results in this type of data, is an ideal application for the Cypher algorithm [Cypher 90] described in section 4.2.5 as it may perform this process in $O(n)$ time. However the Cypher algorithm requires some adjustment in order to be suitable for data resulting from these two threshold images.

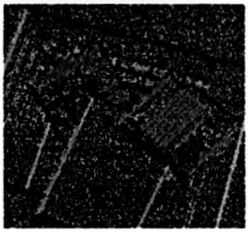
At any stage of the shrinking process any pixel that is set is the result of the shrinking of one or more pixels. As the regions are shrunk a flag is passed along with the shrinking data which reports whether the pixels that have contributed to that pixel have contained a seed point. This is achieved by the use of a logical OR operation. Hence when the region has been shrunk to a single pixel the processor that contains that pixel also contains a flag which reports whether any of the pixels within that region was a seed point. This allows those regions that did not contain a seed point to be removed. The next step of the Cypher algorithm is to reconstruct the region. In the labelling operation this consumes the vast majority of the cycles as the $2\log n$ bit labels have to be passed back as the region grows. In this application no label has to be passed back as it is simply a case of reconstructing a subset of the regions. This means that instead of taking 285 cycles per iteration the operation takes approximately 20 cycles giving an order of magnitude improvement, in addition to the constrained number of iterations gained by the use of the Cypher algorithm.

Hence the time taken to perform this operation is 27 cycles for each of the thresholds followed by $20d$ cycles for the labelling where d is the sum of the sides of the bounding box of the edge and another few cycles to remove regions that do not have a seed pixel. The total time for an image containing chains of 512 pixels long (the maximum for a 256 by 256 image) is only about a millisecond. This represents a significant improvement over previous SIMD implementations of the algorithm.

The use of the thinning with the Sobel and hysteresis thresholding operation provides a reasonable set of edges that may be used as input for processes such as line extraction described in the next chapter. The Sobel may very simply provide an edge magnitude and orientation quantised to eight levels. This provides an input to the edge thinning operation which leaves as output a grey scale image with single pixel edges. This may then be fed into the hysteresis thresholding algorithm to produce a binary edge image. This processing flow is shown in figure 4.9.

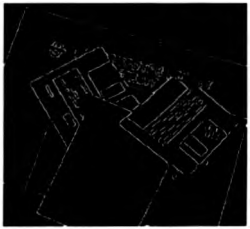
(a) Original Image

(b) Sobel Magnitude



(c) Sobel Orientation

(d) Thinned Image



(e) Hysteresis Threshold

Figure 4.9 - Edge Extraction Processing Flow

Figure 4.9(a) shows an image of two disks. Figures 4.9(b) and (c) show the magnitude and orientation of the result of the Sobel operator highlighting the edges. Figure 4.9(d) gives the image resulting from the thinning operator and finally figure 4.9(e) shows the result of hysteresis thresholding with the lower threshold set to 30 and the upper threshold set to 60. The number of iterations required by the hysteresis thresholding is 329. The times that each of these operations take is shown in table 4.4.

Operation	Time
Horizontal and Vertical Components	24μs
Magnitude and Orientation	50μs
Thin	54μs
Hysteresis Threshold	665μs
Total	793μs

Table 4.4 - Summary of performance on edge extraction stages on WPM

4.3 Intermediate and Transitional Tasks

4.3.1 Histogram Computation

Histograms are one of the most basic ways of extracting non-iconic data from the image. On a conventional SIMD architecture the histogram has to be accumulated on certain specified processors on the array which makes the load very unbalanced which is exactly what should be avoided on SIMD arrays. On WPM the accumulation may be passed on to the controllers which leaves the load very well distributed across the array.

As WPM does not have any direct mechanism to form global accumulation the computation of the histogram has to be broken down into two major phases. The first is to compute the histogram over each cluster and the second is to take the results from each cluster and combine them to form a global histogram. This second stage is only required if a global result is required - there are algorithms, such as Beveridge's segmentation algorithm [Beveridge 87], that process histograms locally and therefore do not need this stage of processing.

The most obvious algorithm for computing histograms on an architecture such as WPM is for the controller to broadcast each grey level value to the PEs [Howarth and Francis 88]. Each PE would then compare this broadcast value with the value it holds and set its output bit if they are equal. The controller then counts the number of PEs responding using the 'count response' mechanism. This algorithm benefits from being simple and also from being an intuitive solution. However, it is very slow mainly due to the time it takes each PE to perform 256 8 bit comparisons. A comparison of b bits takes roughly $2b$ cycles and therefore this operation including the counting is likely to take of the order of 5000 cycles or 0.5 ms.

A better method is to first decode each b bit value i, thus creating a 2^b bit number with a single bit set in the i th position. This removes the need to perform any comparisons and the controller simply loops through j=0 to 255 counting the number of PEs with bit j set. The decoding of an 8 bit value takes approximately 1500 cycles and then counting the 256 bitplanes takes approximately an additional 800 cycles. This 800 cycles is limited by the fact that each count requires 3 cycles to settle. Therefore this algorithm is approximately a factor of two faster than the simple algorithm. However it does consume a large amount of memory (a b bit decode uses 2^b bits plus up to 2^{b-1} bits of temporary storage). It is also clear that the 8 to 256 bit decode still consumes the majority of the cycles used by this algorithm, ideally the time taken should approach the time taken to count 256 values which is a fundamental limit.

A variation on this method makes use of the speed at which the bit serial PEs can manipulate single bits. Instead of decoding the eight bit value both of the nibbles in the byte are decoded independently to form two 16 bit values. Then each bit of one of these decoded numbers is logically ANDed with each bit of the other and this result is then counted, as illustrated in figure 4.10. This algorithm takes the time to count 256 values plus the time to decode two 4 bit numbers (as the AND operation and the PE memory reads are overlapped with the count). It takes 64 cycles to decode a 4 bit value (compared with 1500

cycles to decode an 8 bit value). Therefore the time taken by this operation is roughly 900 cycles or 90 μs which is not significantly longer than the limit of the time taken to count 256 times.

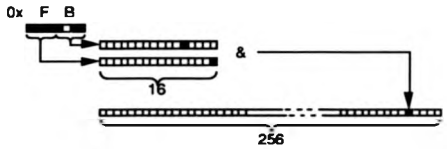


Figure 4.10 - Computation of Histogram using decoding

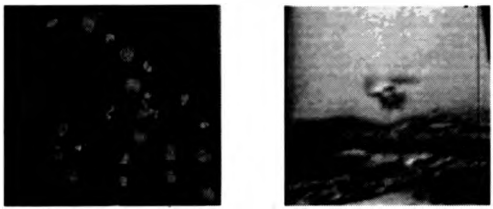
Having computed the histogram on each cluster independently the results then need to be accumulated over the whole array. Originally WPM was to have the facility to form a global count, as has, for example, the IUA [Weems 84]. However it lacks that facility as each cluster is identical and there is therefore no one single processor which can have the control of this. The count therefore has to be accumulated at the iconic or symbolic layers.

As a result of the previous stage the 256 histogram bins are stored in the memory shared between the cluster controller and the symbolic processor. The symbolic processors may begin accumulating the values as they arrive in the dual port memory at the same time as the controller and iconic array are computing the next value. The accumulation may take place using distance doubling techniques by adding the value to that of the immediate neighbour, then adding that to the neighbour two steps away etc. This takes a total of 6 steps for each value. The accumulation of successive values may also be overlapped.

4.3.2 Size Filtering

Size filtering is a method that has proved successful [WSTL 90] in detecting small 'blob' like regions in noisy images. Figure 4.11 shows two example

images that have objects that fit this criterion. The image on the left is of a microscope slide of a smear test. A count of the number of particles of various sizes is required from this image. The image on the right is an infrared image of a helicopter. The helicopter is near the centre of the image, the task in this case is to attempt to find it.



(a) Microscope Slide *(b) Helicopter*

Figure 4.11 - Example images

The median filter (section 4.2.4) has been widely used for the suppression of high frequency noise whilst attempting to retain the integrity of the edge information. In addition to this valuable role is may be seen that median filters can remove small regions that differ from the surrounding pixels.

Original		1 1 2 1 1 1 1 1 1 4 4 5 4 1 1 1 1 1 2 2 2 1 1 1 1 1 1
Median	3	1 1 1 1 1 1 1 1 4 4 4 4 1 1 1 1 1 2 2 2 1 1 1 1 1
	5	1 1 1 1 1 1 1 4 4 4 4 4 1 1 1 1 1 2 2 2 1 1 1 1
	7	1 1 1 1 1 1 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1 1
	9	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 4.12 - 1 dimensional median spot removal

Figure 4.12 illustrates how a one dimensional median filter may be used to remove regions of a certain size or smaller. The first line of the figure shows the original data, the other lines show the result of applying one dimensional median filters of the given sizes. The median filter of size 3 removes regions of a single pixel as may be seen on the left. The region of value 2 on the right

contains 3 non background pixels therefore a median of size 7 is required to remove it. Finally in order to remove the main region of size 4 pixels a median of size 9 is needed.

This shows how a median can be used to remove regions of varying sizes but in addition to this it may also be used to find regions of a given size. This may be achieved by performing two median operations and subtracting one result from the other. If the first median filter removes regions smaller than n pixels in size and the second removes regions smaller than m pixels, then if the result of one is subtracted from the other, only regions of sizes between n and m are left. Figure 4.13 shows this principle applied to the data given above in figure 4.12.

Original	1 1 2 1 1 1 1 1 1 4 4 5 4 1 1 1 1 1 2 2 2 1 1 1 1 1 1
Median 5	1 1 1 1 1 1 1 4 4 4 4 4 1 1 1 1 1 2 2 2 1 1 1 1 1
7	1 1 1 1 1 1 1 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1 1
Difference	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0

Figure 4.13 - One dimensional Size filtering

In figure 4.13 two median filters of size five and seven pixels have been applied to the original data. The results from applying the median of size seven are then subtracted from the results from the median of size five giving a result in which only regions of a size between the two are left - as shown on the last line. In a practical situation the sizes of the regions may not be known to such accuracy and therefore a pair of medians with a wider difference may have to be used. This operation may be extended to two dimensional data by applying two dimensional median filters allowing the removal of two dimensional regions.

The major problem with this method is the time it takes to compute the result of large median filters. This may be eased by using separable median filters. Separable median filters, like separable convolution filters, work by processing the image with an n by 1 filter followed by processing the result of that with a 1 by n filter. In the case of convolution this gives the same result as if the image were convolved with the n by n mask given by convolving the

n by 1 mask with the 1 by *n* mask. In the case of the median filter the result is not guaranteed to be the same result as the *n* by *n* filter but researchers such as Ruttimann and Webber [Ruttimann 86] suggest that in the general case 'separable median filters are for most practical applications equivalent to the corresponding one-pass two-dimensional median filters'. When the separable median is extended to the size filtering problem it is found that it is also suitable. Figure 4.14 shows the results of applying 4 different size separable medians to the microscope image from figure 4.11(a).

(a) 7 x 7 median

(b) 13 x 13 median

(c) 17 x 17 median

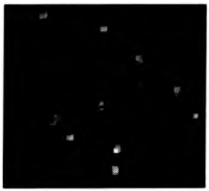
(c) 23 x 23 median

Figure 4.14 - Application of Separable Median Filters

In figure 4.14 it may be seen that the application of the 7 by 7 median has smoothed the image somewhat but has not removed any of the particles. The results of the 13 by 13 median show that a number of the particles have been

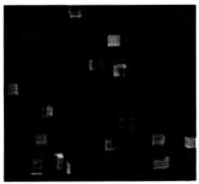
suppressed without affecting the larger particles. The 17 by 17 leaves the larger particles and the 23 by 23 median leaves very little except for a few smears where a number of particles have merged to form a low intensity smudge.

If the result of each of the three smaller filters is then subtracted from the result of the next larger median then three difference images are produced, these are shown in figure 4.15. These define three differing sizes, small, medium and large. Some particles appear in two of the images due to them being on the border of these sizes but these may be simply removed by a comparison of the centroids and radii.



(a) 13 x 13 - 7 x 7

(b) 17 x 17 - 13 x 13

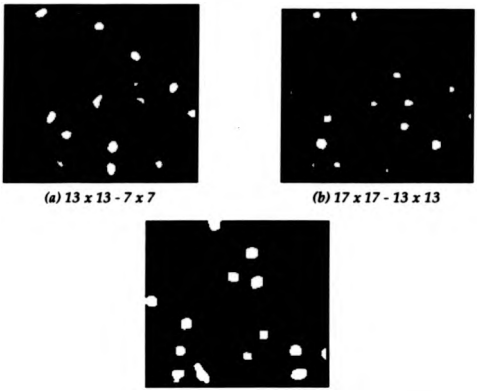


(c) 23 x 23 - 17 x 17

Figure 4.15 - Difference images

These may then be thresholded to produce three sets of regions as shown in figure 4.16. These three images give a good detection of the particles in the

original image. The centroid of each detected region then may be found to give the final result.



(a) 13 x 13 - 7 x 7 *(b) 17 x 17 - 13 x 13*

(c) 23 x 23 - 17 x 17

Figure 4.16 - Thresholded images giving detection

Section 4.2.4 has described the implementation of two dimensional median filters. This implementation is also valid for each of the one dimensional filters used in this application. Both the subtraction and the threshold operation are very simple and quick to perform on WPM. The performance figures for the above operation performing 4 different size median filters and subtracting and thresholding the results is shown in table 4.5. This table shows that WPM can perform this operation very well, taking a total of 1.6 ms for the entire detection operation.

7 x 7 Median	153 µs
13 x 13 Median	333 µs
17 x 17 Median	512 µs
23 x 23 Median	680 µs
3 Subtractions	8 µs
3 Thresholds	5 µs
Total	1691 µs

Table 4.5 - Performance of the Size Filter on WPM

4.3.2 Radii Segmentation

Having located a convex object such as a particle from an image such as those in figure 4.11 some applications may require each object to be accurately segmented. Such an application is outlined in [WSTL 90] and [Atherton 90]. In these references the application is range determination from the change in size due to motion towards the object and therefore a more accurate segmentation is required than that given as a side effect by the size filter. An approximate centroid and size is available from the detection phase and these may be used to assist a segmentation algorithm. This information allows an algorithm to be used that searches for the edge of the object as the approximate location of the edge is known. This gives a much more robust algorithm than one that has no external knowledge about the scene.

Firstly an edge detection algorithm is applied to the original image. Then if a one dimensional plot is taken from the centre of the object radially outwards in any given direction, the edge of the object will be crossed. This plot is a radial slice. If there is a change in grey level between the object and the rest of the image then this will be represented in the plot. A typical plot is shown in figure 4.17. The search window leading to this plot can be narrowed by

making use of the estimated size of the object to give a restriction on the minimum and maximum length of the spoke used. The maximum and minimum should bracket the boundary of the object on all spokes.

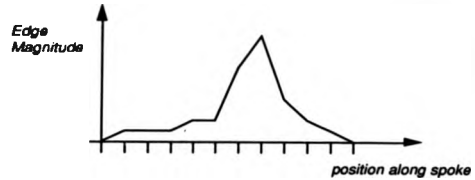


Figure 4.17 - Grey level plot of a radial slice

Having obtained a plot such as figure 4.17, the point of crossing the edge of the object is sought. The crossing point is picked as that with the maximum edge intensity within the search window. The width of the search window depends upon the reliability of the radius of the object and whether the centroid given is in fact near the real centroid of the object.

If this operation is performed for a number of equally spaced orientated slices then a segmentation of the object may be built up by linking the crossing points of successive slices. The more crossing points then the finer the segmentation. This process is illustrated in figure 4.18. Each spoke shown in figure 4.18 has two marks which indicate the search window. A plot is taken within this window and the blob shows the maximum position on the spoke.

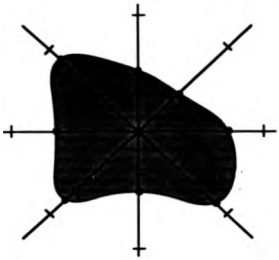
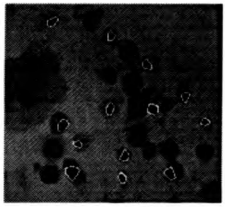
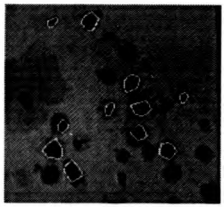


Figure 4.18 - Radii Segmentation

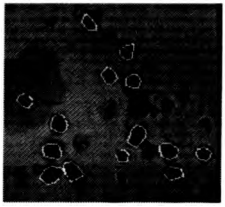
There will occasionally be errors in these crossing points due either to gaps in the circumference of the object or internal edges inside the object that are within the search window. The problem with internal edges can be minimised by having a small search window. The spokes may be smoothed to remove those spokes with a radius that is significantly different from their neighbours using a median filter. In addition to this, in order to obtain a convex segmentation, the convex hull of the crossing points is used. The result of the segmenter on the images shown in figure 4.11 are given in figure 4.19, the segmentation is shown by a white outline. It may be seen that the segmentation is of a very high quality.



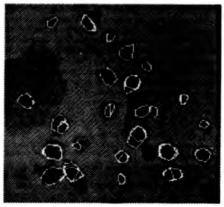
(a) 13 x 13 - 7 x 7



(b) 17 x 17 - 13 x 13



(c) 23 x 23 - 17 x 17



(d) Combination of (a)-(c)

Figure 4.19 - Segmentation results

The radii segmenter may be implemented on the WPM in the following manner. Given a minimum radius of r_{min}, a maximum radius r_{max} and the processor array containing a bit image C, such that a pixel $C_{x,y}$ is set iff there is an object centre at x,y. Each different spoke orientation θ is then processed in turn as follows. C is shifted in direction θ by r_{min} pixels, this gives another bit image D. Then image D is logically ORed with the result of D shifted by a single pixel in direction θ, this is performed r_{max}-r_{min} times. This leaves image D containing a straight line of pixels beginning r_{min} pixels from every pixel x,y such that $C_{x,y}$=1 in direction θ ending r_{max} pixels from x,y. The edge image is then logically ANDed with D leaving only the pixels of interest. The lines are simple to project in any orientation using any line drawing

algorithm, the same orientation is used over the whole image at once enabling the instruction stream to be modified according to the shift pattern produced by the line drawing algorithm. The result of this stage of the algorithm is a radial slice of orientation θ for each object.

The maximum value within each connected set of pixels, or slice, then has to be found. In a simple case reported in [Atherton 90] only one object, and hence only one centroid, was considered at a time. In this situation the maximum may be found simply by the use of the some/none response to find a global maximum. In the general case there may be several spokes within any cluster from a number of objects and a given spoke may well be split by a cluster boundary. This is guaranteed for spokes longer than 16 pixels. Therefore the simplistic method is only suitable in certain restricted circumstances.

In situations where the some/none response may not easily be used to compute the maxima then an alternative method to perform this is to allow each processor to compare its value with the next element on the spoke. This then propagates the maximum value along the spoke in a similar way to the connected component algorithm. Ideally as many of the spokes as possible should be considered at once. If a connected component type method is used to find the maximum then spokes that have crossed may not be considered at the same time. However due to the nature of propagating labels to neighbours on SIMD processors only a small degradation in performance will occur if the maxima of spokes of the different orientations are computed serially. This is because the maxima will only need to be propagated to the neighbour along the spoke rather than to all neighbours.

To make the calculation more efficient the spokes that radiate out 180° apart may be considered at the same time. The only condition that would cause these spokes to cross and hence this algorithm to break down is if two spokes of the same orientation run into each other. This will only occur if two centroids are processed that are within the given radius of each other. This should not occur as the objects will in that case intersect. However if it does

occur then the result is that the maxima are passed along to both regions. This would result in the segmentation being the intersection of the two objects which is a reasonable result.

The time taken by this implementation is the sum of the time to draw the spokes, the time taken to find the maxima on each spoke and the time taken to compare this maxima with the edge image to find the actual pixel location. The spokes are drawn serially and therefore the time taken is nkr_{max} where n is the number of spokes and k is the time taken to draw one pixel of the spoke. The maximum is found over two of the spokes at the same time therefore taking $n(r_{max}-r_{min})j/2$ cycles where j is the time taken to propagate and compare in a single direction at once, j is proportional to the number of bits in the edge magnitude image. Finally the time to locate the actual pixel is the time taken to perform a simple comparison. On the WPM the average value of k is equal to 3 cycles and j, with 8 bit data, is on average equal to 124 cycles. Therefore for 16 orientations with r_{max} equal to 12 and r_{min} equal to 8 the time taken to perform this is 460 μs.

4.3.3 Convex Hull

Finding the convex hull of a set of points is an intermediate operation that may be performed on any of the layers of the pyramid reasonably efficiently. The choice of implementation depends upon the form of the data. If there is only a single set of points then the SIMD array can provide an efficient implementation, however if there are several sets of points each of which requires a convex hull independently of the other then it may be easier to extract the points to the controller and to perform the operation at either that level or the symbolic level.

SIMD Implementation

A parallel version of the Jarvis algorithm [Jarvis 73] is used to compute the convex hull on the SIMD array. The first step of Jarvis' algorithm is to find

the point with maximum y coordinate, then if there are several of these the one with the maximum x coordinate is chosen - this is a guaranteed hull point. Secondly the point that makes the minimum angle to this point is found, this will also be a hull point. This is repeated until the original point is reached again. An addition necessary to make this simple algorithm work in all cases is that after two iterations, when three hull points have been found, any points that are interior to the polygon formed by the hull points are pruned.

This pruning causes some significant problems for any implementation as it involves a complex check with all of the other points to prune them from the search. It is however possible to reduce this pruning operation to the following condition without affecting the outcome of the algorithm. The first point is that with the maximum y,x coordinate, the next point is chosen is the next hull point anticlockwise etc. Once the point with the minimum y coordinate has been reached then from that stage onwards all points with a smaller y coordinate than the current hull point should be pruned. This condition while also requiring the comparison with all points is a much simpler check to perform as it simply involves a comparison of the y coordinates.

This may be clarified by study of figure 4.20. In this figure the dark blobs are the set of points. A blob with a circle around it is a point that has been marked as a hull point and a blob with a cross is a point that has been pruned because its coordinate is greater than or equal to the new hull point, pruning does not start until the angle computed is greater than 180°. The sequence of images illustrate the steps in the operation going from left to right and top to bottom.

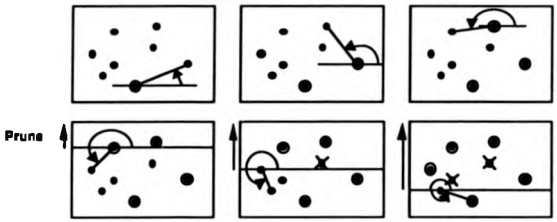


Figure 4.20 - Example of Convex Hull Algorithm

This algorithm may be implemented as follows. The maximum y coordinate is easily found by a global maximum operation. Then the coordinates of that point are broadcast to all PEs holding a point, each of them then finds in parallel the angle between the start pixel and their pixel. Instead of actually computing this angle the tangent may be used thus avoiding an expensive arctangent instruction. This may be done by computing the quadrant and the ratio of the difference of the x coordinates and the difference of the y coordinates. From this the point with the minimum angle may be found by using the some/none response. This operation may be repeated for the whole set of points. The pruning becomes active when the minimum angle has become greater than 180° or the operation has reached the 3rd quadrant. The pruning becomes a simple threshold operation, comparing the y coordinate with that of the current hull point. This is much simpler than the more complex pruning operation needed by the original Jarvis algorithm. This algorithm takes $O(m\log^2 n)$ steps for m hull points as the coordinates have logn bits in an image size n and a divide is required. The time taken for a set of points with 15 hull points is 639 µs, the majority of which time is taken by a single logn bit divide operation for each hull point.

This implementation uses both broadcast and some/none. This will work when either there is only a single set of hull points to be processed or when

each set is contained within a cluster which has no other hull points within it. In the case of a single set of hull points then there is no difficulty, in the case of multiple sets then it may be advantageous to shift the hull points to be contained within a single cluster. This is eased because the operation has no communication between processing elements and the data processed is not iconic but instead it consists of coordinates which may be shifted to different positions on the grid.

Symbolic/Controller Based Implementation

An algorithm may be implemented on the symbolic processor array that operates on the coordinate values as they are extracted from the iconic array. Each point, as it is extracted, may be compared with the points that have already been extracted and if it is within those then it may be ignored, if it is not then it might delete other points. This algorithm is not particularly efficient unless the structure of the extraction algorithm is such that the points become available sequentially with a reasonable interval between them as might be the case as for example with the radial segmenter. Alternately a sequential or MIMD version of the algorithm described above may be used.

4.3.4 Shape Property Extraction

There are a whole set of features that may be extracted from a successful segmentation of an object, these include centroid, area, extreme coordinate values, perimeter, a large set of moments and even the lengths of radial spokes as given by the radial segmenter. These vary in the ease of computation but generally the associative response from the iconic array makes this extraction process fairly efficient.

The area of a segmented object may be very simply computed by setting a flag in those processors that are a part of the object and performing a count, this takes 4 cycles including the set-up time. If the object extends over cluster

boundaries then the areas of the object within each cluster are accumulated independently and summed at the symbolic layer. Perimeters may be found nearly as simply. Every processor that contains a pixel which is a part of the object checks its neighbours to see if they are also, if they aren't then that processor contains a boundary pixel and it sets a flag. These flags are then counted in the same way as the area.

The centroid of an object is found by each processor that contains a pixel which is a part of the object passing each bit of its x coordinate in turn to the count response beginning with the top bit. Then the controller multiplies the running total by two (by shifting it) and reads in the count and adds it to the running total. When this has been performed for each bit the running total is divided by the area to give the x coordinate of the centroid. The y coordinate is found in a similar manner.

Extreme coordinate values are found by the use of the some/none to find the minimum or maximum values. Other shape properties may be found in similar ways using the associative responses.

4.4 Conclusions

This chapter is intended to show that the Warwick Pyramid Machine performs well at both iconic and intermediate level processing. Many algorithms have been described including several novel algorithms and many novel implementations of existing algorithms. Two processing flows have been given, one a simple shape detection, extraction and description flow and the other an edge detection and enhancement flow that is used as the prerequisite for the processing that takes place in the following chapters. The next chapter describes in much more detail the extraction of straight lines from the image and chapter six describes a symbolic operation that makes extensive use of the symbolic level of the machine.

5

Straight Line Extraction

5.1 Introduction

Straight lines form a key component of many image analysis systems, examples may be found in [Stockman 82] [Cass 88] [Grimson 90]. This is because straight lines are a common feature in a human made world and the goal of most image analysis tasks involves locating and recognising artificial objects. Furthermore, straight lines are low level features from which more complex properties may be constructed, including polygons and piecewise linear curves. As a result straight lines are often an important early stage in the iconic to intermediate data translation process.

Many different methods have been proposed to detect and extract straight lines, the most popular of which by far has been the Hough transform and methods based on it. Both the Hough transform and the other algorithms

described in this chapter work on an edge image which may be extracted using methods outlined in the previous chapter. Alternatively if a line image (e.g. a scanned engineering drawing) is being processed then the preprocessing simply consists of thinning and thresholding as necessary.

This chapter initially describes some of the non-Hough methods in order to give background but then concentrates on the Hough transform. In the past the Hough transform has proved difficult to implement efficiently on many parallel architectures, the reasons for this and some methods of overcoming these problems are suggested. One of the reasons is that it is a global transformation which requires much global communication. The implementation described here overcomes this by extracting straight lines locally and remapping the transform as a set of projections. It is shown that this is not only more efficient but also gives a more useful result. A method is devised to take these local linear features and to form global features where appropriate. An efficient implementation of this on the Warwick Pyramid Machine is described. Finally some results are given with some comparative execution times.

5.2 Non-Hough Methods

Straight lines may be found simplistically by searching for contiguous edge pixels that exhibit no variation in orientation. For example, if an edge image were represented by a chain code then straight lines could be found by searching the chain code for sections that contain repetitions of any particular code. However methods such as this are only capable of finding very simple lines that contain no deviation and hence are of little practical value. More sophisticated algorithms such as those described by Ramer and Pavlidis [Ramer 72] [Pavlidis 74] attempt to fit straight lines to the pixel data using matching algorithms such as least squares fitting. A set of contiguous pixels is extracted and a straight line is fitted to the set using a least squares error fit. Having fitted this straight line to a set of pixels that may or may not be a straight line in reality, a split and merge algorithm is necessary, driven by the

deviation of the fitted line from the data, to more closely match straight line segments to the edge image.

Burns et al [Burns 86] use a more sophisticated algorithm based on a similar least squares method. Initially an edge orientation image is formed using a similar method to that described in the previous chapter. This image is then quantised into eight or sixteen equally spaced orientation partitions. A connected component labelling operation is then performed on the quantised orientation image to attempt to segment it into regions with each region containing only edge pixels of similar orientations. Finally a weighted least squares fit is applied to the pixels within each connected region to find the best fit straight line, the weights being derived from the edge magnitude.

