Technical Report

UCAM-CL-TR-892
ISSN 1476-2986

Number 892

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Pipelined image processing for pattern recognition

A. Daniel Hall

July 2016

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted October 1991 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Queen's College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Preface

I would like to thank my supervisor Neil Wiseman for his invaluable help and guidance during my research and the other members of the Rainbow Graphics Group for their support. I would also like to acknowledge the support of Professor Needham as head of the Cambridge University Computer Laboratory. I am grateful to Dr. Henry S. Baird of AT & T and Professor Per-Erik Danielsson of Linkoping University for providing material referenced in this thesis. The benchmark test of the Kurzweil K-5200 OCR machine was performed by Leslie Plumb of Xerox Imaging Systems.

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration.

I hereby declare that this dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university.

I further state that no part of this dissertation has already been or is being currently submitted for any degree, diploma or other qualification.

Summary

Image processing for pattern recognition is both computationally intensive and algorithmically complex. The objective of the research presented here was to produce a fast inexpensive image processor for pattern recognition. This objective has been achieved by separating the computationally intensive pixel processing tasks from the algorithmically complex feature processing tasks.

The context for this work is explored in terms of image processor architecture, intermediate-level image processing tasks and pattern recognition.

A new language to describe pipelined neighbourhood operations on binary images ('PiNOLa': Pipelined Neighbourhood Operator Language) is presented. PiNOLa was implemented in Modula-2 to provide an interactive simulation system ('PiNOSim': Pipelined Neighbourhood Operator Simulator). Experiments using PiNOSim were conducted and a design for a topological feature extractor was produced.

A novel algorithm for connected component labelling in hardware is presented. This algorithm was included in the PiNOSim program to enable the component labelling of features extracted using the language. The component labelling algorithm can be used with the topological feature extractor mentioned above. The result is a method of converting a binary raster scan into a stream of topological features grouped by connected object.

To test the potential performance of a system based on these ideas, some hardware ('GRIPPR': Generic Real-time Image Processor for Pattern Recognition) was designed. This machine was implemented using standard components linked to a PC based transputer board. To demonstrate an application of GRIPPR an Optical Character Recognition (OCR) system is presented. Finally, results demonstrating a continuous throughput of 1500 characters/second are given.

Contents

List of Figures

Chapter 1.

Introduction

Image processing for pattern recognition is one of the most stimulating subjects in science. Historically, image processing has often been at the fore-front of computer theory and practice. Philosophically, image processing provides a link between the technological products of our society and an appreciation of the working of our minds. Commercially, image processing provides the basis for computer technology to be adopted in complex real-world applications.

1.1 The image processing problem

Real world image data is often characterised by two things:

- An enormous amount of sensory data: the weakness of the von Neumann machine is the memory access bottleneck; a simple function may require several memory accesses, and processes which could be performed in parallel must be performed sequentially. The same simple function implemented in dedicated hardware can be much faster and use much less silicon and interconnect, allowing for several functions to be performed in parallel without excessive cost. The resulting speed is the sole justification for image processing hardware, as any digital circuit can be simulated on a von Neumann processor, given time and memory.

- Interpretation of sensory data is algorithmically complex: specialised image processors tend to be bad at coping with algorithmic complexity. This is because specialised image processors are normally image-parallel machines which are optimised to perform low-level sensory manipulation. However, von Neumann sequential processors are ideally suited to algorithmically complex processing as a language hierarchy can easily be constructed; micro-code, machine-code, assembler, programming language, application.

In the book 'Intermediate-Level Processing' edited by M.J.B.Duff[1] image processing is characterised as having three levels; low, intermediate and high. The low-level processing transforms the original n by n pixel image into another n by n array of

values. Intermediate-level processing transforms this array of values into a list of features. This list of features is then processed at the high-level to interpret the contents of the image.

A fast throughput is attainable at the low-level of processing by adopting some form of image parallel processing. This is normally achieved by performing neighbourhood functions which transform the value of an image pixel based on its original value and the values of its neighbours. Such neighbourhood operations have a very simple mapping on to cellular or pipeline image processing hardware.

At the high-level of processing there is normally much less data than at the low-level. Thus the throughput of the high-level of an image processing application can often be matched to the low-level throughput without resorting to massively parallel processing.

The intermediate-level of processing accesses all the n by n transformed image data, but the task of generating topological features requires non-local processing. Therefore there is no simple mapping of intermediate level tasks such as topological feature extraction or component labelling onto image parallel hardware. Thus it is the intermediate-level of processing which can be the potential bottleneck in many image processing applications. One way to increase the throughput of the intermediate-level is to provide specialised intermediate-level hardware.

1.2 Structure of the thesis

This thesis touches on many broad areas of research; image processing, computer architecture, pattern recognition, optical character recognition. The intention has been to integrate work in all these areas to produce a complete demonstration of an image processing application. This could only be achieved by adopting a structured approach to the research. The chapters of the thesis mirror this structure as each phase of work produced important results. Briefly, the phases of the research were; background research, initial experiments, simulation (design/implementation), experiments, system design, system implementation, results.

Several threads run through the thesis structure; The development from ideas about image processing architecture to physical hardware implementation. The development of feature extraction and component labelling algorithms. The development of a pattern recognition approach matched to the new hardware and algorithms. These threads are brought together in a specific OCR application which demonstrates the throughput of the complete system.

The content of the following chapters is given below:

- Chapter 2. Background and Motivation: This chapter surveys the relevant parts of the image processing field. Sections are included on image processing architecture, image processing tasks and pattern recognition. Finally an argument providing the motivation for research is outlined.

- Chapter 3. Initial experiments, PiNOLa specification and PiNOSim implementation: This chapter describes the early stages of research and the development of a new language, PiNOLa; Pipelined Neighbourhood Operator Language, to describe a class of hardware dedicated image processors. The implementation of an interactive PiNOLa simulation, PiNOSim; Pipelined Neighbourhood Operator Simulator, is also described.

- Chapter 4. Experiments with PiNOSim and feature extractor design: This chapter explores the use of the PiNOSim program. A new deterministic feature extractor is introduced and its characteristics are discussed.

- Chapter 5. Component labelling and GRIPPR design: A new algorithm for real-time component labelling of continuous images is expounded. This algorithm is then integrated with the feature extractor introduced in chapter 4 to produce a design and simulation of a Generic Real-time Image Processor for Pattern Recognition (GRIPPR).

- Chapter 6. GRIPPR Implementation: The specification of the hardware and the software components of the system is given in the context of an A4, 300 d.p.i., OCR task. The GRIPPR hardware and software optimisation and implementation is then described. Finally some statistics revealing the computational loading of the components of the GRIPPR system are presented.

- Chapter 7. Pattern Recognition and OCR: In this chapter a simple method for pattern recognition is presented. This method is implemented in an application specific OCR system. The recognition and learning processes are discussed as well as the higher level OCR dependent processing required to format lines of text etc. Results generated during the learning process, which characterise the pattern classification performance of the system, are presented.

- Chapter 8. Results and Conclusions: This chapter starts with a look at the final system configuration. Some relative statistics for a commercial OCR system and the GRIPPR-OCR system are given for comparison. Finally, some areas for further investigation are discussed and some conclusions are drawn.

Chapter 2.

Background and Motivation

In this chapter a broad overview is given of the different aspects of image processing for pattern recognition. An argument is also developed which attempts to justify some of the features that might be useful in a new system.

2.1 An image processing model

The purpose of image processing and pattern recognition is to convert sensory data into symbolic representations which can then be processed at a semantic level. Processing at the extreme ends of this transformation appears to be well understood. At the sensory level algorithms for such tasks as spatial filtering and dynamic thresholding are well researched. This research is supported by orthodox mathematical tools such as communications theory and Fourier analysis. At the semantic end of the transformation lies the logical processing performed by standard computational languages.

The area between the extreme ends of this process is less well understood and can be referred to as the 'Conceptual Gap'. This consists of the intermediate-level image processing mentioned in the introduction as well as the higher-levels of pattern classification and contextual processing. Why does the Conceptual Gap exist? Obviously, the enormous data processing requirements inherent in many image processing tasks mean that available computational resources often restrict the types of application that are feasible. However, even given sufficient raw processing potential we may still find that crossing the Conceptual Gap is far from easy. One explanation for this might be that the algorithms required to cross the Gap are numerous and not well contained, i.e. each depends on the other in a highly complex way. This is an argument sometimes put forward by Neural Net protagonists, i.e. that some problems may be so hyper-complex that only a self-organising, self-learning machine can perform them[2]. Another approach proposed by M.J.B.Duff is to design parallel architectures which could cope with complex interaction at the higher levels.

- *"The picture that emerges is of a form of pyramid in which inter-processor connectivity and processor complexity both increase in passing from lower to higher levels"* M.J.B.Duff [3]

The conventional/engineering approach has been to cross the Conceptual Gap for simple applications and then slowly attempt more complex related applications evolving algorithms, technology and strategies along the way. The history of OCR is a demonstration of the success of this conventional approach. Starting with very simple template matching of special type faces in fixed sizes the techniques have evolved to the present sophisticated feature and context driven multi-fount, variable-scale commercial recognition machines.

Two important simplifications often characterise the conventional approach to crossing the Conceptual Gap; bottom-up processing and processing stage independence:

- **Bottom-up processing:** This is the basic paradigm of many image processing and pattern recognition applications. The idea is that there is a one way flow of information and control, as in Fig .2.1, from sensory data to semantic processing. This is by no means the only paradigm; for instance, some applications make use of coarse visual information to focus attention on interesting sections of the image where the detailed image is processed[4]. The human visual system in particular is different in this fundamental respect. Various experiments have shown that our ability to direct and orientate the eye in relationship to a scene is basic to our ability to locate, recognise and learn visual patterns[5]. The thing that necessitates and enables this top-down process in humans is the active nature of our vision. The ability to actively locate objects for recognition within our visual environment is both an added complexity and a simplification. The added complexity is in arbitrating the control of the eye between the different reflex, involuntary and voluntary mechanisms that are known to control the movement, focus and aperture of the eye. The simplification is that we can look at an object in a way which makes it look 'right', i.e. the image of the object can be located on the retina in such a way as to facilitate recognition. Contrast this with the passive vision system implicit in the bottom-up approach, here objects for recognition are located at the intermediate levels of processing with no control over the resolution or exposure of the imaging system hardware.

- **Processing stage independence:** The other simplification often made is that the steps shown in Fig.2.1 are independent. This simplification limits the algorithmic complexity caused by close interaction between stages. However, a number of perhaps useful feedback mechanisms therefore have to be omitted, e.g. the pattern recognition process could tune the parameters of the sensory transformation dynamically, or the feature extractor could select a method of recognition based on the feature types and density, etc. etc..

A typical example of the bottom-up, stage independent approach is shown in Fig 2.1. The exact position and nature of the stages in the model depends on the algorithms used, e.g. some approaches will place the feature extraction stage before the object location stage. The stages of the model are:

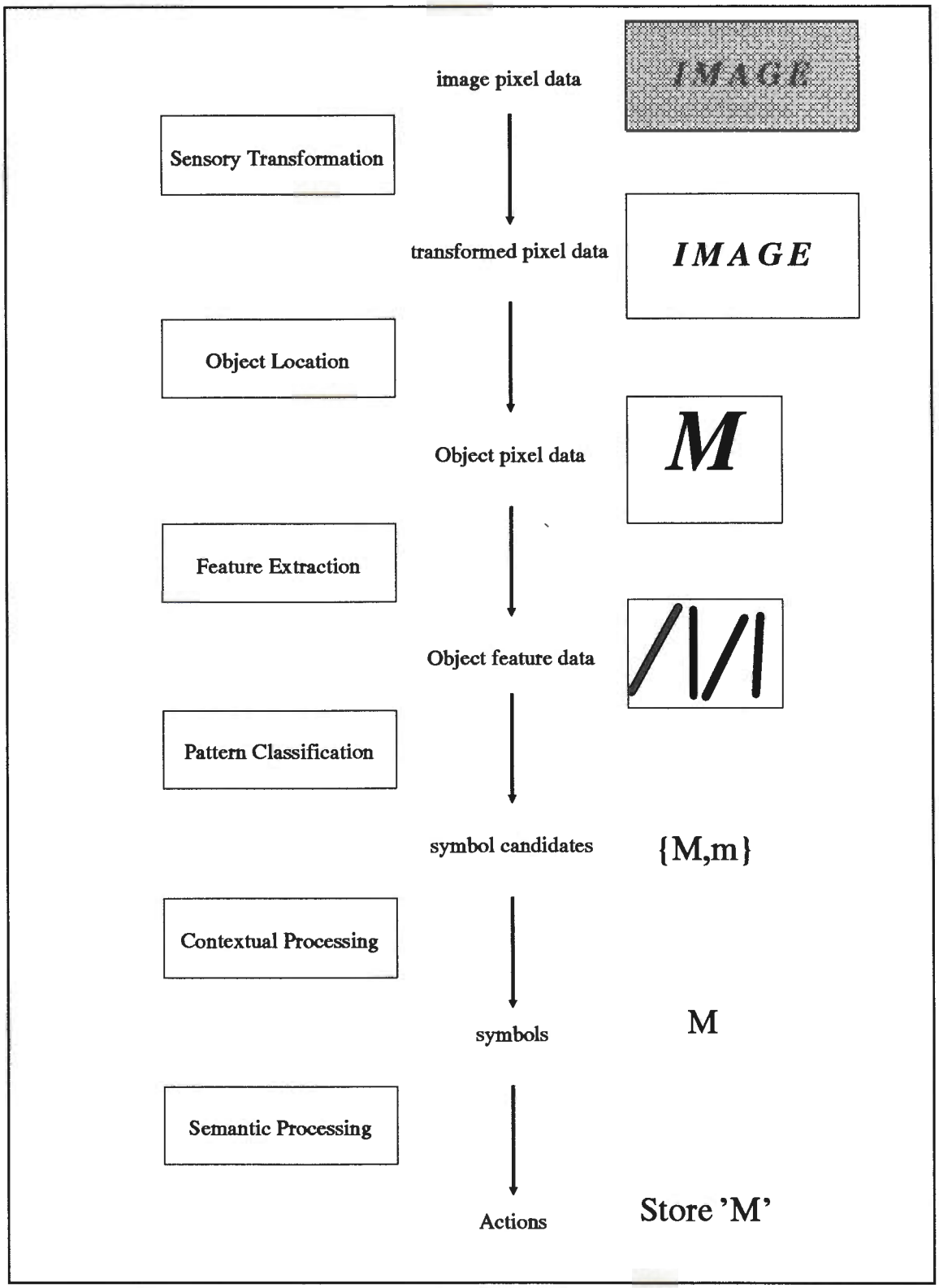


Fig 2.1: bottom-up, stage independent image processing model

- **Sensory Transformation:** This type of processing is often performed on multilevel image data, to perform such tasks as filtering or contrast enhancement. The purpose of this level of processing is generally to improve the intelligibility of the image and to reduce the information required to describe the image. The output from this stage of processing is often a binary image.

- **Object Location:** Before recognition, objects within the image must be located, this process is often referred to as segmentation.

- **Feature Extraction:** This stage of processing transforms pixel data into feature data. Features generally describe some facet of the image, e.g. a line, and have parameters, e.g. line length and orientation.

- **Pattern Classification:** When an object has been located this can be mapped onto one or more possible symbols.

- **Contextual Processing:** Information which is not local to the pattern classification object can be used to determine which if any of the symbols passed from the pattern classification stage are plausible.

- **Semantic Processing:** The symbolic information describing the image is processed to perform some useful action.

This simple model of image processing allows one to split different stages in the processing pipeline between physically different processor architectures optimised for their particular function.

2.2 Hardware image processing architectures

Within the context of the type of image processing model shown in Fig.2.1 various processor architectures can be adopted at one or more stages of processing. A brief guide to the types of processor used for image processing is therefore appropriate.

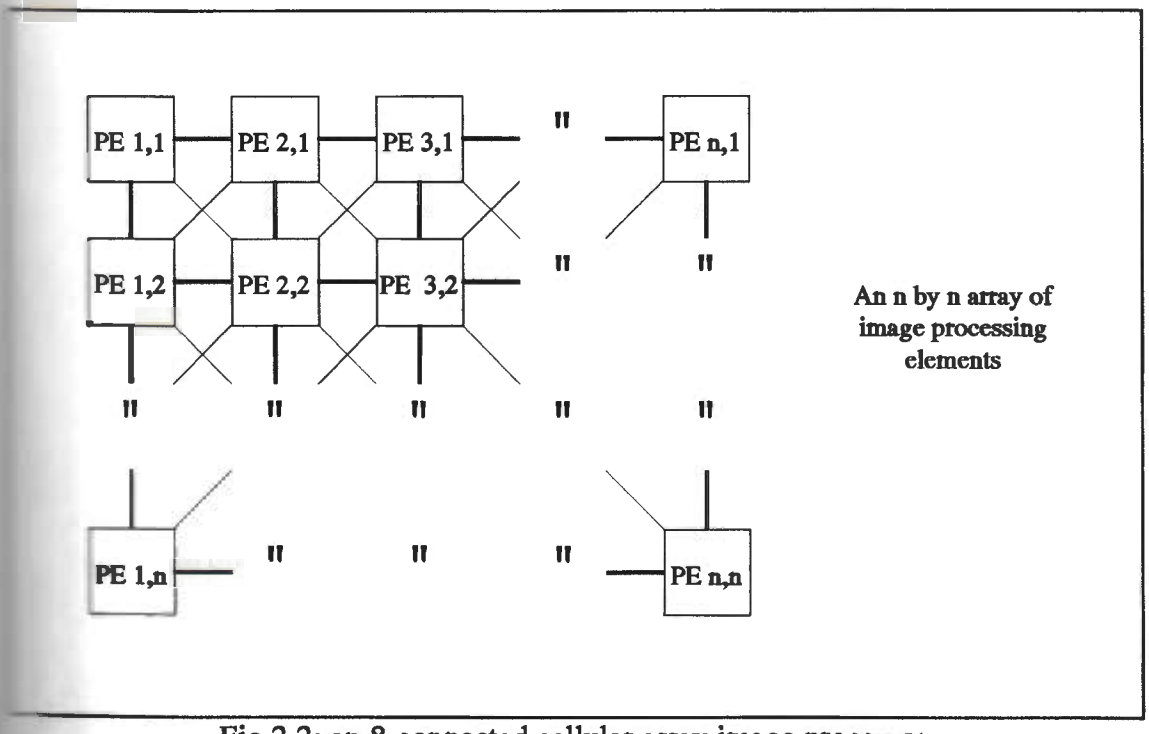


Fig.2.2: an 8-connected cellular array image processor

2.2.1 Cellular arrays

A cellular array image processor is formed from a two dimensional array of processors as in Fig.2.2 where there is a geomorphic mapping between an image element and a physical processor. Each processor is linked to its immediate neighbours in a hexagonal or square grid. Generally cellular arrays correspond to the SIMD (Single Instruction, Multiple Data stream) architecture where one instruction is broadcast to all Processing Elements, each P.E. acting on a different part of the image data.

The motivation for producing these architectures is that they are highly optimised for performing algorithms based on neighbourhood logic. In each clock cycle such a processor can apply a neighbourhood function to the entire image in parallel. Such processors are therefore ideal for the low level stages in the standard image processing model such as sensory transformation and feature extraction.

The first detailed work on cellular arrays dates from 1959 when Unger[6] proposed a processor based on a 4-connected array of 36 by 36 processing elements. This work laid the foundations not only of cellular array architecture, but also of neighbourhood logic. The basic principle of assigning P.E.'s to image pixels and linking each P.E. to its neighbourhood so that neighbourhood functions could be executed in parallel was established by Unger's design. Unger also foresaw the need for non-local image processing particularly with respect to determining the connectivity of the image and proposed specialised 'link circuits' which could propagate information beyond the immediate neighbourhood quickly.

The implementation of Unger's ideas had to wait for improvements in digital circuit technology. Physical hardware implementations of array processors did not emerge until the mid 1960's with the ILLIAC III computer developed by McCormick[7], though design problems meant that this processor never found commercial application.

In the 1970s the cellular array architecture based on CLIP 2[8] and CLIP 3/4[9], developed at University College London, produced processors with a large number of discrete processing elements and these have found commercial application particularly in Biomedical recognition[10].

In the 1980s the evolution of programmable pipelined processors performing neighbourhood logic led to criticism of the cellular array architecture in terms of its computational efficiency[11]. This criticism should be seen in the context of the greater image sizes starting to be processed at that time due to improvements in digital imaging technology in the 1970's and 1980's.

Limitation of cellular arrays

In theory as image resolution increases the potential advantage of cellular arrays also increases. However, in practice the architecture soon becomes limited by its I/O bandwidth. A 1024 by 1024 cellular array could perform 100 neighbourhood operations on a 1024 by 1024 image in only 100 clock cycles. However, even if a whole line could be entered into the array simultaneously this would still take 1024 clock cycles to fill the array giving a processor utilisation of under 10%. The practical problems of having over a million processors also means that almost no real world

application could justify the extreme cost involved.

In response to this shift in technological balance different cellular strategies have been adopted.

Tiling

Tiling has been adopted to split pictures into smaller sub-images so that the I/O bandwidth can be balanced with processor speed. However, there are some drawbacks to tiling; it increases the hardware and software complexity and some non-local functions which rely on global propagation of signals may be poorly defined at the tile boundaries.

Embedded sensor arrays

The other development in cellular array application has been to incorporate the array into imaging devices to perform the processing of images actually on the CCD silicon[12][13]. This approach overcomes the input bandwidth limitation, as images are already mapped onto physical cellular arrays. Provided a reasonable compression is afforded by the cellular processing the output bandwidth may not be a problem.

A more general criticism of cellular arrays in the system context is that their speed may not be well matched to the speed of other components in a system. If the task is complex, such as many pattern recognition tasks, then the low-level neighbourhood functions executed efficiently by the cellular array may form only a limited proportion of the processing required. Thus the cellular array may be idle whilst higher non-parallel algorithms process the output of the array. This problem has led to research into applying cellular array processors to higher level tasks, thereby equalising the throughput of the different components of a system. However, whereas a cellular array is ideally suited to performing neighbourhood logic, efficiency falls as intermediate-level, non-local algorithms are attempted[1]. Even such comparatively low-level tasks as component labelling are not performed efficiently on cellular logic arrays[14].

To more neatly match computational resources with application requirement and to allow for higher-level data dependent algorithms various image parallel architectures not bound by the strict 'cellular' paradigm are being developed. On the one hand in the evolution of existing cellular architectures such as the CLIP series to incorporate non-local data communication and linear rather than 2-dimensional arrays as in the CLIP7A[15]. On the other hand there is the continuing basic research into different types of array processor; SIMD, Systolic, Wavefront and MIMD. It is becoming increasingly clear that as applications become more sophisticated parallel processors will have to become more flexible, allowing for different types of parallelism and processor topologies to be applied to different stages in an application process[16].

2.2.2 Pipeline image processors

A pipelined image processor based on neighbourhood operations is architecturally very easy to implement. A raster scan of the image is fed into the neighbourhood

operation unit which constructs a window on the input image by storing some image lines in shift registers. A function is then applied to the image window which generates an output pixel. For each pixel of the input image the shift registers are clocked and the window is moved to generate the next output pixel.

In Fig.2.3, a three stage pipelined 3x3 neighbourhood operation applies an erosion function to the input image.

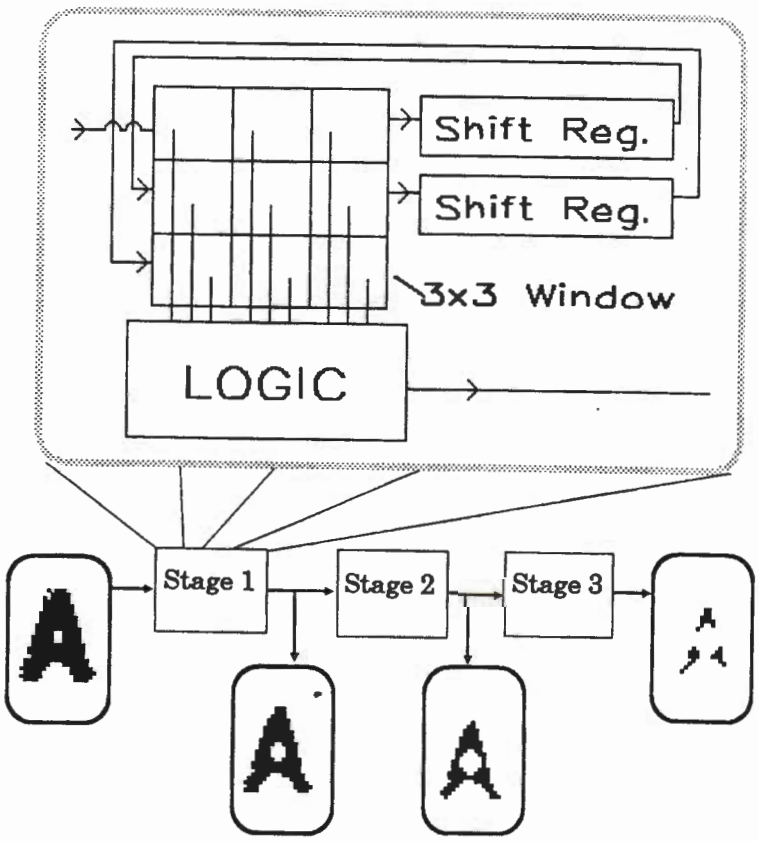


Fig.2.3: a three stage pipeline image processor

The main advantage pipelined processors have over cellular arrays is that for functions of limited complexity they require much less hardware to implement. Again, a pipeline image processor can be effectively targeted at the lower level tasks of sensory transformation and feature extraction.

The basic pipelined picture processing approach is attributed to Golay in 1965[17], his ideas led to the development of the Golay Logic Processor (GLOPR). In the mid 1970's more sophisticated architectures based on multiple image planes where produced, such as the BIP (Binary Image Processor) developed by S.Gray[18] and later PICAP developed by B.Kruse[19][20]. These relatively complex machines were followed by simpler modular architectures e.g. the Cytocomputer architecture developed by Sternberg[21][22] in the late 1970's and the UPIC[23] and PIFEX[24] architectures of the 1980's.

Early Pipelined Processors

The Golay processor was based on a hexagonal neighbourhood as this was considered to be better suited to connectivity preserving transformations. Whilst the architecture Golay was considering was essentially cellular this was achieved by successively applying the processor to all the neighbourhoods in the image. GLOPR was applied to biomedical image processing problems.

The BIP was a more complex machine than GLOPR, which was optimised to perform template matching operations for OCR. BIP had two picture planes and a function could be specified on the basis of the pixel values corresponding to the same location in both planes.

PICAP

PICAP - developed in the university of Linkoping, Sweden - was a general purpose programmable pipelined image processor. Even in its early specification PICAP was a very complex machine[19], e.g. PICAP allowed a function to be the product of its 3x3 neighbourhood and the state of parallel picture planes a type of processing which was inspired by BIP. Facilities were also included for component labelling and iterative and interrupted functions. The high complexity of PICAP should be seen in the terms of the high cost and low integration of electronic components of that time. The temptation was to squeeze every last drop of efficiency out of computational hardware. This trend is mirrored in the development of the Complex Instruction Set Computers of that time.

The complexity and sophistication of the PICAP hardware led to the development of specialised algorithms which are highly dependent on the particular facilities of the PICAP architecture.

For example, a component labelling algorithm was developed which could label one simply connected object in each pass of the image by use of a special interrupt function. This function interrupts the normal local neighbourhood operation of the processor to fetch a new component label from a non-local stack whenever a new un-labelled connected object is located.

The resulting algorithm is inefficient but it demonstrates how solutions to higher level problems in an image processing task have to be tackled in the low-level image processor if the system throughput is to justify the expense of the image processing hardware. Unfortunately, the inherent inefficiency involved in calculating non-local functions on a neighbourhood processor often compromise the hoped for level of performance.

Cytocomputers

The real breakthrough with the cytocomputer architecture was its hardware modularity. Whereas PICAP tried to maximise the utilisation of a discrete number of stages by specifying complex functions, the cytocomputer architecture sought to make available a physical neighbourhood processor for each stage in an image processing task by simplifying and thus reducing the cost of each stage. This development was clearly prompted by the emergence of LSI and VLSI technologies

in the late 1970s and early 1980's. The cytocomputer architecture included separate types of modules for 2-dimensional (binary) operations and 3-dimensional (multilevel) convolutions.

The efficiency of the Cytocomputer architecture, for the type of problem the authors where considering, led to the conclusion that:

- *"Certainly, parallel arrays will continue to be designed and built; however, any advantages to the users of such systems are in question."* R.M.Lougheed, D.L.McCubbrey [11]

PIFEX

The programmable pipelined processor based on multilevel and binary neighbourhood operations reached technological maturity with the PIFEX architecture. PIFEX adopted VLSI techniques to produce low-cost modules which could perform 12-bit per pixel image processing of high resolution images. Rather than include different types of modules as in the Cytocomputer architecture, PIFEX incorporated two multilevel convolvers, a table-lookup unit and a binary neighbourhood comparison operator in each module. PIFEX is used commercially for such tasks as satellite image processing.

Dedicated systems

As well as relatively expensive programmable general purpose pipelined image processors a number of hardware dedicated systems have been devised e.g.[25][26][27][28]. There are two main advantages that dedicated systems have over programmable systems:

- They can be highly optimised to perform particular image processing tasks. Thus complex algorithm specific structures can be implemented which would not be appropriate in a programmable system.

- Without the overhead of making the processor programmable the hardware implementation can be more efficient e.g. a dedicated system might be faster, cheaper and physically smaller than a programmable system performing the same tasks.

