

University of South Wales



2060491

Bound by
Abbey Bookbinding Co.,
Cardiff
Tel: (0222) 395882
Fax: (0222) 223345

University of Glamorgan
Prifysgol Morgannwg



Pontypridd
Mid Glamorgan CF37 1DL
Telephone 0443 480480
Fax 0443 480558

**A MULTI RESOLUTION MODULAR SENSING SYSTEM
FOR ROBOTIC APPLICATIONS**

Jonathan Andrew Ware.

Department of Mathematics and Computing
University of Glamorgan
Pontypridd.

A thesis submitted in partial fulfilment of the requirements of the
Council for National Academic Awards (CNAA) for the
degree of Doctor of Philosophy.

September 1992.

TABLE OF CONTENTS

Table of Contents	<i>i - v</i>
Table of Figures	<i>vi - ix</i>
Acknowledgements	<i>x</i>
Declarations	<i>xi - xii</i>
Abstract	<i>xiii</i>
Thesis Outline	<i>xiv</i>
1. Introduction	
1.1 Background	2
1.2 Automated Manufacture	2
1.3 Types of Robot in Current Use	4
1.4 Sensory Devices	5
1.5 Multi-Sensor Systems	6
1.6 The Aim of Project	7
1.7 Summary	8
2. An Overview of the Prototype System	
2.1 Introduction	10
2.2 Description of the Prototype System	10
2.2.1 Object Location	10
2.2.2 Object Identification	12
2.2.3 User Interface	13
2.3 Processing Requirements	15
2.4 Summary	16
3. The Development of an Efficient Data Structure for Modelling a 3-D workspace	
3.1 Introduction	18
3.2 Octrees	18
3.2.1 Results of Investigations Using Standard Octrees	20
3.3 The Modified Octree	21
3.4 The Li and Telfer Quadtree Modification	25
3.4.1 The Li and Telfer Modification Applied to Octrees	27
3.5 Run-Length Encoding	28
3.6 Conclusion	34

4. Parallel Processing	
4.1 Introduction	36
4.2 Parallel Processing Techniques and Architectures	36
4.2.1 Parallel Processing Architecture	37
4.2.2 Allocation of Processors to Processes	38
4.3 Transputer Systems	39
4.4 The Transputer and Occam	41
4.4.1 Occam Constructs	42
4.4.2 The Suitability of Occam	43
4.5 Transputer Projects	44
4.6 Summary	45
5. The Development of a Transputer Network	
5.1 Introduction	48
5.2 Implementation Principles	48
5.3 Implementation of the Network	49
5.4 Allocation of Transputers	50
5.5 Routing of Data Around Network	51
5.6 System Testing	54
5.7 Problems Encountered	55
5.7.1 Projection Rounding Errors	55
5.7.2 Processor Performance Mismatch	56
5.8 Summary	57
6. Object Identification	
6.1 Introduction	59
6.2 Overview of Object Identification	59
6.2.1 The Position of Objects in Space	59
6.2.2 Identification of an Object by its Geometrical Features	60
6.3 Overview of the Developed System	61
6.4 Details of the Developed System	63
6.5 Modifications to the System	65
6.5.1 Calculation of the Weighted Certainty Factor	65
6.5.2 Calculation of the Z-Score	66
6.6 Object Set Classification	69
6.7 Discussion and Conclusions	71

7. Testing of the Object Identification Module	
7.1 Module Testing	73
7.2 Results of Testing	73
7.3 Further Testing	77
7.4 Conclusions	79
8. Conclusions and Future Work	
8.1 Introduction	81
8.2 Summary	81
8.2.1 Advantages of using a Parallel Processing Architecture	82
8.2.2 Advantages of a Modular Sensing System	83
8.2.3 Additional Benefits of Work	85
8.3 Conclusions	86
8.3.1 The Data Structure Used	87
8.3.2 The Suitability of the Transputer Architecture	88
8.3.3 The Viability of the Completed System	89
8.4 Suggestions for Future Work	90
8.4.1 More Dynamic Processor Allocation	91
8.4.2 Introduction of a Rule Based System	91
8.4.3 Extension to the use of Certainty Factors	92
8.4.4 Object Identification using a Neural Network	93
8.4.5 The Addition of Further Modules	94
8.5 Final Remarks	96
References	98
Appendix 1 - Transputer Equipment and Algorithms	
A1.1 The Editing Environment	1 - 2
A1.2 The Harlequin Image Processing Transputer	1 - 3
A1.3 Hardwire Link Configuration	1 - 4
A1.4 Software Petitioning	1 - 7
Appendix 2 - Details of Algorithms Employed	
A2.1 Introduction	2 - 2
A2.2 Optimal Thresholding	2 - 2
A2.3 Noise Removal	2 - 4
A2.4 Object Segmentation	2 - 5
A2.5 Calculation of Number of Holes	2 - 7
A2.6 Calculation of Convex Hull	2 - 8
A2.7 Calculation of Principal and Secondary Axis	2 - 9

Appendix 3 - Calibration and Calculation

A3.1	Camera Calibration	3 - 2
A3.2	Calculation of Projection Through Workspace	

Appendix 4 - Equipment Used

A4.1	Introduction	4 - 2
A4.2	The Tandy 3000	4 - 2
A4.3	The RTX robot arm	4 - 3
A4.4	The CCD cameras	4 - 7

Appendix 5 - Published Papers

A5.1	Ware J.A. Roberts G. Davies R.A. Miles R. & Williams J.H. (1990) "A modular sensing system for robotic control", The Second International Conference on Applications of Transputers, Southampton University, July, pp. 78-85.	4 - 3
------	--	-------

The paper also appears in the proceedings of the Fourth World Conference on Robotic Research, Society of Manufacturing Engineers, held at Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1991, pp 6:13 - 6:23 under the title "A parallel processing architecture for robotic control."

The SME will also be publishing the paper in their special addition of Transactions on Robotics Research.

A summary of the paper was also presented at the IEE colloquium on Parallel Processing: Industrial and Scientific Applications, at The IEE, Savoy Place, London, 18'th June 1990, under the title "A transputer based modular sensing system."

A5.2	Ware J.A. Roberts G. & Davies R.A. (1991) "Determining the location and identity of components in an A.M.P. environment", IEE Third International Conference on Software Engineering for Real Time Control, Cirencester, September, pp 109 - 114.	4 - 14
------	--	--------

The paper also appears in the proceedings of VISION '90, Society of Manufacturing Engineers, held in Detroit, Michigan, November 1990, pp 9:1 - 9:10 under the title "Building and storing a model of a robot's workspace using a series of 2-D images."

A summary of the paper was also presented at the IEE colloquium on Computer Image Processing and Plant Control, at The IEE, Savoy Place, London, 18'th May 1990, under the title "The location of components in an automatic manufacturing process."

- A5.3 Ware J.A. Roberts G. & Davies R.A. (1990)** 5 - 29
"A multiple camera system to facilitate object manipulation within a robot workspace", The Second Symposium on Personal Computers in Industrial Control, Warren Spring Laboratory, Stevenage, November, pp 81 - 88.
- A summary of the paper was also presented at the IEE colloquium on Binary Image Processing - Techniques and Applications, at The IEE, Savoy Place, London, 8'th March 1991, under the title "Determining the location of components in an A.M.P. environment."
- A5.4 Ware J.A. Roberts G. & Davies R.A. (1991)** 5 - 38
"A user interface for robotic sensing systems: A multi-faceted approach", Colloquium on HCI: Issues For The Factory, The IEE, Savoy Place, London, 21'st February.
- A5.5 Ware A. & Kidner D. (1991)** 5 - 47
"Parallel implementation of the Delaunay triangulation within a transputer environment", The Second European Conference on Geographical Information Systems (EGIS '91), Brussels, pp. 1199-1208.

TABLE OF FIGURES

Figure 2.1	The prototype work cell.	11
Figure 2.2	Shows how successive views can be used to determine the enclosing volume of an object resting on a workbench.	12
Figure 2.3	Shows the plan, side elevation and front elevation of a robot's workspace containing a single object.	12
Figure 2.4	Shows the main tasks that will be performed and the expected frequency.	14
Figure 3.1	An upright cubical region can be successively broken down into progressively smaller octants until the resulting octant represents a single voxel.	19
Figure 3.2	Shows how a 3-D object can be represented by an octree.	20
Figure 3.3	Shows the octree from figure 3.2 and its equivalent modified octree.	21
Figure 3.4	Shows a standard octree from and its equivalent modified octree.	21
Figure 3.5	Shows how a branch and its connected leaves can be replaced by a single entity.	22
Figure 3.6	Shows the format of the two dimensional array used to hold the modified octree data structure.	22
Figure 3.7	Shows a pyramid whose height and length of base is equal to the height and length of the workspace.	24
Figure 3.8	(1) Maximum number of entries in table using standard branch and node approach. (2) Maximum number of entries in table using branch and twig approach.	24
Figure 3.9	(1) Number of entries in table at end using standard branch and node approach. (2) Number of entries in table at end using branch and twig approach.	24
Figure 3.10	(1) Time taken to build table using standard branch and node approach. (2) Time taken to build table using branch and twig approach.	25
Figure 3.11	(a) A binary image. (b) A two dimensional area recursively divided into progressively smaller quadrants.	26
Figure 3.12	The 2D object of figure 3.7(b) can be represented by a quadtree.	26
Figure 3.13	In the Li and Telfer representation there are sixteen primitive node types.	27
Figure 3.14	(a) Shows how the state of the work cell could be recorded without recourse to a table entry for each lowest level voxel. (b) Shows how the data structure in figure 3.10(a) could be further reduced by combining the sum of eight binary values into their denary equivalent.	28
Figure 3.15	Depicts three sets of numbers that can be compacted using run-length encoding techniques.	29

	Figures
Figure 3.16 (a) Object (or volume) segmented into planes. (b) Planes split into lines.	31
Figure 3.17 Data structure schema for the volume segment representation.	31
Figure 3.18 Shows an object and its three end on views.	32
Figure 3.19 (1) Time taken to build table using Octree branch and twig approach. (2) Time taken to build table using Partial run-length encoding.	34
Figure 3.20 (1) Table entries using Octree branch and twig approach. (2) Table entries using Partial run-length encoding.	34
Figure 4.1 A von Neumann architecture has three major components.	36
Figure 4.2 A schematic diagram of the T800 transputer.	39
Figure 4.3 Transputers can be arranged in a variety of ways for example:- (a) a tree, (b) a mesh.	40
Figure 4.4 (a) As the number of processors in a network increases so does the communications overhead incurred. (b) There is an optimum number of processors in a network for minimising task turnaround time.	40
Figure 5.1 The model of the workspace was held on eight transputers. Each transputer holding a section of 128 by 128 by 128 voxels.	50
Figure 5.2 Transputer layout for prototype system.	51
Figure 5.3 Format for passing data from sensor to model.	52
Figure 5.4 A diagrammatical view of the processes executing concurrently on the first processor in the network.	53
Figure 5.5 A diagrammatical view of the processes executing concurrently on all but the first processor in the network.	53
Figure 5.6 (a) Actual path of two 'parallel lines' projected through the workspace. (b) Calculate path of two 'parallel lines' projected through the workspace.	55
Figure 5.7 Additional transputers were added to network to balance the work load.	56
Figure 5.8 Modified data packet for inter-processor communication.	57
Figure 6.1 For each feature the mean and standard deviation of the normal distribution is calculated.	67
Figure 6.2 With scale invariant objects the same describing feature values are obtained at both A and B.	70
Figure 7.1 Object set 1.	73
Figure 7.2 Object set 2.	74

	Figures
Figure 7.3	Object set 3. 76
Figure 8.1	A diagrammatical overview of the system. 85
Figure 8.2	(a) Using the intersection of views to determine the position of objects can lead to incorrect conclusions being made. 90 (b) Using the intersection of views to determine the position of objects can produce very good results.
Figure 8.3	(a) Using a third view at the intersection of the first two views can produce a more accurate model of the workspace.. 92 (b) Using a third view at the intersection of the first two views can produce a less accurate model of the workspace.
Figure 8.4	The three views of the object reveal different information about its position. 93
Figure 8.5	Shows how additional modules could be added to the system 95
Figure A1.1	Shows how five sub-folds can be held within a single fold. 1 - 3
Figure A1.2	The Harlequin is designed to provide the essence of a high performance transputer based workstation. 1 - 4
Figure A1.3	A diagrammatical view of the inter transputer connections in the final system implementation. 1 - 5
Figure A2.1	A binary image made up of two unimodal distributions. 2 - 2
Figure A2.2	An image with four connected objects 2 - 5
Figure A2.3	Each black pixel is given a unique serial number. 2 - 6
Figure A2.4	Each object has a unique number. 2 - 6
Figure A2.5	Shows the stream direction towards an object. The object has 4 and 3 concavities in view. 2 - 7
Figure A2.6	Shows the patterns that needs to be counted for discrete binary images when the stream direction is taken as NW to SE. 2 - 7
Figure A2.7	A binary image where $X=2$ and $Y=2$ therefore $E=0$. 2 - 8
Figure A2.8	Shows how the convex hull of a 2-D image is produced. 2 - 9
Figure A2.9	Every object has at least one principal and secondary axis, although some may have many. 2 - 9
Figure A3.1	Shows the error introduced into the system when the image is assumed to be a parallel projection of the object. 3 - 2
Figure A3.2	Position through workspace given by X, Y and Z. 3 - 4
Figure A4.1	Shows the components of the RTX. 4 - 4
Figure A4.2	Shows the dimensions of the RTX. 4 - 6
Figure A4.3	Shows the range of movements of the RTX. 4 - 7

Table 3.1	The octree data structure can be represented in the form of a table.	23
Table 3.2	Represents the y,z plane.Each entry in the table represents a row of voxels perpendicular in the y,z plane.	33
Table 3.3	Contains details related to non-empty voxels.	33
Table 6.1	The mean and standard deviation for each feature is calculated.	64
Table 6.2	The object set can be represented by a table.	64& 67
Table 6.3	The feature's certainty factor is calculated for each object.	65
Table 6.4	The feature's weighted certainty factor is calculated for each object.	66
Table 6.5	Shows the output produced when an attempt is made to identify the test object with threshold set at S.D.=1.	68
Table 6.6	Shows the output produced when an attempt is made to identify the test object with threshold set at S.D.=0.5.	69
Table 7.1	Feature table for object set 2, when objects were measured in a scale invariant environment.	74
Table 7.2	Feature table for object set 2, when objects were measured in a scale variant environment.	75
Table 7.3	Shows the range of significant values at which objects from object set 2 were identified correctly in a scale invariant environment.	75
Table 7.4	Show the range of significant values at which objects from object set 2 were identified correctly in a scale variant environment.	76
Table 7.5	Feature table for alphabet when letters were measured in a sale invariant environment.	77
Table 7.6	Shows the range of significant values at which letters from alphabet were identified correctly in a scale invariant environment.	78
Plate A1.1	Shows the 'reconfigurable links', as they appear in the expansion slots of the host computer.	1 - 6
Plate A4.1	An overall view of the equipment used during the course of the research.	4 - 2
Plate A4.2	Shows the 'reconfigurable links', as they appear in the expansion slots of the host Tandy 3000.	4 - 3
Plate A4.3	The RTX's end effector, before it was modified to carry a camera.	4 - 5
Plate A4.4	The RTX's end effector, after it was modified to carry a camera.	4 - 5

Acknowledgements

I wish to express sincere thanks to my supervisors at the University of Glamorgan, Dr. G.Roberts and Dr. R.A.Davies, and to Dr. J.H.Williams, my supervisor from Cardiff University, for the encouragement, guidance and constructive criticism they have provided during this research project.

Thanks are also extended to Dr. R.G. Miles, of The Bristol Transputer Centre, for his help during the initial stages of the work on transputer networking, and to Dr. D.B. Kidner for his partnership during the work on parallel Delaunay triangulation.

Finally, thanks are due to the Royal Signals and Radar Establishment (R.S.R.E.) and to The Fulbright Commission, for their financial support. R.S.R.E partially funded the work on parallel Delaunay triangulation, while without the support of The Fulbright Commission it would not have been possible to present the research covered in this thesis to such a wide spectrum of international researchers and practitioners.

Certificate of Research

This is to certify that, except where specific reference is made, the work presented in this thesis is the result of the investigation undertaken by the candidate.

Candidate Andrew Ware

Director of
Studies J. Roberts

A Multi Resolution Modular Sensing System For Robotic Applications

Jonathan Andrew Ware

This thesis documents the research that has led to the development, in prototype form, of a modular sensing system for use in a robot's work cell. The system implemented overcomes one of the major limitations of existing sensing systems, that is the difficulty of altering their sensing characteristics. The majority of sensing systems that are currently available are inflexible in that the addition of extra sensors requires, at the very least, substantial changes to both hardware and software. What these systems require is a facility through which users can easily, and readily, make changes to the configuration of the sensors they employ.

In the modular system described, the sensors that provide the information about the robot's work cell are independent of the algorithms that make use of the information. That is, the sensors do not need any knowledge as to when or how the information they provide will be used. Similarly, the algorithms that make use of the data do not need any knowledge as to the provider of the data. This separation of data provider from data user enables the software that controls the sensors (and even the sensors themselves) to be upgraded without corresponding changes to the data user software. Additional sensors can easily be added to the system while redundant sensors can simply be removed.

The location of objects within the robot's workspace is achieved by building a model of the workspace using the information provided by a number of sensors. As a prerequisite to model construction three problems had to be addressed. Firstly, the information extracted from different sensors is generally at different resolutions. Secondly, the representation of 3-D space requires large amounts of computer memory. Thirdly, the production of the 3-D model, particularly when a large number of sensors are involved requires a substantial amount of processor time. The first two problems were addressed using a data structure that allowed compact data storage, while the final problem was reduced by identifying parallel aspects of the processing and implementing them on a network of transputers.

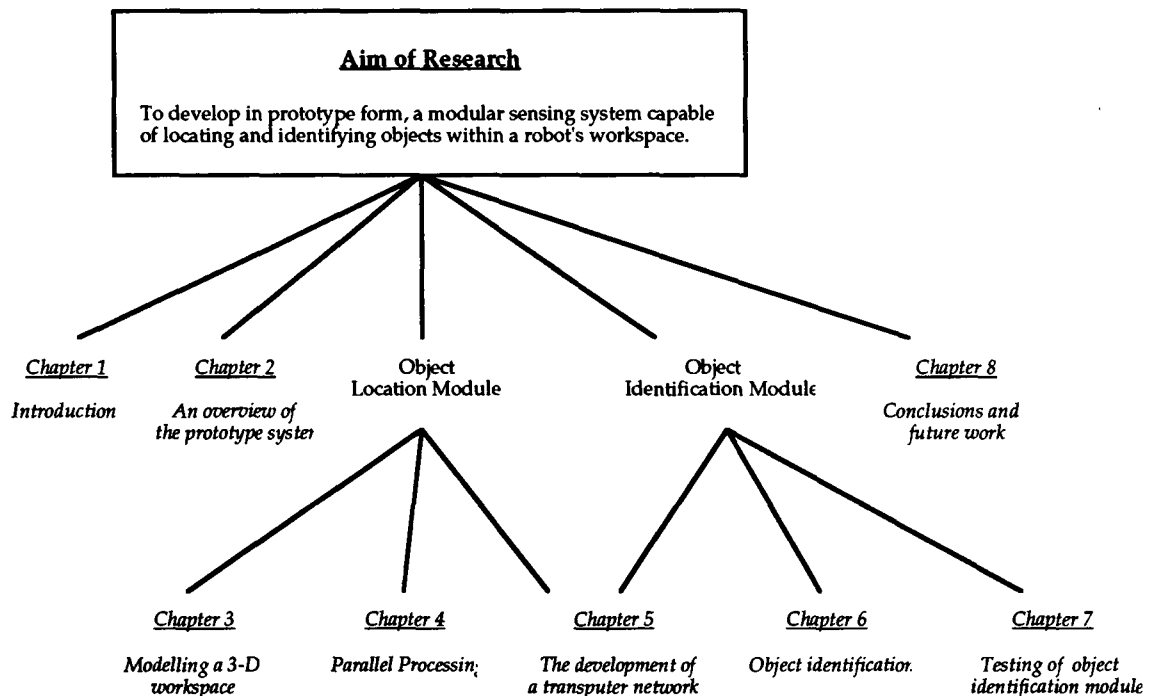
After the objects within the robot's workspace have been located, the next stage is to identify them. The identification is achieved by calculating the degree of match between measurable characteristics of the object to be identified and the same measurable characteristics of known objects. The degree of match, which is similar but not identical to the correlation function, between the object to be identified and each known object is then used to determine, if possible, the required identity of the object.

The work contained within the thesis not only demonstrates the feasibility and benefits of a modular sensing system, over traditional sensing system, but has brought to light some points that will need further thought before a fully functional system is produced. The last chapter contains, in addition to a full and detailed list of conclusions made during the research, a summary of some of these areas that still require further work.

Thesis Outline

If the current rate of deployment of robotic sensing systems into the manufacturing environment is to be maintained, then there needs to be an improvement in both the computational efficiency and computational power available to these systems. This thesis documents the research that has led to the development and implementation, in prototype form, of a modular sensing system serviced by a parallel processing architecture that facilitates the achievement of both these requirements.

Chapter 1 provides the background and rationale for the research, while chapter 2 outlines the function of the two main modules of the sensing system. The development of a suitable data structure to model the robot's workspace is given further consideration in chapter 3. This is followed by an overview of parallel processing techniques and architectures in chapter 4 before the implementation of the data structure on one such architecture is detailed in chapter 5. Chapters 6 and 7 detail the process of object identification. Chapter 8 draws the thesis to a close with a summary of conclusions reached and some recommendations for future work.



Chapter 1

Introduction

1.1 Background

Today's automated manufacturing techniques⁽¹⁾, which are used in the mass production of items from television sets and washing machines to pens and pencils, are all well advanced. It is now the norm to build expensive, yet cost-effective, dedicated assembly systems when the number of discrete products is high. These expensive dedicated systems are, however, uneconomical for the production of batches where the number of discrete products is low. For these small-run batches a more flexible approach is required. One such approach is programmable industrial automation.

Programmable industrial automation is now an integral part of many manufacturing processes. This is due, in part, to the marriage of robotic and computer technology which has provided a powerful, flexible, and cost effective tool for employment on the factory floor. While it has been argued in some quarters⁽²⁾ that progress has not been as significant as it could have been, there is no doubting the impact that both technologies have had on the manufacturing industry. However, if the rate of progress in deploying these two technologies is to be maintained, then there needs to be a significant improvement in both the computer power available to the designers of new systems and to the way in which this additional power is employed.

The research and analysis documented in this thesis has led to the design, development, and implementation of a modular sensing system capable of modelling a robot's workspace. The purpose of this introductory chapter is to set the scene for the remainder of the thesis and seeks to introduce the field of automated manufacture and robotics before giving an overview of the aims and objectives of the work undertaken.

1.2 Automated Manufacture

Nearly 20 years ago Merchant⁽³⁾ envisaged an automatic factory consisting of a number of modular subsystems, each controlled by a hierarchy of small computers that interfaced to a large central computer. In the system he envisaged the subsystems would perform the seven manufacturing functions namely: product design, product planning, part forming by work stations, material handling by different computer-controlled devices, assembly of parts and subassemblies, inspection of assemblies, subassemblies and parts, and organisation of product information. Almost two decades have passed since Merchant first described the factory of the future and yet, on the whole, the manufacturing industry still falls

far short of the vision he gave.

Redford⁽²⁾, reflecting on the use of industrial robots during the last 20 years, makes the comment that *'it is, perhaps, unfortunate that the initial use of robots was for spot welding, paint spraying and materials handling. These operations which are invariably cost justifiable are characterised by a highly articulate, not-very-accurate end effector. This led to a decade of unsuccessfully attempting to use this kind of manipulator for assembly where only limited articulation is usually required but where repeatable positioning is important.'*

Redford suggests that instead of realising that developments were going along the wrong course and stopping to reconsider their position, designers and manufactures exerted considerable efforts trying to control uncontrollable equipment. He points out that this folly, when compounded by a seeming obsession to make manipulators 'like people' - that is to endow them with the anthropomorphic properties of vision, high-level tactile sensing and artificial intelligence - led to systems that, while in theory should be capable of anything, were, in reality, capable of doing virtually nothing. Redford concludes his paper on a positive note stating that while the last 20 years have not been very constructive they have been illuminating. Today's researchers have learned from past mistakes and are now taking a more pragmatic view of the situation, basing their efforts on assembly activities with goals that are achievable.

While progress might not have been as great as first hoped, it is fair to say that a substantial number of manufacturing tasks have now been automated using industrial robots. The automation has been facilitated not only by improvements in robot technology, but also from the capability of the robot to interact with the external world⁽⁴⁾.

The addition of sensors to the robot's workspace has led to the breakdown of many of the main obstacles to the broader application of robots in industry, including the mechanical manipulation of randomly oriented parts^(5,6). (Note that Redford describes the efforts exerted in solving problems such as bin-picking as 'inappropriate' suggesting that they stem from the incorrect premise that assembly is a pseudo-random unconstrained process.)

1.3 Types of Robot in Current Use

Robots are generally seen as anthropomorphic, combining human qualities of intelligence, mobility and manipulative ability. The robot's end effector - it may be a spray painter, screw driver, welder, or gripper - equates to the tool in the hand of a human. In a technical sense, industrial robots are universally applicable kinetic machines which, in contrast to conventional machines, can carry out a relatively complicated series of movements.

There is some divergence of opinion as to what a robot actually is. For example, the Electric Machinery Law of Japan defines an industrial robot as 'an all purpose machine equipped with a memory device and a terminal device (for holding things), capable of rotation and of replacing human labour by automatic performance of movements'. The American Robot Industry Association however, defines a robot as 'a manipulator designed to move material, parts, tools, or specialised devices, through variable programmed motion for the performance of a variety of tasks'.

There are, in general, six categories of robot recognised throughout the industrial world, these being:-

- (1) Manual Manipulator - a manipulator that is worked by an operator.
- (2) Fixed sequence robot - a manipulator which repetitively performs successive steps of a given operation according to a predetermined sequence, condition, and position, and whose set information cannot be easily changed.
- (3) Variable sequence robot - a manipulator which repetitively performs successive steps of a given operation according to a predetermined sequence, condition, and position, and whose set information can be easily changed.
- (4) Playback robot - a manipulator which can reproduce, from memory, operations originally executed under human control.
- (5) NC (numerical control) robot - a manipulator that can perform a given task: the task being coded as a sequence of numerical data.
- (6) Intelligent robot - this robot uses sensory perception (visual and tactile) to independently detect changes in the work environment or work condition and, by its own decision making faculty, proceeds with its operations accordingly.

Although robots are extensively used in many of the world's industrialised countries, most of these are of robot categories 1-5. Much research and

development work is still required on sensory control systems before intelligent robots achieve their full potential. In the research documented in this thesis it is these intelligent robots that are under consideration. A brief overview of how sensory devices can be used to enhance these robots will now be given to set the scene for further discussion.

1.4 Sensory Devices

Robots, like humans, must gather extensive information about their environment in order to function effectively. This information is gathered and fed to the robot via sensors (7,8,9). Sensors may be categorised by their characteristics, by the types of physical phenomena that they measure or by application. These categories do overlap, but will still be useful in understanding the many types of sensors and the way in which they can be evaluated, selected and applied.

Sensors can be broadly divided into two types: contact and non-contact. Contact sensors such as switches, probes and feelers must touch an object directly in order to operate. Non-contact sensors can operate at a distance by sensing magnetic fields, sound waves, light, x-rays, infrared light and so on.

Contact sensors may operate by closing an electrical switch, by moving a contact on a potentiometer, or by generating a voltage in a piezoelectric crystal, a change in resistance in a piezoresistive material or some other physical phenomenon. Usually these sensors convert a mechanical movement to an electrical or electronic change. In fact, since all input to a computer must eventually be a digital signal, all of these other signals must be converted to digital codes before being used by a control computer.

Non-contact sensors measure electromagnetic or sonic phenomena; magnetic fields, electric fields, visible light, ultraviolet light, infrared light and x-rays are all electromagnetic phenomena. Ultrasonic distance measurement is based on the time taken by a sound wave to travel from a transceiver to an object and back. The sound wave usually travels at high frequency above the human hearing range.

1.5 Multi-Sensor Systems

Multi-sensor robotic systems are now in use in a large number of application areas. Durrant-Whyte et al ⁽¹⁰⁾, list the advantages of multi-sensor systems (as opposed to single sensor systems) as being two fold. Firstly, using many different types of sensors to obtain information means that the good points of one sensor can be used to compensate for the shortcomings in another. Secondly, by providing redundancy, overall system robustness can be increased⁽¹¹⁾. Multi-sensor systems have the disadvantage, however, that data from each sensor must be merged in order to give a corporate view of the working environment.

The merging of the information provided by multi-sensors should ideally lead to the production of a complete and unambiguous model of a previously uncharted area being produced. Much research has been carried out into the use of multiple sensors to provide a more complete scene analysis than could be obtained by a single sensor. For example, work at the AI Vision Research Unit at the University of Sheffield ⁽¹²⁾ has investigated the use of two cameras to provide robots with stereo vision. The stereo algorithm developed (named PMF after the surname initials of the three authors) enables the information provided by twin cameras to be combined in such a way as to facilitate object range detection as well as object identification.

The Artificial Intelligence Vision Research Unit at Sheffield have also developed a multi-transputer vision system. The system is able to locate objects within a scene by applying 3D geometry techniques to scene information. The 3D scene information required is obtained using binocular image pairs of the scene.

The parallel vision system ⁽¹³⁾ has been produced by implementing TINA ⁽¹⁴⁾ - a sequential machine vision system - on a parallel processing engine named MARVIN⁽¹⁵⁾. MARVIN (Multiprocessor ARchitecture for VIsioN) which is a hybrid parallel machine, hosted by a Sun Workstation, containing 24 T800 transputers and programmed in Parallel C.

Results from Sheffield indicate that combining TINA and MARVIN enables objects to be located in about 10 seconds - this compares favourably with an earlier version of their system which took well over an hour to perform a similar task.

1.6 The Aim of the Project

The aim of this research was to develop in prototype form, a modular sensing system capable of locating and identifying objects within a robot's workspace. In particular, the research sought to:-

- (i) develop a system that contains the facility to add additional sensors without requiring major changes to hardware and software,*
- (ii) investigate the application of parallel processing techniques to improve the overall efficiency of the system.*

The rationale for this research project arose from earlier work carried out as part of an MSc dissertation on robotic sensing systems (16). The aim of the MSc project was to investigate how a low cost vision system might be used to determine the position (that is the orientation and location) of industrial components resting on a factory workbench.

While, on the whole, the system developed as part of the MSc project worked well, it did contain a number of deficiencies. Firstly, components had to be located directly in the sensor's field of view - there was no facility to "search" for components. Secondly, although the algorithms employed performed their task reasonably accurately, they were very slow in execution.

Obviously, these two limitations made the system developed inappropriate for use in most factory situations. In the average manufacturing environment, parts might be missing or lying outside the view of the sensor. In most cases it is also of paramount importance that the positioning of parts be calculated in 'real time', thus allowing the manufacturing process to be carried out as efficiently as possible.

The thesis concluded with several recommendations for future work and it is the investigation based on two of these recommendations that form the basis of this current research. These were:-

- (i) that further research be carried out to establish if the use of parallel processing techniques would improve the overall efficiency of the system.
- (ii) that the system be modularised.

Combining the two techniques of parallel processing and modularity, led to the idea that each module could be implemented independently on a different processor. For example, the algorithm to determine object position could become a "standalone" module while the limitations imposed on component location could be overcome by developing a "find" module. This "find" module would be responsible for locating components within a work area before handing over control to the identification module.

The modularisation idea could be extended so that each sensor had its own control module. This would help overcome one of the main problems that inhibits the widespread use of intelligent robotic systems, that is the complexity of adding additional sensors to a robotic work cell. The majority of sensing systems that are currently available are inflexible in that the addition of extra sensors requires, at the very least, substantial changes to both hardware and software.

In the modular system envisaged the sensors that provide the information about the work cell would be independent of the algorithms that made use of the information. That is, the sensors do not need any knowledge as to when or how the information they provide will be used. Similarly, the algorithms that make use of the data do not need any knowledge as to the provider of the data.

This separation of data provider from data user is similar to that described by Hansen et al (17). It will enable the software that controls the sensors (and even the sensors themselves) to be upgraded without corresponding changes to the data user software. Additional sensors could easily be added to the system while obsolete sensors could simply be removed.

1.7 Summary

If the current rate of deployment of robotic sensing systems into the manufacturing environment is to be maintained, then there needs to be an improvement in both the computational efficiency and computational power available to these systems. This thesis documents the research that has led to the development and implementation, in prototype form, of a modular sensing system serviced by a parallel processing architecture that facilitates the achievement of both the above requirements.

Chapter 2

An Overview of the Prototype System

2.1 Introduction

In order to investigate the viability of a modular sensing system that uses parallel processing techniques, a prototype version has been developed. The prototype allows the location and identification of objects within a robot's workspace to be determined. These two functions are usually a prerequisite to object manipulation and in the prototype have been implemented as two separate modules: (i) an object location module, and (ii) an object identification module. In addition to the implementation of these two modules consideration has also been given to the development of a suitable user interface.

This chapter provides an overview of the prototype system. It points out the need for a suitable data structure to hold a model of the robot's workspace and the reasons why a parallel processing architecture was required to implement the system.

2.2 Description of the Prototype System**2.2.1 Object location**

As a prerequisite to object identification and object manipulation the robot controller must first know where the objects are and in what position they are lying. One way of providing the robot controller with this information is to build a 3-D representation of the robot's workspace (18). This 3-D representation may be constructed from information obtained from various sensors within the workspace. This information may be used to provide "upper bounds" (19, 20) on the locations of the objects within the robot's workspace. The accuracy of this "upper bound" in locating the object depends on the number and nature of the sensors used. While the "upper bound" will normally enclose the object it might not always do so, if noise is introduced into the system via the sensors.

When a suitable data structure had been developed (the development and implementation of the data structure is detailed in chapter 3) the next stage was to demonstrate its application by building a model of a prototype work cell. The prototype work cell, which was rectangular (figure 2.1 shows its dimensions), was made of white plastic so that a good visual contrast existed between the cell and the objects placed within it.

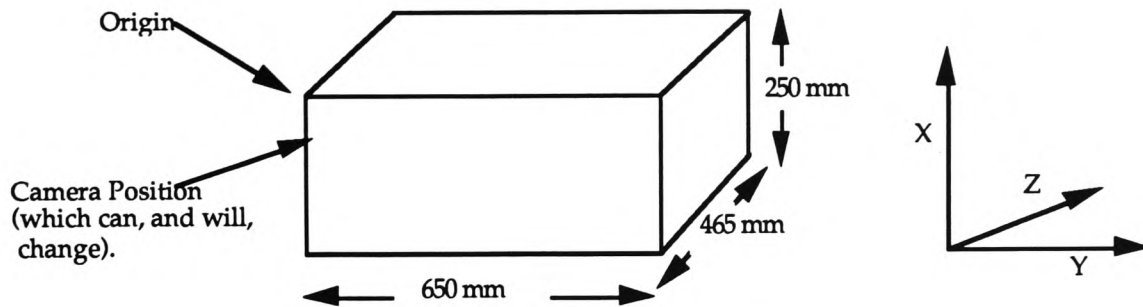


Figure 2.1 - The prototype work cell.

Initially the views of the cell required to produce the model were obtained using a CCD camera mounted on the end of a SCARA type robot arm. The robot arm gave great flexibility in positioning the camera in relation to the work cell, enabling multiple views to be obtained.

Although, in the first instance, only a single sensor was used to obtain the required views, additional sensors were added later. Each time a new sensor was added to the system or a characteristic of an existing sensor was changed (such as the focal length of the camera) the sensor had to be calibrated. (Appendix A3.1 describes the procedure for calibrating a camera. A similar, although obviously not identical, procedure must be developed for each type of sensor.)

Modelling of the Work Cell. To construct this 3-D model, the projection of the 2-D image produced by each sensor through the workspace is calculated (figure 2.2). The dimensions of this projection will depend on the orientation and angle of view of the sensor (appendix A3.2). The composite 3-D model of the robot's workspace is then constructed by taking the intersections of each of the projections (21). Where an intersection occurs, the values associated with the intersection are added to the values already stored in the model, the values being weighted according to the sensor's reliability. For example, the data from a tactile sensor could be regarded as more "reliable" than that from a vision sensor, which in turn could be regarded as more reliable than that from an ultrasonic sensor. Thus, each section in the composite model contains a value representing the 'certainty factor' that the corresponding region of work cell contains part of an object.

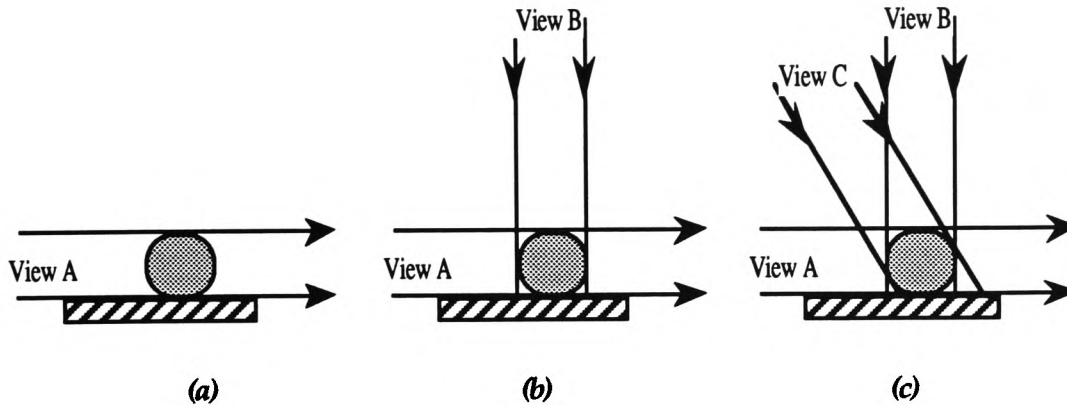


Figure 2.2 - Shows how successive views can be used to determine the enclosing volume of an object resting on a workbench

Using the Model to Locate Objects. When the workspace has been modelled the next step is to use that model to locate the objects within it. The location of an object refers to its coordinates in space. Since an object occupies a finite volume of space, its coordinates would normally be defined relative to a specific point on the object. A correct physical interpretation of location of an object would require the determination of its centre of gravity. However, in this system the model is used to produce three, two-dimensional views of the workspace (figure 2.3). The centroid of each object in the three orthogonal views is calculated and used to give a rough estimate of its location.

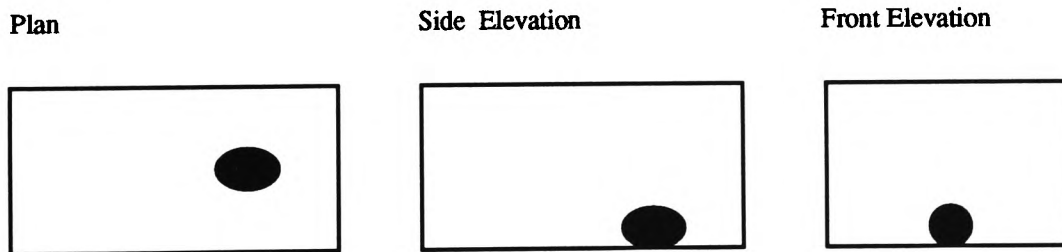


Figure 2.3 - Shows the plan, side elevation and front elevation of a robot's workspace containing a single object.

2.2.2 Object Identification

In some applications the location of an object might be all that is required to enable the robot to manipulate it in the required fashion. For example, the required operation might be to 'move all objects from the workspace and place them into an industrial bin'. For this operation it does not matter what the objects are or in what order they are moved.

In a more realistic application, however, the requirement might be to 'place the round peg into the round hole and then the square peg into the square hole.' For this operation the objects (i.e. the pegs) must not only be located but also identified. Identification is carried out by measuring certain features of the object to be identified and then comparing them with a previously stored database of object feature measurements.

When object features have been compared against the database values some conclusions can be made e.g., 'the object is a round peg' or 'the object cannot be identified.' Depending on the conclusion reached, the operation can continue as appropriate.

2.2.3 User Interface

To enable the system to be used effectively, there has to be a facility for passing information between the operator of the system and the system itself. This communication facility is known as the user interface. The interface has been divided into three layers, each designed for different levels of user operation. At the lowest level, the user is concerned with the hardware and software employed to extract information from the workspace. The middle level deals with the way in which the extracted information is presented to the higher level users. At the highest level the interface is responsible for allowing the user to deal with the extracted information in the required way. The view each user has of the system is, therefore, dependent on what operation he/she is carrying out. This type of user interface has been termed a task-oriented user interface (22).