All of these techniques suffer from the fact that the operation is very local in nature. Breaks in the edge cause the algorithms to break down and local edge orientation information is notoriously inaccurate. Hence these algorithms tend to be somewhat unstable in use.

5.3 The Hough Transform

5.3.1 Introduction

Described generally, the Hough transform is an algorithm that detects loci of objects. In its most common form it is used to detect straight lines within two dimensional images, however it is not restricted to this application and may be used to detect arbitrarily complex objects [Ballard 81]. Straight line detection is achieved by creating a set of orientated projections in the image plane and integrating the image along each set of these straight contours forming a transformed image in which straight lines in the image plane appear as peaks in the transform domain or parameter space. The Hough transform alters the problem from a difficult global detection task in the image space to a local detection task in the parameter space.

The Hough transform has become a very commonly used algorithm in image processing [Skingley 87] [Duda 72]. Because of this interest, the flexibility of the algorithm and the implementational difficulties it poses for parallel architectures, it has been included as a component of both of the DARPA image understanding benchmarks [Rosenfeld 87] [Weems 88] and that has driven many implementations on diverse parallel architectures.

The Hough transform was first described in a United States patent in 1962 [Hough 62] but it has been pointed out [Deans 81] that it is in fact merely a special case of the more general Radon transform first described in 1917 [Radon 17]. The Radon transform can be applied to problem spaces of any dimension, however here the problem space will be restricted to single images and straight lines and thus is two dimensional. The Radon transform of a grey level image f(x,y) is a set of orientated projections of the image [Helgason 83] as given in equation (1).

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, d(\rho - x\cos\theta - y\sin\theta)\, dx\, dy \qquad (1)$$

where d(r) is the Dirac delta function which is zero except where r = 0 where it is 1. For a discrete image of size N by N equation 1 may be rewritten as follows:

$$\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\, d(\rho - x\cos\theta - y\sin\theta) \qquad (2)$$

The term d(ρ - x cos θ - y sin θ) in this definition forces the integration of the image f(x,y) along the straight line given by equation 3.

$$\rho = x\cos\theta + y\sin\theta \qquad (3)$$

The Hough transform as described by Hough uses the more common and simple representation of a straight line given in equation 4, where m and c are the slope and y axis intercept respectively.

$$y = m x + c \tag{4}$$

However, this causes difficulties as both the slope and the intercept in this are unbounded. If the original form of the Hough transform is modified to use the 'normal parameterisation' given in equation 3 as described by Duda and Hart [Duda 72] then this transform is simply the case of the Radon transform where the input image is binary [Cypher 87]. In practice the term Hough transform is commonly applied to the complete transform given in equation 2 and is therefore used for both grey scale and binary images. It is interesting to note that the term Hough transform is generally somewhat misused to describe any straight line extraction algorithm that makes use of projections. Equation 3 may be simply rewritten as follows:

$$\rho = \sqrt{x^2+y^2}\ \cos\left(\theta - \tan^{-1}\left(\frac{y}{x}\right)\right) \tag{5}$$

The operation of the Hough transform can be seen by reference to the example given in figures 5.1 and 5.2. If each of the three colinear points in figure 5.1 is transformed using equation 5 then three sinusoids are created in the transform domain (figure 5.2), the point at which they cross gives the parameters ρ,θ. These are the parameters of the straight line that passes through all three points given in figure 5.1.

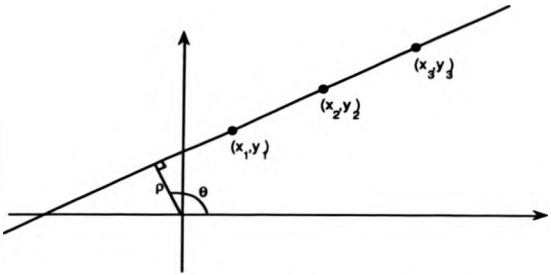


Figure 5.1 - 3 colinear points on line $\rho = x\cos\theta + y\sin\theta$

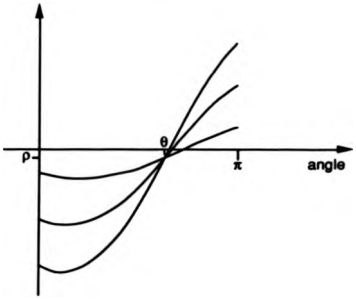


Figure 5.2 - Transform space of the 3 colinear points given in figure 5.1

This process may be summarised as follows [Duda 72] [Deans 81].

1. A point in the image space corresponds to a sinusoid in the parameter space.

2. A point in the parameter space corresponds to a straight line in the image space.

3. Points lying on the same straight line in the image space correspond to sinusoids that pass through a common point in the parameter space.

4. Points lying on the same sinusoid in the parameter space correspond to straight lines that pass through the same point in the image space.

Generally the Hough transform is performed by quantising the continuous parameter space into a discrete accumulator array. The sinusoids produced using equation 5 are then plotted into this array which becomes effectively a two dimensional histogram with each bin containing a value corresponding to the number of sinusoids that cross its edges. The straight lines can then be found by finding bins in this discrete accumulator array that contain local maxima. For an image containing a single straight line this is simply a case of finding the bin that contains the maximum value across the whole array, the coordinates of this then give the values ρ and θ. From these the infinite line linking the component pixels can be found as shown in figure 5.1 which then has to be compared with the original data in order to find the endpoints. In the more general case of finding multiple straight lines within an image then each peak has to be found separately.

Finding these multiple peaks can be a complex task as some peaks may be caused by false alarms whereas some genuine peaks may be obscured. If θ is quantised into n_θ quantisation levels (where n_θ is generally greater than the resolution of the original image) then each edge pixel in the image contributes to n_θ bins in the parameter space. Normally that edge pixel will only be a contributor to a single straight line in the image space and therefore n_θ-1 of the bins will have been added to unnecessarily. These add to the background clutter resulting in small peaks being difficult to distinguish from the background. The parameter space of the edge detected disk image is

shown below in figure 5.3. This figure shows the difficulty that is encountered when attempting to locate the peaks in the parameter space resulting from a complex scene. Peaks that lay in the outer parts of the space are not too difficult to detect, however those in the central areas of the space lie in large amounts of background clutter.

Figure 5.3 - Hough parameter space of image 'disk3'.

In addition to the problems of finding peaks there are several different errors that can lead to a blurring of these maxima which introduces an error in the result obtained from the peak, such as:

- Insufficient quantisation of ρ causes points that are adjacent in the image to create sinusoids that pass though the same bins in the parameter space near the peak thus blurring the peak, making the location uncertain and introducing additional erroneous peaks. Short straight lines give rise to sinusoidal intersections that are at very shallow angles. Finding these intersections accurately is a well known ill-conditioned problem.

- If the straight line that is being transformed is not a single pixel width line but instead a thin stripe of several pixels then there are a number of single pixel width straight lines of slightly varying ρ and θ that can

be plotted through the stripe. This leads to a blurring of the peak in both the ρ and θ dimensions.

Given n_e edge points in the image and quantising the angle to n_θ levels the Hough transform takes $O(n_e n_\theta)$ time. n_e is proportional to n^2 and n_θ is proportional to n therefore the problem is $O(n^3)$. The amount of storage required by the parameter array is $O(n_\rho n_\theta)$. Therefore, while ideally large values are required for both n_θ and n_ρ, this is limited by both the computation and the memory consumption of the algorithm.

5.3.2 Detecting Shapes with the Hough transform

As the Hough transform is simply an integration along projections it can be extended very easily to detect different shapes [Ballard 81] [Deans 81] [Davies 88] [Yuen 89] by making use of different projections. One of the most simple of which is the modification to detect circles. A circle can be described by equation 6.

$$r^2 = (x - x_0)^2 + (y - y_0)^2 \qquad (6)$$

where r is the radius of the circle, and (x_0, y_0) is the centre.

From this it can be seen that a circle is described by three independent parameters all of which are bounded by the image size. This means that the parameter space will be 3 dimensional, one dimension being the radius, one the x coordinate of the centre and the other being the y coordinate of the centre.

Ballard [Ballard 81] describes a more general method for using the Hough transform to detect completely arbitrary non-analytic shapes by using the orientation of boundary points to map the shape into the parameter space. This has advantages over traditional matched filter convolution methods as translational, rotational and scale changes in the image space simply produce translations in the parameter space.

5.3.3 Enhancing the Hough Transform

One of the greatest difficulties with the Hough transform is finding the peaks in the quantised parameter space. Many of the straight lines that the Hough transform is used to detect will have component pixels that are close together. Pixels with similar coordinate values will generate similar sinusoids in the parameter space. This obviously makes the peak harder to find as the intersection is between curves with only a very small angle between them. Figure 5.4 illustrates this problem, it depicts the parameter space given by a single short straight line. It can be seen from this that the peak is somewhat blurred.

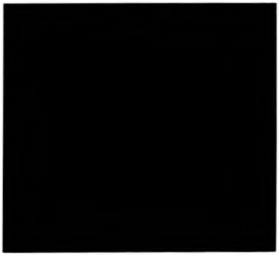


Figure 5.4 - Example of the parameter space given by the transform of a short straight line

The problem of locating peaks is exaggerated by the remainder of the sinusoids that are not near the intersection, these can very easily interfere with the peaks generated from other straight lines causing errors in those peaks.

A partial solution to both of these problems is possible if some edge orientation information is available. In many cases the Hough transform is applied to images that are the result of edge detection. In these cases it is

possible to generate an edge orientation image in addition to the edge magnitude image. If the gradient of each pixel that is to be transformed is known to a certain degree of tolerance then any straight line that passes through it will have the same orientation. This places a restriction upon the number of orientations that need to be considered and thus equation 5 becomes:

$$\rho = \sqrt{x^2+y^2}\cos\left(\theta - \tan^{-1}\left(\frac{y}{x}\right)\right) \quad (\theta_0 - \Delta\theta) \le \theta \le (\theta_0 + \Delta\theta) \qquad (7)$$

where $\Delta\theta$ is the error associated with the gradient of the pixel. This is known as a polarised Hough transform. In the most extreme case it has been proposed [Lotufo 89] that a single orientation (i.e. $\Delta\theta=0$) be considered. This method does somewhat limit the robustness of the algorithm and can only be used in simple imagery.

This enhancement makes significant reductions in computation as the number of values of ρ that need to be calculated is heavily reduced and it also makes the parameter array easier to interpret. The computation changes from $O(n^3)$ to $O(n^2\Delta\theta)$. Figure 5.5 shows the accumulator array generated from the same short straight line as figure 5.4 but this time using equation 7 with $\Delta\theta$ set to 5%, i.e $\pi/20$.

Figure 5.5 - The parameter space given by the transform of the same straight line as figure 5.4 but with the addition of orientation information

Another consideration that gives rise to errors in the parameter space is the thickness of the line. If the line is greater than a single pixel in width then there will be a spreading of the peak in the parameter space in both ρ and θ. This problem is generally solved by running an edge thinning operator on the edge image before transforming the image.

An interesting method that eases the peak finding problem is described by both Li et al. [Li 86] and Illingworth and Kittler [Illingworth 87]. The two methods described are essentially the same except in detail. Both use a hierarchical approach to the quantisation of the parameter space. The n-dimensional parameter space is recursively divided up into n-dimensional hypercubes (squares in the case of detecting straight lines). Initially, the transform is applied transforming the image to a low resolution parameter space. Edge pixels that contributed to areas of the parameter space that exceed a certain threshold are noted and the transform is reapplied to those areas of the parameter space in higher resolution. This can then give any quantisation resolution required by continuing the process of zooming in as long as necessary, while making use of less storage space and executing much faster than a conventional transform giving the correspondingly accurate

accumulator array. This is achieved by effectively ignoring the less important parts of the parameter space.

5.3.4 Using Projections

If the Hough transform is approached from a different viewpoint then some of the problems described above may be eased. Consider in turn each quantised value of θ. If, for a given value of θ, ρ is computed for each pixel in the entire image[1] then the Hough transform may be understood as a set of projections or as a matched filter. Figure 5.6 shows a 16 by 16 image, computed using equation 3, containing the values of ρ for θ equalling 45°. The values of ρ have been truncated to the integer values shown.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	18
3	4	5	6	7	8	9	10	11	12	13	14	15	16	18	19
4	5	6	7	8	9	10	11	12	13	14	15	16	18	19	20
5	6	7	8	9	10	11	12	13	14	15	16	18	19	20	21
6	7	8	9	10	11	12	13	14	15	16	18	19	20	21	22
7	8	9	10	11	12	13	14	15	16	18	19	20	21	22	23
8	9	10	11	12	13	14	15	16	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16	18	19	20	21	22	23	24	25
10	11	12	13	14	15	16	18	19	20	21	22	23	24	25	26
11	12	13	14	15	16	18	19	20	21	22	23	24	25	26	27
12	13	14	15	16	18	19	20	21	22	23	24	25	26	27	28
13	14	15	16	18	19	20	21	22	23	24	25	26	27	28	29
14	15	16	18	19	20	21	22	23	24	25	26	27	28	29	30
15	16	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Figure 5.6 - Image computed by $\rho = x\cos 45^\circ + y\sin 45^\circ$

The underlying binary edge image is considered as a mask and then it is compared with the image given in figure 5.6. If the mask and the image of figure 5.6 are logically ANDed together then the result is that any lines in the image that match the projection are labelled with the ρ value of that projection. This may be seen in the following two figures. Figure 5.7 is an

[1] Instead of the conventional implementation of computing ρ for a varying θ on each pixel in turn.

example of a thresholded edge image and figure 5.8 is the image resulting from ANDing this with the image given in figure 5.6. This resultant image should therefore show straight lines of orientation 45°.

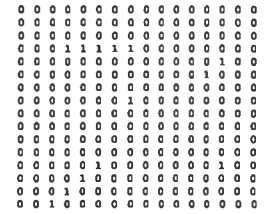


Figure 5.7 - Example edge image

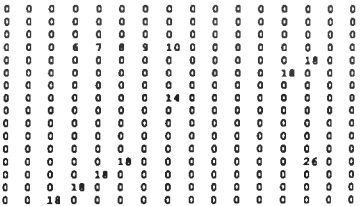


Figure 5.8 - The resultant image

Having computed this labelled image then the straight lines of that particular orientation may be found by searching this image for multiple occurrences of a single label. The label count produced gives the height of the peak in the equivalent parameter space. In the example in figure 5.8 there are 6 occurrences of the label 18 and a maximum of 1 of any other label. This suggests that there is a straight line at angle 45° with six pixels set on it. To find the actual length of the line the endpoints need to be found. This is a

simple operation given the labelled image in figure 5.8. This process is repeated for each orientation under consideration.

Figure 5.9 shows the parameter space formed by applying the traditional method of computing the Hough transform to the image in figure 5.7. This parameter space is generously sized at twice the size of the original image. Seven local maxima with a value of 3 or greater have been circled. The largest peak does correspond to the correct line; however, of the other local maxima only one matches a correct line and that is blurred over two bins which reduces the accuracy of the result. Despite the parameter array being unusually large at twice the size of the image it may be seen that the results have to be very carefully extracted from it in order to minimise errors. The longest line gives a good peak of size 6 because it is made up from pixels that are spread across the image. This means that the peak is made up from the intersection of more diverse sinusoids than, for example, is the case for the peak of size 5, that is blurred because the constituent pixels are contiguous. This illustrates the point that the Hough transform works best for lines that are non-contiguous. Despite the best line generating a good peak of size 6 the additional peak of size 4 to the right of this peak has also been caused by the same component pixels and hence is an error. Another point of importance that may be seen from figure 5.9 is that the parameter space is not fully used, this is partly because only a small part of the image is being used by the edge pixels and partly that some parts of the parameter space generated by equation 5 cannot be crossed by a sinusoid.

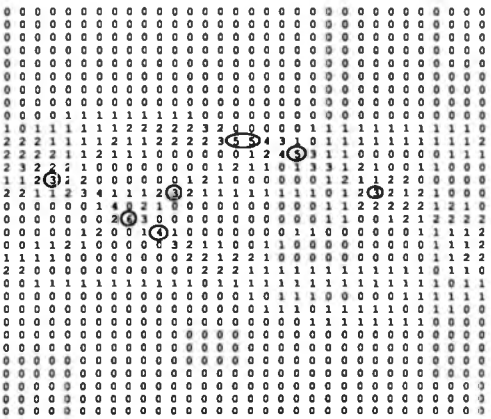


Figure 5.9 - The parameter space of the image in figure 5.7

Having extracted those labels of significance from figure 5.8, the endpoints of the line segment corresponding to each label may be simply be found by finding the extremes of each set of pixels with a single label.

As this projection line extraction method is simply a reordering of the computation of the conventional Hough method then some of these benefits are illusory. In the example given above, the peak of size four in figure 5.9 will also occur using the projection method for that given orientation. However, when using projections it is possible, once the first line has been found, to remove those pixels that contributed to it from consideration. This prevents these pixels from triggering more, possibly false, lines. The removal of these pixels leads to an iterative method where the best line in the image is extracted first, the pixels that made up that line are then removed from the image and then the next best line is found etc.

This iterative method considers the best line in a given iteration to be the longest line found on that iteration. A different criterion that can be used to judge the quality of a line is the ratio of the number of pixels set to the length of the line, this gives a measure of the density of the line.

5.4 Hierarchical Methods

5.4.1 Introduction

It has been shown that both the Hough and the other methods used for the extraction of straight lines are seriously flawed. It was shown that one way in which the Hough method can be improved was by the use of projections and iteratively extracting the lines. However this is impractical for large images as the lines are extracted sequentially. A method that may overcome this problem and others is to extract the lines hierarchically. Hierarchical methods involve splitting the image up into a number of sub-images and processing these sub-images independently.

5.4.2 Advantages of Hierarchical Extraction

One of the features of the Hough transform is that it is a global transformation. This means that a straight line that is fragmented may be joined and evidence of a straight line may be compiled from the whole image. However, this feature can also be its weakness. It is this feature that makes the transformation difficult to implement efficiently on parallel architectures as each image point may contribute to a large area of the parameter space. In addition, the benefits of this global approach are somewhat in doubt as the evidence for straight lines is generally fairly local and global straight lines that are not justified at the local level are difficult to justify at the global level without the benefit of a model of the scene. For example, in figure 5.10 the two collections of points 1 and 2 will in many cases be easier to match together as a single line than the two collections 3 and 4. This is because as described above the contributions that 3 and 4 make to the

parameter space will be blurred and the peak hard to separate whereas the sinusoids made by 1 and 2 will intersect with a large angle thus making the peaks more pronounced. This is certainly not desirable in all applications, in fact it is not clear that groups 1 and 2 should be matched at all as there is little assurance that such an edge, with a large gap in it, is in fact formed from a single object in the image. It appears that there needs to be maintained some locality in the data, yet genuine long straight lines need to have their component fragments grouped together as a single line.

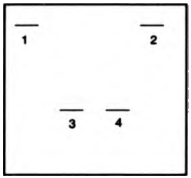


Figure 5.10 - Local vs Global Line Evidence

A natural solution to this problem is to use a hierarchical extraction method where the straight lines are extracted locally and then some global unifying process takes place to allow global lines to be formed that are supported by locally obtained line segments. This may be achieved by splitting the image up into a number of tiles and then performing the line extraction process on each of these tiles independently [Princen 90] [Bongiovanni 90] [Francis 90]. The result of this is that a number of local straight line segments are extracted from each tile.

It is likely that at least some of these line segments will have been fragmented by the fixed boundaries of these tiles. Therefore, a further process of merging these fragments needs to take place. This solution offers several key advantages over the global line extraction process in addition to those outlined in the previous paragraph.

The number of possible lines with a given angle in an image or the largest number of parallel lines is proportional to the size of the image. This is also the case for the number of different orientation lines representable within the image. To put this in terms used previously both n_θ and n_ρ are proportional to n. Therefore, if the line extraction process is performed locally over small sub-images then the number of different projections that need to be considered in each sub-image is less than the number of different projections that would have to be considered in the complete image. While this is true it has to be born in mind that any global line constructed from a number of shorter component lines of a limited number of orientations does not have the same limits. For example, if a tile has a line of orientation 0° running along its bottom row of pixels and the tile to the right has another line of orientation 0° running along the next to bottom row of pixels then the global line formed by combining these two line segments has an orientation greater than 0° yet less than the smallest non zero orientation representable within an individual tile. Therefore the line matching and merging process needs to take this into consideration and to allow an increase in the quantisation of the orientation when merging line segments. If this is done then limiting the number of orientations in each tile does not necessarily limit the number of global orientations. If this is not done as for example in [Bongiovanni 90] then the complete set of global orientations needs to be searched for within each tile so that nothing is lost as the line matching process proceeds.

5.4.3 Review

Several researchers have recently noted the advantages to be gained from hierarchical methods for straight line extraction. The method described by Bongiovanni et al [Bongiovanni 90] is based on computing a global Hough transform hierarchically on a pyramid of SIMD processors. They compute a Hough transform on each 8 by 8 patch of pixels at the base of the pyramid resolving the peaks into the PE array - they then combine these at the upper levels of the pyramid to form the longer lines. This method, because it attempts to reproduce the global results, while it has some of the advantages

of the hierarchical method described above misses others. Therefore although the Hough transform is computed locally, global properties still dominate, forcing a full range of orientations to be computed at each tile.

In [Davies 90] Davies and Wilson describe a line extraction process using the multiple resolution Fourier transform. This breaks the image into blocks of a range of sizes and analyses the spectrum of each block. A single straight line is assumed in each block which will appear in the spectrum of the block as an ellipse orientated perpendicular to the line. The block size is varied allowing a hierarchical set of straight lines to be built up. In [Davies 90] there is no combination of neighbouring blocks to form global straight lines.

Princen et al [Princen 90] describe a hierarchical method that performs local Hough transforms on overlapping tiles. The ρ and θ of each line segment, or the 'foot of the normal' is used to initiate another Hough transform searching for colinearity of these parameters. This technique is discussed in more depth in section 5.5 when describing the Warwick method given by Francis et al [Francis 90].

5.5 Warwick Hierarchical Line Extraction

5.5.1 Introduction

It has been stated above that there are certain advantages to be obtained from treating the extraction of straight lines in a hierarchical manner. The method described in this section uses this approach combined with the projection method given in section 5.3.4 for the local extraction of the straight line segments. It is believed that these two methods complement each other well to give a good overall straight line extraction algorithm that produces good results with potential improvement in performance on parallel architectures.

5.5.2 Matching

As described in section 5.4 when line segments are extracted from sub-images it is necessary for some merging of fragments to take place in order to produce lines that extend across the whole image. The first stage in this process is to identify which line segments have been fragmented, this is achieved by searching for colinear segments, or by matching line segments.

The most obvious method of extracting colinear line segments from the global set of lines is to perform another Hough transform on some criteria based on the set of line segments. If each line segment is transformed (rather than the pixels that make up the line) then the transformation is applied to a much reduced data set when compared to the original transformation and therefore the computation is much reduced[2].

The simplest means of applying the transformation to the line segments is to select a candidate pixel from each of the segments and to search for colinear pixels in a similar way to an ordinary search for straight lines. The most obvious choice for the pixel is the centre of the line segment as this is least likely to be affected by a deviation from colinear. Clearly if the line segments are colinear then the centres of the lines will also be colinear. The search may be performed by computing a Hough transform putting the line segment centre into equation 5. In this case the orientation information can be extracted accurately as it is computed over the length of the line segment rather than a small local neighbourhood as in the case of the edge orientation. This means that equation 7 may be used with $\Delta\theta$ being inversely proportional to the line length. This method is illustrated by figure 5.11.

[2] Noting that the time taken is proportional not to the size of the image but to the number of points to be transformed.

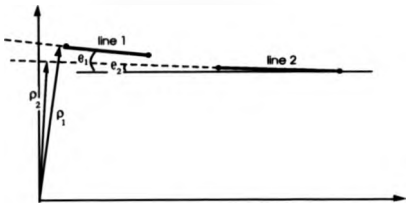


Figure 5.11 - The use of ρ and θ to compare line segments

An alternative approach that improves on this, is to compare each line with each other line to check whether they are colinear. This is potentially computationally less efficient but does not require the use of a parameter space and therefore does not introduce the problems involved with quantising the space. This is feasible because the number of points to be compared is equal to the number of line segments unlike earlier where the number of points was equal to the number of pixels above some threshold value. This operation is performed by comparing the ρ and θ values of each line segment with those from every other line segment. If they are within certain limits, which are set according to how much merging is to take place, then the lines may be considered colinear. After all of the comparisons have taken place then all line segments that have been matched are merged together to form longer lines. This method introduces an additional concept in that a further criteria for merging can be applied - the gap between the line segments. This gives the key advantage of allowing the rejection of pairings of lines that are too widely separated. This gap may be measured in terms of the lengths of the lines being compared.

A problem that occurs with both of the above methods is that the effects are not necessarily uniform across the image. For example, consider figure 5.12 where there are two pairs of lines ((1&2) and (3&4)). The pairings differ from

each other only by a translation and therefore should either both match or neither match.

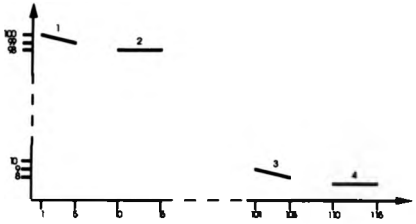


Figure 5.12 - Example line segments

The values of ρ and θ computed for each of these four lines are shown in table 5.1.

	ρ	θ
1	97.2568	1.32582
2	98	1.5708
3	34.1975	1.32582
4	8	1.5708

Table 5.1 - Parameters of line segments in figure 5.12

From table 5.1 it can be seen that while the pairs of line segments have identical orientations (as expected as only a translation is involved) the values of ρ differ widely. A matching algorithm might well consider line segments 1 and 2 as colinear as both the ρ and θ values are very similar. However, the same is not true for line segments 3 and 4 as in this case the ρ values are sufficiently different that it would be very difficult for a matching algorithm to justify merging the two line segments. This is obviously a fatal flaw in this method, as taken locally the pairs of line segments are identical in their relationship to each other and therefore should be treated in the same manner. When examined more closely with the aid of diagrams such as

figure 5.13 the cause in the disparity between the values of ρ becomes more clear.

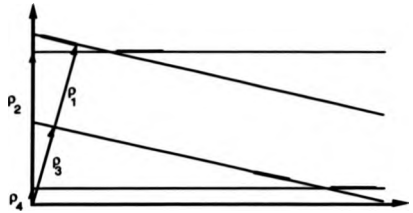


Figure 5.13 - Calculation of ρ and θ for example line segments

The method described by Princen et al [Princen 90] uses the foot of the normal instead of the centre of the line segment. Their method involves retaining the value of ρ and θ from each line segment and using this local value to compute a global value for use in comparison with other line segments. The line segments are matched with neighbouring line segments using equation 8, where Δx and Δy are the x and y offsets of the local origin and ρ_0 and θ_0 are the values of ρ and θ of the line segment. This method somewhat overcomes the problems shown in figure 5.12 although the problem is merely limited rather than completely solved.

$$\rho = \rho_0 \cos(\theta - \theta_0) - \Delta x \cos\theta - \Delta y \sin\theta \qquad (8)$$

Figure 5.14 shows two sets of colinear line segments again differing from the other by only a translation. The grid imposed on the line segments is the grid corresponding to the local areas over which the line segments are extracted. The crosses are the local origins of each local area.

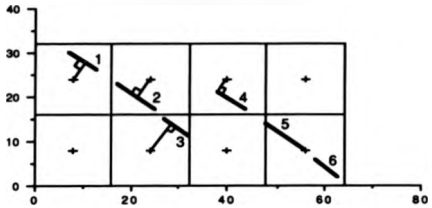


Figure 5.14 - Princen's method applied to 2 sets of line segments

Segment #	1	2	3	4	5	6
ρ	28.85	28.60	28.58	38.55	40.00	40.92
θ °	56.30	53.13	51.34	56.30	53.13	51.34

Table 5.2 - The parameters of the given line segments

If the values of ρ and θ given in table 5.2 obtained from each line segment locally in figure 5.14 are plotted then it is found that as expected they are approximately colinear. Now as each of the two sets of line segments differ by only a translation from each other they should be equally colinear.

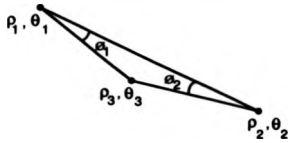


Figure 5.15 - Point colinearity measures

Figure 5.15 illustrates a set of three near colinear points and the measures ø1 and ø2 which are the angular deviations of the third point from linear. In the first set of line segments $ø_1$=2.48° and $ø_2$=1.39° and in the second $ø_1$=1.66° and $ø_2$=0.959°. Therefore while these examples are both clearly very close to colinear one appears more colinear than the other by a factor of nearly 50%. This should not be the case for there exists some threshold value where one of the set of lines would be matched as colinear whereas the other would not.