However, the limiting factor on the application of such dedicated systems has been a lack of generally applicable algorithms which can be implemented. Normally image processing algorithms are tailored to the requirements of particular applications, making the design of dedicated systems less cost-effective.

For instance in their paper "An automatic Wafer Inspection System Using Pipelined Image Processing Techniques", Haruo Yoda et al present an image processor based on pipelined neighbourhood operators dedicated to that particular task[25]. The image processor is highly optimised, using forward and reverse sequential/feedback operations not normally available in general purpose hardware. The image processor also uses a fixed number of erosion/dilation stages to achieve image smoothing in the context of the sorts of noise produced by their imaging system. The resulting system has found application in RAM manufacture yield analysis, replacing human operated fault diagnosis of failed components.

Whilst being highly practical and cost-effective dedicated non-programmable pipelined image processors have not so far found general application, as the exact nature of the processing function required normally depends greatly on the system task. However, if algorithms which could be applied to a variety of tasks could be developed then a generic image processor could be formed by a dedicated pipeline image processor.

2.2.3 Neural nets

There are many types of artificial neural network. They generally describe a processor based on a number of simple elements or 'neurons'. Neurons often have a boolean on/off output which is triggered by summing the inputs to the neuron and comparing the result with a threshold value. More sophisticated neural nets simulate the variable pulse rates found in biological neurons by propagating analog outputs. The sensitivity of a neuron to a particular input is generally given an analog weight. The connections between neurons could be very complex with connections between every neuron in the network. Often this is simplified into fully connected layers of neurons with one way connections between neurons in different layers.

The learning process is normally implemented by altering the weights attached to the neuron connections. There are many ways to control this process which usually rely on increasing a neuron's sensitivity to those inputs that are 'on' when a specific pattern is presented. Neural nets normally have to be trained to associate a given input pattern with a particular response. Though rules also exist for allowing neurons to automatically cluster the input patterns into classes without external identification during the learning process.

Neural net strategies have been proposed for all levels of image processing, the justification for this being the example of animal vision systems where all processing is achieved by neural type structures.

There are strong links between biological and computational research into neural structures and visual pattern recognition. From the biological perspective this is because photo-receptors are amongst the easiest to stimulate in a controlled environment, thus much of the research into neural mechanisms has been targeted at the visual structures of biological systems. From the computational perspective one of the key motivations for neural research has been to understand and replicate the high performance of the human visual system.

One of the earliest attempts to produce an artificial neural net was the Perceptron proposed by Rosenblatt[29]. The Perceptron was based on three layers of units corresponding to different types of cell function. The first layer of sensory units (S-units) transmit an 'all-or-nothing' binary impulse on the basis of the the image pixel value presented to each unit. The S-units therefore constitute a sort of simplified retina. The S-units connect to a set of association-units (A-units), each S-unit connecting to a different but overlapping set of A-units in a random way. Some of the connections from the S-units to the A-units are excitatory and others are inhibitory. The A-units transmit a binary value to the response units (R-units), on the basis of a fixed threshold of an algebraic sum of the excitatory and inhibitory inputs to it. Again the A-units connect to different, overlapping random sets of R-units.

The output of the R-units is the output of the machine and is also fed back into the machine in the form of inhibitory connections to those A-units which do not connect to it. Learning is by reinforcement of those A-units and R-units that are stimulated, in other words once an R-unit becomes or is made active to a given pattern it will become active more readily the next time that that pattern or similar related patterns are presented. The reinforcement rule is intentionally vague[30] but in the case of the Mark I Perceptron and in most other implementations it is implemented by variable weights assigned to the inputs of the A/R-units. When an A/R-unit is made active the weighting of those excitatory inputs which are on is increased, making the subsequent triggering of the unit more likely.

Computationally, the Perceptron has a highly parallel structure which seemed to offer the prospect of much greater processing power than the von Neumann sequential programmable machines starting to find application at that time. However, the processing potential of the neural/connectionist model remains largely untapped. Criticism of Perceptrons has often centred on their inability to perform image processing tasks such as determining connectivity, i.e. Minsky-Paperts 'Perceptrons'[31]. This criticism may have been unfair for two reasons. First, it is not clear that connectivity forms a necessary basis for visual pattern recognition[32]. Second, the Perceptron was a very simple even stylised neural net, in reality the retina and visual cortex include many specialised cells which may perform some of the tasks not achievable by the simple Perceptron.

To address these questions, and fuelled by the underlying physiological research into vision, a lot of effort has gone into emulating the behaviour of specialised mechanisms in the human brain. Thus very advanced models and sometimes even hardware exist which mimic human retinal cells[33], feature extraction cells[34] and binocular vision[35].

There is however, a more general problem with neural nets, which is that they do not cope well with rule based algorithmic as opposed to sensory complexity. Thus a simple rule such as 'select the object of medium size' cannot easily be programmed into a limited artificial neural net. Many useful tasks contain a great deal of algorithmic complexity; ignoring this can lead to machines that cannot make use of readily available rule-based knowledge about how things can be done.

There are two ways around this problem:

- One could build a complete neural based system which is capable of all levels of semantic as well as sensory processing. Such a system would be able to learn by instructions as well as experience and would be able to follow rules to perform tasks such as counting or sorting. However, such a system may not be attainable without resorting to levels of neural complexity not presently achievable.

- The other more practical solution for present day systems is to integrate the neural net with a programmable component to perform the algorithmically complex tasks. The question is, what is the proper division of tasks for a given system?

Given the development of neuroscience from the study of sensory processing, it is no surprise that neural nets are often presented as pre-processors and recognisers to

be controlled by a higher-level programmable component. The assumption that is implicit in this approach is that algorithmic complexity is confined to the higher semantic levels of processing.

The validity of this assumption very much depends on the particular system being implemented. In the context of an OCR system a sophisticated neural pre-processor may be inappropriate. This is because algorithmically complex tasks such as object location and separation depend on the processing of the data used for pattern classification e.g. if there is a ligature joining two characters we may wish to divide the features describing the object in two[36], which is algorithmically complex.

Generally, if we wish to make the greatest use of rule based information about how a task may be performed then this can be achieved by confining the non-algorithmically programmable components (neural-nets) of a system to the core pattern classification task. This limitation on simple neural nets was recognised by Rosenblatt:

"Statistical separability alone does not provide a sufficient basis for higher order abstraction. Some system, more advanced in principle than the perceptron, seems to be required at this point." Rosenblatt [29]

Whether or not one adopts a neural strategy, the justification for research into any technique is that one cannot apply an algorithm until it has been discovered. Work currently in progress on emulating the functions of the retina and cochlea give a profound insight into the sort of algorithms which are required to perform complex tasks such as continuous speech recognition or stereoscopic vision.

2.2.4 von Neumann machines

The von Neumann architecture is particularly suitable for intermediate and higher levels of feature processing where the operations tend to be algorithmically complex.

Massively parallel architectures may be capable of executing 1,000 or even 10,000 times the number of instructions per second that can be performed on a sequential processor. However, such statistics give little insight into the practical consequences of parallel versus sequential processing for computing systems and real world applications. In several aspects the sequential von Neumann processor turns out to be better than a simple comparison of integer operations per second might suggest.

Computational aspect

Utilisation: The sequential processor normally achieves 100% utilisation of its processing potential, i.e. given a problem the processor executes instructions at its maximum rate until the problem is complete. On a parallel processor however, utilisation is often limited by the innate degree of parallelism of the problem. Also, if a parallel processor cannot perform all the steps in a task (e.g. an image-parallel processor cannot efficiently perform high level pattern classification tasks) then the utilisation of the parallel hardware will be limited by the throughput of other components of the system.

Inter-process communication: Procedures executed on a sequential processor can communicate with each other at the maximum processor memory bandwidth through shared variables. When a process is split between many physical processors the inter-process bandwidth is normally considerably lower than the processor local bandwidth. The MIPs statistic is only dependent on local memory bandwidth. Such comparisons therefore give a pessimistic view of the relative performance of the sequential processor. Maintaining a complex global data structure, as is useful when determining the connectivity of images, is typical of the sort of problem which would be limited by the inter-processor bandwidth on a parallel processor

Technological aspect

Sequential processors are used in almost every sphere of human activity. The implication of this general applicability, whether *a priori* it is deserved or not, is that a great deal of human resources have gone into making them extremely efficient. One could argue that other technologies might perform better in certain situations given the same level of investment. However, the point is that investment in a technology is conditioned by its general application. When application-specific processors are designed the state of the art silicon technology adopted in the latest sequential processors may not be available. Also the lag time of actually implementing the processor means that there may be a large handicap for the application specific processor in its underlying technological base.

Programming aspect

The main advantage of sequential programming is the ease with which a language hierarchy can be constructed. The reason for this is that a sequential process explicitly defines when information is available, i.e. when a statement is executed the proceeding statement is guaranteed to already have produced its result. When parallel algorithms are introduced to exploit parallel processing hardware the temporal complexity of scheduling the information flow in a process is exposed to the programmer[37]. Also, for whatever reason, a great deal of application specific image recognition know-how is embedded in sequential software. The need is to develop approaches which do not require the re-writing of all this sequentially defined programming effort.

Though sequential processors have some very good characteristics it is of course possible to speed up any task for a fixed level of technology by adopting some form of parallelism. The question is can we retain the concrete advantages of sequential processing whilst reaping the benefit of some perhaps limited parallelism? Image processing is characterised be a few computationally intensive tasks which could easily be performed in parallel hardware. In particular neighbourhood functions are easy to implement efficiently either in cellular arrays or pipeline processors. However, the high cost of programmable cellular arrays and pipeline processors would limit the application of such an approach. A dedicated processor could be implemented very efficiently as neighbourhood operators are inherently simple to construct. However, in order to justify the design cost of a dedicated approach the functions to be executed should be generally applicable, otherwise the technology would risk becoming redundant as processor speeds increase and the particular 'faster' application changes.

Another path to parallelism, whilst retaining the advantages of a basically sequential approach, is to adopt 'coarse-grained' or 'process' parallelism. This approach simply divides a sequentially defined program into reasonably self-contained sub-programs which can then be executed on separate sequential processors in parallel. The advantage of this approach is that the program can be developed as a series of concurrent processes and these can then be implemented on different numbers of physical processors to provide a range of cost/performance options at any particular level of sequential processor technology. The main draw-back of this approach is that the amount of parallelism which can be achieved is severely limited, also care has to be taken to physically place processes on processors in such a way that the loading of processors is balanced and a high level of utilisation can be achieved. The sort of image processing model shown in Fig 2.1 fits well onto the coarse grained approach as the different levels of processing can be reasonably independent with a simple ordered flow of data from low-level processes to high-level processes.

2.2.5 Technological perspective

The question of which architecture to use for image processing is not simply a product of their theoretical merits, but also of the underlying technological base. For example, as imaging technologies improve there is a possibility for improving the performance of pattern recognition by increasing the resolution of the image. However, this has the effect of altering the balance between low-level pixel processing and higher level feature processing. This is because the feature density will remain roughly constant for a given task even though more image data is generated as the marginal cost of imaging falls.

Meanwhile the overall performance of von Neumann computers has been doubling roughly every five years. State of the art RISC processors will soon be performing over 100 MIPs, the T9000 next generation transputer is designed to perform over 150 MIPs. Coupled with this is a steady increase in memory packing density leading to faster more memory intensive algorithms.

The effect of these changes, higher resolution and faster processing, is to bring the feature generation throughput of hardware and the feature processing throughput of software more into balance. Previously hardware image processors have had to perform some of the higher level tasks, to which they were not well suited, in order to achieve a system throughput commensurate with their low-level image processing speed.

Within this context the need is increasingly for image processors which only perform the pixel processing dependent tasks. Firstly, because this is the most computationally intensive part and therefore the potential bottleneck for image processing applications. Secondly, because all the feature processing for many applications can now be performed at real-time speeds on simple microprocessor pipelines. Allowing microprocessors to perform all the feature processing allows greater algorithmic complexity to be adopted and reduces the cost and complexity of the hardware image processor. Ofcourse, there will continue to be some tasks that will require fine-grain parallelism at every stage, perhaps supported by massively parallel, topologically flexible, architectures such as the Connection Machine[38].

2.3 Intermediate-level image processing tasks.

To get an idea of what type of processing architecture and strategies have to be adopted at the intermediate-level it is necessary to look at the two main tasks normally associated with this level; connectivity determination and feature extraction.

2.3.1 Connectivity determination

A connected object is made up of a set of neighbouring foreground pixels. The rule used to determine the pixels neighbourhood (4 or 8 connected) determines the type of connected object.[39]

Is connectivity important?

The Minsky-Papert criticism of neural nets[31] was partly based on the inability of the Perceptron to determine the connectedness of images. To counter this neural net protagonists point to the difficulty even humans have in recognising complex connected objects.

Both points are valid. One cannot expect to train a simple Perceptron to recognise a fish and then expect it to recognise two fish as anything but a mess. This is because it cannot fundamentally recognise that there are two distinct objects in the image. On the other hand if we use connectedness to separate the fish this only works if the fish images do not overlap.

In fact one could argue that locating objects is an algorithmically complex task and that there is no single level of processing between image and pattern recognition output which can determine where an object is located. This is the challenge for image processing hardware, how can we propagate sufficient information so that objects for recognition can be located at different levels?

Having stated that connectivity information is not always sufficient for object location, it is however very useful. Sometimes, as in circuit board analysis connectivity information can be used directly to determine if tracks touch etc.[40]. Sometimes connectivity is usable for object location as in many OCR tasks. The ultimate justification for propagating connectivity information about an image is that this facilitates higher levels of object location. For example if a character breaks into two connected components, provided we recognise the fault, connectivity information allows us to eliminate the information relating to the break and to re-connect the character. This can be achieved by the following type of algorithm:

IF fault in adjacent connected objects
THEN
Merge feature Lists
remove features from overlapping broken ends.
END

Component Labelling

The process of component labelling is one of assigning labels to pixels in an image

by nearest neighbour propagation. These labels are then processed so that the connectedness of the image is revealed.

Component Labelling can be performed on sequential processors[14], pipeline processors[20] or Cellular arrays[41].

The basic raster scan sequential method is quite straightforward[42]. Simply go through the image sequentially assigning new provisional labels whenever a pixel is found whose nearest neighbours have not already been assigned a label. If a pixel has a neighbour that has already been assigned a label then the pixel is assigned the same value as its neighbour.

This process might produce the following result for a simple image containing two connected components.

```
.................
..11..22...33.....
..11..22...33.....
..11..22...33.....
..111111....33....
.................
```

A record is kept of the label values that are assigned to neighbouring pixels, such as the values 1 and 2 in the above picture. Label values are then grouped into a list structure so that label values that belong to the same connected object are placed into the same list.

Finally the provisional value labels are replaced by a list value. The above picture would generate two provisional label lists as there are two connected objects. The final output would be:

```
.................
..11..11...22.....
..11..11...22.....
..11..11...22.....
..111111....22....
.................
```

The main objection to component labelling as a method for resolving connectivity is that it is relatively slow when performed on a sequential processor. This is due to the fact that every pixel in the image must have its label neighbourhood inspected before label assignment. Though this speed limitation can be alleviated by performing the provisional label assignment in pipelined hardware, and by pre-processing the image to reduce the complexity of its connectedness[43] [44].

The other criticism of component labelling is that it is difficult to determine when an object is complete until the image is complete. This is because finding out when a value is no longer being used requires non-local processing. This follows from the paper by Stanely M. Selkow[45] analysing the ability of a linear array of processors to determine connectivity in one pass of an image.

"Although one may have anticipated that the complexity of many of the properties investigated would be less sensitive to the size of the picture, it may be seen that the connectivity property, which is basic to many of the properties studied, necessitates a global computation." S.M.Selkow[45].

Other algorithms for resolving the connectivity of raster scanned images rely on

maintaining a tree structure of either edge points [46][47] or pixel runs [48][14]. Though more complex algorithmically, these methods are more efficient on a sequential processor as fewer physical data items have their connectivity determined. Also, the connectivity can be resolved once the object is complete, because the persistence of the data items is bounded, e.g. when a run is encountered we already know that it is complete. The time taken by these methods can also be justified as they can generate useful topological feature data, in contrast to the component labelling method which generates no useful feature data by itself.

Pixel run method

The pixel run method relies on converting the image into a series of one-dimensional connected elements called runs. Runs are then partitioned into two types, blocks and hinges. A block is any run which is connected to not more than one other run in the line above and the line below. Any run connected to more than one run, above or below is a hinge. The following picture illustrates the decomposition of a connected object into hinges and runs:

```
.............     .............
.............     .............
....XXXXX.....     ....BBBBB.....
..XXXXXXXXX.....   ..HHHHHHHHH.....
..XXX...XXX.....   ..BBB...BBB.....
...XX...XX......   ...BB...BB......
..XXXXXXXXX.....   ..HHHHHHHHH.....
..XXX...XXX.....   ..BBB...BBB.....
...XX...XX......   ...BB...BB......
..'..XXXXX.....    ....HHHHH.....
.............     .............
.............     .............     B=Block,H=Hinge
```

Instead of processing pixel data the algorithm now processes block and hinge data. The number of data items can be further reduced by amalgamating blocks into 2-D structures. The connectivity of the blocks and hinges is represented in a tree structure. The important difference between the pixel run method and the component labelling method is that when the last block or hinge of a connected object is encountered, it is known that the object is complete. With the component labelling method, when the last provisional label associated with an object is encountered it is not known whether that label is unique or whether it has been propagated to some other part of the image.

The relative positions of the hinges and the block/hinge intersections can be used as topologically dependent features for higher level pattern recognition tasks. However, the run-length method cannot easily be adapted to the labelling of other features such as skeletonization features which can be generated in hardware.

Edge point method

The edge point method constructs a chain-coded contour of the image as it is raster scanned. The objects are known to be complete when the contour is closed.

In contrast to the Pixel run method of connectivity determination, the method based on edge points generates little feature data, instead the output of the algorithm is a stream of chain-coded contours. These have the ability to reconstruct the image fully and are therefore useful for image compression, but further processing is required to

generate a higher level description of the image. One advantage of the edge point method is that it is well adapted to multilevel connectivity determination[47].

2.3.2 Feature extraction

Normally, an important step in any pattern recognition task is the transformation of sensory input data into higher level features.

Feature extraction is performed as features generally have two useful properties.:

- There are fewer features than sensory inputs. Thus processing features instead of sensory inputs uses processor time and memory more efficiently.

- Features are better discriminators than sensory inputs. i.e. each feature says 'more' about the object to be recognised than any particular sensory input. This facilitates the recognition process.

Feature extraction has two main drawbacks:

- The transformation from sensory to feature data may itself be highly costly, as it will require computationally intensive sensory processing.

- The feature transformation may be lossy so that not all the sensory data which might discriminate between patterns is present in the feature data. This can lead to ambiguous recognition. Though this has to be seen in the context of a recognition process which may itself be lossy, i.e. unless we are to maintain a database containing an exact template of every instance of the training set, some ambiguity due to the data compression inherent in most pattern recognition approaches is inevitable.

A myriad of very different feature extraction algorithms exist. From simple pixel counting schemes to sophisticated approaches based on skeletonization, Euler numbers, Hough transforms, characteristic loci, Fourier transformation, hinges and runs, etc.

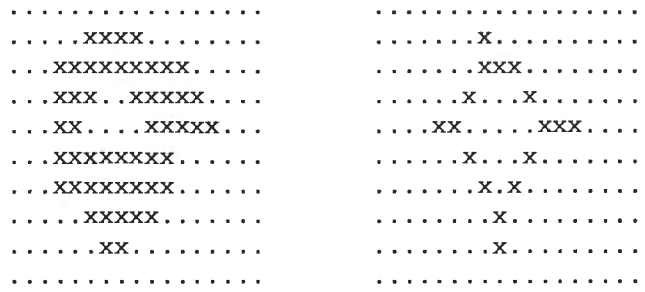
In turn these different approaches are executed on a variety of computational architectures: von Neumann machines, neural nets, cellular and pipeline processors, custom silicon devices or even analog circuits. Feature types can be subdivided into topological and non-topological. Topological features have the property that they fully describe the topology of the input image. This may be useful in itself for recognition or it may be used to reconstruct the image at a higher level as in circuit diagram analysis, e.g.[49].

An important class of feature extraction algorithms is that based on binary operations on the 3x3 pixel neighbourhood. These can be efficiently performed on cellular array or pipeline image processors.

3x3 binary neighbourhood operations can be used to generate a variety of local features describing e.g., edge points, corners, junctions. Topological features can be generated by performing transformations of the image that retain the connectivity of the image. One such class of topological feature extractor is based on performing

skeletonization [50].

Skeletonization is the process of forming the endoskeleton of foreground connected components from an input image. The effect of skeletonization can be seen in the following simple illustration.



There are many issues relating to the exact algorithm used and type of skeleton formed[50]. We will return to some of these issues in the context of pipelined neighbourhood operators in chapter 4.

Skeletonization of an image is accomplished by successively removing edge pixels of the foreground components whilst maintaining the connectivity of the image. This task is performed until only components of 1 pixel thickness remain. One motivation for skeletonization is that if the nodes of the skeleton are used as features these features describe the topology of the original image. These topological features can then be used for direct recognition of different topological classes. The relative geometric positions of these nodal features are also good discriminators for many visual pattern recognition tasks.

The big problem with skeletonization is that it requires an indeterminate number of stages to generate the final skeletonized output. This is because only pixels from the edges of connected components can be removed at each stage and still maintain the connectivity of the components. Therefore it is difficult to make a machine with a reasonable number of discrete stages which can guarantee to skeletonize an arbitrary image, because objects in the image may be very fat so requiring a large number of stages. This problem is exacerbated as imaging resolution increases.

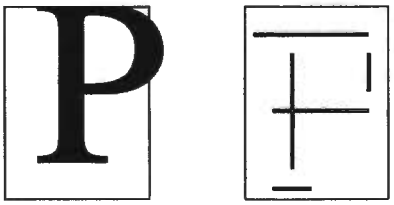
One solution to the problem is to keep the image in memory and repeat the neighbourhood operations iteratively until only skeletons are found in the image. However, this produces a time indeterminate process with high latency between picture input and feature output. This also implies that skeletonization cannot easily be applied to a continuous image as might be generated in an industrial conveyer belt vision application.

Run Length Features

The number of pixels, relative location and connectivity of run lengths in an object are sometimes used as features. This could be the product of a run length connectivity algorithm[48]. Run lengths are often used because images are run length coded for compression reasons anyway. The nature of run length features makes them well suited to the recognition of printed text founts where the aspect ratio and therefore length of run is consistent. Also, it has been demonstrated that one can generate vector data by processing the run length coding of an image [51]. An example of the sort of features that can be generated by this method are given in[52].

Here, features represent lines at different angles in the image placed relative to other lines from the same character.

The above picture shows the decomposition of the letter 'P' into a set of stroke



features. Each feature has some parameters associated with it, i.e., length, x,y position in the bounding box and angle.

The main drawback of run length features is that even if the run lengths themselves are generated in hardware, the run length processing required to generate vectorised data is still quite high. Also, the selection of features which can be generated efficiently does not correspond to those features which may be generated by accessing the pixel neighbourhood directly.

2.4 Pattern recognition

The requirements of the higher-level pattern recognition task determine what feature extraction and type of object location is useful. There is no point in propagating information from the feature extraction process which is not required for pattern classification. Likewise, enough information should be propagated that it can actually differentiate between the pattern classes required to be recognised.

In their paper 'Character Recognition - A Review' V.K.Govindan and A.P.Shivaprasad classify pattern recognition approaches into one of two categories[53]. These are the Statistical/Decision-theoretic, and the syntactic/linguistic/grammatical/structural approaches. This separation divides pattern recognition strategies into one of two philosophical dispositions.

The first is to regard the features as information about which initially we know nothing. Statistical analysis can then automatically weight those features that are found to be good at differentiating patterns. The input features are normally mapped onto a vector of fixed length n, the input pattern can be mapped into an n-dimesional pattern space, and a pattern class can be defined as an enclosure within that space[54].

The second is to assume that the structure of the feature data can be interpreted in some consistent way. This structure can be revealed by processing the interrelations between features[55]. Formal language-theoretic models can then be used to construct a grammar describing learned patterns and the syntax of input patterns can then be mapped onto pattern classes by syntax analysis[56].

The performance of the two dispositions depends on the task. If the topology of a

character always describes a pattern fully then the structural/syntactic approach is very good. In this case the statistical/decision theoretic approach performs less well as it does not 'see' the topology as a unique feature as it is only revealed when the structure or connection tree is parsed.

On the other hand if the image is prone to random discontinuities then a syntactic approach may be inappropriate as the discontinuities will completely transform the topology of the character. However, with this type of noise the statistical approach can often (depending on the feature mapping) locate the character as the 'best' or 'nearest' choice. The syntactic approach cannot locate the 'nearest' choice in a useful way as it is not operating in a uniform pattern space, e.g. the nearest topology if there is a discontinuity is likely to be the wrong character.

A problem with the Statistical approach which is brought out in the paper "Feature Identification for Hybrid Structural/Statistical Pattern Classification" by H.S.Baird[57], is that it is reliant on the input class-conditional distributions being uni-modal. That is, all patterns to be classified as the same must tend to a contiguous area in n-dimensional space. Thus, in converting between structural features (such as those described in the previous section) and a fixed length vector for statistical recognition care has to be taken that this mapping leads to a consistent interpretation, i.e. small changes in the input parameters must lead to small changes in the statistical classification vector.

To overcome this problem H.S.Baird proposes a hybrid structural/statistical approach[57]. In this approach the mapping of input features onto the statistical fixed length vector is by means of clustering of the input features. The input features are clustered during the learning process to form regions in the structural feature space. During the run-time recognition of patterns, a fixed length vector is formed by a bit being set for each cluster region that intersects an input feature. The resulting n-bit vector is used to classify the character using a Bayesian statistical classifier.

Optimally, and without any time or memory constraints, one would seek to use a combination of the two dispositions[56], as in the above example. The main drawback with such schemes can be their relatively slow execution speed; the above hybrid recognition produces a classification speed of only 6 characters/second[57].

2.5 Motivation of research

Interest in the low and intermediate levels of image processing was fuelled by the perception that this often constitutes the bandwidth bottleneck in visual recognition systems. Also, the throughput of higher levels of pattern classification and semantic processing can be increased to match increases in low/intermediate processing by the adoption of process parallelism on microprocessor pipelines or 'fine-grain' parallelism perhaps based on neural/connectionist nets.

The continuing increases in visual resolution and microprocessor speeds suggest a type of image processor for pattern recognition based on the close coupling of image parallel hardware and micro-processors. The image parallel hardware could perform the computationally intensive pixel processing tasks with the higher algorithmically complex levels of feature processing performed on microprocessors.

If the cost of the system was modest and the processing was general purpose then such an image processor could find application in a range of visual pattern recognition tasks. Within this context the high cost and restricted structures of programmable cellular or pipeline processors would be inappropriate. Therefore, dedicated pipeline processors based on binary neighbourhood operators were investigated.

The major obstacle to the implementation of the type of image processor envisaged was found to be the lack of suitable intermediate-level algorithms. Suitable algorithms for feature extraction and connectivity determination would allow for all pixel dependent processing tasks to be implemented in dedicated hardware. The difficulty in constructing such algorithms is that the intermediate-level algorithms must generate non-local information about the low-level pixel data, though only image-local processing is performed easily on image parallel hardware.

Chapter 3.

Initial Experiments, PiNOLa specification and PiNOSim implementation

This chapter describes the early stages of the research and the simulation language which evolved from this work.

3.1 Initial work

Before starting the research presented here a very simple 3x3 binary neighbourhood operator had been implemented as a TAHC06 gate array. One of the initial areas of exploration was how to integrate this gate array with a line store and scanner to produce a binary low-pass spatial filter. The resulting circuit shown in Fig 3.1 is

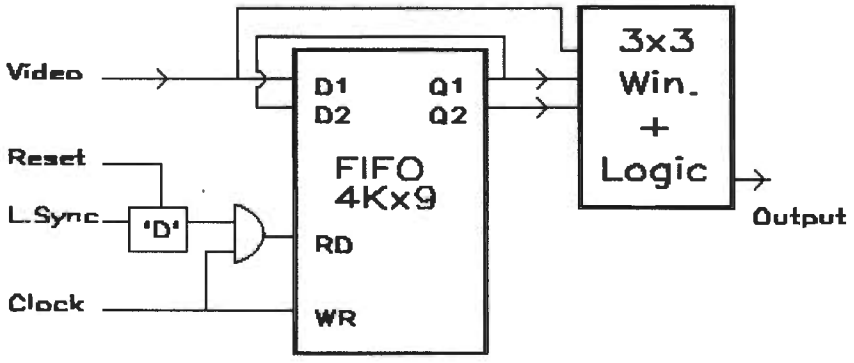


Fig 3.1: 3x3 neighbourhood operator and line store circuit

little more than the gate array and the FIFO component used to implement a variable length line store. After receiving a reset signal, the circuit in Fig.3.1 writes binary pixels into the FIFO until the first line-sync. The circuit then reads and writes data synchronously so maintaining the length of the line store. Two lines of the image are stored so that a 3x3 window can be constructed from three rows of three binary latches in the gate array, as shown previously in Fig.2.3

This work was presented in a paper, 'Pipelined Logical Convolvers for Binary Picture Processing', published in Electronic Letters[58]. The paper demonstrates the simplicity, speed and low cost of dedicated neighbourhood operator pipelines based on modern components. (Fast, single chip 4K deep FIFOs as required for variable width, 300 d.p.i. line stores have only been easily available since 1990). However, as was mentioned in the paper, to realise the potential of 3x3 binary operators requires a powerful and interactive simulation language. The justification for this conclusion is two fold:

- Even a single 3x3 binary operator can be extremely complex. Each instance of the 3x3 window can be given a binary output, independently of the output given to any other instance of the 3x3 window. As there are 2^9 different possible 3x3 windows, 512 bits are required to describe a 3x3 operator uniquely. It follows that there are 2^{512} distinct 3x3 binary operator functions. If the four post processed neighbours from a sequential 3x3 operator are also used to determine the output then there are approximately 2^{8192} distinct functions. To select one function from 2^{8192} requires the ability to split the specification of the operators into separate sub-functions. For instance we might wish to specify the effect of an operator on the connectivity of the image separately from its effect on the edges of objects in the image.