The most frequent user of the system will be the operator responsible for teaching it to recognise new objects and controlling the location and recognition of objects within the robot's workspace. These functions require no changes to the way information about the objects is obtained from the sensors or presented to the user.

At a different level, and on a less frequent basis, it might become necessary to modify the way in which the information extracted from the sensors is presented to, and subsequently used by, the operator. It is feasible, for example, that a change would be required in the way in which results are presented.

In addition to the two levels of operation already stated, there will be situations where the characteristics of the sensing system will require changing. This will occur when sensors are either removed from, or added to, the system. All three tasks will normally be undertaken at different time intervals and by people with

different skills and knowledge about the system.

It is helpful to list the most common functions of the system along with the frequency at which the task is carried out. The main tasks can be grouped as shown in figure 2.4:-

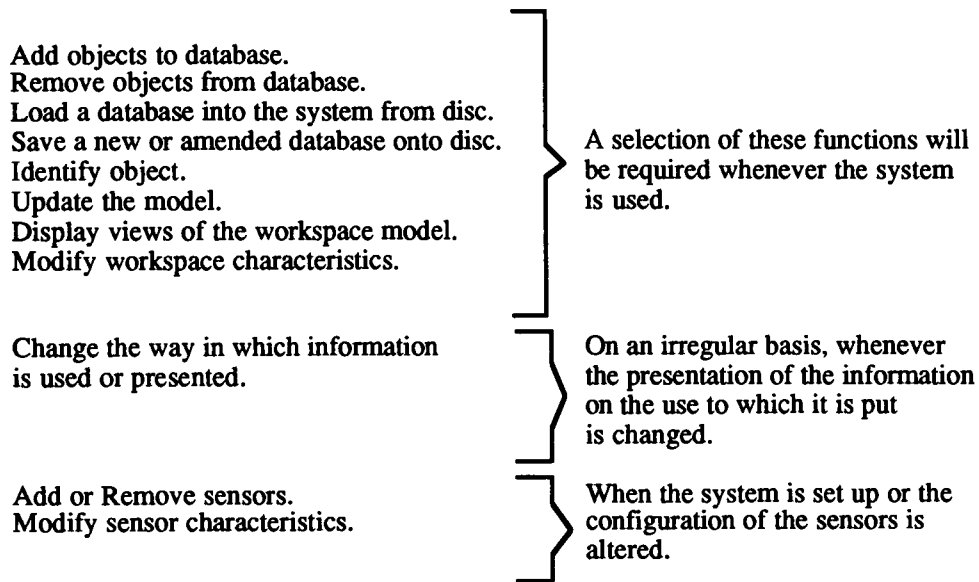


Figure 2.4 - Shows the main tasks that will be performed and the expected frequency.

From figure 2.4 it can be seen that the tasks fall naturally into three layers:-

- (i) **Layer 1** The highest layer provides the interface for the operator, who will probably have no programming or technical expertise. The operator is responsible for teaching the system to recognise new object sets and for controlling the system throughout the working day.
- (ii) **Layer 2** The middle level is where changes can be made in the way information provided by the sensors is used. As an example of the type of changes that can be made at this level consider the following scenario; "A system has been set up to locate and recognise objects from a given object set (which is stored in a database). If an object is located, but is found not to be part of the set, then the object is removed from the workspace and ignored. Management have, however, decided that it would be advantageous to keep track of the frequency with which these unidentified objects have to be removed." The changes to facilitate this amendment would be made at this middle level.

(iii) **Layer 3** The lowest level is where sensors are either added to or removed from the system. If an extra sensor is to be connected then extra processing power will be required to process the data it provides. Each sensor must be connected to the system in such a way that any data it captures is made available to the rest of system in a usable format. Ensuring that communication protocol is adhered to is the responsibility of the layer 3 user.

The interface consists of a series of hierarchical and interconnected menus⁽²³⁾ which allow the user to perform the required operation, or display the required information, or if appropriate list further options.

The menus in the interface have been implemented so as to:-

- 1) optimise the number of times a user has to interface with the system,
- 2) minimise the learning requirements of the user,
- 3) minimise the use of user manuals,
- 4) guide users through the required task,
- 5) ensure that required operations are carried out in the correct order,
- 6) maximise clarity and control.

Menus have been arranged into hierarchical groups which the designer has thought to be sensible, comprehensible and convenient. The wording of the menu options is such that the user should have a clear understanding of what will happen when a given path is taken. However, should a user require the order and wording to be changed then these can easily be implemented via the layer 2 user interface.

The number of options in each menu varies between four and eight. This is consistent with the findings of Miller⁽²⁴⁾. Miller's work examined the way in which users responded to a variety of different hierarchical menu systems. From his experiments he concluded that the best performance was obtained with four to eight options per menu and the lowest error rate with eight selections per menu.

2.3 Processing Requirements

As can be seen from the foregoing description of the prototype system, it is both flexible and expandable as well as being modular. The computer architecture chosen to meet the processing requirements of the system should ideally follow the same pattern. For example, if an additional sensor is added to the system then it should be possible to provide the extra processing power required to deal with

the data provided by the sensor without adversely affecting the rest of the system.

In addition to being flexible, expandable, and modular, the processing facility needs to be powerful. As the robotic work cells to be modelled become more complex (that is they contain more robots connected to more sensors and the tasks they perform become more complex) so the computational requirements of the system will increase. Until the present day, an increase in computational requirement has been met by the development of, and improvements to, VLSI technology. Today's microcomputers have a similar performance to mainframes of a few years ago, but at a fraction of the cost.

It should be noted, however, that this dramatic improvement in the cost to power ratio has been facilitated by an increase in transistor density rather than an increase in raw clock speed, the former having increased by a factor of about 350 times in the last fifteen years while the latter by a factor of about 30 ⁽²⁵⁾. Jones ⁽²⁶⁾ points out that the low-cost, high density transistors of VLSI technology have been used to extend word length to 8, 16 and then 32 bit data paths. The capability of the CPU has also been enhanced by techniques such as microcoding and instruction pipelining, and the addition of specialised functions such as floating point units. While these improvements in single processor technology have been dramatic, the scope for improvement is coming to an end and further substantial improvements are not likely. It is for this reason that attention is now being given to parallel processing architectures.

2.4 Summary

Chapter 2, outlines the function of the two main modules of the sensing system and describes the purpose of the user interface. The chapter also explains the approach to be adopted when modelling the workspace, before concluding with a profile of the powerful, flexible and modular computer architecture needed to implement the system.

Chapter 3

*The Development of an
Efficient Data Structure for
Modelling a 3-D Workspace*

3.1 Introduction

As a prerequisite to object identification, its location within the robot's workspace must be determined. In the context of this project, location refers to the position of objects relative to a predefined point, either inside or outside the robot's workspace. To facilitate this precondition, a computational model of the workspace will first be built and subsequently examined in order to determine the position of the objects. (It should be noted that building a model of the workspace is not the only way to determine the location of objects, but for the purpose of this modular system this approach is considered most suited.)

For the purpose of modelling, the workspace can be considered to consist of a large number of cubic volumes. The most conceptually simple representation for the model would be a three dimensional array, with each cubic volume taking up one entry in the data structure. This simple storage structure would only be suitable, however, when the number of cubic volumes to be stored was reasonably low. If the size of the workspace was large or the size of the cubic volumes were small, then the size of the data structure would be prohibitively large. It is probable that the developed system will be used in situations where either or both of these disqualifying situations apply.

Therefore, for reasons of storage efficiency, a more compact data structure than the three dimensional array is required. At the outset of the research it was envisaged that a hierarchical data structure would provide the required solution.

3.2 Octrees

Hierarchical data structures are becoming increasingly important representation techniques in such areas as computer graphics, image processing, GIS, and robotics. These hierarchical structures are based on the principle of divide and conquer (27). It was to one of these structures, namely the octree (28,29), that consideration was first given when searching for a suitable data structure to represent the workspace.

The octree is a hierarchical data structure which has two principal uses. In the first, it is used to represent a 3-D object, the object's volume being decomposed into progressively smaller parts. In the second, it is used to represent the breakdown of 3-D space into progressively smaller sections, thus making locationally specific data access more efficient (30). In the case of the modular sensing system, it is a 3-D space that requires breaking down, allowing algorithms designed to investigate the workspace to home in on different parts of the work cell.

An octree is a tree data structure. Starting with an upright cubical region of space that contains the object, the space is initially divided into eight smaller cubes called octants, which are labelled 0 - 7 (see figure 3.1). If an octant is completely filled by an object, the corresponding node in the octree is marked black; if it is empty, the node is marked white; and if only partially filled by an object, the node is marked grey. Octants represented by grey nodes are decomposed into eight sub-octants each of which is again tested to determine if it is completely filled, partially filled or empty. The decomposition continues until all octants are either full or empty or until a desired level of resolution is reached. These lowest level octants are called voxels. Those voxels that are only partially filled with an object are approximated as full or empty by some criteria.

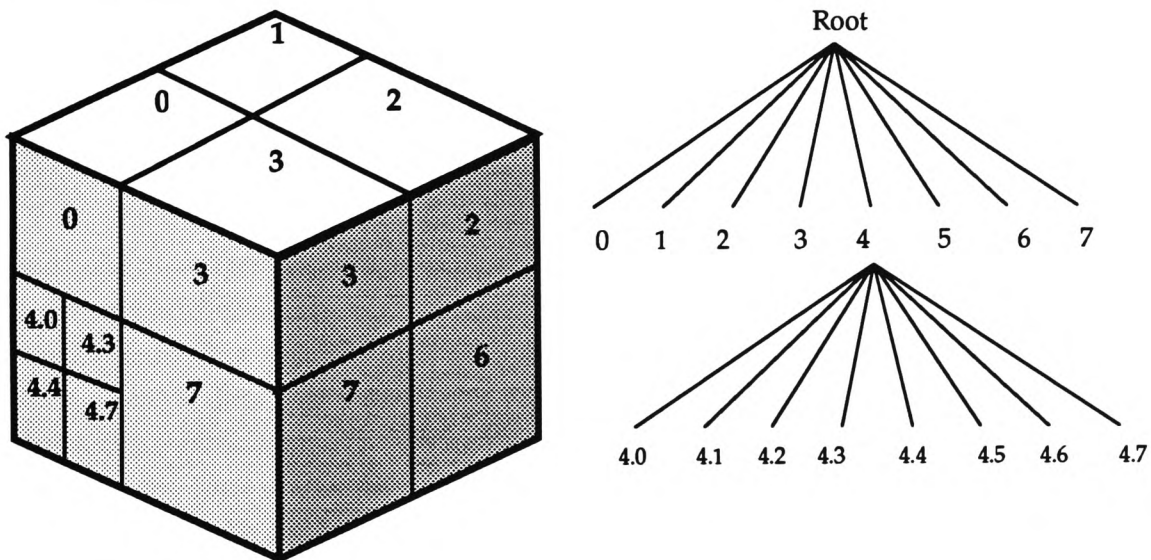


Figure 3.1 - An upright cubical region can be successively broken down into progressively smaller octants until the resulting octant represents a single voxel. As shown this decomposition can be represented as an octree.

The octree, as its name suggests, consists of a root, branches and leaves. As an example of how the octree data structure can be used to represent a workspace, consider the simple example in figure 3.2. The root contains a representation of the whole of the workspace, while its constituent octants are represented by either branches or leaves. At the lowest level of decomposition, voxels, or completely full or completely empty octants are represented as leaves.

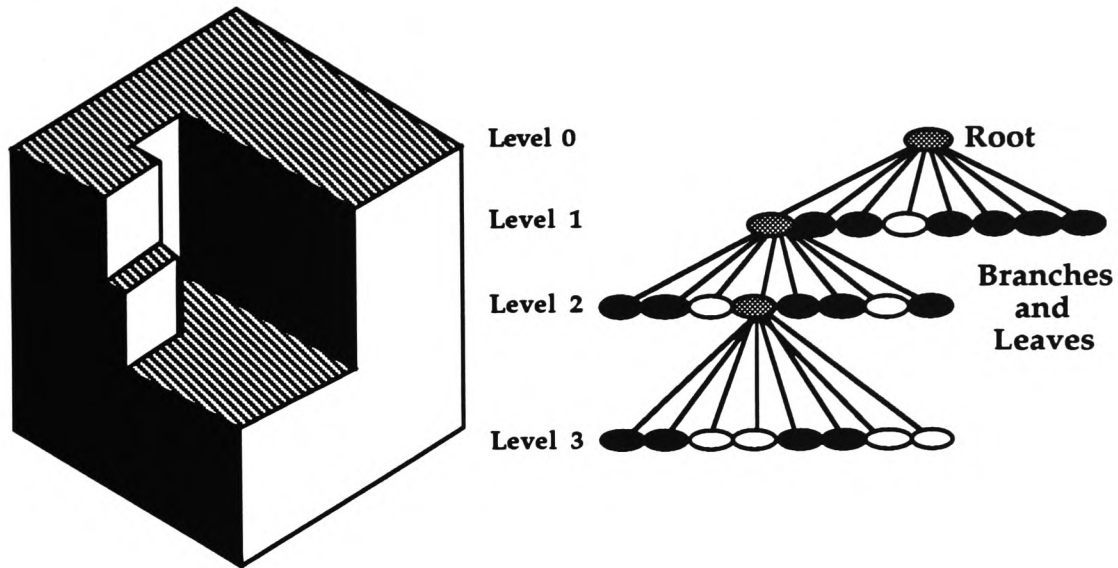


Figure 3.2 - Shows how a 3-D object can be represented by an octree.

3.2.1 Results of Investigations using Standard Octrees

Analysis of the results, obtained from trials carried out to generate standard octrees for several different objects, led to the conclusion that, because of its propensity to require extensive memory and high computation time, the standard octree was not a suitable data structure for the intended robotic application. It was also noticed that a considerable amount of the storage requirement was for the leaf entries - the nodes at the lowest level of decomposition - which suggested that valuable storage savings could be achieved if the standard octree could be modified so that it dispensed with the need for these low order data entries. Investigation into the feasibility of such a modification eventually led to the devising of a revised octree structure which successfully achieved this objective.

(Note:- this new approach is in contrast to that taken by other researchers, who have sought to reduce storage requirements by only storing terminal nodes. For example, the linear octree₍₃₁₎ provides considerable savings in storage, over conventional octrees, by assigning each of the lowest level nodes a unique address. The address relates directly to the position of the node in the octree, thus dispensing with the need for retaining parent nodes. This approach, however, is by its very nature computationally intensive and is thus unsuitable for applications such as the modelling of a robot's workspace.)

3.3 The Modified Octree

Figure 3.3 shows the octree of figure 3.2 together with the octree obtained if the leaves connected to the branch at the penultimate level of decomposition are removed. Comparison of the two octrees shows that the removal of the lowest order leaf nodes has resulted in a saving of 8 nodes, the equivalent of a 32 percent saving.

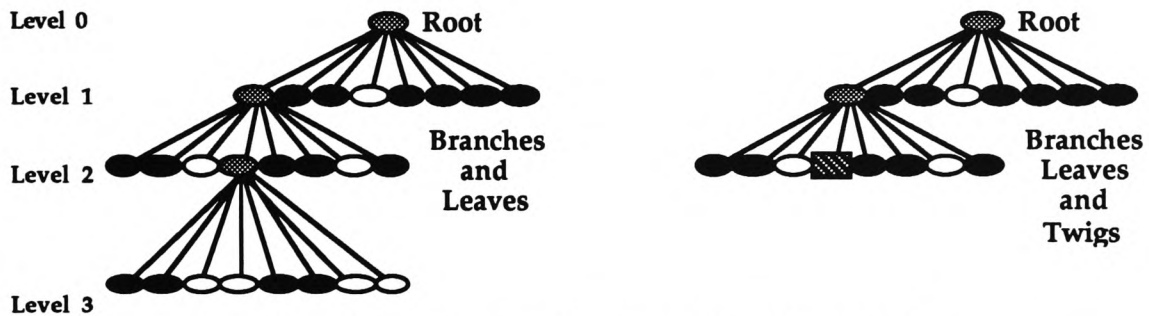


Figure 3.3 - Shows the octree from figure 3.2 and its equivalent modified octree.

As a second example consider the standard octree and its equivalent modified octree, shown in figure 3.4. As can be seen the number of nodes have been reduced from 57 to 25, a saving of 56 percent. As a general rule, if there are 'X' branches in the penultimate level of decomposition then, a saving of '8X' is obtained by the removal of the leaves connected to these branches.

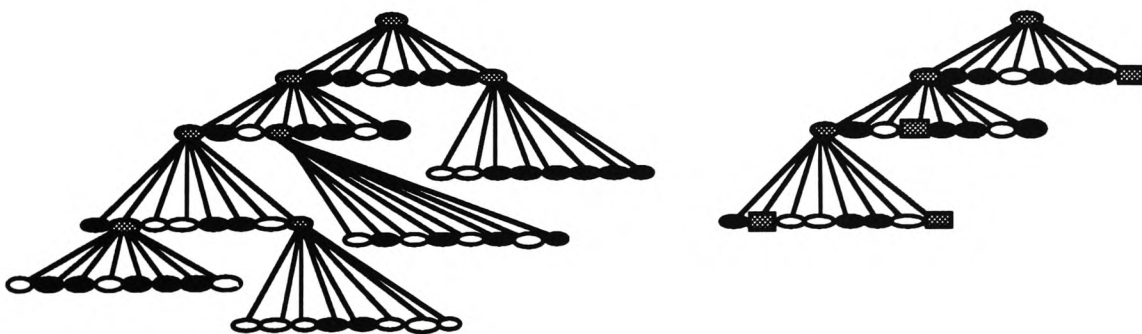


Figure 3.4 - Shows a standard octree and its equivalent modified octree.

The removal of these lowest level leaves has been achieved, with no loss in modelling accuracy, by the development and introduction of a new type entity to compliment the root, branch and leaf entities already described. In keeping with the tree analogy, the author has termed this new entity a 'twig'.

The twig entity replaces the branches in the penultimate level of decomposition, together with its associated leaves, with a single entity. This is achieved by storing information appertaining to the leaves as part of the parent node, rather than the parent node acting as a connector to the leaves. A visual description of this modification is provided by figure 3.5.



Figure 3.5 - Shows how a branch and its connected leaves can be replaced by a single entity.

The modified octree is represented in the computer as a two dimensional array, which has the structure shown in figure 3.6. In order to aid understanding of the structure a description of each of the fields will now be given.

Node	Node Type	Level	Octant (0 -> 7)	Number Set	P(0)	P(1)	P(2)	P(3)	P(4)	P(5)	P(6)	P(7)
------	-----------	-------	-----------------	------------	------	------	------	------	------	------	------	------

Figure 3.6 - Shows the format of the two dimensional array used to hold the modified octree data structure.

NT - Node Type Table entries can either relate to the whole object (a root entry) or to part of an object (a branch or twig entry). The root entry 'R' has a pointer for each of the highest level octants. These pointers are either set to zero if the octants are empty (as in the case of octant 3), or to the table entry associated with that octant. A branch entry 'B' (which represents an octant) has a pointer for each of its constituent lower level octants. Again, as with the root, the pointers to empty octants are set to zero while the others point to the associated lower levels.

At the penultimate level a branch breaks down into leaves as opposed to further branches. These lowest level branches have been termed twigs. A twig has eight values, one for each of its attached leaves, these values are either '1' (leaf is full), or '0' (leaf is empty).

L - Level Each entry in the tree is given a level indicator to indicate its height in the tree. The root branch is at the lowest level and the eight octants that it breaks down into are at level 1. This process of levelling continues until the required resolution is obtained.

O - Octant Number Each octant is given an octant number (in the range 0 to 7) as in figure 3.1, which when combined with the level indicator uniquely defines the octant that the table entry represents.

NS - Number Set In order to facilitate easy geometric calculations each branch table entry contains a count of the number of voxels (lowest level octants) that are full.

p(0) - p(7) Each table entry contains an indicator for each of its eight constituent octants. In the case of a branch entry the indicators act as pointers to the lower level table entries associated with it. In the case of a twig entry the indicators are set to either a one or a zero. If indicator is set to a one then the voxel (the name given to the lowest level octant) which it relates to is full and if the indicator is zero then the voxel is empty.

To demonstrate the use of this structure, table 3.1 shows how the modified octree in figure 3.3 would be represented by the algorithm developed to model the robot's workspace.

Node	Node Type	Level	Octant (0 -> 7)	Number Set	P(0)	P(1)	P(2)	P(3)	P(4)	P(5)	P(6)	P(7)
1	R	0	0	428	2	3	4	0	5	6	7	8
2	B	1	0	44	9	10	0	11	12	13	0	14
3	T	1	1	64	1	1	1	1	1	1	1	1
4	T	1	2	64	1	1	1	1	1	1	1	1
5	T	1	4	64	1	1	1	1	1	1	1	1
6	T	1	5	64	1	1	1	1	1	1	1	1
7	T	1	6	64	1	1	1	1	1	1	1	1
8	T	1	7	64	1	1	1	1	1	1	1	1
9	T	2	0	8	1	1	1	1	1	1	1	1
10	T	2	1	8	1	1	1	1	1	1	1	1
11	T	2	3	4	1	1	0	0	1	1	0	0
12	T	2	4	8	1	1	1	1	1	1	1	1
13	T	2	5	8	1	1	1	1	1	1	1	1
14	T	2	7	8	1	1	1	1	1	1	1	1

Table 3.1 - The octree data structure can be represented in the form of a table. (As an example of how the values in the column headed 'number set' were calculated, consider the entry at node 2, which has 44 lowest level voxels full. The 44 is obtained by summing the values at the next level of decomposition, that is, 8+8+0+4+8+8+0+8.)

As stated, the introduction of a twig entity is a modification of the standard octree data structure which can lead to considerable storage savings. These savings are clearly demonstrated by figure 3.8 and figure 3.9, which show the results obtained when generating an octree for a pyramid whose height and length of base is equal to the height and length of the workspace (figure 3.7). However, as seen from figure 3.10 the computational time (which represent the CPU utilisation when algorithms were run on a VAX 8650) for the modified structure had not been sufficiently reduced to make it viable for a real time modelling system. This meant that unless a further modification to the data structure could be found, that would significantly reduce the computational requirement, then octrees, even in the modified form, would be unsuitable for the intended application.

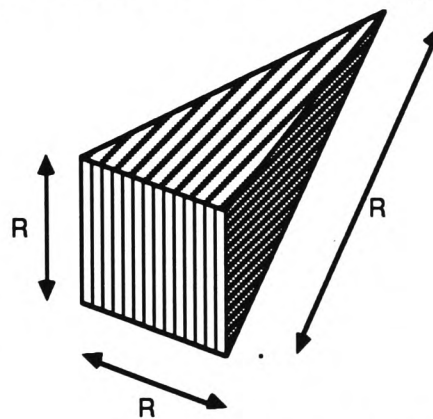


Figure 3.7 - Shows a pyramid whose height and length of base is equal to the height and length of the workspace.

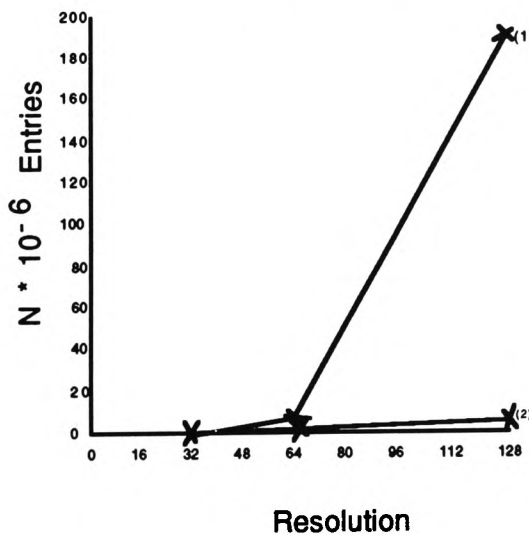


Figure 3.8

- (1) Maximum number of entries in table using standard branch and node approach.
- (2) Maximum number of entries in table using branch and twig approach.

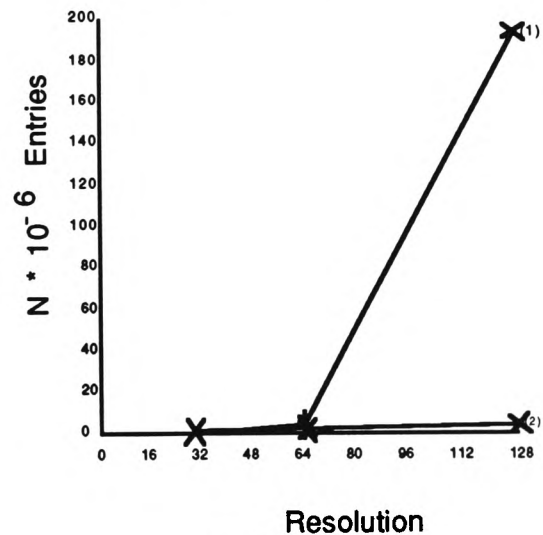


Figure 3.9

- (1) Number of entries in table at end using standard branch and node approach.
- (2) Number of entries in table at end using branch and twig approach.

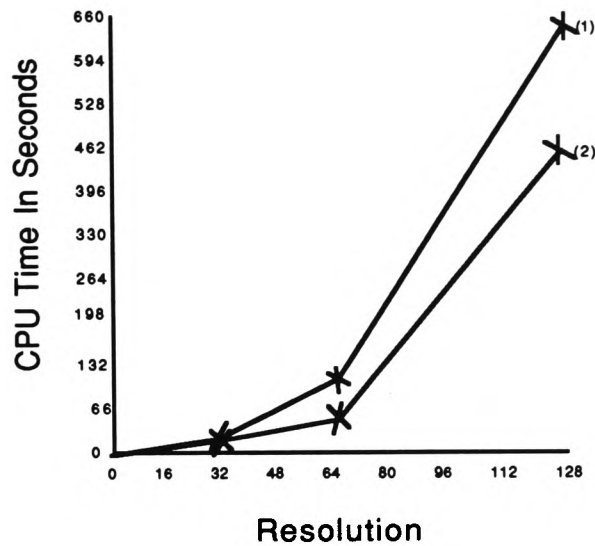


Figure 3.10

- (1) Time taken to build table using standard branch and node approach.
 (2) Time taken to build table using branch and twig approach.

3.4 The Li and Telfer Quadtree Modification

A study of the work carried out by other researchers, revealed that Li and Telfer (32) had produced a modification to the quadtree hierarchical data structure (33) for storing binary images, which give a considerable reduction (approximately 65%) in the number of nodes when compared to the original classical (or traditional) quadtree. This modified structure was also shown to be, in certain circumstances, more computationally efficient than the traditional structure. It was therefore decided, as part of the modular sensing system project, to investigate if a comparable saving could be obtained by applying a similar modification to the octree data structure.

A brief description of the work carried out by Li and Telfer on quadtrees will aid in the understanding of the modification made to the octree. The quadtree is a hierarchical data structure which can be used to represent a 2-D binary image. The structure is formed by recursively dividing the image into progressively smaller quadrants. This process continues until the resulting quadrants are of uniform value (either completely white or completely black). As an example of the quadtree data structure consider the binary image in figure 3.11(a), the image can be broken down into progressively smaller sub-quadrants (similar to 3.11(b)) and then stored as a quadtree (figure 3.12).

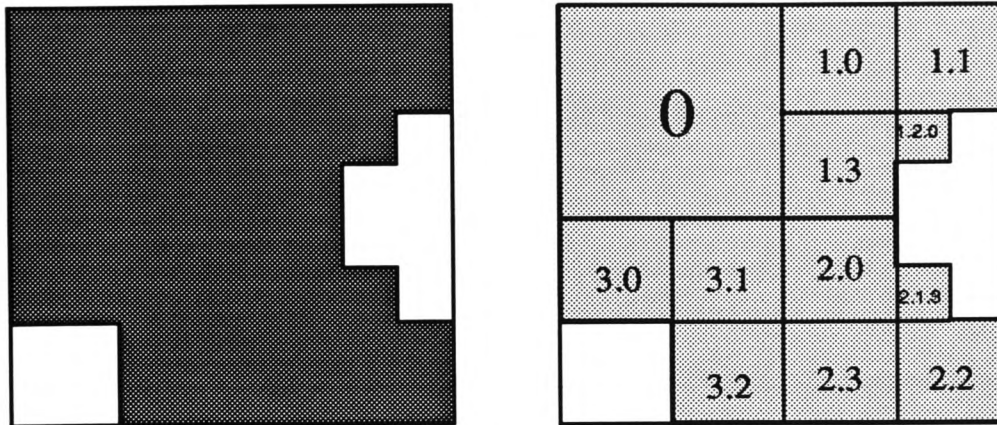


Figure 3.11

(a) A binary image.

(b) A two dimensional area recursively divided into progressively smaller quadrants.

The recursive decomposition of the image will continue until all pixels (that is lowest level quadrants) within a quadrant are homogeneous. In the modified structure implemented by Li and Telfer decomposition continues only until the immediate four sub-quadrants of a quadrant are independently homogeneous (that is each sub-quadrant is either wholly black or wholly white). When this level of decomposition occurs a primitive node is said to have been reached. (If one or more immediate sub-quadrants of a quadrant is not homogeneous then a non-primitive node has been reached, and decomposition continues.) For a binary image there are sixteen different types of primitive nodes (figure 3.13).

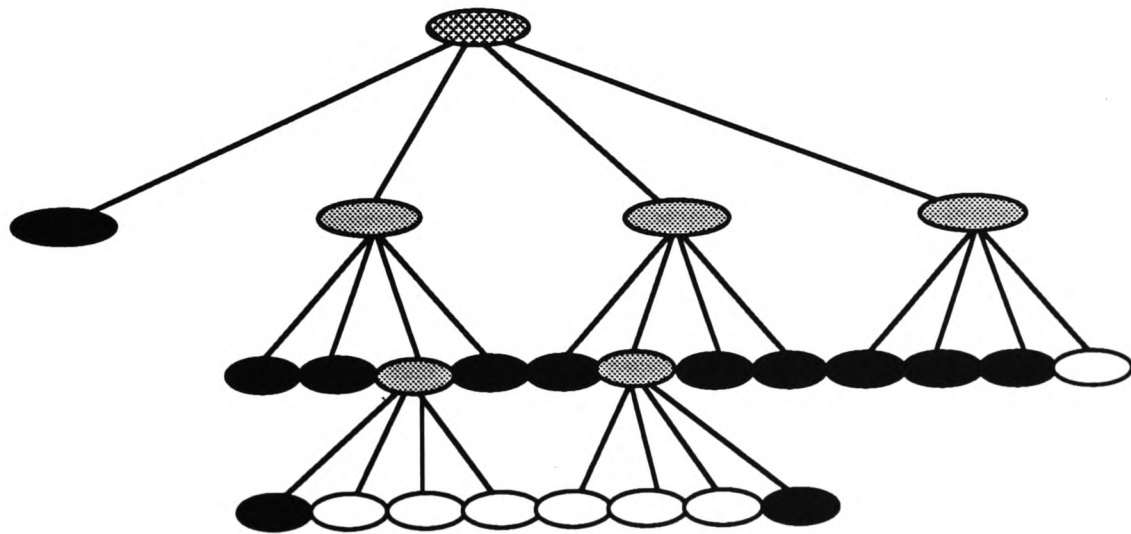


Figure 3.12- The 2D object of figure 3.7(b) can be represented by a quadtree.

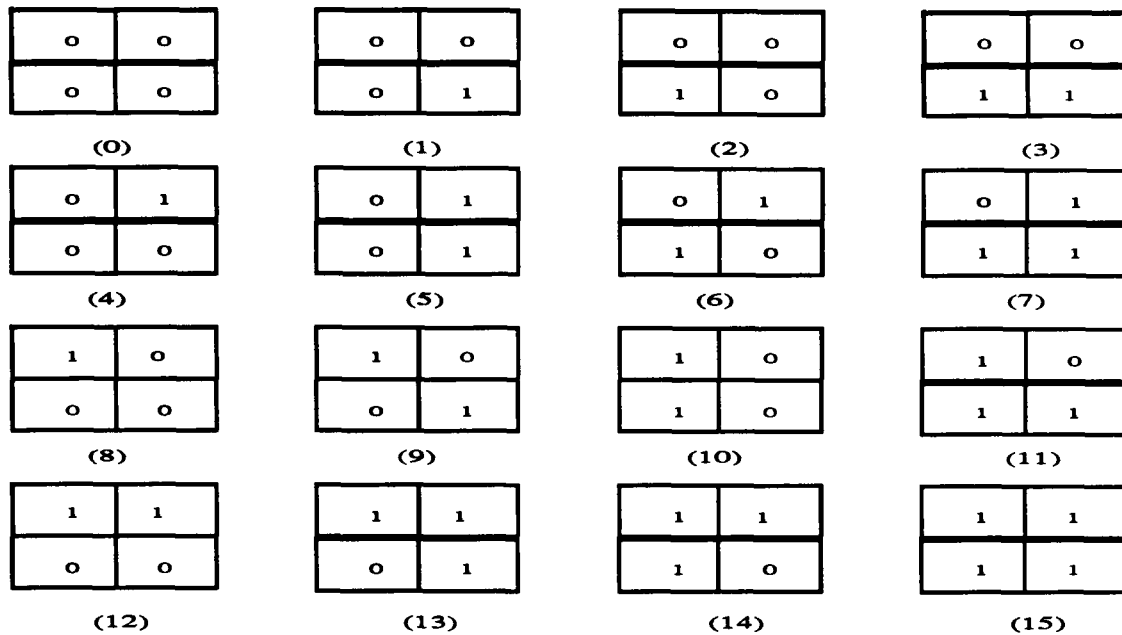


Figure 3.13 - In the Li and Telfer representation there are sixteen primitive node types.

From figure 3.13 it can be seen that each primitive node except for the first and last (code 0 and code 15) correspond to a non-terminal node in the traditional quadtree. In the traditional quadtree the total number of nodes (T) is given by $T=4G+1$ where G is the number of non-terminal nodes. (For example, the quadtree in figure 3.12 has 6 non-terminal nodes and 25 nodes - the 25 can be obtained using the said formula:- $[6*4]+1$.) For a quadtree of a few hundred nodes G is approximately $T/4$ and it is therefore reasonable to assume that the number of nodes in the primitive quadtree to be about a quarter of those in the related traditional quadtree.

3.4.1 The Li and Telfer Modification Applied to Octrees

In a similar scheme to that employed by Li and Telfer for the quadtree, each voxel in the octree can be represented by a single binary digit. Using these binary digits an eight bit binary code can be produced to represent the state of the penultimate level in the octree. There are 256 codes ranging from 00000000 (all octants empty) through to 11111111 (all octants full). These binary codes are the same as the entries stored in the 'twig' pointers. A diagrammatical representation of these codes is shown in figure 3.14(a). These binary codes can also be converted to their denary equivalent and stored at the next highest level as in figure 3.14(b). Using this refinement to the 'branch and twig' approach further storage savings were obtained (typically 20%+). Unfortunately it was found that this increase in storage efficiency could only be achieved by an increase in processing requirements which more than offset the storage gains.

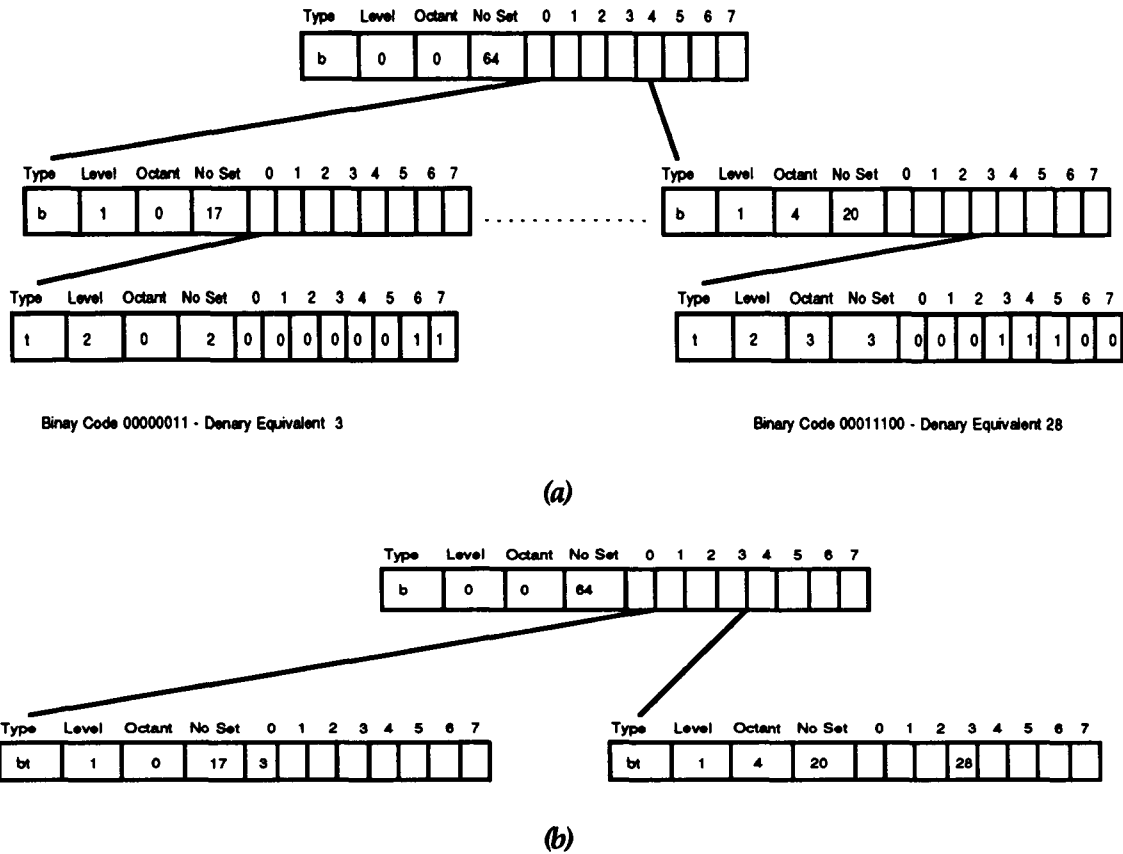


Figure 3.14

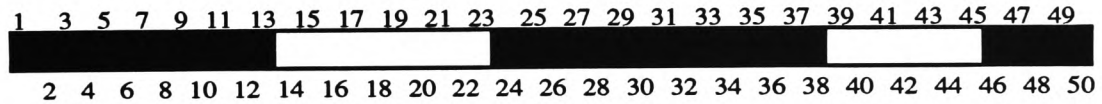
- (a) Shows how the state of the work cell could be recorded without recourse to a table entry for each lowest level voxel.
- (b) Shows how the data structure in figure 3.10(a) could be further reduced by combining the sum of eight binary values into their denary equivalent.

3.5 Run-Length Encoding

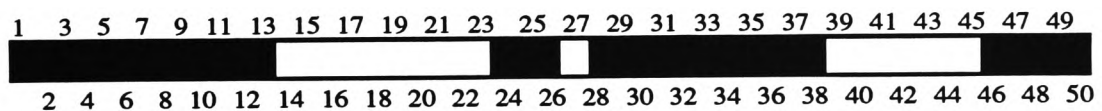
Having concluded that octrees would not provide a viable data structure for modelling the robot's workspace, attention was turned to the possibility of using non-hierarchical structures to achieve the desired objective. One such structure which seemed to hold potential was run-length encoding which is a technique developed for storing data in a compressed form. As an example of its usage, consider how it could be used to store the binary image in figure 3.15(a), where black and white pixels are represented by 1 and 0 respectively. There are three variations of run-length encoding in common use, they are:-

- (1) where the data structure consists of the first location of each run of black pixels followed by the length of the run (alternatively the same could be done for the white pixels). The data structure for the quoted example would be:- 1, 13, 24, 15, 46, 5.

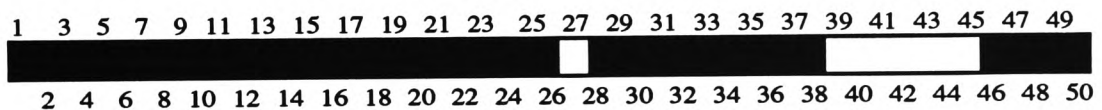
- (2) where the data structure consists of the first and last values of each run of black pixels (alternatively the same could be done for the white pixels). The data structure for the quoted example would be:- 1, 13, 24, 38, 46, 50.
- (3) where the first element in the data structure indicates the start colour of the line (that is either a 1 or 0), followed by the locations where colour changes. The data structure for the quoted example would be:- 1, 14, 24, 39, 46.