The method of detecting colinear line segments using modifications to the Hough transform has been shown to have several problems. Firstly there is the serious problem that the process is not uniform across the image and secondly many of the original problems of the Hough, such as matching two short line segments that have a large gap between them, are not necessarily overcome. A method that does overcome these problems at some computational cost is illustrated in figure 5.16. In this case each line segment is compared with each other line segment but instead of comparing ρ and θ the three parameters shown are used. The three parameters [Lowe 87] are the distance from the middle of the shorter line segment to the extension of the longer line segment (d), the smallest gap between the endpoints of each line segment (g) and the difference in angle between the two line segments (θ_d). If all of these measures are within the specified bounds then the two line segments are considered colinear.

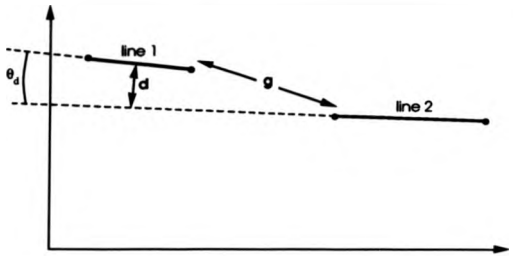


Figure 5.16 - Line Segment Colinearity Parameters

The angular separation, θ_d, clearly has to be small for the line segments to be considered colinear. The gap, g, is more complex and should be related to the length of the line segments being matched. For example, if the two line segments are small with a large gap between them, then there is less justification than if the lines are long compared to the size of the gap. The value d gives a measure of the lateral separation of the two line segments. If this is large then the lines should not be matched. For example, line segments with a gap equal to d and a zero angular separation may only be considered colinear if they are not too far apart - in which case they are parallel.

This method produces results that are uniform across the image which is obviously desirable. However, it still requires that every line is compared with every other line, this is an $O(n_s^2)$ task where n_s is the number of line segments. This can be reduced by only comparing a line segment with other line segments that are within a given angular separation of it. This may be achieved by sorting the line segments into bins and only comparing adjacent bins. A bin sorts of n_s data such as this, with a known range, is an operation that only takes $O(n_s)$ time. The comparisons then take $O(2\theta_d n_s^2 / \pi)$ time and if the ratio of θ_d / π is low then this is cost effective.

An enhancement that can be made to this method is to iterate the matching and merging process so that line segments that are very well matched to each other are matched and merged first and the result of this is then processed again with the parameters relaxed somewhat to allow less rigourous matching to take place. This enhances the operation, allowing high quality lines to be built up before being affected by matches of a lesser quality. It also allows the process to operate in a hierarchical fashion matching the local line segments prior to matching those further apart. In fact this speeds up the operation as it reduces the overall number of comparisons by merging several good local line segments to a single line which may then be compared with global lines rather than comparing each of the component line segments individually.

5.5.3 Merging

Introduction

Having marked two or more line segments as approximately colinear using the methods outlined above it is clear that a strategy needs to be devised to combine these line segments into a single straight line. This is a simple but very important operation that is somewhat overlooked in the literature. Most simple methods of merging the line segments tend to introduce errors by placing large dependencies on short and potentially unreliable line segments. The simplest method is to form the new line by joining the two most extreme points in the set of line segments.

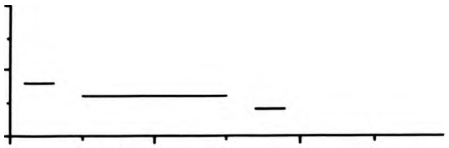


Figure 5.17 - Example Line Segments

This is demonstrated by figure 5.17 and figure 5.18. In figure 5.17 there are 3 line segments that have been marked as approximately colinear which all have an orientation of 0°. If a line is drawn from the two most extreme points on this set of line segments then the orientation of the resultant line is very different from that of the component segments. It is also the case in this example that the most significant line segment makes no contribution to the composited line.

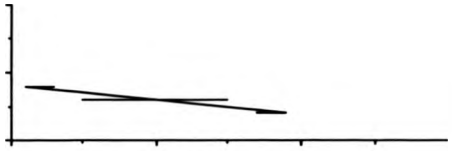


Figure 5.18 - Linking extreme endpoints

The ideal technique will produce the same composite line if the component line segments were split further - ideally as if the line segments were broken down to their component pixels. This may be achieved by fitting a straight line using a generalised least squares on the end-points of the line segments with the length of the line segments as weights. This gives the parameters of the new line which then needs some additional processing to find the new endpoints.

Finding Line Parameters

In order to derive the new line let there be n endpoints where x_i and y_i are the x and y coordinates of the i^{th} endpoint and w_i is the length of the line segment that contains endpoint i. Let the new merged line fitted to those endpoints be l(x) such that:

$$l(x) = c_0 + c_1 x \qquad (9)$$

The coefficients c_0 and c_1 have to be chosen to minimise the sum of the squared deviation of each endpoint from the line l(x).

i.e. minimise

$$e^2 = \sum_i w_i(y_i - l(x_i))^2 \tag{10}$$

or $$e^2 = \sum_i w_i(y_i - c_0 - c_1x_i)^2 \tag{11}$$

Differentiate with respect to each of the coefficients and solve:

$$\frac{\partial(e^2)}{\partial c_0} = \sum_i -2w_i(y_i - c_0 - c_1x_i) = 0 \tag{12}$$

$$\frac{\partial(e^2)}{\partial c_1} = \sum_i -2w_ix_i(y_i - c_0 - c_1x_i) = 0 \tag{13}$$

These equations may be rewritten as follows

$$\sum_i w_iy_i - c_0\sum_i w_i - c_1\sum_i w_ix_i = 0 \tag{14}$$

$$\sum_i w_iy_ix_i - c_0\sum_i w_ix_i - c_1\sum_i w_ix_i^2 = 0 \tag{15}$$

giving c_1 and c_0

$$c_1 = \frac{\sum_i w_i\sum_i w_iy_ix_i - \sum_i w_iy_i\sum_i w_ix_i}{\sum_i w_i\sum_i w_ix_i^2 - \left(\sum_i w_ix_i\right)^2} \tag{16}$$

$$c_0 = \frac{\sum_i w_ix_i^2\sum_i w_iy_i - \sum_i w_iy_ix_i\sum_i w_ix_i}{\sum_i w_i\sum_i w_ix_i^2 - \left(\sum_i w_ix_i\right)^2} \tag{17}$$

This gives the slope and intercept of the straight line fitted to those given endpoints. If the weighting chosen for each endpoint is the length of the line

segment then the line fitting process will favour the longer line and will not distort a long line with a shorter line. The formulation of the straight line given in equation 9 by c_0 and c_1 is not generally as useful as the implicit formulation given in equation 18 as both c_0 and c_1 are unbounded.

$$ax + by + c = 0 \tag{18}$$

It may be seen by comparing equation 18 with equation 9 that

$$c_0 = \frac{-c}{b} \qquad \text{and} \qquad c_1 = \frac{-a}{b}$$

Therefore from equations 16 and 17

$$a = \sum_i w_i y_i \sum_i w_i x_i - \sum_i w_i \sum_i w_i y_i x_i \tag{19}$$

$$b = \sum_i w_i \sum_i w_i x_i^2 - \left(\sum_i w_i x_i\right)^2 \tag{20}$$

$$c = \sum_i w_i y_i x_i \sum_i w_i x_i - \sum_i w_i x_i^2 \sum_i w_i y_i \tag{21}$$

As an example of this method the endpoints from the three lines given in the example in figure 5.17 are as follows, the set of weights consists of the line length corresponding to each end-point.

$x = \{1,3,5,15,17,19\}$

$y = \{4,4,3,3,2,2\}$

$w = \{2,2,10,10,2,2\}$

The summations are computed :-

$\sum_i w_i = 28$ $\qquad$ $\sum_i w_i x_i = 280.0$

$\sum_i w_i y_i = 84.0$ $\qquad$ $\sum_i w_i x_i^2 = 3820.0$

$\left(\sum_i w_i x_i\right)^2 = 78400.0 \qquad \sum_i w_i y_i x_i = 776.0$

This gives the following results for the coefficients

$c_1 = -0.06275 \qquad c_0 = 3.62745$

$a = 1792 \qquad b = 28560 \qquad c = -103600$

Therefore the gradient is -0.063 and the axis intercept is 3.63. Figure 5.19 shows this new line together with the original three line segments. It can be seen that this line provides a good match to the data that does not ignore the two shorter lines yet does not over-emphasise them.

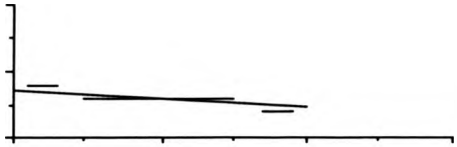


Figure 5.19 - Line fitted to line segments

The result from the weighted least squares method may be compared with the unweighted least squares fitting of the same data as shown in figure 5.20. The fitted line here has $c_0 = 4.03226$ and $c_1 = -0.10323$. In this case the results from the unweighted method are very similar to the extreme end-point algorithm.

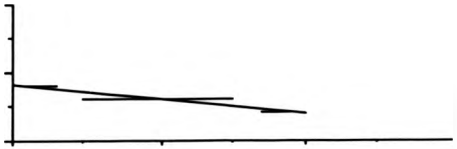


Figure 5.20 - Unweighted fitted line

Note that equation 10 defined the error as being the deviation in the y values of the line. This, while being the common way of measuring error in such circumstances, will work less well as the gradient of the line increases. This may be overcome by estimating the gradient of the line using the gradient obtained by linking the two most extreme endpoints and if this is greater than 45° then either reflect the line segments in y=x or define the error as being the deviation in x values.

Finding Endpoints

The process above gives the parameters of the new line but does not provide suitable endpoints. The line may be truncated at the x or y coordinates of the extreme endpoints or at the nearest points on the line to those endpoints. The most satisfactory and general solution is the nearest points on the line to the endpoints of the line segments as shown in figure 5.21.

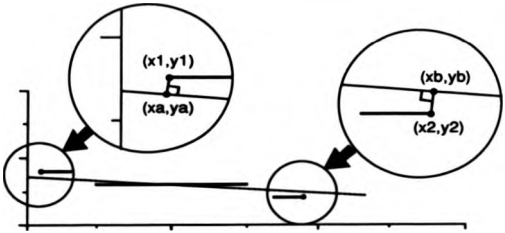


Figure 5.21 - Finding endpoints

If the most extreme endpoints of the matched set of line segments are (x_1,y_1) and (x_2,y_2) and the matched line is $ax + by + c = 0$ then the fitted line segment is bounded by (x_a,y_a) and (x_b,y_b),

$$x_a = \frac{b^2x_1 - aby_1 - ac}{a^2 + b^2}, \qquad y_a = \frac{a^2y_1 - abx_1 - bc}{a^2 + b^2}$$

$$x_b = \frac{b^2x_2 - aby_2 - ac}{a^2 + b^2} \quad , \qquad y_b = \frac{a^2y_2 - abx_2 - bc}{a^2 + b^2}$$

5.6 Further Line Processing

Having extracted line segments from the image then there are a number of simple features based on lines that may be sought, the most common of these are parallel lines and corners.

5.6.1 Parallel Lines

Using the line segment matching criteria given in figure 5.16 parallel line segments are those with a very small or zero value angular separation (θ_d) and a non-zero perpendicular separation (d). Figure 5.22 shows four examples of pairs of line segments that fit into these categories. Of these, pair (a) would probably be matched as colinear, pair (b) again might be considered colinear. Pair (c) fits into the classification of parallel but the top line is so insignificant compared with the bottom line that it might be removed. Finally pair (d) appears to completely fit the definition of parallel. It may be seen from these examples that the strict definition of parallel is not tight enough for useful work and hence even such a simple feature as this is in practice somewhat difficult to extract.

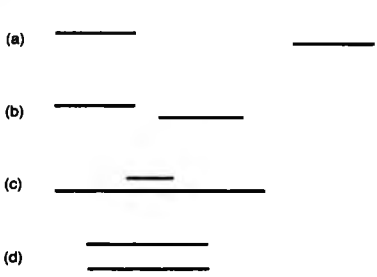


Figure 5.22 - Examples of 'parallel' lines

5.6.2 Corners

Having extracted straight lines from the image then it would seem that corners should be easy to extract. If line segments intersect within some distance of their endpoints then the intersection may be marked as a corner. In practice, however, because most edge detectors break down at corners, generally the line segments will not intersect at the corner but instead terminate somewhat short of the intersection. Therefore a corner detecting algorithm must be able to 'extend' both line segments in search of intersections. This extension however may generate false corners as intersections will be triggered with nearby line segments that do not intersect. Other corners will be generated from line segment intersections that are not in fact due to corners of real objects in the image such as those caused by occlusions.

5.6.3 Conclusions

In both the cases of parallel lines and corners a number of the features extracted will be false, this is because it is very difficult to derive a definition of exactly what is meant by a corner or parallel line that stands up to use. This is of some significance if the features are to be used for recognition purposes.

In the case of features such as straight lines there is a very low expectation of the feature being completely false. This likelihood of error in corners and parallel lines reduces somewhat the value of these features for matching purposes. Therefore even simple features such as parallel lines or corners appear to be too high level to be able to be extracted from the image without some knowledge of the image or a model.

5.7 Line Verification

For some applications (for example model based recognition) it is helpful to have some bounds on the accuracy of the lines that have been extracted from the image. Several errors may occur while extracting lines, the angle of the line may be in error and the end-points may have a translational error either due to an angular error or an error in the length of the line or a translation of the line. Each of these errors causes a different problem to a recognition task. The angular error causes an error in the orientation of the model which as will be shown in the next chapter, can cause significant errors in the spatial location of the object. The alteration in the length of the line is of low significance as this is expected by a recognition algorithm because it can be caused by obscuration and therefore the line may correctly be foreshortened. Finally a translation in the line causes a small error in the recognition process that is of little significance if it is of a limited amount. However, while the major application of line extraction covered in this thesis is recognition based on lines it must be considered that other applications exist, therefore each of these errors should be kept to a minimum.

Another consideration is that the line matching and merging process may have merged lines that should not have been merged. This could for example take place when two colinear lines with a large break are merged. Without a model of the scene it is impossible to know whether they form a single line or two separate lines. The process outlined above will merge such a pair of lines if they satisfy some conditions such as the angular separation between the two lines being small. Note that the traditional Hough

transform does not have the option of not splitting two colinear lines even if the separation is very large. At this stage however it is possible to break a longer line in the light of the now available global evidence.

The output of the line matching and merging process is a tree structure of line segments, see for example, figure 5.23. Note that each line is described by its endpoints. Each line in this tree may then be verified with the original image (or an edge detected image) to check that the line is not greatly in error, perhaps caused by some error propagated from the local stages. If a line is largely in error then the component lines (those lines below it in the tree) can be checked also to find where the error originates and the line may be broken or adjusted to reduce this error.

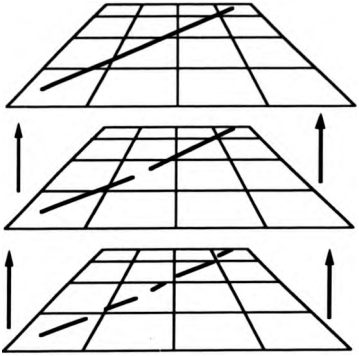


Figure 5.23 - Line Hierarchy

If $\{p_i\}_{0 \le i \le n-1}$ is a line of pixels in the image plane and u is a unit vector of the fitted line, then the error may be defined as :-

$$err^2 = \frac{\sum_{i=0}^{n-1} | (p_i - p_0) \times u |^2}{n}$$

If this value is below a certain level then the line may be split back to the next level in the tree. At the bottom level of the tree the line segments should pass this value in order to have been extracted.

5.8 SIMD Line Extraction Implementations

There has been much work in recent years on implementing the Hough transform on diverse architectures including MIMD arrays [Lotufo 89], associative arrays [Duller et al 89], systolic arrays [De Groot et al 88], linear [Fisher and Highnam 89] and mesh SIMD arrays [Silberberg 85] [Guerra and Hambrusch 89] [Rosenfeld et al 88] [Cypher et al 87] as well as dedicated hardware [LSI Logic 88] and even CORDIC processors [Timmerman 89]. The implementations on SIMD machines are of prime importance here. These further split into two major categories, those for mesh connected arrays and those for arrays with some enhanced means of communicating. This communication is so important as the Hough transform is a global operation as it is capable of detecting image-wide patterns.

5.8.1 Mesh Connected SIMD

The key papers on implementing the Hough transform on mesh connected SIMD arrays are the following. In [Silberberg 85] Silberberg presents an implementation on the GAPP, in [Guerra 87] and [Guerra 89] Guerra and Hambrusch present two algorithms for a general square mesh of SIMD processors, in [Cypher 87] Cypher, Sanz and Snyder present five new algorithms for two different architectures, one of which is a mesh. Finally

Rosenfeld et al review past methods including Silberberg's in [Rosenfeld 88]. Interestingly Rosenfeld concludes that 'a mesh-connected computer composed of bit-serial PEs is not very efficient at implementing an algorithm such as the Hough transform, which requires global shifting and summarization of data from all parts of the array'. Most of these methods involve new ways of shifting the values of ρ around the array as they all remap the parameter space onto the SIMD array. Therefore they are all limited to a discrete parameter space that is the same size as the image.

5.8.2 Enhanced SIMD networks

The two different enhanced arrays considered here are the hypercube connected Connection Machine and the Hughes/UMass Image Understanding Architecture (IUA) which has a Gated Connection Network (GCN) or coterie network to assist global operations. The Connection Machine's hypercube network allows faster accumulation of the results especially when care is taken to minimise collisions [Little 87] but still gives an ungainly and slow solution. The gated connection network of the IUA gives a more interesting implementation [Shu 88]. The network is set up in such a way that it subdivides the image into regions that match the projections used by the Hough. Then along each independent region, a count is performed using a distance doubling technique. The results accumulate at the edges of the array and they are positioned back onto the array in the appropriate places again using the GCN.

5.9 Line Extraction on WPM

5.9.1 Method

The ideas developed in the previous sections for improving the accuracy and computability of the Hough transform also provide an ideal implementation for a hierarchical associative MSIMD architecture such as the WPM. The previous section mentioned some of the many attempts to devise cunning implementations of the Hough transform on SIMD architectures - the

number of which gives some guide as to the perceived importance of both the Hough transform and the SIMD architecture.

All of these 'implementations' compute the Hough transform for the sake of the Hough transform, as if that itself were the goal. This misses the point somewhat as the parameter space itself is of little value for image processing - unlike for example the Fourier domain. The Hough transform is a tool for locating and extracting shapes - in this case straight lines, therefore the goal should be to find these straight lines and not simply to produce a parameter space. Therefore, producing a Hough parameter space and then remapping this space onto the SIMD array is only of value if it will assist the goal of finding these features. In fact remapping the parameter space onto the SIMD array can potentially be damaging, not only in the time taken to achieve this but also as it encourages the implementation to restrict the size of the parameter array to that of the SIMD array. As this is commonly taken to be the same size as the image this will enforce a parameter array to be of the same resolution as the image which is insufficient for accurate extraction of features.

Therefore the implementation discussed below will not simply produce the Hough parameter space but is an implementation intended to extract the straight lines from the image.

5.9.2 Implementation

The approach used is based on the hierarchical technique from section 5.4 using the approach to the Hough transform discussed in section 5.3.4 to extract the local straight lines. The Warwick Pyramid Machine [chapter 3] consists of a number of clusters. Each cluster consists of a 16 x 16 SIMD array, a cluster controller and a symbolic processor. The cluster provides an ideal platform for this hierarchical approach. The clusters may process their patch of the image independently of each others therefore being able to compute their local transforms in a data dependent fashion rather than being in lock-step synchronisation with each other. Having computed local transforms the

endpoints may be passed to the cluster controller as they are extracted. Once the endpoints are stored in the cluster controller then they may be accessed by the symbolic processor. The symbolic processors may then match these endpoints to generate a set of global lines.

Local Straight Line Extraction

Each cluster may be considered independently as the lines will be extracted from each 16 by 16 patch with no reference to its neighbours. As shown in section 5.3.4 the local lines may be extracted by computing the value of ρ for each given value of θ over the whole set of pixel processors.

Firstly each given value of θ is broadcast to all of the PEs and they all compute ρ using equation 3. For a 16 by 16 patch of image and a given θ there are at most 31 different values of ρ. Therefore the precision of the arithmetic required is limited to a small number of bits. The processors are not suitable for the efficient computation of sines and cosines and the fact that they are all processing the same value of θ at once, means that it is appropriate to broadcast the values of $\cos(\theta)$ and $\sin(\theta)$ to the processors as they are used.

Having computed ρ over the pixel array for the particular value of θ the ρ image is masked with the threshold image as shown in section 5.3.4. This very simple operation on the array leads to a set of labels which need to be counted in order to find the label that occurs most frequently. This may be done using the associative response of the cluster with the count response. This gives the value of ρ that corresponds to the longest straight line segment for a particular value θ. This is repeated for each value of θ resulting eventually in a ρ,θ pair corresponding to the longest line segment in that part of the image. Having found the strongest line segment its component pixels are removed from the data and the process is repeated to find the next best line segment.

As each line segment is extracted its endpoints need to be found. The pixels on the line segment may be identified by their label. Those pixels with that label are set while the others are cleared, which enables the most extreme

points to be located either by the use of the edge register or by performing maximum and minimum extraction operations on both the masked x and y coordinates using the some/none response.

Line Matching and Merging

Having extracted the endpoints of the line segments these may be then matched to each other following the method outlined in section 5.5.1. The output of the previous stage from the cluster controller is a set of matching line segment endpoints stored in the dual port memory shared between it and its symbolic processor. The symbolic processor then takes each line segment in turn and compares it with all of one of its neighbours' line segments, noting whether they match. This proceeds in parallel across the symbolic array. These matched line segments are then merged before the next stage of matching takes place. Merging is a very fast analytical operation that proceeds serially on each set of matched segments but in parallel across each set. As described in section 5.5.1 this may then be repeated with the parameters relaxed somewhat.

5.10 Results

This section shows some results of the hierarchical line extraction process on several images. Results are shown of several iterations of matching and merging as the process extends to line segments placed further apart and the parameters are relaxed. The first example is an image of a single disk. Figure 5.24 shows the short line segments extracted from each cluster independently, at this point there are 121 line segments in the image.

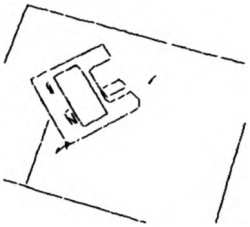


Figure 5.24 - Line segments from image 'disk1'

Figures 5.25 - 5.29 then show a succession of five images at various stages of the matching and merging process. Figure 5.25 has 78 line segments left, it can be seen that the long high quality lines have formed quite well at this stage whereas the messier parts near the disk itself are only slightly altered. The parameters for this allowed a maximum gap to line length ratio of 0.2 and an angular difference of 5° and a separation of only one pixel.

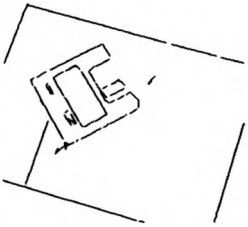


Figure 5.25 - Result of first iteration of matching and merging line segments given in figure 5.24

Figures 5.26-5.29 have 59, 50, 38 and 34 line segments respectively and involve steady increments of the parameter values used for figure 5.25.

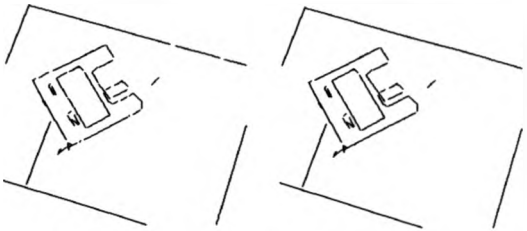


Figure 5.26 and 5.27 - Iterations 2 and 3 of 'disk1' line merging

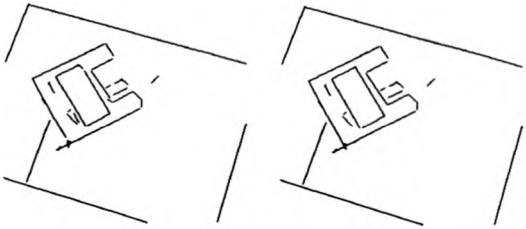


Figure 5.28 and 5.29 - Iterations 4 and 5 of 'disk1' line merging

If each line in figure 5.29 is then checked using the method described in section 5.7 the plot shown in figure 5.30 is obtained. This plot shows the error for each of the 34 line segments, the error being the average deviation along the length of the line segment in pixels.

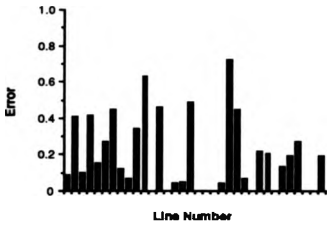


Figure 5.30 - Errors in line segments from figure 5.29

If these errors are then used to prune the image several erroneous lines are removed as shown in figure 5.31.

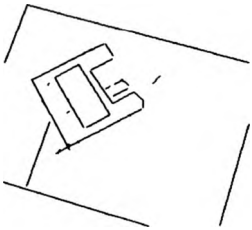


Figure 5.31 - Pruning of figure 5.30 based on error values

Figures 5.32 to 5.36 illustrate the same process for a more complex scene, here there are 225, 162, 151, 141 and 127 line segments in the images respectively.

Figure 5.32 - Line segments from image 'mac'

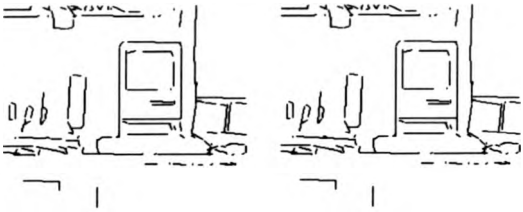


Figure 5.33 and 5.34 - Iterations 1 and 2 of 'mac' merging process

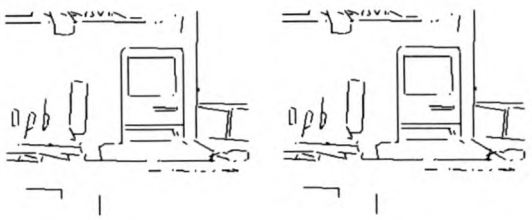


Figure 5.35 and 5.36 - Iterations 3 and 4 of 'mac' merging process

On the Warwick Pyramid Machine the extraction of line segments from a cluster takes approximately 1 ms for 32 values of θ including the extraction of the endpoints and the passing of them to the symbolic processor. If it is wished to gain the extra quality output obtained by iterating this process then this figure is multiplied by the number of line segments in a cluster. This is generally at most three or four. The line matching and merging process is very fast as the number of line segments compared is low due to the hierarchical approach taken and the reduction due to not comparing lines that are out of range.

The number of comparisons at each iteration of the disk process was 2999, 2053, 1329 and 892, these are distributed over the array of processors such that the maximum number of comparisons performed by any processor was 499, 399, 248, 155. The time a processor takes to perform a comparison depends on at which stage the comparison terminates. Only 4.6% of comparisons performed pass the check for d, 1.17% pass the comparison with θd and 0.29% pass the comparison with g. As a single comparison on a T800 Transputer takes on average approximately 48 μs this stage takes a total of 62 ms ((499+399+248+155)*48). The number of merge operations that took place were only 84. Each processor takes 260 μs to perform a merge giving a rough total for the entire match and merge stage of just over 60 ms plus 5 ms to

generate the line segments. This compares with times quoted [Rosenfeld 88] for other large scale parallel machines (GAPP and MPP) of between 130 ms and 920 ms to simply compute a 128 by 128 parameter space. This would then need to be processed and the lines and endpoints found. Even then the results on those machines are unlikely to be of the same quality as those given above.

5.11 Conclusions

This chapter has described Hough transform based methods of straight line extraction and many of their limitations that not only reduce their effectiveness but also the speed and ease of their implementation on parallel architectures. Various ways of easing or removing these limitations have been described and then it has been shown that these enhancements not only improve the algorithm but also provide a natural and efficient implementation on a machine such as the WPM.

6

Recognition

6.1 Introduction

The most important task of a recognition system is to identify certain known objects in the image. There are many applications for such a system, for example, to measure some aspect of the object, to manipulate it or to use the location of the object in order to determine camera position - navigation. In the case of navigation the object might be a composite object or a complete scene. Sometimes the location, size and orientation of each occurrence of each object in the image will be required as output along with some measure of the certainty of the match. Other times an object might be tracked as it or the camera moves. The objects that the recognition system is capable of locating must have been described in some way in advance to the system.

This description may take several forms, the most common of which are geometric and feature based models.

Geometric models take the form of two or three dimensional shape descriptions of each object to be recognised. The objects will typically be artificial and contain only simple elements such as lines and parametrically described curves. Such models might, for example, originate from a CAD package.