- Dedicated pipelined neighbourhood operators are easy to interconnect, very complex pipelines can thus be constructed. The simulator should therefore be able to describe any pipeline, parallel pipeline or tree structure.

Existing languages for describing neighbourhood operations tend to be aimed at specific architectures, often including powerful extensions only available on particular hardware. For example, the language which programs the PICAP machine uses notation which specifies the iterative behaviour of functions[20], so that a function may be repeated an indeterminate number of times dependent on the input image. This function is only meaningful if the underlying architecture, i.e. a frame store, is available.

The evolution of a new language followed from looking at how pipelined neighbourhood operators could be implemented efficiently in custom silicon. This research stemmed form a perception that a useful dedicated device would have to consist of a large number of discrete stages, say 30 or 40. If these stages could be implemented efficiently in custom VLSI then all the stages could be placed on one chip with a single wide line-store bus.

The architecture devised to implement neighbourhood operators in VLSI later developed into a language for describing such operators. The language still has the potentially useful property of being able to generate an efficient VLSI design from the specification of an operator in the language.

Each line in the AND plane in Fig.3.2 specifies a 3x3 template which generates either a 'high exception' or 'low exception' in the OR plane. These 'exception' signals over-ride the default value of the operator which is the value of the centre pixel of the 3x3 window. This method has the merit of allowing any possible function to be constructed whilst only requiring the specification of those templates which are to effect the output relative to the input.

The decomposition of the neighbourhood operator definition into an AND OR plane structure allows the operator to be specified in custom silicon without resorting to a 512 bit look-up-table which would take up much greater silicon area.

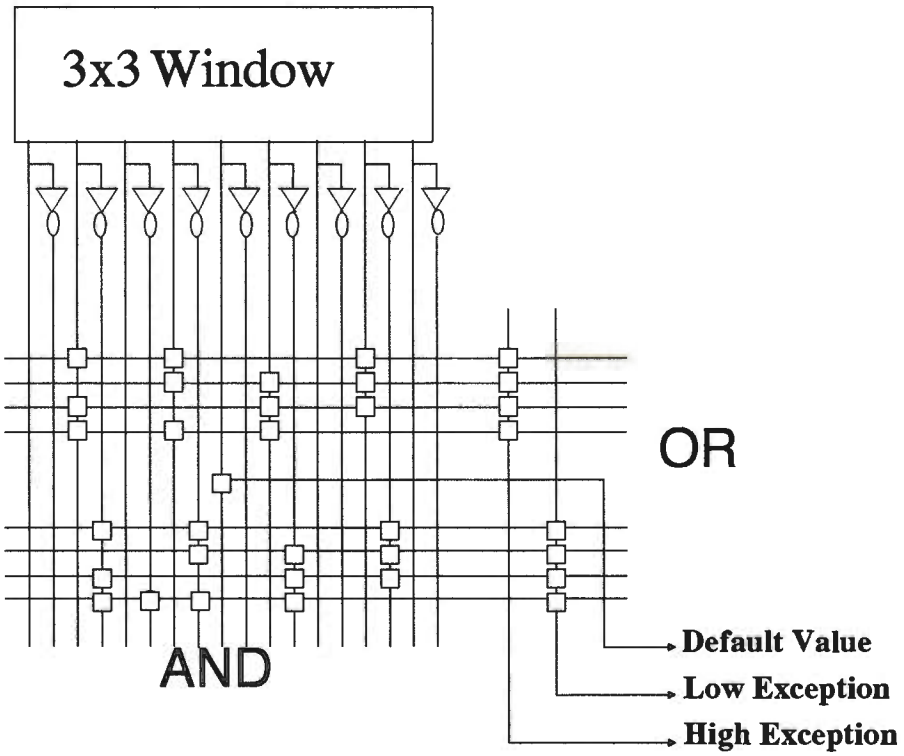


Fig 3.2: 3x3 operator implemented as an AND OR plane

3.2 PiNOLa specification

The 3x3 templates implicit in each line in the AND plane in Fig 3.2 can be represented as a list of 3x3 symbols and an 'exception' value. The exception value generated by a template is either 1 or 0. A '0' in the template definition represents a low value at that location in the 3x3 window, a '1' represents a high value and a '-' is don't care.

A neighbourhood operator requiring three lines in the AND plane could be specified in the following way:

```
1       {high exception}
0 1 0
- - -
- 1 -

1       {high exception}
1 - 1
0 - 0
1 - 1

0       {low exception}
1 1 1
0 - 0
1 1 1
```

This is the starting point for the development of PiNOLa (Pipelined Neighbourhood Operator Language). Each 3x3 template represents an entry in the AND plane generating either a 1 or a 0 exception to the pre-processed default value, i.e. if the 3x3 window is given the following labels;

```
a b c
d e f
g h i
```

The boolean equation for the function specified by the above templates is;

```
e' := e*/[Any low exception set] + [Any high exception set]

e' := e*/[ a*b*c*/d*/f*g*h*i ] + [(/a*b*/c*/g*h*/i) +
(a*c*/d*/f*g*h*i)]
```

Where e' is the value of the pixel corresponding to e in the post-processed image. If a low exception and a high exception are generated for the same 3x3 window the output becomes ambiguous. (Though the above boolean equation makes an explicit assumption about the precedence of high exceptions). To resolve this PiNOLa uses the concept of a template hierarchy. Each template can be given a hierarchy value 'h' whose default is 0. Whenever a potential ambiguity arises this is resolved by accepting the output value with the highest hierarchy. If the programmer specifies an operator which contains an ambiguous output, i.e. differing outputs from templates with the same hierarchy, this is trapped as an erroneous specification. Many levels of hierarchy can be specified as in the following function which generates an edge without corners foreground components of the image.

```
0,h1 {make all foreground background, level 1}
- - -
- 1 -
- - -

1,h2,s {retain all edges, level 2}
- 0 -
- 1 -
- - -

0,h3,s {delete all corners, level 3}
- 0 -
0 1 -
- - -
```

The 's' parameter in the above definition simply generates all reflective and rotational symmetric templates, which makes the definitions more succinct. The effect of the above templates is therefore to: invert all foreground pixels, unless that pixel is at a 4-connected edge, unless that pixel is a 4-connected corner pixel in which case it is inverted.

A template may include an 'i' parameter which generates the inverse function, e.g.

```
1,i
0 0 0
0 1 0
0 0 0
```

The above template changes the value of pixels surrounded by the opposite value. The 'i' (invert) parameter applies a function specified in terms of the foreground to the background as well, or visa versa.

To extend the power of the template descriptors further, the concept of an OR AND OR plane structure was used, which could produce more compact silicon implementations for some functions. Upper case letters in the template are used to specify a high OR group. Lower case letter specify a low OR group. e.g.

```
1
A A A
a - a
B B B
```

specifies a function which outputs a high exception whenever there is a high value in the top row, a high value in the bottom row and at least one low value in the middle left and right edges.

i.e.

```
e' := e + [ (a+b+c) * (g+h+i) * (/d+/f) ]
```

As well as the standard 3x3 neighbourhood functions there is another class of 'feedback' or 'sequential' function. These functions look at the four post processed neighbours available in a pipelined 3x3 neighbourhood operation.

A 'feedback' function is defined as;

```
e' := f(a',b',c',e,f,g,h,i)
```

Using the pre and post processed values generates what is referred to here as a 'hybrid' function, i.e.:

```
e' := f(a,b,c,d,e,f,g,h,i,a',b',c',d')
```

A feedback function is implemented in PiNOLa simply by adding an 'f' parameter to the template definition, e.g. the following 3x3 template produces a feedback blocking function. The effect of this function on an image is shown in Fig. 3.3.

```
1,f
- 1 -
1 0 -
- - -
```

Hybrid functions can be specified in two ways. Firstly, simply by mixing feedback with standard templates as in the following:

```
0,f,h1  {remove from left, feedback}
- - -
0 1 -
- - -

1,h2 {inhibit removal of lines if the left edge has this profile}
0 0 -
0 1 1
0 0 -
```

Hybrid functions can also be specified directly as one template. This is useful if the interpretation of the post-processed neighbourhood is to be dependent on the full 3x3 pre-processed neighbourhood.

```
1,f
0 1 0  1 0 0
0 0 0  0
- 1 -
```

Feedback and hybrid functions have the important property that they can generate effects which are propagated in the direction of the scan. Within the context of the type of image processor which we are trying to evolve, feedback functions may be able to generate non-local information from an image. This is because information

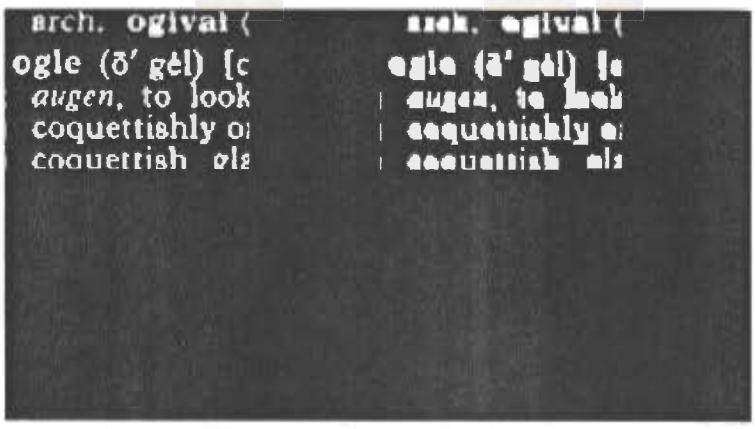


Fig.3.3: effect of 'blocker' feedback function

is propagated through the image by such functions.

The effect of the non-local propagation property can be seen in the blocker function where in one pass the foreground is propagated in the direction of the scan beyond the 3x3 neighbourhood.

Using hierarchy it is also possible to split the definition of feedback functions between a connectivity relationship e.g. 'Maintain the connectivity of the image', and an effect e.g. 'erode the foreground components of the image'.

The following three templates specify a function of this type.

```
1,f,h2,s,i {maintain 4 connect relationship, level2}
a 1 b
a 1 b
a 1 b

1,f,h2,s,i {maintain 4 connect relationship, level2}
a a a
a 1 1
a 1 0

0,s,h1     {delete edges function, level1}
- 0 -
- 1 -
- - -
```

The above templates specify one pass in a connectivity-maintaining edge erosion function. The connectivity relationship templates are given a higher priority than the edge erosion template. Thus the function deletes pixels from all edges where deleting a pixel does not alter the 4-connectedness of the image.

A PiNOLa program can include any number of neighbourhood operator descriptions, specified in the following format:

```
BPP name 1
list of templates
END
  ..
  ..
BPP name n
list of templates
END
```

The names given to BPPs (Binary Picture Processors) can then be used to describe a pipeline using the following FLOW statement:

```
FLOW
  BPPname - BPPname <
                    > BPPname
                    > BPPname <
                    |         > BPPname
                    |         > BPPname - BPPname
                    > BPPname
END
```

To produce a compact and visually intelligible interface the FLOW description has been given a 2 Dimensional structure. The pipeline symbols have the following meanings.

- '-' pipeline continuation from left to right.
- '<' pipeline continuation from left to down.
- '>' pipeline divergence from above to down and right.

some parameters can also be applied to instances of operators called in the pipeline FLOW descriptor.

- 'd' display output of operator
- 'mN' make a pipeline of N consecutive operators

3.3 PiNOSim implementation

Having found a need for a pipelined neighbourhood operator simulator and evolved a language to describe the simulation, an implementation 'PiNOSim' was produced. PiNOSim was implemented in Modula-2. The program is divided into three modules, Parser, Compiler and Run-time system.

Parser Module

The parser uses recursive descent procedures to construct a list of operators and a pipeline tree of operator instances. Syntax errors are reported to the user.

Compiler Module

The compiler takes the operator list and generates a list of 3x3 sub-templates for each template in the definition. These sub-templates correspond to the original AND OR format derived earlier. The sub-templates are then purged to remove duplicate sub-templates, and sorted into hierarchical order. Each possible value of the 3x3 window is then compared with the sub-template list to generate a 512 bit look up table (8192 bits for Hybrid functions). Instances of these look up tables are then inserted into the pipeline tree which is then passed to the run-time module.

Run-Time Module

The run-time module takes the pipeline tree and sets up a concurrent process for each node in the tree. Then chunks (1024 bits) of the input image are passed to the top node which then passes processed chunks to its child processes. Those operator instances that are to display to the screen are allocated a display window.

All the processes in the run-time module were designed to mimic the operation of hardware, e.g. the line store was implemented as a variable length FIFO, the 3x3 window as an address to a 512 bit look up table and the image data as streams of boolean values. Optimisations such as run-length coding of image streams or hierarchical inspection of the 3x3 neighbourhoods were avoided to maintain the correspondence between software and hardware. This direct correspondence enables the simulation of the underlying hardware as well as the higher level PiNOLa programs. One consequence of this approach is that it allows the extension of the hardware

simulation beyond the PiNOLa language, something which was vital for the investigation of hardware connectivity processing described in Chapter 5

Chapter 4

Experiments with PiNOSim and Feature Extractor Design

PiNOSim was used to investigate a number of image transformations some of which are included here. This investigation led to the development of a new feature extraction algorithm.

4.1 Edge pixels

One of the easiest features which can be generated from an image are the edge pixels between background and foreground components of an image. The edge pixels have the useful property of retaining a great deal of the original information in the image; this is the basis of run-length encoding etc. In terms of pattern recognition, edge pixels allow the area of the input object to be determined accurately. The area of an object is a very useful statistic for some pattern recognition applications as it is highly immune to variance caused by the rotation of the object. This method of recognition is used in the automatic identification of shoe segments[59]. The parameters of this particular recognition task are that nothing is known about the orientation of the shoe segments and the shoe segments vary greatly in size, making area classification a very useful initial step in the recognition process.

A 4-connected edge can be generated by the following operator:

```
BPP gen4con {inverts all foreground non-edge pixels}
  0
  1 1 1
  1 1 1
  1 1 1
END
```

The drawback of using edge pixels as features to be propagated from the image parallel hardware is that there may be a lot of edge pixels and each individual pixel tells us very little about the object. The implication of this is that a lot of processing

has to take place at a higher, perhaps non-parallel, level to extract super-features from the boundary data. This is the case for the shoe segment identification machine mentioned above, here the boundary data is transformed into a set of 18 equally spaced radii from a calculated centroid position. The lengths of the radii are then used to classify the object. The overall throughput of the system is 2 seconds per shoe segment using a three processor pipeline and a hardware edge extraction operator. The extremely difficult recognition problem in this application requires such a high-level approach, but this would not be feasible if the application required a much faster throughput. If we wish to recognise objects at a faster rate then we can move the computational effort down from the software process towards the image parallel hardware. This can be achieved by generating features that are better discriminators in the hardware neighbourhood operator pipeline.

4.2 Erosion and dilation

As well as generating features, certain applications may require various pre-processing neighbourhood operations to perform such tasks as the removal of small objects, spatial low-pass filtering and object separation. These effects can be achieved by performing erosion and dilation. An erosion operator removes the outer layer of pixels from the edge of foreground objects, a dilation operator adds a layer of pixels to the edge of foreground objects. Performing a series of erosion stages followed by a series of dilation stages has the effect of removing small 'salt and pepper' components of the image and of smoothing jagged edges of the image. By introducing the concept of maintaining connectivity during the dilation stages, erosion and dilation can also be used effectively to separate loosely connected components of an image, something which is used in Biomedical imaging[10]. This type of processing is demonstrated by the following PiNOLa program the output of which is shown in Fig.4.1. The picture is first eroded for five stages and then thickened for the same number of stages whilst maintaining the 4-connectivity of the image. The input picture is shown top left and the process progresses left/right top/down. The final picture shows the segmented image restored to its original size.

```
BPP Erode
0,s
- 0 -
- 1 -
- - -
END

BPP ThickMod
{maintain 4 connect}
  1,f,h7,s,i
  a 1 b
  a 1 b
  a 1 b

{maintain 4 connect}
  1,f,h7,s,i
  a a a
  a 1 1
  a 1 0
```

```
  1,s      {thicken}
  - 1 -
  - 0 -
  - - -
END

BPP Input
  1
  1 1 1
  1 1 1
  1 1 1
END

FLOW 1
  Input /d - Erode /d,m5 - ThickMod /d,m5
END
```

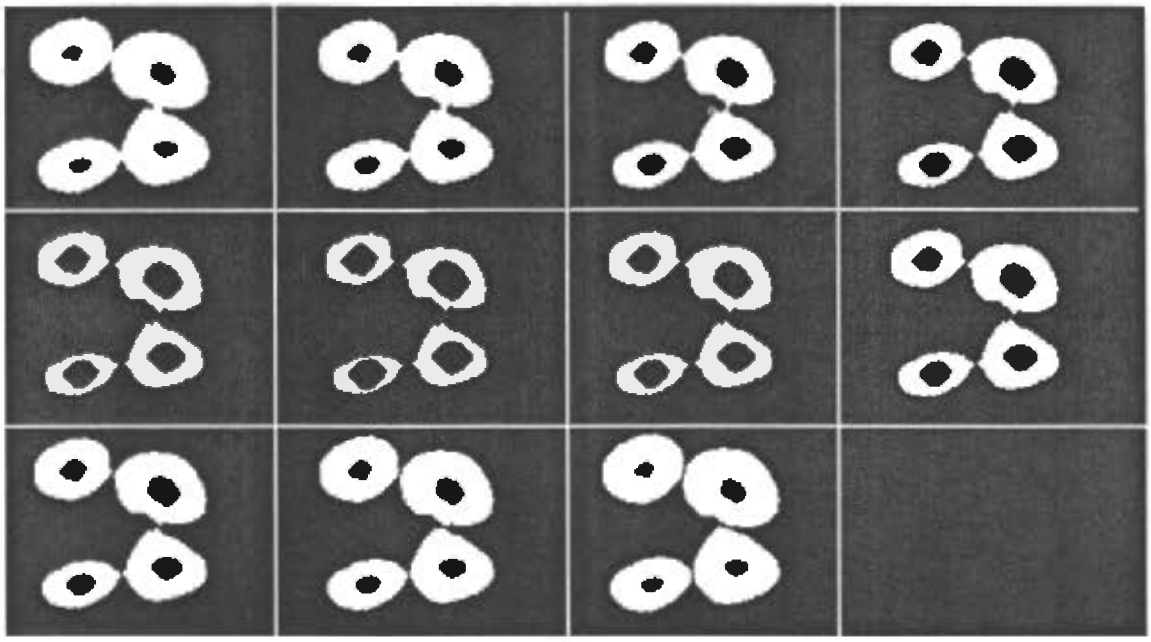


Fig.4.1: Connected object separation by erosion/dilation

The main drawback of erosion and dilation for image smoothing is that it affects the connectivity of the image. In the context of a particular application this effect may be tolerable or even desirable as in the above example. However, the particular effect desired will vary with the particular application. In terms of OCR processing we might often wish to reconnect objects in an image prone to discontinuities or disconnect characters that are joined together as ligatures. The problem is that *a priori* we do not know which effect would be more useful and we might wish to apply different transformations to different parts of the same image. Therefore erosion and dilation is most useful in those applications where quality of the images is very consistent as in circuit inspection tasks.

4.3 Skeletonization

Skeletonization is the reduction of all the foreground components in an image to lines of one bit thickness by a process of connectivity preserving erosion or thinning[50]. A further distinction is sometimes made between connectivity preserving shrinking (CPS)[60] and skeletonization. The difference between CPS and skeletonization is that during the erosion process skeletonization not only preserves the connectivity of the image but also the ends of thin lines. Thus, if we apply CPS to a simply connected object, after a number of stages the object is represented as a point, whereas the skeletonization process will generate a skeleton made up of thin lines representing the limbs of the original connected component. From the point of view of feature extraction therefore skeletonization is more interesting as the end nodes of the skeleton represent topological features in the image.

One of the problems with skeletonization is that it can be quite a slow process. If feedback functions are not used then a 3x3 operator can only peel off pixels from one side of the foreground components at a time and still maintain the connectivity of the image. Thus skeletonization has to proceed by alternately removing pixels from north, south, east, west, this process therefore requires four stages per layer of pixels to be removed. Pixels can be removed from all four sides at each stage by adopting a hybrid function which makes use of the post-processed neighbourhood pixels. However, care has to be taken to minimise skew due to the direction of scan in the image as feedback functions are not symmetrical in their effect. The following PiNOLa statements describe a hybrid operator 'ThinFour' which removes pixels from all four sides of the components of the image whilst attempting to minimise skew. The result of three iterations of the operator is shown in Fig.4.2.

```
BPP ThinFour
{maintain 4 connect}
  1,f,h7,s
  a 1 b
  a 1 b
  a 1 b

{maintain 4 connect}
  1,f,h7,s
  a a a
  a 1 1
  a 1 0

{maintain formed edges}
  1,h7,s
  0 0 0
  1 1 0
  0 0 0

  1,h7  {skew templates}
  0 0 0
  1 1 0
  1 1 0

  1,h7
  - 1 0
  1 1 0
  0 0 0
```

```
  1,h7
  0 1 1
  0 1 1
  0 0 0

  1,h7  {skew to avoid pixel stranding}
  1 1 1
  1 1 1
  1 1 0

  0,s   {delete edges}
  - 1 -
  - 1 -
  - 0 -

  0,s    {delete corners}
  - - -
  - 1 1
  - 1 0
END

BPP Input
1
- - -
- 1 -
- - -
END
FLOW
  input /d - ThinFour /d,m3 {display the input then 3 iterations of
FastSkel}
END
```

The above algorithm is similar to the 3x3 sequential thinning algorithm described by Hilditch[61]. If a larger neighbourhood can be contemplated then skeletonization by removal of pixels from all four sides can be performed using a 5x5 window[39].

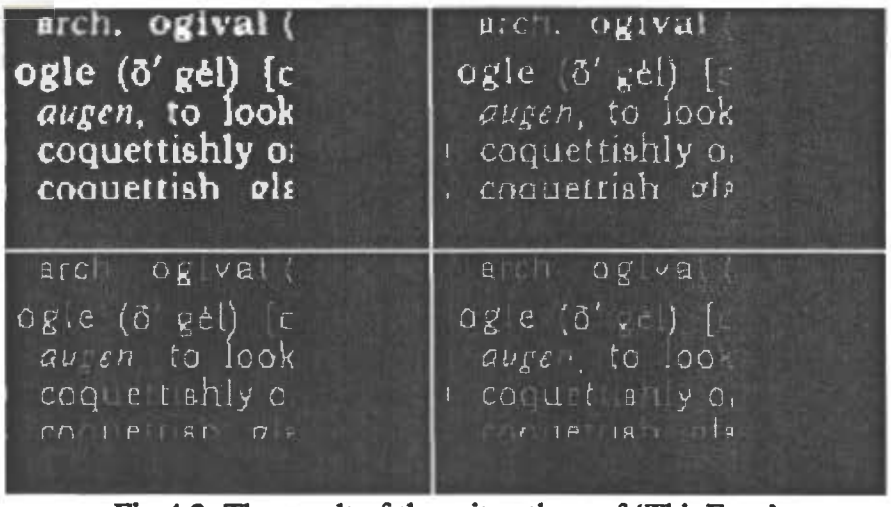


Fig.4.2: The result of three iterations of 'ThinFour'

4.4 FSkel, RSkel and BSkel

Skeletonization provides the basis for topological feature extraction not based on prior image segmentation, but it is not deterministic. An indeterminate number of stages are required to skeletonize an image containing components that are of

arbitrary thickness.

Following on from the initial experiments with skeletonization, a new type of pseudo skeletonization function called 'FSkel' was developed. The interactive ability of PiNOSim to allow the specification of functions with different hierarchies allowed experiments with different types of connectivity and edge erosion to be performed easily. Instead of attempting to minimise skew in the skeletonization process, it was realised that by ignoring skew it is possible to generate a pseudo skeleton in only one pass of the image.

This is achieved simply by removing all pixels not required to maintain connectivity in a feedback operator. The resulting 'FSkel' operator is given as;

```
BPP FSkel
 {maintain 4 connect}
   1,f,h7,s,i
   a 1 b
   a 1 b
   a 1 b

 {maintain 4 connect}
   1,f,h7,s,i
   a a a
   a 1 1
   a 1 0

   0,f   {remove all pixels}  -- this operator removes all pixels
   - - -                      -- not required to maintain
   - - -                      -- connectivity
   - - -
END
```

The effect of the FSkel operator on the foreground and background of an image is shown in the fourth column of Fig.4.3. This gives a somewhat peculiar result, but clearly there is a one bit thickness output image with the same connectivity as the original image, a skeleton of sorts.

By experimenting with the preserving of other local properties not just connectedness, a particularly interesting operator was discovered, RSkel. This operator preserves the right hand edge of the image as well as its connectedness.

```
BPP RSkel
 {maintain 4 connect}
   1,f,h7,s,i
   a 1 b
   a 1 b
   a 1 b

 {maintain 4 connect}
   1,f,h7,s,i
   a a a
   a 1 1
   a 1 0
```

```
0,f    {remove from left}   -- this operator removes all pixels
- - -                        -- not required to maintain connectivity
0 0 1                      -- or at the right edge
- - -
END
```

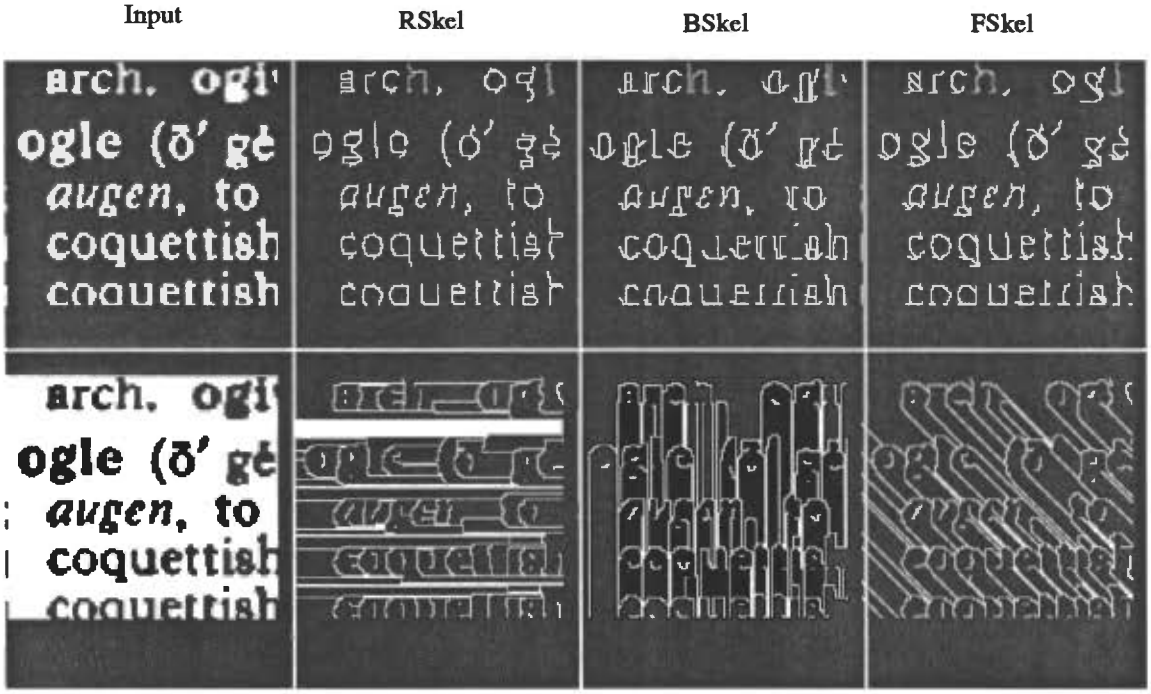


Fig 4.3: RSkel,BSkel and FSkel foreground and background

The output from the RSkel operator is shown in the second column of Fig.4.3. On analysing the relationship between nodes in the post RSkel image with the input image, a very interesting property was observed. The end nodes of the RSkel output all specify a point on the maximum extent of a foreground vertical protrusion into the background. One could propagate the location and direction of these end nodes as features which describe an important facet of the topology of the components in the input image.

There is a corresponding operator BSkel which preserves the bottom edge of the input image as well as its connectedness. The end nodes of the BSkel output describe the horizontal protrusion of the foreground into the background. The effect of this operator is shown in column three of Fig.4.3.

By performing the R/BSkel operations in parallel, on separate streams of the same image, we can generate different 'views' of the original image. Features from these different views could then be associated with the components of the original image in order to derive feature data describing the image.

The end nodes of RSkel and BSkel describe the limb topology of the image. Taking the junction nodes would have been sufficient to describe the enclosure topology as well. However, if instead we perform RSkel and BSkel on the background and take

the end nodes of this transformation, the enclosure topology information is provided as well as the extent of all background protrusions into the foreground. The effect of these transformations on the background is shown in the bottom row of Fig 4.3.