(a)



(b)



(c)

Figure 3.15 - Depicts three sets of numbers that can be compressed using run-length encoding techniques.

As can be seen, for the chosen example, the amount of data that has to be stored using run-length encoding is considerably less than if each number had been stored individually. Obviously the saving in storage depends directly on the data set to be stored (the more disjointed the set the less the saving). It is conceivable that using run-length encoding could lead to an increase in storage utilisation (consider the case when the numbers 1, 3, 5, 7, 9, 11, and 13 are to be stored).

As the set of numbers to be stored changes so the data structure will change. Two possible changes are that the structure will become more fragmented (as numbers are removed from the set) and that the structure will become more coalesced (as numbers are added to the set). As an example of fragmentation, consider the new data structure when 27 is deleted from the number set (figure 3.15(b)).

Using technique 1 the data structure now looks like:- 1, 13, 24, 3, 28, 11, 46, 5.

Using technique 2 the data structure now looks like:- 1, 13, 24, 26, 28, 38, 46, 50.

Using technique 3 the data structure now looks like:- 1, 14, 24, 27, 28, 39, 46.

As an example of coalescing, consider the new data structure when the numbers 14 through to 23 are added to the number set (figure 3.15(c)).

Using technique 1 the data structure now looks like:- 1, 26, 28, 10, 46, 5.

Using technique 2 the data structure now looks like:- 1, 26, 28, 38, 46, 50.

Using technique 3 the data structure now looks like:- 1, 27, 28, 39, 46.

As can be seen from the examples, method 3 requires less storage than the other two methods. It does however, require more processing in order to determine whether a pixel is black or white. For this reason, only methods 1 and 2 will be given further consideration.

While the same number of elements have to be stored for both methods 1 and 2, method 2 has two advantages over method 1 which makes it more appropriate for use in the application currently under consideration. These advantages are (i) coalescing and fragmentation of the data structure are easier to handle when the start and finish values of a run of numbers are known; (ii) it is easier to deduce if a voxel is full or empty when the start and finish values of a run of numbers are known. These two advantages are, in reality, very small but serve to tip the balance in favour of method 2.

Martin and Aggarwal⁽¹⁹⁾ have implemented a three dimensional version of run-length encoding which facilitates the modelling of objects. Their data structure is based on splitting objects into "volume segments", each representing a part of the object which is parallel to some coordinate axis. This was achieved by segmenting the objects into planes parallel to the $z=0$ plane, figure 3.16(a). The volume between each z -plane is further split into "lines" (lines are rectilinear parallelepipeds) parallel to the $x=0$ axis, figure 3.16(b). These lines were either completely empty or contained one or more volume segments.

The structure then maintained an ordered pair of values for each volume segment: the values being the y-coordinates of the segment endpoints and were ordered into lists of segments having the same x-coordinate, i.e., colinear. The x-level lists were then coalesced into z-level, ordered lists by common z-coordinate, i.e., coplanar. From the top down this structure was a set of "planes" parallel to the z=0 plane, that were ordered by x-value. Each "line" comprised a set of disjoint segments that were ordered by endpoint y-values. Figure 3.17 shows a schematic of a volume segment structure using linked lists to order the various components.

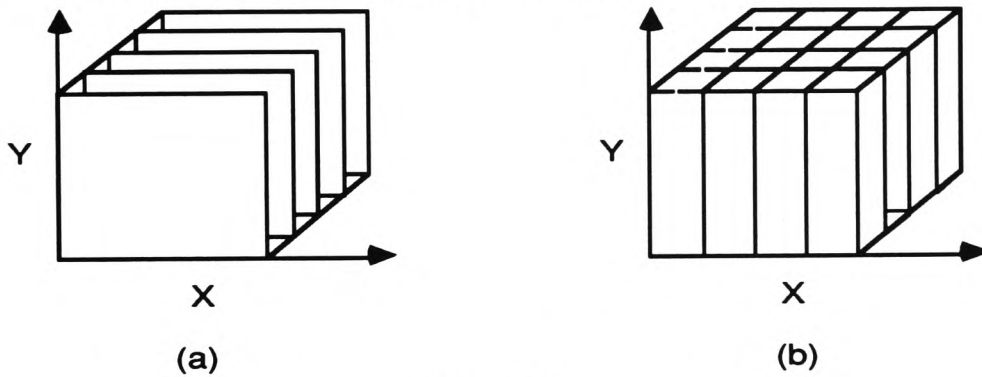


Figure 3.16

(a) Object (or volume) segmented into planes.

(b) Planes split into lines.

In a general situation the primary advantage of this structure is that the process of determining whether an arbitrary point is within the surface boundary consists of a simple search of three ordered lists: select a "plane" by z-coordinate; select a "line" by x-coordinate; and finally, check for inclusion of y-coordinate in a segment.

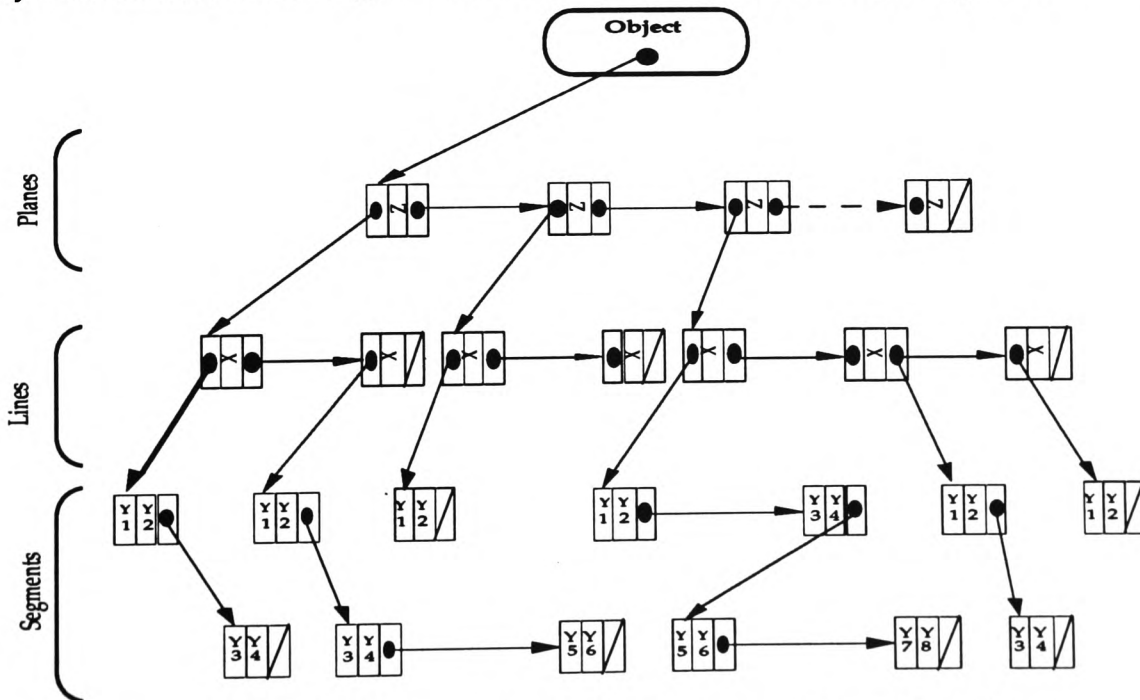


Figure 3.17 - Data structure schema for the volume segment representation.

For the present project, a modification of the structure proposed by Martin and Aggarwal has been developed and implemented. This modified structure has been termed 'partial run-length encoding.' With partial run-length encoding a table is used to represent one of the planes (either x,y or y,z or x,z). Each entry in the table represents a column or row of voxels perpendicular to the chosen plane.

If all the voxels in the column or row are empty then the corresponding entry in the table is zero. If there are any full voxels in the column or row then the corresponding table entry is set to point to the appropriate entry in a second table holding details about full voxels. Each entry in this second table contains details related to non-empty voxels. If two or more adjacent voxels have the same details then only one table entry is required to hold all the relevant information. As an example consider how the object in figure 3.18 could be represented using partial run-length encoding.

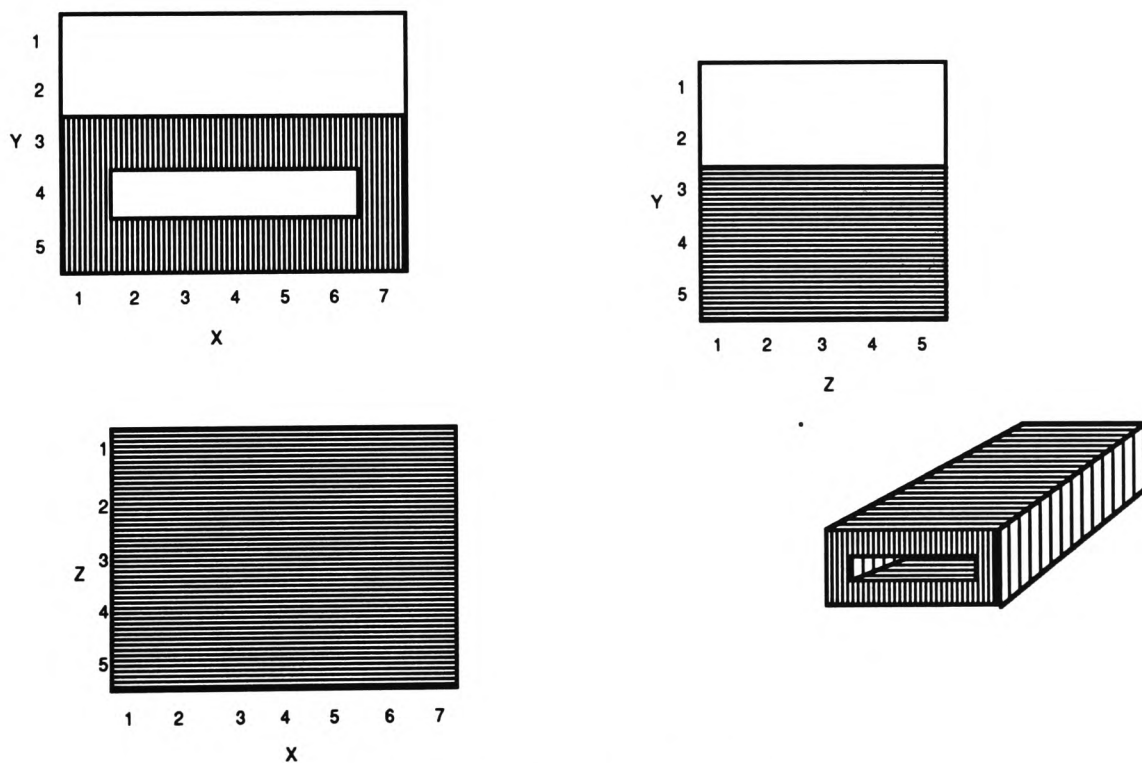


Figure 3.18 - Shows an object and its three end on views.

It can be seen that each voxel in the volume has been labelled using an X, Y, Z, coordinate system. For each non-empty parallelepiped, perpendicular to the Y, Z, plane, an entry has been set in the table that represents the plane (table 3.2). Each non-zero entry in this table acts as a pointer to a location in a second table that holds the run-length data structure for each parallelepiped perpendicular to the Y,

Z, plane. To help clarify the situation consider the parallelepiped perpendicular to the Y, Z, plane where Y=4 and Z=4. In the related parallelepiped the X values set are 1 and 7, that is the voxels Y=4, Z=4, X=1, and Y=4, Z=4, X=7, are full. The entry in table 3.1 for position Y=4, Z=4, points to the first related entry in the second table, that is position 12. From position 12 in the second table there is a forward chain pointer (labelled 'next' in the diagram) to location 13. Location 13 contains the information relating to the second full voxel (Y=4, Z=4, X=7) and a backward chain pointer (labelled 'last' in the diagram) to location 12.

1	0	0	0	0	0
2	0	0	0	0	0
3	1	2	3	4	5
4	6	8	10	12	14
5	16	17	18	19	20
	1	2	3	4	5

Z

Pos	X-Start	X-End	Last	Next
1	1	7	0	0
2	1	7	0	0
3	1	7	0	0
4	1	7	0	0
5	1	7	0	0
6	1	1	0	7
7	7	7	6	0
8	1	1	0	9
9	7	7	8	0
10	1	1	0	11
11	7	7	10	0
12	1	1	0	13
13	7	7	12	0
14	1	1	0	15
15	7	7	14	0
16	1	7	0	0
17	1	7	0	0
18	1	7	0	0
19	1	7	0	0
20	1	7	0	0

Table 3.2 - Represents the y,z plane. Each entry in the table represents a row of voxels perpendicular in the y,z plane.

Table 3.3 - Contains details related to non-empty voxels.

The partial run-length encoding method has been extended to provide automatic production of intersections of the projected views from the various sensors. This has been achieved by extending the second table to include a certainty factor indicating the likelihood that a voxel (or the set of voxels represented by one entry in the table) is empty or full. When information from a sensor is added to the composite 3-D model of the robot's workspace then it is necessary to check for any intersection of the new view with those views already known. If there are intersections then this will lead to the possibility of parts of the partial run-length encoding needing to be coalesced while other parts will become fragmented.

3.6 Conclusion

At the outset of the research it was envisaged that the model of the workspace would be held using a hierarchical data structure and attention was first turned to the octree. It was soon discovered, however, that the amount of storage required was a severe disadvantage. This limitation led to the development of compressed versions of the data structure. Each modification while serving to alleviate one problem, introduced further performance limitations. After much effort, the octree was abandoned in favour of a run-length encoding technique.

The run-length coding data structure developed (which has been termed partial run-length encoding) has proved well suited for the task at hand. As can be seen from figures 3.19 and 3.20 (which were obtained using the object in figure 3.7) the advantage of partial run-length encoding over the octree method, in both computational time and table entries, is considerable.

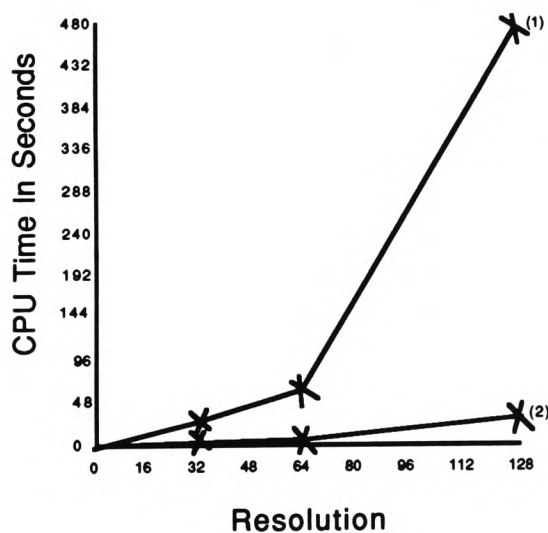


Figure 3.19

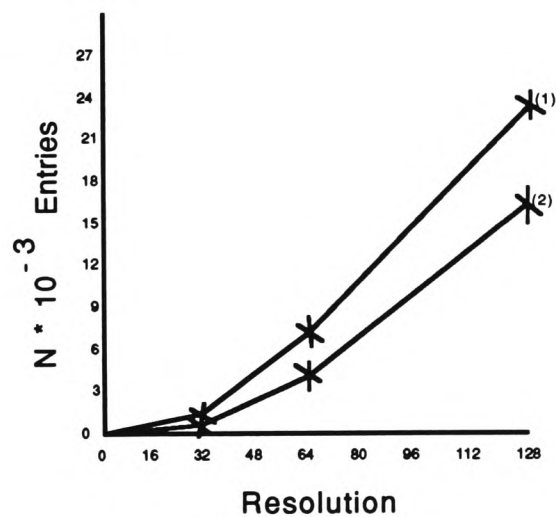


Figure 3.20

- | | |
|---|---|
| <p>(1) Time taken to build table using Octree branch and twig approach.</p> <p>(2) Time taken to build table using Partial run-length encoding.</p> | <p>(1) Table entries using Octree branch and twig approach.</p> <p>(2) Table entries using Partial run-length encoding.</p> |
|---|---|

This superiority is not only in terms of computational and memory requirements but also in the ease with which it lends itself to the representation of non-regular workspace. The timings obtained clearly indicate that a partial run-length encoding data structure provides a more efficient data structure, for the intended application, than the octree.

Chapter 4

Parallel Processing

4.1 Introduction

As indicated in section 2.3, the building of the modular sensing system requires a parallel processing architecture that is flexible, expandable, and modular. There are many parallel architectures in commercial use and the success of the system is largely dependent on the selection of the correct processor. In this chapter, an overview of parallel processing techniques and architectures is provided, before a more detailed description of the transputer, the processor chosen as the basic building block for the system, and its applications are given.

4.2 Parallel Processing Techniques and Architectures

Traditionally computers have consisted of a single processor responsible for performing all the computational work of the system. Programmers divide the task to be performed into a list of steps, called a process, which is then sequentially executed by the processor. The construct of these single processor systems is still based on the design (figure 4.1) proposed by von Neumann nearly 50 years ago.

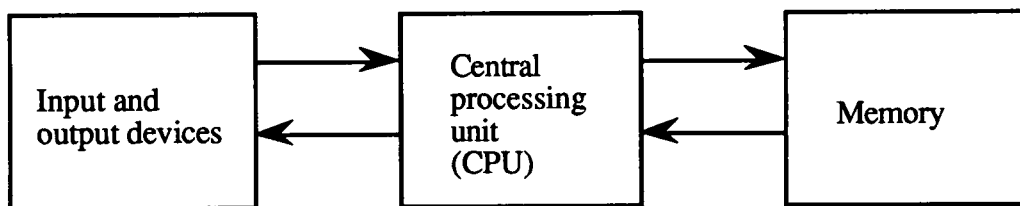


Figure 4.1 -A von Neumann architecture has three major components, these are:

- (i) *the central processing unit (CPU), which performs the basic operations*
- (ii) *the memory, which holds*
 - (a) *the algorithm specifying the operations to be performed;*
 - (b) *the information, or data, upon which the operations are to act;*
- (iii) *the input and output devices (I/O devices), through which the algorithm and then data are fed into memory, and through which the computer communicates the results of its activities.*

Since the time of their inception, von Neumann computer architectures have been in a continuous state of evolution. However, it is only since 1960 that their maturity has been strongly driven by the development of integrated circuit technology. Indeed, as Lee ⁽³⁴⁾ points out "*this relentless technology-push and the corresponding technology-pull, expressed by Grosch's law (viz. for a given implementation technology, processor power is proportional to the square of the price) has stimulated the development of ever increasingly powerful computers*

over four generations of evolution of the von Neumann architecture.

Although the IC technology-push continues unabated, the inherent memory-processor bottleneck of sequential processing is severely impeding further development of von Neumann architectures. Design enhancements, in terms of memory bandwidth and processor power are becoming increasingly cost-ineffective, due to both complexity and physical limitations."

As traditional von Neumann architectures approach the limits of their potential, designers of automated manufacturing systems are turning their attention to parallel processing architectures.

4.2.1 Parallel Processing Architectures

Parallel processing architectures⁽³⁵⁾ can be crudely broken down into two broad classes: "coarse grain" and "fine grain." Coarse grain architectures link a small number of processors, each with a relatively high amount of processor power, while fine grain architectures link a great many low powered processors (an example of such a machine being the connection machine with 65,536 processors⁽³⁶⁾).

As well as level of granularity, parallel architectures can be divided into four classes according to the way they process data. The four classes first listed by Flynn^(37, 38) are single instruction single data (SISD), single instruction multi data (SIMD), multi instruction single data (MISD) and multi instruction multi data (MIMD).

SISD are traditional von Neumann architecture, while SIMD and MISD are array processors and pipe line processors respectively⁽³⁴⁾. The fourth class of architecture, MIMD machines, are made up of arrays of SISD machines executing concurrently. The Flynn taxonomy provides only a crude separation of machines and a more sophisticated analysis is provided by Hockney and Jesshope⁽³⁹⁾ and also Krishnamurthy⁽⁴⁰⁾.

One of the main hindrances to the wider use of parallel processing architectures is the difficulty of detecting the inherent parallelism in problems. This, coupled with a lack of any good, general purpose, automatic parallel code generators (for example compilers that produce parallel executable code, from sequential source code), makes writing a parallel system seem a daunting task.

However, in the case under consideration, where the parallel operation of modules is self evident, parallel techniques would seem to be the best solution as conventional von Neumann computers do not provide the computer power or flexibility to execute the required process quickly and efficiently.

4.2.2 Allocation of Processors to Processes

Parallel processing involves either splitting the process to be performed into several subprocesses and performing them on different processors concurrently (i.e. on a MISD architecture), or splitting the data that is to be processed between a number of processors and executing multiple copies of the process simultaneously (i.e. on a SIMD architecture). For example, Marrow and Perrot⁽⁴¹⁾ have clearly indicated that image processing algorithms can be implemented, on multi processor systems, using both 'image parallelism' and 'task parallelism' techniques. Image parallelism refers to systems where the image is partitioned over the available processors with each processor executing the whole process. While task parallelism means that the process being implemented is split into sub-tasks which are distributed over the processors which then operate simultaneously.

In general, processes can be assigned to processors at one of three stages. These three stages being compile time, load time, and run time. For example, using Occam within the TDS (Appendix 1.1) the configuration of the network to be used must be known at compile time in order to allow the assignment of processors to processes.

However, using other programming environments the assignment of processes to processors can take place at load time. In these cases the loader first sends out a 'worm' that works its way around the network building up a map of the transputers in the network. The loader then calculates the best strategy for allocating processors to processes. In the third case processes are allocated to processors as the program executes. With many programs the logical flow of execution will vary from run-to-run depending on the data input. During one run a set of processes may be executed a large number of times, while in another run the same set of processes may remain dormant. If processors are allocated to processes as and when required, then optimal allocation is possible.

4.3 Transputer Systems

As stated earlier the computer architecture needs to be flexible, extendible and modular. For this reason, the transputer is an ideal building block for such a system⁽⁴²⁾. The name transputer is a composite of the two words **transistor** and **computer**, which reflects its design - that of a processor, memory and communication facility all built onto a single chip. Figure 4.2 shows a schematic diagram of the T800, which is one of the most commonly used transputers.

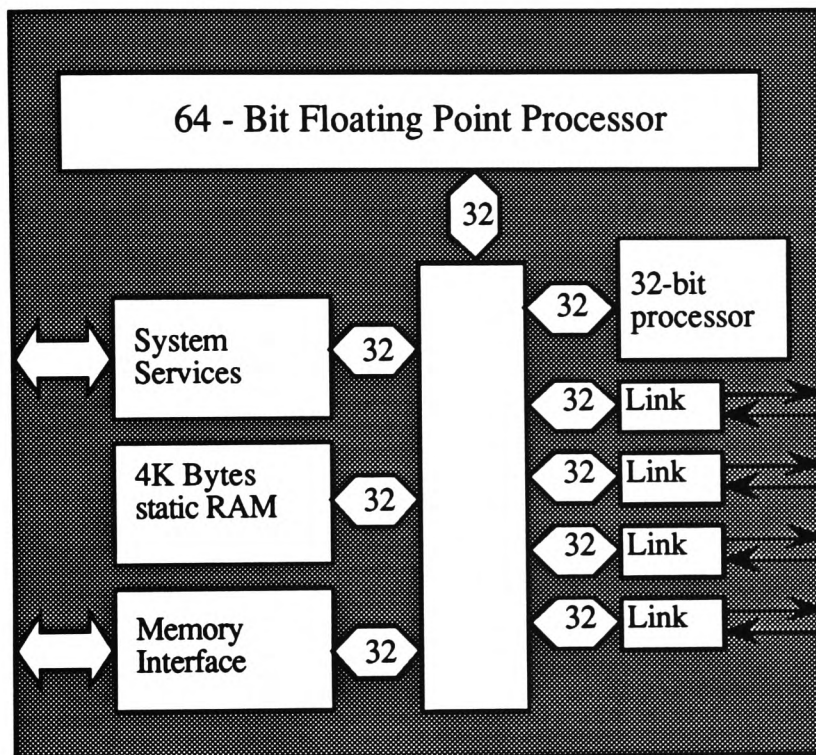


Figure 4.2 - A schematic diagram of the T800 transputer.

Transputers can be programmed as standalone processors (that is as SISD machines) or, as is more usually the case, linked together to form a network of processors (that is a MIMD machine). These networks can be almost any size or shape - the main limitation being that each transputer can only be linked to a maximum of four other transputers. Most transputer networks do, however, have some sort of building block structure - such as the tree (figure 4.3(a)) or mesh (figure 4.3(b)). These type of structures are easier to program and control than those where transputers are arranged in an ad hoc fashion.

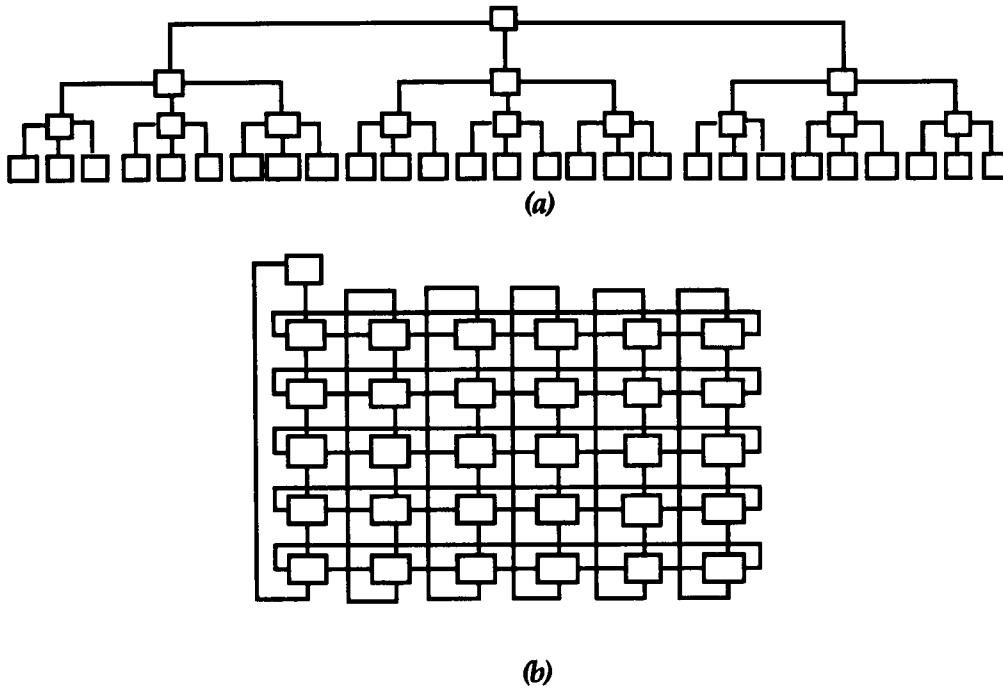


Figure 4.3 - Transputers can be arranged in a variety of ways for example:- (a) a tree, (b) a mesh.

It should be noted, that adding more processors to a network does not necessarily make it faster. The more processors in a system the more inter-processor communication is likely to take place. Figure 4.4, shows what might happen if care is not taken in controlling the amount of communication overhead. (A full and detailed study on designing efficient algorithms for parallel computers is given by Quinn(72).)

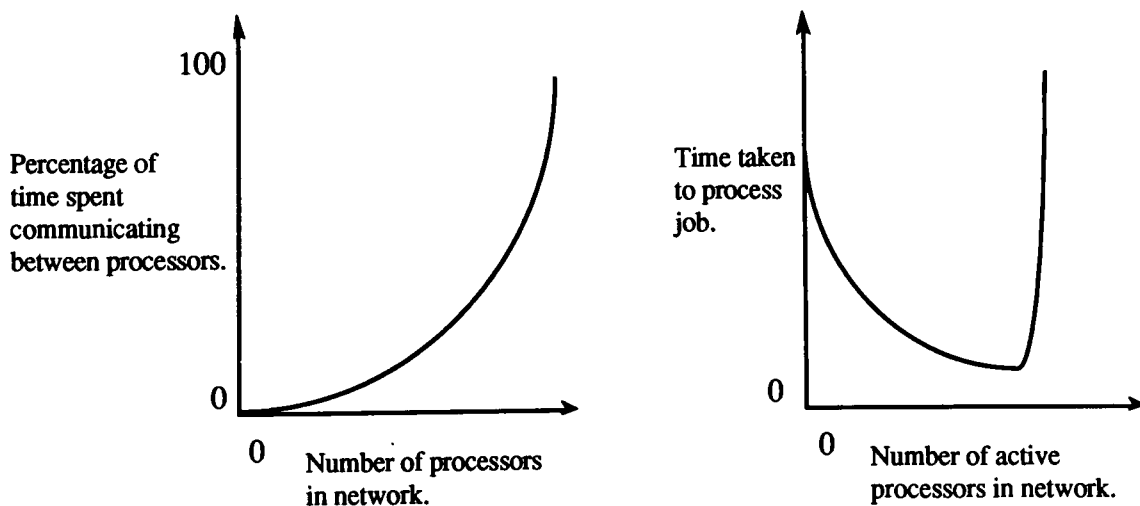


Figure 4.4

(a) As the number of processors in a network increase so does the communication overhead incurred.

(b) There is an optimum number of processors in a network for minimising task turnaround time.

4.4 The Transputer and Occam

Occam⁽⁴³⁾ is the programming language developed at Inmos Ltd. for programming the transputer. The design of the transputer and Occam are intrinsically linked. As Pountain⁽⁴⁴⁾ points out "*the transputer's hardware features (e.g., the on-chip serial links) were chosen to support the Occam model of concurrency, and the instruction set was devised to optimise execution of Occam programs. In turn, the features of the Occam language compiler were shaped by the hardware operations and performance, given the state of VLSI technology in the early 1980's.*"

In Occam, the problems associated with concurrent execution and communication were investigated and overcome at the design stage of the language. This is in contrast to the cases in which parallel versions of sequential languages have been produced.

An Occam program consists of sets of processes which communicate with each other via fixed, unidirectional, and synchronised, communication links called channels. David May, the designer of Occam, was greatly influenced in his design of the Occam channel by the work of Prof. Tony Hoare on Communicating Sequential Processes⁽⁴⁵⁾ (CSP). The basic principle of CSP is that parallel computations can be implemented by a number of purely sequential von Neumann machines communicating with each other over a network of self synchronising communication channels.

Channels can be either 'soft linked' or 'hard linked'. Soft linked channels are used when the two communicating processes are executing concurrently on the same transputer, while with hard linked channels a physical connection has to be made between two communicating processes executing on different transputers. In Occam there is no logical difference between a soft linked channel and a hard linked channel.

As already stated, the design of Occam and the transputer are intrinsically linked. It should not be assumed, however, that Occam is only suitable for programming the transputer. Welch⁽⁴⁶⁾, for example, has provided very strong arguments as to why Occam should be adopted by other areas of the computing fraternity. Welch notes that "*Classical procedural programming languages (like C and FORTRAN) have no foundation in mathematics - but they do have a superficial resemblance that is dangerously misleading. They were devised either before, or with scant regard to, the problems encountered by mathematicians trying to find a formal basis for computing. When the semantics of these languages eventually came to be*

examined, they were found to be incomplete, inconsistent and - worse of all - very complicated. This has serious practical consequences! With no simple and precise understanding of how our basic tools either work or interact with each other, we can have little confidence in the products we make with them."

Despite its superiority, in terms of a solid mathematical foundation, too much money, effort and political muscle have been deployed in placing these more traditional procedural languages into the hands of programmers for Occam to displace them for general use. As Welch states, however, "*one of the important roles of Occam is to show the way out of the mess*" these languages have created. It is to be hoped that future implementations of these languages will incorporate some of the rigorous mathematical foundation of Occam.

4.4.1 Occam Constructs

An Occam process is similar to a statement in a conventional sequential language and can be combined into complex sets of processes using programming structures such as IF, FOR, and WHILE. Processes communicate with each other by writing to named channels. Only one process can write to a given channel and only one process can read from it. In addition to this restriction, a process can only write to a channel when the receiving process is ready to receive along that channel. This type of read and write synchronisation enables communication between two processes to take place without the need for buffering.

The simplest type of process set consists of a series of processes in which each process is executed sequentially. This simplest form of process set is built using an Occam SEQ construct. The SEQ construction implies that the set of processes following it are to be executed sequentially.

```
SEQ
  A := A + 1
  B := B + 2
  C := C + 3
```

A more complex type of process set can be constructed using the Occam PAR statement. PAR specifies that the processes in the following block are to be executed concurrently.

```
PAR
  A := A + 1
  B := B + 2
  C := C + 3
```


With some parallel languages the programmer must take great care to ensure that all processes within the languages equivalent of Occam's PAR construct, are logically independent. In Occam this is taken care of by the compiler, which does not allow a variable to be used more than once in two different process sets executing concurrently. For example, the following set of Occam statements are syntactically incorrect and will be rejected by the compiler:-

```
PAR
  A := A + 1
  A := A + 2
```

By combining the PAR and SEQ constructs more complex process sets can be constructed.

```
SEQ
  PAR
    A := A + 1
    B := B + 2
    C := C + 3
  PAR
    D := A + B
    E := A + C
```

4.4.2 The Suitability of Occam

Occam was intended to be the assembly language of the transputer. If compared to other assembly languages, Occam is very-high level indeed, but compared to some high-level languages it lacks such facilities as complex data structures (records etc.), dynamic memory allocation, and recursion. Occam is a static language in which all resource requirements must be known and allocated at compile time (this means it does not support a stack or a heap).

In addition to these limitations, researchers at Queen's University⁽⁴⁷⁾ have identified three further shortcomings of Occam. First, they note that although each processor in a synchronous array will run a similar program, each must be programmed separately (certainly those on the edge of the array will have to be treated differently). Second, data alignment operations (eg. accessing a neighbour's data) must be coded as explicit I/O operations using Occam's '!' (write to channel) and '?' (read from channel) primitives. Third, any modification to the processor array (eg. addition of more processors for better performance) means that the existing Occam processes must be changed internally.

In view of these limitations, the team at Queen's University designed Latin (a Language for Transputer Interconnected Networks) as an alternative language for

programming synchronous parallelism. While the problems noted by Queen's are obviously very significant and their solution has obvious merits⁽⁴⁸⁾, the system documented in this thesis uses a different approach.

Firstly, the main body of code is kept separate from the communication procedures. Using this approach the main body of code can be run unaltered on any transputer in the network irrespective of its position within the array. Secondly, the alignment operations are performed using the communication procedures described in section 5.5. Finally, the scope of data for which each processor is responsible for is calculated and passed to the processors as parameters at run time. This use of parameters is in contrast to where the scope of each processor is explicitly coded into the algorithm.

In conclusion, the author, while being very aware of Occam's limitations, has none the less found it to be the natural language for programming the transputer. The shortcomings of the language, while causing difficulties for the programmer, are not insurmountable.

4.5 Transputer Projects

While transputers have only a relatively short history their versatility and usability have made them appeal to a wide range of computer users. The first three international conferences on the Applications of Transputers - held in Liverpool in 1989, Southampton in 1990 and Glasgow in 1991, under the auspice of the SERC/DTI Transputer Initiative - have served to highlight the diversity of application areas. It can be noted, from the papers being published, that the transputer technology is reaching a state of maturity - there is a growing trend for development work to be undertaken using languages other than Occam⁽⁴⁹⁾ and programming environments other than the Transputer Development System.

This has been made possible by the development of a number of operating systems^(50,51,52) for multi transputer networks and the implementation of parallel versions of languages originally designed as sequential languages^(53,54,55). A cross fertilisation of ideas is also taking place with advances and ideas from one application area being applied to other application areas. For example the techniques developed as part of this robotic sensing project have been applied to an application in Geographical Information Systems⁽⁵⁶⁾.

Multi transputer systems have been used for many high performance scientific and engineering computer applications^(57,58,59,60). Hey⁽⁶¹⁾ presented an overview of these multi transputer systems as an invited speaker at the ACM International Conference on Supercomputing, held in Amsterdam, in June 1990.

Hey confined himself to those applications that fall into the 'supercomputing' league. (The term 'supercomputing' is used to describe any computer system delivering Cray-1 performance on some given problem.) Hey divides his paper into three sections, they describe the past, present and future use of transputers to form supercomputers. Hey defines a transputer as '*a single VLSI chip integrating processing, memory and communication hardware.*' He notes that the only commercial paradigm at present is the Inmos line of transputers, but points out that Intel have now produced a transputer-like component, the iWarp. Intel have also produced (in collaboration with MIT) a product named the J-chip which can also be categorised as a transputer.

As already noted transputers fall into the class of architecture termed coarse grain. There are however, examples of very large scale transputer topologies. For instance, Parsytec a company based at Aachen in Germany, have plans to build a system which will incorporate 65,000 of the new T9000⁽⁶²⁾ transputers. The system will be capable of performing at the rate of one teraflop (a million million floating point operations per second). It will be so fast, that it might be capable of an accurate weather forecast by modelling an entire world hemisphere - and doing it so fast as not to be overtaken by the weather. Parsytec first designed the 65,000 processor machine, then broke it down into building block of the lowest common denominator: in this case a 64 processor GigaCube. While Inmos have yet to release a working version of this new transputer and Parsytec have only engineering drawings for its system, industry is taking the idea very seriously. For example, Parsytec have already pre-sold six GigaCubes, including two to U.S. defence research centres.

4.6 Summary

The principles of parallel processing are very straightforward. Several processors are linked together, and the workload of an application is shared out amongst all the processors in the network. Not only does such a system offer a great deal of computing power, but its performance can be extended as required, simply by adding more processors to the network.

The workload of an application can be allocated to the processors by either splitting the process to be performed into several subprocesses and performing them on different processors concurrently, or splitting the data that is to be processed between a number of processors and executing multiple copies of the process simultaneously.

There are many parallel processing architectures commercially available, but for the intended application the transputer has been deemed the best suited. A full description of how the transputer has been used in the development of the prototype system is given in chapter 5.

Chapter 5

The Development of a Transputer Network

5.1 Introduction

Following the development of a suitable data structure to hold the model of the workspace (chapter 3) the next step was to implement it on a network of transputers. There are several chips in the transputer family, but the two most important ones are the T414 and T800, both have a rating of 10 MIPS (million instructions per second) at 20 MHz. The T414 has 2K of on-board RAM, whilst its more powerful brother, the T800 has 4K as well as its own on-board floating point unit, which works in parallel with the main CPU and is capable of 1.5 Mflops (million floating point operations per second).

The transputer system developed uses a number of boards each with four transputers and also a number of special image processing transputers (appendix A1.2) manufactured by Quintek. The algorithms employed in the system were coded and tested using the TDS editing environment (appendix A1.1).

5.2 Implementation Principles

To enable processes to run concurrently on several processors, a means of sending data from one processor to another is required. To keep communication overheads to a minimum it is important that this facility sends data via the best possible route around the network and that the number of times a process needs to communicate with other processors is optimised. It is also important to ensure that processor power is used as effectively and efficiently as possible.

To help meet these criteria the following rules were developed, by a process of evolution:-

- 1) **Start the processors doing useful work as soon as possible and keep them doing useful work for as long as possible.**
- 2) **Data transfer between processors should be via the best possible route.**
The best route is not necessarily the shortest as "traffic jams" might develop along certain paths and it would be quicker if these were bypassed.
- 3) **Data transfer should be kept at an optimum level.**
The optimum level of information transfer depends on two factors. Firstly, it may be quicker to duplicate processes on more than one processor than to send information from processor to processor. Secondly, if one processor is generating information to be shared by other processors in the network then

the information can be passed at one of three stages. These three stages being:-

- (i) after all the information has been generated,
- (ii) after a given amount of information has been generated,
- (iii) as and when the information is generated.

While the choice of option will depend on the system being implemented the correct decision is crucial to the efficient execution of parallel processes.

4) When data transfer is required it should be given priority over other tasks.

This enables the receiving transputers to make use of the transferred information as soon as possible.

5.3 Implementation of the Network

After the development of a data structure for storing a 3-D workspace described in chapter 3, it was possible to implement the prototype system outlined in chapter 2. The system developed allows the following functions to be carried out:-

- (1) Update the model. The process of updating the model can be broken down into the following stages:-
 - (a) Obtain a view of the workspace via a sensor.
 - (b) Calculate the projection of the view through the workspace.
 - (c) Add the projection information to the workspace model.

This process may be executed concurrently on any number of transputers; each transputer being connected to a different sensor.

- (2) Use the model to locate objects within the workspace. Once the workspace has been modelled, the next step is to use that model to locate the objects within it.
- (3) Modify the Workspace Characteristics. It is envisaged that the sensing system being developed will be suitable for employment in a large number of different applications. As the application changes, so the characteristics of the robot's workspace will vary. In one application the robot's workspace might be a room while in a second application the robot's workspace might be a workbench. It was therefore necessary to build into the system a means by which changes in the dimensions of the workspace can be reflected in the workspace model.