Feature or attribute based models are described in terms of a collection of features such as size, colour, texture, simple shape information such as aspect ratio or a set of radii lengths and moments, including perhaps some transformation-independent measures. These models can be used to describe more general and less clearly defined objects than the geometric models, such as objects that occur in nature. The choice of features used in these models tends to be determined by the processing of a large set of test images as a training set. This enables features to be found that distinguish the object from clutter. The number of features used varies considerably depending upon the application.

The best model to use in a given situation depends on the type of object to be recognised, the sensor, the environment, such information as whether the scale or orientation is known, whether the object is man-made, the speed of recognition and the accuracy required.

Occlusions and shadows can cause difficulties for both of these modelling techniques as they tend to reduce the number of features available, cause errors in some of those features that are visible and add erroneous confusing features. For example, an occlusion can obscure an important feature or block a portion of it which may cause the feature detector to believe it is a different feature. Shadows might also add new features or block or diminish existing features.

This chapter only considers the recognition of rigid objects; that is objects that remain the same shape and do not deform. Objects that do not conform to

this include, for example, articulated objects such as an anglepoise lamp, a book which may be open or closed, or an animal that may be in any number of different poses. Articulated objects may be dealt with by recognising each individual component and forcing a relationship between them, but objects such as animals are generally beyond the scope of this form of recognition.

This chapter describes the important features of key recognition systems reported in the literature and offers some analysis of the contribution of the work. For a more general review see [Grimson 90]. A new recognition system is proposed that will detect and recognise two dimensional objects, of an unknown scale in cluttered surroundings with possible occlusions. While systems for recognising two dimensional objects have been developed in the past [Tucker 88] [Cass 88] [Beveridge 89] the proposed system makes use of the data in a different way and does not make compromises that may result in missing a correct match. Unlike other systems this system allows the object to be of an unknown size, this is an essential feature for systems that do not work in a fixed environment but introduces many extra difficulties. These difficulties are discussed and methods are described that overcome them. The proposed system attempts to find the objects, verify that the hypotheses are indeed correct and then produce an accurate location, scale and orientation of each occurrence of the objects in the image plane. It is then shown that the Warwick Pyramid Machine is a suitable platform on which to base such a recognition system and some comments are made on efficiency and load balancing. While the system described here only recognises two dimensional objects, the techniques described are very general and apply to three dimensional recognition also.

The proposed recognition system uses a geometric model to describe objects. It then performs a thorough search for objects that fit this description. The method described here allows weak candidate objects to be selected that might be rejected by other systems as extensive verification of the match takes place.

Having found candidate objects it is then possible to reinforce these by the use of attributes or features that do not fall within the bounds of the geometric

modelling technique. This allows the user of the system to extract the maximum available information from the object being modelled. If attribute modelling is used alone then complex shape information is ignored, however if geometric modelling is used alone then features such as texture and colour are not used. If information is available then it is important, given the difficulty of the overall task, to give the recognition system access to as much of it as possible. If the attribute information is either not available or of little value then the system will work without, however the shape information does need to be given, although it may be incomplete.

6.2 Model Based Recognition

6.2.1 Introduction

This section gives an introduction to various methods of performing model-based recognition. Methods used by other researchers are described with some analysis and comments on the suitability of each approach and the success or failure of the techniques used.

6.2.2 Geometric model based techniques

The major distinguishing feature between different geometric model based recognition systems is the search technique used to locate the model in the image. Because the image feature space is very large most techniques involve some method of pruning the search. These methods vary in complexity and willingness to trade false alarms for accidently rejecting correct matches. If the data is of poor quality then the search space also tends to be poor, therefore the algorithm used must take great care when pruning to avoid removing correct matches. However, if low quality matches are accepted then there must be some mechanism to verify the match.

Another important difference is the set of features used by the system, these vary from very simple features such as lines through corners, corner pairs and surfaces etc. Generally as the complexity of the feature increases then the

work required by the recognition component of the system decreases and the work required by the feature extraction component increases.

Finally the number of models the system is capable of handling efficiently is an important aspect in some circumstances. Some systems e.g. [Tucker 88] have been designed to match a large number of models whereas others have been designed for only a small number.

The three most important ways of performing the search are by transformation clustering, graph searching or alignment methods.

Transformation Clustering

The most common form of geometric based recognition is by the use of transformation clustering. Transformation clustering techniques accumulate independent pieces of evidence (from individual model/data feature pairings) and then find clusters of evidence that point to a consistent solution. Transformation clustering techniques tend to be fairly resistant to occlusions and other problems and yet are able to cope with arbitrary positioning of the object.

Typically in these recognition systems the transformation required to transform each model feature to each data feature is computed. This gives a very large set of transformations. In an ideal case with perfect image features and a perfect model the transformations required to transform each model feature to the corresponding image feature will be identical. In this ideal case if there are m model features and n data features then amongst the mn transformations there will be m identical. This transformation should correspond to that required to transform the model to the occurrence of the model in the image, whereas the other transformations will have been caused by the combination of random model and image features and therefore will be random. Transformation clustering techniques attempt to find this peak and hence find both the transformation required to transform the model to the object and those image features that belong to the object.

Clearly real images and models are rarely ideal, therefore a method more robust than finding m identical transformations needs to be used to find the peak. Errors will occur in the collection of the features from the image due to noise and deficiencies in the feature extraction algorithm. In addition the model is likely not to match completely the object in the image. For example, if the model searched for is a floppy disk, the object in the image could be of a different manufacturer's disk, therefore while having the same overall shape it might have a differently shaped label or have prominent text or logos unique to that disk. Further errors can be caused by occlusion of the object by other objects, shadows thrown across the object causing variance in brightness or simply areas where there is no contrast between the object and the background. Figure 6.1 illustrates many of these points. In this image there are two disks present, one of which is occluded. The disks are of a slightly different design yet still fit the same overall description. One of the disks has a very low contrast between it and the background at the top which makes feature detectors give an incomplete set of features.

Figure 6.1 - Example Image

Given all of these reasons for having errors in the data many different algorithms have been designed to cope with imperfect data, or a transformation space that has poorly defined peaks. Two approaches [Huttenlocher 88] [Grimson 90] are generally described for finding these modes. The first is 'k-means clustering', which is merely one example of a large number of clustering algorithms that have been developed in many other areas such as economics [Duran 74], biological sciences and linguistics [Hartigan 75]. The second approach involves quantising the parameter space using a method similar to that used in some Hough transform algorithms. The generalised Hough transform [Ballard 81] is the basis for a large number of recognition systems that use transformation clustering [Thompson 87] [Silberberg 85] [Davies 89]. Here the term Hough transform is used in a generic sense to simply mean a technique that re-parameterises the data into a discrete parameter space similar to hashing.

Clustering by Quantising the Parameter Space

The transformations from a model feature to an image feature may be considered as p-dimensional vectors, where p is the number of components of the transformation. In the case of two dimensional objects with a known size this is three (rotation, and two orthogonal translation components), without a known size p is four. To recognise three dimensional objects, depending on the perspective model used and the scale information, p can be six or more.

The transformations can then be considered as points in a p-dimensional space. If this space is quantised, by quantising each of the p components of each transformation, then every point is entered into a single bin in the p-dimensional discrete array. Adding a point to a bin is achieved by incrementing the value stored in that bin. Peaks may then be located by finding which bins in this discrete parameter space contain large numbers of points. Each bin having a non-zero volume corresponds to a range of transformations. This range is proportional to the coarseness of the

quantisation, therefore the transformation corresponding to a peak in the parameter space is only known approximately. One way of computing this transformation is to make it equivalent to the transformation corresponding to the centre of the bin. An alternative method is to find all of the transformations that contributed to this bin and find their centroid to give a slightly improved, yet computationally more intense, estimate.

The technique of quantising the parameter space has a number of problems [Grimson 88] [Jain 88]. The problem facing a designer of such a system is to make the size of the bins such that the whole of a peak will be contained within a single bin. Yet to achieve this can decrease the accuracy of the resulting transformation. The major criticisms are as follows :-

1. Similar points, even if they are very close to each other, will be in different bins if they are on different sides of a quantisation boundary. This means that it is impossible to guarantee that a peak will be wholly contained within a single bin.

2. For high dimension parameter spaces the discrete table can get very large, using large amounts of memory and making the search very slow. For example, if the problem is six dimensional and each component is quantised into only 20 bins then the table has 64 million bins, probably consuming a total of 256 megabytes of storage. This is likely to be beyond the reach of most computer systems yet 20 bins per component is a very low resolution of quantisation, too low for useful applications. If, for example, there were only a few thousand transformations to be entered the search of many millions of bins for a peak is inefficient.

3. The likelihood of random peaks is high. There are many factors that influence this but mainly they are the size of the parameter table and the errors expected in the features. Grimson [Grimson 88] analyses this phenomenon in depth.

The first well known solution to (1) is to smooth the discrete parameter space so that each point will appear in several neighbouring bins. This involves a potentially very slow filtering operation to a large multi-dimensional space and has the effect of blurring the peaks being sought. The second is to compute the expected errors in the features and to plot the point in each bin it is possible it would have been in if there had been no errors. Each of these techniques increases the number of points and thus increases the chance of a random large peak.

Solving the problem of over-large parameter tables by decreasing the resolution of the quantisation of the parameter space does achieve the goal of reducing the size of the table but again increases the likelihood of error while decreasing the accuracy of the result. Another method of reducing the size of the parameter space is to reduce the dimensionality of the problem. For example, in two dimensional recognition problems based on linear features it is common not to consider the scale component but instead to allow a range of transformations. For example, if a model line segment is scaled to fit a data line segment, only a single translation is needed to describe the transformation. However, if the scale is ignored then there is a range of possible translations equal in number to difference in lengths between the two lines, this therefore gives a range of similar transformations. Therefore by ignoring the scale not only has the system been limited to recognise only objects of a given size but it has also decreased the dimensionality of the parameter table by one dimension and yet increased the number of transformation vectors entered into that table. Clearly this will increase the average number of points in each bin.

An illustration of some of these problems may be found in [Thompson 87]. Thompson and Mundy describe an algorithm that uses clustering in a discrete parameter space to find three dimensional objects. They decide that the six dimensional space is too large to search at once so it is split into a one, a two and a three dimensional space. Clustering then takes place in each of the parameter spaces in turn beginning with the two dimensional space which is

of two rotations. In the example given in their paper a model with 5 features produces approximately 20,000 transformation vectors when searched for in a given image. Each rotation is quantised to a resolution of 2°, giving a table with 32400 bins. An error of 15° is allowed in each feature which means that each transformation vector is entered into approximately 56 different bins. Therefore the total number of transformations is 1.1×10^6, this gives an average number of approximately 35 points in each bin. As a peak of only 5 points is being sought, as the model has only 5 features, a large number of false alarms can be expected. In this case the problem is not solved by increasing the resolution of the quantisation as the problem is the expected error in each feature.

Algorithms that cluster transformations by quantising them do have some immense problems to overcome. In some restricted application areas they are computationally efficient (yet memory intensive) but tend only to work in circumstances where the data is simple or with few errors. For example, in the case of the use of the two dimensional Hough transform for detecting straight lines mentioned in the previous chapter this technique may work well if the lines sought are long and few in number. The same principle applies to the use of this technique in recognition, if the clutter is low and the objects well defined it may be the best technique to use. However in the application discussed in this chapter a more robust recognition system is desired.

Cass [Cass 88] studies the problem of two dimensional recognition with a fixed scale. He notes that for each data feature there is a potential error in the orientation. If $\mathbf{d}_i$ represents the position of data feature i and $\mathbf{m}_j$ represents the position of model feature j then there is a range of orientations that will bring model feature $\mathbf{d}_i$ to within the orientation range of the data feature $\mathbf{m}_j$. If R_θ is a two dimensional rotation matrix rotating by some angle θ and $\mathbf{t}$ is a translation vector such that

$$\mathbf{d}_i = R_\theta \mathbf{m}_j + \mathbf{t}$$

Then

$$t = d_i - R_\theta m_i$$

Therefore it can be seen that each of the range of values of θ gives a differing value of t. This means that an error in the orientation will in turn cause an error in the translation component. This reduces the likelihood of the transformations clustering in the translation components when in fact the only error is in the orientation. Cass overcomes this problem by introducing the concept of 'transformation sampling'. Transformation sampling involves computing t for each of a range of possible values of θ so that each transformation bin within that range is added to. Cass also has to add a range of values of t to accommodate a difference in the scales of the features as his method assumes a fixed scale. Thus if Θ is the uncertainty in orientation, R the uncertainty in position, $\delta\theta$ is the orientation sample size, δt the translation sample size and l the average difference in model-data line segment length, then for m model features and n data features the number of different transformations considered is $nm\left(\frac{2\Theta}{\delta\theta}\right)\left(\frac{2R}{\delta t}\right)$.

Cass's method provides a significant improvement on the conventional quantised parameter space methods for it attempts to ensure that a peak will be contained within a single bin. However, this is achieved at the cost of a large increase in the number of transformations entered in the quantised parameter space without solving the other problems inherent in the quantisation method.

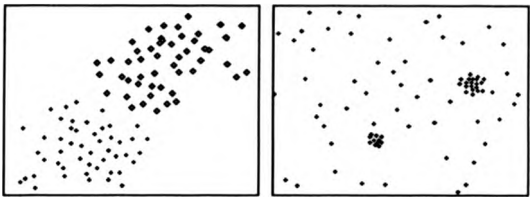
A quite different improvement on the conventional quantised parameter space method is offered by Tucker et al [Tucker 88]. In this method the parameter space is formed in a conventional fashion but the peaks are found by estimating bin density hierarchically. Initially the density of a bin is considered to be the number of transformation vectors entered into it divided by the volume of that bin. However the authors note that the size of the peaks that it works for is limited by the size of the bin as noted above. Their

solution to this problem is to estimate the density of a selection of groups of bins hierarchically and to take the density as the maximum over the selection of sizes. This has the advantage of not being tightly bound to the bin positions, however it only offers a partial easing of the problem as a sharp peak split by a bin boundary may have a low density in each and still have a low density in the enlarged bin that encompasses both bins.

Non-discrete Clustering Methods

Clustering is a device used in many fields such as economics, biological sciences and engineering. Several good reviews of clustering are available [Duran 74] [Jain 86] [Jain 88]. Clustering techniques tend to work best in specific circumstances where the algorithm chosen matches the type of data. For example, some work best where there is a small number of clusters to be found, others where the restriction is that the number of clusters sought is known in advance.

Traditionally the clustering problem is to find a number of clusters of points such that each point belongs to one and only one cluster. This has to be slightly altered in the recognition application as there is no need to place every point into a cluster and it is possible that a point may belong to more than one cluster as, for example, in the case of a feature belonging to two objects if one is occluding the other. Figure 6.2(a) illustrates the conventional clustering problem and figure 6.2(b) the transformation clustering problem. The conventional clustering problem would be to separate the two clusters, whereas the recognition problem is to find the two modes that appear in the right-hand figure.



(a) Conventional Clustering Problem *(b) Transformation Clustering*

Figure 6.2 - Examples of Clusters

The points within each cluster must satisfy some grouping constraint. A common constraint is that a point is classified as belonging to a cluster if it is within a given distance of the centroid of the cluster. However, even this example is open to interpretation as there are many possible means of computing a distance in p-dimensional space. Generally the Euclidean, Manhattan or Mahalanobis distances are used. Both the constraint and the distance metric used is critical to the problem of clustering.

If the distance between every $point_i$ and $point_j$ are considered together then the following n by n proximity matrix is obtained.

$$D = \begin{bmatrix} 0 & d_{0,1} & d_{0,2} & \cdots & d_{0,n-1} \\ d_{1,0} & 0 & d_{1,2} & \cdots & d_{1,n-1} \\ d_{2,0} & d_{2,1} & 0 & \cdots & d_{2,n-1} \\ . & . & . & \cdots & . \\ d_{n-1,0} & d_{n-1,1} & d_{n-1,2} & \cdots & 0 \end{bmatrix}$$

note that $d_{i,i} = 0$ for all i and $d_{i,j} = d_{j,i}$

Most techniques use this matrix as the basis of the clustering. Several of the most commonly used clustering algorithms are described below.

Johnson's algorithm: Johnson's algorithm is commonly used and may be found in [Duran 74] and [Jain 88]. This algorithm is one of the simplest clustering algorithms yet is effective but computationally inefficient. Initially each point is considered a cluster, then a proximity matrix is formed. The closest pair of clusters i and j are merged to form a new cluster k. Then the two rows ($d_{i,0..n-1}$ and $d_{j,0..n-1}$) and columns ($d_{0..n-1,i}$ and $d_{0..n-1,j}$) corresponding to the old clusters are removed from the proximity matrix and a new row ($d_{k,0..n-1}$) and column ($d_{0..n-1,k}$) added. This new row and column can be computed in several ways. The single-link method involves giving each new entry the minimum of the two old corresponding entries. The complete link method gives the new entry the maximum of the two old entries. A modification to this involves retaining the points that contributed to a given cluster and computing the centroid of them. This algorithm is then iterated until only a single cluster is left recording each merge operation in a dendrogram. Finally multiple clusters may be extracted by study of this structure.

K-means: K-means best works in the case where the number of clusters required (k) is known in advance. In that case k well scattered points are chosen at random from the complete set of points. These are provisionally marked as cluster centres. Then each point is assigned to its nearest cluster centre. Each cluster centre is then recomputed as the centroid of the points that have been assigned to it. This then iterates by using these new cluster centres to recompute the nearest cluster centre to each point. After several iterations the cluster centres generally become stable and a clustering is formed where every point has been assigned.

ISODATA: The ISODATA algorithm is another example of a partitional squared error clustering technique. The algorithm works as k-means but the number of clusters is not fixed, new clusters may be created or old clusters may be merged. This allows the algorithm to respond to a failing in the initial conditions such as the number of clusters not being known exactly in

advance. A cluster may be split if the within-cluster scatter is too large whereas two clusters may be merged if the between-cluster disparity is low.

Conclusions

These traditional clustering methods although used for transformation clustering [Grimson 90] are not well suited to the recognition problem. One difficulty is that the number of clusters is not known in advance. Another is that there is no need to cluster every point and a point may belong to more than one cluster.

Graph Searching

There are several ways of representing the problem of finding clusters in terms of graphs. The first discussed here involves the creation of an interpretation tree, the second involves creating a threshold graph and finding the maximum clique within it. This second method is closely related to the clustering methods described above

Interpretation Trees

An interpretation tree can be constructed as follows. At the top of the tree a starting node is created. Then a node is created for the match between the first model feature and each image feature and an edge is put between the starting node and each of these new nodes. Each of these nodes is considered in turn. An edge is put between the node being considered and a new node created for each match between the second model feature and each image feature. This is done in turn for each model feature progressing down the tree so that a given level of the tree corresponds to a given model feature. Figure 6.3 shows the interpretation tree created for a task with three model features and three image features.

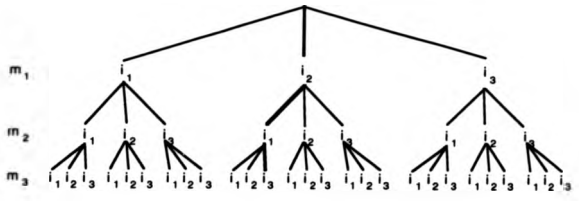


Figure 6.3 - Example Interpretation Tree

A path from the top of the tree to the bottom represents a possible set of model and image features. For example, the highlighted path in figure 6.3 represents model feature 1 matched with image feature 2, model feature 2 matched with image feature 1 and model feature 3 matched with image feature 3. Every possible combination is represented by a path, therefore there are n^m paths.

Each node has associated with it a transformation required to transform model feature *i* with image feature *j*. Each node is then marked with another transformation which represents some average of the transformations on the path above it in the tree. Each edge linking node *i,j* with node *i+1,k* is then given a weight that represents how well the transformation associated with node *i,j* matches the transformation associated with node *i+1,k* . Then the sum of edge weights along each path is calculated. The path with the lowest weighting gives the most likely set of model/image feature matches.

The obvious problem with this method is the size of the tree. If there are m model features and n image features, then at the first level of the tree there are n nodes, at the second n^2 at the m^{th} level n^m. Thus for m=5 and n=20 there are over three million nodes. It is clear that for normal problems when both m and n are generally larger than this then the size of the tree is prohibitive.

The task becomes one of pruning the interpretation tree so that the problem is feasible yet retaining enough of the tree so that the chance of pruning a correct solution is minimised.

The simplest method of doing this involves taking the best matching branch at each node of the tree below the top, simplifying the tree to size nm. This approach is computationally very efficient, but unfortunately in many circumstances it fails to work. This algorithm places too much credence on the early model features, thus unbalancing the problem in favour of matching the features near the top of the tree. At a given node there are n possible branches to choose from. The one chosen depends on how well it matches the nodes above it in the tree. This places a reliance on the first transformation being correct or nearly correct. Another problem is that correct branches can easily be missed. This can occur because the averaged transformation on the path so far is slightly deviant from correct transformation, so an incorrect branch may match better than the correct branch. This can increase the deviancy of the path.

Grimson and Perez [Grimson 87] prune the tree less extensively than this by removing everything below an inconsistent node. A path retains some averaged value of the nodes along it. A node should be consistent with all nodes on the path above it in the tree. Therefore any nodes that are outside of some error bound of the retained value of the path at that node, or the averaged value of the nodes above it in the path, can be removed along with all of the nodes below it.

A problem with a technique such as this algorithm is that it forces each model feature to match to an image feature. This is desirable when all model features appear in the image but in the case of occlusion some may well be missing. For example, in the interpretation tree an 'optimal' path is constructed from the top of the tree to the bottom. This will clearly look for a match with every model feature even in the case where some model features do not exist. What happens then is that a sub-standard match will be made by

using the best match it can find. It is possible that this sub-standard match has a very large error yet is still the best that can be found because the feature might not exist in the image. As each path is given a weighting in order to judge which path is best, then this incorrect pairing may reduce the correct path below that of an incorrect path that does not have such an extreme error within it.

Grimson and Perez sidestep this problem by allowing a null option at each node which means that if a good enough match can not be found then it is left blank and the feature is assumed missing. This overcomes this particular problem yet makes the threshold point at which a node is rejected crucial to the overall success of the task. An additional problem with their form of pruning is that a single feature that is badly in error may severely affect the chances of the whole data set matching.

An analogous problem occurs with transformation clustering but in this case the problem is one of having too many occurrences of a particular image or model feature. When clustering transformations no judgement is made about where a transformation comes from. Therefore unlike the graph matching techniques these algorithms make no attempt to use transformations from as many different model and image features as possible. This can introduce matches that consist of many duplications. This problem is discussed further in section 6.3.4.

Clustering as Graph Searching

Bolles [Bolles 82] describes an algorithm that forms a graph from the transformations where each pair of model and image features gives a node. If two points are within a given distance of each other then the nodes corresponding to those points are joined to give an undirected graph. The metric used to determine whether to link two nodes is heuristic using the following criteria.

1. The features the transformations derive from must be different.

2. The spatial (image) distance between the features must be 'approximately' the same.

3. The orientations must be 'approximately' the same.

Clearly the decisions made on these points are crucial to the success of the algorithm. The formation of this graph is analogous to having formed the proximity matrix when clustering.

Having formed this graph the algorithm then seeks to find the maximal clique. It is unlikely that the whole model would form a clique therefore this algorithm generally will match parts of the model to the image.

This algorithm is a derivative of the well known [Jain 88] technique using threshold graphs. A threshold graph is an undirected, unweighted graph with no self loops or multiple edges. An edge is added to the graph G(t) between nodes i and j if the transformations i and j have a disparity less than t.

Alignment

Alignment methods [Huttenlocher 88a,b] seek out correspondences between individual model and data features. In the case of two dimensional objects with no scale a single pairing of line segments gives the whole of the transformation. This transformation becomes an hypothesis about the pose of the object in the image space. This hypothesis then has to be checked by transforming the entire model using the estimated transformation. Having aligned the model with that data feature it then provides predictions about the positions of other data features which may be compared to the real data features to give a measure of the validity of the match. This method requires a very efficient and effective verification technique as unlike the methods outlined above, which first produce a small number of well supported hypotheses, this method produces every possible pairing of image and data features as hypotheses, each of which need to be verified in turn. However this may well work quickly if the match can be identified early on in the

search. This becomes significantly more complex if a number of features are necessary to describe a unique transformation such as a problem in which scale is incorporated.

6.2.3 Attribute Classification

Feature or attribute classification works by finding f potentially unrelated features from an object. Again these are considered as f-dimensional vectors. However, here the process of interpreting the parameter space is one of classification, not one of finding peaks. That is, given a vector (or set of attributes), the task is to compare it with a parameter space which contains volumes that are already associated with different objects.

One of the advantages of this method is that a significant amount of the processing is performed off-line by the creation of the partitioned parameter space. However, creating this parameter space is itself a significant problem. The conventional method of performing this is to process a large training set of images with many examples of every object that the system is to encounter. The operator then informs the system of the identity of each object in the training set. If the attributes have been carefully chosen then each object will have a distinct volume in the parameter space. If this is not the case then if possible extra attributes should be added that separate these volumes further. Also, attributes that do not offer any significant contribution towards distinguishing objects may be removed. This selection of attributes is critical to the success of the overall process.

Having obtained a well defined parameter space the idea behind this technique is that a new vector is created from the object of interest in the image. This new vector is then compared with the volumes in the parameter space and the one it is within gives the object.

In addition to the problem of finding a representative training set and then processing it, another significant problem with attribute classification is the segmentation of the object. In order to be able to extract the attributes that belong to an object to be able to classify it, the object needs to be separated

from the rest of the image. For large non-uniform objects this is not necessarily any easier to perform than the complete recognition process, for without any input from a model about the shape of an object it is not clear (except for simple uniform-contrast objects) how to decide which components of the image belong together as constituent parts of the same object.

Another significant problem with this method is dealing with occlusions. Occlusions might split the object making even a simple object impossible to segment correctly. Also occlusions will cause many of the conventional attributes such as moments to fail, although some such as texture or colour may still be used.

6.3 Warwick Recognition System

6.3.1 Introduction

Section 6.2 has described many of the commonly used algorithms in recognition. Also described are some of the significant contributions that have influenced the work in this section. The rest of this chapter is devoted to describing a recognition system that overcomes the problems encountered in the previous section. The recognition of two dimensional objects is addressed here with the useful addition of the scale element, while this work does not discuss the recognition of three dimensional objects from two dimensional images, the majority of the elements described here are very general and apply in full to problems of either dimensionality. The approach taken has been to develop a very high quality system that does not have heuristic rules for pruning the search space which inevitably lead to missing matches. An important element in any recognition system is the verification stage of the processing. This is because in order to detect weak occurrences of objects many false alarms may occur which have to be rejected.

The system described here clusters transformations in a non-quantised parameter space by searching for volumes of high density within it at various levels of resolution. The introduction of the scale element causes some

problems as it can cause deviation in the translation element in a similar way to that outlined above for errors in rotation. This system extends that of Cass [Cass 88] to scale and shows how some of the problems introduced by the extra transformations need not be a problem if the clustering process is applied carefully. Verification and adjustment of the final pose is computed analytically. Finally it is shown that this method is very suitable for use on a parallel architecture as the computation may be distributed and requires very little non-local communication.

6.3.2 Features Used

As described earlier in this chapter, as the features used increase in complexity then the recognition system decreases in complexity as each feature contributes more information to the matching process. For example, matching a model line segment to a potentially incomplete image line segment fixes the angle and one component of the translation. Matching a corner gives the angle and the complete translation, whereas matching a pair of corners gives the angle, translation and scale. In addition a corner will match only to other corners that enclose approximately the same angle thus reducing the number of matches. However, the more complex the feature is the more difficult it is to extract from the data and the more error prone the result is. This effectively off-loads work from the model-based system to the feature extraction system. In addition to this it is more likely that the feature is obscured and therefore more likely that a possible match will be missed. Simple features may have some error in their pose but are unlikely to be completely erroneous; more complex features, even corners, are very difficult to extract reliably and the result is that the recognition system has to cope with unreliable features. A recognition system based on line segments may be fairly sure that the feature is correct to within some degree and does correspond to a feature in the image. However, features such as corners are generally based on the intersection of lines which may well be coincidental giving rise to features that are not just slightly inaccurate but in fact completely false.

The features that have been chosen for this system are therefore straight line segments. There are several additional reasons for this. Firstly straight line segments are relatively simple to extract and some limits to their error may be devised, (cf chapter 5). Secondly straight lines appear to be a useful feature in a human-made world and are a feature from which more complex features may be built including curves. However, even though straight line segments have been chosen the recognition system described here will function with many other features, very few parts of this description are specific to line segments.

6.3.3 Computing Transformations

Having extracted the features, in this case line segments, from the image these features have to be compared with the features in the model. As described above the transformation required to transform each model line segment to each image line segment is computed. When computing the transformation great care needs to be taken to attempt to keep the transformations clustered in the parameter space.