4.5 Feature extractor

The RSkel and BSkel operators can be used as the basis of a topological feature extractor. Each function generates two types of end nodes which may easily be identified in another stage, e.g the BSkel foreground operator generates left and right handed end nodes corresponding to left and right limb protrusions of foreground objects into the background. There are therefore eight different end node types, two each for the RSkel and BSkel foreground and background functions. This is best shown by looking in detail at the effect of the operators on a simple image.

```
  input          RSkel          BSkel        back. RSkel     back. BSkel
............    ............    ............    ............    ............
..++++++++..    .........T..    ............    .....+++++++    ....++++....
..++++++++..    ....++++++..    ....++++....    ....+.......    ...+....+...
..++....++..    ...+.....+..    ...+....+...    ...+...t....    ..+......+..
..++....++..    ...+.....+..    ...+....+...    ...+...+....    ..+......+..
..++....++..    ...+.....+..    ...+....+...    ...+...+....    ..+......+..
..++....++..    ...+.....+..    ...+....+...    ...+...b....    ..+.l++r.+..
..++++++++..    ....++++++..    ...+....+...    ...++.......    ..+......+..
..++++++++..    .........B..    ..L++++++R..    ....++++++++    ..+......+..
............    ............    ............    ............    ..+......+..
```

```
'.' background pixel
'+' foreground pixel
'T' foreground RSkel top end node
'B' foreground RSkel bottom end node
'L' foreground BSkel left end node
'R' foreground BSkel right end node
't' background RSkel top end node
'b' background RSkel bottom end node
'l' background BSkel left end node
'r' background BSkel right end node
```

Thus the effect of the feature extraction is to transform the image into the following feature map:

```
  input          feature map
............    ............
..++++++++..    .........T..
..++++++++..    ............
..++....++..    .......t....
..++....++..    ............
..++....++..    ............
..++....++..    ....l.rb....   (r,b overlap)
..++++++++..    ............
..++++++++..    ..L.....RB..   (R,B  overlap)
............    ............
```

This transformation produces a very compact description of the image which is not dependent on the pixel resolution only the topological complexity of the components in the image. Naturally this process is not loss-less, because the feature map contains no information about non-enclosing curvature of the components of the image. Thus

this transformation cannot distinguish as separate classes patterns which are only differentiated by this type of curve. For instance, the images formed by the letters 'U' and 'V' cannot be differentiated as separate classes on the basis of the relative curvature of their limbs. More features describing different facets of the image could be generated to give curvature information. However, there is a trade-off between generating more features and the time taken to process these features. The objective of this research was to demonstrate real-time continuous recognition, therefore in the context of the system which is described in the following chapters, it was decided not to extend the feature base.

Generally the R/BSkel features have a very good pattern differentiation to data ratio, i.e. each feature says a lot about the object and is simply defined by a three bit feature type and its X and Y co-ordinate. In the context of OCR a typical character will be defined by about ten R/BSkel features. Another useful property of the features is that they give the exact extent of the foreground/background protrusions, this geometric information turns out to be a highly invariant parameter for pattern recognition and also gives the exact bounding box of the character, something which is useful for higher-level OCR processing.

The characteristics of the R/BSkel features determines the style of recognition which is evolved in subsequent chapters. Each feature describes a non-image-local structural/topological facet of the image, i.e., 'here is a limb or enclosure pointing in this direction'. Each feature also contains geometric information, i.e., 'here is a point on the maximum extent of this limb or enclosure'. All the features of a connected component taken together uniquely identify the topology of that component. The reconstruction of this topology without additional connectivity information would be a difficult computational task. However, reconstruction is not a necessary basis for recognition.

Chapter 5

Component labelling and GRIPPR design

5.1 Component labelling algorithm

The following quote is from the book 'Connected Components in Binary Images: the Detection Problem', by Ronse and Devijver 1984:

"Our work is based on the assumption that the picture is scanned in raster mode and only a small part of it is accessible simultaneously. This requires that the detection of connected components be achieved in real-time and in a sequential fashion.

Within this framework, two possible approaches can be advocated. One is based on border finding techniques, ... the second ... is based on tracking runs of the figure..."[48].

I would like to suggest one more approach which can be adopted, given the same assumptions, this is described below.

Motivation

The original motivation for the connectivity research was to segment the transformed image generated by the feature extractor described in the previous chapter. The idea was that a component label assigned in hardware could be used to map features onto a connected object in software.

Thus the division of the connectivity process would be between pixel dependent processing in hardware and component label processing in software. This arrangement is shown in Fig.5.1; here the serial scanned image is passed to feature extraction and component labelling hardware. The output of the hardware process is a stream of component labelled features and connectivity information. These features can then be grouped in a software process to generate a stream of connected object feature lists.

The requirement was also to maintain a data structure that could reveal the connectivity of the image in real-time, i.e. an object is known to be complete when the last line containing that object has been processed. This is so that pattern recognition may proceed in parallel with the image processing. Otherwise the advantage of having a continuous topological feature extractor would have been lost.

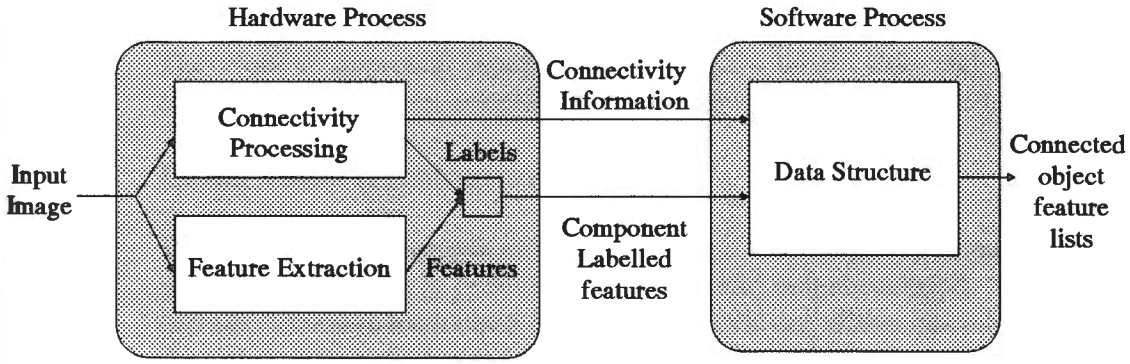


Fig.5.1: connectivity hardware and software processes

5.1.1 Hardware connectivity process

The minimum connectivity information which needs to be propagated by the hardware to perform the above task is to signal when two provisional label values are connected and to signal when a provisional label value has terminated.

Thus the hardware must propagate two data types to the software, JoinVal and EndVal:

- A JoinVal type is propagated when two edge pixels are found to have different label values. The parameters are the values of the labels.
- An EndVal type is propagated when a label value is no longer current. The parameter passed is the value of the label which is no longer current.

Thus our image from the background chapter generates the following connectivity information:

```
.................     connectivity info  (J=JoinVal E=EndVal)
..11..22...33.....
..11..22...33.....
..11..22...33.....       J(1,2)  E(2)
..111111....33....       E(1)    E(3)
.................
```

However, as already stated in the background section, there is no way of computing when a provisional label value is no longer current by the application of neighbourhood logic. This can been seen by looking at the provisional label value assignment in the following image.

```
..................
..1111............
..1..111111.......
..1...1...1.......
..1...............
..................
```

Without propagating non-local information there is no way of knowing at the time when an end-node is discovered whether its label value has terminated or whether there are other components elsewhere in the image assigned the same label value.

Therefore an approach based on processing the label values rather than components of the image has been adopted to determine when a label value has terminated.

Label value termination

To propagate the component labels through the image we already need to store a whole line of labels in order to access the local neighbourhood. This means that we can also access in each clock cycle two label values, where one label is directly above the other. Simply by using these values to index two boolean arrays, we can determine at the end of each line which values were in the last line but not in the current line.

The algorithm works by initialising the boolean arrays at the start of each line of image to be blank. Then, as a line of image is processed, a bit is set corresponding to the top label value in one array and the bottom label value in the other array, for each pixel clock. When a line of image is complete, the array indexed by the top label value contains a bit set for each label value present in the previous line. The other array, indexed by the bottom label value, contains a bit set for each label value in the current line. Thus, by cycling through the arrays at the end of each line, it is possible to determine which label values were in the previous line and are not in the current line, and signal these values as terminated.

The following example demonstrates how this algorithm works.

```
                  prev vals  cur vals  EndVal info.
..................     -         -
..11..22...33.....     -       1,2,3
..11..22...33.....   1,2,3     1,2,3
..11..22...33.....   1,2,3     1,2,3
..111111....33....   1,2,3     1,3       E(2)
..................   1,3         -       E(1) E(3)
```

An EndVal is generated whenever a label value was found to be in the previous line but not in the current line.

The apparent drawback is the time it takes at the end of each line to cycle through the boolean arrays which must contain one entry for each possible label value. For very large complex images, such as pages of text, this might be over 10,000 label values making the algorithm totally impractical.

The solution to this problem is to recycle the provisional labels in hardware as soon

as an EndVal is generated. Now only enough label values to describe one line of the image are required; as label values are either propagated from one line to the next through the components of the image or they are recycled into a label value store. The largest number of distinct connected components in one line of a binary image is linewidth/2, i.e. one distinct connected component every other pixel. (If multilevel images were to be processed in a similar fashion then the maximum number of connected components on a line is still bounded to one per pixel.)

Recycling label values in this way also produces a number of other useful properties:

- Only a small number of bits are required to describe the labels, therefore processing of labels becomes much cheaper and simpler. Also serial transmission of labels becomes faster and the software data structure requires less memory.

- Labels are continually recycled, therefore processing can proceed continuously on images of infinite length. This is particularly important in many industrial inspection applications, where stopping the scan may be difficult.

- The method is guaranteed never to run out of label values irrespective of the complexity of the input image.

Also, as the number of discrete component label values is less than the number of pixels in a line it is possible to perform the algorithm without any pause between lines. This can be achieved by having two sets of Boolean arrays for the termination logic. The function of these arrays can then be time-multiplexed so that when one set of arrays are being set by the top/bottom label values, the other set is cycled through to determine which label values have terminated. The function assigned to each set of arrays then alternates for each line of image input.

5.1.2 Software connectivity process

At the software end, the simplest data structure to support the algorithm is a linked list of label value vectors pointing to a list of features, as in Fig.5.2. In hardware, features are assigned the values corresponding to their position in the image. When a feature arrives it is appended to the feature list pointed to by its label value.

A JoinVal type causes the two label value vector lists that contain JoinVal label values to be joined together. Care has to be taken that the two label values are not already in the same list and if this is the case no action should be taken.

An EndVal type causes the feature list pointed to by the label value to be copied to another member of the label value's vector list. If the vector list is empty then all the values that have ever pointed to that connected object are complete and the feature list is transmitted to the recognition stage.

The following table illustrates the operation of the algorithm. The symbols 'A' to 'E' represent different features that are generated from the image and component labelled. The connectivity and feature information is generated by the hardware process. The software data structure consists of vector lists associating component labels; symbols '1' to '3', with features.

```
..................
..1111............
..1..111111.......
..1...1...1.......
..1...............
..................
```

Without propagating non-local information there is no way of knowing at the time when an end-node is discovered whether its label value has terminated or whether there are other components elsewhere in the image assigned the same label value.

Therefore an approach based on processing the label values rather than components of the image has been adopted to determine when a label value has terminated.

Label value termination

To propagate the component labels through the image we already need to store a whole line of labels in order to access the local neighbourhood. This means that we can also access in each clock cycle two label values, where one label is directly above the other. Simply by using these values to index two boolean arrays, we can determine at the end of each line which values were in the last line but not in the current line.

The algorithm works by initialising the boolean arrays at the start of each line of image to be blank. Then, as a line of image is processed, a bit is set corresponding to the top label value in one array and the bottom label value in the other array, for each pixel clock. When a line of image is complete, the array indexed by the top label value contains a bit set for each label value present in the previous line. The other array, indexed by the bottom label value, contains a bit set for each label value in the current line. Thus, by cycling through the arrays at the end of each line, it is possible to determine which label values were in the previous line and are not in the current line, and signal these values as terminated.

The following example demonstrates how this algorithm works.

```
                   prev vals   cur vals   EndVal info.
..................     -           -
..11..22...33.....     -         1,2,3
..11..22...33.....   1,2,3       1,2,3
..11..22...33.....   1,2,3       1,2,3
..111111....33....   1,2,3       1,3        E(2)
..................   1,3           -        E(1) E(3)
```

An EndVal is generated whenever a label value was found to be in the previous line but not in the current line.

The apparent drawback is the time it takes at the end of each line to cycle through the boolean arrays which must contain one entry for each possible label value. For very large complex images, such as pages of text, this might be over 10,000 label values making the algorithm totally impractical.

The solution to this problem is to recycle the provisional labels in hardware as soon

as an EndVal is generated. Now only enough label values to describe one line of the image are required; as label values are either propagated from one line to the next through the components of the image or they are recycled into a label value store. The largest number of distinct connected components in one line of a binary image is linewidth/2, i.e. one distinct connected component every other pixel. (If multilevel images were to be processed in a similar fashion then the maximum number of connected components on a line is still bounded to one per pixel.)

Recycling label values in this way also produces a number of other useful properties:

- Only a small number of bits are required to describe the labels, therefore processing of labels becomes much cheaper and simpler. Also serial transmission of labels becomes faster and the software data structure requires less memory.

- Labels are continually recycled, therefore processing can proceed continuously on images of infinite length. This is particularly important in many industrial inspection applications, where stopping the scan may be difficult.

- The method is guaranteed never to run out of label values irrespective of the complexity of the input image.

Also, as the number of discrete component label values is less than the number of pixels in a line it is possible to perform the algorithm without any pause between lines. This can be achieved by having two sets of Boolean arrays for the termination logic. The function of these arrays can then be time-multiplexed so that when one set of arrays are being set by the top/bottom label values, the other set is cycled through to determine which label values have terminated. The function assigned to each set of arrays then alternates for each line of image input.

5.1.2 Software connectivity process

At the software end, the simplest data structure to support the algorithm is a linked list of label value vectors pointing to a list of features, as in Fig.5.2. In hardware, features are assigned the values corresponding to their position in the image. When a feature arrives it is appended to the feature list pointed to by its label value.

A JoinVal type causes the two label value vector lists that contain JoinVal label values to be joined together. Care has to be taken that the two label values are not already in the same list and if this is the case no action should be taken.

An EndVal type causes the feature list pointed to by the label value to be copied to another member of the label value's vector list. If the vector list is empty then all the values that have ever pointed to that connected object are complete and the feature list is transmitted to the recognition stage.

The following table illustrates the operation of the algorithm. The symbols 'A' to 'E' represent different features that are generated from the image and component labelled. The connectivity and feature information is generated by the hardware process. The software data structure consists of vector lists associating component labels; symbols '1' to '3', with features.

```
                connect. info   feature info       data struct.
..................
.A11.B22..C33.....              [A,1] [B,2] [C,3]  {1:A} {2:B} {3:C}
..11..22...33.....
..11..22...33..... J(1,2)                          {1,2:A,B}   {3:C}
..111111D...33E... E(2)           [D,1] [E,3]      {1:A,B,D} {3:C,E}
.................. E(1) E(3)                         {A,B,D}  {C,E}
```

When the last label value is removed from a vector list then the features are propagated as connected objects, i.e. {A,B,D} and {C,E}.

The above algorithm is not optimal as there is a large overhead in joining linked lists whilst maintaining the integrity of their structure. Instead a more complex algorithm which relies on a group data type was implemented. This requires no list management for most JoinVals as it is immediately obvious if two values already point to the same group. The joining of two groups is relatively rare. This algorithm is detailed in Chapter 6, GRIPPR implementation.

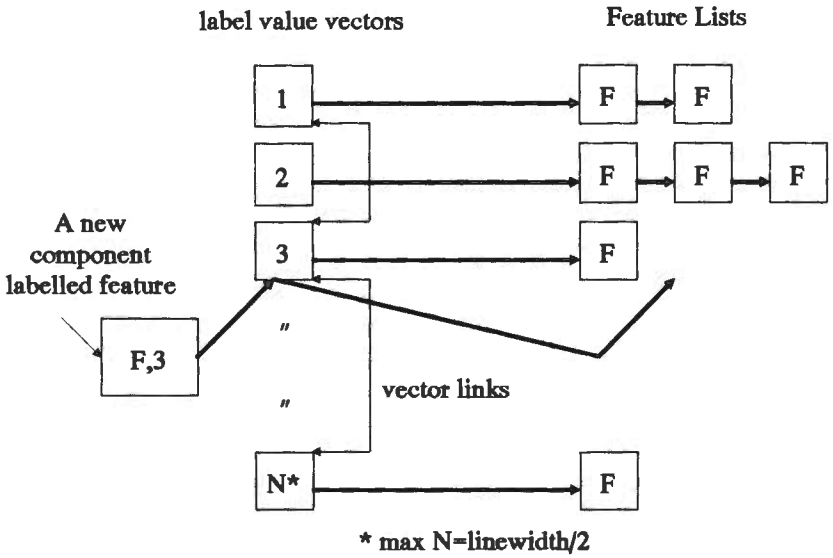


Fig 5.2: Simple connectivity data structure

5.2 GRIPPR design

Having produced algorithms for pipeline feature extraction and component labelling, it was now possible to consider the design of an image processor; GRIPPR (Generic Real-time Image Processor for Pattern Recognition).

5.2.1 PiNOLa language extensions

The first step was to simulate the interaction of the component labelling algorithm with the feature extractor. This was achieved by extending the PiNOLa language to include a component label generator called from within a PiNOLa pipeline flow

descriptor, e.g.:

```
FLOW
  Dummy <
        > operator A
        > operator B
        > ID
END.
```

The 'ID' command assigns labels to the components of the image presented by the operator 'Dummy'. The latency of the ID process is the same as the latency of the operators A and B. Thus labels generated by ID can be used to associate pixels in the transformed outputs of A and B with objects in the original image.

To generate features such as end nodes from an operator it is not necessary to have a separate stage. Instead, it is only required to have one or more piggy-back look up tables which generate a different output from the same 3 x 3 window.

This principle was included in PiNOLa with the language extensions shown in the following program.

```
BPP RSkel                          -- This part remains unchanged
 1,h7,f,s  {Maintain 8 connect}
 1 a -
 b 1 a
 - b 1

 1,h7,f,s  {Maintain 8 connect}
 1 a -
 0 1 a
 - 1 -

 1,h7,f,s  {Maintain 8 connect}
 1 a -
 0 1 a
 1 a -

 1,h7,f,s  {Maintain 8 connect}
 - 1 -
 0 1 0
 - 1 -

 0,f     {delete all pixels not at right edge}
 - - -
 0 1 1
 - - -
END

BPP dummy  {does nothing}
 1
 1 1 1
 1 1 1
 1 1 1
END
```

```
BPP EndSpot {This produces an output when the end node}
0           {of a skeleton is located}
- - -
- - -
- - -
1,s
- 0 0
- 1 0
- 0 0
END
FLOW
  Dummy   <
          >  RSkel /p1+EndSpot+/b1"E"
          >  ID
END
```

The format of the PiNOLa piggy-back feature extractor is:

```
BPPname /p1+BPPname+/b1"id1" .. /pn+BPPname+/bn"idN"
```

Thus the statement - RSkel /p1+EndSpot+/b1"E" - specifies a feedback operator RSkel with a non-feedback operator EndSpot looking at the same 3x3 window. EndSpot generates features identified by the string "E". These features are component labelled by the ID process and are then passed to a higher level process for grouping by connected object.

```
   input image         RSkel output         EndSpot output       connected object output
.................   .................   .................    (feature type + x,y)
.....xxxxx.......   .........x.......   .........E.......
....xxxxxxx......   .....xxxxxx......   .................
...xx....xx......   ....x.....x......   .................
...xx....xx......   ....x.....x......   .................
...xxxxxxxx......   .....xxxxxx......   .................
...xxx..xxx......   ....x.....x......   .................
...xx....xx......   ....x.....x......   .................
...xx....xx......   ....x.....x......   ....E.....E......   (E:10,2  E:5,9  E:11,9)
.................   .................   .................
.................   .................   .................
```

In the above pictures the stages of image transformation performed by the language extensions are shown. The input image is transformed by the feedback operator RSkel and then the piggy-back operator EndSpot generates a feature labelled 'E' by applying a different look up table to the RSkel 3x3 window.

The implementation of these language extensions in PiNOSim was quite tricky, given that the different processes do not actually run in parallel but concurrently. The time frame of 1024 pixels for each concurrent neighbourhood operator instance before switching context produced a reasonable performance, but it makes the synchronisation between features and component labels difficult.

The feature extractor was implemented in PiNOSim as a list of 512 bit look up tables that could be associated with a particular instance of a concurrent neighbourhood operator process. Each piggy-back feature extraction operator generates an entry in a timing stack, whenever the output of the feature operator is high.

The component labelling process, 'ID', was implemented as a special type of concurrent process called in the same way as the standard neighbourhood operator processes. Internally instead of a binary line-store the process accesses a line-store of label values. This enables it to construct the label value neighbourhood and so to perform the label propagation task as well as generating ValJoin data.

Two other data structures are required for the label value processing in the component labelling algorithm. The first is the end logic memory arrays; these are used to determine which value labels have become non-current in the last line of image. The second data structure is a FIFO store of label values. The label propagation task accesses the top of the label value FIFO to fetch new label values. The ValEnd logic writes label values that have become non-current into the bottom of the label value FIFO.

The component labelling process generates a timing stack containing labels, ValJoin and ValEnd data types. After each 1024 bit time frame a new Module called STAT is called. The STAT Module interleaves the feature and connectivity stacks so that they appear to have been generated in parallel. STAT also generates useful information about the numbers and types of information propagated.

Finally yet another Module, SORT is called which is passed the interleaved feature and connectivity stack. The SORT Module performs the connected object feature grouping task, and it generates a list of connected object feature maps scaled onto 16 x 16 grids which are then stored as a text file.

The benefit of including the feature extraction and component labelling processes as extensions to the PiNOLa language is that it allows experimentation with many different structures. PiNOSim allows any type of operator, pipeline or feature extraction supported by the language to be simulated with the component labelling algorithm.

5.2.2 GRIPPR simulation

At this stage it was possible to simulate the performance of many different feature extraction operators and to see how they performed for a variety of images. This demonstrated the robustness of the feature extractor based on the RSkel and BSkel operators described in the previous chapter.

When component labelling the end nodes of the background R/BSkels, there is a potential problem in associating these nodes with a foreground object. However, one of the useful properties of the R/BSkel operators is that the end nodes are always at an edge. Therefore, one pixel to the right of a background RSkel end node is a foreground pixel. Likewise one pixels below a background BSkel end node is a foreground pixel. We can therefore associate any background end node with a connected foreground object simply by expanding the objects in the image sent to the component labelling algorithm by one pixel along the left edges and one pixel along the top edges.

The following PiNOLa FLOW descriptor associates background end nodes with foreground connected objects.

```
FLOW
 dummy <
       > invert <                              -- invert image
       |        > RSkel /p1+EndSpot+/d1"1RS"
       |        > BSkel /p1+EndSpot+/d1"2BS"
       > ExpandTopLeft - ID                     -- expand image
END.
```

Where ExpandTopLeft is defined as;

```
BPP ExpandTopLeft
1,     {Expand Top}
- - -
- - -
- 1 -

1,     {Expand Left}
- - -
- - 1
- - -
END
```

This might cause some objects that were previously unconnected to become connected. However, one could restrict ExpandTopLeft to retaining the 8-connectivity of the image. This works because background end nodes are never generated between two distinct 8-connected objects. The easiest way to understand this is to see that to generate an end node a partial enclosure must be created in the R/BSkel operator window. An enclosure in a 3x3 window is only possible if there is only one 8-connected object in the window. e.g. the following 3x3 enclosure leaves no room for more than one 8-connected object.

```
- 1 -
1 0 1
- - -
```

Thus we will never generate a background node at a location where, because we are maintaining the 8-connectivity of the image, we failed to expand the image.

The purpose of the component labelling hardware is merely to associate the features that are generated with connected objects. Within this context there is no reason not to process the image passed to the component labeller so as to reduce the amount of connectivity information which needs to be propagated to the software process, provided the association task can still be accomplished. Two approaches were adopted:

- The input to the feature extraction and component labelling hardware is first passed through a low-pass spatial filter which reduces the connectivity content of the image by removing unconnected or poorly connected pixels. This reduces the amount of connectivity information which is generated by the aliasing of straight edges of image components. The same approach is adopted by Gotoh T. et al in their paper 'High-speed algorithm for component labelling'[44]. This paper describes an optimised version of the standard component labelling algorithm, i.e. no recycling of label values.

- A further optimisation of the process is possible in our case as only the component labelled features are output, not the entire image. By passing the input to the component labeller through a blocking function, as described in Chapter 3, we can substantially reduce the connectivity complexity of the image by filling in holes in the foreground connected components. This can be done whilst maintaining the connectivity of the image so that the labelling of foreground components enclosed by other foreground components is not affected.

The final description of the GRIPPR image processor is given by the following PiNOLa code. The 'smooth' pre-processing function is complex as it not only performs a low pass spatial filtering but also thickens thin components of the image to aid the feature extraction process, whilst maintaining the 8-connectedness of the image. The RSkel and BSkel operators have been modified to preserve 8 rather than 4-connectivity. Finally, the 'blocker' function performs the blocking function described above as well as expanding the top and left edges to enable the association of background features with foreground components.

```
[ This is GRIPPR ]

BPP RSkel
  1,h7,f,s {Maintain 8 connect}
  1 a -
  b 1 a
  - b 1

 1,h7,f,s
 1 a -
 0 1 a
 - 1 -

 1,h7,f,s
 1 a -
 0 1 a
 1 a -

 1,h7,f,s
 - 1 -
 0 1 0
 - 1 -

 0,f
 - - -
 0 1 1
 - - -
END

BPP BSkel
 1,h7,f,s {Maintain 8 connect}
 1 a -
 b 1 a
 - b 1

 1,h7,f,s
 1 a -
 0 1 a
 - 1 -

 1,h7,f,s
 1 a -
 0 1 a
 1 a -
```

```
 1,h7,f,s
 - 1 -
 0 1 0
 - 1 -

 0,f
 - 0 -
 - 1 -
 - 1 -
END

BPP smooth
 1,f,h10 {low-pass spatial filter and thicken}
 0 0 1
 1 0 0
 1 1 1

 1,f     {Thicken top}
 - 0 -
 - 0 -
 1 1 1

 1,f     {Thicken left}
 - - 1
 0 0 1
 - - 1

 1,h8,f  {thicken, override background 8 connectivity}
 - - -
 - 0 1
 - 1 0

 1,h8,f  {thicken, override background 8 connectivity}
 - - -
 1 0 -
 0 1 -
 1,s,i,f,h2 {smooth}
 - 1 -
 1 0 1
 - - -

 1,h7,f,s,i {Maintain 8 connect}
 1 a -
 b 1 a
 - b 1

 1,h7,f,s,i
 1 a -
 0 1 a
 - 1 -

 1,h7,f,s,i
 1 a -
 0 1 a
 1 a -

 1,h7,f,s,i
 - 1 -
 0 1 0
 - 1 -
 END

BPP invert
 0,i
 - - -
 - 1 -
 - - -
END

BPP dummy
1
1 1 1
1 1 1
1 1 1
END
```

```
BPP block
 1,h7,f,s,i {Maintain 8 connect}
 1 a -
 b 1 a
 - b 1

 1,h7,f,s,i
 1 a -
 0 1 a
 - 1 -

 1,h7,f,s,i
 1 a -
 0 1 a
 1 a -

 1,h7,f,s,i
 - 1 -
 0 1 0
 - 1 -

 1,f     {blocker function}
 - 1 -
 1 0 -
 - - -

 1,f     {expand left}
 - - -
 0 0 1
 - - -

 1,f     {expand top}
 - 0 -
 - 0 -
 - 1 -
END

BPP PRTspot
0
- - -
- - -
- - -

1,h2
0 0 0
0 1 0
- - -
END

BPP PRBspot
0
- - -
- - -
- - -

1,h2
- - -
0 1 0
0 0 0
END

BPP PDRspot
0
- - -
- - -
- - -

1,h2
- 0 0
- 1 0
- 0 0
END
```

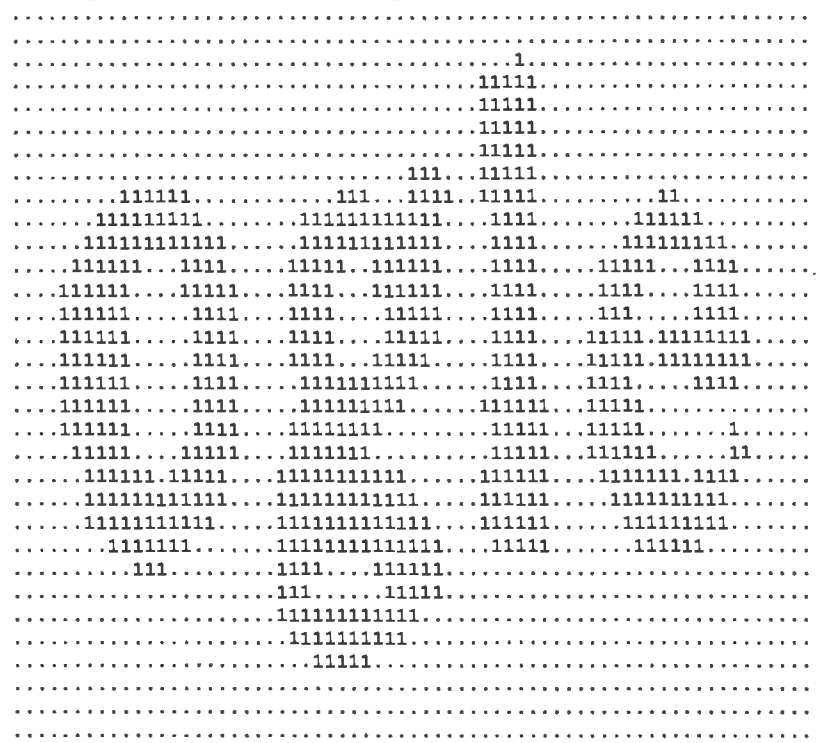
```
BPP PDLspot
0
- - -
- + -
- - -

1,h2
0 0 -
0 1 -
0 0 -
END

FLOW
 smooth <
        >  dummy  <
        |         > RSkel /d/p1+PRTspot+/b1"1RT"/p2+PRBspot+/b2"2RD"
        |         > BSkel /d/p1+PDLspot+/b1"3DL"/p2+PDRspot+/b2"4DR"
        > block - ID
        > invert  <
                  > RSkel /d/p1+PRTspot+/b1"5RT"/p2+PRBspot+/b2"6RD"
                  > BSkel /d/p1+PDLspot+/b1"7DL"/p2+PDRspot+/b2"8DR"

END
```

The effect of GRIPPR on the following scanned image can be seen in the feature maps generated from the image by the PiNOSim execution of the above program.