- (4) Modify Sensor Characteristics. Each time a new sensor is added to the system or a characteristic of an existing sensor is changed the sensor has to be calibrated. After sensor calibration the information obtained is stored for future reference.

5.4 Allocation of Transputers

The model of the prototype work cell (figure 5.1) was held on eight transputers, each storing the model for one eighth of the workspace. A CCD camera connected to the 'Harlequin frame grabber and image processing' board (see appendix A1.2) was used to obtain multiple views of the work cell. The Harlequin, in addition to acting as a camera interface, was used to display the three orthogonal views obtained from the work cell model.

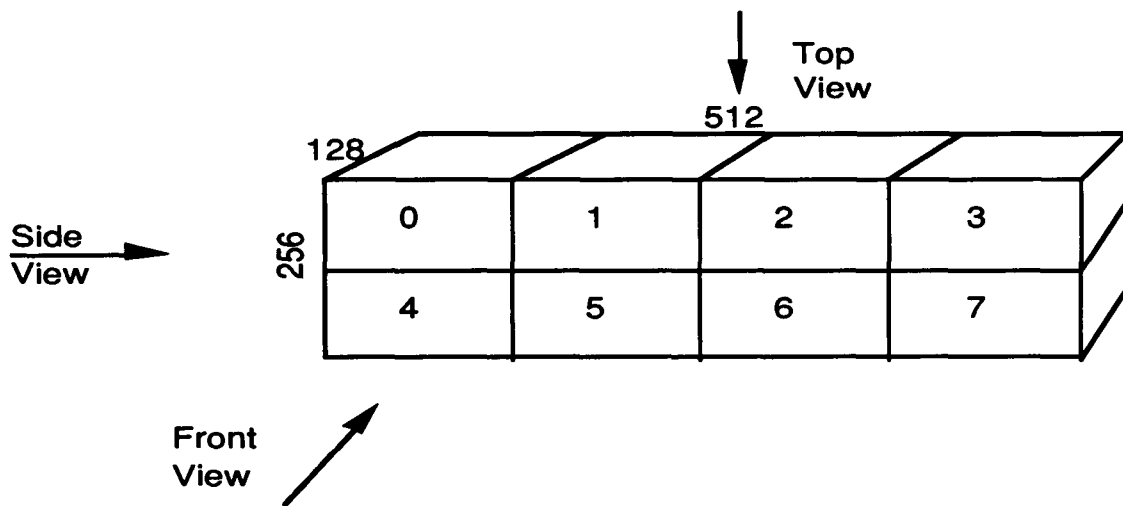


Figure 5.1 -The model of the workspace was held on eight transputers. Each transputer holding a section of 128 by 128 by 128 voxels.

The transputer configuration for the prototype system is illustrated in figure 5.2. The functions described previously are distributed between them in the following manner.

Harlequin

- 1) Obtain an image from the camera.
- 2) Threshold the image and remove noise (see appendix A2.2 and A2.3).
- 3) Project the image through the work cell.
- 4) Build view of the work cell model from the partial views provided by the network transputers and analyse.

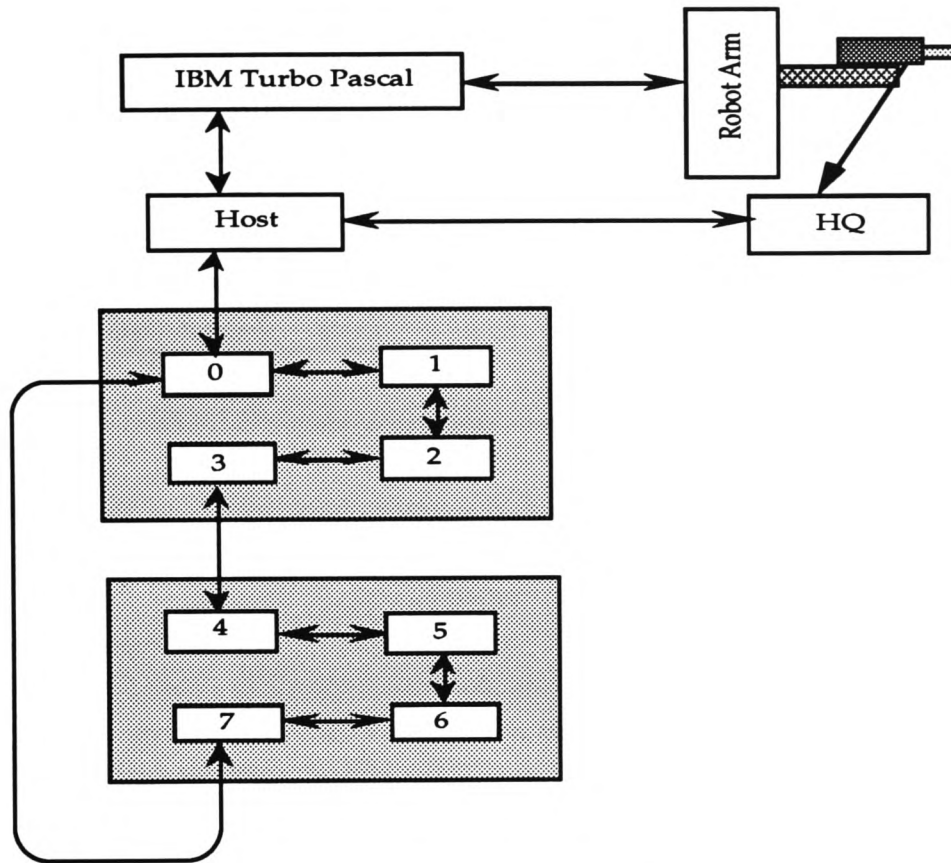


Figure 5.2 - Transputer layout for prototype system.

Host

- 1) Provide the interface between the user and the system.
- 2) Provide the interface between the Harlequin and the network.

Transputers in the network

- 1) Update the data structure with data received from the Harlequin relating to the projection.
- 2) Create a view of the work cell and send it to the Harlequin for analysis.

5.5 Routing of Data Around the Network

To facilitate communication between the transputers in the network each transputer has, in addition to the main algorithm being executed, a set of 'communication procedures.' These communication procedures allow the processor to receive and send data in the required manner.

Data to be sent from one processor to another is packaged into a one-dimensional array. The first element in the array indicates the destination address, the second the source address and the third the quantity of data being sent. The sending processor uses the first two items of information to determine the best route to the required destination. This communication packet is depicted in figure 5.3.

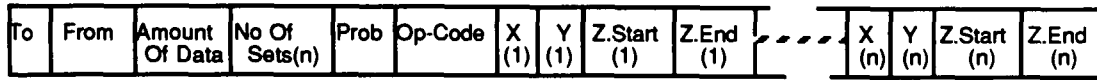


Figure 5.3 - Format for passing data from sensor to model.

Each transputer in the network, therefore, has the following processes executing concurrently:-

- (1) process request.
- (2) a 'get info' process to receive data from both the nearest clockwise and anti-clockwise transputers.
- (3) a multiplexer to send data to the nearest clockwise transputer. This data may have been processed by the sending transputer, or the sending transputer may be acting as a link in the chain. (The multiplexer is required to collect data from the 'get info' and 'image process' routines and pass it on via a single connection.)
- (4) a multiplexer to send data to the nearest anti-clockwise transputer. This data may have been processed by the sending transputer, or the sending transputer may be acting as a link in the chain.

In addition to the above, the first transputer in the network has the facility to communicate with the host transputer. Figure 5.4 and figure 5.5 show the algorithms running concurrently on the different transputers in the network.

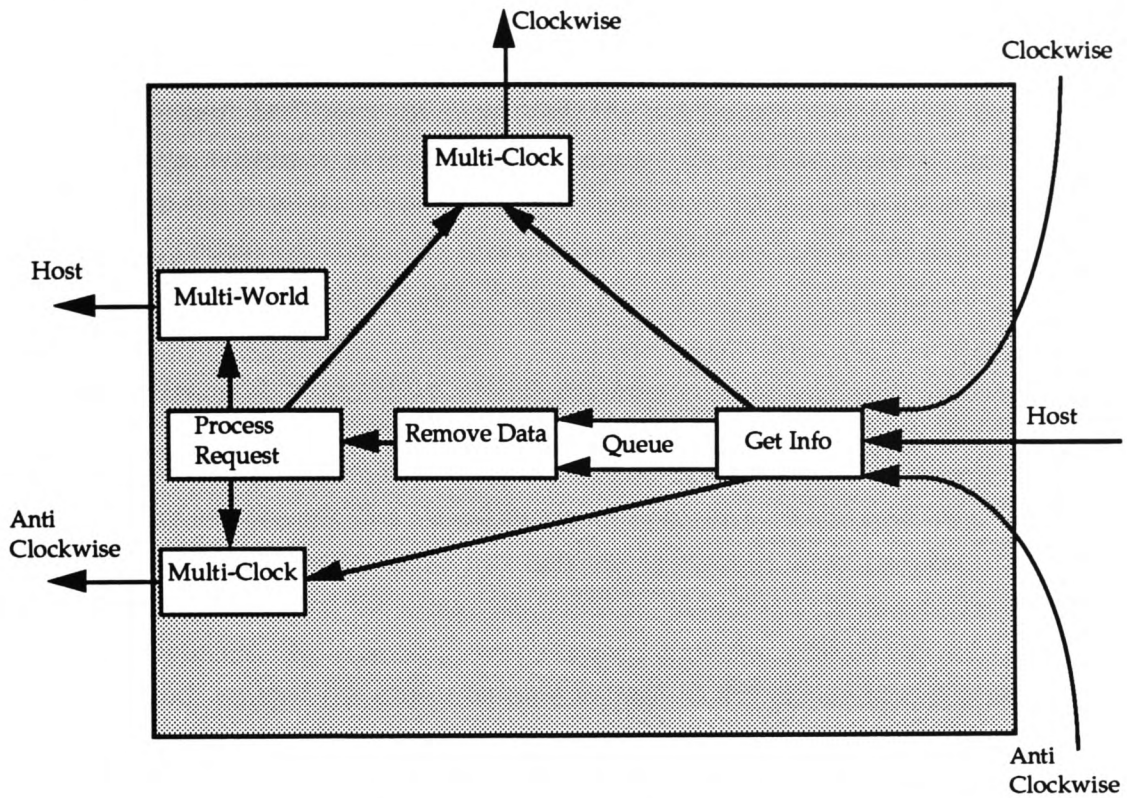


Figure 5.4 - A diagrammatical view of the processes executing concurrently on the first processor in the network.

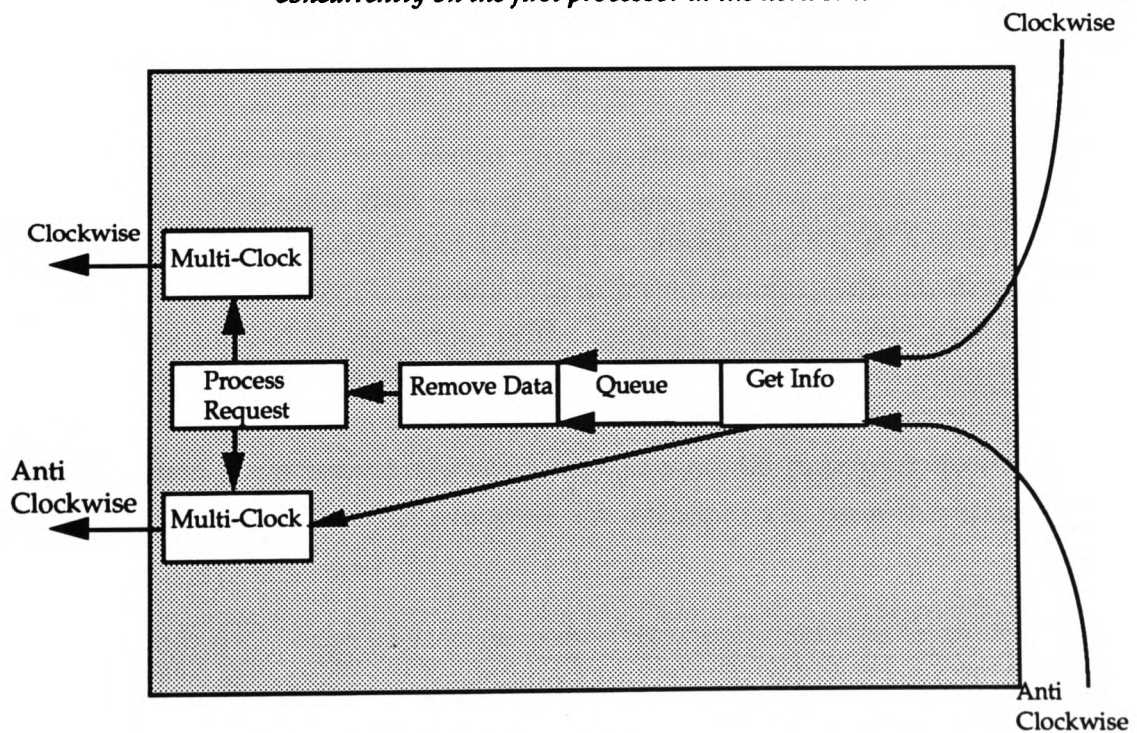


Figure 5.5 - A diagrammatical view of the processes executing concurrently on all but the first processor in network.

5.6 System Testing

After the system had been implemented, the next stage was to test its ability to locate objects within a prototype workspace. To enable testing, the processes executing on the network were extracted. Extraction puts all the code necessary for the loading, communication and execution of processes into a file that is 'bootable' by a program executing on the host PC⁽⁶³⁾.

The procedure for system testing was as follows:-

- (1) A Turbo Pascal program was written to interface the robot arm with the transputer network (the Pascal program executing on the host PC). The Pascal program was responsible for driving the robot arm so that the camera repeatedly scanned the workspace until an object was detected. Once an object was detected the scanning was stopped and the orientation of the camera adjusted so that the object became located in the centre of the camera's field of view.
- (2) The 2-D image was then processed in order to remove noise and its projection through the workspace calculated. The information associated with projection was then packaged before being sent to the transputers that held the associated parts of the model.
- (3) The camera was then raised to a position vertically above the object's location. The robot then moved the camera so that it scanned through an arc between the horizontal and vertical plane until the object was relocated. As with the first view the camera was adjusted automatically so that the object became located in the centre of the camera's field of view. This 2-D image was then processed as before.
- (4) The final stage was to use the workspace model produced, to calculate a rough estimate of object's location. (A description of how this is achieved is given in section 2.2.1.)

5.7 Problems Encountered

5.7.1 Projection Rounding Errors

Due to rounding errors, parts of the projection were given higher certainty values than they warranted while other parts were given lower certainty values than they warranted (figure 5.6). While these errors in certainty values did not lessen the ability of the model to accurately represent the workspace, the data structure became so fragmented that it was inefficient in both memory usage and time taken to traverse from one end of a data structure to the other.

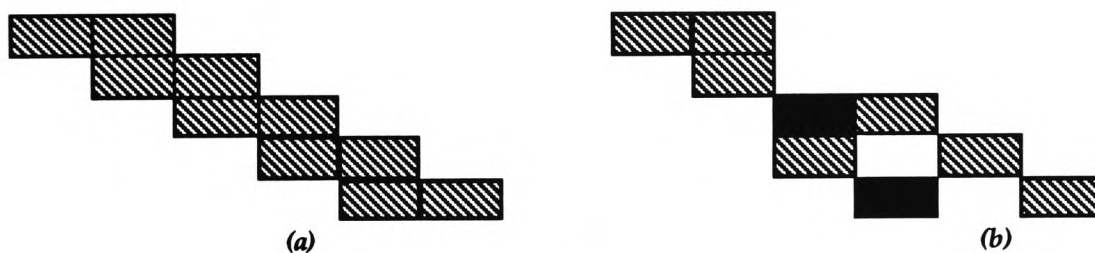


Figure 5.6

(a) *Actual path of two 'parallel lines' projected through the workspace.*

(b) *Calculated path of two 'parallel lines' projected through the workspace.*

In order to overcome the problem of parts of the projection being given higher certainty values than they warranted, it was decided to modify the data structure so that each entry contained an indicator of when it was last updated. This indicator took the form of a view number. If, for example, two views were taken of a workspace then all the entries associated with the first view were given a view number of one, while those from the second view were given a view number of two. If entries in the second view intersected with entries from first view then the view number was updated, from one, to two. In addition, a check was made to ensure that an entry was not updated (due to rounding errors) more than once for any given view.

While the solution to the problem might seem an overkill it has the advantage of being simple to implement and provides a useful piece of information, the view number, for when the workspace is analysed.

To overcome the converse problem, that is where parts of the projection received lower certainty values than they warranted, a low pass filter was used to smooth the values in each region of the model.

5.7.2 Processor Performance Mismatch

It was found that, while the Harlequin was calculating the projection through the workspace (a calculation intensive task), the network transputers were idle for a large proportion of the time. Obviously, to comply with rule 1 of the implementation principles, the calculation of the projection had to be performed more quickly. This required increase in speed was obtained by splitting the projection calculation over more processors. The splitting of work to help balance the workload of each transputer was not, however, without its drawbacks.

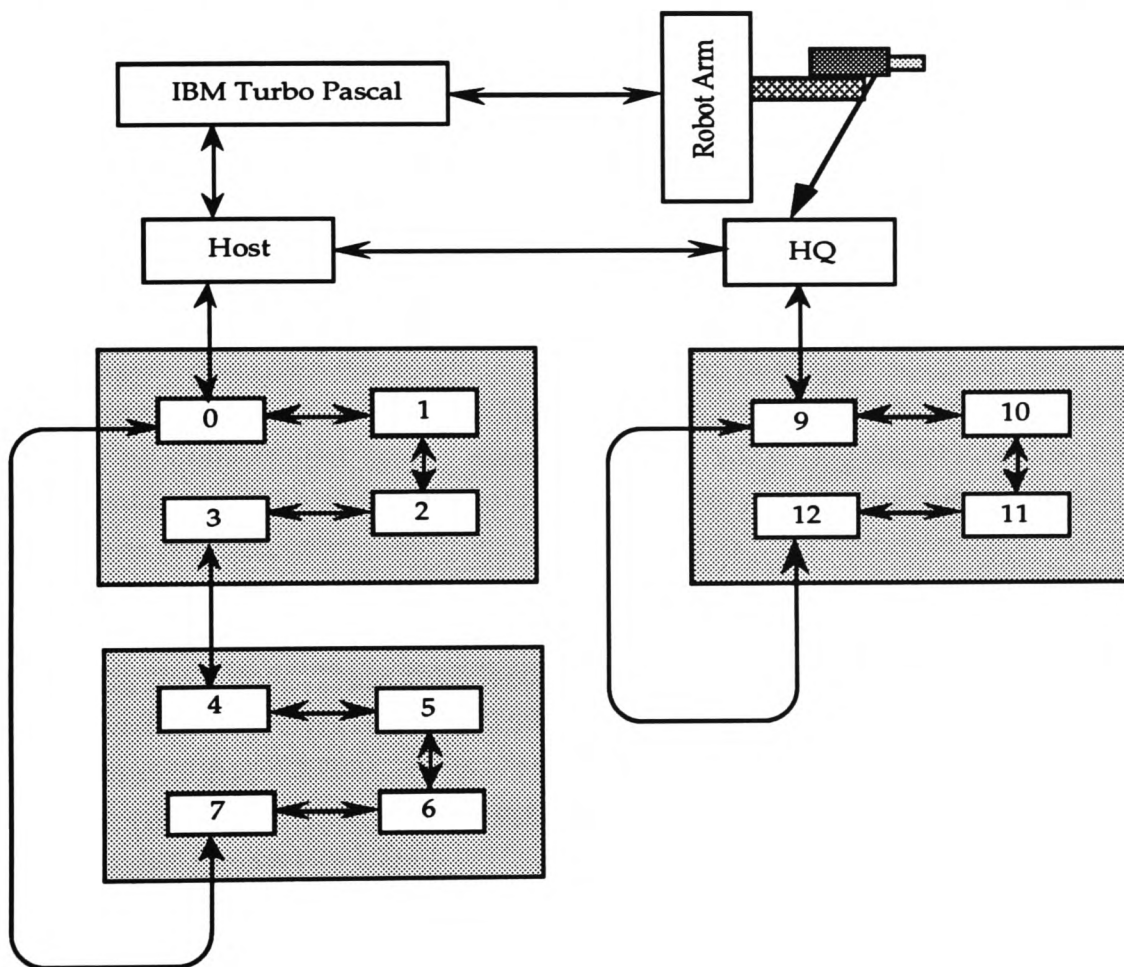


Figure 5.7 - Additional transputers were added to network to balance the work load.

The Harlequin stores its imaged data in byte format whereas the networking system implemented (Appendix A1.3) required data to be packaged and sent in integer format. One solution to this incompatibility of data and packet type is to simply convert all the byte values into integers. This however, due to the number of bytes involved ($512 * 256$ in a typical image) proved to be a slow process. A more satisfactory solution was to rewrite the processes executing on processors nine

through twelve (figure 5.7) to accept and send data packages of the format shown in figure 5.8. If there is no data of type 'byte' to be communicated then only the integer part of packet need be sent (with the 'number of bytes' indicator set to zero).

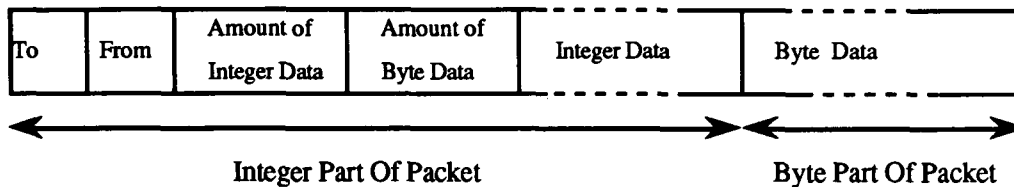


Figure 5.8 - Modified data packet for inter-processor communication.

Using this 'two type packet' to send data between Harlequin and transputers calculating projection the workload of the transputers becomes more balanced. The image clean up (noise removal), however, remained the responsibility of the Harlequin a task for which it is highly suited.

5.8 Summary

This chapter describes the implementation of the prototype modular sensing system on a network of transputers. The transputers, which were arranged to form a series of bidirectional loops, proved to be a suitable architecture for the implementation. The bidirectional loop structure enables the network to be extended as required.

To aid the development of the sensing system, three implementation rules were devised by the author. These rules have subsequently been presented at a number of international conferences, in particular at The Second International Conference on Transputer Applications, at Southampton 1990. As a result of the debate that arose out of these presentations, the author has now rewritten the third rule as two separate rules (see section 5.2).

Chapter 6

Object Identification

6.1 Introduction

Traditional object identification techniques are very situation dependent and are, on the whole, difficult to adapt to changing circumstances. This chapter describes the development and implementation of a novel object identification system which has the capacity to be used in a wide variety of application areas.

6.2 Overview of Object Identification

An object can be described in terms of its characterising features. These features can be categorised into two groups: those that represent the object's position in space and those that describe the geometry of the object.

6.2.1 The Position of Objects in Space.

At any given time an object is at a specific location in space, it is oriented in a specific direction, and it may be moving in a certain direction. The determination of these features represents the most basic level of object identification. Therefore, in order to define fully an object's position, three basic features must be determined: the object's location, its orientation, and its motion. (In the prototype system developed objects were assumed to be stationary and, therefore, their motion did not have to be considered.)

Location. The location of an object refers to its coordinates in space. Since an object occupies a finite volume of space, its coordinates are typically defined relative to a specific point on the object. Two items of information are generally required. Firstly, the distance of the object (at some point on the surface) from the observer must be measured. Secondly, the orientation of the object relative to some reference line must be determined. In a simple visual inspection situation on a production line, all objects should pass through a specified field of view at a constant distance from the observer, and so, therefore, direction can be measured by determining the horizontal and vertical position of each object within the field of view. In this case, it is not necessary to measure distance.

Orientation. An object may be oriented in many different ways without changing its location. Orientation refers to the direction of a specified axis of the object. A simplified approach to specifying orientation is to calculate the axis of least moment of inertia of the image. This is a mathematically determined axis around which, if the object were spinning, it would offer less

resistance to a change in motion than around any other axis. The mathematical calculation of the least moment of inertia is described by Teague⁽⁶⁴⁾. Intuitively this axis of least moment of inertia, is the "lengthwise" axis of a long object.

Motion. A scene may change over time as a result of the motion of objects. There are certain factory situations, such as the avoidance of collisions, when it is useful to determine the speed of an object along with its direction of motion. Motion results in a change in the location and/or orientation over time.

6.2.2 Identification of an Object by its Geometrical Features.

In many application areas it is necessary to know the identity of objects, as well as their position in space. This identification of objects can be achieved by measuring and analysing features of the object which describe its geometry.

When the system has completed the process of analysing object features, some conclusions must be made about the findings, such as the verification that a part is or is not present, the identification of an object based upon recognition of its features, or the establishment that certain parameters of the object fall within acceptable limits. Based upon these conclusions, decisions can be made about the object or the production process. These conclusions are arrived at by comparing the results of the analysis with a previously defined set of standard criteria. These standard criteria describe the expected characteristics of the object, and are developed either through a programmed model of the object or by building an average profile of previously examined objects. The two most commonly used methods of interpreting objects, i.e. feature weighting and template matching, are briefly described below.

Feature Weighting. Where several features of an object must be measured in order to identify it, a simple feature weighting method may be used to consider the relative contribution of each feature to the analysis. For example, in order to identify a valve stem from among a group of stems of several sizes, the image area may not be sufficient by itself to ensure positive identification. The measurement of height may add some additional information, as may the determination of the centroid of the object. Each feature would be compared with a standard for the 'goodness-of-fit' measurement. Features that are known to be the most likely indicators of a match would be more heavily weighted

than others. A weighted total 'goodness-of-fit' score could then be determined to indicate the likelihood that the object has been correctly identified.

Template Matching. In this method, a mask is electronically generated to match a standard model of an object. When the system inspects other objects in an attempt to recognise them, it aligns the model of each object with that of the standard object. The similarity between the actual and standard object is a measure of 'goodness-of-fit'. A threshold value can then be assigned to test for "pass" (positive match) or "reject" (no match). A certainty factor, which presents the degree of confidence that a correct interpretation has been made, is normally calculated along with the "go" or "no go" conclusion.

Variations on these two approaches are used in most commercially available sensing systems. Although conceptually simple, they can give powerful results in a variety of manufacturing applications requiring the identification of objects. However, if objects to be recognised are subject to distortion and variability, the template matching approach is less appropriate. For example, it would be difficult to classify an object with a piece missing without using a very large number of templates. In such cases it is natural to look for features and characteristics which distinguish one object from another.

For this reason the remainder of this chapter will concentrate on a novel feature extraction technique to identify objects within a robot's workspace.

6.3 Overview of the Developed System

Once objects have been located within the robot's workspace (using the method described in chapter 2), the next stage is to recognise what the objects are and in what position they are lying. It is assumed that the objects are part of a known finite set of objects.

As already stated, objects can be described in terms of their features and, thus to facilitate object recognition, a set of describing features is produced for all objects within the object set. In cases in which several object features must be measured in order to identify it, a simple factor weighting method is used to consider the relative contribution of each feature to the analysis. Each feature is compared with a standard for the 'goodness-of-fit' measurement. Features that are known to be the most likely indicators of a match being more heavily weighted than others. A

weighted total 'goodness-of-fit' score is then determined to indicate the likelihood that the object has been correctly identified. The features used in the prototype are:-

- (1) Ratio of perimeter length squared to object area.
- (2) Number of holes in object.
- (3) A ratio calculated using the length of two lines which intersect at right angles at the centre of gravity of object; one line being the axis of least moment of inertia of the object. (See appendix A2.7.)
- (4) Ratio of convex hull perimeter length squared to convex hull area.

It should be noted that other, or additional, features could have been used to enable object identification. The features employed in the prototype system serve only as an example of the features that might be used in a fully-fledged system.

As feature values are extracted from standard test objects, they are added to a set of object features. (Each entry in the object feature set consists of a feature type together with the mean and standard deviation of several measurements of that feature.) As objects within the object set change so will the usefulness of the various features extracted. For example, if the object set contained various non circular shaped assembly pieces then the average degree of roundness of their edges might give a good indication as to their identity. However, in another case, such as a set of various sized disks, then knowing the average degree of roundness of edges would not help in object identification. To allow for this change in feature usefulness, each feature value must have a weighting factor calculated for it. Thus, for each set of objects those features best suited to uniquely identifying objects will have greatest weighting.

The functions allowed by the system are the ability to:-

- i) Add objects to a set. Whenever a new object set is added to the system or additional objects are included in an existing object set, then the feature values used to identify the objects must be determined and recorded in the set of object features.
- ii) Remove objects from a set. If an object no longer forms part of a given object set, then the feature values used to identify the object must be removed from the set of object features. If a whole object set becomes obsolete, then the set of object features can be removed from the system by deleting the relevant DOS file.

- iii) Load a set into the system from disc. To enable the system to identify an object from a given set of objects it must have access to the appropriate set of object features. To allow the system the required access, the DOS file that contains the required set of object features must be loaded into the system.
- iv) Save a new or amended set onto disc. If a new set of object features is added to the system or an existing set of object features changes (as is the case when objects are added to or removed from the set) then the set of object features must be stored to a DOS file.
- v) Identify object. When an attempt is made to identify an object, the feature measurements that describe the object to be identified are matched against the current set of objects. The object from the set of object features that has the greatest overall similarity with the object being identified is selected. If the degree of similarity is above a certain threshold (the threshold will vary from application to application) then a positive match is declared.
- vi) Clear set. In certain circumstances it may be advantageous to be able to clear the current set of object features from the system. This facility has no effect on the DOS file that contains the set of object feature values.

6.4 Details of the Developed System

During the development of an object identification module several 'variations on a theme' were investigated. There follows a brief description of the main units of the system before the variations are considered in detail.

Firstly, the mean and standard deviation of feature measurements from a number of different views are calculated, as in table 6.1. (It should be noted that, for the purpose of obtaining different views, the object is rotated in the x-y plane only, the z axis remaining fixed. If changes are made in the object's orientation, with respect to the z axis, then different mean and standard deviation values will be obtained and additional, corresponding, entries in the table will be required.) Even within the x-y plane, it is possible to place the object in an infinite number of orientations, with respect to the camera, and thus obtain an infinite number of different views. If the true mean and standard deviation were to be calculated for all possible views this would require an infinite number of measurements. Obviously it is thus not practical to calculate the true mean and standard deviation for all possible views. However, if a large enough sample is taken then an accurate estimate of the true mean and standard deviation can be obtained. The measurements obtained

approximate a normal distribution and hence all calculations use these estimated values as a basis for determining probability. (The reasons different values are obtained, when taking measurements from various views, include such phenomena as subtle changes in the lighting, shadowing effects and imaging noise.)

Feature	'Model' Data From Object				Mean	S.D.	
1	14	17	-----	16	21	17.00	0.94
2	2	2	-----	2	2	2.00	0.00
3	2	3	-----	2	3	2.45	0.23
4	12	14	-----	11	12	12.90	0.46

$$x' \text{ (mean)} = \frac{\sum \text{'model' Data From Objects}}{\text{Number of times featured measured}}$$

$$\text{S.D.} = s \text{ where } s^2 = \frac{\sum (x - x')^2}{n - 1}$$

Table 6.1 - The mean and standard deviation for each feature is calculated.

Secondly, the values obtained are added to the table which represents the whole of the object set (table 6.2).

OBJECT	Feature 1		Feature2		Feature 3		Feature 4	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
1	17.00	0.94	2.00	0.00	2.45	0.23	12.90	0.46
2	12.25	1.99	2.00	0.00	17.00	0.69	12.00	0.12
3	9.75	0.86	2.00	0.00	12.00	1.32	6.00	1.20

Table 6.2 - The object set can be represented by a table.

When an object is to be identified, the required describing features are measured and compared with the values in the object table. This comparison enables the degree of similarity between each object in the table and object under consideration to be calculated. Using these similarity values, deductions about the identity of the object can be made. The degree of similarity is calculated by finding the certainty factor that individual features could have come from individual objects (figure

6.3). The sum of these certainty factors is found for each object, with the object having the largest sum being the most likely candidate to match the object under consideration.

Object	Mean	S.D.	Measured	CF
1	17.00	0.94	15	$p_1 = 0.0168$
2	12.25	1.99	15	$p_2 = 0.0836$
3	9.75	0.86	15	$p_3 = 0.0000$

Table 6.3 - The feature's certainty factor is calculated for each object.

The individual certainty factors are given by the probability that the measured value could occur in the normal distribution under consideration. (The probabilities were calculated using the Trapezium rule, as opposed to using lookup-tables which, while being computationally more efficient than calculating the values each time they were needed, required excessive memory space.)

When the system has completed the process of analysing object features, some conclusions must be made about the findings, such as the verification that an object is part of the recognisable object set or is some miscellaneous object. Based upon these conclusions, certain decisions can be made about the object or the production process.

On the one hand, the decision might be that the identified object be manipulated in a given fashion. While on the other hand, the decision arrived at might be to halt production until a miscellaneous object is removed.

6.5 Modifications to the System

6.5.1 Calculation of the Weighted Certainty Factor

A modification to the procedure described in 6.4 applies a weighting to the certainty factors in order to improve the overall performance of the system. For each feature measured, the certainty factor that it relates to a particular object is calculated for each object in turn as earlier. The sum of these certainty factors is then found and used to calculate the weighted certainty factor that the feature has come from each particular object. Finally, the sum of the weighted certainty factors for each object is then calculated, the one with the greatest sum being the most likely candidate for a match.

Object	Mean	S.D.	Measured	Obj CF	Weight CF
1	17.00	0.94	15	p1 = 0.0168	p1/pp = 0.1673
2	12.25	1.99	15	p2 = 0.0836	p2/pp = 0.8327
3	9.75	0.86	15	p3 = 0.0000	p3/pp = 0.0000
				pp = 0.1004	

$$pp = p1+p2+p3$$

Table 6.4 - The feature's weighted certainty factor is calculated for each object.

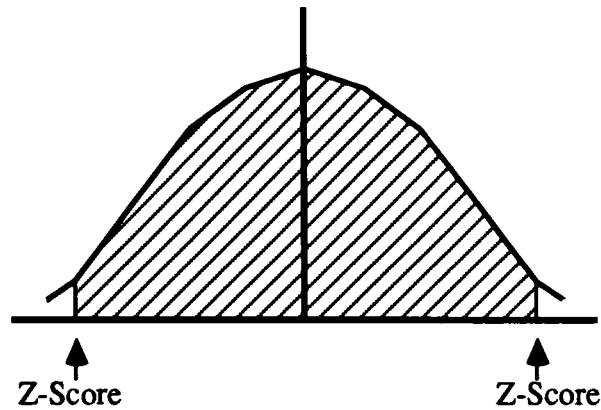
6.5.2 Calculation of the Z-Score

A further modification to the system already described uses z-scores instead of actual certainty factors in an attempt to reduce misclassification. Taking each feature in turn the total feature z-score for all the objects in the object set is calculated.

The z-score is a measure of the number of standard deviations a given value is away from the mean. This value can be either positive or negative. However as the distribution is symmetrical about the mean then the sign can be ignored (figure 6.1).

The percentage contribution each object's z-score makes to the total feature z-score is then calculated. The higher the contribution each z-score makes, to the total, the less likely it is that the feature measurement belongs to the object. Therefore when the percentage contribution is calculated it is subtracted from 100, thus giving the significance value actually used.

It has been found that when high z-scores are included in the calculation, erroneous deductions may result. (This is because other objects with a smaller, and yet significant, z-score are given a higher percentage 'goodness-of-fit' measure than they would otherwise have.) As a high z-score indicates that the object being identified bears little resemblance to the object that produced the z-score it is quite acceptable to exclude the object from any further calculations. The magnitude at which z-scores are considered too high for inclusion in further calculation has been termed the 'z-score threshold value.'



$$\text{Z-Score} = \frac{\text{abs}(x' - x)}{\text{S.D.}} \quad \text{where } x' \text{ and } x \text{ are the mean and actual value.}$$

Figure 6.1 - For each feature the mean and standard deviation of the normal distribution is calculated.

An example will help clarify the procedure. An item that forms part of the object set depicted by table 6.2 (copied from earlier) is to be identified. The four describing features are measured and found to be 12, 2, 16.5 and 12.2 respectively.

OBJECT	Feature 1		Feature2		Feature 3		Feature 4	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
1	17.00	0.94	2.00	0.00	2.45	0.23	12.90	0.46
2	12.25	1.99	2.00	0.00	17.00	0.69	12.00	0.12
3	9.75	0.86	2.00	0.00	12.00	1.32	6.00	1.20

Table 6.2 (copied) - The object set can be represented by a table.

Z-Score for feature 1

$$\text{Object 1 Z-Score} = \frac{\text{abs}(17-12)}{0.94} = 5.32$$

$$\text{Object 2 Z-Score} = \frac{\text{abs}(12.25-12)}{1.99} = 0.13$$

$$\text{Object 3 Z-Score} = \frac{\text{abs}(9.75-12)}{0.86} = 2.62$$

$$\text{Total} = 5.32 + 0.13 + 2.62 = 8.07$$

For each object calculate:- $100 - ((z\text{-score} / \text{Total}) * 100)$

$$\text{Object 1 :- } 100 - ((5.32 / 8.07) * 100) = 34.08$$

$$\text{Object 2 :- } 100 - ((0.13 / 8.07) * 100) = 98.39$$

$$\text{Object 3 :- } 100 - ((2.62 / 8.07) * 100) = 67.53$$

Note that only those features with a z-score less than the threshold value need to have the above calculation performed as it is only they that are included in further calculation. If the threshold was set to '1' (as in table 6.5) then it would not have been necessary to do the calculation for object 1 or object 3. If the threshold was set to '0.5' (as in table 6.6) then even some of the feature measurements for object 2 are not included in the calculations.

The value chosen for the threshold will vary according to the object set and the application. The correct selection of threshold is critical to the whole viability of the system. On the one hand, if the threshold is set too high then objects that are quite obviously not the object under consideration might cause incorrect conclusions to be reached. On the other hand, if the threshold is set too low then object features that should be included in the calculations are discounted too early in the proceedings and again misclassification can arise.

	Object 1	Object 2	Object 3	Total
Feature 1	5.32 0.00	0.13 98.39	2.62 0.00	8.07
Feature 2	0.00 0.00	0.00 0.00	0.00 0.00	0.00
Feature 3	61.09 0.00	0.72 98.89	3.41 0.00	65.22
Feature 4	1.52 81.79	1.69 80.05	5.17 0.00	8.36
Object Total	81.79	277.33	0.00	

Object number 2 which is object2, when threshold is set at S.D.=1.

Table 6.5 - Shows the output produced when an attempt is made to identify the test object with threshold set at S.D.=1.

	Object 1	Object 2	Object 3	Total
Feature 1	5.32 0.00	0.13 98.39	2.62 0.00	8.07
Feature 2	0.00 0.00	0.00 0.00	0.00 0.00	0.00
Feature 3	61.09 0.00	0.72 0.00	3.41 0.00	65.22
Feature 4	1.52 0.00	1.69 0.00	5.17 0.00	8.36
Object Total	0.00	98.39	0.00	

Object number 2 which is object2, when threshold is set at S.D.= 0.5.

Table 6.6 - Shows the output produced when an attempt is made to identify the test object with threshold set at S.D.=0.5.

The idea of using a threshold to eliminate an object, whose feature value has no correlation to the feature measurement being considered, from the identification procedure has also been applied to both the original certainty factor and weighted certainty factor implementations.

6.6 Object Set Classification

In general object sets will fall into one of two categories. The first category being made up of object sets that consist of features that were measured in a scale invariant environment, while the second is made up of those sets that contain features measured in a scale variant environment. A scale invariant set is one in which the features used to describe the objects are independent of the scale of object i.e., whatever the distance between the object and the sensor the describing parameters remain constant. A scale variant set is one in which the feature used to describe objects are dependent of the scale of object i.e., if the distance between the object and the sensor is altered then the describing parameters reflect the change.

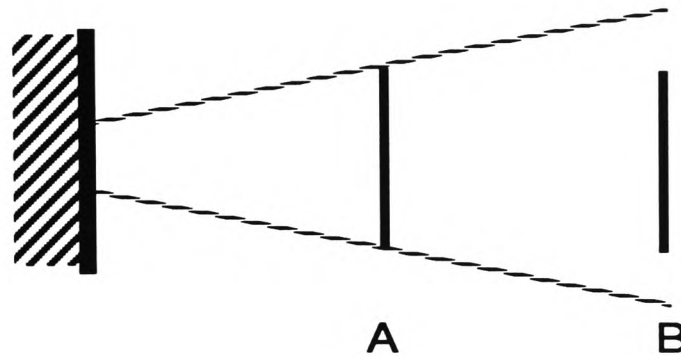


Figure 6.2 - With scale invariant objects the same describing feature values are obtained at both A and B.