The major problem with the computation of the transformation is to attempt to prevent any error in either the scale of the line segment or the orientation of the line segment from causing the other components of the transformation to have an additional error.

$$\mathbf{t} = \mathbf{d}_i - s\mathbf{R}_\theta\mathbf{m}_j$$

The translation component of the transformation is shown above, where $\mathbf{m}_j$ is the position of model feature j and $\mathbf{d}_i$ is the position of data feature i. It can be seen from this that any error in either the scale (s) or the orientation (R_θ) gives rise to an error in the translation (t). Therefore an error in either the scale or the orientation of the feature that causes it to spread the cluster in those dimensions also causes the cluster to spread in the translational dimension. This is a problem for the translation component should be

allowed to cluster on its own merits. It is clear that errors in the scale and orientation do not affect each other.

The problem of the rotation is addressed first as illustrated in figure 6.4. In this figure one of the data line segments has a small orientation error. Each model line segment is rotated by the appropriate angle to orientate it with the matching data line segment. This error in the data line segment orientation causes the rotated model to have the corresponding model line segment translated some way from the others.

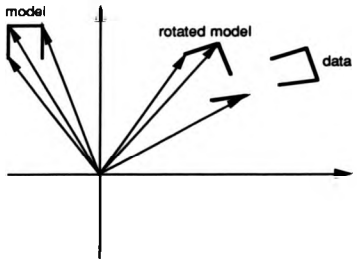


Figure 6.4 - Error in rotation component

This error is magnified by the distance of the centre of rotation from the line segment being rotated. If the centre of rotation is shifted closer to the model then the translational error resulting is reduced as shown in figure 6.5.

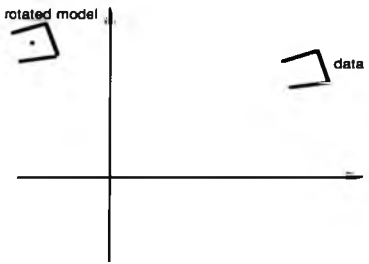


Figure 6.5 - Model rotated about its centre

The effect of rotating the model line segments by the two different angles in this case produces such a small error that it is hardly noticeable - yet it is still there. This effect is most easily achieved by maintaining the model centred on the origin. However, this technique is only partially effective in that it only reduces the problem somewhat and still passes errors through to the translation dependent upon the distance of the model line from the origin.

As described earlier in this chapter Cass [Cass 88] offers a solution to this problem by the use of transformation sampling. He uses the method to attempt to force a single bin to contain the complete cluster. In this case however no attempt is being made to quantise the parameter space so slightly differing results are needed. Initially the error in the scale is ignored and the orientation is sampled within the known error in the orientation of the line segment. The idea is that if the sampling interval is fine enough then one of the orientation samples will be very close to the real orientation and thus will pass through a minimal error to the translation.

However unlike Cass' method, the application discussed here does not need to fix the sample rate in order to place a transformation in each bin as the pose space is not quantised. In addition it does not require that the

translation be sampled as there is no attempt to unify the transformations but simply to minimise the effects of errors in one component on the other. Any error in the translation component that is due to a real error in the position may be considered on its own when clustering.

An error in the scale factor clearly also has an effect on the translation in a similar way to the rotation as shown in figure 6.6. The solution to this may be approached in the same fashion as the rotation.

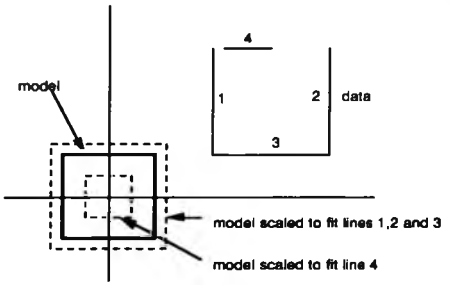


Figure 6.6 - Translational Error caused by scale

Unlike an error in rotation, an error in scale is inevitable regardless of the quality of the feature extraction process. This is because it may be caused by occlusions to the data.

The side-effect of both of these factors is to increase the number of transformations in the parameter space. This will increase the time taken to perform the clustering but, as shown in section 6.3.5, can be implemented in such a way that the clusters formed do not include multiple occurrences of the same pairing and hence the accuracy of the process is not affected.

6.3.4 Weighting the Transformations

Having computed a set of transformations the vectors need to be compared with each other. In order to achieve this, some weighting function needs to be applied to each of the components of the vectors. There are two reasons for the weights, firstly that the range of each of the components varies significantly and secondly that the importance of each component is not necessarily the same.

This second point may be seen by a study of a line picture such as figure 6.7. A small error in rotation (c) or even a large error in scale (b) leaves the figure recognisable yet a small error in translation (d) dramatically degrades the picture. If the features come from a single real object then it is likely that some obscuration or distortion can take place but random translational errors in individual features are not likely. Therefore the translational component should have a relatively high weighting. A large error in scale is tolerable and is in fact expected, so the scale component should have a very low weighting.

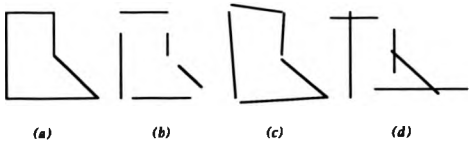


Figure 6.7 - Distortion of model (a) by scale (b), rotation (c) and translation (d)

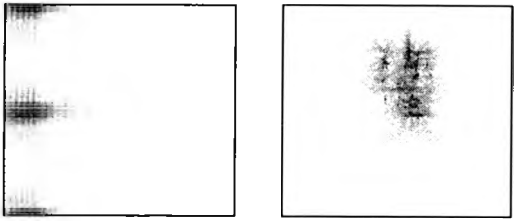
The other cause for weighting is the range of the values of each component. Rotation is limited to the range 0 to π. The scale factor varies in the range $\frac{1}{\sqrt{2}n}$ to $\sqrt{2}n$ although in practice has a much more limited range than this. The translation in theory varies from $-2n^2$ to $2n^2$ but again is much more limited than this in practice. Over a large number of test images it has been found

that 95% of scale factors are in the range $\frac{1}{\sqrt{n}}$ to $\sqrt{n}$ and 90% of translations are in the range 0 to n.

6.3.4 Transformation Clustering

In section 6.2 several non-discrete clustering algorithms were described. However none of the algorithms are particularly well suited to this type of data and the requirements from it. As has been shown the data is not of the type that is typically clustered. Another problem with most of the methods described is that they were based on heuristic clustering or pruning and could easily miss possible matches. Many ways of finding the peaks in the transformation space have been described yet none were ideally suited to the data. If the nature of the transformations is studied the problem can be understood better. Some of the features of these transformations are outlined below.

1. The parameter space is very sparse, see figure 6.8 (a) and (b). Figure 6.8(a) shows the parameter space of just the rotation and the scale whereas fig 6.8(b) shows the parameter space of the x and y components of the translation. Therefore the clusters on each figure are compressed into the two dimensions and therefore actually appear a lot more dense than is really the case.



(a) Rotation and Scale *(b) Translations*

Figure 6.8 - Example parameter space

2. It is not known in advance how many occurrences of the object are in the image or even whether or not the object is in the image. Therefore the number of clusters is not known.

3. Each transformation comes from a particular model feature - data feature pairing. In the ideal case the peak will contain one transformation from each model feature matched to a unique data feature. It is possible that some image features will appear in two clusters but ideally not twice within the same cluster.

The solution to these criteria appears to be to cluster by estimating densities of the parameter space. The clusters may also be considered as volumes of the parameter space that have peak densities, therefore the problem may be thought of as one of searching for these modal values. This density may be computed for each transformation vector by counting the number of vectors within a given radius and then dividing by the volume given by that radius. This however somewhat limits the density estimation in that it is less able to detect clusters much larger or smaller than the diameter of the region. This may be improved by considering a variety of sizes of radius and estimating the density at each scale. Then the density for that transformation vector may be marked as the largest of the set of densities given by the set of radii.

Before it is possible to find the clusters clearly some distance measure needs to be defined. Euclidean distance appears to be the most suitable as it is invariant to translation and rotation and is the most familiar. When the weighting given in the previous section is considered this may be computed as follows for a pair of transformation vectors t_i and t_j in n dimensional space.

$$d_{ij}(t_i, t_j) = \left[\sum_{k=1}^{n} w_k (t_{ki} - t_{kj})^2\right]^{1/2}$$

Therefore the clustering algorithm may be simply expressed as follows

```
for each transformation vector i
     count1 := count2 := 0
     for each transformation vector j!=i
          if ( dij < radius1) count1++;
          if ( dij < radius2) count2++;
     density1 := count1/volume1
     density2 := count2/ volume2
```

The major problem with this is caused by the additional vectors rising from the transformation sampling algorithm. This method produces transformations that cluster well for real occurrences of the model yet it also leads to a large number of transformations with very similar components resulting from each pairing of a given data and model feature, especially if the transformation sampling interval is small as it should be in order to minimise the error that the method has been introduced to minimise. This means that these transformation vectors are very likely to cluster with each other. However, if references to both the data and model features are stored with the transformation it is possible to know which features gave rise to that transformation. This means that it is possible to avoid clustering duplications of the same model/data feature pairing simply by comparing only those that originate from different pairs of features. This is a very important aspect of the algorithm as it may also be used to prevent forming a cluster that contains several occurrences of the same model feature which is clearly undesirable as ideally a cluster should contain a single occurrence of each model feature that is matched by a data feature. To prevent the cluster containing duplications of another model/data pairing a record may be made of the pairings matched so far and only the one with the minimum radius should be recorded. The result of performing this extra check when forming the cluster is that only high quality clusters are formed. Also the only detrimental effect of transformation sampling is processing speed as the number of extra vectors will now not be considered for inclusion in the clusters.

Generally the result of this is a number of clusters around each real cluster, approximately one for each sample. These clusters are very close to each other as they originate from the same set of model/data pairs and would have clustered together if it were not for the rules outlined above. A very simple stage of postprocessing may then follow to select the single cluster for each occurrence of the object.

A very simple check may then be performed to remove clusters formed from a very small number of data features. A cluster has to be allowed to have small numbers of duplications of data features within it - but not duplications of pairings. This is because a fragmented model feature may correctly match two or more data features. However matching a very small number of model features is very likely to cause errors as the smaller the number the more likely the occurrence may be random. This is especially the case for matching a single model line segment to a fragmented data line segments when in practice very little information may be gained about a model from a single fragmented data line. Therefore a simple check is to remove clusters that have a very small number of matches.

6.3.5 Consistency Verification

Introduction

If the recognition system is to function well in the cases of noisy and occluded objects then the clustering algorithm must be set to detect weak clusters. This is likely to lead to false alarms. Therefore some processing needs to be performed to check the validity of a cluster or set of correspondences. Given a set of correspondences a transformation has to be computed from the set of data line segments to the corresponding model set and an error has to be computed that is low if the data lines correspond well to the model lines. In the ideal case the transformation is simple to compute as it is simply the transformation from any one of the line segments to the corresponding model line segment. However in the case of noisy or obscured data as described above then the transformations from each model line segment in

the matched set vary slightly and a new transformation needs to be computed that best fits the set.

One solution is to take the mean or even median transformation. This, however, does not produce a very satisfactory result as generally data line segments are shorter (due to truncation) than the model line segments they match thus giving on average a reduced scale. This in turn distorts the other transformation components. Another solution is to measure the height or width of the convex hull of both the data and the model lines - this can then give an overall scale factor. However again this is prone to noise and gives poor results in cases that have rogue data line segments. An iterative approach using such techniques as Newton-Raphson has been used to solve the problem in the past [Lowe 87]. These methods can produce adequate results given a good initial estimate of the transformation however a better method would be to produce an analytical result that has no dependencies upon initial conditions.

Beveridge et al [Beveridge 89] were the first to describe an analytical method of computing the optimum transformation given a set of model-data correspondences. Their method involves minimising the squared deviation of each data line segment end point from each model line segment. Their method fixes the scale and results in a fourth order polynomial in the rotation angle. The method described here is based on the method given by Beveridge but is extended for the more general case of a varying scale and is simplified somewhat to a second order polynomial in the rotation angle despite an extra degree of complexity.

Analytical Solution

The first stage of the process is to define the error or distance that the transformed data line segment is from the corresponding model line segment. The sum of these errors gives an overall error for the transformation and hence with an optimum transformation gives an error

for the set of correspondences. As this error is being minimised then it l
key to the algorithm.

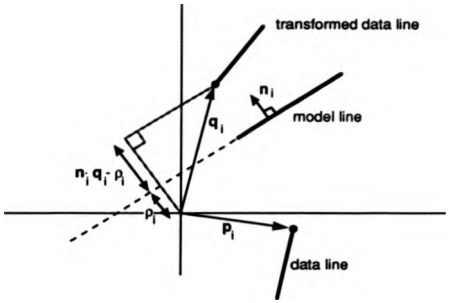


Figure 6.9 - The error measure

Figure 6.9 shows how each line segment correspondence error is meas
The error is basically the perpendicular distance from the model line seg
to the transformed data line segment endpoint. Let p_i be the data
segment endpoint i and q_i be that endpoint following a transformation.
error of this point q_i from the model line to which p_i has been matched
shown, $n_i q_i - \rho_i$ where n_i is the unit vector perpendicular to the model
and ρ_i is the distance of the model line from the origin in the direction g
by n_i. This error is computed for each endpoint of each line segment tha
a correspondence. Note that as this error measures perpendicular dist
from the model line then no account is made of any displacement paral
the model line. In the general case this is accounted for when computin
overall error by line segments of differing orientations, however in the
of a model wholly consisting of line segments of a single orientation then
technique breaks down. The solution to this uncommon problem
compute the error in two orthogonal directions as described in [Beveridge

The error associated with a given transformation/set of correspondences is the sum of the errors of each individual line pairing,

$$e = \sum_i (n_i \cdot q_i - \rho_i)^2$$

where

$$q_i = sRp_i + t$$

$$\Rightarrow \quad e = \sum_i (n_i \cdot (sRp_i + t) - \rho_i)^2$$

$$= \sum_i (s\, n_i \cdot (Rp_i) + n_i \cdot t - \rho_i)^2$$

As $p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ then Rp_i may be written as $S_i r$

where $S_i = \begin{pmatrix} x_i & -y_i \\ y_i & x_i \end{pmatrix}$ and $r = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$

Therefore

$$e = \sum_i (s\, n_i \cdot (S_i r) + n_i \cdot t - \rho_i)^2$$

First minimise with respect to the translation, t. If e is differentiated with respect to each of the components of t in turn and then the result of this is set to zero the following is obtained:

$$2\sum_i (s\, n_i \cdot (S_i r) + n_i \cdot t - \rho_i) n_i = (0\ 0)^T$$

$$\Rightarrow \quad \sum_i (s\, n_i \cdot (S_i r)) n_i + \sum_i (n_i \cdot t) n_i = \sum_i \rho_i n_i$$

$$\Rightarrow \quad \sum_i s\, n_i n_i^T (S_i r) + \sum_i n_i n_i^T t = \sum_i \rho_i n_i$$

$$\Rightarrow \quad \sum_i n_i n_i^T t = \sum_i \rho_i n_i - s\sum_i n_i n_i^T S_i r$$

Therefore let $t = a - sBr$

where $a = M^{-1}\sum_i \rho_i n_i$

$B = M^{-1}\sum_i n_i n_i^T S_i$

$M = \sum_i n_i n_i^T$

The matrix M is not singular except in the case described above when all of the line segments in the model are parallel.

Therefore substituting this translation back into the error,

$$\Rightarrow \quad e = \sum_i (s\, n_i \cdot (S_i r) + n_i \cdot (a - sBr) - \rho_i)^2$$

$$= \sum_i (s\, n_i \cdot (S_i r) + n_i \cdot a - s n_i \cdot Br - \rho_i)^2$$

$$= \sum_i (s\, n_i \cdot ((S_i - B) r) + n_i \cdot a - \rho_i)^2$$

Let $C_i = S_i - B$ then

$$e = \sum_i \left(s n_i \cdot C_i r + n_i \cdot a - \rho_i\right)^2$$

$$= s^2 \sum_i (n_i \cdot C_i r)^2 + 2s \sum_i (n_i \cdot C_i r)(n_i \cdot a - \rho_i) + \sum_i (n_i \cdot a - \rho_i)^2$$

$$= s^2 \sum_i (n_i \cdot C_i r)^2 + 2s \sum_i (n_i \cdot a - \rho_i)\left(n_i^T C_i r\right) + \sum_i (n_i \cdot a - \rho_i)^2$$

If $k = \sum_i (n_i \cdot a - \rho_i)^2$ and $x^T = \sum_i (n_i \cdot a - \rho_i)\left(n_i^T C_i\right)$ then

$$e = s^2 \sum_i (n_i \cdot C_i r)^2 + 2sx^T r + k$$

Then eliminate the scale,

$$\frac{\partial e}{\partial s} = 2s \sum_i (n_i \cdot C_i r)^2 + 2x^T r = 0$$

$$\Rightarrow \quad s \sum_i (n_i \cdot C_i r)^2 + x^T r = 0$$

$$\Rightarrow \quad e = -sx^T r + 2sx^T r + k$$

$$= sx^T r + k$$

$$= \frac{-(x^T r)(x^T r)}{\sum_i (n_i \cdot C_i r)^2} + k$$

but

$$\sum_i (n_i \cdot C_i r)^2 = \sum_i \left(n_i^T C_i r\right)\left(n_i^T C_i r\right)$$

$$= \sum_i \left(r^T C_i^T n_i\right)\left(n_i^T C_i r\right)$$

$$= \sum_i r^T \left(C_i^T n_i n_i^T C_i\right) r$$

$$= r^T W r \text{ where } W = \sum_i \left(C_i^T n_i n_i^T C_i\right)$$

$$\Rightarrow \quad e = \frac{-(x^T r)(x^T r)}{r^T W r} + k$$

Expand the vectors and re-introduce the angle of rotation gives

$$e = \frac{-x_1^2 r_1^2 - 2x_1x_2r_1r_2 - x_2^2 r_2^2}{w_1 r_1^2 + (w_2 + w_3) r_1 r_2 + w_4 r_2^2} + k$$

$$= \frac{-x_1^2\cos^2\theta - 2x_1x_2\cos\theta\sin\theta - x_2^2\sin^2\theta}{w_1\cos^2\theta + (w_2 + w_3)\cos\theta\sin\theta + w_4\sin^2\theta} + k$$

$$\Rightarrow \frac{\partial e}{\partial \theta} = \Big((w_1\cos^2\theta + (w_2 + w_3)\cos\theta\sin\theta + w_4\sin^2\theta) \cdot$$
$$(2x_1^2\sin\theta\cos\theta - 2x_1x_2(\cos^2\theta - \sin^2\theta) - 2x_2^2\sin\theta\cos\theta)$$
$$- (-x_1^2\cos^2\theta - 2x_1x_2\cos\theta\sin\theta - x_2^2\sin^2\theta) \cdot$$
$$(-2w_1\cos\theta\sin\theta + (w_2+w_3)(\cos^2\theta - \sin^2\theta) + 2w_4\sin\theta\cos\theta)) \Big) /$$
$$\Big((w_1\cos^2\theta + (w_2 + w_3)\cos\theta\sin\theta + w_4\sin^2\theta)^2 \Big)$$

$$\Rightarrow \frac{1}{2}(w_1(1 + \cos2\theta) + (w_2 + w_3)\sin2\theta + w_4(1 - \cos2\theta))$$
$$(x_1^2\sin2\theta - 2x_1x_2\cos2\theta - x_2^2\sin2\theta)$$
$$- \frac{1}{2}(-x_1^2(1 + \cos2\theta) - 2x_1x_2\sin2\theta - x_2^2(1 - \cos2\theta))$$
$$(-w_1\sin2\theta + (w_2 + w_3)\cos2\theta + w_4\sin2\theta) = 0$$

$$\Rightarrow \frac{1}{2}((w_1 - w_4)\cos2\theta + (w_2 + w_3)\sin2\theta + (w_1 + w_4))$$
$$((x_1^2 - x_2^2)\sin2\theta - 2x_1x_2\cos2\theta)$$
$$- \frac{1}{2}((x_2^2 - x_1^2)\cos2\theta - 2x_1x_2\sin2\theta - (x_1^2 + x_2^2))$$
$$((w_4 - w_1)\sin2\theta + (w_2 + w_3)\cos2\theta) = 0$$

Let $\theta = 2\varphi$

$$\Rightarrow \quad \tfrac{1}{2}(w_1 - w_4)(x_1^2 - x_2^2)\cos\theta\sin\theta - (w_1 - w_4)x_1x_2\cos^2\theta + \tfrac{1}{2}(w_2 + w_3)(x_1^2 - x_2^2)\sin^2\theta$$

$$-(w_2 + w_3)x_1x_2\sin\theta\cos\theta + \tfrac{1}{2}(w_1 + w_4)(x_1^2 - x_2^2)\sin\theta - (w_1 + w_4)x_1x_2\cos\theta$$

$$+\tfrac{1}{2}(w_4 - w_1)(x_1^2 - x_2^2)\cos\theta\sin\theta$$

$$-\tfrac{1}{2}(w_2 + w_3)(x_2^2 - x_1^2)\cos^2\theta + (w_4 - w_1)x_1x_2\sin^2\theta$$

$$+(w_2 + w_3)x_1x_2\sin\theta\cos\theta + \tfrac{1}{2}(w_4 - w_1)(x_1^2 + x_2^2)\sin\theta$$

$$+ \tfrac{1}{2}(w_2 + w_3)(x_1^2 + x_2^2)\cos\theta \quad = \quad 0$$

$$\Rightarrow \quad (w_4 - w_1)x_1x_2(\cos^2\theta + \sin^2\theta) + \tfrac{1}{2}(w_2 + w_3)(x_1^2 - x_2^2)(\sin^2\theta + \cos^2\theta)$$

$$+ \tfrac{1}{2}\Big((w_1 + w_4)(x_1^2 - x_2^2) + (w_4 - w_1)(x_1^2 + x_2^2)\Big)\sin\theta$$

$$+ \Big(\tfrac{1}{2}(w_2 + w_3)(x_1^2 + x_2^2) - (w_1 + w_4)x_1x_2\Big)\cos\theta \quad = \quad 0$$

$$\Rightarrow \quad (w_4 - w_1)x_1x_2 + \tfrac{1}{2}(w_2 + w_3)(x_1^2 - x_2^2)$$

$$+ \tfrac{1}{2}\Big((w_1 + w_4)(x_1^2 - x_2^2) + (w_4 - w_1)(x_1^2 + x_2^2)\Big)\sin\theta$$

$$+ \Big(\tfrac{1}{2}(w_2 + w_3)(x_1^2 + x_2^2) - (w_1 + w_4)x_1x_2\Big)\cos\theta \quad = \quad 0$$

$$\Rightarrow \quad k_1\cos\theta + k_3 = k_2\sin\theta$$

Where

$$k_1 = \left(\frac{1}{2}(w_2 + w_3)(x_1^2 + x_2^2) - (w_1 + w_4)x_1x_2\right)$$

$$k_2 = \frac{1}{2}\left((w_1 + w_4)(x_1^2 - x_2^2) + (w_4 - w_1)(x_1^2 + x_2^2)\right)$$

$$k_3 = (w_4 - w_1)x_1x_2 + \frac{1}{2}(w_2 + w_3)(x_1^2 - x_2^2)$$

$$\Rightarrow \quad k_1^2\cos^2\theta + 2k_1k_3\cos\theta + k_3^2 = k_2^2\sin^2\theta$$

$$\Rightarrow \quad k_1^2\cos^2\theta + 2k_1k_3\cos\theta + k_3^2 = k_2^2 - k_2^2\cos^2\theta$$

$$\Rightarrow \quad k_1^2\cos^2\theta + k_2^2\cos^2\theta + 2k_1k_3\cos\theta + k_3^2 - k_2^2 = 0$$

$$\Rightarrow \quad \left(k_1^2 + k_2^2\right)\cos^2\theta + 2k_1k_3\cos\theta + k_3^2 - k_2^2 = 0$$

$$\Rightarrow \quad \cos\theta = \frac{-2k_1k_3 \pm \sqrt{4k_1^2k_3^2 - 4\left(k_1^2 + k_2^2\right)\left(k_3^2 - k_2^2\right)}}{2\left(k_1^2 + k_2^2\right)}$$

Now k_1, k_2 and k_3 may be computed for the set of line correspondences from known data allowing the optimum rotation (θ) to be calculated. This may then be substituted back to give the optimum scale and translation. Finally the error is computed. This error gives a figure that measures the quality of the match.

A simple example of this process is shown in figures 6.10(a) and (b). The model is a square shown in the bottom left of figure 6.10(a). The data line segments are also shown on figure 6.10(a). The data line segments are of

slightly varying orientations and one is significantly shorter than the others. The square shown in figure 6.10(b) overlaid in grey on top of the data line segments shows the optimum match of the model to the data, the scale change is just over 1.5 and the rotation is approximately -8°. Visually the fit appears as good as could be achieved. The error computed from this is 0.0508789 pixels squared.

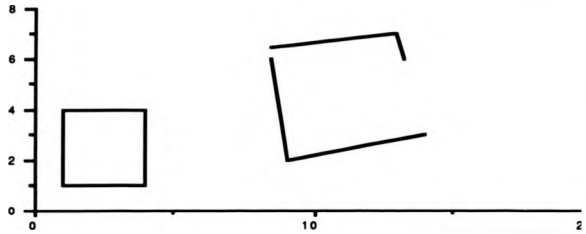


Figure 6.10(a) - Model and Data

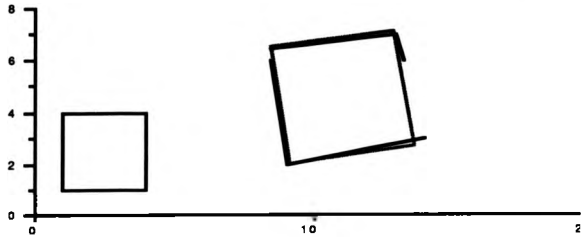


Figure 6.10(b)- Optimised Match Overlaid on Data

Attribute classification

As described in section 6.2, the most significant problem with conventional attribute classification for recognition applications is matching the attributes to the object or segmenting the object. For example, if no segmentation is available many attributes such as moments or simple shape measurements cannot be computed. Once the pose of the object in the image has been computed using the methods described above a near perfect segmentation has been obtained; this allows certain of the measures used in attribute classification to be used to verify the match rather than identify it. This also has the advantage of allowing the user to give all of the information that is available concerning the object to assist the search, which includes features such as colour and texture that have no place in the conventional geometric model. A simple example is from the disk image in figure 6.1, an attribute useful here might be a dark blob at a fixed location on the disk - such as the write protect tab or to use the fact that the sliding cover is always metallic.

An additional use for this, highlighted in the disk image is if the task were, for example, to locate a certain manufacturers disks in a collection of disks then it would be difficult to specify a geometric model that could distinguish accurately between disks and in fact this could degrade the process of recognition. An improved method would be to use the geometric based recognition process to segment the objects and then to use some attribute unique to each disk to identify the correct one. In this case the presence of a logo at a known position and orientation in the image could be used.

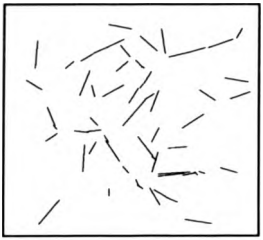
6.4 Results

The following four pages each show an example of the recognition process.

Figure 6.11(a) gives a set of data lines in which is concealed a stick picture of a giraffe. This data is taken from [Beveridge 90], in which Beveridge describes this image as being very difficult to process well. His algorithm, which greatly

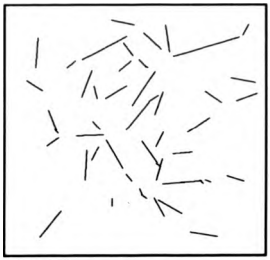
depends on initial conditions, fails to locate the giraffe in 92% of cases. This image proved to be the easiest of the four images shown here, producing a single very pronounced cluster that matched 7 of the 9 model lines perfectly. Figure 6.11(d) shows the final position of the model on the data and it can be seen that the match and final positioning is of a very high standard.

Figure 6.12 shows the image 'disk1' with a single disk which also only produces a single cluster and again matches very well. Figure 6.13 has an obscured disk and a disk that does not quite match the model. This produces a total of seven clusters of which five were very easily rejected by their huge errors (>200) or lack of feature pairs (<4). Finally figure 6.14 shows a different model, that of a Macintosh computer on a desk. Again a match is obtained with little difficulty.