As there are four connected objects in the above image, PiNOSim generates four feature maps. The number labels in the feature maps are taken from the FLOW definition in the GRIPPR specification.

```
New object: Ax= 50 Ay=  8 W=14 H=15
                    ||||||||||||||||
N= 1  x=11  y= 1  -           1
N= 2  x=12  y=16  -
N= 4  x=15  y=13  -
N= 1  x=15  y=12  -          5
N= 6  x=12  y=12  -
N= 3  x= 1  y=11  -          8
N= 2  x=15  y= 9  -
N= 4  x=16  y= 8  -        3       4
N= 3  x= 8  y= 8  -               2
N= 8  x=10  y= 6  -
N= 5  x=10  y= 4  -3
                  -           6  1
                  -              4
                  -
                  -
                  -           2

New object: Ax= 41 Ay=  2 W= 6 H=21
                    ||||||||||||||||
N= 4  x=16  y=16  -           1
N= 2  x=16  y=16  -
N= 3  x= 1  y=15  -
N= 8  x= 1  y=13  -3
N= 3  x= 1  y= 4  -
N= 1  x=11  y= 1  -
                  -
                  -
                  -
                  -
                  -
                  -
                  -8
                  -
                  -3
                  -                42

New object: Ax=  6 Ay=  7 W=15 H=17
                    ||||||||||||||||
N= 2  x=10  y=16  -           1
N= 4  x=16  y=12  -
N= 6  x=10  y=12  -
N= 8  x=11  y=11  -          5
N= 7  x= 8  y=11  -
N= 3  x= 1  y=10  -
N= 5  x=10  y= 4  -
N= 1  x=11  y= 1  -
                  -
                  -3
                  -       7  8
                  -         6     4
                  -
                  -
                  -
                  -         2
```

```
New object: Ax= 24 Ay=  6 W=14 H=22
                    |||||||||||||||
N= 1  x=15  y= 1  -        1    6 1
N= 2  x=10  y=16  -
N= 4  x=16  y=14  -
N= 3  x= 1  y=14  -        5
N= 8  x=10  y=14  -
N= 6  x=10  y=14  - 3    7  86    4
N= 5  x=10  y=14  -
N= 7  x= 6  y=14  - 8
N= 7  x=12  y= 9  -            7
N= 8  x= 2  y= 8  -
N= 4  x=16  y= 6  -
N= 8  x=10  y= 6  -
N= 6  x=10  y= 6  -
N= 7  x= 7  y= 6  -3    7   865   4
N= 3  x= 2  y= 6  -
N= 5  x= 8  y= 4  -          2
N= 6  x=13  y= 1
N= 1  x= 8  y= 1

NoPatterns=4 NoNodes=57 Average=14
```

Once GRIPPR had been fully designed and simulated, work began on the implementation of a GRIPPR machine.

Chapter 6

GRIPPR Implementation

To evaluate any complex system it is necessary to get a flavour of the detailed low-level implementation. This is because systems that may appear to be theoretically elegant can become less so at the implementation stage. Neural nets are a good example of a theoretically elegant idea which is sometimes limited by incompatible implementation technology. Neurons are simple to construct as mathematical equations but when these are transferred to 2-dimensional silicon with a limited interconnect the implementation can become turgid. Biological neural systems utilise a 3-dimensional interconnect wherein the number of neurons a connection can reach grows as a cube law of its length, a technology not presently available. Cellular arrays usually have a very elegant hardware implementation as their structure is regular, 2-dimensional and tessellates well. However, the detail of programming such arrays to perform non-local neighbourhood tasks and input/output operations can result in lower performance than might be expected from a comparison of the relative Pixel-Ops per second statistics.

The good technological fit of the GRIPPR approach to image processing for pattern recognition should become clear as the detail of its implementation is discussed. Performing only image local operations in the hardware produces a relatively simple machine with a deterministic performance. Performing non-local operations on a von Neumann processor allows the utilisation of sophisticated data structures for determining the connectivity of the image.

To implement a system based on the GRIPPR design it was necessary to find a convenient task. A4 document OCR was selected as a good demonstration task as many systems already exist for comparison, and imaging hardware is readily available. The TAXAN TX-300 scanner was chosen as a suitable imaging device because it was cheap and has a very simple, fully documented, image interface.

6.1 Specification

The specification of the GRIPPR hardware implementation is derived from the specification of the TX-300 scanner. The TX-300 scanner generates an A4 binary

image at 300 d.p.i. in 9.9 seconds. The average throughput of the scanner is therefore 850K pixels per second. This is made up of an 800ns (1.25Mhz) video clock and a line-sync between lines of image lasting 965us.

To process the image in real time, GRIPPR must be able to deal with the maximum text density in standard documents. The maximum text density is of course almost infinite, but in the context of text which is recognisable when scanned at 300 d.p.i. the maximum text density was found to be about 100 characters per square inch. e.g.

```
12345678901234567890
22222222222222222222
33333333333333333333
44444444444444444444
55555555555555555555
```

This gives 9350 characters per A4 page which produces a maximum throughput of 944 characters per second for the TX-300 scanner. A throughput of 1000 characters per second was therefore made the design goal of the system.

The earlier simulation of GRIPPR gave a maximum of about 20 data items (Features,EndVals and JoinVals) per character input. Therefore, the minimum data bandwidth for the GRIPPR hardware to software interface was 20,000 data items per second.

Using 9 bit component labels the bits per item are as follows:

- Feature Type: identifier=3 bits, Xpos=12 bits, component label=9 bits, Total = 28bits
- EndVal Type: identifier= 3 bits, component label=9bits, Total = 12 bits
- JoinVal Type: identifier=3 bits, component labels=18bits, Total = 21 bits

Clearly, the maximum pixel by pixel output bandwidth of GRIPPR is potentially very high as a feature data type could be generated every clock cycle giving 28 / (800 x 10^{-9}) = 35 M bits per second. However, this is not a problem as provided the sequence of the data items is maintained (including an end of line flag), the data structure will still be valid. A simple FIFO output buffer can therefore be used to equalise the peak output rate to the average throughput. The implication of this type of output buffer is that the different data types are assigned the same width in the store, as the type of the data will not be known until it has been read by the software process. If data is transferred as bytes then the minimum hardware to software bandwidth will be 32 x 20,000 which is 640,000 bits per second.

A B006 transputer evaluation board was available as a potential processor for GRIPPR. Some simple calculations based on the GRIPPR simulation in PiNOSim indicated that one 20Mhz T800 transputer would be able to perform the feature to connected object stream conversion at the necessary real-time rate of 20,000 data items per second. Furthermore, a transputer link provides a very simple high speed interface of 1.2 M Bytes per second, which is ample for GRIPPRs requirements.

The GRIPPR hardware could therefore be specified as a black box with a reset, video clock, video data and line-sync input, and a transputer link output having the same functionality as the GRIPPR PiNOSim simulation.

6.2 Hardware optimisation

Feature Extractor optimisation

A direct translation of the PiNOLa description of GRIPPR would have required two lines of image store per neighbourhood operator stage. Obviously, there is no need for the 'invert' operator as an inversion can simply be implemented as a discrete inverter in the pipeline.

There are six remaining operators; smooth, blocker, RSkel, BSkel, and background RSkel and BSkel. With two line store stores per operator this would give 6x2 = 12 bit wide line store. However, just as standard parallel neighbourhood operators can be executed in a piggy-back fashion sharing the same line store, the same principle can be applied to feedback operators which can share the same pre-processed line store. As five of the operators occur at the same level, four of the twelve line store elements are redundant. Thus a single 9 x 4K deep FIFO provides all the line store for the GRIPPR feature extractor.

Connectivity optimisation

Initially it was the intention to time multiplex the EndVal connectivity logic as discussed in chapter 5. However, this generated a very complex hardware interconnect so instead use was made of the 965us line-sync provided by the TX-300. The GRIPPR hardware therefore has two modes of operation. In the first mode during image data transmission the machine is entirely event driven by the input video clock. During a line-sync, control switches to an internal 10Mhz clock which is used to cycle through the previous and current label value tables to generate EndVal types for component labels which became redundant in the previous line of image data.

Only 511 label values are used, even though theoretically a 2544 pixel wide image could have 1271 separate connected components. In practise however, the aliasing constraints on imaging small objects mean that an image containing 511 separate connected components on a 2544 pixel line is highly unlikely. The advantage of limiting the label values to 511 is that the label line store and the non-current label value store can both be implemented by single 9 bit wide standard FIFO integrated circuits, which simplifies the design.

Output buffer optimisation

Each pixel clock cycle can result in a feature data item or JoinVal data item being written into the GRIPPR FIFO output buffer. The sequence is not important as the feature component label value is still valid after a JoinVal. If speed is critical, a two stage output buffer could be designed so that both JoinVal and Feature data items could be written simultaneously. The first stage of the buffer would have a wide bus 'funnelling' into the a second 32 bit stage by means of some control logic.

However, as speed is not critical (the input clock cycle is 800ns and the FIFO cycle time is 100ns) a more elegant solution is to time multiplex the output buffer. This works by writing feature data items during one phase of the input clock and JoinVal data items during the other phase of the clock.

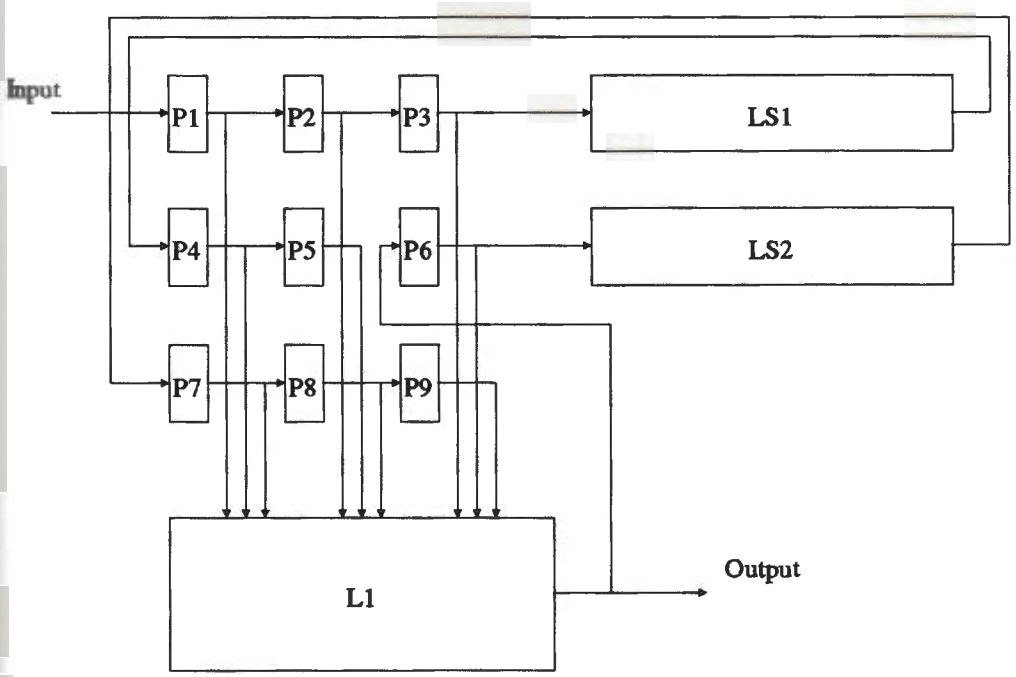


Fig 6.1:Pipelined 3x3 binary feedback neighbourhood operator

6.3 GRIPPR hardware

Feature extractor

The basis of the feature extractor is the pipeline feedback 3x3 binary neighbourhood operator shown in Fig.6.1. The line-store components LS1 and LS2, and the 1 bit latches P1..P9, are used to construct the 3x3 neighbourhood. The line-store components are implemented using a 4K x 9 bit FIFO. The line-store FIFO is automatically set to the width of the input image minus three by reading in the first line of the image minus the first three pixels. Thereafter the FIFO mimics a shift register of that length by executing both a read and a write cycle for each input pixel.

At each pixel clock cycle, data is propagated in the direction of the arrows shown in Fig 6.1. The outputs from the 3x3 window, P1..P9, are passed to the logic component L1 which performs the particular operator function. In the case of a 'feedback' operator, the output of the operator is fed back into the latch P6; there is therefore no direct connection between P5 and P6. All the operators used in GRIPPR turn out to be of the feedback type.

The logic component L1 is constructed from a 512x1 bit PROM. The 3x3 window simply addresses the PROM as a look-up-table to generate the output value in each clock cycle. Piggy back operators as described earlier can simply by implemented by a 512xn bit PROM where each field in the look-up-table represents a different

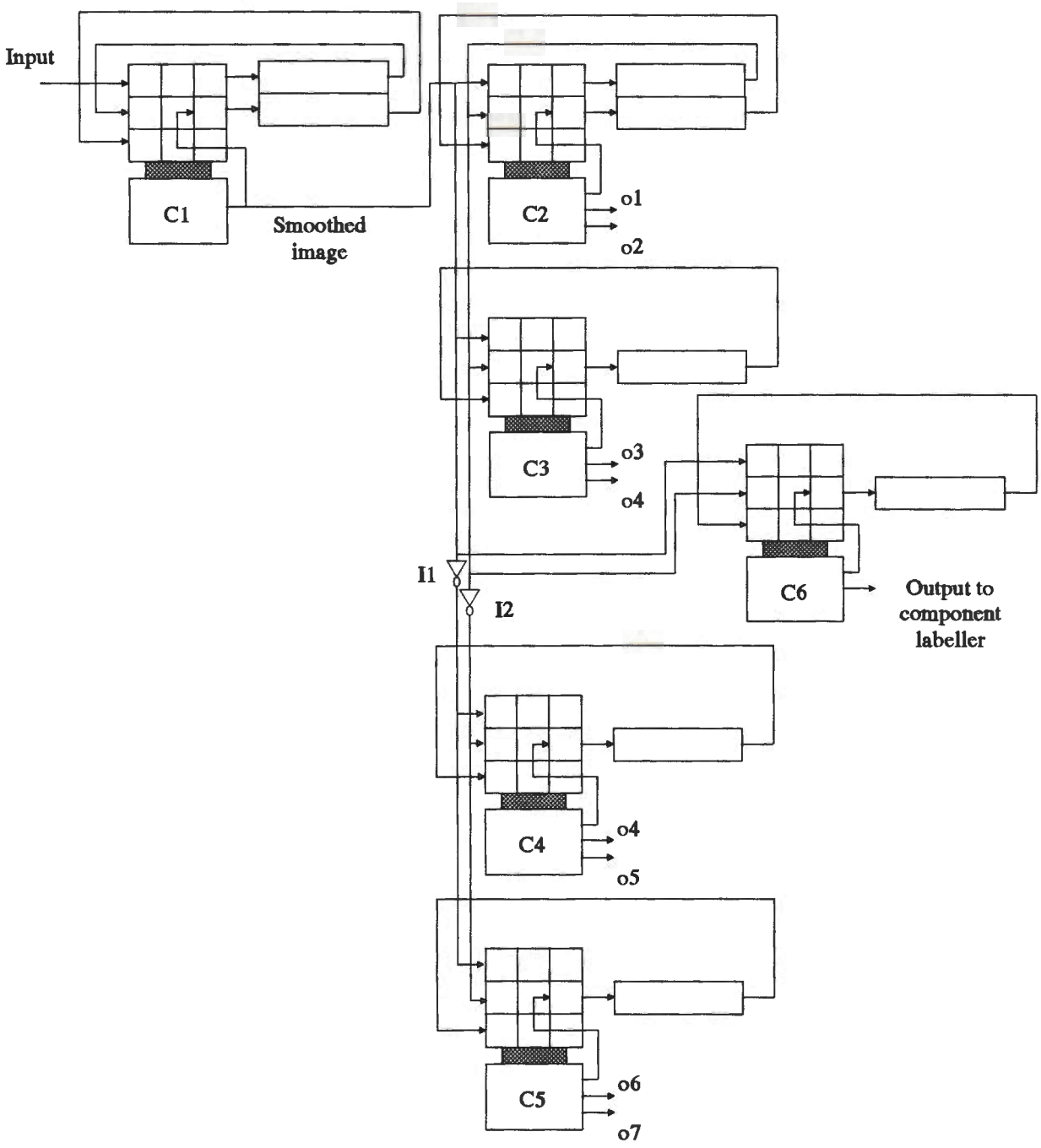


Fig 6.2: The GRIPPR feature extraction pipeline

function, only one of which is fed back to the 3x3 window.

It seems natural to show the pixels entering the neighbourhood operator at the top left and being propagated left-right, top-down. However, one should realise that assuming a left-right, top-down raster scan the 3x3 image presented by P1..P9 is rotated by 180 degrees, as the pixel represented by P1 is the newest and therefore the rightmost pixel in the bottom row of the window.

The speed of the device is only limited by the speed of the FIFO line store as 512 bit RAM and latch access times are relatively fast. To execute a Read/Write cycle on

the 80ns access time FIFOs which were used takes a minimum of 100ns, giving a maximum throughput of 10 M pixels/second. FIFOs with cycle times of 35ns are available if greater throughput is required.

The complete feature extractor diagram is shown in Fig.6.2 . Note the line-store optimisation of the neighbourhood operators C2..C6, that all the share the same first line-store component.

The operator C1 executes a smoothing function which reduces the number of small features in the image by effectively performing a form of low-pass spatial filtering. This filtering also improves the performance of the component labelling function of GRIPPR by reducing the instance of label joins and label terminations.

The operators C2..C5 perform the R/BSkel transformations and the end node feature extraction. This is implemented by using the first 3 output bits of 512x4 bit PROMs. One bit is fed back to the 3x3 window to generate the R/BSkel transformation, the other two bits signal the presence of different types of end-nodes in the image.

The operator C2 performs the RSkel transformation and its outputs o1 and o2 represent the top and bottom end nodes of that function. The operator C3 performs the BSkel function and its outputs o3 and o4 represent the left and right end nodes of that function.

Instead of specifying different background functions, the operators C4 and C5 are identical to C2 and C3 except that the input image is inverted by the logic gates I1 and I2.

The operator C6 has three functions. First it provides the correct pixel delay, so that labels that are assigned to the feature outputs o1..o8 actually correspond to the same connected component in the original image. Second, the operator performs the connectivity maintaining expansion required so that background nodes can be associated with their corresponding foreground objects. Third, the operator performs the blocking function, described earlier, which reduces the amount of connectivity information which needs to be propagated by the GRIPPR component labelling hardware.

Anyone familiar with state of the art semi-custom gate array technology will note that the whole feature extractor, minus the line store components, could easily fit onto a single chip. Though 57 latches and 14 512x1 bit look-up-tables are used, much of this circuitry would be redundant in a highly optimised design. For instance, many of the latches could be shared and the look-up-tables could be replaced by the OR-AND-OR structure referred to in the PiNOLa specification chapter.

Such a device would still be limited by the external line store access time. High speed line store components[62] designed for HDTV applications could be used to give a maximum throughput of 50M pixels/second.

Finally, it is certainly technologically feasible to place the entire circuit including the line store, which is only 8 bits wide and a few thousand bits deep, on a full custom VLSI chip. Such a device could have a cycle time approaching 10ns giving a maximum throughput of nearly 100M pixels/second.

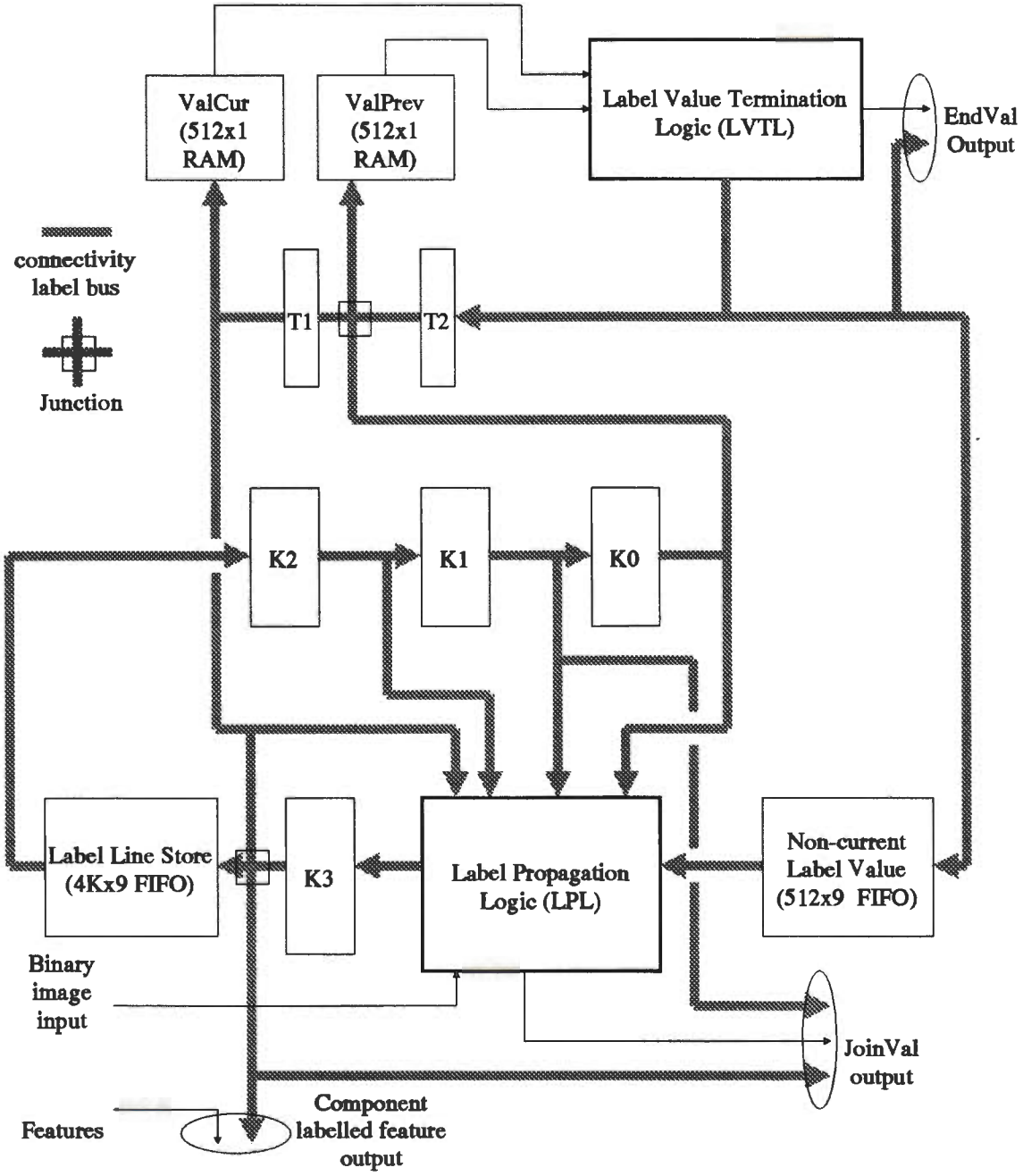


Fig 6.3: The GRIPPR component labelling hardware

Component Labeller

The component labelling hardware appears to be an order of magnitude more complex than the feature extraction hardware. This is because the feature extraction complexity is neatly hidden in the look-up-tables whereas the complexity of the component labeller is expressed in its fairly hideous inter-connect.

512 component labels are used, 0 always being assigned to background pixels. All the label buses shown in Fig. 6.3 are therefore 9-bits wide.

The label line store is constructed from a 4Kx9bit FIFO which is initialised in the same way as the feature extractor line store to mimic a shift register which is image-width minus 3 bits deep. The component label and feature line store FIFOs share the same read and write clocks.

The label value FIFO which is used to store non-current label values is constructed using a 512 x 9bit FIFO. This is initialised to contain the 511 non-zero label values.

The local label neighbourhood used to describe the component label propagation rules has the following visual mapping assuming a top/down, left/right scan.

K0 K1 K2

K3

This corresponds to the 9 bit label latches K0..3 in Fig. 6.3, though again the mapping is not obvious.

The current line value memory ValCur and the previous-line value memory ValPrev are implemented as 512 x 1 bit RAM's. ValCur and ValPrev are initialised to contain all 0's before each line of image is input.

The component labeller has two modes of operation

In the first mode the component labeller is controlled by the input pixel clock. Whilst in this mode the Label Propagation Logic (LPL) assigns labels to input pixels. Background pixels are always assigned the value 0. Whenever a new foreground connected component as located, i.e. when K0..K3 are all null and the input pixel is set, then a new label value is fetched from the label value FIFO by the LPL and inserted into the label store K3 at the next clock cycle. If the input pixel is connected to any of the neighbourhood value stores, i.e. one of K0..K3 is non-null, then the value of the non-null value store is propagated to K3 at the next clock cycle.

The other function of the LPL is to signal when two label values are found to be labelling the same connected component in the image. This is done whenever K1 and K3 have different values and K2 is null. The values K1 and K3 are then propagated to the software system so that features which may have been assigned to the values in K1 and K3 separately are known to originate from the same connected component of the image.

The LPL is implemented by a set of 22V10 Programmable Logic Devices (PLDs) which perform the null and inequality comparisons, and by a set of tri-state buffers which enable the different values to be routed into the K3 label store. The operation of the LPL is controlled by another PLD executing the following rules (InPic is a boolean value representing the input pixel).

*routeK3 := /(K3=0) * (InPic)*

*routeK1 := (K3=0) * (InPic) * /(K1=0)*

*routeK2 := (K3=0) * (K1=0) * (InPic) * /(K2=0)*

*routeNew := (K0=0) * (K1=0) * (K3=0) * (InPic)*

routeNull := /InPic

*Join := (K2=0) * /(K1=0)* /(K2=0)* (K1K2)*

The actual implementation is somewhat more complicated due to the complex timing characteristics of the various circuits, but the basic logic is the same.

In this mode of operation the ValPrev and ValCur memories are addressed by the label stores K0 and K3 respectively. For each clock cycle the bit addressed by K3 is set to 1 in the ValCur memory and the bit addressed by K0 in the ValPrev memory is set to 1. The effect of this is that after a line of image is input, ValPrev has a bit set for every value that was previously in the label line store and ValCur has a bit set for every value that is currently in the label line store.

In the second mode of operation the component labeller is controlled by an internal 10Mhz clock. This mode of operation is entered during the line-sync between lines of image. The label value termination logic (LVTL) takes the internal clock and cycles through the 511 non-zero addresses of the ValPrev and ValCur memories. Whenever a value is found whose corresponding bit is set in the ValPrev memory and not set in the ValCur memory an EndVal is signalled . The value which is no longer current is thereby propagated to the software system so that the value can be dissociated from any features which may have been labelled by it, and so that the completeness of connected objects can be revealed in real-time. When an EndVal is signalled the value is also written into the label value FIFO so that it can be re-used in the next line if needed.

As it reads the ValCur and ValPrev memories the LVTL also sets all values to 0 so that the memories can be used again for the next line of the image.

This method implies that there has to be sufficient time between lines of image to cycle through the ValCur and ValPrev memories. However, if there was not sufficient time to do this between lines of the image the LVTL could operate in a time-multiplexed fashion with two sets of memories. The sets of memories would alternate between performing the read and write states.

Whilst the component labeller may appear complex it would actually be simpler than the feature extractor to implement in semi-custom or custom technologies as are there are no look-up-tables masking the complexity.

The need to have the non-current label values in a physical FIFO store appears to limit the speed of the device. However, on inspection of the algorithm it becomes clear that at most one new label is required every two clock cycles. Thus the maximum speed of a circuit designed around a 35ns cycle time label FIFO would be 57 M pixels/second. In Practice the implementation limit is therefore the propagation delay implicit in the component label routing equations; as determining an inequality between two numbers, then using that information to enable a 9 bit latch, takes a significant multiple of the minimum gate delay for a particular technology.

A TAAC20 gate array with 2000 gates should be able to perform all the non-memory functions associated with the component labeller. Assuming a propagation delay of about 10 gate delays taking 4ns each, a throughput in the order of 25M pixels/second should be feasible. Full custom design might give about 50M pixel/second throughput.

6.4 Fabrication and testing

To reduce the cost of implementation and allow for hardware de-bugging it was decided to implement GRIPPR using two rather than four layer PCBs. The PCB CAD system available was the now outdated REDAC and REDBOARD schematic design and layout software from RACAL.

Initially GRIPPR was designed as four 'Euro' sized cards to take advantage of the inherent modularity of the system. The modules were: feature extractor, label propagation logic (LPL), label value termination logic (LVTL) and output buffer. However, in reality the interconnect between these modules was too complex to fit easily onto a back plane. The design for the machine was therefore rationalised to fit onto a single board measuring roughly 12″ by 14″.