It is beneficial to allow for the identification of both scale variant and scale invariant object sets. This enables the same system to be used for an even wider range of applications than was originally envisaged. If an object set consists of objects that are identical, except for their scale, then a scale invariant approach to object identification is inappropriate. (An example of such a set is a batch of variable size pipe washers.)

To enable the system to recognise scale variant objects then the object features used must be modified to make them scale invariant. This is achieved by making the following alterations to the scale invariant parameters:-

Scale Invariant

Object Area
(Perimeter Length)²

Number of Holes in Object

Ratio of two parts of Principal Axis
Ratio of two parts of Secondary Axis

Convex Hull Area
(Convex Hull Perimeter Length)²

Scale Variant

Object Area
Perimeter Length

Number of Holes in Object

Ratio of two parts of Principal Axis
Ratio of two parts of Secondary Axis

Convex Hull Area
Convex Hull Perimeter Length

Obviously if a set of object features is scale variant then the distance between object being identified and camera must be the same as the distance between the camera and the objects when the model data was calculated.

6.7 Discussion and Conclusions

It has been assumed that the measurement used to calculate the mean and standard deviation for object features are normally distributed. This is a reasonable assumption when a large number of images are used to calculate the required values. It will not, however, always be practical for a user to take a large enough sample in order to find an accurate estimate of the true mean and standard deviation (for instance when the object set is large).

In cases where only a small sample can be taken, then the method described will require modification. The assumption made so far is that the sample means are normally distributed (this is sometimes referred to as the central limit theorem), but this depends on the sample considered being large. In situations where only a small sample is taken then there is no justification for assuming that the means are normally distributed.

Statisticians have shown that the means of small samples will show a greater spread than those of larger samples and, although their certainty factor distribution will be similar in shape to a normal distribution, they will be flatter. This type of certainty factor distribution has been named the 't-distribution'. It has also been shown that as the size of samples is increased, so the t-distribution will approximate to the normal distribution. This implies that for each different sample size there will be a different t-distribution. It is thus essential that when calculating a t-score (the t-score is calculated in a similar way to the z-score and used instead of the normal distribution) that the correct t-distribution is used.

It should be noted that when the distribution is normal only the z-score is required to calculate the required certainty factor. This is in contrast to a t-distribution where both the t-score and the sample size must be known.

Chapter 7

Testing of the Object Identification

Module

7.1 Module Testing

When the original identification module and the two subsequent modified versions had been implemented they were tested in order to evaluate their ability to identify objects. The testing, which was carried out for both scale variant and invariant environments, consisted of measuring how accurately objects, from a number of different object sets, could be recognised.

To enable this testing to take place, the system first had to be provided with a set of model object feature measurements. This set consisted of the mean and standard deviation of the measurements obtained when objects were placed in a number of different orientations with respect to the camera. Using this model set of feature values an attempt was then made to identify each object from the set.

7.2 Results of Testing

During testing, when objects from a set were dissimilar, correct identification occurred in almost all cases. An object is said to be dissimilar from other objects in its set if it has at least one feature measurement that is significantly different from the feature measurements of the other objects. For example, while the objects in figure 7.1 are identical except for the number of holes, the number of holes makes each object dissimilar from the others in the set. This dissimilarity facilitates correct object identification.

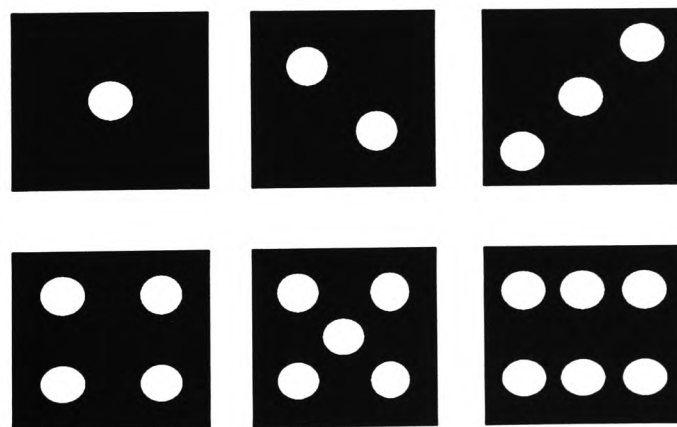


Figure 7.1 - Object set 1.

All three variations of the module provided correct object identification for sets whose members were feature dissimilar. However, further testing proved that when object set members were feature-similar the Z-Score method was superior, in terms of recognition capabilities, than the two probability approaches. This can be

clearly seen from the results obtained for object set 2 (figure 7.2). Although the objects in the set look dissimilar, the measurement shown in table 7.1 and table 7.2, indicate that they are in fact feature similar. (Note: The significant level range at which objects were correctly identified varied very slightly as the feature value set was constructed, tested and then reconstructed and retested. This variation was due to slight changes in the mean and standard deviations recorded because of such phenomena as imaging noise.)

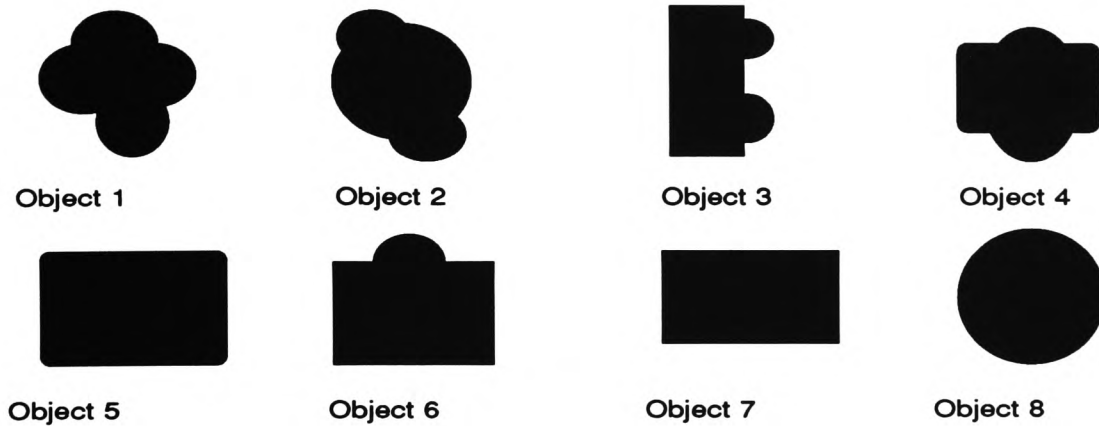


Figure 7.2 - Object set 2.

Object	Feature 1		Feature 2		Feature 3		Feature 4	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
1. Object 1	2.96	0.15	0.00	0.00	10.89	0.51	3.55	0.30
2. Object 2	3.27	0.15	0.00	0.00	10.00	0.48	3.57	0.21
3. Object 3	2.44	0.27	0.00	0.00	6.55	0.48	3.28	0.45
4. Object 4	3.18	0.23	0.00	0.00	10.73	0.58	3.70	0.33
5. Object 5	3.10	0.29	0.00	0.00	9.93	0.36	3.42	0.51
6. Object 6	2.96	0.25	0.00	0.00	8.21	0.41	3.60	0.32
7. Object 7	3.06	0.30	0.00	0.00	9.84	0.49	3.40	0.45
8. Object 8	3.70	0.14	0.00	0.00	10.00	0.42	3.87	0.19

Table 7.1 - Feature table for object set 2, when objects were measured in a scale invariant environment (the features used are those listed in Section 6.3.).
 As can be seen, from the measurements obtained, the objects are feature-similar. This is particularly noticeable for feature 2, the number of holes in the object.

Object	Feature 1		Feature 2		Feature 3		Feature 4	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
1. Object 1	24.03	0.30	0.00	0.00	11.00	0.47	27.45	0.63
2. Object 2	26.91	0.37	0.00	0.00	10.11	0.47	28.69	0.52
3. Object 3	21.50	0.52	0.00	0.00	6.61	0.63	26.49	0.85
4. Object 4	25.65	0.51	0.00	0.00	10.81	0.53	28.65	0.58
5. Object 5	29.65	0.76	0.00	0.00	10.02	0.43	31.52	0.89
6. Object 6	27.23	0.64	0.00	0.00	8.13	0.40	30.51	0.86
7. Object 7	25.43	0.63	0.00	0.00	9.99	0.56	26.94	0.99
8. Object 8	28.58	0.31	0.00	0.00	10.01	0.44	29.18	0.27

Table 7.2 - Feature table for object set 2, when objects were measured in a scale variant environment (the features used are those listed in Section 6.3.).
As can be seen, from the measurements obtained, the objects are feature-similar. This is particularly noticeable for feature 2, the number of holes in the object.

When the model feature measurements had been obtained, an attempt was made to identify each object from the set. This procedure was repeated a number of times, with different threshold values being used to determine if a feature described an object or not. The results of testing are shown in table 7.3 and table 7.4.

The tables show the range of significance levels for which objects, from object set 2, were correctly identified. As can be seen from these results, the ability of the system to recognise objects depends, to a large degree, on the threshold value chosen. Furthermore, there is no single threshold value at which all objects can be identified. This means that if correct object identification is to be facilitated for all objects, then additional features will have to be added to the parameter set.

Object	Z-Score	Probability Method	
		No Weight (S.D.)	Weight (S.D.)
1. Object1	0.55 -> 5.00	0.05 -> 0.70	0.05 -> 0.70
2. Object2	0.95 -> 1.20	1.05 -> 1.15	1.05 -> 1.15
3. Object3	0.10 -> 5.00	1.75 -> 2.05	1.75 -> 2.05
4. Object4	0.15 -> 5.00	0.30 -> 1.85	0.30 -> 1.85
5. Object5	0.10 -> 5.00	1.00 -> 1.15	1.00 -> 1.15
6. Object6	0.05 -> 5.00	0.15 -> 3.20	0.15 -> 3.20
7. Object7	0.15 -> 0.40	0.95 -> 1.00	0.95 -> 1.00
8. Object8	0.05 -> 5.00	0.65 -> 1.65	0.65 -> 1.65

Table 7.3 - Show the range of significant values at which objects from object set 2 were identified correctly in a scale invariant environment.

Object	Z- Score	Probability Method	
		No Weight (S.D.)	Weight (S.D.)
1. Object 1	0.70 -> 5.00	0.05 -> 0.05 0.20 -> 2.25	0.05 -> 0.05 0.20 -> 2.25
2. Object 2	0.60 -> 5.00	0.20 -> 0.50 1.00 -> 1.00	0.20 -> 0.50 1.00 -> 1.25
3. Object 3	1.00 -> 5.00	0.90 -> 5.00	0.90 -> 5.00
4. Object 4	0.10 -> 1.35 1.70 -> 5.00	0.15 -> 0.95	0.15 -> 3.10
5. Object 5	0.05 -> 5.00	0.10 -> 0.50 0.80 -> 3.55	0.10 -> 3.55
6. Object 6	1.20 -> 5.00	0.10 -> 0.10 0.40 -> 3.05	0.10 -> 4.50
7. Object 7	0.30 -> 5.00	0.05 -> 0.15	0.95 -> 1.00
8. Object 8	0.30 -> 5.00	0.65 -> 1.65	0.65 -> 1.65

Table 7.4 - Show the range of significant values at which objects from object set 2 were identified correctly in a scale variant environment.

As can be seen from tables 7.3 and 7.4, if all objects in a set have similar feature values then, the weighting method makes no significant difference to the results as compared with the non-weighted approach.

It should also be noted that for some object sets correct identification of its constituent members can only take place in a scale variant environment. For example, using object set 3 (figure 7.3), it is impossible to identify the objects in a scale invariant environment. In this case the set members are by their very nature scale variant, i.e. the set is one in which the features used to describe objects are dependent on the scale of the objects. If the distance between the object and the sensor is altered then the describing parameters would reflect the change.

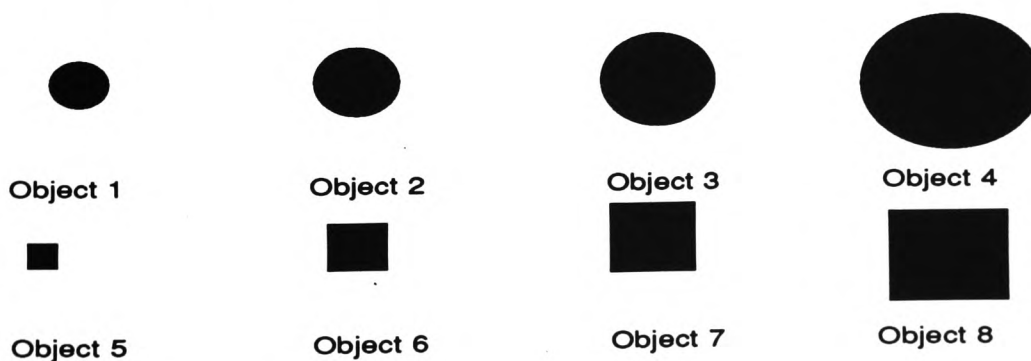


Figure 7.3 - Object set 3.

7.3 Further Testing

In order to allow a more detailed analysis of the differences between the three different module implementations, testing was repeated using a larger object set. The set chosen for the test consisted of the capital letters of the alphabet. The feature measurements obtained when the system was working in a scale variant environment are shown in table 7.5.

Once the feature measurements had been obtained an attempt was made to identify each object from the set. This procedure was repeated a number of times, with different threshold values being used to determine if a feature described an object or not. The results of testing are shown in table 7.6.

Letter	Feature 1		Feature 2		Feature 3		Feature 4	
	Mean	S.D	Mean	S.D	Mean	S.D	Mean	S.D
A	0.95	0.13	1.00	0.00	15.58	0.62	2.74	0.27
B	1.54	0.13	2.00	0.00	9.40	0.49	3.69	0.32
C	0.66	0.11	0.00	0.00	7.75	0.44	3.85	0.27
D	1.33	0.14	1.00	0.00	9.80	0.49	3.54	0.40
E	0.49	0.12	0.00	0.00	9.26	0.56	3.27	0.45
F	0.55	0.12	0.00	0.00	13.30	0.64	3.16	0.21
G	0.54	0.08	0.00	0.00	9.93	0.56	3.70	0.30
H	0.45	0.11	0.00	0.00	10.11	0.45	3.29	0.44
I	0.96	0.18	0.00	0.00	9.77	0.56	2.79	0.47
J	0.72	0.11	0.00	0.00	9.49	0.71	3.22	0.30
K	0.50	0.09	0.00	0.00	9.13	0.47	3.21	0.39
L	0.69	0.14	0.00	0.00	14.29	0.68	3.10	0.33
M	0.38	0.10	0.00	0.00	9.36	0.56	3.19	0.37
N	0.41	0.10	0.00	0.00	9.91	0.41	3.39	0.54
O	1.75	0.21	1.00	0.00	10.08	0.43	3.95	0.23
P	1.11	0.15	1.00	0.00	12.80	0.84	3.47	0.27
Q	1.51	0.13	1.00	0.00	9.84	0.56	3.70	0.27
R	0.80	0.09	1.00	0.00	10.23	0.65	3.46	0.45
S	0.49	0.10	0.00	0.00	9.71	0.50	3.47	0.39
T	0.63	0.10	0.00	0.00	17.10	0.67	3.19	0.39
U	0.53	0.09	0.00	0.00	11.02	0.54	3.38	0.30
V	0.63	0.13	0.00	0.00	14.79	0.74	2.74	0.20
W	0.49	0.36	0.00	0.00	8.62	0.51	2.88	0.28
X	0.57	0.10	0.00	0.00	10.05	0.47	3.38	0.47
Y	0.69	0.12	0.00	0.00	15.88	0.68	3.05	0.26
Z	0.53	0.12	0.00	0.00	9.97	0.54	3.29	0.43

Table 7.5 - Feature table for alphabet when letters were measured in a scale invariant environment (the features used are those listed in Section 6.3.).

Letter	Z Score	Probability Method	
		No Weight (S.D.)	Weight (S.D.)
A	0.05 -> 5.00	0.05 -> 3.45	0.05 -> 1.20
B	0.05 -> 5.00	0.05 -> 5.00	0.05 -> 5.00
C	0.05 -> 5.00	0.05 -> 1.45	0.05 -> 1.45
D	1.05 -> 5.00	0.25 -> 0.40	0.25 -> 0.40
E	0.05 -> 0.80	0.15 -> 0.20	0.15 -> 0.15
F	0.75 -> 2.10	2.35 -> 3.35	2.35 -> 3.35
G	0.05 -> 5.00	1.50 -> 1.55	1.50 -> 1.55
H	0.70 -> 0.70	0.25 -> 0.40	0.25 -> 0.35
I	0.05 -> 5.00	0.60 -> 0.75	0.60 -> 2.60
J	0.50 -> 5.00	0.60 -> 0.90	0.60 -> 0.90
K	0.15 -> 0.20	0.05 -> 0.20	0.05 -> 1.20
L	0.05 -> 0.40	0.55 -> 0.65	0.55 -> 0.65
	0.70 -> 5.00		
M	0.15 -> 0.70	0.05 -> 0.15	0.05 -> 0.05
N	0.10 -> 0.30	0.60 -> 0.70	0.60 -> 0.70
	0.90 -> 5.00		
O	0.05 -> 5.00	0.95 -> 1.10	0.95 -> 1.10
P	0.20 -> 5.00	0.50 -> 3.35	0.50 -> 1.35
Q	0.15 -> 0.15	0.80 -> 0.80	0.80 -> 0.80
	0.25 -> 5.00		
R	0.10 -> 5.00	0.65 -> 3.50	0.65 -> 3.50
S	0.10 -> 5.00	0.60 -> 0.70	0.60 -> 0.70
T	1.15 -> 5.00	0.20 -> 1.90	0.20 -> 1.90
U	0.40 -> 5.00	1.00 -> 2.65	1.00 -> 1.45
V	0.05 -> 5.00	2.35 -> 3.75	2.35 -> 3.75
W	0.45 -> 5.00	0.50 -> 0.55	0.50 -> 0.55
X	0.05 -> 0.30	0.85 -> 0.95	0.85 -> 0.95
Y	0.80 -> 1.55	0.15 -> 1.35	0.15 -> 0.35
Z	0.25 -> 0.60	0.55 -> 0.55	0.55 -> 0.55
	0.95 -> 5.00		

Table 7.6 - Shows the range of significant values at which letters from alphabet were identified correctly in a scale invariant environment.

The results are similar to those obtained for smaller object sets; the Z-Score method being more adept at identifying objects over a wider range of threshold values. (Note: The significant level range at which objects were correctly identified varied very slightly as the feature value set was constructed, tested and then reconstructed and retested. This variation was due to slight changes in the mean and standard deviations recorded because of such phenomena as imaging noise.)

7.4 Conclusions

While, on the whole, the object identification module proved capable of recognising the different members of object sets, some misclassification did occur. The misclassification occurred with objects whose feature values were not significantly different from other objects in their object set. These misclassifications can be minimised by careful selection of the threshold value, while any unresolved problems can be overcome by the addition of further features to the object feature list.

Chapter 8

Conclusions and Future Work

8.1 Introduction

Most sensing systems to date have been designed around a small number of sensors using ad hoc techniques to combine the information the sensors provide into a coherent and integrated format. This thesis has described the research and analysis undertaken in order to develop and implement, in prototype form, a modular sensing system which overcomes many of the difficulties inherent in these more traditional systems. In particular, the research documented has led to the development of the techniques and processes necessary to enable additional sensors to be incorporated into a robot's sensing system without recourse to major changes to either software or hardware.

This concluding chapter summarises both the benefits derived and the conclusions reached from the research, before providing suggestions as to how the work might be progressed in the future.

8.2 Summary

At the outset of the research, its aims were stated as being twofold. Firstly, to develop a system that contains the facility to add additional sensors without requiring major changes to hardware and software. This has been achieved by the modularisation of the system. The second objective, to investigate the application of parallel processing techniques to improve the overall efficiency of the system, has led to a system that is intrinsically parallel. The modularity of the system is such that it is now difficult to see how the system could have been implemented using anything other than a parallel processing architecture.

The research documented in this thesis has not only demonstrated the benefits of parallel processing techniques over sequential systems but has also brought to light the advantages of a modular sensing system as opposed to conventional systems. These advantages can be summarised under the following broad headings.

(i) Advantages of using a parallel processing architecture.

- 1) Processing power can be applied to the task at hand as and when needed.
- 2) The need to try and fit tasks that are by their very nature parallel into a sequential ordering is eliminated.

(ii) Advantages of a modularised system.

- 1) Divide and conquer techniques can be applied.
- 2) Modifications can be confined to individual modules.
- 3) A library of building blocks can be established.
- 4) The financial outlay can be spread over a longer period.
- 5) A multi-layered (or multi-faceted) user interface can be constructed.

In the following subsections these advantages are first expanded upon before the benefits of the research to other non related subject areas are given.

8.2.1 Advantages of using a Parallel Processing Architecture

- 1) Processing power can be applied to the task at hand as and when needed. The topology of the network implemented means that additional transputers can be easily added to any of the bidirectional loop subsystems as required. This means that if a particular task was slowing down the operations of the rest of the system, additional transputers could be employed to speed up the task.

It should be noted however, that should any of the bidirectional loops become too large then performance degradation will occur. This degradation will occur because the transputers in the loop would have to spend a larger percentage of their time communicating between each other. In the very unlikely scenario of degradation becoming so bad (the number of transputers in any given loop would have to be considerably more than at present) that the viability of the system was put in jeopardy then either the loop would have to be split into sub loops or an alternative networking topology would have to be implemented.

- 2) The need to try and fit tasks that are by their very nature parallel into a sequential ordering is eliminated. In the prototype developed there are many tasks being performed simultaneously. These tasks range from obtaining data from sensors, to updating the workspace model. It would be both difficult and undesirable to have to order these tasks into a list that could be processed sequentially. The use of a parallel processing architecture has enabled the tasks to be executed in their natural order, which for some of them is concurrently.

8.2.2 Advantages of a Modular Sensing System

During the period of this research project many benefits of a modular sensing system over conventional systems have come to light. As stated earlier these advantages can be summarised under five broad headings. These five headings will now be expanded upon with reference to modular sensing systems in general and the system documented in this thesis in particular.

- 1) Divide and conquer techniques can be applied. The prospect of automating an entire manufacturing process, in all but trivial cases, is a daunting prospect. The scope for design errors and implementation difficulties increases greatly with any increase in size, or difficulty, of task. If the system is broken down into subsystems then these can (if they are chosen carefully) be automated independently of the other subsystems. Problems that develop during automation are then likely to be less dramatic in impact. This modular approach means that less complex subsystems can be automated first, with the experience gained providing a useful aid when automating more complex subsystems.

The application of the divide and conquer principle, to the system documented in this thesis, has led to the development of two separate modules. The first being a module to locate objects within a robot's workspace and the second a module to identify these objects. The object location module, which was developed first, took far longer to produce than the identification module. This difference in development time was caused by two factors. Firstly, the object location module is conceptually more complex than the identification module. Secondly, and more significantly, to enable the data structure for modelling the robot's workspace to be implemented on an array of transputers, a means of inter-transputer communication had to be developed. The design and implementation of this communication facility took a considerable amount of time and effort to develop. However, when the networking algorithm had been written it was available for use in the object identification module.

- 2) Modifications can be confined to individual modules. If the system requires modification in the future then, if a modular approach has been adopted, only the subsystems affected need be changed. These changes can be made and validated independently of the rest of the system.

This advantage became evident during the development of the object identification module. While developing the module several 'variations on a

theme' were implemented. These variations required rewriting substantial parts of the module. However, these changes were made without any alterations to the object location module being required. This independence of modules again demonstrates the benefits of a modular sensing system.

- 3) A library of building blocks can be established. Some of the subsystems may have elements common to other subsystems. These common elements can be made into common library modules. Using library modules saves time and effort as common modules only have to be designed and tested once.

The addition of further modules to the system implemented will benefit not only from the availability of the networking algorithm but also from the considerable experience gained during the implementation of the first two modules. It is thus highly probable that any future modules, of similar complexity to the two already written, could be implemented in a much shorter period.

- 4) The financial outlay can be spread over a longer period. Initially only a minimum configuration system need be purchased and then extended later as required.

During the duration of the research the number of transputers available to the author has risen gradually from one to twenty. Whenever more transputers become available (their procurement was on an ad hoc basis rather than to meet specific needs) they were added to the system without any major amendments to the software running on the transputers already installed. This method of adding extra transputers clearly demonstrates that with a modular sensing system the financial outlay can be spread over a longer period. Initially only a minimum configuration system need be purchased and then extended later as required.

- 5) A multi-layered (or multi-faceted) user interface can be constructed. The splitting of the system into modules has, in addition to the advantage listed above, a further advantage particular to the application under discussion. Modularisation has had the effect of enabling a multi-level user interface to be produced. (A diagrammatical representation of this multi-level user interface is shown in figure 8.1).

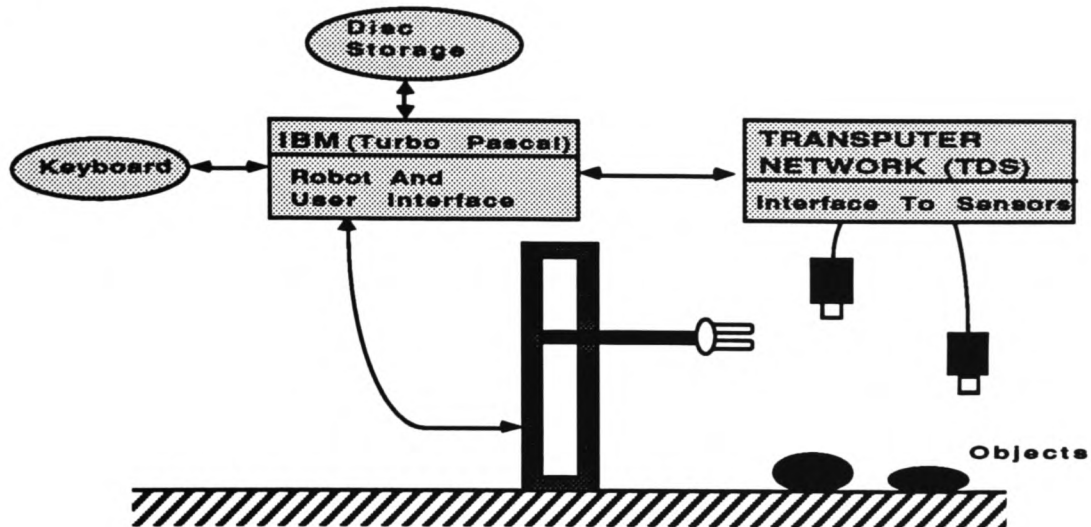


Figure 8.1 - A diagrammatical overview of the system.

Each level of user from implementer, modifier, and day-to-day user, has a view of the system which is consistent with their needs. For example, the day-to-day user does not have to know that the code that controls the sensors is written in Occam or that the results that the system generates are presented to them using an interface coded in Pascal. Similarly, the person responsible for coding the user interface or the program to control the robot arm requires no knowledge of the Occam subsystem.

The advantages of a multi-level (or layered) user interface can be summarised as:-

- (i) The underlying technology is revealed to the user on a 'need to know basis'. This leads to a steeper learning curve for the majority of users.
- (ii) The highest layers can be modified quickly and easily to suit individual requirements.
- (iii) Layers can be modified and updated independently of the rest of the system.

8.2.3 Additional Benefits of Work

As concluded at the end of chapter 3 the data structure finally chosen to hold the workspace module was a partial run length encoding structure. While considerable work was carried out on modifying octrees to meet the requirements of the system the octree could not compare with partial run length encoding in either storage or computational efficiency. The work on octrees has however, been presented at both conferences and colloquia and, from the interest shown by delegates, will hopefully be of benefit in other areas of 3-D modelling.

The work carried out, as part of this project, on the transputer has also attracted interest at home and abroad. In particular several Universities in America, where the transputer is virtually unheard of, have expressed interest in the work. While some of the work on transputers is obviously project dependent, other parts are applicable to a wider range of subject areas. For example, the author has already used the implementation rules of chapter 5.2 (developed through a process of evolution), together with the networking facility described in chapter 5.5, to aid the implementation of an application in Geographical Information Systems⁽⁵⁶⁾. It is possible that in the future these rules will require further refinement. They are however, at the present time, excellent criteria by which to judge a system's efficiency.

8.3 Conclusions

The construction of a prototype modular sensing system has served not only to demonstrate the feasibility and benefits of building such a system, but has also brought to light some points that will need further thought before a fully functional system is produced.

The conclusions reached can be summarised under the following broad headings.

(i) The data structure used.

- 1) Although conceptually simple the octree is an unsuitable data structure for holding the workspace model.
- 2) Partial run length encoding provides an effective means for holding the workspace model.

(ii) The suitability of transputer architecture.

- 1) The transputer is a suitable architecture for use in the modular sensing system.
- 2) A more dynamic allocation of processors to processes might improve the overall efficiency of the system.
- 3) Occam is the natural language for programming the transputer.

3. The viability of the system is beyond doubt and, therefore, work should begin in producing a fully operational system, using the prototype as the base. However,
 - (i) Modifications to the way in which data is acquired would lead to improvements in the object location module.
 - (ii) The use of additional features in the object identification module would further eliminate any problems with object misclassification.

These conclusions will be expanded upon in the following subsections before being used as a spring board for suggesting possible areas of further research.

8.3.1 The Data Structure Used

- 1) Although conceptually simple the octree is an unsuitable data structure for representing the workspace model. At the outset of the project it was proposed that the model of the workspace would be stored using a hierarchical data structure such as an octree. These types of data structure are, however, by their very nature expensive in terms of both storage and computational efficiency. It was therefore realised, from the start, that some modifications would be required to traditional octrees in order to suit them to the needs of the current modelling system. However, even after considerable effort had been exerted in modifying these conventional octrees they still fell short of the required standard in both memory and computational efficiency. It was therefore decided to put hierarchical data structures to one side in order to investigate the use of other types of data structures.

From the results documented in chapter 3 it is clear that, under the given circumstances, this was the correct decision. It is important, however, not to be too disparaging of the octree. It might well be that in future years when computational efficiency is not so critical (for example after the release of the new T9000's transputer which will be at least ten times faster than existing transputers) that octrees might become a viable option. Octrees could also prove useful in situations where the workspace, to be modelled, is small or where only low resolution modelling is required.

- 2) Partial run length encoding provides an effective means for representing the workspace model. After much consideration and preliminary investigation it was decided to implement a partial run length encoding data structure to compare the results obtained with those achieved using the octree. The results, in terms of both computational and memory considerations, showed that the

partial run length encoding technique was far better suited to the modelling task at hand than the octree.

8.3.2 The Suitability of the Transputer Architecture

- 1) The transputer is is a suitable architecture for use in the modular sensing system. The system developed has been implemented on processors that are by-and-large homogeneous. The choice of these homogeneous processors, namely the transputer, was chiefly based on their availability to the researcher and their ability to perform the task more efficiently than a single processor system. On the whole the transputer has proved to be an ideal architecture, for the implementation of the system. Its design enables it to be easily used as a building block in expandable and flexible networks. There are, however, a few disadvantages in using the transputer. For example, it has a distributed memory with each transputer having no access to the information stored on other transputers. The object identification module for instance would have been better suited to a system where each processor had access to a shared memory. Any disadvantages incurred in using the transputer are, however, easily outweighed by the advantages of its flexibility.

Harnessing the processing power of the transputer networks to the needs of robotic sensing systems required the development of a fluency in parallel systems design equal, but different, to the skills required in the development of traditional sequential systems. Some tasks are inherently sequential and, even when tasks can be broken down into sections that can be executed in parallel, the scope of these sections is not always self evident. Time, care, and skill, must all be applied if an efficient and truly parallel system is to be developed. Thus before any real development work can be carried out, a grasp of the basic principles of parallel processing must be obtained.

- 2) A more dynamic allocation of processors to processes might improve the overall efficiency of the system. While transputers provide a relatively cost effective means of modelling a robot's workspace, it is still important that they are used as efficiently as possible. In the current implementation each transputer, in the bidirectional loop used to model the workspace, is assigned a portion of the model relating to a given area of the workspace. This is the obvious way of allocating transputers as it is the most conceptually simple. However, in certain circumstances this type of allocation does lead to inefficient use of some transputers in the network. For example, consider the case where the majority of the workspace is empty and only a relatively small part of it

contains any objects. The transputer allocated to the part of the workspace which contains the objects will be kept very busy while the other transputers will have relatively little to do.

Obviously, if a method of transputer allocation could be found that was both conceptually simple and allowed a more balanced distribution of the workload, then this would in some circumstances be of substantial benefit to the system.

- 3) Occam is the natural language for programming the transputer. All the algorithms implemented on the transputer network have been coded in Occam. This has proved to be a very effective language for coding the application although its use did require a considerable amount of time by the author to master some of its more unusual features. In contrast with the views of other researchers (see chapter 4), it is the author's opinion that Occam is the natural language for programming the transputer.

8.3.3 The Viability of the Completed System

- 1) Modifications to the way in which data is acquired would lead to improvements in the object location module. Although in general the object location module described in chapter 2.2 works well for detecting the presence and position of objects within a robot's workspace, confusion can sometimes arise.

For example consider the plan view of a robot's workspace containing three objects (figure 8.2). If three views are taken of the robot's workspace then it is possible to end up with three intersecting views, figure 8.2(a). Using the premise that 'the more views that intersect in a given area the greater the likelihood an object exists in that area', a wrong conclusion would be drawn.

Obviously, figure 8.2(a) is the worse possible case and it is equally likely that three views would have resulted in a more correct conclusion being deduced (for example figure 8.2(b) - the best possible case).

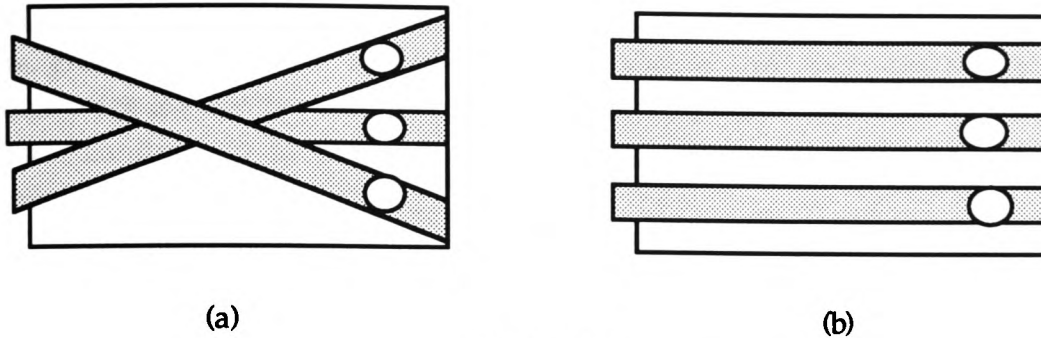


Figure - 8.2

(a) *Using the intersection of views to determine the position of objects can lead to incorrect conclusions being made.*

(b) *Using the intersection of views to determine the position of objects can produce very good results.*

In the prototype, the problem of misinterpreting intersecting views has been overcome by always taking a vertical view of the workspace. If this vertical view is obtained from a sensor with a high degree of accuracy in its imaging capability, then correct object location can be achieved. In the long term this might prove to be an unsatisfactory solution to the problem and further thought is given to its solution in sections 8.4.2 and 8.4.3.

- 2) The use of additional features in the object identification module would further alleviate any problems with object misclassification. The problems encountered during object recognition are minimal and for most practical purposes it is possible to facilitate the recognition of objects from a given object set. For those object sets where problems do occur, the addition of further, more appropriate, features to the feature set would rectify most outstanding problems with misclassification.

It might however prove worthwhile to look at the possibility of using a neural network to facilitate object recognition. This train of thought is pursued in section 8.4.4.

8.4 Suggestions for Future Work

What is apparent at the conclusion of this research project is the wide scope for further work in the field of modular sensing systems. Due to time constraints, many avenues of potential interest that have come to light during the project, have not been pursued fully. There are more unanswered questions and more ideas for possible research in the mind of the author at the end of the project than there were at the beginning.

With the solution to each problem encountered along the way have come more ideas, greater expectations, and thus further problems. The suggestions given for further work are not therefore an exhaustive list of all the ideas that have arisen but are confined to those areas that the author hopes to investigate in the foreseeable future.

8.4.1 More Dynamic Processor Allocation

As noted in section 8.3.2 the division of the workspace model amongst the transputers in the network on the basis of area can, in certain circumstances, lead to an unbalanced workload. The simplest way to lessen the effect of this imbalance is to make each transputer responsible for several small and disjointed segments of the workspace model. The possibility of an unbalanced workload, although not eradicated, would thus be significantly reduced.

A better, although more complex, solution would be to allocate the segments of the workspace to the transputers dynamically as the model is being built. In this way areas would be distributed relatively evenly amongst the transputers according to what the area contained. As each transputer would not know in advance what segments it will be allocated it would have to build up a "map" of the segments it and other transputers in the network were holding. Each transputer would, therefore, know something about everywhere and everything about somewhere.

8.4.2 Introduction of a Rule Based System

The problem of intersecting views illustrated in section 8.3.3, is likely to be reduced, although it could conceivably get worse, if additional views are taken from different positions. This solution, however, gives rise to another problem that is as damaging to the viability of the system as the one it solves.

As the number of views increases so the time to obtain those views and add their projection to the data structure also increases. The number of views that can be held within the data structure is also restricted by the limited memory of the individual processors.

The solution to these problems can be stated simply enough "do not waste time obtaining views that are not required, but obtain those views that will be of greatest benefit when analysing the workspace."

If humans were asked to analyse the workspace, depending on the workspace layout, then intuitively they would know which view to take. The problem that needs to be solved therefore becomes "how can human intuition be imputed to the computer?"

One solution might be to introduce a rule based system for determining which view should be obtained next. An example rule might be:-

(rule) If two views intersect then a third view should be obtained that intersects the first two at their intersection.

In practice, if two views were obtained of the workspace as in figure 8.3 which intersected then a third view that met the criteria of the above rule would make clear that there was no object at the intersection of the two views. However, if the same rule was applied to the workspace shown in figure 8.4 then it can be seen that the error in conclusion is compounded.

From the above it is clear that the choice of rules is an important factor that might prove valuable research material.

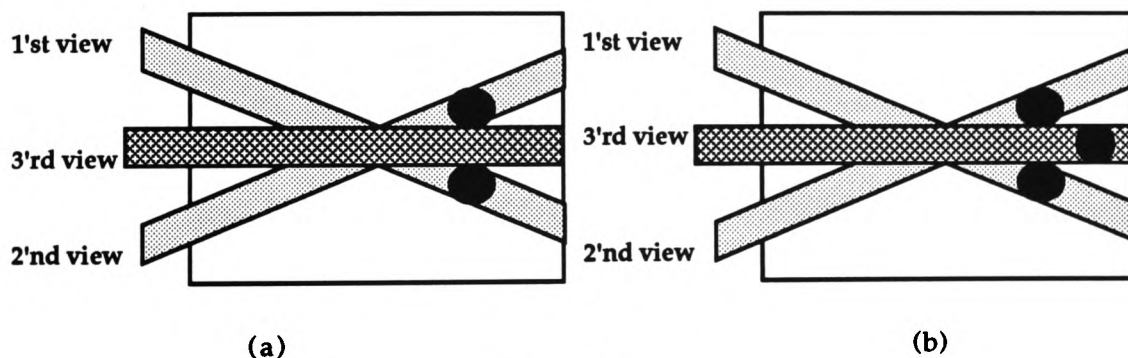


Figure - 8.3

- (a) *Using a third view at the intersection of the first two views can produce a more accurate model of the workspace.*
- (b) *Using a third view at the intersection of the first two views can produce a less accurate model of the workspace.*

8.4.3 Extension to the use of Certainty Factors

In order to construct the required three dimensional model the projection of the two dimensional image produced by each sensor through the workspace is calculated. The dimensions of this projection will depend on the orientation and angle of view of the sensor. In the present system it is assumed that each voxel that the projection passes through has an equal probability of being filled with part of the object.

As the orientation of a sensor is known, it is possible to deduce more information about the likelihood of objects being in a given position than at present. For example consider figure 8.4 where an object is viewed from three directions A, B and C.