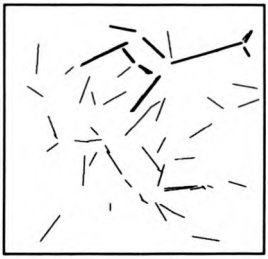


(a) Original Data

(b) Model of Giraffe



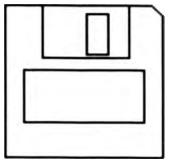
(c) Processed Lines



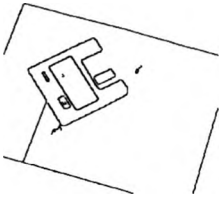
(d) Final Result

Figure 6.11 - Giraffe Image

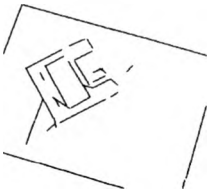
(a) Original Image



(b) Image of Model



(c) Thresholded Edge Image



(d) Straight Lines

13555 Transformation vectors
16 Lines in model
32 Lines in data

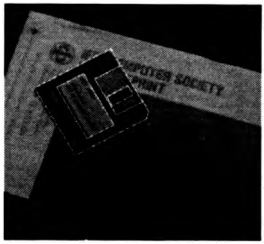
1 cluster, 7 model/data matches

Centroid
s=1.08, θ=0.97, x=85.9, y=103.3

Matching error 21.6 pixels^2

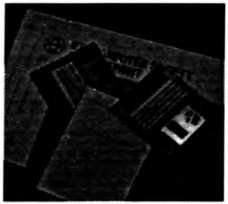
Adjusted transformation
s=0.988, θ=-1.06, x=-90.1, y=60

(g) Match Statistics

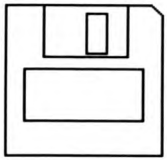


(f) Final Model Position

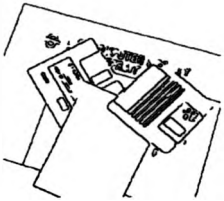
Figure 6.12 - Disk1



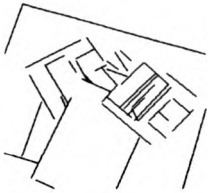
(a) Original Image



(b) Image of Model



(c) Thresholded Edge Image

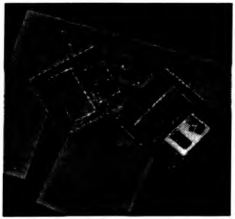


(d) Straight Lines

17927 Transformation vectors
16 lines in model, 43 in data
7 clusters, 5 rejected
Centroids
s=0.97, q=-1.07, x=-87.7, y=68.7
s=-0.99, q=0.84, x=63.6, y=269.2

Matching errors
14.1 and 26.7 pixels^2
Adjusted transformations
s=0.974, q=-1.07, x=-87.7, y=69
s=-0.989, q=0.833, x=64.3, y=268

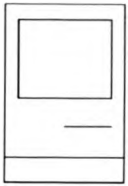
(g) Match Statistics



(f) Final Model Positions

Figure 6.13 - Disk3

(a) Original Image



(b) Image of Model

(c) Thresholded Edge Image

(d) Straight Lines

22573 Transformation vectors
12 Lines in model
64 Lines in data

2 clusters, 9 and 3 matches

Centroid
s=2.7, θ=0.004, x=-329, y=-118

Matching error 112 pixels^2

Adjusted transformation
s=2.7, θ=0.004, x=-329, y=-118

(g) Match Statistics

(f) Final Model Position

Figure 6.12 - MacPlus

6.5 Implementation

Having described the details of the recognition system, this section will detail the implementation of it on the Warwick Pyramid Machine. All of this processing takes place on the MIMD symbolic layer and therefore the implementation is suitable for any mesh connected MIMD array of processors.

The input to the recognition system is a set of line segments to each of the symbolic processors from the processing described in the previous chapter, and one or more models as defined by the user. For the images considered here the number of data line segments varies from zero to five or six on each of the symbolic processors giving of the order of a hundred or so line segments across the whole image. The model is usually of the order of ten line segments.

The first task is to compute the transformation vectors. As the model contains very little data it is simple, and not inefficient, to broadcast the model to each processor in the symbolic array. Then each symbolic processor computes the range of transformations for each of its set of line segments to each line segment of the model. If a maximum error in orientation of $\pm\Theta$ radians is allowed and the resolution of the quantisation of the orientation is $\delta\theta$ then a total of $2\Theta/\delta\theta$ orientations are used for each pairing of line segments. If the same process is applied to the scale factor with an error of a total of $\pm S$ and a resolution of δs and if there are m model line segments a total of $4\Theta Sm/\delta\theta\delta s$ transformations are computed for each of the set of line segments stored in each processor.

The location of each line segment is of no importance to the calculation of the transformations as no communication is required, therefore the load may be somewhat balanced by distributing the line segments evenly over the array. However, as this part of the processing consumes very little time this distribution must be very quick or it is not worthwhile to perform. A simple step such as, 'if the number of line segments on any processor exceeds some

value then pass some of them on to any neighbours with fewer', is appropriate.

Having computed the transformations, they are stored along with a unique reference to the model and data line segments from which the transformation is derived. The next task is to compare the transformations with themselves in order to compute the regions of the pose space with a high density. Naively, this involves comparing each transformation vector with every other transformation vector and hence requires massive random global communications. In order to reduce this load, the transformations may be distributed over the array of processors in such a way as to keep the communications local and prevent having to compare the transformation vectors with all other vectors. This load balancing can dramatically reduce the amount of computation and hence it is possible to justify performing a reasonable amount of processing to achieve this.

The two most heavily weighted components of the transformation vectors are used as keys for a two dimensional bin sort which results in the vectors being distributed across the array so that related transformations are likely to end up in the same processor. Then, if each of the radii used for the clustering process is less than the range of the transformations covered by a single bin, the communications may be confined to nearest neighbour communications. This is because it is guaranteed that no processor further away than the neighbouring processors can contain a transformation within the radius used for clustering. This condition is generally easily satisfied when using an 8 by 8 array of processors, and hence an 8 by 8 bin sort, as the radii used for clustering can be very small compared to an eighth of the range of the transformation component because of the transformation sampling technique.

The cost of this load balancing operation is simply the cost of sorting the data into the bins. The sort over an array of parallel processors is simply an operation to bin the transformations into a set of bins, which takes O(n) time for n transformations. In most cases this cost is low compared to the gains

made from restricting the communications to nearest neighbour. To perform this sort operation the transformation vectors are moved across the array until they pass over the correct column and then moved down that column until they are also in the correct row.

The example used in the following pages is from the 'mac' image of figure 6.14 which results in 22571 transformation vectors. If each was to be compared with all of the others this would require 255 million transformation comparisons. Figure 6.15 displays this example split over an 8 by 8 array using the two translational components as keys for the sort. In this example each of the components is quantised with equally spaced partitions. The number of vectors in each bin after redistribution is shown in figure 6.16 with the row and column sums shown to the right and below. As each processor has to compare all of its vectors with those vectors in its bin and in the bins to the right, below and below right then from figure 6.16 the total number of comparisons may be computed. The value for this equally spaced partition case is 138 million, which therefore has reduced the computation by 117 million comparisons.

If figure 6.16 is studied further it may be computed that the maximum number of comparisons performed by any one processor is 37 million. This leads to an additional measure which is the total number of comparisons divided by the largest number of comparisons performed by any processor. The total number of comparisons is what would be performed by a serial machine using the same partitioning technique. If all processors have to wait for the slowest to finish then this figure gives the speed-up obtained by using n processors. In this case where n = 64 the speed-up is 138/37 = 3.7.

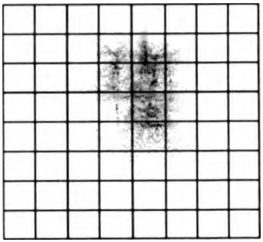


Figure 6.15 - Histogram of the two translational components of the vectors split over an 8 by 8 array

15	16	4	11	28	11	8	13	106
0	13	65	1130	2644	563	15	0	4430
0	0	99	2152	4386	946	0	0	7583
0	0	45	1157	4543	839	6	0	6590
0	0	40	363	2430	521	5	0	3359
0	22	27	127	122	106	17	0	421
0	23	7	0	2	17	9	0	58
15	0	0	0	0	0	0	10	25
30	74	287	4940	14155	3003	60	23	

Figure 6.16 - Distribution of vectors over the array given in figure 6.15

The ideal is the case where the vectors are evenly distributed over the array so that each bin has exactly the same number of vectors within it and hence each

processor performs the same amount of computation. In this ideal case the number of comparisons would be 24 million with a speed-up of 56. The ideal speed up is only 56 rather than 64 as the processors on the right hand column and the bottom row perform less computation than the other processors. This ideal is unlikely to be achieved for it is unlikely that an even distribution can be obtained.

The space may be partitioned in a different way as shown in figures 6.17 and 6.18. This partitioning, which has unequally spaced partitions, gives a total number of comparisons of 25 million and the maximum performed by any one processor at 860,000. This represents a speed up of 29 over a single processor with this enhanced partitioning and a speed up of 43 over the partitioning given in the previous example. This shows that the partitioning is very important as this has reduced the maximum number of comparisons performed by any processor from 37 million in figure 6.15 to 860 thousand in figure 6.17. Note that with no partitioning and a single processor the number of comparisons was 255 million.

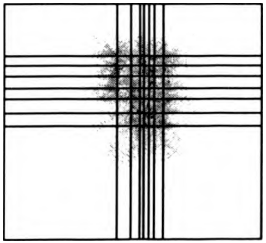


Figure 6.17- Histogram of the two translational components of the vectors more evenly split over an 8 by 8 array

477	316	463	518	420	171	81	278	2724
404	352	398	396	463	276	288	271	2848
186	242	281	368	551	412	397	334	2771
611	440	320	230	260	315	310	281	2767
309	352	358	262	302	420	530	308	2841
166	347	405	483	246	371	431	370	2819
288	397	281	230	242	443	513	539	2933
309	365	341	334	325	416	259	520	2869
2750	2811	2847	2821	2809	2824	2809	2901	

Figure 6.18 - Distribution of vectors over the array given in figure 6.17

It can be seen therefore that careful computation of the positions of the partitions is essential in order to achieve good performance and also to achieve the best use of the 64 processors. The partitioning shown above was computed by considering the two dimensional slice as a two dimensional histogram and performing a form of histogram equalisation on it. In order to maintain the locality of the data references the bins must remain rectangular. This is achieved by equalising the histogram in each of the two dimensions independently, i.e. first by attempting to position the horizontal partitions such that each column of bins has the same number of vectors and then positioning the vertical partitions such that each row of bins also has the same number of vectors. Figures 6.18 shows the sums of the rows and columns which are remarkably more even than those given in figure 6.16.

Having distributed the vectors as evenly as possible each processor performs the comparisons and isolates those vectors that are a part of dense regions. These dense regions give a set of correspondences which may be used to

compute an optimal fit for those correspondences and hence a measure of how well the match fits the data.

Finally for matches that fit the model well further checking may take place by instructing the SIMD array to search in given locations for attributes or features that should be present, such as the write-protect tab on the disk model or maybe a logo on the Macintosh. At this stage of processing the absence of these features probably should not cause the outright rejection of the match as this may have been caused by occlusion but instead it should be considered as giving a measure of certainty.

6.6 Performance Issues

Performance issues have already been touched upon in the previous section when discussing load balancing. This section gives more details of the actual performance of the method.

The first part of the task is to generate the transformation vectors. For each model/data line segment pairing stored on each processor the following steps are taken. Firstly the scale and orientation disparity is computed, then this is adjusted by $\delta\theta$ and δs for each of the range of orientations and scales used. This gives a set of slightly different paired scales and orientations. The translation components are then measured by transforming the model line segment using each orientation and scale pair and finding the translation needed to match the data line segment exactly.

Typically, approximately 25 transformation vectors are produced for each model/data line segment pairing. The computation of these vectors requires a small number of arithmetic and trigonometric operations. The time taken for the 25 vectors on a floating point Transputer is about 800μs and hence an example with 12 model lines and 5 data lines on a given symbolic processor takes 48 milliseconds. A more modern processor such as the Intel i860 takes about a third of this time.

The first stage of the distribution process involves the vectors being sorted into equally spaced bins. This involves each vector being passed along the columns until its x translational component is within the range covered by that column and then once all vectors are in the correct columns passing them up and down that column until they are in the correct row. On an 8 by 8 array a vector will on average have to pass along 4 processors in each direction. The 'mac' image shown in the previous section has 22571 vectors. The process of distribution for this image takes about 50 milliseconds.

Once the vectors are in the correct bins they are then bin-sorted on each into a further 8 by 8 bins which gives over the whole array a 64 by 64 histogram of their concentration. This takes less time than the previous sort as no communication is required and the number of elements being sorted is much reduced as each processor performs an independent sort. The number of vectors required in each row and column to give an equal distribution is known - simply the total number of vectors divided by 8. The columns of the histogram are summed from the left until this number is reached. Any columns remaining are then passed to the processor to the right, or if more columns of vectors are needed to reach this total they are fetched from the processor on the right. This operation is only performed by 8 of the processors at once but is fairly simple and only requires the communication of a small number of vectors at once over 8 communications channels in parallel. This is repeated for each row to give a reasonably even distribution. This complete sorting process for the example given above takes approximately 100 milliseconds.

This example showed that the maximum number of comparisons performed was 860,000. Each comparison involves only a small number of arithmetic operations and runs on a floating point Transputer in approximately 35μs. This gives a total time of about 30 seconds. This emphasises the value of performing the sort operation which took only 100 milliseconds to save hundreds of seconds.

Having computed the set of correspondences the final consistency checking is very fast taking less than a millisecond as it is completely analytical and is only applied to a very small number of data. Hence the complete recognition process runs in about 30 seconds on an 8 by 8 array of floating point Transputers gaining about a factor of 30 improvement over a single processor performing the same task.

6.7 Conclusions

This chapter has described many of the techniques used for the recognition of rigid objects by other researchers. A technique is described that addresses many of the problems encountered by these researchers and is extended to the detection of two dimensional objects that may be of an unknown scale. It has been shown that this extension introduces many additional complications which are also addressed in novel ways. A novel analytical method of verification and position adjustment is given that greatly enhances the accuracy of the algorithm.

An implementation and means of load balancing this application are given for the symbolic processing layer of the Warwick Pyramid Machine with some measure of the performance of both the algorithm and the implementation of the algorithm on the architecture.

7

Performance Analysis of WPM

7.1 Introduction

The previous three chapters have described the implementation of a large range of image analysis tasks on the Warwick Pyramid Machine. In addition to this some measures of the performance of each of these tasks were given. While this set of tasks does not constitute a complete benchmark, it does represent a sufficient range of problems, including two very different complete image analysis tasks, to give some insights into the performance of the machine.

This chapter analyses the features of WPM and describes which of these features were actually useful in the implementation of this set of tasks. An

analysis is made of the estimated performance of each of these tasks on the architecture if these features were not present. Also some features that were not included in the architecture are examined to see if they would make a useful contribution.

The architectural features that come under consideration are the communications networks, the nature of the associative response including 'count' and 'some/none', the inclusion of a multiplier and locally addressable memory to the iconic array, the need for some additional processing at the intermediate level, the nature of the processing at the symbolic level and the need for such a pipelined architecture.

7.2 Performance Summary

Firstly a summary of the performance of the tasks described in the previous three chapters will be given. The tasks described in chapter 4 as iconic tasks are Sobel edge detection, convolution, median filtering, connected component labelling, hysteresis thresholding, and edge thinning. The performance of each of these algorithms is summarised in table 7.1. This table shows that the performance of all of these tasks is well within the inter-frame period. Therefore, if this performance could be matched by the other components of the architecture, then WPM would be able to perform fairly complex image analysis at frame rate.

Operation		Time
Sobel		24.4μs
Convolution	(3 x 3 non-separable)	110μs
	(7 x 7 separable)	200μs
Median	(3 x 3 non-separable)	247μs
	(7 x 7 non-separable)	1.66 ms
Connected Component		2.6 ms
Hysteresis Thresholding		1 ms
Edge Thinning	(binary)	63μs
	(orientated)	54μs

Table 7.1 - Performance of Iconic Operations

Table 7.2 gives the performance figures for the intermediate, translational tasks. These tasks tend to be much more data dependent than the iconic tasks and hence the performance figures shown are for the specific examples given in chapters 4 and 5 and should be considered as representative rather than exact figures. This table includes the size filter example, which executes in under two milliseconds. This task, when combined with the segmenter, convex hull and edge detection, gives a processing flow for extracting blob-like objects of a variety of given sizes. It may be seen that this processing flow does in fact execute faster than frame rate.

Table 7.3, which describes the performance of the symbolic layer, shows that this layer does not perform nearly as well as the other two layers of the machine. This can also be deduced by studying problems such as certain implementations of the connected component labelling and the histogram generation. Both of these make use of the symbolic layer to process the

values produced by the clusters to form global properties. In each of these cases the performance of the algorithm is reduced by a factor of at least two by this, fairly small, part of the processing.

Operation	Time
Size Filter	1.7 ms
Histogram	90μs + 1 ms
Radii Segmenter	460μs
Convex Hull	639μs
Selection of Shape based Features	~1μs/shape/feature
Extraction of Line Segments	~1 ms/iteration

Table 7.2 - Performance of Intermediate Level Operations

Operation	Time
Vector Production	48 ms
Bin-sort of Vectors	100 ms
Vector Comparison (860,000)	30 s

Table 7.3 - Symbolic Layer Performance

It was stated in chapter 2 that there is very little agreement on the requirements of the symbolic layer of processing, this is partially because very little symbolic processing has actually been mapped onto parallel machines in the past and partially because there is no agreement on the methods to use. Chapter 6 only contains a single example of symbolic processing, and

therefore care does have to be taken that this might not be completely representative of algorithms used for this application. Bearing this in mind, however, this example does appear to show that the very coarse grained MIMD array of processors currently used is not adequate for the task and is certainly not well matched to the processing performed by the other layers of the machine. This is important for there is little point in having two layers of processing that produces features so rapidly and efficiently when they are then going to sit idle for so long waiting for the symbolic layer to catch up. Clearly this represents a mismatch that should either be solved by devoting fewer resources to the iconic processing or better matched resources to the symbolic layer.

7.3 Analysis of Architectural Features

7.3.1 Introduction

This section describes many important features of WPM including some that were not incorporated in the design of the machine for a number of reasons. An attempt is made to analyse the value of each of these features by studying the number of applications in which the feature has proved valuable and in what way this has occurred. The contribution that it has made is balanced against the cost of the additional hardware complexity required for that feature and the impact caused by its removal. The contribution of the features is considered individually in most cases, as if the alteration of several are considered at once then the nature of the machine can be changed. If that occurs then basing the discussion on the performance of the algorithms used becomes less valid, as in practice very different solutions are used on even subtly different parallel architectures.

7.3.2 Intra-Layer Communications

The means of communication available between processors is bit-serial, nearest neighbour, at both the iconic and symbolic levels. Two important issues are whether this is a useful inter-connection network and whether it offers sufficient bandwidth.

This network appears to match the early iconic operations such as Sobel or convolution very well and is nearly the optimal network for this type of operation, giving very little overhead. The fact that it only provides single bit communication is not a problem when connecting bit serial processors. Of the iconic tasks described here only the connected component labelling operation makes any use of global communications. It is not apparent how the addition of any other point to point communications mechanisms could be of value to any of these tasks except for a very small gain by having 8 way connected neighbourhoods. Connected component labelling requires that the labels are passed through neighbour connections in order to ensure connectivity and is not benefited by enhanced communications.

However, the addition of a different form of communication could be of some value to certain iconic processing. It has already been shown that a gated connection network, which retains the advantages of the mesh, can dramatically improve the performance of the connected component labelling operation when combined with the associative response [Shu 90]. This form of network can also be used to assist the extraction of straight lines [Shu 88].

Such intermediate algorithms as the accumulation of histograms or the matching and merging of line segments that make use of the symbolic layer would only slightly benefit by improving the communications at this layer since in these cases the amount of data to communicate is low, as is the distance it is necessary to pass that data.

The symbolic layer uses global communications to sort the transformation vectors and then uses nearest neighbour communications to compare them.

The sort operation could be enhanced by a better, higher bandwidth network, but this operation does not take a significant amount of the time taken by the whole process when compared with the computation. The comparison with vectors stored in neighbouring processors has a low communication to computation ratio and thus again is not significantly hampered by the network.

In summary therefore, the communications networks as currently proposed match the requirements of these algorithms very well and based on these results there is no justification for improving the network.

7.3.3 Inter-Layer Communications

Iconic - Intermediate

The iconic to intermediate communication is performed by two associative response mechanisms, 'some/none' and 'count'. These associative responses have both been used extensively in many examples in this work and have been the exclusive means of extracting data from the iconic array. The some/none response was used most widely, often for the extraction of maxima. The count was used less widely but still made a significant contribution to several algorithms.

The implications of not having these features is difficult to analyse. This is because if either were not present, or even were a little slower in operation, then completely different algorithms would be used to perform these tasks. For example, if the count response were significantly slower then the histogram would be accumulated on the iconic processors using a distance doubling technique such as that used on the DAP [Reddaway 83]. If this response were removed then it would have to be replaced with an alternative as clearly there must be some mechanism to extract data from the array. The alternative would probably either take the form of allowing the controller to read from a single processor at once - like a memory cell, or be something like

dual ported memory allowing access to a single bit from a number of processors. Both of these introduce a very different model of processing.

If, for example, the perimeter, area or maxima is required, with the associative responses these are found very simply. Without these the image has to be processed in other ways, either by extracting the whole image and operating on it using a serial processor, or by iconic accumulation methods for which efficient implementations tend to be difficult, and sometimes impossible, to devise.

Therefore these associative responses have been found to be excellent ways of extracting local properties from the iconic array. The only enhancement of any significance would be to devise some way of collecting the responses globally as this is not performed efficiently by the symbolic layer, and is also not the purpose of that layer.

Intermediate-Symbolic

The connection between the intermediate layer and the symbolic layer is made using dual ported memory. The type of data that has to pass through this is, for example, centroids and radii of objects, line segment endpoints and measures of certainty, in addition to control instructions passing from the symbolic processor to the cluster controller. These types of data do not require a high bandwidth connection and the communication is simply a sharing of data between the controller and symbolic processor and not a part of a larger network. Hence this need could be met by many methods of connecting processors but the dual port memory is a simple and cost-effective means of doing this that performs perfectly adequately.

7.3.4 Symbolic Layer

The symbolic layer has been shown to be something of a bottleneck. Firstly, the imbalance in processing between the time it takes to extract the features and the time taken to process these to find the objects is very significant.

Secondly, a number of the iconic and intermediate level processing algorithms make use of the symbolic layer to perform global matching and to perform some manipulation of the data before passing it back to the original layer. However this processing is not performed well by the symbolic layer as it has not been designed for this and is only used because of a lack of alternatives for performing this global manipulation. Hence there is a possible requirement for some additional hardware to carry out this task.

The first criticism here is of a mismatch between the iconic and symbolic processing layers. The lack of performance of the symbolic array means that the architecture cannot be efficiently used in a pipeline fashion with the symbolic array processing one image when the iconic array has started processing the next, as it was intended. This mode of operation means that the iconic array remains idle for most of the time.

If the problem of symbolic processing is re-examined with the knowledge gained from the application considered in chapter 6, it can be seen that there is a significant amount of data to be processed at the symbolic layer. It may also be seen that the data is coarse grained, generally involving floating point operations. It appears that the operations are generally SIMD in nature rather than MIMD, as the parallelism is gained from data parallelism and there is little control deviation. This does not match very well with the basic assumptions made in the design process and perhaps goes some way to explaining the mismatch.

Having noted this, consider mapping the recognition problem onto an SIMD machine. This gives a different set of problems. Consider a bit-serial mesh-connected SIMD array. If a similar solution to that given in chapter 6 is used then each processor is given a vector which it must compare with vectors that may be stored some distance away. The load balancing methods used in chapter 6 are significantly more complex to implement on such a machine and the loading of the data into the appropriate places in the array becomes a difficult problem.

A much simpler method may be used to implement this task that forces each vector to be compared with every other vector. In this case if each processor stores a vector and a copy of the vectors is passed over the array then the communication problems are nearly eliminated and the task again becomes computation bound. The DAP processors used in the WPM take about 1225μs to perform each step of this process. The operation iterates over all of the vectors. As in the example used in the previous chapter there are 23571 vectors a bit serial SIMD machine with one processor per vector will take about 29 seconds to perform the operation. This figure is almost exactly the same as the figure given by the MIMD array. Note however that in this case the SIMD array is performing the full range of comparisons, unlike the MIMD array, and also a large amount of the array lies idle. Despite this and the fact that the operation is very floating point intensive the SIMD array of bit serial processors can achieve similar performance to the MIMD array. This is very interesting and shows that given sufficient data parallelism the bit serial SIMD array achieves very good performance despite the technology gap between it and the MIMD array.

While it may be seen that a bit serial SIMD array is adequate for this task an additional problem becomes apparent, namely that of the format of the data and how it is loaded. Given a set of transformation vectors such that there is one per processing element the SIMD array can compete with the MIMD array. However the data does not exist in this form and there are significant problems in translating it into this form efficiently. It is possible that if this could be alleviated with some additional hardware for loading the vectors into the appropriate places in the array, then the symbolic processing could be performed on the iconic array of processors with a slightly enhanced controller to perform the additional role.

The conclusions drawn from this are that this symbolic task can be more efficiently processed by the current bit serial SIMD array than the higher technology MIMD array currently used. The only difficulty with this is the mapping and reorganisation of the data between each step of the algorithm.

This advantage may be further clarified if the performance of the SIMD array is enhanced by either improving the technology used for the array to match that used by the MIMD component or by increasing the performance of an individual SIMD processor. An alternative is to attempt to improve the performance of the MIMD array by either increasing the number of processors - as clearly the parallelism is available in the data, or by increasing the performance of the MIMD processor used. Note that these solutions are very similar, as the MIMD array would be effectively running in a coarse grained SIMD mode.

7.3.5 Iconic Granularity

An important argument for SIMD processing is to keep the processors as simple as possible in order to maximise the processing to control ratio. However, despite this, as SIMD arrays, such as the DAP, CLIP and the Connection Machine are being used for more and more arithmetic processing their designers are finding that adding extra hardware such as local addressing and multipliers to their arrays is an efficient path to follow.

Local Addressing

Giving the SIMD processors some local addressing autonomy allows those algorithms that have a single control flow but some data dependency to operate much more efficiently on the SIMD array. This assists several of the algorithms described in chapter 4. The first stages of the histogram are generated by performing two four-to-sixteen decode operations. With local addressing the decode becomes a much faster single step operation. The median filter in its basic form can make great use of local addressing as at each branch of the operation the same processing flow is applied to different data, dependent upon the result of a comparison. This can give a factor of three improvement in performance. However on the more optimised algorithm, because it has had the amount of data dependent branching reduced, this performance increase is only by a small factor. This small factor is all that may be achieved on other optimised algorithms such as connected

component labelling. A further contribution made by local addressing is for the normalisation of floating point numbers which is an operation that significantly degrades the floating point performance of SIMD arrays.

This provides an interesting insight into the optimisation of SIMD algorithms as the more optimised the algorithm the less gains there are available to be made from local addressing. Local addressing autonomy increases the complexity of the SIMD array significantly in some cases as it generally requires that the memory is kept on-chip. The case for giving some local addressing autonomy depends somewhat on other factors. If some internal memory is available then it might be worth the extra complexity. This is especially the case where the code used on the array is not heavily optimised, as when, for example, it is programmed in a high level language. This would mean that greater performance gains could be obtained than when the machine is programmed in assembly language.

Multiplier

The addition of a multiplier to the iconic level may be achieved in one of several ways governed by how the data words supplied to the multiplier are obtained. The two options are either to get the multiple bit word from a number of processors and or to get a word from each processor. The first option is used by the Connection Machine [TMC 88] which groups a number of bit serial processors to a floating point unit and allows single cycle performance but forces the data to be stored across the processors which may require a remapping of the data. The second option is used by the DAP which takes the data from a processor bit serially by supplying a multiplier to each processor rather than one for every sixteen.

Of the iconic tasks described in chapter 4 only convolution makes use of multiplications and these multiplications are by a constant and hence are not as inefficient as would otherwise be the case. The line extraction algorithm uses multiplications and would be enhanced by the addition of a multiplier, but again the multiplications are by a constant. There is probably not enough

evidence to justify the addition of a multiplier solely for iconic processing purposes and manufacturers are probably adding multipliers to their machines in order to penetrate scientific processing markets.

However if any symbolic processing were to be performed on the SIMD array rather than an MIMD array as suggested in the previous section it may well be the case that a multiplier should be considered as this processing often involves floating point arithmetic.

7.3.6 Multiple Controllers

The use of multiple controllers has several advantages that are independent of performance. Splitting the large SIMD array into many independent sub-arrays removes the problem of passing control to each processor from a single global controller and also makes the machine more modular and scalable. However it is also interesting to see if any performance gains are made from the multiple control paths.