Electronic considerations

De-Coupling: In designing a two layer board there is a danger that noise due to ground bounce will be a problem. Particularly as the circuit is highly parallel with about 200 latches changing state on a common clock cycle. The situation was further aggravated by having to use unnecessarily fast components; the 9-bit latches used for neighbourhood operations and component label propagation are only available in circuits with propagation delays of less than 10ns.

To overcome this potential hazard great attention was paid to the ground and power layout. An interleaved comb structure was adopted with power and ground running along the axis of the I.C.s The gap between power and ground lines was designed so that surface mount ceramic chip capacitors could be soldered underneath the chip carriers at the ground and VCC pins of the I.C.s; this provides very good de-coupling as the capacitors are physically close to the power pins and the capacitor lead connections have minimal inductance. Leaving the exact layout of the de-coupling capacitors out of the schematic and PCB design process also reduced the complexity of the design task.

Timing: The hardware was initially designed to generate internal timing signals, for controlling the state of busses and the output buffer time-multiplexing, by dividing down a high frequency internal clock. However, there are two problems with this approach. First the need to synchronise with external signals means that the worst case synchronised clock cycle will always be shorter than the actual pre-synchronised clock cycle. Second, the high frequency internal clock needed to alleviate the above problem, and to provide the timing resolution necessary for internal signals, would itself introduce further hazards in a circuit which is already prone to noise.

A better approach was found to be the generation of internal timing signals by using digital delay-lines. This is achieved by taking the external video clock and passing

this signal through a delay-line before ORing the delayed signal with the original to generate a higher frequency clock. This makes the operation of the machine entirely event driven during image transmission.

Transputer Links: The T800 transputer has two modes of communication. The standard transputer link throughput of 800K bytes/second is achieved by sending acknowledge bits after each 8 bit packet has arrived. The other, faster mode of communication available on the T800 is optimised for 32 bit transfers and achieves a throughput of 1.2 M bytes/second with a greatly reduced CPU overhead. The T800 communicates in the second mode by issuing a pre-acknowledge to the 8-bit packet transfer after the first bit is received. The effect of this is that acknowledge and data bits are on the link-in and link-out data transmission lines at the same time. To enable the second mode of operation over the required 2 meters of cable great care had to be taken to minimise cross-talk. Individually screened twisted pair cable was used which was terminated and earthed at both ends as near to the line connections as possible.

Fabrication

The PCB layout Gerber photoplot files generated by the PCB CAD system were sent by e-mail direct to the Manchester University PCB fabrication facility. Previously the PCB designed for the smoothing circuit in chapter 3 was plotted 2:1 on to drafting film and given to a local company which was a rather laborious and error prone process.

The code to program the feature extractor PROMs was generated from PiNOSim by translating the internal look-up-tables for each neighbourhood operator into a suitable text file format.

Testing

In order to get a reasonably complex circuit such as GRIPPR working, a great deal of thought has to be given to how the circuit can be tested and de-bugged. Simply plugging all the components together would probably produce a circuit with a great number of errors, making the isolation and debugging of particular errors difficult. GRIPPR was therefore assembled and tested in stages, corresponding to the different modules of the machine.

To enable the functional characteristics of the machine to be tested various programs were written. A test program was written to feed images pixel by pixel from a PC to the machine, and OCCAM programs were written to take the output from GRIPPR for decoding and display.

The electronic performance was tested using a 16 channel digital storage scope, an event counter and oscilloscopes.

First the input, timing and output circuitry was assembled. Communication bugs were corrected at this stage by sending reset signals from the PC to the transputer via GRIPPR. Next the feature extraction pipeline was assembled and this was tested by extending PiNOLa to include an 'export/import' facility, so that images could be output from PiNOSim for external processing and then imported back into the simulation for display. It was very satisfying to see the same images being generated

by software and hardware.

The component label propagation and label value termination logic was difficult to test in stages as they are highly interdependent. One useful test was to throw an image with many thousands of connected objects at GRIPPR and see if it was able to keep recirculating the 511 connectivity labels without dropping any. As the other parts of GRIPPR had already been verified, the bugs that were found at this stage were relatively easy to track down.

For example it was found that the machine was able to recycle label values up to a certain value and then the output buffer would behave erratically. This was traced to a contention problem on a tri-state bus caused by inadequate pull-up resistors. The bus drivers produced glitches on the power lines for label values with several bits set, which in turn corrupted the internal pointers inside the FIFO output buffers. Once identified this and other similar problems were quickly remedied.

6.5 Software optimisation

Before looking at the OCCAM code implementation of the feature stream to connected object stream algorithm it is necessary to look at the optimisations used for efficient real-time performance on a transputer. Optimisation of the algorithm is required as the core process of maintaining the connectivity data structure is indivisible i.e. it would be difficult to divide this process for parallel execution as it relies on an process-global data structure. Thus the connectivity algorithm could form the potential bottle-neck for a system based on GRIPPR.

Three types of Optimisation were found to be useful, i.e. throughput was increased by more than 25%.

Using single dimension array access

The obvious translation from Modula-2 record types to a language with only simple types is to replace the record type with a two dimensional array. The first dimension being the pointer to the record and the second dimension being the type index. However, this requires two levels of indirection for each record element access. A more efficient translation is to have a separate one dimensional array for each record field.

Thus, `Stack^StackPtr.FieldA := v`

becomes, `Stack_FieldA [StackPtr] := v`

not, `Stack [StackPtr] [FieldA] := v`

Avoidance of static chaining

Memory access to global variables in OCCAM is by means of a static chain through the calling sequence of procedures. For an algorithm which relies heavily on the processing of global data structures, and which could have a reasonable depth of procedural calls such as the connectivity algorithm, static chaining would be very

inefficient.

There are two ways around this problem in OCCAM. One can use the OCCAM 'RETYPE' command to bring global variables into local scope when the procedure is called. Alternatively, if the procedure is only called from one location, the procedure code can simply be included as a routine in the calling process. The latter approach can be adopted relatively elegantly within the OCCAM folding editor, which can structure very complex routines into procedure like statements. The result of pursuing this optimisation is that almost the entire connectivity algorithm is coded as one procedure except for a few procedures that are called from many different parts of the algorithm. The reason for not including all procedures in the main process is to keep the size of code as small as possible. This is important so that as much code as possible will fit into the T800 processor's fast on-chip memory.

Use of on-chip memory

Access to the T800 4K byte internal memory is at least three times faster than off chip external memory access. Rather than organising this memory as a cache (which would produce performance effects that would be difficult to predict) the transputer simply places the internal RAM at the base memory address. OCCAM makes good use of this fact by placing scalar values and pointers in a WORKSPACE placed at the bottom of memory. Provided that the program does not use a great number of such variables, all pointers and loop index variables will have very fast access times, which has a dramatic effect on the performance of the code. Normally a program uses under 4K bytes of WORKSPACE thereafter procedure code is placed in the on-chip memory in program lexical order. The reason that single element variable access is more critical than code access is that the transputer instructions are only 8 bits wide whereas the memory access is 32 bits. Therefore four instructions are fetched simultaneously for every code fetch cycle. Care has to be taken to minimise the size of the WORKSPACE and procedure code in order that critical code runs in on-chip memory. The lexical order of declaration of procedures also determines which code runs in on-chip memory, which can be a problem as there is no forward reference facility in OCCAM.

For a comprehensive discussion of transputer performance maximisation see 'Transputer Technical Notes'[63].

6.6 GRIPPR software

The GRIPPR software takes in a stream of Feature, JoinVal and EndVal data items and generates a stream of connected objects. The input stream comes from the GRIPPR hardware and the output goes to an application-specific recognition stage. A connected object generated by the hardware is defined as an x,y location in the input image, an absolute height and width, and a list of features. Each feature has a type corresponding to the outputs o1..o8 from the GRIPPR feature extraction hardware. Each feature also has an object relative x,y co-ordinate which is normalised onto a 16x16 grid. The largest dimension of the object is scaled onto the grid and all the other feature positions are scaled by the same amount, which leaves the aspect ratio of the object unchanged.

Input buffer

The rate at which the software consumes data items depends on their effects. In particular, when the last component label value that is assigned to an object is terminated by an EndVal, the software process must group the features of that object into a list, scale the features and output them to the application specific process.

In order to balance the throughput of the GRIPPR software with the GRIPPR hardware, the FIFO output buffer was made large enough to contain the data generated by one line of text at the maximum text density. The reason for this is that the line of image corresponding to the bottom of a line of text will cause all the connected objects, i.e. characters, in that line to become complete, so all the connected object grouping, scaling and transmission is concentrated in that image line. Given 100 characters per line and 20 data items per character a 2000 item/word deep FIFO should guarantee that when the software process is fully saturated, it is never waiting for input.

Algorithm

There are many ways to implement the connectivity algorithm described in the previous chapter. Some ideal characteristics of a good algorithm might be:

- There should be as little indirection as possible between feature input and assigning that feature to the connectivity data structure, as indirection takes time and features are the most common data item.

- Joining two components should not require all the structures that point to those components to be changed.

- The algorithm should never have to resort to searching all the data structures for references, e.g. when an object becomes complete all the features belonging to that object must be directly available.

- As there are many more component label value assignments than connected components, checking when an EndVal causes a connected component to become complete must be quick.

The simplest algorithm which was mentioned in chapter 5 constructs a linked list of component label value vectors which point to lists of features. JoinVal simply joins label value vector lists and features are appended to the corresponding label value vector feature list. An EndVal causes a label value vector to be dissociated from its vector list and its feature list to be appended to another member of the vector list. When an EndVal causes a label value vector which is not linked to any other label value vector to terminate, its feature list constitutes the connected object.

However, in practise there is no way of linking two lists without some conception of the top and bottom, which implies either searching each vector list for top and bottom each time they are joined, or having top and bottom pointers from each cell which must all be amended from one list when it joins onto another list. Also, continually re-assigning feature lists to new label value vectors every time an EndVal is encountered is inefficient because many component label values never actually point to any feature.

To overcome these problems, the concept of a 'group' is introduced. A group associates label value vectors with features. This adds one level of indirection to feature processing as the label value vectors now point to groups and the groups point to the feature lists. However, JoinVals are now faster as most of the time a null label value vector can simply be assigned to the same group as its non-null partner. When two non-null label value vectors are joined, instead of having to update all the group pointers in one list, the groups are just linked together.

A connected object is complete when no label value vector points to it, this can be determined by a vector counter. Whenever a feature arrives and its label value vector is null it is assigned a new group and the group's vector counter is initialised to 1. As new label value vectors join the group the counter is incremented. As EndVal types dissociate label value vectors from the group, the counter is decremented. When the counter reaches 0, which can only happen once for each group, any groups that are linked to that group are checked. If all counters are 0 the connected object is complete.

Data Structures

There are three complex data types, corresponding to, the label value vector, the group and the feature types. These complex types break down into one-dimensional arrays as described in the software optimisation section. The following OCCAM source code is included so that an idea of the small size and low complexity of the data structures can be gained. The label value vector type is referred to as IDVec, the group type is referred to as GroupVec and the feature type is referred to as NodeVec.

```
VAL INT MaxIDVec IS 512 :
VAL INT MaxNodeVec IS 5000 :
VAL INT MaxGroupVec IS MaxIDVec :
[MaxIDVec] INT      IDVec.Group:
[MaxNodeVec] INT    NodeVec.NType:
[MaxNodeVec] INT    NodeVec.Ab:
[MaxNodeVec] INT    NodeVec.Ot:
[MaxNodeVec] INT    NodeVec.NextN:
[MaxNodeVec] INT    FreeNodeVec:
[MaxGroupVec] INT   GroupVec.TopGroup:
[MaxGroupVec] INT   GroupVec.NextGroup:
[MaxGroupVec] INT   GroupVec.PCount:
[MaxGroupVec] INT   GroupVec.TopN:
[MaxGroupVec] INT   FreeGroupVec:
INT TopFreeNodeStack:
INT TopFreeGroupStack:
```

Fig 6.4 shows a visual 'snap-shot' of the connectivity data structures describing one incomplete connected object. During operation of the algorithm the IDVec.Group pointers change continuously as new label values are joined to existing label values and then the old values are terminated. The Group and Feature data items persist until no label values point to them, at which time all the Features belonging to the Group list represent one connected object.

The label value vector array (IDVec.Group) has a fixed dimension of 512 as there are at most 512 current component label values available in the GRIPPR hardware. The Feature (NodeVec) and Group (GroupVec) data types are implemented as RECORD types with pointers to physical NodeVec and GroupVec locations being held on a stack. When a Feature or Group element is requested a pointer is fetched

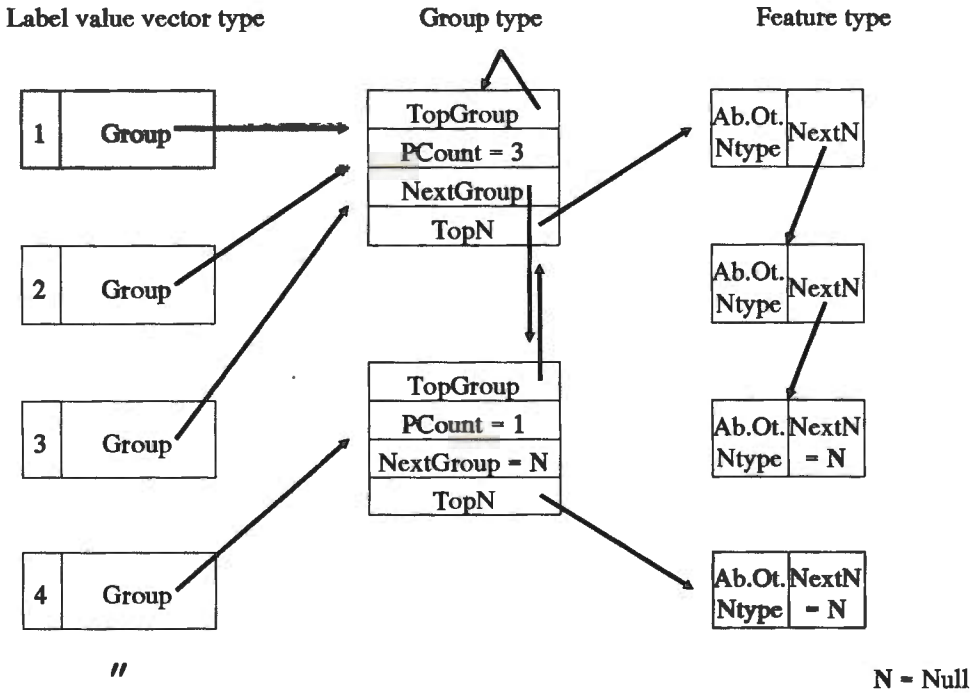


Fig.6.4: Connectivity Data Structure 'snap-shot'

from the top of the corresponding stack. When a Feature or Group becomes redundant, i.e. when a connected object becomes complete and is propagated to the higher-level application process, the pointers are returned to the top of their stacks.

At any time irrespective of the length of the image there can only be 511 incomplete connected objects. However, as a connected object may be formed from any number of Groups a very complex or continuous image might require a large Group stack. We could overcome this by amalgamating Groups whose vector counters were 0 with Groups that it was linked to. However, Features must persist as long as their connected components are incomplete, an infinite image might have an infinitely long connected component and therefore require an infinite Feature stack.

This is not a problem in the present application where the length of the image is limited by the A4 scanning frame, but if a continuous scan system were being produced, e.g. for conveyer belt inspection, care would have to be taken that persistent visual components of the image such as the edges of the conveyer did not exhaust the Feature stack. This type of problem is difficult in real-time systems because one cannot simply stop the scan even if the occurrence of the problem is extremely rare. It would be easy to construct a system whose guaranteed real-time throughput was limited by the need to purge a stack, even if this takes only 10ms once a week.

Procedures

The outermost level of the run-time feature stream to connected object stream process is shown in the following code:

```
WHILE Go
  SEQ
    input ? InWord
    NoItems := NoItems + 1
    CASE  InWord /\ #70000 -- mask data type bits
      #30000
        Y := Y + 1      -- EndOfLine, increment global Y position
      #40000
        ...   do info   -- Feature fold; put features into
                        -- connectivity structure
      #60000
        ...   do join   -- JoinVal fold; associate label values
      #10000
        ...   do end    -- EndVal fold; dissociate label value,
                        -- check for complete object and process
      #70000
        Go := FALSE     -- EndOfFrame, return control to O.S.
```

The '...' symbol signifies an OCCAM 'fold' which simply obscures the program text which lies in the fold. The OCCAM compiler simply 'unfolds' all the program text to discover the lexical order. Folds are used rather than procedure statements for the efficiency reasons given in the software optimisation section. The process simply loops round reading input from the GRIPPR output buffer and switching on the type of data item, i.e. Feature, EndVal, JoinVal, EndOfLine or EndOfFrame. When an EndofFrame item is found the process terminates and returns to the transputer operating system. The EndOfLine type is required so that feature data items may be assigned a 'Y' co-ordinate without having to explicitly transmit this from the GRIPPR hardware with every Feature.

The Feature fold

The purpose of the Feature fold is to assign features to the connectivity data structure. The Feature fold unpacks the 'InWord' 32 bit data item to reveal the x co-ordinate of the feature, its component label value and its type.

The x co-ordinate of the feature is skewed as the horizontal position of an image is delayed/shifted right by two pixels per neighbourhood operator stage, relative to the line-sync. As there is a depth of two operator stages in the GRIPPR pipeline the image is shifted right by 4 pixels. Pixels with a horizontal position of less than 4 are therefore from the preceding line of the image.

Having de-skewed the co-ordinates of the feature, the feature is assigned to the Group that is pointed to by the label value vector corresponding to the feature's component label value. The type of the feature is represented as an 8 bit value, with one bit set for each of the possible GRIPPR feature types o1..o8 If more than one feature type bit is set, as could happen for a diagonal component which has a RSkel top end-node and a BSkel right end-node at the same location, a separate Feature data item is generated for each instance of the feature type. If the label value vector is null i.e. the component label value is not associated with any Group, a new Group is fetched from the Group stack, the label value vector is updated and the Group vector counter is initialised to 1.

The JoinVal fold

The purpose of the JoinVal fold is to associate component label values so that features

that have or will be labelled by those values will be associated with the same connected object. After unpacking the label values one of four situations will occur.

- Both label values are null: A new Group is fetched, the corresponding label value vectors are updated and the Group vector counter is initialised to two.

- One label value is null, the other non-null: The null label value vector is assigned the same Group as the non-null label value vector, and the Group vector counter is incremented.

- Both label value vectors point to the same group: no action is taken.

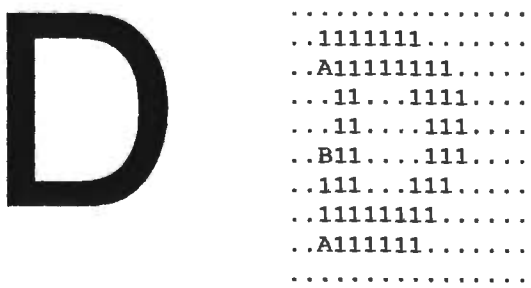
- The label value vectors point to different groups: The two Group lists are joined if the Groups have different 'Top Group' pointers i.e. they are not already joined. The 'Top Group' pointers of all the Groups in one of the Group lists are updated. This operation is potentially expensive, but in practise is relatively rare and Group lists are normally short.

The EndVal fold

The purpose of the EndVal fold is to dissociate label values from Groups and to check for the completion of connected objects. If a complete connected object is found then all the processing up to and including the transmission of the object to the application specific process is performed in this fold.

Normally an EndVal just decrements the vector counter in the Group pointed to by the label value which is terminated. If the vector counter becomes 0 then the other Groups which may be associated with that Group are checked, if all Group vector counters are 0 then the features associated with the Group list constitute a connected object. When a connected object is located the features are copied into 8 arrays one for each feature type and the Feature and Group data elements which were used to describe the object are returned to their respective stacks.

Before the co-ordinates of the features are normalised, pairs of features corresponding to protrusions of 1 pixel depth are removed. Features of this kind are often the consequence of aliasing on straight lines in an image.



The image on the left is scanned and the feature types corresponding to the foreground BSkel left end-nodes, shown as 'A', and the background BSkel right end-node shown as 'B' are generated. As the vertical bar in the image falls on an image cell boundary

the noise inherent in any imaging system causes some small features to appear. The information content of the top AB pair is very low as the the protrusions are only 1 pixel deep and are therefore known to be the likely product of aliasing, rather than a determinant of the connected object pattern class. Removing all such pairs from the feature list is quite fast once the features are separated into different types as each type of feature can only form a pair with one other type of feature.

This sort of image smoothing to remove shallow edge features can be achieved in programmable pipeline or cellular array processors by executing some erosion and dilation operations. However, there are two respects in which a feature level approach may be superior. First, one can question the efficiency of performing many millions of pixel operations per second when the same effect can be achieved at the feature level in only a few thousand cycles per second. Second, the parallel erosion/dilation approach cannot effectively be under the control of the recognition process, e.g. we cannot easily perform different erosion/dilation steps on different components of the same image on an image parallel processor.

Looking ahead to the OCR application implementation, this sort of variable smoothing has been adopted. A two stage recogniser is used, the first stage looks at the features as they are generated by GRIPPR, if this recognition fails the feature pattern is passed to the second recognition stage. The second stage applies the same small protrusion elimination algorithm to the 16 x 16 grid normalised feature co-ordinates. The effect of this is to remove features generated by protrusions that are small relative to the size of the character. Thus the GRIPPR approach allows the significance of features to be varied dynamically for each individual component of the image. The GRIPPR software is designed to demonstrate a high throughput, a slower more sophisticated system could weight the significance of features by the relative depth of the protrusions which generated them.

The bounding box of the connected object is found very efficiently as the feature type and co-ordinate of the extreme top, bottom, left and right edges are already known, e.g. the top of the bounding box is given by the y co-ordinate of the top end node from the foreground RSkel operation. It is often very useful to have the exact bounding box of a connected component as this helps when formatting connected objects into composite characters, words and lines of text. One of the draw backs of the standard symmetrical erosion skeletonization is that it does not give the exact bounding box of the connected component. Once the bounding box has been found, the feature co-ordinates are normalised onto a 16 x 16 grid and the connected object is transmitted down a transputer link to the application specific process.

The Output Buffer process

For an OCR application some of the processing must be related to the formatting of lines of text as opposed to just recognising individual characters. This creates an uneven rate of consumption of connected objects, therefore just as it was necessary to have an input FIFO buffer an output FIFO buffer is also required. (this is assuming the GRIPPR software and the OCR application are not implemented on the same physical processor, in which case efficient utilisation is provided by the concurrence between the different processes.) This output buffer is given a higher priority than the OCR process using the OCCAM PRI PAR statement which ensures that the GRIPPR software process is never waiting to output connected objects while the

OCR process is formatting lines of text. Normally an OCCAM FIFO buffer process is implemented as two concurrent processes. This was found to be unnecessarily inefficient and instead the OCR process is made to explicitly request data from a single buffer process. Starvation of the OCR process is prevented by giving its request a higher priority than the incoming GRIPPR connected components using the OCCAM PRI ALT statement. Blocking of the GRIPPR software is prevented by transmitting the whole buffer at one time to the OCR process and then guarding its data request by a buffer empty flag.

6.7 Performance evaluation

The loading of the different parts of the GRIPPR software process for the test pages Test1 and Test4 from the appendix is given below.

Test1 contains 1,280 12 point characters in 5 founts, made up by 1360 connected objects. Test4 contains 35,000 simply connected dots.

The loading on each part of the software process is given as a percentage of the total processor time used. As the scanner is relatively slow, the software process is often idle waiting for data items from the hardware process.

The loading of the EndVal fold is split between four separate tasks; maintaining the data structure, clustering connected objects once complete, smoothing small protrusions from objects, and normalising and transmitting the feature data.

Process	Test1	Test4
Feature Fold	33.1%	33.2%
JoinVal Fold	3.3%	0.01%
EndVal	4.8%	5.1%
Clustering	26.2%	30.0%
Smoothing	8.5%	1.8%
Scale/transmit	24.0%	30.0%
Total	100%	100%

The following table gives the total time taken by the software process for each page, the time taken to scan the page and the processor utilisation. The final statistic gives the number of objects of the type found in the test image which can be processed per second of processor time, i.e. the maximum attainable throughput of the GRIPPR software.

	Test1	Test4
Processor Time	829ms	7679ms
Time Available	9896ms	9896ms
Processor Utilisation	8.3%	77.6%
Objects Processed	1,360	35,000
Objects/sec processor time	1,641	4,558

The ability to process, scale and transmit over 1,600 reasonably complex patterns or 4,500 simple patterns per second demonstrates the speed attainable when all the computationally intensive pixel processing tasks are performed in hardware. For

some simple applications where all that is required is to count and size objects a throughput of over 5000 objects per second should be attainable on one T800 20Mhz transputer. This throughput is irrespective of the resolution or speed of the image, provided the video bandwidth of the GRIPPR hardware is matched to the imaging system.

Chapter 7

Pattern Recognition and OCR

The application specific pattern classification and higher level processes needed to demonstrate GRIPPR are presented in this chapter.

7.1 Pattern classification and contextual processing for OCR.

The process of converting images into symbolic representation requires not just the classification of patterns in the image but also contextual processing. Generally pattern classification can be regarded as the mapping of transformed sensory data onto a learned state. Contextual processing is the application of non-pattern-local information to constrain the pattern classification output or process.

The distinction between pattern classification and contextual processing is important in the design of a pattern recognition application, because the contextual information available in a particular application will determine what information should be propagated by the pattern classification process. Generally, if there is a high level of redundancy in the text to be recognised, as in English words, then we would much prefer to propagate more than one possible candidate for a character than make a false substitution as contextual processing can arbitrate between different plausible candidates.

Ideally, the pattern classification process would yield a list giving the probability attached to each possible symbol mapping of an input pattern, we could call such a list an 'hypothesis list'. In practice, the probability data might not be needed for contextual processing. We might therefore apply a threshold to the hypothesis list giving a binary list representing all the pattern classes above a certain likelihood for a particular input pattern. This threshold value could be varied dynamically, e.g. dependent on the quality of the image. However, if the pattern classification process is fairly robust we might wish to accept all possible pattern class mappings consistent with the input pattern, i.e. each pattern class not set in the hypothesis list must be actively eliminated as a plausible candidate.

If the pattern classification process is feature driven then we can extend the hypothesis list idea to features. The pattern classification process could generate an hypothesis list of all the pattern classes consistent with each feature in the input pattern. The plausible hypothesis list for the pattern as a whole is simply the dot product of the feature hypothesis lists.

A recogniser based on this plausible hypothesis principle should have a low false substitution rate as it is not concerned with some relative notion of greatest likelihood. Such an approach might therefore be appropriate in applications where contextual redundancy is high such as in Document Image Processing (DIP) document retrieval systems. In DIP systems all paper documents are scanned and tagged for electronic storage as images, to provide intelligent access to such images the documents can also be OCR processed ar scan-time. When a query is generated e.g. find all documents referring to 'Maverick Industries', the contextual information embodied by the query can be used to determine which documents contain plausible candidates for the text string. The nature of this recognition task is that ambiguities must be propagated by the recognition process as the contextual information required for arbitration between plausible pattern class candidates is only available when the query is generated.

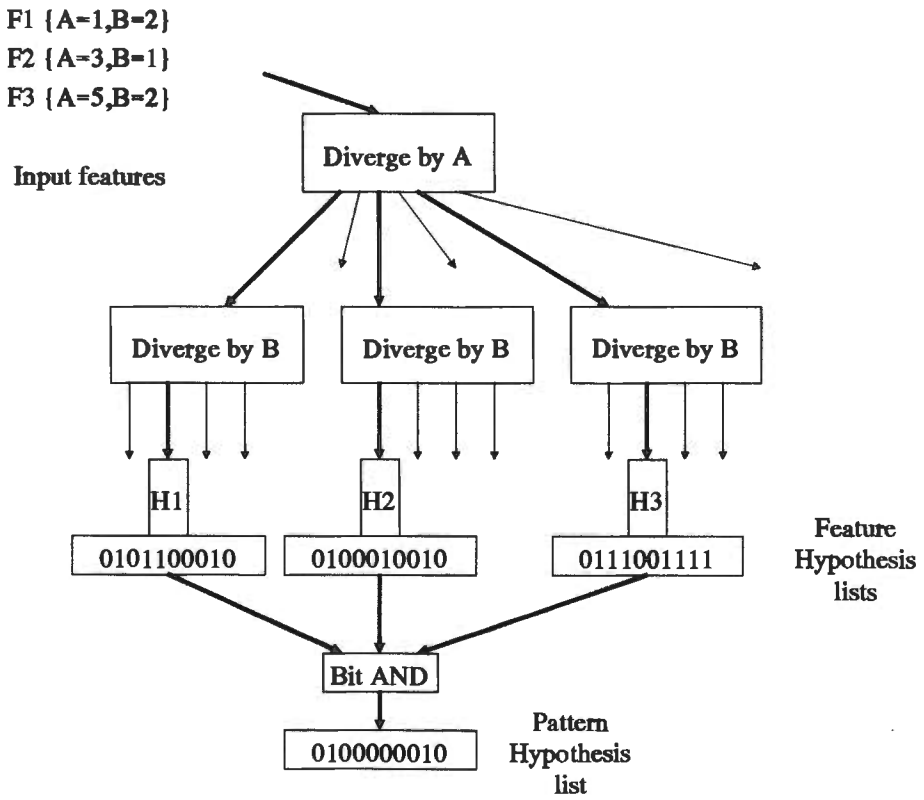


Fig.7.1: Hypothesis list model of pattern recognition

7.2 A simple pattern recognition method

Underlying the above discussion is a very simple model of how pattern recognition may be performed. The good thing about the model is that it allows the easy exploitation of the inherent 32 bit parallelism of 32 bit processors.