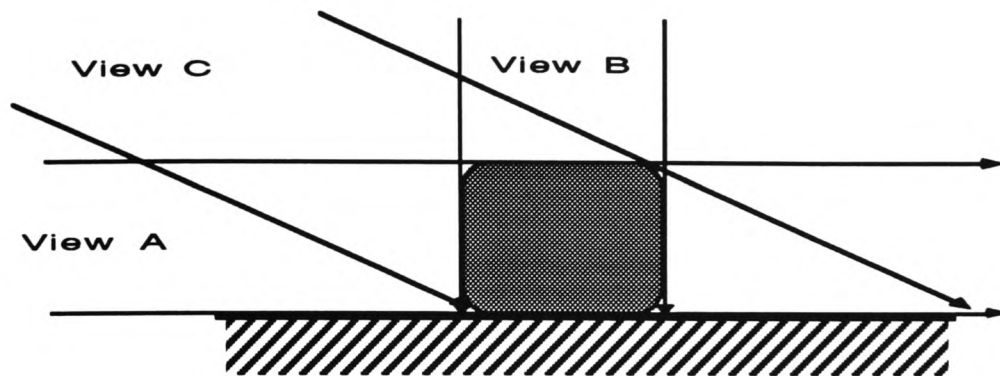


Figure 8.4 - The three views of the object reveal different information about its position.

View A enables the height (y axis) to be determined very accurately while giving no information about the width (x axis). For view B the opposite is true while for view C only an approximation is given for both the height and breadth. The 'certainty factor' used so far could be extended to give a certainty factor in the X, Y and Z planes.

8.4.4 Object Identification using a Neural Network

A neural network has been described as *"humanity's attempt to mimic the way the brain does things in order to harness its versatility and its ability to infer and intuit from incomplete or confusing information."*⁽⁶⁵⁾ This capacity to recognise objects, shapes and other features has been one of the reasons that has led scientists from many disciplines to attempt to model the human brain.

Despite the amount of active research in the area of neural networks there are many problems still to be overcome. Current neural networks consist of relatively small numbers of unsophisticated artificial neurons. Further, there is at present no mathematical way of determining the optimum number of nodes in these networks, or how best to organise them, for any given application. As a result most systems rely on empirical pruning algorithms which start off with a high number of nodes and then prune them until results become unacceptably poor.

To date, the majority of neural networks commercially available are either slow software simulations, running on Von Neumann architectures, or are limited in

scope of operation. This said, if a neural network were available that was both fast and flexible it would seem to be an almost ideal solution to the whole problem of object identification.

A feasibility study has already been carried out by the author into the possibility of developing such a neural network using transputer technology. The preliminary work has been carried out using a simulation package. There are already transputer based neural networks in existence^(66,67,68), but they are on the whole inflexible. In order to make the proposed network flexible it would have to be capable of holding a variable number of nodes on each transputer. While the amount of work involved in implementing this proposal should not be underestimated, the advantage to be gained, not only to the robotic sensing systems but to other 'real time' neural network applications, would be considerable.

8.4.5 The Addition of Further Modules

It is highly likely that the automated factories of the future will have some resemblance to those envisaged by Merchant⁽³⁾ nearly twenty years ago. He foresaw an automated factory consisting of a number of modular subsystems, each controlled by a hierarchy of small computers that interface to a large central computer. In the system envisaged, the subsystems would be responsible for the complete manufacturing process from the initial product design through to the production of the final product.

What constituted a small and large computer in Merchant's day were quite different to what the terms imply today and even his thoughts about a hierarchy of computers might prove to be a little off the mark. What is increasingly likely, however, is the development of a completely automated factory made up of modular subsystems. In the future it might be possible to purchase these subsystems modules 'off the shelf' and then to tailor them to specific needs. To this end it is proposed to add additional modules to the system already documented in this thesis. In particular it is proposed that, in the future, the additional modules will be added (figure 8.5):-

- (i) A module that will facilitate the dimensions of the workspace model to be produced automatically using a CAD type system. The system implemented thus far can be used to locate and recognise objects in a variety of robotic workspaces. These workspaces may range from a flat workbench to a room containing permanent fittings. At present the dimensions of the workspace are entered via the keyboard. The next stage in the evolution of the prototype

is to enable the dimensions of the workspace model to be produced automatically using a CAD type system.

- (ii) A module that will take the information provided by (i) and automatically calculate the optimum position for locating sensors. In the current implementation of the system there is no way of determining the optimal position for placing sensors. One possible way of achieving this would be to develop a set of rules that would use the knowledge provided from the CAD type system to enable the optimum sensor position to be determined.
- (iii) The development of a Job Control Module. At present the robot arm is controlled using joint level commands. An assembly process needs to be broken down into commands such as "open gripper and close gripper." These commands have to be coded individually and explicitly by the programmer. What is required is a higher level command interpreter which can take commands such as "place the smallest round object on top of the largest square object" and convert them into their component joint level commands. Obviously the control system would have to interface with the sensing system in order to determine "what and where is the smallest round object?" and "what and where is the largest square object?"

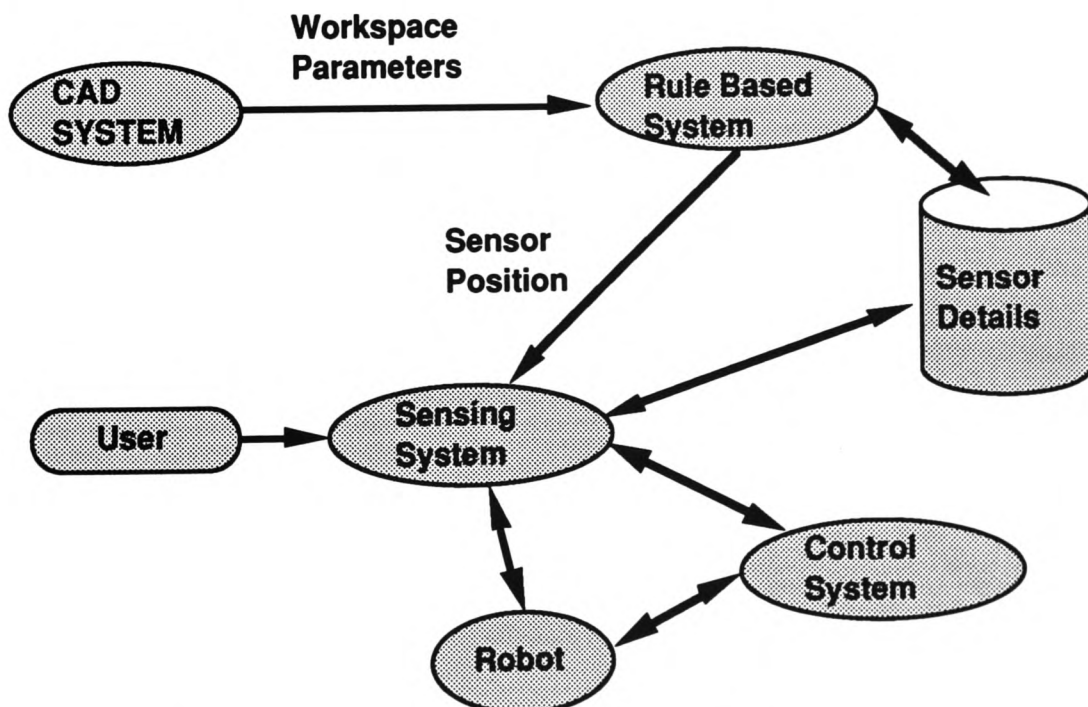


Figure 8.5 - Shows how additional modules could be added to the system.

8.5 Final Remarks

There can be no doubt that as robots continue to be used increasingly in automated manufacture so the expectations placed on their sensing systems will also increase. It is self evident that these sensing systems will, in turn, require greater computing power to control them and to analyse the data they acquire.

In the long term, this computer power will inevitably be provided by some form of parallel processing architecture. While it remains to be seen what this architecture will be, it is highly likely that some form of heterogeneous network of processors will be the ultimate solution; each module in the system being coded and executed on an architecture which best suits it's needs. In the near future, however, processing is likely to be based on homogeneous processors. From the work described in this thesis, it is clear that one possibility for this type of system is a transputer-type architecture. It should be noted, however, that, for various reasons, transputer technology has not had the impact on the computing world first envisaged by either the main transputer manufacturer or the DTI. It is quite probable, if not inevitable, that, in future years, the transputer will be of interest to computer scientists from an historical viewpoint only, in much the same way as the valve and the transistor. It will, nevertheless, have served its purpose in helping computer scientists develop their understanding of both the benefits and limitations of parallel processing.

The development of completely automated manufacturing environments will also require the production of data structures that will allow the integration of subsystems into the corporate whole. To make this integration as simple as possible, the individual data structures should also be as simple as possible. The partial run length encoding structure document in this thesis is not only a very good structure for the application developed, but is also conceptually simple and, thus, a good candidate for inclusion in these larger systems.

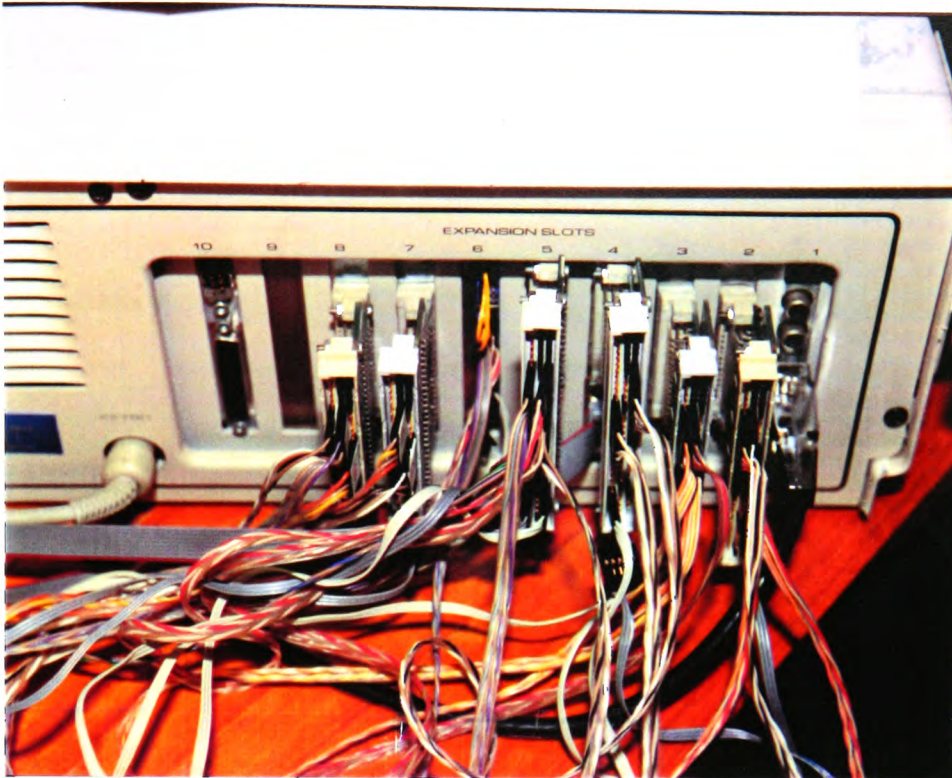


Plate A1.1 - Shows the 'reconfigurable links', as they appear in the expansion slots of the host computer.

References

References

- (1) **Nitzan, D. & Rosen, C. (1976)** "Programmable industrial automation", IEEE Trans. on Computers, Vol. C-25, No 12, December, pp. 1259-1270.
- (2) **Redford, A. (1991)** "Is there hope for robots in assembly?", Assembly Automation, Vol. 11, No. 1, p. 3.
- (3) **Merchant, M.E. (1973)** "The future of batch manufacture", Phil. Trans. R. Soc. Lond., Vol. 275A, pp. 357-372.
- (4) **Cicio, G.P. (1990)** "Integrating the plant floor", Proc. Vision '90, Society of Manufacturing Engineers, MS90-498.
- (5) **Horn, B.K.P. & Ikeuchi, K. (1983)** "Picking parts out of a bin", AI 746, MIT, October.
- (6) **Kelley, R.B. (1984)** "Heuristic vision algorithms for bin-picking", Proc. Fourteenth Int. Symp. on Industrial Robots, pp. 599-610.
- (7) **Nitzan, D. Barrouil, C., Cheeseman, P. & Smith, R. (1983)** "Use of sensors in robot systems", Proc. Int. Conf. Advanced Robotics, Tokyo, Japan, September.
- (8) **Schmitt, L.A. Gruver, W.A., Ansari, A. (1986)** "A robot vision system based on two-dimensional object-oriented models", IEEE Trans. on System, Man, and Cybernetics, Vol. smc-16, No. 4, July-August, pp.581-589.
- (9) **Nitzan, D. (1988)** "Three-dimensional vision structure for robot applications", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 10, No.3, May, pp. 291 - 309.
- (10) **Durrant-Whyte, H.F., Grime, S. & Hu, H. (1990)** "A modular, decentralised architecture for multi-sensor data fusion", Proc. Second Int. Conf. on Application of Transputers, July, pp. 71-77.
- (11) **Durrant-Whyte, H.F. (1987)** "Consistent integration and propagation of disparate sensor observations", The Int. Journal of Robotics Research, Vol.6, No.3. pp. 3-24.
- (12) **Pollard, S.B. (1987)** "The PMF stereo algorithm: theory and implementation", BCS Parallel Processing Specialist Group, Proc. of Parallel Architecture and Computer Vision Workshop, Somerville College, Oxford, March.
- (13) **Rygol, M. Pollard, S. & Brown, C. & Kay, J. (1990)** "MARVIN. AND TINA: A multiprocessor 3D vision system", Proc. Second Int. Conf. on Application of Transputers, July, pp 218-225.

References

- (14) **Porrill, J. Pollard, S.B. Pridmore, T.P. Bowen, J. Mayhew, J.E.W. & Frisby, J. P. (1987)** "TINA: The Sheffield AIVRU vision system", *IJCAI 9, Milan*, pp. 1138-1144.
- (15) **Brown, C. & Rygol, M. (1989)** "Marvin: Multiprocessor architecture for vision", *Proc. Tenth Occam User Group Technical Meeting*, pp. 95-107.
- (16) **Ware, J.A. (1988)** "A vision controlled robotic pick and place system", MSc Project, Dept. of Electrical and Electronic Engineering, Polytechnic of The South Bank, London.
- (17) **Hansen, C. Henderson, T.C. Shilcrat, E. & Fai, W.S. (1983)** "Logical sensor specification", *Third Int. Conf. on Robot Vision and Sensory Control*, pp. 578-583.
- (18) **Chin, R.T. & Dyer, C.R. (1986)** "Model-based recognition in robot vision", *Computing Surveys, Vol. 18, Part 1, March*, pp. 67-108.
- (19) **Martin, W.N. & Aggarwal, J.K. (1983)** "Volumetric descriptions of objects from multiple views", *IEEE Trans. on Pattern Recognition and Machine Intelligence, Vol. PAMI-5 No 2, March*, pp. 150-158.
- (20) **Chien, C.H. & Aggarwal, J.K. (1986)** "Identification of 3D objects from multiple silhouettes using Quadtrees / Octrees", *Computer Vision, Graphics, and Image Processing*, pp. 256-273.
- (21) **Chien, C.H. & Aggarwal, J.K. (1989)** "Model construction and shape recognition from occluding Contours", *IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 11, No. 4, April*, pp. 372-389.
- (22) **Ware, J.A. Davies, R.A. & Roberts, G. (1991)** "A user interface for robotic sensing systems: A multi-faceted approach", *IEE Colloquium on HCI: Issues For The Factory, Savoy Place, London, 21st February*.
- (23) **Hodgson, G.M. & Ruth, S.R. (1985)** "The use of menus in the design of on-line systems: A retrospective view", *SIGCHI Bulletin, Vol. 17, No.1, July*, pp. 16-21.
- (24) **Miller, A.R. (1983)** "MENU: An easy way to simplify CP/M", *Interface Age, November*, pp. 138-141.
- (25) **Jones, D.I. (1990)** "Transputers in control applications", *SERC/DTI Transputer Initiative, August*, pp. 51-59.
- (26) **Jones, S.R. (1988)** "Parallel processing computer architecture", in Fleming, P.J. (Ed.), *Parallel Processing in Control : The Transputer and Other Architectures*, Peter Peregrinus, London.

References

- (27) **Samet,H. & Webber,R.E. (1985)** "Storing a collection of polygons using quadtrees", *ACM. Trans. on Graphics*, Vol.4, No.3, July, pp. 182-222.
- (28) **Jackins,C.L. & Tanimoto S.L. (1980)** "Oct-Trees and their use in representing three dimensional objects", *Computer Graphics and Image Processing*, Vol. 14, No.3, pp. 249-270.
- (29) **Meagher,D. (1982)** "Geometric modelling using octree encoding", *Computer Graphics and Image Processing*, Vol. 19, No.2, pp. 129-147.
- (30) **Jones, C.B. (1989)** "Data structures for three-dimensional spatial information systems in geology", *INT J. Geographical Information Systems*, Vol. 3, No.1, pp 15-31.
- (31) **Gargantini, I. (1982)** "Linear octrees for fast processing of three dimensional objects", *Computer Graphics and Image Processing*, Vol. 20, pp. 365-374.
- (32) **Li,Z. & Telfer,D. (1989)** "Primitive quadtrees and type code quadtree approaches for the representation of binary regions", *IEE Colloquium on Pattern Recognition for Binary Images*, London, 7th April.
- (33) **Samet,H. (1984)** "The quadtree and related hierarchical data structures", *ACM Computing Surveys*, Vol.16, No.2, June, pp. 187-260.
- (34) **Lea,R.M. (1987)** "An overview of the Influence of Technology on Parallelism",in *Jesshope,S. (Ed.), Major Advances in Parallel Processing*, The Technical Press in association with Unicom Seminars Ltd., pp. 3-12.
- (35) **Fox,G.C. & Messina, P.C. (1987)** "Advanced computer architectures", *Scientific American*, Vol. 257, No. 4, October, pp. 44-53.
- (36) **Bhandarkar,S.M. Shankar,R.V. Suk,M. (1991)** "Exploiting parallelism in 3-D object recognition using the Connection Machine", *Robotics and Autonomous Systems*, Vol. 8, No. 4, pp. 291-310.
- (37) **Flynn,M.J. (1966)** "Very high speed computing systems", *Proc. IEEE*, Vol.5A, pp. 1901-1909.
- (38) **Flynn,M.J. (1972)** "Some computer organisations and their effectiveness", *IEEE Trans. Comp.*, Vol. C-21,pp. 948-960.
- (39) **Hockney,R. W. & Jesshope,C.R. (1988)** *Parallel Computers 2*, Adam Hilger: Bristol.

References

- (40) **Krishnamurthy,E.V. (1989)** Parallel Processing: Principles and Practice, Addison-Wesley, London.
- (41) **Morrow,P.J. and Perrot,R.H. (1987)** "An investigation of Low-level image processing algorithms on a transputer network", BCS Parallel Processing Specialist Group, Proc. of Parallel Architecture and Computer Vision Workshop, Somerville College, Oxford, March.
- (42) **Hey, J.G. (1988)** "Practical parallel processing with transputers", ACM Third Conf. on Hypercube Concurrent Computers and Applications, New York, January, Vol.1, pp. 115-121.
- (43) **Dowsing, R.D. (1988)** Introduction to concurrency using Occam, Van Nostrand Reinhold (International).
- (44) **Pountain,D. (1989)** "Occam II", BYTE, October, pp. 279-284.
- (45) **Hoare, C.A.R. (1978)** "Communicating Sequential Processes", Communications of ACM, August, pp. 666-677.
- (46) **Welch, P.J. (1992)** "The Role and Future of Occam", Proc. Transputer Applications - progress & prospects, SERC/DTI Transputer Initiative, March, pp. 152 - 169.
- (47) **Crookes,D. Kilpatrick,P.L. Milligan,P. Morrow,P.J. & Scott,N.S. (1987)** "LATIN: A language for transputer networks", Proc. Seventh Occam User Group, September, pp. 80 - 95.
- (48) **Marrow,P.J. Crookes,D. Kilpatrick,P.L. Milligan,P. & Scott,N.S. (1988)** "A comparison of two notations for programming image processing applications on transputers", Proc. Eighth Occam User Group, March, pp. 1-9.
- (49) **Culloch,A.D. (1988)** "Parallel programming toolkit for 3L - C, Fortran and Pascal", Proc. Eighth Occam User Group, March, pp. 23-30.
- (50) **Graham, I. & King, T. (1990)** The Transputer Handbook, Prentice Hall, London.
- (51) **Perihelion Software (1989)** "The Helios Operating System", Prentice Hall: Hemel Hempstead.
- (52) **Van Renterghem,P. (1990)** "The bridge from Present (Sequential) Systems to Future (Parallel) Systems: The Parallel Programming Environments Express and CS Tools", SERC/DTI Transputer Initiative, March, pp. 59-74.

References

- (53) **Henderson,B. (1990)** "Par. C System - a 'C' Compiler for transputers", SERC/DTI Transputer Initiative, April, pp. 45-51.
- (54) **Curtis,P.L. (1990)** "Modula-2 on transputers", SERC/DTI Transputer Initiative, March, pp. 31-47.
- (55) **Futo,I. & Kacsuk,P. (1990)** "CS-Prolog on Multitransputer Systems", SERC/DTI Transputer Initiative, April, pp. 57-66.
- (56) **Ware, A. & Kidner, D. (1991)** "Parallel implementation of the Delaunay triangulation within a transputer environment", Proc. of The Second European Conference on Geographical Information Systems (EGIS '91), April, pp. 1199-1208.
- (57) **Wallace,D.J. (1989)** "Supercomputing with transputers", Proc. First Int. Conf. on Application of Transputers, pp. 72-81.
- (58) **Wilcke,W.W. et al (1989)** "The IBM Victor Multiprocessor project", Proc. Fourth Int. Conf. on Hypercubes, April, Vol. 1, pp 201-207.
- (59) **Shea,D.G. et al (1990)** "IBM Victor V256 applications", Proc. Second Int. Conf. on Applications of Transputers, July, pp. 1-12.
- (60) **McDonald,K. (1990)** "Supernode II", SERC/DTI Transputer Initiative, February, pp. 67-69.
- (61) **Hey, A.J.G (1990)** "Supercomputing with transputers - past, present and future", ACM Int. Conf. on Supercomputing, June, pp. 479-489.
- (62) **Shepherd, R. (1992)** "The T9000", Proc. Transputer Applications - progress & prospects, SERC/DTI Transputer Initiative, Reading, March, pp. 77 - 81.
- (63) **Inmos (1987)** Transputer development system, Prentice Hall International (UK) LTD.
- (64) **Teague,M.R. (1990)** "Image analysis via the general theory of moments", Journal of Optical Soc. Am., Vol. 70, No 8, August, pp. 920-930.
- (65) **Tazelaar,J.M. (1989)** "Neural Networks",BYTE, August, p. 214.
- (66) **Beynon,T. & Dodd,N. (1987)** "The implementation of multi-layer perceptrons on transputer networks", in Traian Muntean (Ed.), Parallel Programming of Transputer Based Machines, pp. 108-119.
- (67) **Galushkin, A.I. (1991)** "On architectures of neural computers", SERC/DTI Transputer Initiative, March, pp. 45-55.

References

- (68) **Galushkin, A.I. (1991)** "Neurocomputers based on transputers and signal processors", SERC/DTI Transputer Initiative, February, pp. 49-69.
- (69) **Gonzalez,R.C. & Wintz,P. (1987)** Digital Image Processing, Second Edition, Addison-Wesley.
- (70) **Horn, B.P.K. (1986)** Robot Vision, (MIT Electrical Engineering and Computer Science Series), The MIT Press, McGraw-Hill.
- (71) **Yau, M. and Srihari, S.N. (1981)** "Digital Convex Hulls from Hierarchical Data Structures", CMCCS '81 / ACCHO '81, pp.163 - 171.
- (72) **Quinn M.J. (1987)** Designing Efficient Algorithms for Parallel Computers, McGraw-Hill Book Company, New York.

Appendix 1

Transputer Equipment and Algorithms

A1.4 Software Petitioning

As stated earlier, the hardware configuration for the prototype system consists of a number of sets of transputers connected into a series of bidirectional loops. The simplest specification for a working system is shown in figure A1.4, and consists of two such sets. The first set, which consists of transputers 0 - 7, is responsible for storing and manipulating the workspace model. The second set, which consists of transputers 8 - 12, receives and processes the data obtained via the connected sensor, and is also subsequently responsible for analysing the workspace model in order to determine the location of objects within the robot's workspace.

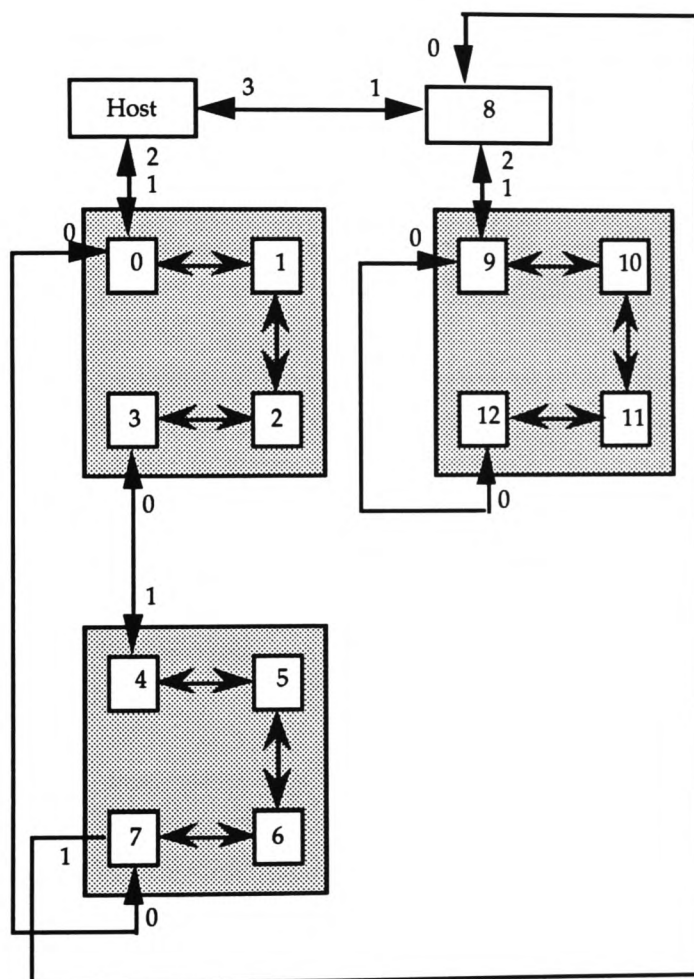


Figure A1.4 - Shows the simplest specification for a working system.

This simplest form of network can be extended in one of two ways. Firstly, when additional sensors are added a new set of transputers is required to control the sensor and process the data obtained via that sensor (see figure A1.3). Secondly, the processing power of individual sets of transputers may be extended by increasing the number of processors connected in the appropriate bidirectional loop.

Such extensions to the hardware can be facilitated without any major changes to the systems software. The way that this software and hardware independence has been achieved has been described earlier, and thus this appendix only describes the software which, in addition to the communication procedures detailed in chapter 5, is executed on each transputer.

Transputer	Software Executed
Host Transputer	<p>An interface mechanism between the transputer subsystem and the host PC.</p> <p>An interface between the object identification module and the workspace modelling module.</p>
Transputers 0 - 7	<p>The software necessary to manipulate the run length encoding data structure in the required manner. This software includes the facility to add additional view projections (calculated by transputers 9-12) to the workspace model. It also provides the image processing facility (transputer 8) with the three views of the model (plane, side elevation and front elevation) necessary for determining object location.</p>
Transputer 8	<p>The interface between the sensor and the camera (the transputer has an image processing plane attached to the transputer).</p> <p>Each time the connected sensor provides a view of the workspace the view is divided into sixteen equal parts. Transputers 9-12 are then each given a one sixteenth portion of the view and instructed to calculate the projection of that portion through the workspace. The view projection calculated is then packaged and sent to the transputers responsible for holding the workspace model. When a transputer has finished processing the portion of the view allocated to it, provided that there are still parts of the view unprocessed, it is sent a further portion.</p> <p>In addition to the view processing facility transputer 8 is also responsible for displaying and analysing the three views</p>

(plane, side elevation and front elevation) of the workspace model, in order to determine the location of objects.

After an object has been located a further view of the workspace is obtained. In this additional view the object to be identified is made the centre of attention. (That is, the view of the workspace is selected such that the object under consideration is in the centre of the field of view.) The view obtained is then preprocessed in order to determine the optimal threshold value and to remove noise. Using its image processing facility built into it, it is quicker for transputer 8 to perform these initial image processing techniques itself, as opposed to distributing the task to several nonspecialised transputers. Feature extraction, however, is a task that is computationally intensive and is best achieved by giving each of the transputers 9-12 a copy of the preprocessed view and then requesting each of them to extract a different feature. Finally, the extracted feature measurements are obtained from transputers 9-12 and analysed in order to facilitate object identification.

Transputers 9-12

Calculate the projection of the view segments, allocated to them by transputer 8, through the workspace. The calculated projections are then packaged and transmitted to the transputers responsible for holding the workspace model.

Extract from the preprocessed view of the workspace, provided by transputer 8, the required measurements needed to facilitate object identification.

A1.1 The Editing Environment

The Transputer Development System (TDS) consists of a plug-in transputer board and development software which runs on it. This combination provides a self-contained environment in which programs can be developed, compiled and executed. Programs can also be developed and compiled on the TDS to execute on a network of transputers, the code being loaded on to the network from the TDS. In this case the combination of transputer board and PC is referred to as the 'host computer', and the transputer network is known as the 'target system.' Finally, as is probably more realistic for most applications, programs can be developed for execution on transputers completely independent of the TDS; these are known as 'standalone' programs.

The principal interface to the system is an editor. As soon as the system starts up the user is placed in an editing environment, and all program editing, compilation and execution can be carried out within that environment, by the use of a set of function keys. Instead of having a special command language to the operating system to manage the filing system, file operations occur automatically as a result of certain editor operations. There is also a set of 'utility' function keys which may be assigned to different functions during a session.

The editor interface is based on a concept called 'folding.' The folding operations allow the text currently being entered to be given a hierarchical structure ('fold structure') which reflects the structure of the program under development. Just as a sheet of paper may be folded so that portions of the sheet are hidden from view, the folding editor provides the ability to hide blocks of lines in a document. A fold contains a block of lines which may be displayed in two ways; open, in which case the lines of the fold are display between two marker lines (called creases), or closed, in which case the lines are replaced by a single marker line called a fold line. Figure A1.1 shows a fold which contains five lower level folds, both in its open and closed state.

To create a fold the user inserts creases around the text to be folded; the fold is closed automatically when the second crease is made. Any text may be placed on the fold line to indicate what the fold contains; this text is called the 'fold header.' A fold may be removed, so that its contents are once again placed in sequence with the surrounding lines. Folds may contain text lines and also fold lines; therefore folds can be nested. Folds can be nested to a maximum depth of 50.

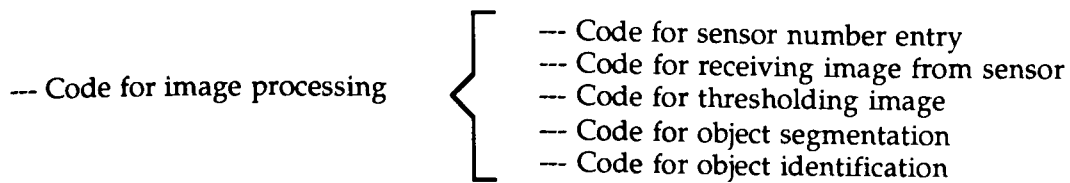


Figure A1.1 - Shows how five sub-folds can be held within a single fold.

A1.2 The Harlequin Image Processing Transputer

The Harlequin is one of a family of compatible transputer products, it conforms to the software standards of the transputer's manufacturers, Inmos, so that it may be connected to other transputer products from Inmos, Quintek and other manufactures. The Harlequin is designed to provide the essence of a high performance, transputer based workstation for vision systems. It combines a quality TV image capture facility with a high speed graphics unit in one slot of an IBM-PC. At the same time it incorporates a very fast processor to provide unencumbered access to the performance needed to modern image analysis systems. The Harlequin may be used in conjunction with other transputer boards to extend the processing power as required.

The unit provides (figure A1.2) 512 x 512 resolution display with a palette selected from a range of 2^{18} colours. The board incorporates an Inmos 32 bit T414 or T800 transputer with 1 MByte of dynamic RAM which supports the Transputer Development System (TDS) from Inmos. In addition the board is supplied with two quarter-MByte image buffers of dual ported video RAM. Video transfer takes place autonomously so that some 99% of the memory bandwidth is available for applications processing. A link interface is provided to the host computer's parallel bus.

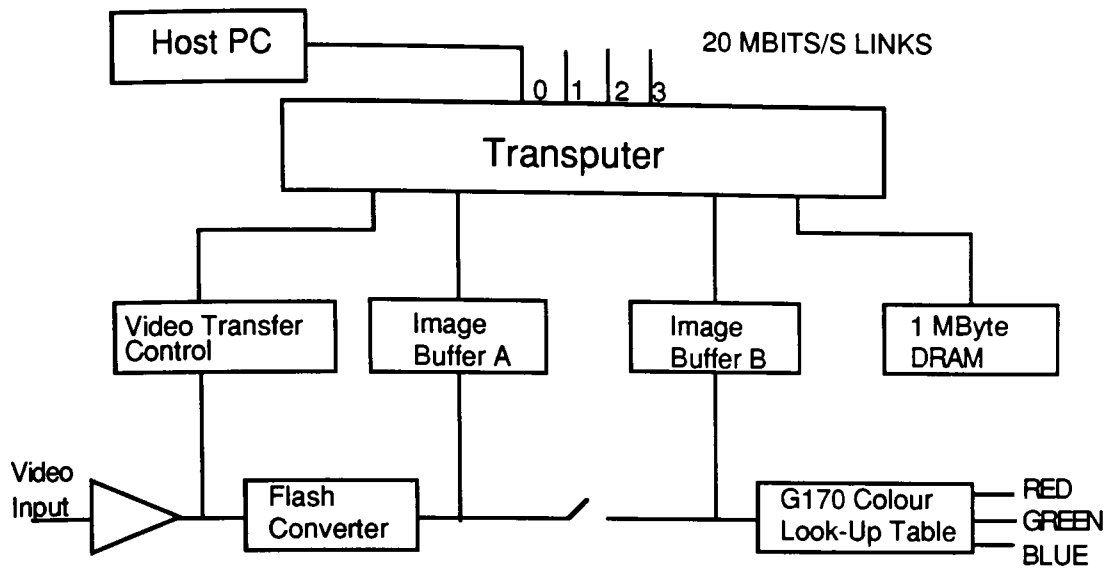


Figure A1.2 - The Harlequin is designed to provide the essence of a high performance transputer based workstation.

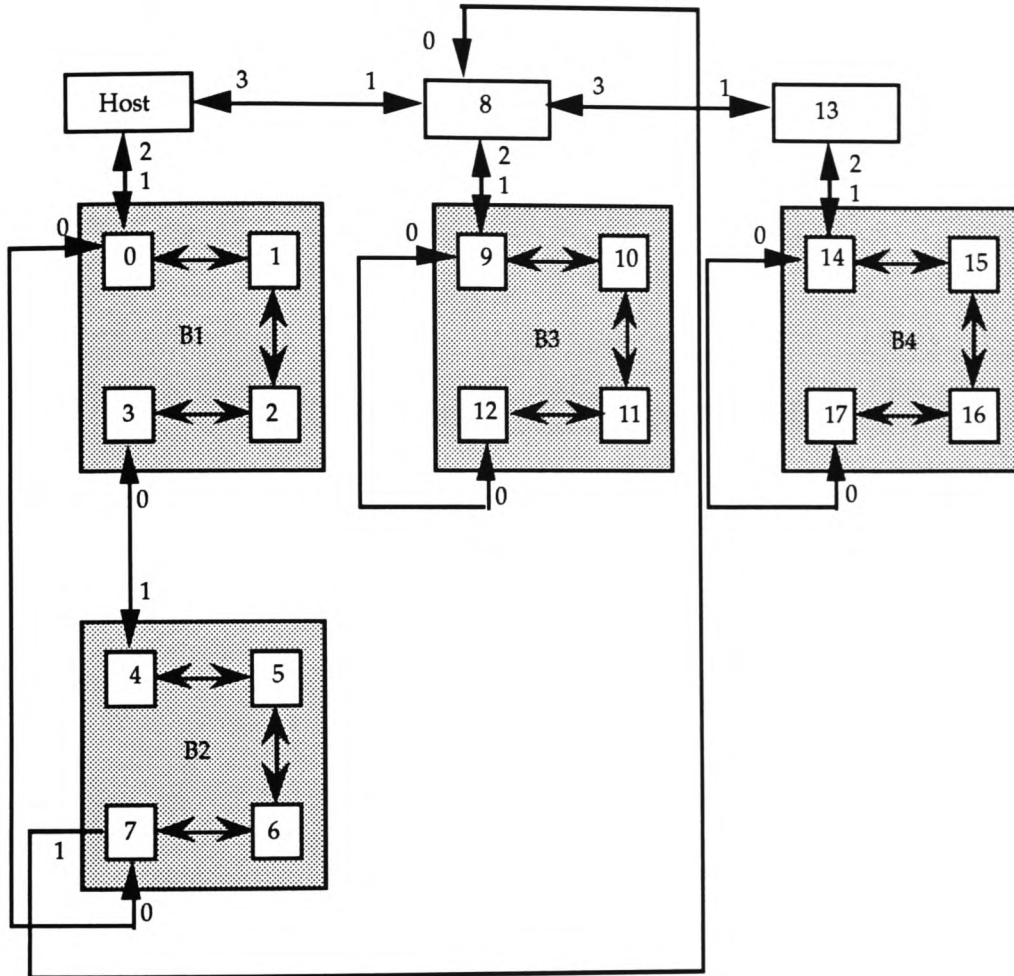
The Harlequin board accepts a standard 1 volt peak to peak monochrome signal of European or US standard, with input impedance of 75 ohms and sampled at up to 20 MHz using an 8 bit monotonic flash converter. The Harlequin can be configured to capture TV signals of formats including PAL, NTSC, RS-170, CCIR, VCR and Laser disk. Four channels may be multiplexed onto the flash converter.

The Harlequin provides a standard RGB output conforming to the RG170 standard on 9-way Dee connector with a 'pin-out' conforming to the PGA standard for analogue colour displays. The image may be displayed on a range of monitors of which the NEC multisync is suitable.

A1.3 Hardware Link Configuration

As stated in chapter 4, an Occam program consists of sets of processes which communicate with each other via fixed, unidirectional, and synchronised communication links called channels. Channels can be either 'soft linked' or 'hard linked.' Soft linked channels are used when the two communicating processes are executing concurrently on the same transputer, while with hard linked channels a physical connection has to be made between two communicating processes executing on different transputers. In Occam there is no logical difference between a soft linked channel and a hard linked channel.

In the system developed hard linked channels can be further broken down into 'on-board' links and 'reconfigurable links.' The 'on-board' links are fixed by the boards manufacture, while the reconfigurable connectors are fixed by the user of the transputers as required. A photograph of the 'reconfigurable links', as they appear in the expansion slots of the host computer, is shown in plate A1.1.



- \longleftrightarrow links that are fixed by the transputer board manufacturer.
- \longleftrightarrow links that are implementation specific. These links require the implementer of the transputer network to make the required connections.
- \longrightarrow A link that is only used for providing a continues boot-path for the transputers in the network.

Figure A1.3 - A diagrammatical view of the inter transputer connections in the final system implementation.

Appendix 2

*Details of Algorithms
Employed*

A2.1 Introduction

This appendix gives a brief outline of the algorithms employed in the object identification module. A fuller description of the optimal thresholding algorithm is given on pages 360 and 361 in the book by Gonzalez and Wintz⁽⁶⁹⁾, while a more comprehensive explanation of the use of moments in image processing is given by Teague⁽⁶⁴⁾.

A2.2 Optimal Thresholding

In the developed system the illumination of the workspace is provided by the ambient lighting in the room. The source and level of this lighting changes throughout the day, giving different levels of contrast between objects and their background. For this reason it is necessary to apply a thresholding algorithm to the image in order to provide a consistent contrast between the two.

Thresholding is the technique of labelling each pixel in a grey scale image as being either part of an object or as part of background. This is achieved by comparing the brightness value of each pixel in the grey scale image to the threshold value. If the brightness value of the pixel is greater than the threshold then it is assigned to one category, otherwise it is assigned to a second category.