These gains can be made in several distinct ways. The first and easiest to use and understand is where each patch of the SIMD array runs the same program yet wanders off in a data dependent fashion. Secondly, at the later stages of iconic processing, once different areas of the image have been separated and marked as having different characteristics, there is intuitively some value in being able to run different algorithms on different parts of the image dependent upon the contents. For example, if the image has been split into say a road region, a natural vegetation region and some buildings, then intuitively it would appear to be useful to be able to process each region differently. This has however not been useful in practice as the very small gains to be made from doing this are entirely offset by the increased complexity in programming such an application.

The initial stages of iconic processing, such as convolution and median filtering have no need for independent control. Connected component

labelling can operate under a single control yet it has been shown to be advantageous to extract the regions locally and then to combine them globally. However this process only makes use of data dependent local control to allow some of the patches to terminate early. A single control would simply force all patches to take the same time as the slowest patch. Forming histograms locally again makes use of the local data extraction but operates under a single control. The local extraction of straight lines does make some use of local control. This is in the situation when a peak count is performed as it is only necessary to record it if the count exceeds the current maximum. A single controller would impose some overhead on this as this occurs approximately a thousand times per iteration and with local control it only needs to be performed approximately a tenth of these times.

In some cases the arbitrary splitting up of the image into 16 by 16 squares is not as convenient as it is in cases such as line extraction or connected component labelling. For example in radii segmentation unless an object is contained wholly within a patch it is difficult to devise a way of efficiently segmenting a number of objects at a time. Computing the convex hull is also more complex in the case where there is more than one region and requires some shifting of data in order for it to be efficient.

This is where MSIMD should be winning over SIMD which often has to process each object sequentially. In practice the implementational difficulties of the fixed boundaries causes significant problems. Shifting the data into separate clusters is also somewhat problematic for several reasons. Firstly algorithms that require inter-processor communication may be disrupted by this, secondly it is not trivial to control such an operation given an arbitrary number of arbitrarily sized objects. Hence the multiple controllers are easier to justify from the ease of construction point of view than from performance.

7.3.7 Intermediate Layer Requirements

It has been suggested above that the symbolic layer is not entirely suitable for the manipulation of the data, such as the labels from the connected component operation, that are passed to it from the iconic arrays. Therefore, it is possible that there is a need for an intermediate level layer that may perform this type of operation that is demanded of the symbolic layer. The original design of the WPM [Howarth 87] [Nudd 88] had such a level of processors as the cluster controller was at that time a significantly more general purpose processor. The original reason for not extending the properties of this layer was that it reduces the usage of the iconic array, but it is now clear that the iconic array is often idle anyway. The enhanced processor necessary for this role would need the capability to form global properties such as a global 'count' and 'some/none' response and be sufficiently fast that the scalar components of tasks would not cause it to neglect the control of the iconic array for too long. This would also have control of any additional hardware included for the purpose of remapping the iconic data.

7.4 Conclusions

This chapter has described each individual feature of the WPM architecture in the light of the image analysis performance study of the previous three chapters. The architecture has in general provided a good programming model for all three levels and it has proven well matched to the needs of real-time image analysis at the bottom two levels of processing. The only significant problem with the architecture that has become apparent in this study is the mismatch of the symbolic processing to the rest of the task. This has been highlighted because of the extraordinary performance of the bottom two levels of the architecture.

The use of associativity as the sole means of extracting data from the iconic array has proven to be a great success with the addition of multiple vertical

communications paths. Having multiple controllers somewhat eases the control of these paths and makes the task of programming simpler yet does not appear to give a great performance gain. This is especially true because the fixed boundaries of the clusters often make it too complex to attempt to make good use of the multiple clusters.

The example of symbolic processing given in chapter 6 can trivially make use of a factor of four increase in the number of processors as the radii of the clusters is well within the boundaries of the partitions this would produce. However, this is only likely to give approximately a factor of two improvement in performance. If the MIMD array is kept then the only hopes for improvement appear to be by updating the Transputers with higher technology processors or by a much larger scale increase in the number of processors.

Performing the symbolic computation on an SIMD array, particularly one enhanced with some means of improving the floating point performance is a further means of enhancement. Unless this symbolic processing bottleneck can be successfully tackled then there is little point in having such a powerful iconic array. This leaves the interesting result that the symbolic processing appears to function adequately on the SIMD layer and also the SIMD layer is idle at this point. This possibly suggests removing the MIMD layer and adding additional hardware to assist the data mapping operation.

8

Summary and Conclusions

This thesis has attacked the problem of the architectural requirements of image analysis. A powerful massively parallel computer architecture has been designed with the intention of matching the problem of image analysis as closely as possible. The principles behind this architecture have been described in some depth and the architecture itself has been detailed. The architecture includes many of the architectural features that have been applied to the image analysis problem including SIMD, MIMD, MSIMD and associativity. This means that the architecture provides a unique platform on which to evaluate these features.

In order to be able to appraise this architecture many different typical image analysis components have been described and implemented. These include many low level operations including several novel implementations, a number of intermediate, translational tasks and a single symbolic recognition task. Within these components there are two complete image analysis tasks that address completely different problems. The first detects and segments blob like objects such as virus particles on a microscope slide and vehicles in infrared imagery. The second involves a geometric model based recognition system for locating certain specified objects in the image.

An efficient means of extracting straight lines from the image was described that offers a significant improvement on existing techniques both in the quality of the result and performance obtained. A robust recognition system was described that also offers a significant improvement on existing systems. This recognition system was supported by a comprehensive consistency verification procedure.

The result of these implementations highlighted several points about the Warwick Pyramid Machine. The most important of these was that the iconic and symbolic layers were not well matched in that the iconic processing array performed significantly better than the symbolic. This meant that the original idea of pipelining the processing flow such that the iconic array was processing one image while the symbolic array was processing the previous image was not valid. Therefore a significant portion of the architecture remained idle at many points of the processing flow. As the architecture was intended to maximise the use of the hardware available by matching it well to the problem this was a significant disadvantage.

A further detail that emerged during this work was that associative methods provided a suitable and efficient means of extracting data from the iconic array. The use of multiple controllers by splitting the array into clusters was shown to have only a small impact on performance yet assisted the modularity of the machine and made it simple to control the array based on

the important multiple associative responses. It was found that the symbolic processing tended to be SIMD in nature rather than MIMD as expected. It was also found that the communications patterns at this level could be made fairly regular without too large an increase in overhead.

On the whole the architecture is successful and with some modification could be made more so. It is clear that a more extensive survey of the requirements of the symbolic layer is required as this area of research is not well developed and what work there is in this area tends to be based on opinions rather than hard evidence.

Both the straight line extraction and the recognition algorithms appear to be very successful on real imagery and I believe they offer a significant contribution on their own merits to the field of image analysis.

Bibliography

[AMT 88] Active Memory Technology Ltd., "AMT DAP 500 - Technical Overview", 1988

[Arbib 87] Arbib M.A. and Hanson A.R., "Vision, Brain, and Cooperative Computation: An Overview", in Vision, Brain and Cooperative Computation, MIT Press, 1987

[Atherton 90a] Atherton T.J., Nudd G.R., Francis N.D., Kerbyson D.J., Packwood R.A., and Vaudin G.J. "A Scalable Multiple-SIMD Architecture For Real-Time Image Understanding", *Proc. IEE Colloquium on 'The role of image processing in defence and military electronics'*, London, April 1990

[Atherton 90b] Atherton T.J., Nudd G.R., Clippingdale S.C., Francis N.D., Kerbyson D.J., Packwood R.A., So Y.K., Vaudin G.J. and Walton D.W., "Detection and Segmentation of Blobs using the Warwick Multiple-SIMD Architecture", *Proc. SPIE/SPSE Symposium on Electronic Imaging Science & Technology*, Santa Clara, February 1990

[Ballard 81] Ballard D.H., "Generalizing the Hough Transform to Detect Arbitrary Shapes", *Pattern Recognition*, 13:111, 1981

[Batcher 80] Batcher K.E., "Design of a Massively Parallel Processor", *IEEE Trans.* C-29, pp 836-840, 1980.

[Batcher 82] Batcher K.E., "Bit-Serial Parallel Processing Systems", *IEEE Trans. on Computers*, 31:5, pp377-384, 1982

[Batcher 83] Batcher K.E., "Architecture of the MPP", *IEEE Computer Society Workshop on Computing Architecture for Pattern Analysis and Image Database Management*, 1983

[Beveridge 87] Beveridge J.R., Griffith J., Kohler R., Hanson A. and Riseman E., "Segmenting Images using Localised Histograms and Region Merging", *COINS Technical Report 87-88*, University of Mass., October 1987

[Beveridge 89] Beveridge J.R., Weiss R. and Riseman E.M., "Optimisation of 2-Dimensional Model Matching", *Proc. DARPA IU Workshop*, pp815-830, 1989

[Beveridge 90] Beveridge J.R., Weiss R. and Riseman E.M., "Combinatorial Optimisation Applied to Variable Scale 2D Model Matching", *ICPR-A*, pp 18-23, 1990

[Blevins 88] Blevins D.W., Davis E.W., Heaton R.A. and Reif J.H., "BLITZEN: A Highly Integrated Massively Parallel Machine", in *Proc. 2nd Symp. on Frontiers of Massively Parallel Compuation*, Fairfax, October 1988

[Bolles 82] Bolles R.C. and Cain R.A., "Recognising and Locating Partially Visible Objects: The Local-Feature-Focus Method", *Int. Journal of Robotics Research*, 1:3, pp57-82, 1982

[Bongiovanni 90] Bongiovanni G., Guerra C. and Levialdi S., "Computing the Hough transform on a Pyramid Architecture", *Machine Vision and Applications*, 3:2, pp 117-124, 1990

[Borkar 88] Borkar S., Cohn R., Cox G., Gleason S., Gross T., Kung H.T., Lam M.,Morre B., Peterson C., Pieper J., Rankin L., Tseng P.S., Sutton J., Urbanski J. and Webb J., "iWARP: An Integrated Solution to High-Speed Parallel Computation", *Proc. Supercomputing 88*, Orlando, Nov 1988, pp330-339

[Burns 86] Burns J.B., Hanson A. and Riseman E.M., "Extracting Straight Lines", *IEEE PAMI*, 8:4, pp 425-455, July 1986

[Canny 86] Canny J., "A Computational Approach to Edge Detection", *IEEE PAMI*, 8:6, pp679-697, Nov 1986

[Cantoni 86] Cantoni V., "I.P. Hierarchical Systems: Architectural Features", in Pyramidal Systems for Computer Vision, ed. V. Cantoni and S. Levialdi, Springer-Verlag, 1986

[Cass 88] Cass T.A, 1988, "Robust Parallel Computation of 2D Model-Based Recognition", *Proc DARPA Image Understanding Workshop*, pp640-650, 1988

[Cypher 87] Cypher R.E., Sanz J.L.C. and Snyder L., "The Hough Transform has O(N) Complexity On SIMD N x N Mesh Array Architectures", *Proc. CAPAMI*, pp115-121, 1987

[Cypher 90] Cypher R.E., Sanz J.L.C. and Snyder L., "Algorithms for Image Component Labelling on SIMD Mesh-Connected Computers", *IEEE PAMI*, 12:2, pp 276-281, Feb 1990

[Danielsson 81] Danielsson P.E., "Getting the Median Faster", *Computer Graphics and Image Processing*, 17:1, pp 71-78, 1981

[Davies 89] Davies E.R., "Occlusion Analysis for Object Detection using the Generalised Hough Transform", *Signal Processing*, 16, pp267-277, 1989

[Davies 91] Davies A. and Wilson R., "Linear Feature Extraction using the Multiresolution Fourier Transform", University of Warwick, *Computer Science Research Report 170*, 1991

[Deans 81] Deans S.R., "Hough Transform from Radon Transform", *IEEE PAMI*, 3:2, pp 185-188, Mar 1981

[DeGroot 88] De Groot A.J., Azevedo S.G., Schneberk D.J., Johansson E.M. and Parker S.R., "A Systolic Array for Efficient Execution of the Radon and Inverse Radon Transforms", *Proc. SPIE Advanced Algorithms and Architectures for Signal Processing* 3, pp 145-153, 1988

[Duda 72] Duda R.O. and Hart P.E., "Use of the Hough Transformation to Detect Lines and Curves in Pictures", *Comms. ACM*, 15:1, pp 11-15, Jan 1972

[Duff 85] Duff M.J.B., "The Clip Parallel Processor", *Computer Bulletin*, pp26-27, Dec 1985

[Duff 86a] Duff M.J.B. (editor), "Intermediate Level Image-Processing", Academic Press, 1986

[Duff 86b] Duff M.J.B. and Fountain T.J. (editors), "Cellular Logic Image Processing", Academic Press, New York, 1986

[Duff 90] Duff M.J.B. and Fountain T.J., "Enhancing the Two-Dimensional Mesh", *Proc. ICPR*, Atlantic City, 1990

[Duller 89] Duller A.W.G., Storer R.H., Thomson A.R. and Dagless E.L., "An Associative Processor Array for Image Processing", *Image and Vision Computing*, 7:2, pp151-158, May 1989

[Duncan 90] Duncan R., "A Survey of Parallel Computer Architectures", *IEEE Computer*, pp5-16, March 1990

[Duran 74] Duran B.S. and Odell P.L., "Cluster Analysis - A Survey", Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1974

[Elliman 88] Elliman D.G. and Mahmood A., "Towards Faster and more Shapely Thinning", *Proc. Parallel Processing for Computer Vision and Display*, Leeds, 1988

[Fisher 89] Fisher A.L. and Highnam P.T., "Computing the Hough Transform on a Scan Line Array Processor", *IEEE PAMI*, 11:3, pp 262-265, 1989

[Flynn 66] Flynn M.J., "Very High-Speed Computing Systems", *Proceedings of the IEEE*, 54:12, pp 1901-1909, Dec 1966

[Francis 89] Francis N.D., "User Manual for psim (pyramid simulator)", *VLSI group memo*, May 1989

[Francis 90a] Francis N.D., Nudd G.R., Atherton T.J., Kerbyson D.J., Packwood R.A., and Vaudin G.J. "Performance Evaluation of the Hierarchical Hough Transform on an associative M-SIMD architecture", *Proc. ICPR*, Atlantic City, June 1990

[Francis 90b] Francis N.D. and Nudd G.R., "Architectural Options for Image Processing Software", *Proc Int Conf. on Implementing Image Processing Systems Strategies and Solutions*, March 1990

[Francis 91] Francis N.D., Atherton T.J. and Nudd G.R., "Measurement of Array and Enhanced Array Architectures for Image Understanding", *ESPRIT MEASURE workshop*, May 1991

[Grimson 87] Grimson W.E.L. and Lozano-Perez T., "Localizing Overlapping Parts by Searching the Interpretation Tree", *IEEE PAMI*, 9:4, July 1987

[Grimson 88] Grimson W.E.L and Huttenlocher D.P., "On the sensitivity of the Hough transform for Object Recognition", *MIT AI memo* #1044, May 1988

[Grimson 90] Grimson W.E.L., "Object Recognition by Computer - the role of geometric constraints", MIT Press, 1990

[Guerra 87] Guerra C. and Hambrusch S., "Parallel Algorithms for Line Detection on a Mesh", *Proc. CAPAMI*, pp99-106, 1987

[Guerra 89] Guerra C. and Hambrusch S., "Parallel Algorithms for Line Detection on a Mesh", *Journal of Parallel and Distributed Computing*, 6, pp1-19, 1989

[Hanson 86] Hanson A.R. and Riseman E.M., "The VISIONS Image Understanding System - 1986", *COINS Technical Report* 86-62, 1986

[Haritgan 75] Hartigan J.A., "Clustering Algorithms", Wiley, 1975

[Helgason 80] Helgason S., "The Radon Transform", Birkhäuser, 1980

[Hildreth 88] Hildreth E.C. and Ullman S., "The Computational Study of Vision", *MIT AI Memo* 1038, April 1988

[Hillis 85] Hillis W.D., "The Connection Machine", MIT Press, Cambridge, 1985

[Hough 62] Hough P.V.C., "Methods and Means for Recognising Complex Patterns", US Patent 3069654, 1962

[Howarth 87] Howarth R.M., "A Heterogeneous Pyramid Array Architecture for Image Understanding", University of Warwick, Dept. Computer Science, Research Report 115, Dec. 1987

[Howarth 88] Howarth R.M. and Francis N.D., "Cluster Programming Language : Definition and user manual", University of Warwick, Department of Computer Science, Research Report 125, July 1988

[Howarth 91] Howarth R.M., "A Multiple Paradigm Parallel Architecture for Image Processing", MPhil Thesis, University of Warwick, May 1991

[Hubel 88] Hubel D.H., "Eye, Brain, and Vision", *Scientific American* Library, NewYork, 1988

[Hunt 89] Hunt D.J., "The AMT DAP - a Processor Array in a Workstation Enviroment", *Computer Systems Science and Engineering*, 4:2, pp107-114, April 1989

[Huttenlocher 88a] Huttenlocher D.P, 1988, "3D recognition of solid objects from a 2D Image", PhD thesis, MIT

[Huttenlocher 88b] Huttenlocher D.P. and Ullman S., "Recognising Rigid Objects by Alignment", *Proc DARPA IU workshop*, pp1114-1122, 1988

[Illingworth 87] Illingworth J. and Kittler J., "The Adaptive Hough Transform", *IEEE PAMI*, 9:5, pp 691-698, Sept 1987

[Illingworth 88] Illingworth J. and Kittler J., "A Survey of the Hough Transform", *CVGIP*, 44, pp 87-116, 1988

[Inmos 85] Inmos PLC, "Transputer Reference Manual", 1985

[Intel 89] Intel Inc., "Inroduction to iWARP", Intel Manual

[Jain 86] Jain A.K., "Cluster Analysis", in Handbook of Pattern Recognition and Image Processing, ed Young and Fu, pp33-57, Academic Press, 1986

[Jain 88] Jain A.K. and Dubes R.C., 1988, "Algorithms for Clustering Data", Prentice Hall

[Jarvis 73] Jarvis R.A. "On the identification of the convex hull of a finite set of points in a plane", *Information Processing Letters*, Vol 2, pp18-21, 1973

[Kushner 82] Kushner T., Wu A.Y. and Rosenfeld A., "Image Processing on MPP:1", *Pattern Recognition*, Vol 15, No 3, pp121-130, 1982

[Lea 91] Lea R.M. and Jalowiecki I.P, "Associative Massively Parallel Computers", *Proc. IEEE*, 79:4, April 1991

[Levialdi 72] Levialdi S., "On Shrinking Binary Pictures", Comms. ACM., V 15, pp7-10, 1972

[Li 86] Li H., Lavin M.A. and Le Master R.J., "Fast Hough Transform: A Hierarchical Approach", *CVGIP*, 36, pp139-161, 1986

[Little 89] Little J.J., Blelloch G.E. and Cass T.A., "Algorithmic Techniques for Computer Vision on Fine-Grained Parallel Machine", *IEEE PAMI*, 11:3,pp244-257, Mar 1989

[Little 87] Little J.J., Blelloch G. and Cass T., "Parallel Algorithms for Computer Vision on the Connection Machine", *Proc. DARPA Image Understanding Workshop*, pp 628-638, 1987

[Lotufo 89] Lotufo R.A., Dagless E.L., Milford D.J., Morgan A.D., Morrissey J.F. and Thomas B.T., "Hough Transform for Transputer Arrays", *Proc. IEE Conf. on Image Processing and its Applications, Warwick*, pp 122-130, 1989

[Lowe 87] Lowe D.G., "Three-Dimensional Object Recognition from Single Two-Dimensional Images", *Artificial Intelligence*, 31, pp255-395, 1987

[LSI 88] LSI Logic, "L64250 Histogram/Hough Transform Processor", preliminary data sheet, 1988

[Nudd 88a] Nudd G.R., Clippingdale S.C., Howarth R.M., Atherton T.J., Francis N.D., Vaudin G.J. and Walton D.W., "Motion Estimation for Video Bandwidth Compression using a Heterogeneous Pyramid Image Processing Architecture", *IAPR workshop on Computer Vision*, pp. 24-27, Tokyo, October 1988

[Nudd 88b] "A Heterogeneous Architecture for Parallel Image Processing", *UK Information Technology '88* , pp. 495-499, Swansea, July 1988

[Nudd 89a] Nudd G.R. and Francis N.D., "Architectures for Image Analysis", *Proc. 3rd IEE Conference on Image Processing and its Applications, Warwick*, pp. 445-451, July 1989

[Nudd 89b] Nudd G.R., Atherton T.J., Howarth R.M., Clippingdale S.C., Francis N.D., Kerbyson D.J., Packwood R.A., Vaudin G.J. and Walton D.W., "WPM : A Multiple-SIMD architecture for image processing", *3rd IEE conference on Image Processing and Its Applications*, pp. 161-165, Warwick, July 1989

[Nudd 90] Nudd G.R., Francis N.D., Atherton T.J., Kerbyson D.J., Packwood R.A., and Vaudin G.J. , "A Hierarchical Multiple-SIMD Architecture for Image Analysis", *Proc. ICPR*, Atlantic City, June 1990

[Nudd 91] Nudd G.R., Francis N.D., Atherton T.J., Kerbyson D.J., Packwood R.A., and Vaudin G.J. "A Hierarchical Multiple-SIMD Architecture for Image Analysis", to appear in *Journal of Machine Vision and Applications*

[Oldfield 85] Oldfield D.E. and Reddaway S.F., "An Image Understanding Performance Study on the ICL Distributed Array Processor", *Proc. CAPAIDM*, pp 256-264, 1985

[Pavlidis 74] Pavlidis T. and Horowitz S.L., "Segmentation of Plane Curves", *IEEE Trans on Computers*, 23:8, pp 860-870, 1974

[Princen 90] Princen J., Illingworth J. and Kittler J., "A Hierarchical Approach to Line Extraction Based on the Hough Transform", *CVGIP* 52, pp57-77, 1990

[Radon 17] Radon J., *Bericht. Saechs. Akad. Wiss.*, Leipzig, Math. Phy. Kl., 69, pp262-277, 1917

[Ramer 72] Ramer U., "An Iterative Approach for the Polygon Approximation of Plane Curves", *CGIP*, 1, pp244-56, 1972

[Reddaway 79] Reddaway S.F., "The DAP Approach", *Infotech State of the Art Report on Supercomputers*, Vol 2, 1979

[Reddaway 83] Reddaway S.F., "DAP and its Application to Image Processing Tasks", *Proc Image Processing Symposium* - RSRE Malvern, July 1983

[Reddaway 83] Reddaway S.F., "DAP and its Applications to Image Processing", *IEE Elec. Div, Colloq. on VLSI Modules for Image Processing*", 1983

[Rosenfeld 87] Rosenfeld A., "A Report on the DARPA Image Understanding Architectures Workshop", *Proc. DARPA Image Understanding Workshop*, pp298-302

[Rosenfeld 87] Rosenfeld A., Ornelas J. and Hung Y., "Hough Transform Algorithms for Mesh-Connected SIMD Parallel Processors", *CVGIP*, 41, pp293-305, 1987

[Russell 90] Russell S., "Hardware Additions to WPM", University of Warwick, Dept. of Computer Science, Project Report, 1990

[Ruttimann 86] Ruttimann U.E. and Webber R.L, "Fast Computing Median Filters on General-Purpose Image Processing Systems", *Optical Engineering*, 25:9, pp1064-7, Sept 1986

[Shu 88] Shu D.B. and Nash G., "Image Understanding Architecture and Applications", Machine Vision, ed J. Sanz, Prentice Hall, 1988

[Shu 90] Shu D. and Nash G., "Finding Connected Components of a Graph on a Gated Connected VLSI Network", Personal Communication, to be published, 1990

[Siegel 86a] Siegel H.J., "PASM: A Reconfigurable Multi-Microcomputer System for Image Processing", in Languages and Architectures for Image Processing, ed. M.J.B. Duff and S. Levialdi, Academic Press, 1986

[Siegel 86b] Siegel L.J. , "Image Processing on a Partionable SIMD machine", in Languages and Architectures for Image Processing, ed. M.J.B. Duff and S. Levialdi, Academic Press, 1986

[Silberberg 85] Silberberg T.M., "The Hough Transform on the Geometric Arithmetic Parallel Processor", *IEEE CS Workshop on Computer Architecture*, pp 387-393, 1985

[Skingley 87] Skingley J. and Rye A.J., "The Hough transform applied to SAR images for thin lines detection", *Pattern Recognition Letters*, 6, pp61-67, June 1987

[Stockman 82] Stockman G, Kopstein S. and Benett S., "Matching Images to Models for Registration and Object Detection via Clustering", *IEEE PAMI*, 4:3, May 1982

[Tanimoto 86] Tanimoto S.L., "Architectural Issues for Intermediate-Level Vision", in Intermediate-Level Image Processing, Editor M.J.B. Duff, Academic Press 1986

[Taylor 85] Taylor C.J., "Serial Architectures for Image Processing", in Image Processing System Architectures, editors J. Kittler and M.J.B. Duff, Research Studies Press, pp 39-46, 1985

[Thompson 87] Thompson D.W. and Mundy J.L., "Three Dimensional Model Matching from an Unconstrained Viewpoint", *Proceedings of the International Conference on Robotics and Automation*, IEEE Computer Society Press, pp 208-220, 1987

[Thompson 88] Thompson D.W. and Mundy J.L., "Model-Directed Object Recognition on the Connection Machine", *Proceedings DARPA Image Understanding Workshop*, pp98-106, 1988

[Timmermann 88] Timmermann D. and Hahn H., "Hough Transform using Cordic Method", *Electronics Letters*, 25:3, Feb 1989

[TMC 89] Thinking Machines Corporation, "Connection Machine - Model CM-2 Technical Summary", version 5.1, May 1989

[Tucker 88] Tucker L.W., Feynman C.R. and Fritzsche D.M, June 1988, "Object Recognition using the Connection Machine", Proc *IEEE conference on Computer Vision and Pattern Recognition*

[Vaudin 89] Vaudin G.J., Nudd G.R., Atherton T.J., Clippingdale S.C., Francis N.D., Howarth R.M., Kerbyson D.J., Packwood R.A., and Walton D.W., "A Generalised Parallel Architecture for Image Based Algorithms", *Eurographics* 1989, Hamburg, September 1989

[Vaudin 91] Vaudin G.J.B., "A Unified Programming System for a Multi-Paradigm Parallel Architecture", PhD Thesis, University of Warwick, Aug 1991

[Walton 89] Walton W., "Options for a Gate Array Count", University of Warwick, Dept. Computer Science, VLSI memo, 1989

[Weems 84] Weems C., "Image Processing on a Content Addressable Array Parallel Processor", PhD Thesis, Univ. Mass., Coins., 1984

[Weems 89] Weems C., Riseman E., Hanson A. and Rosenfeld A., "A Report on the Results of the DARPA Integrated Image Understanding Benchmark Exercise", *Proc DARPA IU Workshop*, pp 165-192, 1989

[Weems 89] Weems C., Levitan S.P., Hanson A.R. and Riseman E.M, "The Image Understanding Architecture", *International Journal on Computer Vision*, pp 251-282, 1989

[Weems 91] Weems C.C, Hanson A.R. and Riseman E.M, "The Architectural Requirements of Image Understanding with Respect to Parallel Processing", *Proceedings of IEEE*, 79:4, pp 537-548, Apr 1991

[WSTL 89] WSTL Ltd., "The Evaluation of Parallel Processing Using Very High Performance (VHP) Microprocessors for Fuze Design", Report for RARDE Contract MGW31B/2143, edited N.D. Francis, Mar. 1989

[WSTL 90] WSTL Ltd., "Image Processing Algorithms from 2D Image Arrays for Fuzes", Report for RARDE Contract MGW31B/2150, edited N.D. Francis, Oct. 1990

[Wysocki 89] Wysocki J., "A Parallel Implemenation of the Canny Operator for the DAP", QMC report 491, 1989

[Yuen 89] Yuen H.K., Illingworth J. and Kittler J., "Detecting Partially Occluded Ellipses with the Hough Transform", *Image and Vision Computing*, 7, pp31-37, 1989

A

Sample Code

This appendix contains sample code to illustrate some of the algorithms given in the text and to show some of the coding methods used. Included within are examples of both a low-level CLASS program and Pyramid C++ programs.