The basis of the model is that each feature of the object to be recognised generates a hypothesis list of which pattern classes it might belong to. The hypothesis lists are represented as n bit words, with a bit set representing each class that is consistent with the feature that points to the hypothesis list. The list of consistent hypotheses for all features from an object is generated by logic ANDing the hypothesis list for each feature in the object.

A recogniser based on the hypothesis list model is shown in Fig 7.1. The figure shows the recognition of a pattern made up of three features. The parameters of the features are used to map each feature onto a hypothesis list representing the plausible candidates for that feature. The plausible candidates for the pattern is the logic AND of the feature hypothesis lists.

There are two basic parameters to the model, feature divergence and class depth:

Feature Divergence: This controls the mapping of a feature onto an hypothesis list. Feature data and/or object data can be used for divergence.

For example the most obvious divergence is by relative scaled x and y co-ordinates, i.e. features are always mapped onto a grid, each location of which corresponds to a hypothesis list. Divergence can be increased by, for example, increasing the grid resolution, diverging between different node types, diverging between different complexities of object or diverging by the size of objects.

There is very little time penalty to divergence as it is quick relative to the n bit ANDs (for n >= 128). The constraint on divergence is the memory capacity of the system. For instance, placing features in raster order and then diverging on the resulting sequence would require about T^{mF} hypothesis lists, where T is the number of types of feature and mF the maximum number of features per pattern. In the present application there are 8 node types and the maximum number of features per object is about 30 which would give an hypothesis list space of 8^{30} lists, which is impractical. Therefore, to optimise the memory utilisation of the system care has to be taken to diverge on that data which discriminates best between different classes.

Class Depth: The other parameter of the model is the number of classes. For instance in OCR we might have a different class for each character in each fount. This serves two purposes. First it gives more information for higher level contextual processing e.g. we may assume that within a word only one fount is used allowing us to remove inter-fount ambiguities. Second the pattern recognition performance is improved because ambiguities caused by allowing characters to be made up of features from different founts does not occur. The effect of increasing class depth is linear in processor time and memory usage for a given level of divergence.

To demonstrate the high throughput of GRIPPR, which is the objective of the OCR system, it was decided to use the minimum class depth of one class per ASCII character. i.e. 128 bit hypothesis lists.

The learning process within this model is extremely simple. Simply start with a blank hypothesis list space. Then using the same feature divergence that will later be used for recognition find the hypothesis list for that feature. Then add the symbol corresponding to the input training pattern to the hypothesis list. This process is repeated for each feature in the training pattern and for all training patterns.

'Fuzzy' recognition in the context of this model is required if we wish to assign a pattern to a class if that pattern has any feature which varies from those presented at the learning stage. i.e. there is no way of simply finding the 'nearest' alternative. Fuzziness can be implemented in several ways within the context of the model. The word 'fuzzy' is used here to imply the classification of patterns which do not correspond exactly to the training set and is not related to Fuzzy Set theory which implies computation using non-binary arrays.

At recognition time several similar features can be generated from a feature by dithering some parameter/s of the feature. All the hypotheses lists generated by these dithered features can then be ORed together to generate a 'fuzzy' list. The 'fuzzy' lists from all the features can then be ANDed together to give the final fuzzy hypothesis list. The main drawback with this approach is that it is slow at recognition time, though if it was only applied infrequently i.e. when non-fuzzy recognition has failed then this might not be a problem.

Fuzziness can be introduced in a similar way at learn time, by generating dithered patterns from each pattern example of a class. This has no effect on run-time speed. The only drawback with this approach is that one has to decide the degree of fuzziness at the learning time whereas one might wish to vary this dynamically or incrementally. The problem can be alleviated by having different recognisers corresponding to different levels of fuzziness, though this requires more memory.

If the recogniser is required to recognise patterns that may have drastically incorrect or missing features, such as a piece of dirt on a character. This can be achieved by ORing some or all of the feature generated hypothesis lists together. For instance one could OR pairs of hypothesis lists together so that no one feature could eliminate any class from the final list. This would produce a very ambiguous output, with lots of potential classes for the input pattern. However, in the context of generating plausible candidates for higher level contextual processing after initial recognition has failed, it could be useful.

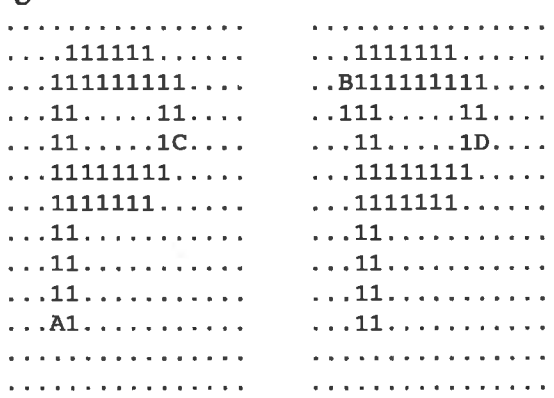
7.3 Character recognition implementation

Various different character recognition strategies were tested. The main requirement of these strategies was that they should be quick and simple to implement and that they should demonstrate the high throughput of the GRIPPR process. The specification of the OCR process throughput was that it should be able to recognise 1000 characters/second on one T800 20Mhz processor and 2 Mbyte RAM. The OCR process has to include some contextual processing and lexical formatting as well as the basic pattern classification task of character recognition.

A structural approach

The first approach to be tested within the hypotheses list model presented earlier was basically structural, i.e. it sought to map characters with similar structures together for differentiation by feature detail. The motivation for adopting this approach was that the x and y co-ordinates of the features generated by GRIPPR are not equally consistent. The 'depth' of the protrusion represented by the feature is always

extremely accurate and invariant. However, the other 'perpendicular' co-ordinate of the feature is more variable, particularly for straight edges where its value becomes ambiguous.

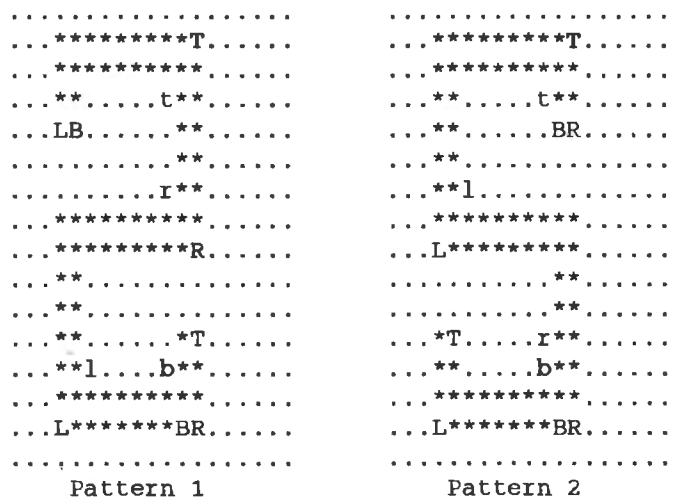


For example, in the above picture the left end-node of the BSkel foreground operation generates the features A and B from two very similar objects. The relative depth of the left hand foreground protrusion into the background is the same for both patterns giving a normalised x co-ordinate of 0 for both features. However, the 'perpendicular' y co-ordinate of the left protrusion is ambiguous as the protrusion is effectively spread from top to bottom. This results in the feature A having a y co-ordinate of 15 whereas feature B has a y co-ordinate of 2. However, the perpendicular co-ordinates of the features do often give useful, consistent, information as is the case for the right end-node foreground BSkel features C and D above. The y co-ordinates of features C and D are consistent as the right protrusion of the letter P is vertically defined.

To alleviate this perceived problem an attempt was made to separate objects with different structures by ordering the features in some consistent way and then using only the more consistent depth co-ordinate of each feature to differentiate between pattern classes that share the same ordering /structure.

In the context of the simple hypothesis list recognition model introduced earlier, the divergence for each feature is first by the type of structure of the object and then by the feature positions within that structure and lastly by the protrusion depth of the feature. However, given that one co-ordinate of a feature may be ambiguous it is difficult to generate a consistent ordering of features to determine the structure. One way round the problem is to generate two ordered sequences of features, one for the x-consistent BSkel features and one for the y-consistent RSkel features.

This system gave a very high hit-rate as both the ordering and the feature data used are highly consistent. However, by effectively throwing so much of the data away some strange ambiguities are caused. As in the following case showing the ordered sequences generated by the characters '2' and '5'.



	Pattern 1	Pattern 2
BSkel features	L0,L0,l2,r8,R10,R10	L0,L0,l2,r8,R10,R10
RSkel features	T0,t2,B3,b14,B15	T0,t2,B3,b14,B15

The ordered lists of features for the above characters are exactly the same. The inability to fundamentally distinguish between a '2' and a '5' is a severe limitation, as the contextual information within numerical strings e.g. telephone numbers, is very low. An attempt was therefore made to incorporate additional information into this character recognition approach, e.g. the perpendicular co-ordinates of the extreme features were used to resolve the '2' versus '5' ambiguity. However, the implementation was starting to become inelegant with no clear method, simply a collection of solutions bundled together by necessity. Also, given that the underlying recognition model is so computationally efficient it seemed a shame to implement a strategy which is dependent on the sorting of features within an object, a task which takes longer than the basic feature divergence and hypothesis list logic AND process for an average character.

A connectivity syntactic approach

In response to these perceptions a connectivity syntactic approach was tested. This approach relies on parsing the connectivity of an object, differentiating between different pattern classes on the basis of the feature types and parameters at each node of the connection tree. The GRIPPR process of connected object extraction effectively parses the connection tree of the input pattern. Thus the order of the features generated by this process should be invariant for the same topology of connected object. Preliminary simulation of the ordering produced by the connected object extraction process indicated that there was a significant amount of topological consistency in the ordering of features.

The character recognition implementation now became very efficient, simply diverging on the feature list ordering, which is already given, and then on the feature type and depth co-ordinate. This solves the ambiguity for the '2' versus '5' problem above as the two patterns generate a different ordering of features, e.g. the 'T' feature type generated on the fifth line from the bottom in both patterns will be near the end of the feature list for the '2', but at the beginning of the feature list for the '5'.

Though this method did not have the same problems as the structural approach adopted previously, the general performance was disappointing. The ambiguity between topologically distinct classes was still quite high, this was mainly due to the

component labelling optimisations performed in the GRIPPR hardware. The blocking operation in particular has the effect of reducing the topological complexity, and therefore the topological information content of the connectivity process. The topological nature of the GRIPPR features would have allowed the object to be reconstructed at a higher level and this could have produced a more topologically discerning ordering for syntactic recognition. However, within the context of the 1000 character/second specification the reconstruction process would have consumed too much CPU time. Another perhaps more important problem found with this syntactic approach is that it suffers from a relatively high false substitution rate. This is caused by small amounts of input pattern variance leading to a different ordering of the same features. Once the order is corrupted all the feature data will be interpreted incorrectly, also any attempt to locate the 'nearest' pattern class by the 'fuzzy' strategies mentioned earlier often leads to the false classification of the pattern. This is a general criticism of the syntactic recognition approach; because the variance of features cannot be treated independently, there is no simple way of uniformly relaxing the specification so as to locate 'nearest' alternatives.

A simple geometric approach

Consideration of how the independence of features could be maximised led to the development of a simple geometric approach to the interpretation of features.

The simplest geometric approach would be to locate each feature onto a 2-Dimensional grid of hypothesis lists. Noise in one feature would then in no way effect the interpretation of any other feature. We could also first diverge on the type of each feature so that only features of the same type were located onto the same grid. This level of divergence would still generate a significant level of unnecessary ambiguity as a pattern with very few features could map onto a pattern class which was always defined by a greater number features. To avoid this the recogniser diverges on the number of features of each type. This last divergence means that feature interpretation is no longer completely independent as the presence or absence of a feature within an object will alter the interpretation of features of the same type. However, the number of protrusions of a given type within a pattern is highly invariant.

The size of the hypothesis list space is the product of the different types of divergence:

Type of feature = 0..7

Number of Features = 0..15

X co-ordinate of Feature = 0..15

Y co-ordinate of Feature = 0..15

If there are no features of a certain type then this information is treated as a special type of feature, mapping onto a different hypothesis list for each type that was absent. The above divergence describes a recogniser with 8x16x16x16 = 32K hypothesis lists, and even this small size is unnecessarily large for alpha-numeric characters as there can be up to 15 protrusions of each type. The reason that such complex patterns were considered was to allow the recogniser to be tested with chinese characters, though this has yet to be attempted.

If noise does vary the number of features of a certain type in the pattern then this is unlikely to lead to the false classification of the pattern as the interpretation of features of other types is still valid. Within this character recognition approach it is possible to deal with the ambiguous 'perpendicular' co-ordinates directly because unexpected variance in one feature will rarely lead to false classification of the object. Therefore, initially we can simply ignore the problem and hope that we have sufficient variance and noise in our training set to enable most patterns to be classified correctly. We can only afford this cavalier attitude because we know that unexpected variance in the perpendicular location of any one feature is likely to lead to the initial non-classification of the pattern rather than false classification. We can contrast this with methods based on pattern global structure or syntax, here unexpected variance in the co-ordinates of one feature can lead to the structure or syntax being corrupted which in turn results in the false interpretation of all features and thus to a more likely false classification.

'Fuzzy' recognition

If the initial pattern classification which is only based on the variance of the training set fails, then we can perform a second stage of 'fuzzy' recognition. The greater variance of the 'perpendicular' co-ordinates of the features can be acknowledged by reducing the discriminatory power of those co-ordinates. The increased ambiguity of the second stage recogniser is acceptable as the proportion of patterns failing the initial classification is relatively low. A more detailed description of the second stage recogniser is given in the following description of the learning process.

The learning process

Thought has to go into the practicalities of the learning process. Generally it is better to expend resources at the learning stage than during the run-time operation of the recogniser. However, one must also match the learning process to the computational and/or human operator resources available. Fortunately, the hypothesis list method used for pattern classification is very efficient to implement at the learning stage. Each feature generated by a pattern in the training set simply goes through the same divergence as during the recognition process, inserting its pattern class in the resulting hypothesis list.

To generate an image with as much variance as possible the training page shown in the Appendix was designed in Postscript. The training page gives a good spread of text sizes in five different founts; Helvetica, Times-Roman, Courier, Palatino and NewCenturySchoolbook. These founts were selected because they are all very common and yet represent quite different type faces, e.g. Helvetica is a sans-serif fount and Courier is fixed width. To allow for the recognition of a very dense page to demonstrate the throughput of the system 6 point Helvetica numerals were also included. The characters are repeated several times in order to capture the aliasing noise always present when a binary image is formed. Composite characters, i.e. ';:!?ij' are separated into their constituent connected components by means of a size threshold and these are then labelled as separate pattern classes.

In order to capture the skew variance of the characters some means had to be found of tilting the image. Using PostScript commands to angle characters by a few degrees on a standard 300 d.p.i. laser printer was found to be useless as when the angle is

resolved at 300 d.p.i. the shape of small characters remains unchanged. A high resolution 1200 d.p.i. photo typesetter was tested, this skewed characters to a sufficient resolution, but also generated stroke widths too fine to resolve by 300 d.p.i. scanner. The solution was to print different versions of the test image with a slanted line printed along the top of the page, after the sheets are printed they can then be guillotined along this line. The sheets are then fed into the roller type scanner flush to the top of the sheet thereby generating angled images. This method can also be used to place a horizontal crop line at the top the page, guillotining to this line ensures that the image is exactly vertical, i.e. it compensates for the +/-1 degree skew often found in printers. Four pages are generated for each pattern class, one centred, one left skewed, one right skewed and one test page without a crop mark. The purpose of the test page is to generate statistics about the performance of the recogniser automatically as part of the learning process.

It proved important during the development of the recognition algorithm to be able to trace back from the learning process to a particular member of the training set, to a physical printed character on a page. Otherwise it would have been difficult to understand the exact nature of the ambiguities and false assignments thrown up during the learn/test process. Backtracking to a printed character was made possible by storing the compressed feature data from the training set on disk and by adopting a suitable naming convention. When a page is scanned it is tagged by its pattern class identifier (normally the ASCII code number) and whether the page is centre, right, left or test. This tag is then used to name a compressed list of connected object feature lists corresponding to the characters in the image. Each connected object is also tagged by its physical location on the page. The final implementation has been trained to recognise 74 different pattern classes, which requires 4x74=296 pages to be processed. The pages with numeric characters contain 480 character instances the other pages contain 360 character instances, giving a total of 111,360 characters. Of these characters, 83,520 comprise the training set and 27,840 comprise the test set.

Printing the training sheets on an apple LaserWriter II took approximately 7 hours. The page scanning process lasted about 5 hours and the final learning program takes about 1 hour to execute. The learning program simply goes through the files in the training set inserting the pattern class given by the file name into the stage-1 and stage-2 hypothesis list corresponding to each feature for each connected object. When this task is complete the resulting recognisers are tested by attempting to recognise the patterns in the test set. Statistics are then generated for the hit rate, the ambiguity rate and the false substitution rate for the stage-1 and stage-2 recognisers separately and as well as their combined performance. The hit rate is the proportion of patterns in the test set which are recognised correctly, though not necessarily uniquely, i.e. it includes ambiguous responses as long as the correct classification is one of the plausible candidates. The ambiguity rate is the proportion of candidates generating more than one plausible candidate. The false substitution rate is the proportion of patterns that produce a classification, but where the correct classification is not a plausible candidate. These results are also decomposed by pattern class so that the performance of the recognition process for individual pattern classes can be assessed. The overall results are given in section 7.4.

The stage-2 recogniser

Three different ways of increasing the second stage recognisers immunity to noise

and variance were adopted:

- Elimination of relatively small protrusions: The simple and computationally efficient algorithm introduce in the GRIPPR implementation chapter for the elimination of aliasing feature noise is adopted for the post scaled features. This has the effect of eliminating features of small relative depth which tend to be more prone to, or more likely the product of, noise than features of greater depth. It would not be valid to perform this filtering and then simply re-apply the pattern to the same recogniser as this might lead to false classification. The algorithm only works because it is applied consistently to the stage-2 recogniser, first to the training set in the learning program, and then to the input pattern in the run-time program.

- Real co-ordinate dithering: The pre-scaled co-ordinates of each feature in the training set are dithered by +/- 1 pixel position, i.e a separate instance of each feature is formed which has a real co-ordinate dithered around its original value, which may or may not correspond to a different grid position once the feature position is scaled. This has the effect of capturing any possible aliasing dependent positional variance not captured directly by the training set.

- Scaled 'perpendicular' co-ordinate dithering: The scaled perpendicular co-ordinates of each feature are dithered +/- 2 grid positions. This has the effect of reducing the discriminatory power of the possibly ambiguous perpendicular co-ordinates of a feature, as mentioned earlier.

Recognition database compression

Instead of storing a full 128 bit hypothesis list for every location in the feature divergence space, only a 32 bit pointer to a hypothesis list table is used. Adopting this form of compression reduced the memory requirement of the hypothesis list storage by a factor of 0.29 for the final OCR system. Each recogniser requires 32K of hypothesis list pointers and 19440 distinct hypothesis lists were used requiring about about 76K 32 bit words. Thus the total recognition database memory requirement is ((32K x 2) + 76K) x 4) = 560K bytes of memory.

Recognition implementation

The run-time recognition process loads a copy of the stage-1 and stage-2 feature divergence pointers and the hypothesis list table. Then the recognition process accepts connected objects from the scanner via the GRIPPR process. The stage-1 recogniser is implemented by calling the following OCCAM2 code for each connected object.

```
HitList := IdListAll   -- set the working hypothesis to include all possible classes
  --{{{  RecogFine
  lastpos := 0
  SEQ i=0 FOR 8     -- for each feature type
    SEQ
      NumOfT := NumType[i]
      IF
        NumOfT  0     -- if number of type i > 0
          SEQ
            SEQ i2=lastpos FOR NumOfT     -- for each feature of type i
              SEQ
                HPtr := FineVec[(((((i<4)+NumOfT)<4)+     -- divergence
                                  List.Ab[i2])<4)+List.Ot[i2] ]
                HitList[0] := HitList[0] /\ HListStack[HPtr]     -- Long AND
                HitList[1] := HitList[1] /\ HListStack[HPtr+1]
                HitList[2] := HitList[2] /\ HListStack[HPtr+2]
                HitList[3] := HitList[3] /\ HListStack[HPtr+3]
            lastpos := lastpos + NumOfT
        TRUE     -- the number of type i = 0
          SEQ
            HPtr := FineVec[i]     -- No features of this type
            HitList[0] := HitList[0] /\ HListStack[HPtr]
            HitList[1] := HitList[1] /\ HListStack[HPtr+1]
            HitList[2] := HitList[2] /\ HListStack[HPtr+2]
            HitList[3] := HitList[3] /\ HListStack[HPtr+3]
  --}}}
```

The above code is extremely efficient as the working hypothesis list can be stored in on-chip memory using the 'PLACE IN WORKSPACE' OCCAM statement. Only additions and shifts are used to index the feature divergence space pointer 'FineVec'. The code for the stage-2 recogniser is exactly the same except that the feature list is filtered to remove features of small depth and a different feature divergence space pointer 'CoarseVec' is used.

Recognition proceeds by first attempting to classify the object with the stage-1 recogniser. If the stage-1 recogniser fails to classify the object then the object is passed to the more accepting stage-2 recogniser.

7.4 Character classification results

The results for the character classification process are generated by testing 27840 separately scanned characters against the stage-1 and stage-2 recognisers and then against the combined recogniser formed by only using the stage-2 recogniser for those patterns that are not classified by the stage-1 recogniser.

	Stage-1	Stage-2	Combined
Hit rate	97.83%	99.93%	99.8%
Ambiguity rate	21%	39.4%	21.6%
False-substitution rate	0.16%	0.01%	0.17%

The stage-1 recogniser exhibits a relatively low ambiguity and hit rate. This is due to the fact that input patterns have to correspond exactly to previously learned features in order to be classified. The stage-2 recogniser exhibits a much higher ambiguity and hit rate as it is more accepting of features which may not correspond exactly to features found in the training set.

By only using the stage-2 recogniser when the stage-1 recognition fails we are able have the relatively low ambiguity of the stage-1 recogniser in most cases, whilst producing a hit-rate close to that of the stage-2 recogniser. The final combined ambiguity rate of 21.6% seems quite high, but this is for a system without any

contextual processing. Therefore, a great deal of inter-case ambiguity, e.g. 'sS,cC,oO,pP,xX,vV,uU' etc. is present. Also, the small patterns representing full stops and the top of i,j are highly ambiguous, but this ambiguity disappears when composite characters are recognised at a higher level.

The false substitution rate of 0.17% for the combined recogniser is due mainly to the stage-1 recogniser. False assignments happen when the variance of the learning set does not fully capture the variance of the test set. The main source of variance in the test set vis a vis the learning set appears to be the fact that the skew variance captured by the learning set is too coarse. Rather than throw away the skew variance present in the test data this data is also entered into the learning process after the above statistics have been generated. The effect of increasing the learning base by a third is to increase the combined ambiguity rate from 21.6% to 21.9%, this extra 0.3% represents those patterns which were most likely to generate false assignments.

7.5 OCR processing

Character recognition is only one part of the OCR process. A minimum requirement for any OCR system is also to maintain the lexical order of the text and to separate words by spaces. The stages in the OCR processing pipeline are shown in Fig 7.2. The program is actually defined in terms of recursive procedures, but these could easily be re-written as separate pipeline con-current process and then placed on separate processors for parallel execution.

The line store window

To overcome the problem of separating a non-lexical stream of characters into lines of text requires that when a character has been recognised the resulting hypothesis list and the positional parameters of the character are placed in a temporary text-line store. This store acts like a sliding window moving down the page, the height of the window is such that it should capture any skewed lines completely within the window. The bottom of the window grows downwards as new characters arrive, when the window reaches the required size the line of text associated with the top character in the window is processed. The top of the window then shrinks to the highest remaining character in the window, like a caterpillar moving down the page. The characters to be associated with a line of text are found by searching for left and right neighbours of the top character. As the search progresses along a line of text the height expectation is adjusted to allow for skewed lines.

Composite character processing

Once the text-line has been found processing proceeds to the recognition of composite characters, i.e. characters such as ';:!?ij' which are made up of more than one connected component. The method adopted is to recognise the components of the character as separate symbols, mapping onto the ASCII codes 1..10 which do not correspond to printable characters anyway. When a hypothesis list is found whose first 32 bit word is non-zero the positional parameters of its neighbours are checked to determine if they overlap in the vertical dimension. If a pair is found which does overlap they are processed by some hand-crafted code to determine which if any of

the composite characters they represent. The redundancy in composite characters is so high that this process is highly reliable. A more sophisticated feature combining approach would have to be adopted to recognise complex composite patterns such as chinese characters or whole alphanumeric words.

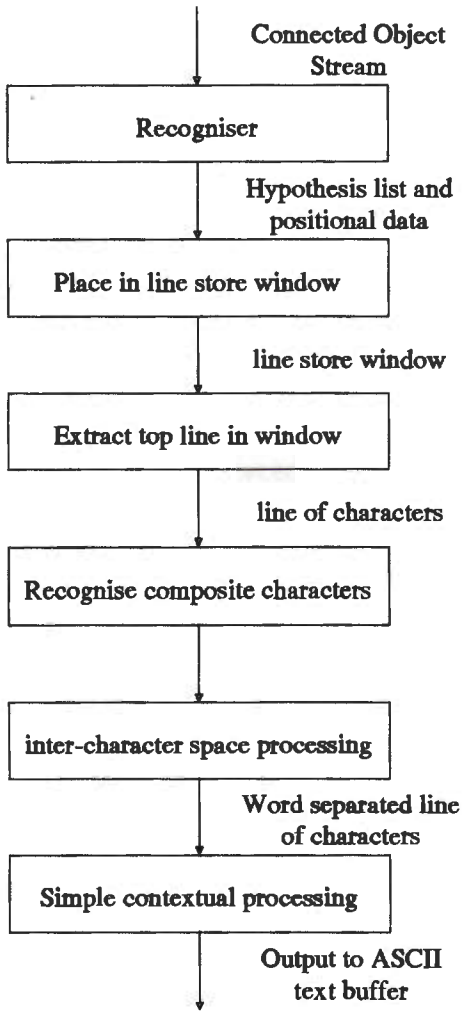


Fig.7.2: Stages in the OCR processing pipeline

Space processing

A lot of effort has gone into the seemingly trivial task of separating lines of text into word strings. Various character local algorithms were tested. The best local algorithms depend on the inequality of spaces before and after a character. However, this only works for certain founts and has particular difficulty with single letter word and fixed space numbers in proportional founts. Other local algorithms rely on relating the height and width of characters to its surrounding spaces. However, a non-character local text-line based algorithm was found to be most effective. This algorithm scans along the text-line forming inter-character spaces into a histogram. The histogram is then analysed in relation to the average character height to discover the local minima which separates inter-character from inter-word spaces. This presumes

that there is a unique threshold value applicable to the whole line, which is generally the case. The threshold value is then applied to the text-line to separate words.

Simple contextual processing to resolve inter-case ambiguity

Inter-case ambiguity accounts for most of the ambiguity generated by the recognition process, as without contextual information it is extremely difficult to differentiate between the character pairs 'sS,cC,vV,pP,1l,...'. After the text-line is separated into word strings some very simple rules can be applied to resolve most inter-case ambiguity. These rules rely on the fact that most characters are recognised unambiguously, we should therefore be able to determine the case of a word and thereby resolve any inter-case ambiguity for individual characters within the word. A typical rule is that if a character to the left of a case-ambiguous character is lower-case then the ambiguous character is also forced to become lower-case. The fact that this rule is not always true may lead to false substitution, this obviously represents an application specific trade-off, one might prefer to leave all contextual processing until a specific query is generated as in DIP retrieval. However, for most applications the trade-off is relatively small as most characters are recognised unambiguously and most words follow the case rules consistently. Some words which may be fairly common in some applications such as 'NewCenturySchlbk' are often processed incorrectly giving the output 'Newcenturyschlbk' as the characters 'cC and 'sS' generally cause inter-case ambiguity. Therefore though the method adopted here has the virtue of simplicity in a practical application a method based on relative character sizes would probably perform more consistently.

Final output

After contextual processing the text-line is finally translated into an ASCII representation. If the hypothesis list is empty, the character is assigned a question mark, '?', if more than one bit is set in the hypothesis list then each corresponding character is printed within curly brackets e.g. '{B,8}'. The resulting representation is not stored on disk immediately as the speed of writing to the 7.7Mhz IBM PC AT host system is slow using the standard BIOS subroutines available through the OCCAM host interface. Therefore, the times taken to process the images presented in chapter 8 do not include the writing of the ASCII representations to disk.

No attempt has been made to provide a more sophisticated user interface as the OCR system is only intended as a demonstration of the underlying processing throughput. The idea is that the GRIPPR design is sufficiently general purpose to allow existing application specific know-how to be adopted rather than reconstruct every possible application of the technology from scratch.