The selection of the threshold value is often made from the histogram of image brightness (figure A2.1). This selection procedure assumes that the histogram will be bimodal with one peak representing the background while the other represents the object. In general, automatic evaluation of the threshold is a non-trivial matter. The distinction between background and object by the recognition of two distinct peaks in the histogram is not always possible.

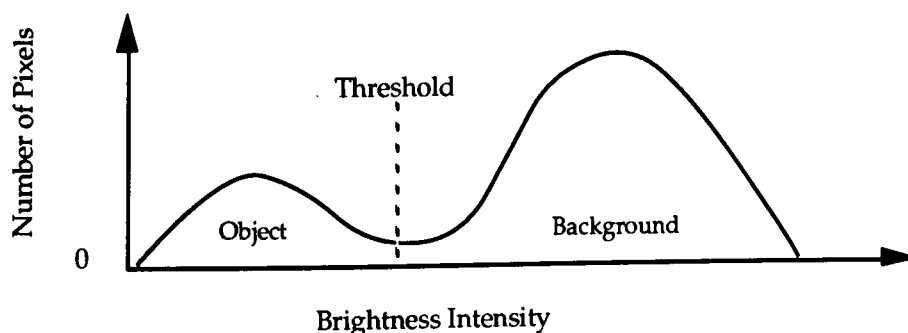


Figure A2.1 - A binary image made up of two unimodal distributions.

In the implementation of a suitable algorithm the following assumptions have been made:-

- 1) There are two principal brightness regions (the object and the background).
- 2) The histogram of the picture is an estimate of the brightness probability density function $p(x)$.

This $p(x)$ is formed from the sum or mixture of 2 unimodal densities corresponding to the object and the background which are allowed to be of different brightness. The mixture parameters are proportional to the areas of the two regions (ie. object and background).

Let $p(x) = P_1 p_1(x) + P_2 p_2(x)$, where P_1, P_2 are the apriori probabilities of the 2 levels based on area, in this case:- $P_1 + P_2 = 1$.

$$P_1 = \frac{1}{\sqrt{2\pi} \sigma_1} \exp \left[-\frac{(x - \mu_1)^2}{\sigma_1^2} \right] \quad P_2 = \frac{1}{\sqrt{2\pi} \sigma_2} \exp \left[-\frac{(x - \mu_2)^2}{\sigma_2^2} \right]$$

It is assumed that both object and background brightness are subject to noise which gives a spread of measurements which are normally distributed with mean μ_1, μ_2 and variance σ_1, σ_2 . It is further assumed that μ_2 the background is $> \mu_1$ the object, this means a threshold T can be defined to separate the two regions

- ie. all pixels with grey level $< T \rightarrow$ object,
 all pixels with grey level $> T \rightarrow$ background.

The probability of misclassifying a background point as an object point is:-

$$E_2 = \int_0^T p_2(x) dx$$

Similarly the probability of misclassifying an object point as a background point is:-

$$E_1 = \int_T^{255} p_1(x) dx$$

Therefore the total error arising from both object and background is:-

$$E(T) = P_2 E_2(T) + P_1 E_1(T)$$

A threshold is required for which this is a minimum. Therefore differentiate $E(T)$ with respect to T (using Liebnitz's rule) and equate to zero. The result is $P_1 p_1(T) = P_2 p_2(T)$

Using the equations given earlier and substituting for $p_1(T)$ and $p_2(T)$ a quadratic equation in T may be obtained.

$$AT^2 + BT + C = 0$$

$$\begin{aligned} \text{where: } A &= \sigma_1^2 - \sigma_2^2 \\ B &= 2\mu_1\sigma_2^2 - 2\mu_2\sigma_1^2 \\ C &= (\mu_2\sigma_1^2 - \mu_1\sigma_2^2) + 2\sigma_1^2\sigma_2^2 \ln(P_1\sigma_2 / P_2\sigma_1) \end{aligned}$$

This would indicate that there are two possible optimum threshold values. If however the variance of the distribution are equal, $A = 0$, and the quadratic reduces to a linear equation. (It is fair to assume that $A=0$, as in general the noise will have arisen from the imaging equipment.)

$$T = ((\mu_1 + \mu_2) / 2) - ((\sigma^2 / (\mu_1 - \mu_2)) \ln(P_1 / P_2))$$

It is worth noting that if $P_1 = P_2$ then the optimal threshold is the average of the mean. The same is true if $\sigma_1 = \sigma_2$ - obviously as this would mean no variation in brightness.

One thing which should be borne in mind is that parameters from the histogram are only estimates and thus will be subject to errors.

A2.3 Noise Removal

Twenty four nearest neighbour filter

With this filter each array element was replaced with the rounded average of the pixel and its twenty four nearest neighbours. It was found that while this removed noise effectively, large areas of the objects to be picked up were also removed.

Twenty four nearest neighbour filter with weighting

With this filter each array element was replaced with the weighted rounded average of the pixel and its twenty four nearest neighbours. The weighting factor is inversely proportional to the distance of the neighbouring pixel from the centre. It was found that this noise removal algorithm was effective in removing noise whilst leaving the image of objects unchanged. The twenty four nearest neighbour filter with weighting was thus adopted as the algorithm to be used in the finished vision system.

A2.4 Object Segmentation

In the image there may be a number of discrete objects. It is considered that any two pixels in a digitised image are connected if a path can be found between them, along which the pixel intensity values are the same. Thus in figure A2.2, A is connected to B but not connected to C. A connected object in a binary image is a maximal set of connected points, that is, a set such that a path can be found between any two of its points and all connected points are included. In figure A2.2 there are four connected objects and four holes (in addition to the background).

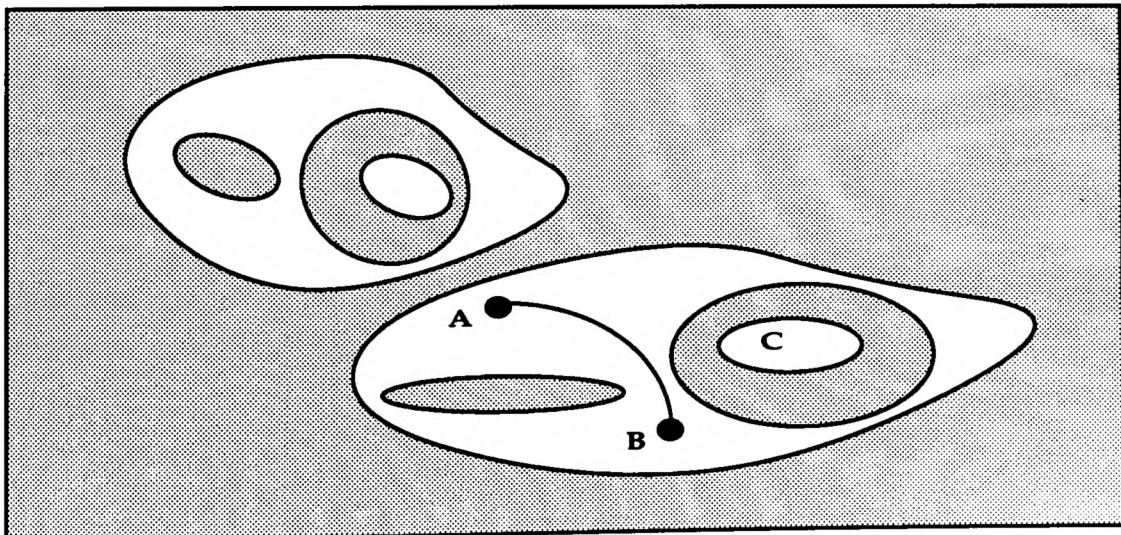


Figure A2.2 - An image with four connected objects.

A summarised description of the algorithm used to find connected points will now be given, a full description being provided on pages 66 and 67 in the book by Horn⁽⁷⁰⁾.

First the digitised image is scanned top to bottom left to right examining each pixel in turn. If the pixel is white (not part of an object) then it is ignored. If however, the pixel is black then it is labelled with a unique serial number (figure A2.3).

0	0	0	0	0	0	0	0	0	0	0	0
0	1	3	0	0	0	0	0	0	0	0	0
0	2	4	0	0	0	0	0	0	0	0	0
0	0	0	0	8	14	0	0	0	0	0	0
0	0	0	5	9	15	20	24	0	0	0	0
0	0	0	6	10	16	21	25	27	29	0	0
0	0	0	7	11	17	22	26	28	0	0	0
0	0	0	0	12	18	23	0	0	0	0	0
0	0	0	0	13	19	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure A2.3 -Each black pixel is given a unique serial number.

The second stage consists of scanning the digitised image top to bottom left to right, examining each pixel in turn. Each non-zero pixel is replaced with the lowest non-zero number of its eight nearest neighbours. This process is repeated until each object in the image is represented by a unique number (figure A2.4).

0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	5	5	0	0	0	0	0	0
0	0	0	5	5	5	5	5	0	0	0	0
0	0	0	5	5	5	5	5	5	5	0	0
0	0	0	5	5	5	5	5	5	0	0	0
0	0	0	0	5	5	5	0	0	0	0	0
0	0	0	0	5	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure A2.4 - Each object has a unique number.

A2.5 Calculation of Number of Holes

The number of holes in an image can be quoted in terms of the Euler number. The Euler number is defined as the difference between the number of objects in an image and the number of holes. If there is only one object in the image then the number of holes is equal to $H=1-E$.

In order to calculate the Euler number a direction called the 'stream direction' is defined relative to the image. The Euler number is the difference between the upstream facing convexities and the upstream facing concavities (figure A2.5).

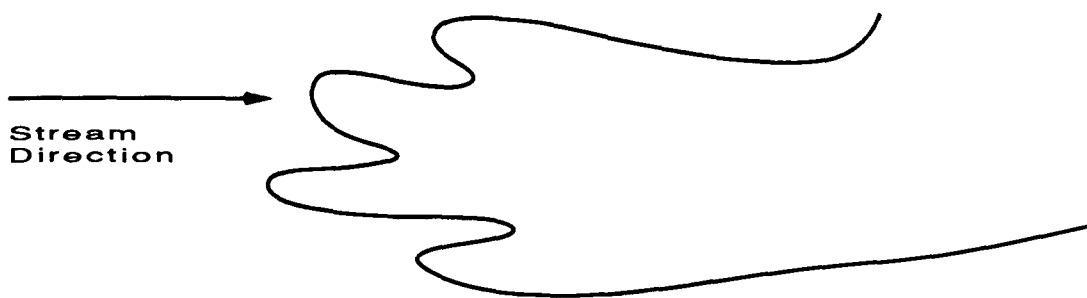


Figure A2.5 - Shows the stream direction towards an object. The object has 4 convexities and 3 concavities in view.

In the case of a discrete (as opposed to a continuous) image the number of convexities and concavities can be determined by counting the number of occurrences of a particular pattern in the image. Although the patterns that have to be counted will be different for each possible stream direction the technique is the same (figure A2.6). If X is the number of upstream facing convexities and Y is the number of upstream facing concavities then the Euler number is given by $X-Y$ (figure A2.7). This is a similar definition to the original concerning holes and objects. It is important however not to confuse holes with concavities and the objects with convexities.

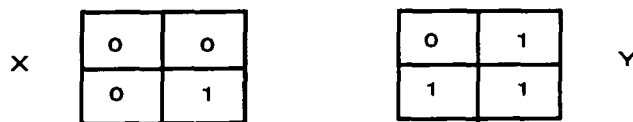


Figure A2.6 - Shows the patterns that needs to be counted for discrete binary images when the stream direction is taken as NW to SE. If a different stream direction was used then other patterns would be required.

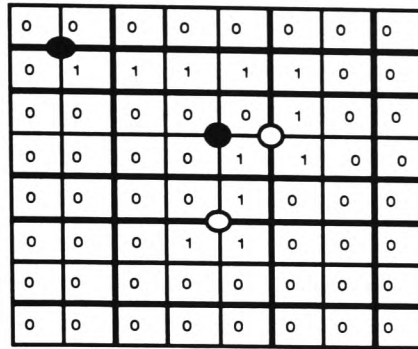


Figure A2.7 - A binary image where $X=2$ and $Y=2$ therefore $E=0$.

A2.6 Calculation of Convex Hull

The convex hull for the 2-D image of a 3-D object is normally calculated using elementary set theory (70). However, in the system developed, where the image is stored using an easily accessible image plane, the convex hull has been calculated empirically. This approach gives a fast and accurate rendering of the required area.

The algorithm starts by finding the "highest peak" in the image and then projecting a line at right angles from that peak. This projection is then rotated clockwise until a second peak is found. The line joining the peaks is the clockwise convex edge for the first peak. The process is repeated recursively, with the second peak becoming the start peak, until the completed convex hull is found (figure A2.8).

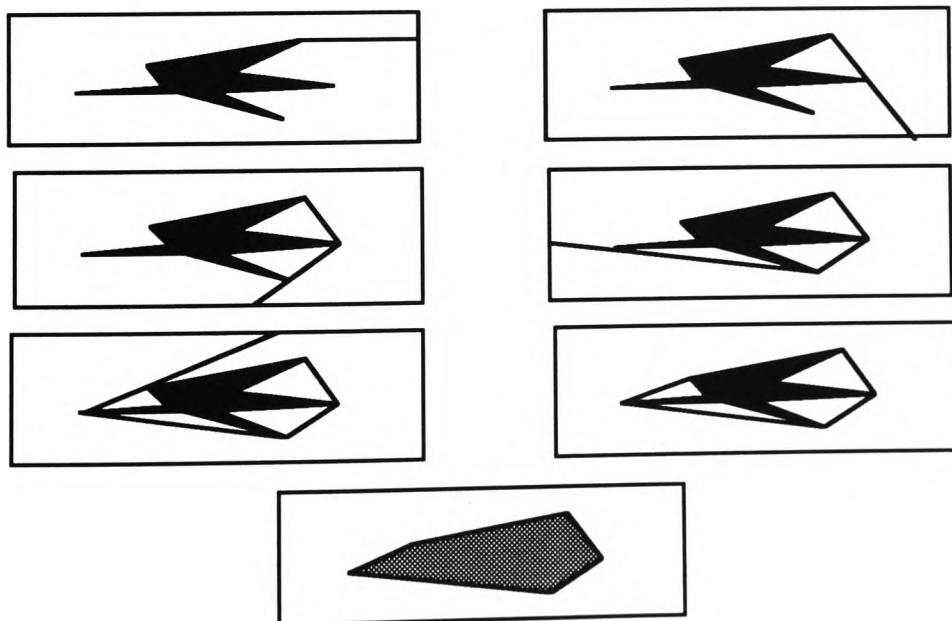


Figure A2.8 - Shows how the convex hull of a 2-D image is produced.

A2.7 Calculation of Principal and Secondary Axis

The calculation of the principal and secondary axis is carried out using the principles of moments described by Teague⁽⁶⁴⁾. Once these axes have been found the ratios between the divided halves of the axes are calculated (figure A2.9). That is, $P_1:P_2$ and $S_1:S_2$ respectively, where the sides have been labelled such that P_1 is greater or equal to P_2 and S_1 is greater or equal to S_2 .

Once these two ratios have been found the 'ratio of the ratios' is then calculated and fed as a parameter to the identification module.

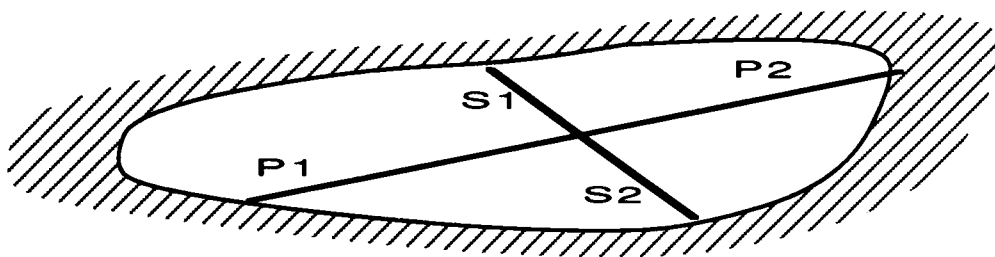


Figure A2.9 - Every object has at least one principal and secondary axis, although some may have many.

Appendix 3

Calibration and Calculation

A3.1 Camera Calibration

The calibration is carried out by placing a black disk of known radius at right angles to the camera. Using a disk means that there are no edges to be aligned with the camera. The distance between the camera and the disk must equal the maximum distance between objects and camera. An image is obtained and the optimum threshold calculated. After image thresholding, the number of pixels per millimetre in both the xI and yI (where xI and yI denote the coordinate system of the camera) directions are calculated. These values are used when calculating the dimensions of unknown objects from their imaged silhouettes.

The above calibration procedure assumes that the camera is uniform in resolution over its imaging surface and that the silhouette obtained is a parallel projection of the object. The validity of the former assumption can be verified from the documentation supplied with camera. The latter assumption results in the calculated bounding volume of some objects being larger than they actually are. The degree of this inaccuracy in measurement depends on the distance of the object from the camera and the angle of view of the camera. This inaccuracy in measurement is decreased when subsequent views are added.

The degree of inaccuracy introduced in the calculation is best illustrated by means of an example. If the object at $L2$ (figure A3.1), gave the same size silhouette as the calibration disk at $L1$, then the error in the calculated object size would be twice the measurement 'e'.

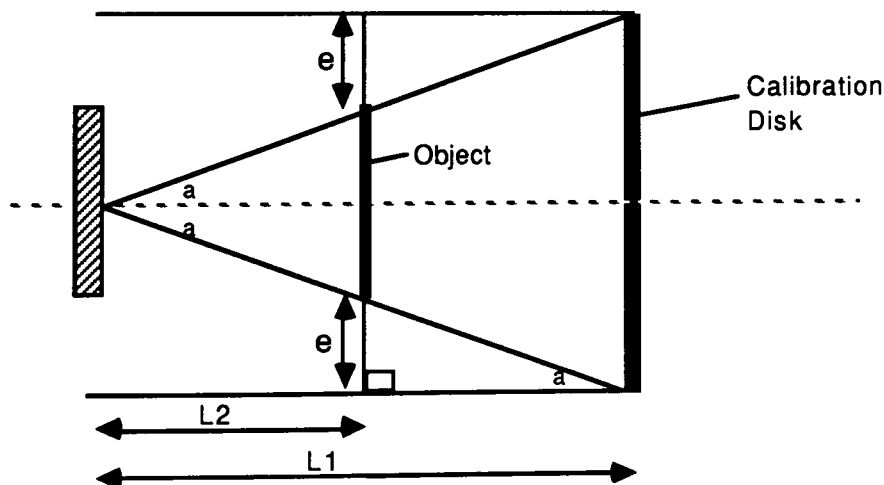


Figure A3.1 - Shows the error introduced into the system when the image is assumed to be a parallel projection of the object.

A3.2 Calculation of Projection Through Workspace

It has been stated earlier that a volumetric representation of an object can be obtained by projecting several two dimensional views of the object through the workspace and finding their intersection. These projections are calculated using the following procedure. In this discussion it will be assumed that the sensor is the aforementioned CCD camera although a similar practice would be used for calculation of projection for any other sensors.

The position of the sensor relative to some fixed reference point must be calculated. Position refers to both camera location (x, y, z coordinates) and orientation (angles it makes with the planes of the coordinate system). In this case two angles are sufficient to describe the orientation of the camera; they are the angle the camera makes with the side elevation (that is the rotation) and the angle it makes with the plan (that is the dip).

The final part of the process is to calculate the image projection through the workspace as follows. Assume a ray of light is passed parallel to the side and top planes and at an angle of ninety degrees to the front plane. This beam passes through the camera coordinates X, Y and Z. Now if the beam of light is moved through an angle 'a' in the horizontal plane and then through an angle 'b' in the vertical plane, a point relative to the camera centre, through which the beam passes, can be calculated (see figure A3.2). If only part of the beam is considered, and the magnitude of the section under consideration, shown as Z1 in figure A3.2, is varied then a series of points through which the beam passes can be calculated. If this process is repeated for each pixel in the image then the series of lines obtained by joining the calculated points forms the projection through the workspace.

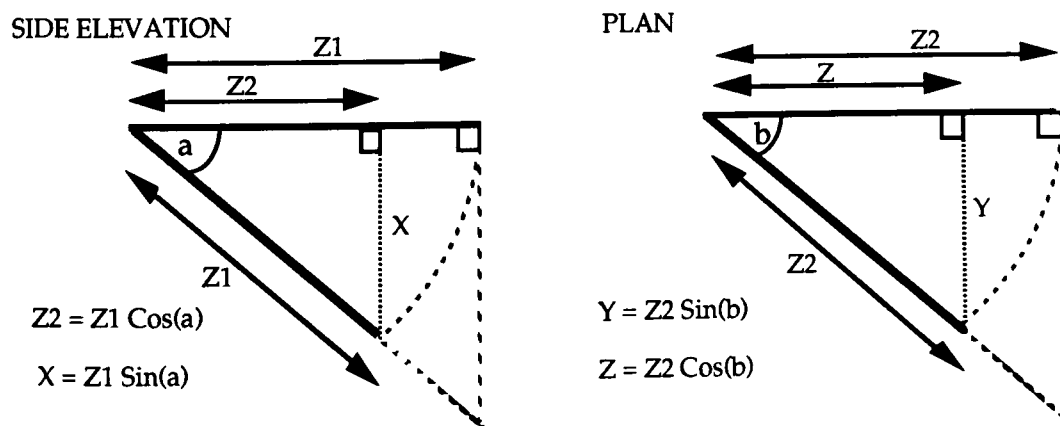


Figure A3.2 - Position through workspace given by X, Y and Z.

Appendix 4

Equipment Used

4.1 Introduction

This appendix gives a brief description of the equipment used during the course of the research. An overall view of the equipment can be seen in plate A4.1.



Plate A4.1 - An overall view of the equipment used during the course of the research.

4.2 The Tandy 3000

All the transputer boards used in the system, including the Harlequins, were housed in a Tandy 3000. The boards fit into the expansion slots in the Tandy 3000 and were connected to each other via hard wires (plate A4.2).

In addition to acting as the host for the transputer boards, the Tandy had a number of other functions. First, it provided the means through which the robot arm was programmed and controlled. Second, it provided the interface between the individual components of the system and the user. Third, it provided the hard-disk storage for the system.

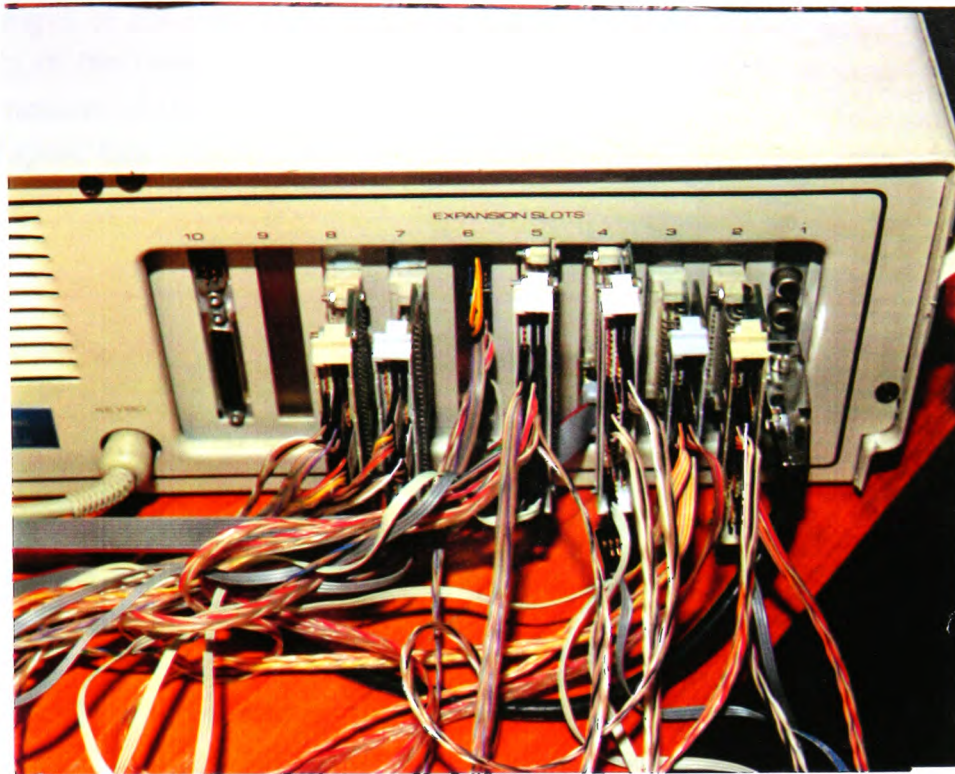


Plate A4.2 - Shows the 'reconfigurable links', as they appear in the expansion slots of the host Tandy 3000.

The specifications of the particular Tandy 3000 used were:-

- Intel 80286 chip;
- 8 MHz clock speed;
- 640K RAM;
- 20 Megabyte hard disk;
- Monochrome monitor;
- 2 * NEC multi-sync monitors (connected to Harlequin boards);
- 3¹/₂ disc drive.

4.3 The RTX robot arm

The robot arm used was the RTX, produced by UMI Ltd. The RTX, which is of SCARA design, was controlled via programs, written in Turbo Pascal, which were executed on the Tandy 3000. The Tandy 3000 was connected to the RTX via its RS232 serial link.

The arm (shown in figure A4.1), which was not designed to lift heavy loads, has had its end effector (plate A4.3) modified to enable it to carry a camera (plate A4.4). The weight of the camera and mount is greater than the recommended lifting capacity of the robot. This means that in order to provide a reasonably accurate measurement of the end effector's position, the RTX must be used at a much slower speed than would be possible with a lighter load.

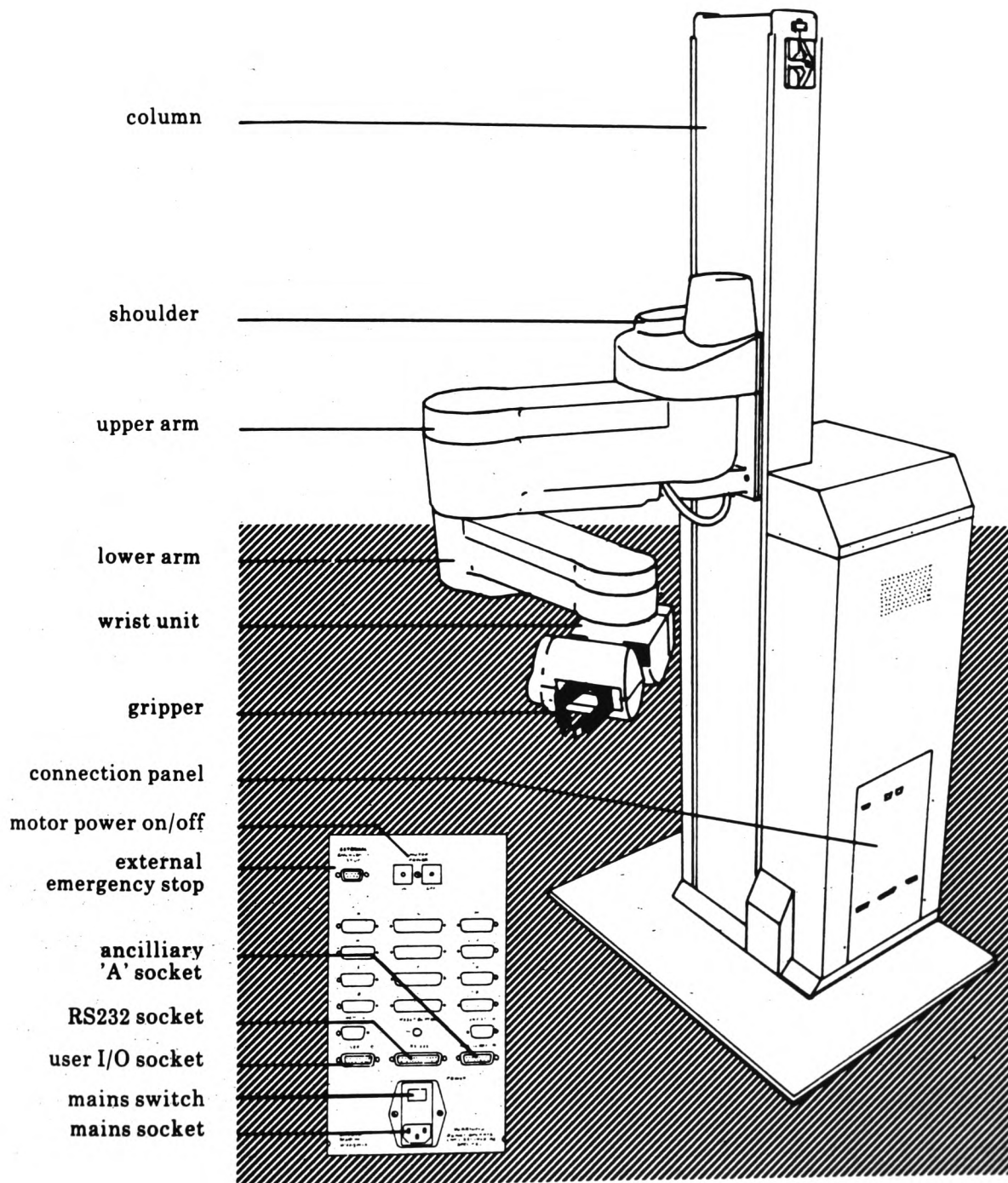


Figure A4.1 - Shows the components of the RTX.



Plate A4.3 - The RTX's end effector, before it was modified to carry a camera.

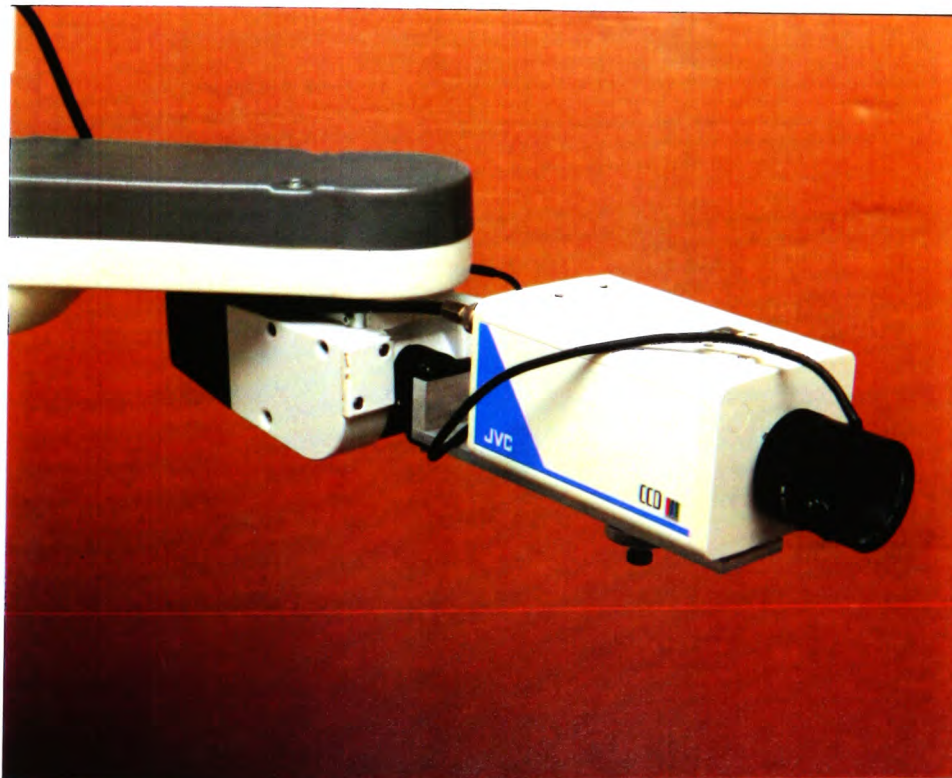
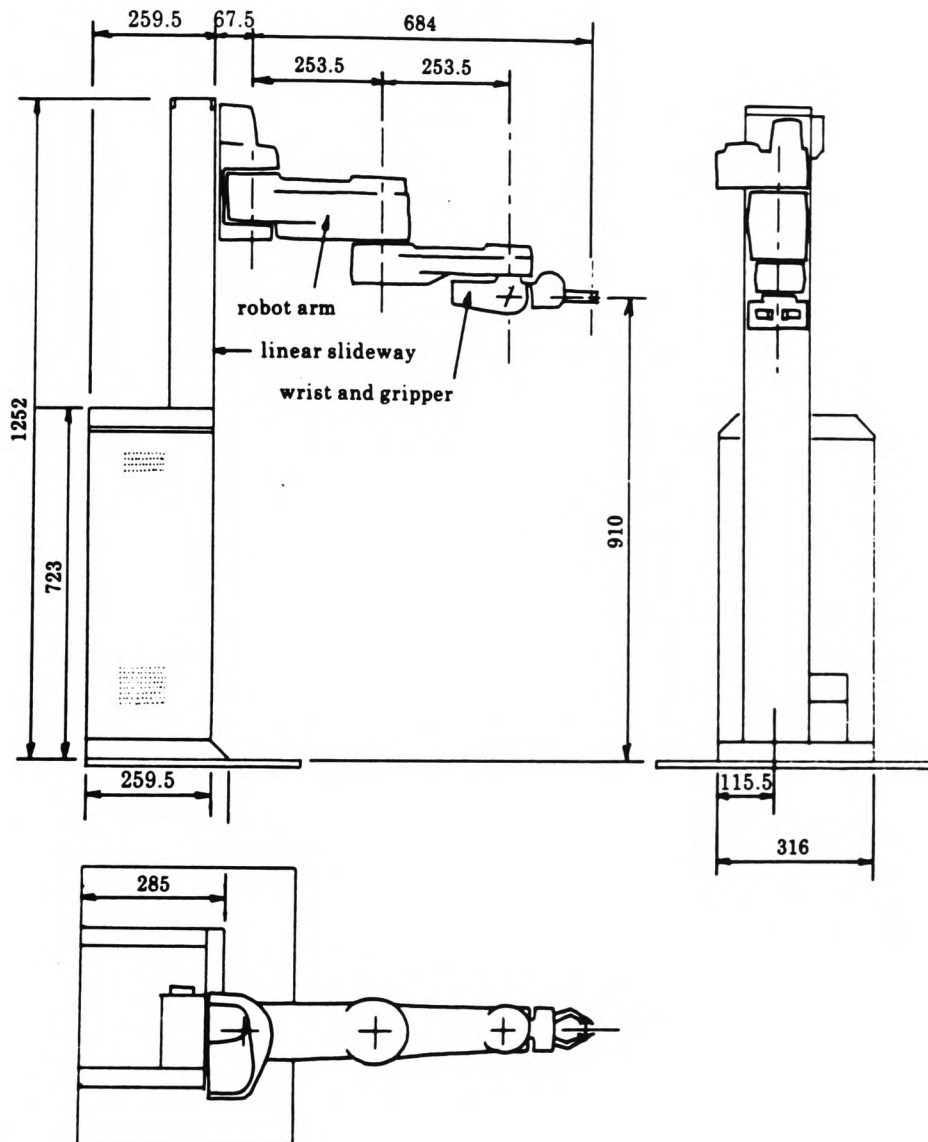


Plate A4.4 - The RTX's end effector, after it was modified to carry a camera.

The dimensions and range of movements of the RTX are shown in figure A4.2 and figure A4.3 respectively.



The overall weight of the arm is 35 kg, 77 lb.

Figure A4.2 - Shows the dimensions of the RTX.

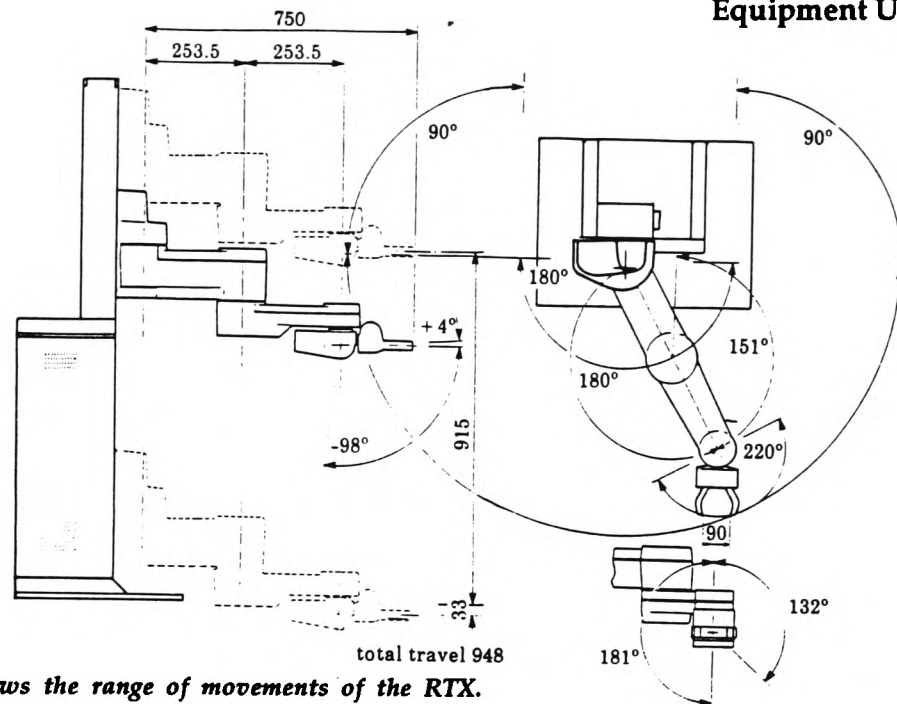


Figure A4.3 - Shows the range of movements of the RTX.

4.4 The CCD cameras

Although the modelling system has been designed to accept data from a wide range of sensing devices, for reasons of cost and availability, it is highly probable that most practical implementations of the system, as with the prototype, will use some form of vision system to acquire the data.

The choice of imaging devices for the prototype system was primarily dictated by cost, and so a number of relatively inexpensive solid state cameras - which use a charged coupled device (CCD) image sensor - were purchased. Each of these sensors, which are housed in a protective casing, is fabricated on a silicon chip using integrated circuit technology. They contain a matrix array (typically 512 by 512) of small, accurately placed photosensitive elements. When light passing through the camera lens strikes the array, each detector converts the portion falling upon it into an analogue electrical signal. The entire image is thus broken into an array of individual picture elements, also known as 'pixels'. The magnitude of the analogue voltage registered for each pixel is directly proportional to the intensity of light in that portion of the image. This voltage represents an 'average' of the light intensity variation over the area of the pixel.

The initial sensing operation performed by the camera results in a series of voltage levels which represent light intensities over the area of the image. This preliminary image must then be processed so that it is in a form suitable for analysis. This preliminary image process, which is carried out using the Harlequin's image plane (appendix A1.2), includes thresholding the image (appendix A2.2) and noise removal (appendix A2.3).

Appendix 5

Published Papers

These papers make up the published work relating to the thesis. It should be noted that the results in the thesis represent a more up to date account of the research than the work presented in these papers. Hence, in the instance of any inconsistency in the work documented in the papers and the thesis, the thesis should be taken as being correct.

A5.1 Paper 1

Ware J.A., Roberts G., Davies R.A., Miles R., Williams J.H.
(1990) "A modular sensing system for robotic control", The
Second International Conference on Applications of Transputers,
Southampton University, July, pp 78-85.

The paper also appears in the proceedings of the Fourth World
Conference on Robotic Research, Society of Manufacturing
Engineers, held at Carnegie Mellon University, Pittsburgh,
Pennsylvania, September 1991, pp 6:13 - 6:23 under the title "A
parallel processing architecture for robotic control."

The SME will be publishing the paper in their special addition
of Transactions on Robotics Research.

A summary of the paper was also presented at the IEE
colloquium on Parallel Processing: Industrial and Scientific
Applications, at The IEE, Savoy Place, London, 18th June 1990,
under the title "A transputer based modular sensing system."

A Modular Sensing System For Robotic Control

J.A.Ware and G.Roberts,
Dept. Of Mathematics and Computing,
The Polytechnic Of Wales, Pontypridd,
Mid Glamorgan, CF37 1DL.

R.A.Davies,
Dept. Of Computer Studies,
The Polytechnic Of Wales.

R. Miles,
The Transputer Centre,
Bristol Polytechnic,
Bristol, BF16 1QY.

J.H.Williams,
School Of Electrical Electronic and Systems Engineering,
University Of Wales, Cardiff, CF1 3YH.

Abstract. One of the main factors that inhibits the widespread use of robotic systems is the complexity of adding sensors to the robot's workspace. The majority of sensing systems that are currently available are inflexible in that the addition of extra sensors requires, at the very least, substantial changes to both hardware and software. This paper describes a transputer based system that contains the facility to add additional sensors without requiring major changes to hardware and software.