CLASS Hough transform

This program is written in CLASS (cluster assembly language). The program extracts line segments over each cluster. Each cluster runs a copy of the program independently of all of the other clusters.

```
/*
      Perform a Hough transform as in Cluster Applications note 1
      Nick Francis
      21/7/89
      Modified 2/90
*/

#include "label.cl"
#include "sign.cl"
#include "macros.cl"
#include "decode32.cl"

#define no_trig_bits 6
#define size_of_trig_entry 19
#define num_thetas 32
#define num_rhos 32
#define min_len_line 4
#define NUM_RHOS R00
#define THETA R01
#define RHO R02
#define OP_PTR R04
#define MAX R05
#define TRIG_PTR R06
#define TMP1 R07
#define TMP2 R08
#define DEC_PTR R09

#define start_results 100

PSEG
cos_theta:      DEFS    no_trig_bits
sin_theta:      DEFS    no_trig_bits
theta_copy:     DEFS    no_trig_bits-1
ypos:           DEFS    5
xpos:           DEFS    5
rho:            DEFS    no_trig_bits+4+2
tmp1:           DEFS    no_trig_bits+4+1
tmp2:           DEFS    no_trig_bits+4+1
edge:           DEFS    1
dummy:          DEFS    1
flag:           DEFS    1
not_flag:       DEFS    1
answer:         DEFS    1
```

```
decoded:        DEFS    num_thetas*num_rhos
dir_diff:       DEFS    no_trig_bits-1
edge_dir:       DEFS    8
dir_flag:       DEFS    1
lower_thres:    DEFS    no_trig_bits-1
upper_thres:    DEFS    no_trig_bits-1

DSEG
                ORIG    start_results
store:          DEFS    1000

CSEG

label_y($ypos)
                ;                       ; SF    ; PEADDR=$ypos+4
label_x($xpos)
                ;                       ; SF    ; PEADDR=$xpos+4

/* Initialise variables */

        ; MOVE SODR NUM_RHOS   ;         ; num_rhos
        ; MOVE SODR OP_PTR     ;         ; $store
                ;               ; SF    ; PEADDR=$answer
        ; MOVE SODR TRIG_PTR   ;         ; $arg0
        ; MOVE SODR DEC_PTR    ;         ; $decoded

/* Initialise orientation threshold set to 5 */
                ;               ; QT    ;
                ;               ; SQ    ; PEADDR=$lower_thres
                ;               ; SF    ; PEADDR=$lower_thres+1
                ;               ; SQ    ; PEADDR=$lower_thres+2
                ;               ; SF    ; PEADDR=$lower_thres+3
                ;               ; SF    ; PEADDR=$lower_thres+4
/* set upper to 27 */
                ;               ; SQ    ; PEADDR=$upper_thres
                ;               ; SQ    ; PEADDR=$upper_thres+1
                ;               ; SF    ; PEADDR=$upper_thres+2
                ;               ; SQ    ; PEADDR=$upper_thres+3
                ;               ; SQ    ; PEADDR=$upper_thres+4

/*
   Calculate all of the rho values ahead of the main loop and
   decode each putting all of the results into the decoded table
*/

for_loop(num_thetas)
/* Load cos(theta) and sin(theta) */
CALL_D  ; MOVE SORY TRIG_PTR    ;       ; SEQ=ALU

/* dir_flag = set if edge direction is near to current theta */
sub(no_trig_bits-1,$dir_diff,$theta_copy,$edge_dir+8+1-no_trig_bits)
abs(no_trig_bits-1,$dir_diff)
/* if diff < lower_thres */
```

```
compare(no_trig_bits-1,$lower_thres,$dir_diff)
                                        ; QC      ;
                                        ; SQ      ; PEADDR=$dir_flag
/* or diff >= upper_thres */
compare(no_trig_bits-1,$dir_diff,$upper_thres)
                                        ; QS      ; PEADDR=$dir_flag
                                        ; CPCQT   ;
                                        ; QC      ;
                                        ; SQ      ; PEADDR=$dir_flag

compared:
/* RHO = X.cos(theta) + Y.sin(theta) */
signed_mult(no_trig_bits,5,$tmp1,$cos_theta,$xpos)
signed_mult(no_trig_bits,5,$tmp2,$sin_theta,$ypos)
convert(no_trig_bits+4,$tmp1)
convert(no_trig_bits+4,$tmp2)
add(no_trig_bits+5,$rho,$tmp1,$tmp2)

decode32reg($rho+5,DEC_PTR,TMP1,TMP2)

/* If not dir_flag then zap rho */
                                        ; ASN     ; PEADDR=$dir_flag
cond_clear(32,DEC_PTR)

finished_calc:

CONT    ; ADD TODRR DEC_PTR             ;         ; num_rhos
CONT    ; ADD TODRR TRIG_PTR            ;         ; size_of_trig_entry

end_for_loop

restart:
        ; MOVE SODR THETA               ;         ; 0
        ; MOVE SODR RHO                 ;         ; 0
        ; MOVE SODR MAX                 ;         ; 0
        ; MOVE SODR DEC_PTR             ;         ; $decoded-1

/*
     Find the best rho and theta pairing
*/

for_loop(num_thetas)
        ; MOVE SODR RHO                 ;         ; 0
FOR_D   ; MOVE SORY NUM_RHOS            ;         ; SEQ=ALU
        ; INC SORR DEC_PTR              ; AS      ; PEADDR=ALU
                                        ; AMS     ; PEADDR=$edge
                                        ; QA      ;
                                        ; SQ      ; PEADDR=$dummy
                                        :         :
                                        :         :
                                        :         :
        ; MOVE SOD NRA                  :         ; ALU=COUNT
        ; SUBS TORAY MAX                ;         ; COND=N
```

```
CCC_D  ;                        ;           ; $new_max
DJMP_S ; INC SORR RHO           ;           ;
       ; INC SORR THETA         ;           ;
end_for_loop

/* Update OP_PTR now that the RHO and THETA have been fixed */
       ; ADD TODRR OP_PTR       ;           ; 2

/* If MAX < min_len_line then stop now */
       ;                        ;           ; COND = N
       ; SUBR TODRY MAX         ;           ; min_len_line
BRCC_D;                         ;           ; $exit

/*
     Store the edge points that correspond to a max
*/
       ;                        ; ASN       ; PEADDR=$flag
       ;                        ; QSN       ; PEADDR=$answer
       ;                        ; AMQ       ;
       ;                        ; SAN       ; PEADDR=$answer

/*
   Remove edge points (flag is set on those who contributed to the max)
*/
       ;                        ; ASN       ; PEADDR=$flag
       ;                        ; AMS       ; PEADDR=$edge
       ;                        ; QA        ;
       ;                        ; SQ        ; PEADDR=$edge

/*
     Find End Points - and put into memory
*/
CALL_D ;                        ;           ; $find_end_points

/* Update result pointers */
       ; MOVE SORY MAX          ;           ; RAM=ALU
       ; INC SORR OP_PTR        ;           ; ADDR=ALU

/* As we know MAX >= min_len_line then we try again */
       ;                        ;           ; COND = N
       ; SUBS TODRY MAX         ;           ; min_len_line
BRCC_D ;                        ;           ; $restart

exit:
HALT   ;                        ;           ;

/*
   Store latest rho and theta
*/
new_max:
       ; MOVE SOAR MAX          ; SQ        ; PEADDR=$flag
       ; MOVE SORY RHO          ;           ; RAM=ALU
       ; ADD TODRY OP_PTR       ;           ; 1 ADDR=ALU
```

```
          ; MOVE SORY THETA          ;          ; RAM=ALU
RET       ; ADD TODRY OP_PTR         ;          ; 2 ADDR=ALU

/*
     Find the start and end point of a line found by the HHT

     This works for 32 different theta values

     Uses THETA, OP_PTR and $flag

     Changes Q $dummy, $not_flag, EDGE, ACC, TMP1, TMP2
*/

find_end_points:
/* Get theta */
          ; MOVE SORY OP_PTR         ;          ; ADDR=ALU
          ;                          ;          ;
          ; MOVE SODR THETA          ;          ; ALU=RAM

/* Invert flag */
          ;                          ; QSN      ; PEADDR=$flag
          ;                          ; SQ       ; PEADDR=$not_flag

/* Shift line into edge register (shift the correct way depending upon theta */
          ;                          ;          ; COND=N
          ; SUBS TODRY THETA         ;          ; 7   /* Y=7-THETA */
BRCC_D;                              ;          ; $gt7 /* bra if theta > 7 */
BRA_D     ;                          ;          ; $le7

gt7:
          ; SUBR TODRY THETA         ;          ; 24 /* Y=THETA-24 */
BRCC_D;                              ;          ; $lt24 /* bra if theta < 24 */

le7: /* Gets here if theta <= 7 or theta >= 24 */
          ;                          ; RSO      ; PEADDR=$not_flag
          ;                          ; RSO      ;
          ; MOVE SODR TMP1           ;          ; ALU=EDGE

/* Now step along the word from both ends to find the first 0 */
          ; MOVE SOD NRA             ;          ; 1 /* set mask */
          ;                          ;          ; COND=Z
FOR_D     ;                          ;          ; 16
          ; AND TORAY TMP1           ;          ;
CCC_D     ;                          ;          ; $got_zero1
XTCC_D    ;                          ;          ; $next1
DJMP_S    ; SHUPZ SHA NRA           ;          ;

next1:
          ; MOVE SOD NRA             ;          ; 0x8000 /* set mask */
          ;                          ;          ; COND=Z
FOR_D     ;                          ;          ; 16
          ; AND TORAY TMP1           ;          ;
```

```
CCC_D   ;                       ;           ; $got_zero1
XTCC_D  ;                       ;           ; $next2
DJMP_S  ; SHDNZ SHA NRA         ;           ;
next2:
RET     ;                       ;           ;

lt24: /* Gets here if 7 < theta < 24 */
        ;                       ; RS        ; PEADDR=$not_flag
        ;                       ; RS        ;
        ;                       ;           ;
        ; MOVE SODR TMP1        ;           ; ALU=EDGE

/* Now step along the word from both ends to find the first 0 */
        ; MOVE SOD NRA          ;           ; 1 /* set mask */
        ;                       ;           ; COND=Z
FOR_D   ;                       ;           ; 16
        ; AND TORAY TMP1        ;           ;
CCC_D   ;                       ;           ; $got_zero2
XTCC_D  ;                       ;           ; $next3
DJMP_S  ; SHUPZ SHA NRA         ;           ;

next3:
        ; MOVE SOD NRA          ;           ; 0x8000 /* set mask */
        ;                       ;           ; COND=Z
FOR_D   ;                       ;           ; 16
        ; AND TORAY TMP1        ;           ;
CCC_D   ;                       ;           ; $got_zero2
XTCC_D  ;                       ;           ; $next4
DJMP_S  ; SHDNZ SHA NRA         ;           ;

next4:
RET     ;                       ;           ;

/* Jumps here when the zero is found (theta <=7 or theta >=24) - posn is in A */

/* Order needs reversing so that it o/ps y,x not x,y */

got_zero1:
        ; MOVE SOA NRY          ; ARO       ; EDGE=ALU
        ;                       ; AMS       ; PEADDR=$flag
        ;                       ; SAN       ; PEADDR=$dummy
        ;                       ; RS        ; PEADDR=$dummy
        ;                       ; RS        ;
        ;                       ;           ;
        ; COMP SODR TMP2        ;           ; ALU=EDGE
        ; MOVE SORY TMP2        ;           ; RAM=ALU
        ; INC SORR OP_PTR       ;           ; ADDR=ALU
        ; MOVE SOA NRY          ;           ; RAM=ALU
        ; INC SORR OP_PTR       ;           ; ADDR=ALU
RET     ; MOVE SOZ NRY          ;           ; /* return with Z flag set */

/* Jumps here when the zero is found (7 < theta < 24) - posn is in A */
got_zero2:
```

```
        ; MOVE SOA NRY           ;          ; RAM=ALU
        ; INC SORR OP_PTR        ;          ; ADDR=ALU
        ; MOVE SOA NRY           ;  AR      ; EDGE=ALU
        ;                        ;  AMS     ; PEADDR=$flag
        ;                        ;  SAN     ; PEADDR=$dummy
        ;                        ;  RSO     ; PEADDR=$dummy
        ;                        ;  RSO     ;
        ;                        ;          ;
        ; COMP SODR TMP2         ;          ; ALU=EDGE
        ; MOVE SORY TMP2         ;          ; RAM=ALU
        ; INC SORR OP_PTR        ;          ; ADDR=ALU
RET     ; MOVE SOZ NRY           ;          ; /* return with Z flag set */

#include "trig_table.cl"
```

PC++ Hough Program

This program has the same function as the previous program except that it is written in pyramid C++, this allows comparisons to be drawn between the ease of programming in each language. Note that this PC++ program, unlike most others, is written at the patch (cluster) level rather than the image level as will be found in other examples in the appendix.

```
#include <par.h>
#include "defs.h"
#include "activity.h"

int cos_table[32] =  {
                     0,   -1   -3,   -4,
                     -6,  -7,  -8,   -10,
                     -11, -12, -13,  -14,
                     -14, -15, -15,  -15,
                     -16, -15, -15,  -15,
                     -14, -14, -13,  -12,
                     -11, -10, -8,   -7,
                     -6,  -4,  -3,   -1
                     };

int sin_table[32] =  {
                     16,  15,  15,  15,
                     14,  14,  13,  12,
                     11,  10,  8,   7,
                     6,   4,   3,   1,
                     0,   -1,  -3,  -4,
                     -6,  -7,  -8,  -10,
                     -11, -12, -13, -14,
                     -14, -15, -15, -15
                     };


Line get_ends(int vert, int horiz, int angle) ;

void
Cluster::hough (Index edge_index, Index dir_index)
{
   const int num_rhos = 32 ;
   const int num_thetas = 32 ;
   const int min_line_length = 5 ;
   BitPatch edge(ram, edge_index) ;

   BytePatch dir(ram, dir_index) ;
```

```
ShortPatch xpos(ram) ;
ShortPatch ypos(ram) ;
ShortPatch theta(ram) ;
ShortPatch trunc_rho(ram) ;
ShortPatch orient_diff(ram) ;
ShortPatch rho(ram) ;
BitPatch active(ram) ;
BitPatch flag(ram) ;
BitPatch store(ram) ;
int max_number=0, max_rho ;
int rho_st, theta_st ; int min_rho ;
int number ;
int line_count = 0 ;
int output_pos = 8 ;
LineList list ;

xpos.label_x() ;
ypos.label_y() ;

/* For each line */
do {
   theta.set(0) ;
   max_number=0 ;
   for (int i=0; i< num_thetas; i++) {
      rho = xpos * ShortPatch(ram, cos_table[i]) ;
      rho += ypos * ShortPatch(ram, sin_table[i]) ;
      trunc_rho = rho / ShortPatch(ram, 16) ;

      // Find max and min values of rho
      max_rho = trunc_rho.get_max() ;
      min_rho = trunc_rho.get_min() ;

      // Check difference in orientation is not 2 (perpendicular)
      active = edge & ((((theta>>3) - dir).abs()) != ShortPatch(ram,2)) ;
      //active = edge ;

      // check each value of rho to find max num responders
      for (int j=min_rho; j<=max_rho; j++) {
         store = active & (trunc_rho == ShortPatch(ram,j)) ;
         number = store.count() ;
         if (number > max_number) {
            max_number = number ;
            rho_st = j ;
            theta_st = i ;
            flag = store ;
         }
      }
      theta += ShortPatch(ram,1) ;
   }

   if (max_number >= min_line_length) {
      // Remove edge points that contributed to the max
      edge &= ~flag ;
```

```
            list.add(get_ends((~flag).and_rows(),(~flag).and_cols(),theta_st)) ;
        }
    //} while (0) ;  // only find best
    } while (max_number >= min_line_length) ; // Loop around for each line

    list.output(x_origin, y_origin, 7) ;

}

Line
get_ends(int vert, int horiz, int angle)
{
    int j,i, xpos1, ypos1, xpos2, ypos2 ; xpos1 = 0 ;

    for(j=0,i=0x0001; ((i & horiz) && (j<16)); j++, i<<=1)
        xpos1++ ;
    xpos2 = 15 ;
    for(j=0,i=0x8000; ((i & horiz) && (j<16)); j++, i>>=1)
        xpos2-- ; ypos1 = 0 ;
    for(j=0,i=1; ((i & vert) && (j<16)); j++, i<<=1)
        ypos1++ ; ypos2 = 15 ;
    for(j=0,i=0x8000; ((i & vert) && (j<16)); j++, i>>=1)
        ypos2-- ;

    if (angle > 15) {
        int tmp = xpos1 ;
        xpos1 = xpos2 ;
        xpos2 = tmp ;
    }

    // Now store data in a new structure
    Line data(xpos1, ypos1, xpos2, ypos2) ;

    return data ;
}
```

Sequential C Connected Component

This program is a conventional C implementation of Cypher's connected component labelling algorithm. It makes use of the Warwick Image Format (WIF) library and hence images are held as wif structures and creating, loading and saving of images etc is dealt with by library calls.

```
#include <stdio.h>
#include <wif.h>

usage()
{
   fprintf(stderr,"Usage: mesh_con_comp -i input -o output\n");
   exit(1);
}

wif_struct *out[101] ;

float foo = 1 ;

void shrink(), write_im() ;
int read_im(), read_bit() ;

/* this should be a concat of i,j,y */
float
get_new_label(i,j,y)
int i,j,y ;
{
   return foo++ ;
}

main(argc,argv)
int argc;
char *argv[];
{
   char input[100];
   char output[100];
   wif_struct *in_wif,*out_wif, *label_wif, *new_label_wif;
   int i,j,n,y,sum;
   float **labels ;
   unsigned char **new_label ;

   if (scanargs(argc,argv,"-i %s",input)!=1) usage() ;
   if (scanargs(argc,argv,"-o %s",output)!=1) usage() ;
   in_wif=read_wif(input);
   label_wif=derive_wif(in_wif,in_wif->y_size, in_wif->x_size, REAL) ;
   labels=label_wif->real_image ;
   new_label_wif=derive_wif(in_wif,in_wif->y_size, in_wif->x_size, BYTE) ;
```

```
 new_label=label_wif->byte_image ;

n = in_wif->x_size + in_wif->y_size ;
if (n>800) {
   fprintf (stderr, "max sum of sides of image = 800\n") ;
   exit(1) ;
}
for (i=0;i<n/8+1;i++)
   out[i] = init_wif(in_wif->y_size, in_wif->x_size, BYTE) ;

for(i=0;i<in_wif->y_size;i++)
   for(j=0;j<in_wif->x_size;j++)
      write_im(i,j,0,in_wif->byte_image[i][j]) ;

sum=1 ;
for(y=0;(y<n-1)&&sum;y++) {/* num iterations equal to sum of sides */
   fprintf (stderr, "shrinking iteration %d\r", y) ;
   for(i=0;i<in_wif->y_size;i++)
      for(j=0;j<in_wif->x_size;j++)
         shrink(i,j,y) ;
   /*
   for(i=0;i<in_wif->y_size;i++) {
      for(j=0;j<in_wif->x_size-1;j++)
         printf ("%d ", read_im(i,j,y)) ;
      printf ("%d\n", read_im(i,in_wif->x_size-1,y)) ;
   }
   printf ("\n") ;
   */
   sum=0 ;
   for(i=0;i<in_wif->y_size;i++)
      for(j=0;j<in_wif->x_size;j++)
         sum+=read_im(i,j,y) ;
}

n=y ;

fprintf (stderr, "\n%d iterations needed\n", n) ;

for(y=n;y>=1;y--) {
   fprintf (stderr, "labelling iteration %d\r", n-y) ;
   for(i=0;i<in_wif->y_size;i++)
      for(j=0;j<in_wif->x_size;j++)
         new_label[i][j] = 0 ;
   for(i=0;i<in_wif->y_size;i++)
      for(j=0;j<in_wif->x_size;j++) {
         if (read_im(i,j,y)) {
            if (read_im(i,j-1,y-1)) {
               labels[i][j-1]=labels[i][j] ;
               new_label[i][j-1] = 1 ;
            }
            if (read_im(i-1,j-1,y-1)) {
               labels[i-1][j-1]=labels[i][j] ;
               new_label[i-1][j-1] = 1 ;
```

```
                }
                if (read_im(i-1,j,y-1)) {
                    labels[i-1][j]=labels[i][j] ;
                    new_label[i-1][j] = 1 ;
                }
                if (read_im(i,j,y-1)) {
                    new_label[i][j] = 1 ;
                }
            }
            else { if (!new_label[i][j]) labels[i][j]=0 ; }
        }
    for(i=0;i<in_wif->y_size;i++)
        for(j=0;j<in_wif->x_size;j++)
            if ((read_im(i,j,y-1)) && (!new_label[i][j]))
                labels[i][j] = get_new_label(i,j,y) ;

    /*
    for(i=0;i<in_wif->y_size;i++) {
        for(j=0;j<in_wif->x_size-1;j++)
            printf ("%4.0f ", labels[i][j]) ;
        printf ("%4.0f\n", labels[i][in_wif->x_size-1]) ;
    }
    printf ("\n") ;
    */
  }

  fprintf (stderr, "%4.0f regions found\n", foo-1) ;

  write_wif(label_wif,output);
}

void
shrink(i,j,y)
int i,j,y ;
{
  int here, north=0, west=0, north_west=0, value ;

  here = read_im(i,j,y) ;
  if (i!=0) north = read_im(i-1,j,y) ;
  if (j!=0) west = read_im(i,j-1,y) ; ;
  if ((j!=0)&&(i!=0)) north_west = read_im(i-1,j-1,y) ; ;
  value = ((here+west+north>1)+(here+north_west>1))>0 ;
  write_im(i,j,y+1,value) ;
}

int
read_bit(bit, num)
int bit, num ;
{
  int mask = 0x01 << bit ;

  return (num&mask)>0 ;
}
```

```
int
read_im(i,j,y)
int i,j,y ;
{
   int pos = y/8 ;
   int bit = y - pos*8 ;

   if ((i<0)||(j<0)) return 0 ;
   if (i>=out[0]->y_size) return 0 ;
   if (j>=out[0]->x_size) return 0 ;

   return read_bit(bit, out[pos]->byte_image[i][j]) ;
}

void
write_im(i,j,y,value)
int i,j,y,value ;
{
   int pos = y/8 ;
   int bit = y - pos*8 ;
   int mask = 0x01 << bit ;

   if ((i<0)||(j<0)) return ;
   if (i>=out[0]->y_size) return ;
   if (j>=out[0]->x_size) return ;

   if (value)
      out[pos]->byte_image[i][j] |= mask ;
   else
      out[pos]->byte_image[i][j] &= ~mask ;
}
```

PC++ Connected Component

This program is an image wide PC++ program to perform the connected component labelling operation. A 'where' statement is used to provide conditional support for the SIMD layer.

```
*
      PC++ Implementation of Cypher's labelling algorithm

      (SIZE = size of array.  This is expected to be at most 256, if greater than
      this then adjustments are needed to get_new_label and labels might need
      to be LongImage)
*/
#include <par.h>
#include "defs.h"
#include "activity.h"

extern ByteImage xImage, yImage ; // Coordinate Images

// Generate a new label from the iteration number and x and y coordinates
IntImage
get_new_label(int y)
{
      return (((IntImage(y)<<16)&xImage)<<8)&yImage ;
}

// Label the image
BitImage
label(BitImage &image)
{
      BitImage       store[2*SIZE] ;
      IntImage       labels ;
      BitImage       new_label ;
      ByteImage      here, north, west, north_west ;
      int y, count = 1 ;

      store[0] = image ;

      // Shrinking operation
      for(y=0; (y<2*SIZE-1 && count); y++) {
            here = ByteImage(store[y]) ;
            north = ByteImage(store[y].north()) ;
            west = ByteImage(store[y].west()) ;
            north_west = ByteImage(store[y].north().west()) ;
            store[y+1] = ((ByteImage(here+north+west > ByteImage(1)) +
                  ByteImage(here+north_west>ByteImage(1)))>ByteImage(0)) ;
            count = store[y].count() ;
      }
      // cout << "Number of shrinking iterations=" << y << "\n" ;
```

```
    // Iterate labels backwards
    for(; y>=1; y--) {
        new_label = BitImage(0) ;
        where(store[y-1] && store[y].east()) {
            new_label = BitImage(1) ;
            labels = labels.east() ;
        }
        where(store[y-1] && store[y].east().south()) {
            new_label = BitImage(1) ;
            labels = labels.east().south() ;
        }
        where(store[y-1] && store[y].south()) {
            new_label = BitImage(1) ;
            labels = labels.south() ;
        }
        where(store[y-1] && store[y]) {
            new_label = BitImage(1) ;
        }

        where(!store[y] && !new_label) {
            labels = IntImage(0) ;
        }
        where(store[y-1] && !new_label) {
            labels = get_new_label(y) ;
        }
    }

    return labels ;
}
```

PC++ Median (version 1)

This is a PC++ implementation of Danielsson's median filter.

```
/*
    PC++ Implementation of Danielsson's median algorithm

    performs a 'med_size' by 'med_size' median filter on input image

    parameters:   image as a ByteImage
    compile time constant, the size of the median (this has to be odd)
*/
#include <par.h>
#include "defs.h"
#include "activity.h"

const int med_size = 9 ;

ByteImage
median(ByteImage &image)
{
    const int mask_size = med_size * med_size ;
    const int median_pos = int(mask_size/2) + 1 ;
    ByteImage     in[mask_size];
    BitImage  valid[mask_size];
    BitImage  out[8] ;
    int   i,j,k,b;
    ByteImage n,m;

    // Shift Image into each processor (only works for odd sizes of med_size)
    k=0;
    in[k++] = image ;
    for(i=3; i<=med_size; i+=2) {
        in[k++] = in[k-1].west() ;
        for(j=0; j<i-2; j++)
            in[k++] = in[k-1].south() ;
        for(j=0; j<i-1; j++)
            in[k++] = in[k-1].east() ;
        for(j=0; j<i-1; j++)
            in[k++] = in[k-1].north() ;
        for(j=0; j<i-1; j++)
            in[k++] = in[k-1].west() ;
    }

    for(i=0;i<mask_size;i++)
        valid[i] = BitImage(1) ;

    n=ByteImage(0);
    for(i=0;i<mask_size;i++) // count number of 0's in msb
        n+=valid[i] && !in[i][7];

    // Compute median
```

```
    for(b=7;b>1;b--) {
        where (n>=ByteImage(median_pos)) {
            out[b] = 0 ;
            for(i=0;i<mask_size;i++)
                valid[i] &= in[i][b];
            m=ByteImage(0);
            for(i=0;i<mask_size;i++) // count number of 1's
                m+=valid[i] && in[i][b-1];
            n-=m;
        } else {
            out[b] = 1 ;
            for(i=0;i<mask_size;i++)
                valid[i] &= in[i][b];
            m=ByteImage(0);
            for(i=0;i<mask_size;i++) // count number of 0's
                m+=valid[i] && in[i][b-1];
            n+=m;
        }
    }

    out[0] = (n<ByteImage(median_pos)) ;

    return ByteImage(out) ;
}
```

PC++ Median (version 2)

This is another PC++ implementation of Danielsson's median filter but is improved over the previous implemenation that presented some performance overhead for an SIMD implementation due to the large 'where' statement. This has been adjusted to minimise the length of the conditional part of the code.

```
/*
      PC++ Implementation of Danielssons median algorithm

      performs a 'med_size' by 'med_size' median filter on input image

      parameters:    image as a ByteImage
      compile time constant, the size of the median (this has to be odd)

      This version has been optimised for SIMD implementation.  The where
      statement has been reduced to only one line which means that the amount
      of code that is conditional (and hence very bad on SIMD) is reduced
      dramatically
*/
#include <par.h>
#include "defs.h"
#include "activity.h"

const int med_size = 9 ;

ByteImage
median(ByteImage &image)
{
      const int mask_size = med_size * med_size ;
      const int median_pos = int(mask_size/2) + 1 ;
      ByteImage      in[mask_size];
      BitImage  valid[mask_size], flag;
      BitImage  out[8] ;
      int    l,j,k,b;
      ByteImage n,m;

      // Shift Image into each processor (only works for odd sizes of med_size)
      k=0;
      in[k++] = image ;
      for(i=3; i<=med_size; i+=2) {
            in[k++] = in[k-1].west() ;
            for(j=0; j<i-2; j++)
                  in[k++] = in[k-1].south() ;
            for(j=0; j<i-1; j++)
                  in[k++] = in[k-1].east() ;
            for(j=0; j<i-1; j++)
```

```
            in[k++] = in[k-1].north() ;
        for(j=0; j<i-1; j++)
            in[k++] = in[k-1].west() ;
    }

    for(i=0;i<mask_size;i++)
        valid[i] = BitImage(1) ;

    n=ByteImage(0);
    for(i=0;i<mask_size;i++) // count number of 0's in msb
        n+=valid[i] && !in[i][7];

    // Compute median
    for(b=7;b>1;b--) {
        flag=!(n>=ByteImage(median_pos));
        out[b]=flag;
        for(i=0;i<mask_size;i++)
            valid[i] &= (flag==in[i][b]);
        m=ByteImage(0);
        for(i=0;i<mask_size;i++) // count number of 1's
            m+=valid[i] & (flag != in[i][b-1]);
        where (flag) {
            n+=m;
        } else {
            n-=m;
        }
    }

    out[0] = (n<ByteImage(median_pos)) ;

    return ByteImage(out) ;
}
```