The output of the GRIPPR based OCR system for test page 1 given in the Appendix is shown below, (the symbol 'Q' is substituted for '?'):

```
{ILt}he qu ick brown fox j umped ove r the lazy dog 9 0 8 {11} 7 2 6 3 5 4
THE QU{Ir}CK BROWN FOX JUMPED OVER THE LAZY DOG
The qu ick, brown. fox: j umped; {OQo}verQ the - l az y! ( dog) *
the quick brown fox jumped over the {Il}azy dog 9081726354
THE QU{Il}{CG}K BR{Oo}WN F{Oo}{Xx} JUMPED {Oo}VER THE LAZY DOG
The quick, brown. fox: jumped; overQ {lt}he -lazy! (dog? *
the quick brown foxjumped over the lazy dog 9081726354
THE QUICK BR{Oo}WN F{Oo}XJUMPED {Oo}VER THE LAZY DOG
The quick, brown. fox: jumped; overQ the -lazy! (dog) *
the quick brown fox jumped over the lazy dog 9081726354
THE QUICK BROW? F{Oo}{Xx} JUMPED {Oo}VER THE LAZY DOG
The quick, brown. fox: jumped; overQ the -lazy! (dog) *
the quick brown foxjumped over the lazy dog 9081726354
THE QUICK BR{Oo}WN F{Oo}{Xx} JUMPED OVER THE LAZY DOG
The quick, brown. fQx: jumped; overQ the -lazy! (dog) *
t he qu ick brown fOx j umped over {rt}he lazy dog 9 0 8 1 7 2 6 3 5 4
THE QUICK BROWN FOX JUMPED O{UV}ER THE LAZY DOG
The qu ick, brown. fox: jumped; {Oo}verQ the - lazy! (dog) *
the quick brown fox jumped over {lt}he {Il}azy dog 9081726354
THE Q{Uu}{Il}CK BR{Oo}WN F{Oo}{Xx} JUMPED {Oo}VER THE LAZY DOG
The quick, brown. fox: jumped; overQ {lt}he -lazy! (dog) *
the quick brown fox jumped over the lazy dog 9081726354
THE QUICK BROWN FOXJUMPED {Oo}VER THE LAZY DOG
The quick, brown. fox: jumped; overQ the -lazy! (dog) *
the quick brown fo{Xx} jumped over the lazy dog 9081726354
THE QU{1It}CK BROW? F{Oo}{Xx} JUMPED {Oo}VER THE LAZY DOG
The quick, brown. fox: jumped; {Oo}verQ the -lazy! (dog) *
the quick brown fox jumped ov?r the lazy dog 9081726354
THE QUICK BR{Oo}WN FOX JUMPED {Oo}VER THE LAZY DOG
The quick, brown. fox: jumped; overQ the -lazy! (dog) *
```

The output shows a good recognition performance, with particularly few false substitutions. The ambiguities which are present are mostly inter-case ambiguities e.g. {Oo} or {Xx}. The word separation algorithm is not very well honed and sometimes fails to separate words or separates characters which should be part of the same word, particularly the Courier numbers. The statistics describing the recognition performance for the different test pages is given in chapter 8.

Chapter 8

Results and Conclusions

8.1 Final system description

The photograph in Fig 8.1 shows how the OCR system is set up for use. The box under the TAXAN scanner contains the GRIPPR hardware and an RS232 circuit to command the scanner. To scan a document it is placed in the scanner and the green start button is depressed. This causes the scanner to feed the document through at its maximum speed and the image to be processed in real-time by the rest of the system. The three red error lights correspond to different FIFO flags in the circuit; image too wide for the line store, empty non-current component label value store, full output buffer. Any detected error causes all processing to stop and the machine has to be re-initialised (switched on and off).

A photograph of the inside of the GRIPPR hardware box is shown in Fig 8.2. The plastic card is included for size comparison this is also approximately the size a transputer module. The empty sockets along the top and bottom of the PCB were included to allow for the easy modification of the hardware; some of these sockets were populated during the hardware de-bugging process. Fig 8.3 shows the processor implementation of the final OCR system.

8.2 Performance evaluation

Three test pages are used to characterise the GRIPPR-OCR performance. These pages were also scanned into a state of the art dedicated OCR machine the Kurzweil K-5200 from Xerox Imaging Systems for comparison. The test pages are shown in the Appendix.

Test1

Test1 is a page of English words (The quick brown fox...etc.) in the five founts learnt by the GRIPPR-OCR system. The semantic context in the document is sufficient for

the very simple contextual inter-case arbitration processing implemented by the

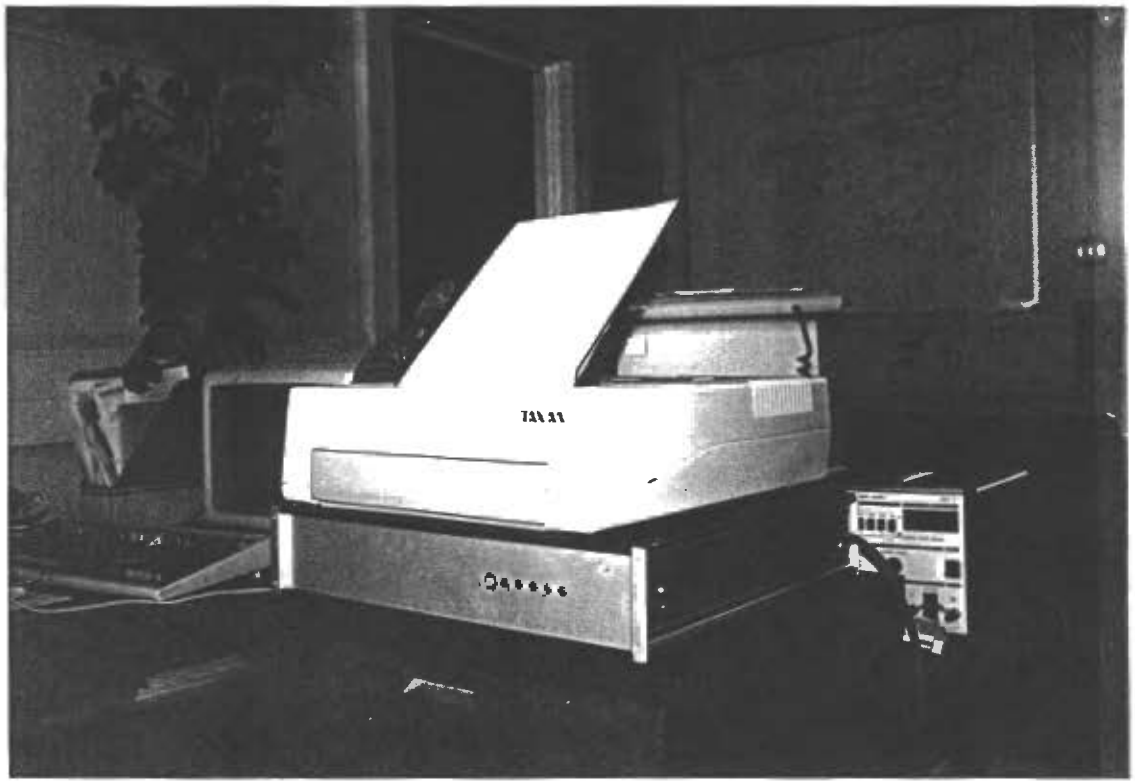


Fig. 8.1: GRIPPR-OCR system, external view

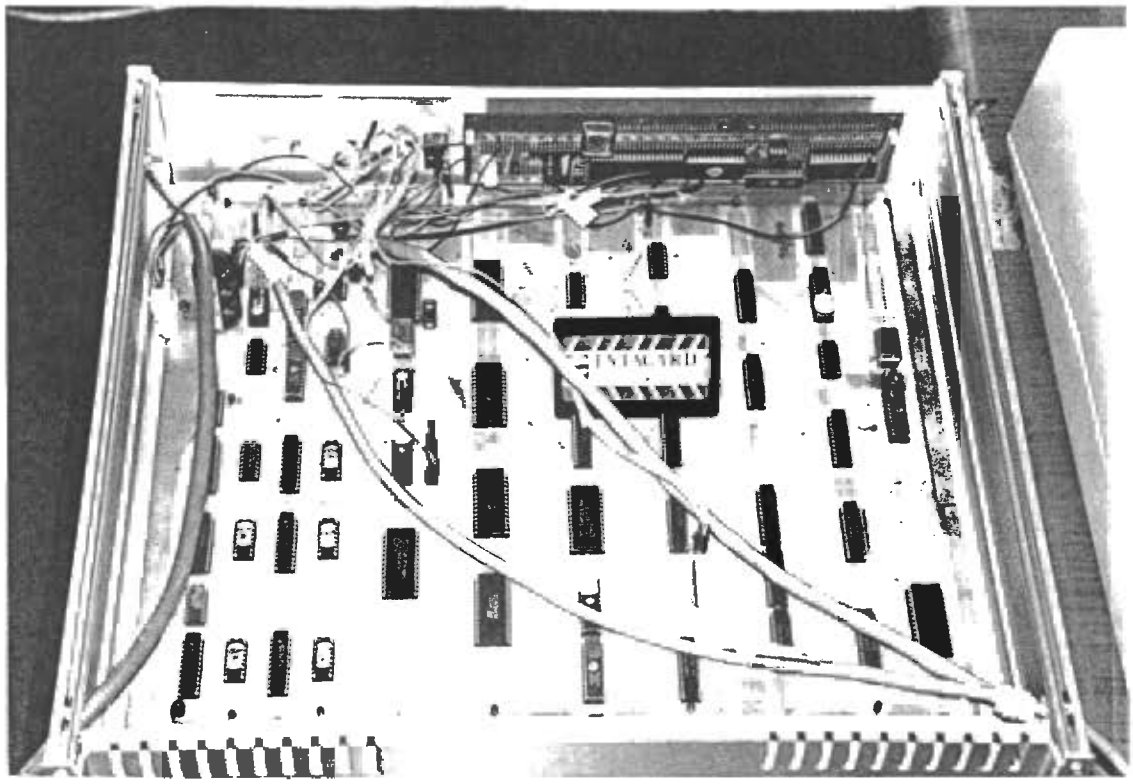


Fig. 8.2: GRIPPR hardware, internal view

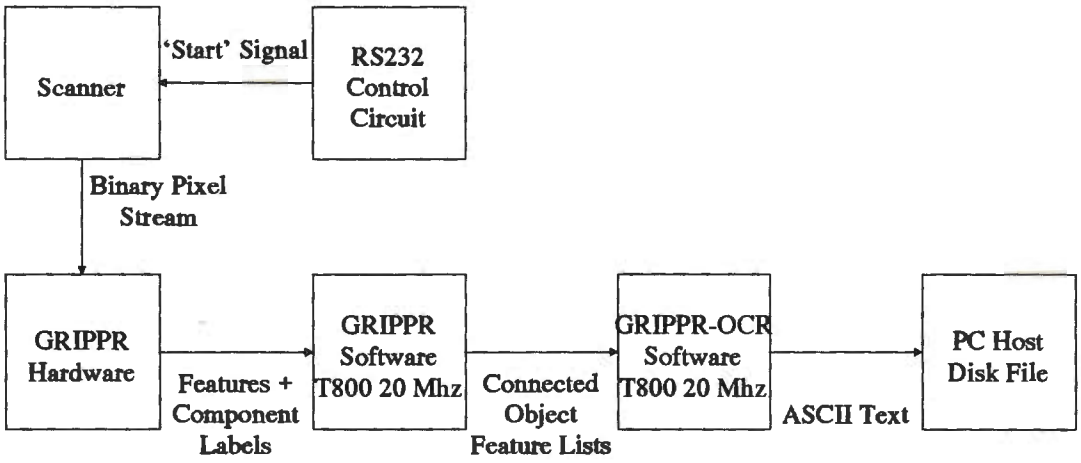


Fig. 8.3: GRIPPR-OCR System

GRIPPR-OCR system.

Test2

Test2 contains the same characters as in test1, but mixed up so that the semantic context information in the document is lost. The fount size is still the same so the relative size of characters can still be used to arbitrate between upper and lower case letters, though this technique is not used by the GRIPPR system. For this test page the inter-case contextual processing of the GRIPPR-OCR system was switched off.

Test3

Test3 is a page of very dense numbers intended to demonstrate the high throughput of the GRIPPR-OCR system. A very dense page has to be used as the scanning speed of the TAXAN TX-300 scanner is slow. Ideally, the experiment would have been conducted with a faster scanner processing a less dense document faster. Even though the document contains 15070 characters the utilisation of the GRIPPR-OCR system is still less than 100%, i.e. the processor is often idle waiting for input from the scanner.

The following key is used in the results table:

FS : False Substitution rate, percent of total

AM : Ambiguity rate, percent of total

HR : Hit rate, percent of total

PU : Processor Utilisation, percent of available (In the case of the GRIPPR-OCR machine this is the utilisation of the processor in the pipeline which has the greatest loading. This turns out to be the feature stream to connected object processor rather than the OCR application specific processor).

TT : Time Taken, seconds

CS : Characters per second

CU : Characters per second of software processor utilisation

Test1 (1280 char)	K-5200	GRIPPR-OCR
FS	0.46%	0.08%
AM	0%	3.1%
HR	99.5%	99.7%
PU	100%	8.5%
TT	23s	9.9s
CS	55.7	129.3
CU	"	1521.2

Test2 (1380 char)	K-5200	GRIPPR-OCR
FS	3.3%	0.15%
AM	0%	15.7%
HR	96.7%	99.7%
PU	100%	9.2%
TT	49s	9.9s
CS	28.2	139.4
CU	"	1515.2

Test3 (15070 char.)	K-5200	GRIPPR-OCR
FS	0.006%	0.05%
AM	0%	0.7%
HR	99.993%	99.7%
PU	100%	78%
TT	64s	9.9s
CS	235.5	1522.2
CU	"	1951.5

Very few hard conclusions can be drawn from the above statistics, except that the GRIPPR-OCR machine is very fast and the K-5200 is a highly developed commercial product which uses contextual information to achieve a very good OCR performance. Furthermore the ability of the K-5200 to recognise many more founts than the GRIPPR-OCR machine invalidates any direct comparison of recognition performance.

The processor utilisation of the K-5200 is given as 100% for each case as the processing task takes longer than the image scan. The GRIPPR-OCR processor utilisation of 78% for the dense number page test3 indicates that the maximum throughput of the system is about 1950 such numbers per second. This compares with 235.5 per second for the admittedly superior performance of the K-5200. However, the throughput of the K-5200 for the more typical test page Test1 of only 55.7 characters per second shows that its processing throughput is very dependent on the ratio of image to characters.

The much slower speed of the K-5200 in processing the context free test2 page relative to test1 demonstrates how important the semantic context of characters is for the K-5200. The relatively high false substitution rate of 3.3% for this test indicates the trade-off inherent in any system which resolves ambiguities during the OCR process. Interestingly, some of the false substitutions made by the K-5200 are

between topologically distinct characters e.g. 's' and 'z'. This indicates that the K-5200 relies more heavily on geometric rather than topological information compared with the GRIPPR-OCR system.

The consistently low false substitution rate and high speed of the GRIPPR-OCR system indicates that a system based on the GRIPPR approach might find application in DIP OCR processing for document retrieval.

8.3 Conclusions

One of the salient results of this research has been that an OCR demonstration system could be realised at all in the time available. This indicates that GRIPPR embodies an inherently practical approach to image processing.

Theoretical Conclusions

The main product of the work presented here has been the development of new algorithms for feature extraction and component labelling. The feature extraction algorithm based on the R/BSkel operations is a solution to the problem of how to generate topological features from an image in fast deterministic hardware. The component labelling algorithm presents a way of resolving the connectivity of images in real-time and for continuous scans, wherein all the pixel dependent operations can be performed in hardware.

These algorithms are brought together in the GRIPPR approach, which converts a stream of pixels into a stream of connected object feature lists. The importance of GRIPPR is that it gives a framework within which further research can be conducted. The tools for further research are provided by the PiNOSim program which can be used to simulate the many strategies and functions which could modify GRIPPR.

Further work

Possible further work includes some feature extraction which addresses the current blindness of GRIPPR to curvature. Research could also investigate algorithms which effect the connectivity of images passed to the component labeller, for example by a series of erosion/dilation stages.

Another avenue for investigation is the application of the GRIPPR approach to non-binary images. It might be possible to extend both feature extraction and component labelling algorithms directly to multilevel data. Alternatively, a different GRIPPR process could be assigned to different aspects of the same image and connected components could be associated at a higher level.

The other area for future work is to extend the broad concept of separating algorithmic complexity from computationally intensive processing to the pattern classification process. The core pattern classification process is computationally intensive, particularly if we are to envisage 'many class' recognition as would be required for the recognition of chinese characters, cursive text and character founts. The simple model of recognition developed for the OCR application is time proportional to the class depth. The speed of many class recognition is therefore limited by the time

taken to logic AND the hypothesis lists as the length of the list grows. A simple piece of hardware could act as a recognition server and data base by performing the computationally intensive long (1000 bits or more) ANDs given an hypothesis list pointer provided by a sequential process. This long AND bit-parallel processor could be based on wide bus memory architectures or possibly VRAM (Video RAM) technology.

Alternatively, the pattern classification process could be handed over to a Neural/Connectionist Net. It would be interesting to see how well Neural Nets performed, as the geometric/topological characteristics of the data generated by GRIPPR appear to make it well suited to this type of processing. Perhaps GRIPPR could provide an objective comparison of the relative performance of the current Neural/Connectionist pattern classifiers with conventional methods of pattern classification.

Technological Conclusions

The exploitation of hardware parallelism to perform all the computationally intensive pixel processing tasks in image processing for pattern recognition has produced an approach with some potentially useful characteristics. GRIPPR allows the specification of any application to be split between the pixel throughput requirement of the GRIPPR hardware and the feature processing throughput of the GRIPPR software:

- Pixel Processing: The processing of pixel data in the GRIPPR hardware means that a particular implementation can be matched to the requirements of an imaging system and that the video bandwidth of the processing can be guaranteed. Also, the continuous recycling of component label values in the connectivity process means that GRIPPR is uniquely well adapted to the processing of continuous images of infinite length. The relatively slow pixel throughput of 1M pixels/second for the OCR application presented here is merely a reflection of the throughput of the scanner used for image input. The low complexity and limited physical size of the GRIPPR hardware make the real-time processing of images at up to 50M pixels/second plausible.

- Feature Processing: By freeing the sequential von Neumann processing components from performing the computationally intensive pixel processing tasks, GRIPPR allows the algorithmically complex feature processing throughput of these components to match the throughput of imaging hardware. The higher level non-image parallel processing tasks are relatively easy to split between sequential processors by adopting process parallelism, so that the feature processing throughput of systems based on GRIPPR can also be efficiently matched to the requirements of particular applications.

Appendix

The Appendix contains five pages used in the training and testing of the GRIPPR-OCR system.

Example training page

- This page shows the input used to train the GRIPPR-OCR machine to recognise the character '8'. The page contains 5 founts in various text sizes. Four such pages where generated for every character to be recognised, two of the pages were skewed and one page was used to test the recogniser performance during the learning process.

Test page 1

- This is a page of English words, containing 1,280, 12 point characters in 5 founts.

Test page 2

- This is a page of letters without any semantic context, containing 1380 characters in 5 founts.

Test page 3

- This is a very dense page containing 15070 numeral characters designed to demonstrate the high throughput of the GRIPPR-OCR system.

Test page 4

- This page contains 35,000 simply connected objects (full-stop characters).

888888888888888888888888
88888888888888888888
888888888888888888888888

888888888888888888888888

888888888888888888888888

888888888888888888888888

888888888888888888888888
88888888888888888888
888888888888888888888888

888888888888888888888888

888888888888888888888888

888888888888888888888888

888888888888888888888888
88888888888888888888
888888888888888888888888

888888888888888888888888

888888888888888888888888

888888888888888888888888

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

the quick brown fox jumped over the lazy dog 9081726354

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

The quick, brown. fox: jumped; over? the-lazy! (dog) *

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

tGhOe,D9qYu ZiAc0.LkEbH rT8oR:wEnVf 1OoDx;EjP7u MmUpJ?e2XdO
oFvN6!eWrOt Rh3B*e)KlCaI 5zUy-Q(dEo4HgT

5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
5678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456

References

[1] Duff M.J.B., Ed., 'Intermediate Level Processing', Academic Press, London, 1986.

[2] Aleksander I.: 'Myths and realities about neural computing architectures', Parallel Processing and Artificial Intelligence, ed. Reeve M., Zenith S.E., pp. 1-13, John Wiley & Sons, Chichester, England, 1989.

[3] Duff M.J.B.: 'Complexity', Intermediate Level Processing, Chapter 19, pp 307-314, Academic Press, London, 1986.

[4] Taylor C.J., Dixon R.N., Gregory P.J., Graham J.:'An Architecture for Integrating Symbolic and Numerical Image Processing', Intermediate Level Processing ed. Duff M.J.B., Academic Press, London, 1986.

[5] Gregory R.L., 'Eye and Brian, the Psycology of Seeing', 4th ed., Weidenfeld and Nicholson, London, 1990.

[6] Unger S.H., 'A computer orientated towards spatial problems', Proceedings IRE, Vol. 46, pp 1744-1750, 1958.

[7] Mcormick B.H., 'The illinois pattern recognition computer', IEEE Transactions on Elctronic Computers, Vol. EC-12, 1963, pp 791-813.

[8] Duff M.J.B., Watson D.M., Fountain T.J.: 'A Cellular Logic Array for Image Processing', Pattern Recognition, Vol. 5, pp. 229-247, 1973.

[9] Duff M.J.B., Watson D.M., 'The cellular logic array image processor', Computer Journal, V.20, pp. 68-72, 1977.

[10] Preston K., Duff M.J.B., Levialdi S., Norgen P.E., Toriwaki J.: 'Basics of Cellular Logic with Some Apllications in Medical Image Processing', Proc. IEEE, vol. 67, No. 5, pp. 826-856, 1979.

[11] Lougheed R.M., McCubbery D.L., 'The Cytocomputer: A practical Image Processor', The 7th Annual Symposium on Computer Architecture, IEEE, 1980.

[12] Chen K., Astrom A., Danielsson P.E., 'PASIC. A smart sensor for computer vision', Proc. 10th International Conference on Pattern Recognition, IEEE, June 1990.

[13] Awcock G.J., 'An Edge Detecting Image Sensor Architecture', IEE Colloquium 'Binary Image Processing Techniques and Applications', March 1991.

[14] Rosenfeld A., Kak A.C.:'Digital Picture Processing', Academic Press, New York, 1982

[15] Fountain T.J., Matthews K.N., Duff M.J.B., 'The CLIP7A Image Processor', IEEE transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 3, 1987.

[16] Dew P. and Manning L. 'Comparison of Systolic and SIMD architectures for Computer Vision Computations' in 'Systolic Arrays' ed: Moore W., McCabe A., Urquhart R., Hilger, Bristol, 1987.

[17] Golay M.J.E.: 'Apparatus for Counting Bi-Nucleate Lymphocytes in Blood', U.S. Patent No. 3,214,574, 1965.

[18] Gray B.S., 'Local properties of binary images in two dimensions', IEEE Transactions on Computers, Vol. C-20, pp. 551-561, 1971

[19] Kruse B.: 'A parallel picture processing machine', IEEE Trans. Comput., vol. C-22, No.12, pp. 1075-1087, 1973.

[20] Kruse B.:'Design and implementation of a picture processor', Linkoping Studies in Science and Technology Dissertations No13, Linkoping, Sweden, 1977.

[21] Sternberg S.R. 'Automatic Image Processor' U.S. Patent, No. 4,167,728, Sept 1979.

[22] Lougheed R.M., McCubberey D.L., Sterberg S.R.: 'The Cytocomputer: Architectures for Parrallel Image Processing', Proc. IEEE Workshop on Picture Data Description and Management, pp. 281-286, August 1980.

[23] Wambacq P., Van Eycken L., De Roo J., Oosterlinck A., Van den Berghe H.: 'Description of Two Hardware Convolvers as a part of a General Image Computer', IEEE Pattern Recognition and Image Processing, 1981.

[24] Gennery D.B., Wilcox B.: 'A Pipelined Processor for low-level vision', Proc. IEEE conference on Computer Vision and Pattern Recognition, 1985.

[25] Yoda H., Ohuchi Y., Taniguchi Y., Ejiri M., 'An Automatic Wafer Inspection System Using Pipelined Image Processing Techniquies', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 1, January 1988.

[26] Riganati J.P., Vitols A. 'Two-Dimensional Binary Data Enhancement System' U.S. Patent, No. 4,003,024, January 1977.

[27] Kondo T., 'Line Figure Connecting Apparatus', U.S. Patent, No. 4,855,933, Aug 1989.

[28] Persoon E., 'A Pipelined Image Analysis System Using Custom Integrated Circuits', Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 1, 1988.

[29] Rosenblatt F., 'The Perceptron:a probabilistic model for information storage and organization in the brain', Psychological Review 65, pp 386-408, 1958.

[30] Block H.D., 'The Perceptron: a model for brain functioning', Reviews of Modern Physics 34, pp 123-135, 1962.

[31] Minsky M., Papert S., 'Preceptrons', MIT Press, Cambridge, Mass, 1969.

[32] Alexandar I., Morton H., 'An Introduction to Neural Computing', Chapman and Hall, London, 1990.

[33] Sivilotti M.A. Mahowald M.A. Mead C.A. 'real-time visual computations using analog CMOS processing arrays', Advanced Research in VLSI:Proceedings of the 1987 Stanford Conference, MIT Press, pp. 295-312, 1987.

[34] Fukushima K., Miyake S., Ito T., 'Neocognitron:a neural network model for a mechanism of visual pattern recognition', IEEE transactions on Systems, Man and Cybernetics SMC-13, pp 826-834, 1983.

[35] Bienenstock E.L., Cooper L.N., Munro P.W., 'Theory for the development of neron selectivity:orientation specificity and binocular interaction in visual cortex', Journal of Neuroscience, Vol. 2, pp 32-48, 1982.

[36] Boccignone G., Marcelli A., Steensma J. 'Investigation on a Structural Solution of Merged Characters Segmentation in OCR'. Lecture Notes in Computer Science, No. 399, 'Recent Issues in Pattern Analysis and Recognition', Springer-Verlag, Berlin, 1989.

[37] Kronsjo L., 'Computational Complexity of Sequential and Parallel algorithms', John Wiley and Sons, 1985.

[38] W. Daniel Hillis, 'The Connection Machine', MIT Press, Cambridge Mass., 1985.

[39] Rosenfeld A., 'Connectivity in Digital Pictures', Journal of ACM, vol 17, pp. 146-160, 1970.

[40] Ye Q.Z., Danielsson P.E., 'Inspection of Printed Circuit Boards by Connectivity Preserving Shrinking', IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, No. 5, September 1988.

[41] Cypher R., Sanz L.C., Snyder L., 'An EREW PRAM Algorithm for Image Component Labelling', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11, No.3, pp. 258-262, March 1989.

[42] Rosenfeld A., Pfaltz J. L., 'Sequential Operations in Digital Picture Processing', Journal of the ACM, Vol. 13, No. 4, pp. 471-494, 1966.

[43] Komeichi M., Ohta Y., Gotoh T., Mima T., Yoshida M., 'Video-Rate Labelling Processor', SPIE Vol. 1027, 1988, pp. 69-76, 1988.

[44] Gotoh T., Ohta Y., Yoshida M., 'High-Speed Algorithm for Component Labelling', Systems and Computers in Japan, Vol. 21, No. 5, 1990.

[45] Selkow S.M., 'One-Pass Complexity of Digital Picture Properties', Journal of the ACM, Vol. 19, No. 2, April 1972.

[46] Cederberg R.L.T., 'Chain-Link Coding and Segmentation for Raster Scan Devices', Computer Graphics and Image Processing, Vol. 10, pp. 224-234, 1979.

[47] Danielsson P.E., 'An improved Segmentation and Coding Algorithm for Binary and Nonbinary Images', IBM J. Research Developments, Vol. 26, No. 6, Nov. 1982.

[48] Ronse C., Devijver P.A., 'Connected Components in Binary Images: the Detection Problem', Research Studies Press, Herts, England, 1984.

[49] Okazaki A., Kondo T., Mori K., Tsunekawa S., Kawamoto E., 'Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 3, May 1988.

[50] Stefanelli R., Rosenfeld A., 'Some Parallel Thinning Algorithms for Digital Pictures', Journal of the ACM, Vol. 18, No. 2, April 1971, pp. 255-264, 1971.

[51] Pavlidis T., 'A vectorizer and feature Extractor for Document Recognition', Computer Vision, Graphics and Image Processing, Vol. 35, pp. 111-127, 1986.

[52] Baird H.S., Kahan S., Pavlidis T, 'Components of an Omnifont Page Reader', IEEE Proceedings eighth International Conference on Pattern Recognition, Paris, France, Oct. 1986.

[53] Govindan V.K., Shivaprasad A.P., 'Character Recognition - A Review' Pattern Recognition, Vol. 23, pp. 671-683, 1990.

[54] Devijver P.A., Kittler J., 'Pattern Recogntion, A Statistical Approach', Prentice-Hall, 1982.

[55] Pavlidis T., 'Structural Pattern Recognition', Springer-Verlag, 1977.

[56] Fu K.S., 'Syntactic Pattern Recognition and Applications', Prentice-Hall, 1982.

[57] Baird H.S., 'Feature Identification for Hybrid Structural/Statistical Pattern Classification', Computer Vision, Grpahics, and Image Processing, Vol. 42, No. 3, June 1988.

[58] Hall A.D., 'Pipelined Logical Convolvers for Binary Picture Processing', Electronic Letters, Vol.25, No.15, 1989.

[59] Tout N., Norton-Wayne L., Reedman D., 'Automated Identification of Shoe Upper Components', IEE Colloquium on: Binary Image Processing and Applications, March 1991.

[60] Ye Q.Z., Danielsson P.E., 'Inspection of Printed Circuit Boards by Connectivity Preserving Shrinking', IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 10, No. 5, September 1988.

[61] Hilditch, 'Linear skeletons from square cupboards', in Machine Intelligence IV, Meltzer B., Michie D., Eds. Edinburgh University Press, pp. 403-420, 1969.

[62] Jeung Y.C., 'A 50Mbit/sec. CMOS Cideo Linestore System', SPIE Vol. 1001 Visual Communications and Image Processing, 1988.

[63] 'Transputer Technical Notes', Prentice Hall International (UK), 1989.