The process of extracting the information from the sensors is separated from the processes that use the information. This separation of information provider from information user enables the software that controls the sensors (and even the sensors themselves) to be upgraded with no corresponding changes to the information user software. Additional sensors can easily be added to the system while obsolete sensors and those with poor imaging characteristics can simply be removed.

1. Introduction

The manipulation of objects by a robot within its workspace requires knowledge about each object's location and orientation. A 3-D representation of the robot's workspace is thus required. This 3-D representation can be built up from information obtained from various sensors within the workspace.

At the outset of the research it was envisaged that before a 3-D model of workspace could be constructed the following would have to be addressed:-

- (1) The information extracted from different sensors will generally be at different resolutions.
- (2) The representation of 3-D space will require large amounts of computer memory.
- (3) The production of the 3-D model, particularly when a large number of sensors are involved will probably require a substantial amount of processor time.

It was also proposed that (1) and (2) may be solved by using a hierarchical data structure while (3) might be addressed by identifying parallel aspects of the processing and implementing them using a network of processors. The development and description of this data structure is detailed elsewhere [1]. The present paper describes the process by which the information to build the structure is obtained from multiple sensors in parallel with the process of identifying objects.

2. Modelling System Developed

The information extracted from multiple sensors is used to provide "upper bounds" on the locations of the objects within the robot's workspace. The accuracy of this "upper bound" in locating the object depends on the number and nature of the sensors used, but always encloses the object. The software is responsible for taking into account the reliability of information provided by different sensors.

To construct this 3-D model, the projection of each 2-D sensor image through the workspace is calculated [2, 3]. The dimensions of this projection will depend on the orientation and angle of view of the sensor. The composite 3-D model of the robot's workspace is then constructed by taking the intersections of each of the projections. Where an intersection occurs, the values contained in the intersecting voxels are added to the values already stored, the values being weighted according to the sensor's reliability. Thus, each voxel in the composite model will contain a value representing the 'certainty factor' (not, strictly speaking the probability) of the voxel being part of an object.

Once the workspace has been modelled, the next step is to use that model to locate the objects within it. The location of an object refers to its coordinates in space. Since an object occupies a finite volume of space, its coordinates would typically be defined relative to a specific point on the object. A correct physical interpretation of location of object would require the determination of its centre of gravity. However, in the system implemented the model is used to produce three two-dimensional views of the workspace (plan, side elevation and front elevation). The centroid (i.e., centre of gravity for the two-dimensional image) of each view is then calculated and used to give an estimate of the location of that object.

The system developed allows the following functions to be carried out:-

- 1) Update the model,
- 2) Display view of model,
- 3) Modify workspace characteristics,
- 4) Modify sensor characteristics.

2.1. Update the model

The update of the model may be broken down under the following headings:-

- (a) Obtain a view of workspace via sensor.
- (b) Calculate projection of view through workspace.
- (c) Add projection information to workspace model.

This process may be executed concurrently on any number of transputers; each transputer being connected to a different sensor.

2.2. Display views of workspace model

This process produces the three two-dimensional views of the workspace used to give an estimate of the location of objects within the workspace.

2.3. *Modify Workspace Characteristics*

It is envisaged that the sensing system being developed will be suitable for employment in a large number of different applications. As the application changes, so the characteristics of the robot's workspace will vary. In one application the robot's workspace might be a room while in a second application the robot's workspace might be a workbench. It was therefore necessary to build into the system a means by which changes in the dimensions of the workspace could be reflected in the workspace model. The resolution of the workspace model is constrained by the memory available on the transputers holding the model.

2.4. *Modify Sensor Characteristics*

Each time a new sensor is added to the system or a characteristic of an existing sensor is changed the sensor has to be calibrated. After sensor calibration the information obtained is stored for future reference.

3. Identification Of Objects Using Feature Extraction

All objects can be described in terms of their features. For some objects several features may be required to correctly identify them. These features may be anything from the volume and mass to the average curvature of an object.

For cases in which several object features must be measured to identify an object, a simple factor weighting method may be used to consider the relative contribution of each feature to the analysis. For example, to identify a valve stem from among a group of stems of several sizes, the image area may not be sufficient by itself to ensure positive identification. The measurement of height may add some additional information, as may the determination of the centroid of the object. Each feature would be compared with a standard for the goodness-of-fit measurement. Features that are known to be the most likely indicators of a match would be more heavily weighted than others. A weighted total goodness-of-fit score could then be determined to indicate the likelihood that the object has been correctly identified.

In the system developed, this feature extraction process is carried out in parallel with other modules in the system. To facilitate recognition of the object, a set of describing features is first produced for all objects within the object set. These features include:-

- (1) Ratio of the length of two lines which intersect at right angles at the centre of gravity of the object; one line being the axis of least moment of inertia of the object.
- (2) Ratio of perimeter length to object area.
- (3) Euler Number.

As the features are extracted they are added to a table of features. (Each entry in the table consists of a feature type together with a feature value.) If only a limited amount of computer power is available for object recognition then it is better to utilise it in extracting those features that uniquely identify an object from within a set. However, if there is sufficient processor power to extract the whole set of features for each object in view then a more reliable classification of objects may be obtained.

Using a Transputer network (where processor power can be expanded to meet demand) a feature table may be produced for the object set. Each object in the table will have a set of features associated with it together with a set of values for those features. The feature values are calculated using model data.

As objects within the object set change so will the usefulness of the various features extracted. For example, if the object set contains various shaped assembly pieces then the average degree of roundness of their edges may give a good indication as to their identity. However, in another case, such as a set of various sized disks, then knowing the average degree of roundness of edges does not help in object identification.

To allow for this change in feature usefulness, each feature value will have a weighting factor associated with it that will be recalculated each time the object set is changed. Thus for each set of objects, those features best suited to uniquely identifying objects will have the greatest weighting.

When an object is to be classified, an attempt is made to extract from it all the features listed in the table. The extracted features are then compared with the table values. Thus, the degree of correlation between each table entry and the object to be classified can be calculated.

When the system has completed the process of analysing object features, some conclusions must be made about the findings, such as the verification that an object is, or is not part of the recognisable object set. Based upon these conclusions, certain decisions can be made about the object or the production process.

On the one hand, the decision might be that the identified object should be manipulated in a given fashion. On the other hand, the decision arrived at might be to halt production until some unrecognised object is removed.

4. Configuration Of Transputer Network

The model of the prototype workspace was held on eight transputers, each storing the model for one eighth of the workspace. From figure 1 it can be seen that a more detailed description of the cell is held for the Y axis than for the X and Z axis. The level to which each axis is represented can be varied by the user depending on application.

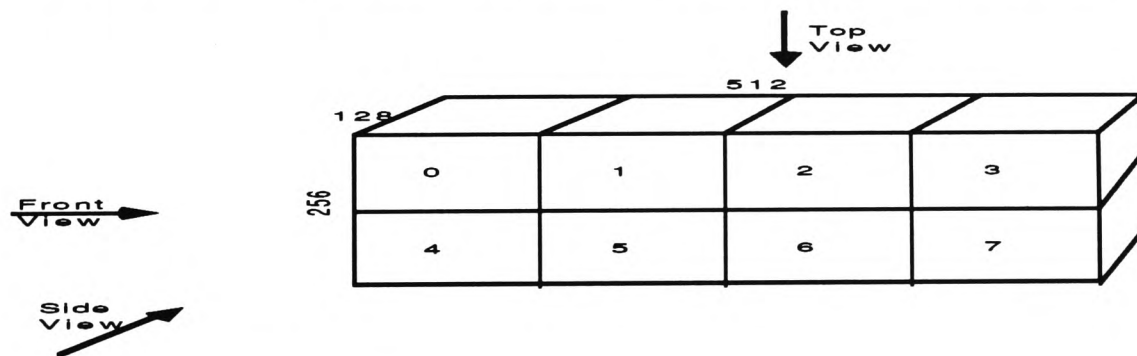


Figure 1 - Prototype workspace. The workspace which was 465 by 250 by 650 millimetres in size being held on eight transputers. Each section is broken down into 128 by 128 by 128 voxels.

Initially, multiple 2-D camera images were obtained using a CCD camera mounted on a SCARA robot arm. The robot arm gave great flexibility in positioning the camera in relation to the workspace. The CCD camera was connected to a 'Harlequin* frame grabber and image processing' board that in addition to processing the multiple views from the camera was used to display the three orthogonal views of the workspace model.

*Manufactured By Quintek Of Bristol.

The transputer configuration for the prototype system is illustrated in figure 2. The transputers that held the data structure were arranged so that they formed an extendible bidirectional loop.

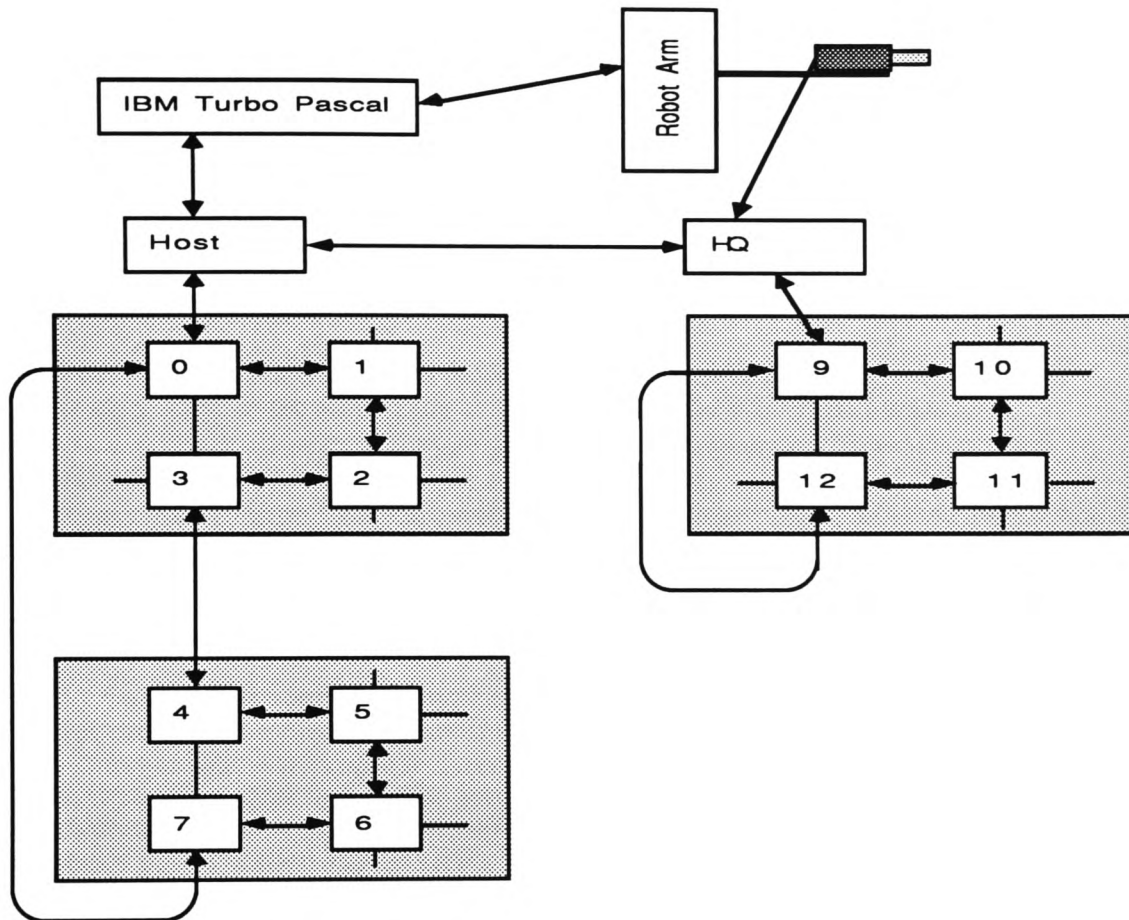


Figure 2 - Shows the initial system developed. The CCD camera is connected directly to the Harlequin frame grabber / graphics board. Transputers 0-7 hold the workspace model while transputers 9-12 are responsible for feature extraction.

Data to be sent from one processor to another is packaged into a one-dimensional array (that is a list of numbers). The first element in the array indicates the destination address of data, the second the source address of data and the third the quantity of data being sent. The sending processor uses the first two pieces of information to determine the best route to the required destination. This communication packet is depicted in figure 3.

To	From	Amount Of Data	No Of Sets(n)	Prob	Op-Code	X (1)	Y (1)	Z.Start (1)	Z.End (1)	-----	X (n)	Y (n)	Z.Start (n)	Z.End (n)
----	------	----------------	---------------	------	---------	-------	-------	-------------	-----------	-------	-------	-------	-------------	-----------

Figure 3 - Format For Passing Data From Sensor To Model.

Harlequin

- 1) Obtain image from CCD camera.
- 2) Threshold image and remove noise.
- 3) Project image through workspace.
- 4) Build view of workspace model from partial views provided by network transputers and analyse.

Host

- 1) Provide interface between user and system.
- 2) Provide interface between Harlequin and network.

Transputers in network (0 - 7)

- 1) Update data structure with data received from Harlequin relating to projection.
- 2) Create a view of the workspace and send it to the the Harlequin for analysis.

Transputers in network (9 - 12)

- 1) Responsible for feature extraction.
- 2) Use the features extracted to facilitate object identification.

5. Implementation Principles

To enable the processes, which were coded in OCCAM, to run concurrently on several processors a means of sending data from processor to processor is required. To keep communication overhead to a minimum it is important that this facility sends data via the best possible route around the network and that the number of times a process needs to communicate with other processors is optimised. It is also important to ensure that processor power is used as effectively and efficiently as possible.

To help meet these criteria the following rules were developed:-

- 1) Start the processors doing useful work as soon as possible and keep them doing useful work for as long as possible.
- 2) Data transfer between processors should be via the best possible route around the network. The best route is not necessarily the shortest as "traffic jams" might develop along certain paths. Obviously, it would be quicker if these were bypassed, but if there is no possibility of deadlock occurring [4] and the probability of "traffic jams" developing is low then they are best ignored.
- 3) Data transfer should be kept at an optimum level and when transfer is required it should be given priority over other tasks (this enables the receiving transputers to make use of the transferred information as soon as possible).

The optimum level of information transfer depends on two factors. First, it may be quicker to duplicate processes on more than one processor than to send information from processor to processor. Second, if one processor is generating information to be shared by other processors in the network then the information can be passed at one of three stages. These three stages being:-

- (i) after all the information has been generated,
- (ii) after a given amount of information has been generated,
- (iii) as and when the information is generated.

While the choice of option will depend on the system being implemented the correct decision is crucial to the efficient execution of parallel processes. When making a decision the system designer should bear in mind rule 1. In the system being described, option (ii) proved to be the best choice.

Each transputer in the network therefore has the following processes executing concurrently:-

(1) a 'get info' process to receive data from both nearest clockwise and anti-clockwise transputers.

(2) a multiplexer to send data to the nearest clockwise transputer. This data may have been processed by the sending transputer, or the sending transputer may be acting as a link in the chain. (The multiplexer is required to collect data from the 'get info' and 'process request' routines and pass it on via a single connection.)

(3) a multiplexer to send data to the nearest anti-clockwise transputer. Again this data may have been processed by the sending transputer, or the sending transputer may be acting as a link in the chain.

In addition to the above, the first transputer in the network has the facility to communicate with the host transputer. Figure 4 shows the processes that run concurrently on the first transputer, while the other transputers in the network have all but the multi-world process running concurrently on them.

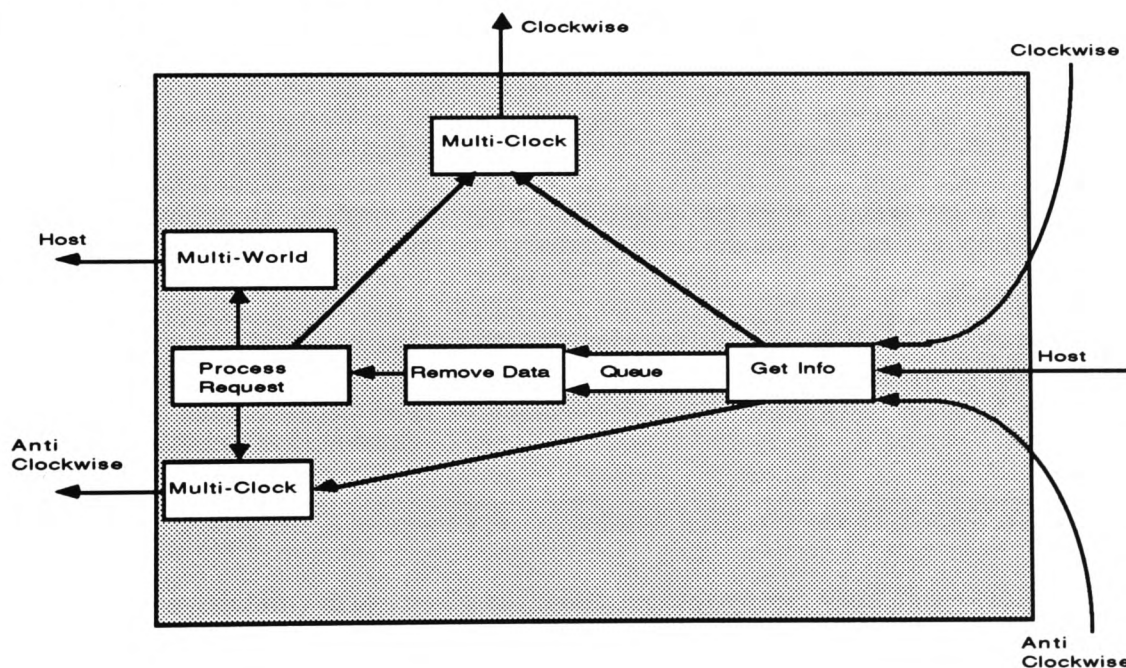


Figure 4 - A diagrammatical view of the processes running concurrently on the first processor in the network.

6. Testing The System

After the system outlined had been implemented, the next stage was to test its ability to locate and recognise objects, from a predefined object set, within a prototype workspace. To enable testing, the processes running on the network were extracted. Extraction puts all the code necessary for the loading, communication and execution of processes into a file that is 'bootable' by a program running on host PC [5].

A Turbo Pascal program was written to interface the robot arm with the transputer network (the Pascal program running on the host PC). The Pascal program was responsible for driving the robot arm so that the camera repeatedly scanned the workspace until an object was detected. Once an object was detected the scanning was stopped and the orientation of the camera adjusted so that the object became located in the centre of the camera's field of view.

The 2-D image was then processed to remove noise and its projection through the workspace calculated. The information associated with projection was then packaged before being sent to the transputers that held the associated parts of the model.

The camera was then raised to a position vertically above the object's location. The robot then moved the camera so that it scanned through an arc between the horizontal and vertical plane until the object was relocated. As with the first view the camera was adjusted automatically so that the object became located in the centre of the camera's field of view. This 2-D image was then processed as before.

In parallel with the object location process a second procedure attempts to identify the objects. The identification process matches features extracted from each object with a database of predefined object features. A statistical analysis is then used to determine the identity of each located object.

The next stage was to use the workspace model produced to determine the intersection of the two projections. This intersection was then used to determine the three two-dimensional views of the object within the workspace. The centroid of each view was then calculated and used to give the location of the object.

The system testing revealed that while on the whole the system worked well there are several areas of design which need refining. If objects within the workspace are occluded then in certain cases the intersecting views obtained from the different sensors will produce incorrect results. This can be overcome by better selection and placement of sensors, but a more satisfactory solution will be the inclusion of A.I. techniques to overcome any anomalies.

An additional problem is that the workload of the transputers is not as balanced as it might be. This leads to valuable processor power being wasted. A more dynamic processor allocation strategy will therefore be developed.

The object identification module also requires further work in order that a wider set of objects can be identified. At present only objects that come from sets with a fair degree of dissimilarity can be recognised. This increase in recognition ability will be achieved by increasing the range of features extracted and by improvements to the way feature importance is calculated.

7. Discussion

The availability of an easily adaptable sensing system would help to motivate an expansion in the use of robots in manufacturing industries. One factor that limits the adaptability of current systems is their lack of modularity. What is required is a system where by manufacturing industries can acquire different modules to meet their changing requirements without having to purchase a complete new system.

The nature of the task under consideration suggests that an extendible parallel architecture would be better suited to providing the processing requirements of such a system, than a traditional Von Neumann architecture. In general, one of the difficulties in using parallel architectures is that it is difficult to recognise inherent parallelism in problems. However, in the present task the parallel operation of modules is self evident.

The design of the transputer makes it a cost effective processor to meet the computational requirements of such a system. The transputer allows processor power to be expanded and applied, when and where needed. For example, if it were required to double the size of the prototype work cell without loss in resolution this could easily be achieved by doubling the number of network transputers from 8 to 16. Similarly, if an

extra camera were to be added to the system (as in figure 5) then that would simply require the addition of an extra Harlequin board.

One disadvantage of using a transputer system however, is the lack of shared memory. In the object identification module, for example, it would have been simpler and more efficient to have a single copy of the image being shared by several processors.

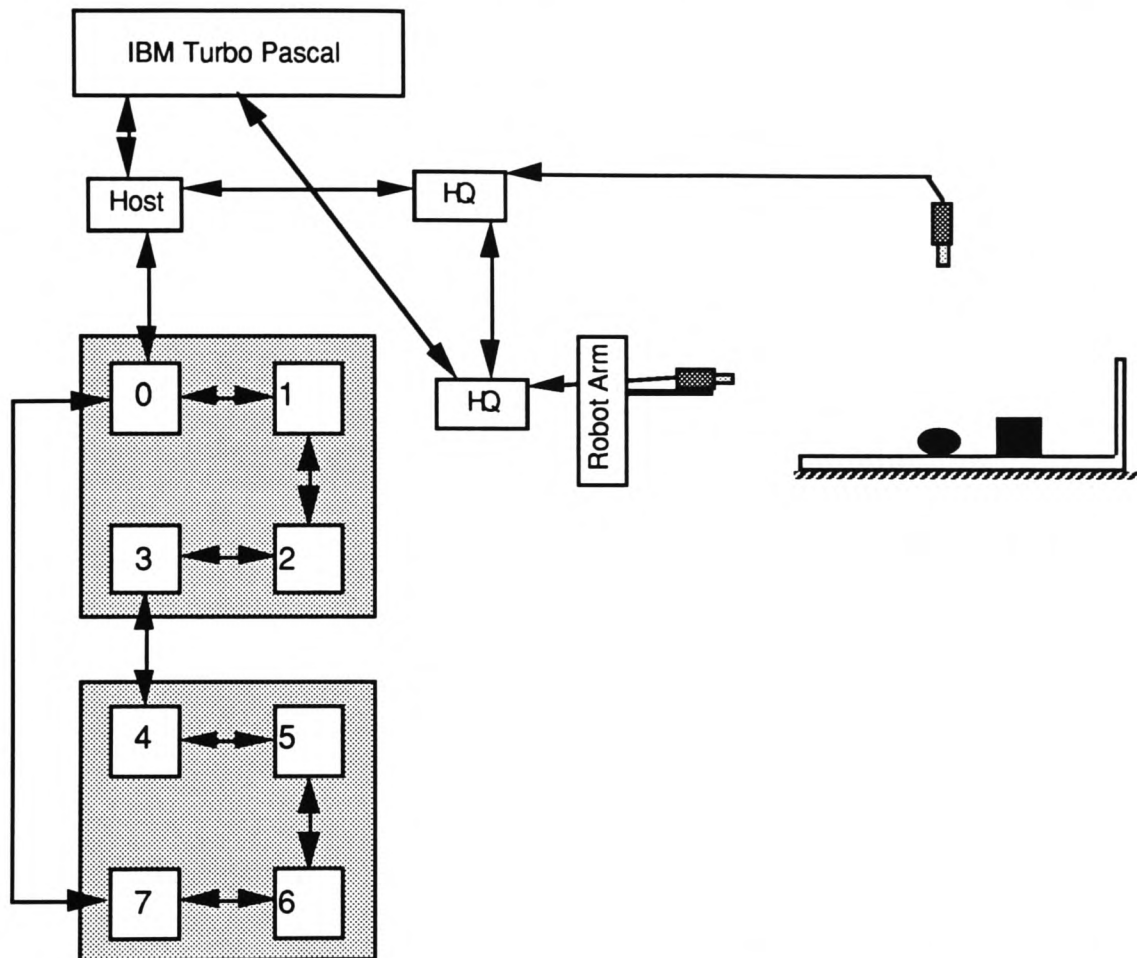


Figure 5 - Shows a system with two cameras but with no object identification modules.

8. Conclusions

This paper has outlined a prototype of a transputer based modular sensing system. The authors believe that such a system could form the basis for an adaptable system that could be configured to meet the requirements of a wide variety of applications in automated manufacturing.

9. References

- [1] Ware,J.A., Davies,R.A., Roberts,G., Williams, J.H.,"A method for storing a 3-D workspace", Draft version available from Dept. Of Mathematics and Computing, The Polytechnic Of Wales, Pontypridd, Mid Glamorgan, CF37 1DL.
- [2] Chien,C.H., and Aggarwal,J.K., 1989, "Model Construction and Shape recognition from occluding Contours", IEEE Transactions on pattern analysis and machine intelligence, Vol. 11, No. 4, pp 372-389.
- [3] Martin,W.N., and Aggarwal,J.K., 1983, "Volumetric descriptions of objects from multiple views", IEEE transactions on pattern recognition and machine intelligence, VOL PAMI-5 No 2.
- [4] Ditel,H.M., 1984, "An introduction to operating systems", Addison-Wesley Publishing Company, London.
- [5] Inmos, 1987, "Transputer development system", Prentice Hall International (UK) LTD.

A5.2 Paper 2

Ware J.A., Roberts G., Davies R.A. (1991) "Determining the location and identity of components in an A.M.P. environment", IEE Third International Conference on Software Engineering for Real Time Control, Cirencester, September, pp. 109 - 114.

The paper also appears in the proceedings of VISION '90, Society of Manufacturing Engineers, held in Detroit, Michigan, November 1990, pp 9:1 - 9:10 under the title "Building and storing a model of a robot's workspace using a series of 2-D images."

A summary of the paper was also presented at the IEE colloquium on Computer Image Processing and Plant Control, at The IEE, Savoy Place, London, 18'th May 1990, under the title "The location of components in an automatic manufacturing process."

DETERMINING THE LOCATION AND IDENTITY OF COMPONENTS IN AN A.M.P. ENVIRONMENT

J.A.Ware, R.A.Davies, G.Roberts.

The Polytechnic Of Wales, UK.

INTRODUCTION.

This paper describes a transputer based system that facilitates the identification and manipulation of components in an automatic manufacturing process (A.M.P.). As a prerequisite to component identification and manipulation their location within the robot's workspace must be determined. In the system to be described, the location of objects is achieved by building a model of the workspace using information provided by sensors located within the robot's workspace. This model is then used to determine the location of components within the workspace.

The system has been constructed using a modular approach (1). In particular the hardware, software and the user-interface have been modularised. The modules are loosely coupled and thus changes can be made to individual modules independently of the other two (2). This modularity means that the sensors that provide the information about the workspace are independent of the algorithms that make use of the information. That is, the sensors have no knowledge as to when or how the information they provide is to be used. Similarly, the algorithms that make use of the data have no knowledge as to the provider of the data.

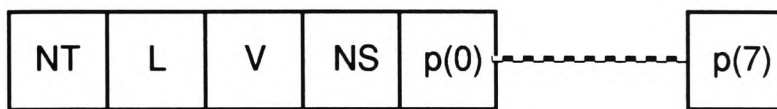
As a prerequisite to model construction three problems had to be addressed. First, the information extracted from different cameras is generally at different resolutions. Second, the representation of 3-D space requires large amounts of computer memory. Third, the production of the 3-D model, particularly when a large number of cameras are involved requires a substantial amount of processor time. The first two prerequisites were addressed using a data structure that allowed compact data storage, while the final prerequisite was met by identifying parallel aspects of the processing and implementing them on a network of processors.

DEVELOPMENT OF A SUITABLE DATA STRUCTURE.

In the search for a suitable data structure to represent the workspace consideration was first given to octrees. The octree is a hierarchical data structure which has two principal uses. In the first, it is used to represent a 3-D component, the component's description being decomposed into smaller and smaller parts. In the second, it is used to represent the

breakdown of 3-D space into smaller and smaller voxels. In the case cited it is a 3-D space that requires breaking down, allowing algorithms designed to investigate the workspace to home in on different parts of the workspace.

The octree as its name suggests consists of a root, branches and leaves. The root contains a representation of the whole of the workspace which is subsequently broken down into eight octants. Each octant, which is represented by a branch of the tree, is broken down into a further eight octants. This decomposition continues until the resulting octants are either full or empty. In the context of this paper a full voxel is a voxel which is wholly or partially filled with a component. An empty voxel is one that is completely empty. The whole tree may be represented in the form of a table. Each branch taking up one entry within the table. A typical table entry is as follows:-



NT - Node Type. A node (or entry in table) represents either a branch or twig in the tree. A branch is a node that can be broken down into sub-branches while a twig contains the information related to eight of the lowest level octants. (The introduction of a twig table entry is a modification of the common branch and leaf approach and is explained later.)

L - Level. Each branch in the tree is given a level indicator to indicated its height in the tree. The root branch is at the lowest level and the eight octants that it breaks down into are at level one. This process of division into levels until the required resolution is obtained.

V - Voxel. Each node is also given a voxel number which when combined with the level indicator uniquely identifies the voxel which the table entry represents.

NS - Number Set. In order to facilitate easy geometric calculations each branch table entry contains a count of the number of lowest level voxels that are full in that branch.

p(0) - p(7). Each table entry also contains eight indicators related to the eight sub-octants related to it. In the case of a branch entry the indicators act as pointers to the table entries associated with the sub-octant. In the case of a twig entry the indicators are set to either a one or a zero. If indicator is set to a one then the lowest level voxel is full and if the indicator is zero then the lowest level voxel is empty.

As stated, the introduction of a twig table entry is a modification of the common branch and leaf approach. As can be seen from figure 1 and 2 the saving in table entries with this new approach is considerable in the case of lower resolution octrees. However, this is

not so for higher resolution octrees. This is due to the fact that with higher resolution octrees a larger proportion of the original table entries are branch nodes. The figures represent those obtained when generating an octree for a pyramid whose height and length of base are equal to the height and length of the workspace respectively.

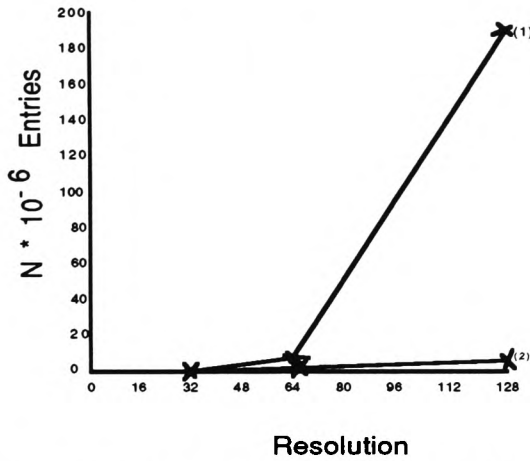


Figure 1 - [1] Maximum number of entries in table using standard branch and node approach.
[2] Maximum number of entries in table using branch and twig approach.

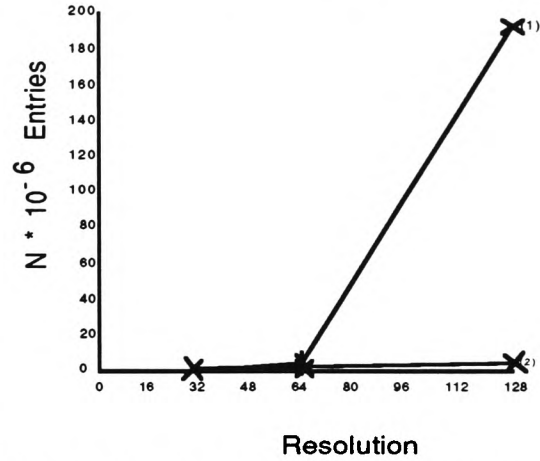


Figure 2 - [1] Number of entries in table at end using standard branch and node approach.
[2] Number of entries in table at end using branch and twig approach.

As can be seen from figure 3 the computational time (which represent the CPU utilisation when algorithms were run on a VAX 8650) for both these methods was considerable. This meant that unless a further modification to the data structure could be found, that would reduce computational requirement, octrees would be unsuitable for the intended application.

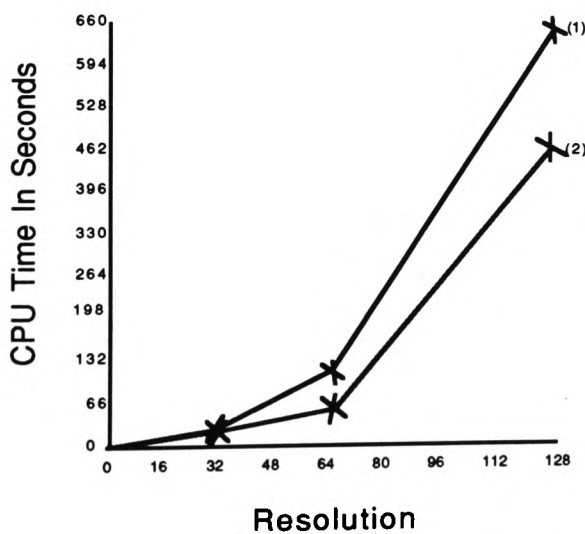


Figure 3- [1] Time taken to build table using standard branch and node approach.
[2] Time taken to build table using branch and twig approach.

Modification To Octree. Li and Telfer (3) have produced a modification to the quadtree hierarchical data structure (4) for storing binary images. This modified quadtree gives a considerable reduction in the number of nodes when compared to the original quadtree. The authors of this paper have found that a comparable saving is obtained by applying a similar modification to the octree data structure.

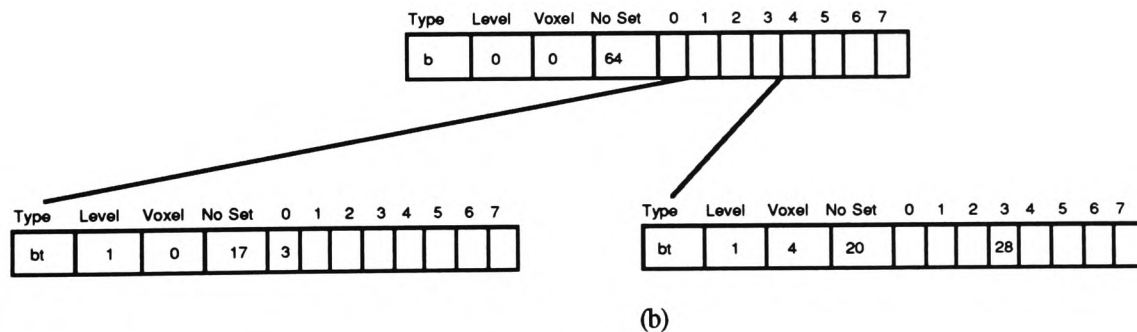
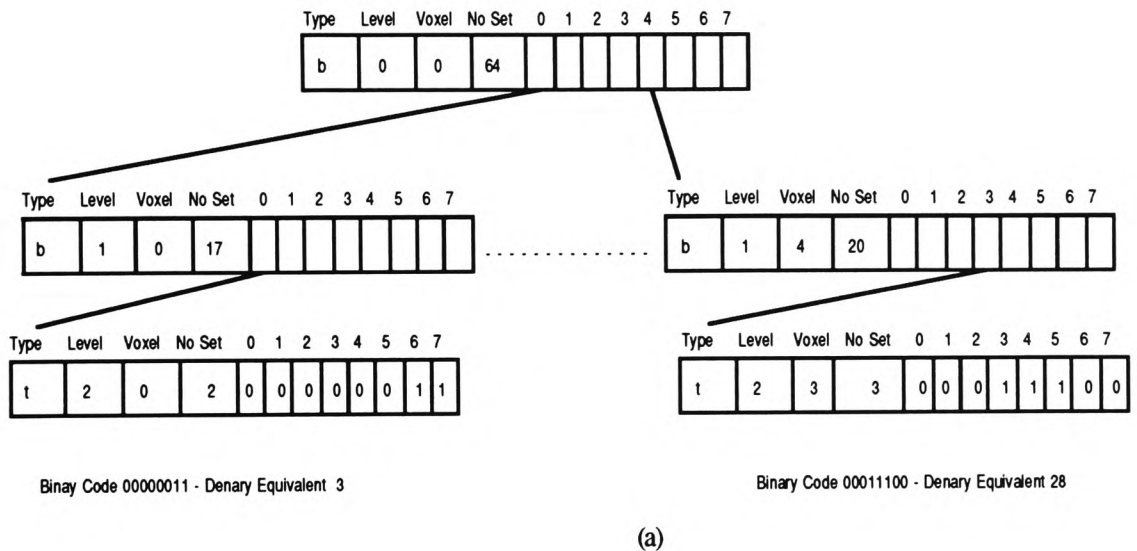


Figure 4 - [a] An eight bit binary code and its denary equivalent.
 - [b] Eight bit binary codes stored, as their denary equivalent, at a higher level in the tree.

As detailed for the 'branch and twig' method, the lowest level voxel in the primitive octree can be represented by a single binary digit. The next highest level can thus be represented by an eight bit binary code or its denary equivalent, figure 4(a). These binary codes (or their denary equivalent) can then be stored in the 'twig' pointers at the next level up the tree, figure 4(b). Using this refinement to the 'branch and twig' approach further savings were obtained (typically 20%+). Unfortunately, this increase in storage efficiency was matched by an increase in processing requirements. Thus, even with the above modifications, the use of octrees is not viable for the current application.

Another limitation of using hierarchical data structures, such as the octree, for representing information related to three-dimensional space is that they are regular structures while most three dimensional spaces are irregular. Representing non-regular space with a regular data structure can lead to both inefficient storage techniques and long computation times.

Run-Length Encoding. In the light of experience gained from octrees attention turned to non-hierarchical data structures. One such structure which seemed to hold potential was a run-length encoding technique devised by Martin and Aggarwal (5). The data structure was based on splitting components into "volume segments". Each volume segment represented part of the component where each part was parallel to the coordinate axis. This was achieved by segmenting the components into planes parallel to the $z=0$ plane, figure 5(a). Each z -plane was further split into lines with edges parallel to the x,y axis, figure 5(b). These lines were either completely empty or contained one or more volume segments (rectilinear parallelepipeds).

The structure then maintained an ordered pair of values for each volume segment; the values being the y -coordinates of the segment endpoints and were ordered into lists of segments having the same x -coordinate, i.e., colinear. The x -level lists were then coalesced into z -level, ordered lists by common z -coordinate, i.e., coplanar. From the top down this structure was a set of "planes" parallel to the $z=0$ plane, that were ordered by x -value. Each "line" comprised a set of disjoint segments that were ordered by endpoint y -values. Figure 6 shows a schematic of a volume segment structure using linked lists to order the various components.

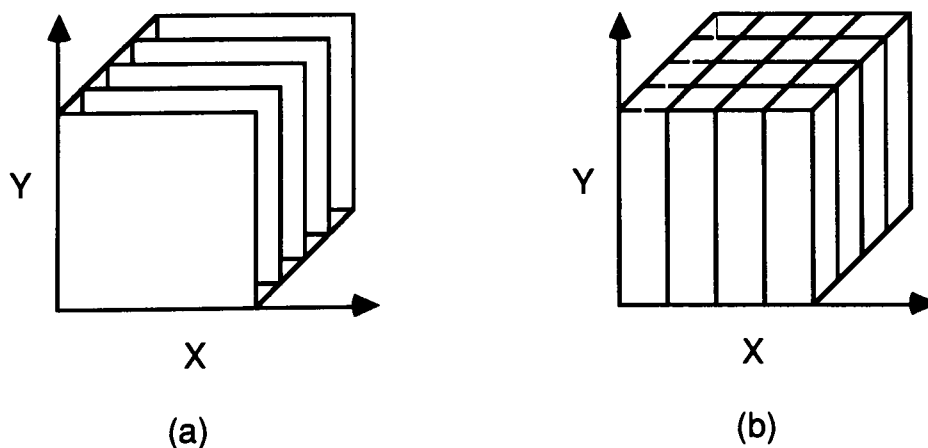


Figure 5 - [a] component (or volume) segmented into planes
- [b] planes split up into lines

In a general situation the primary advantage of this structure is that the process of determining whether an arbitrary point is within the surface boundary consists of a simple search of three ordered lists: select a "plane" by z-coordinate; select a "line" by x-coordinate; and finally, check for inclusion of y-coordinate in a segment.

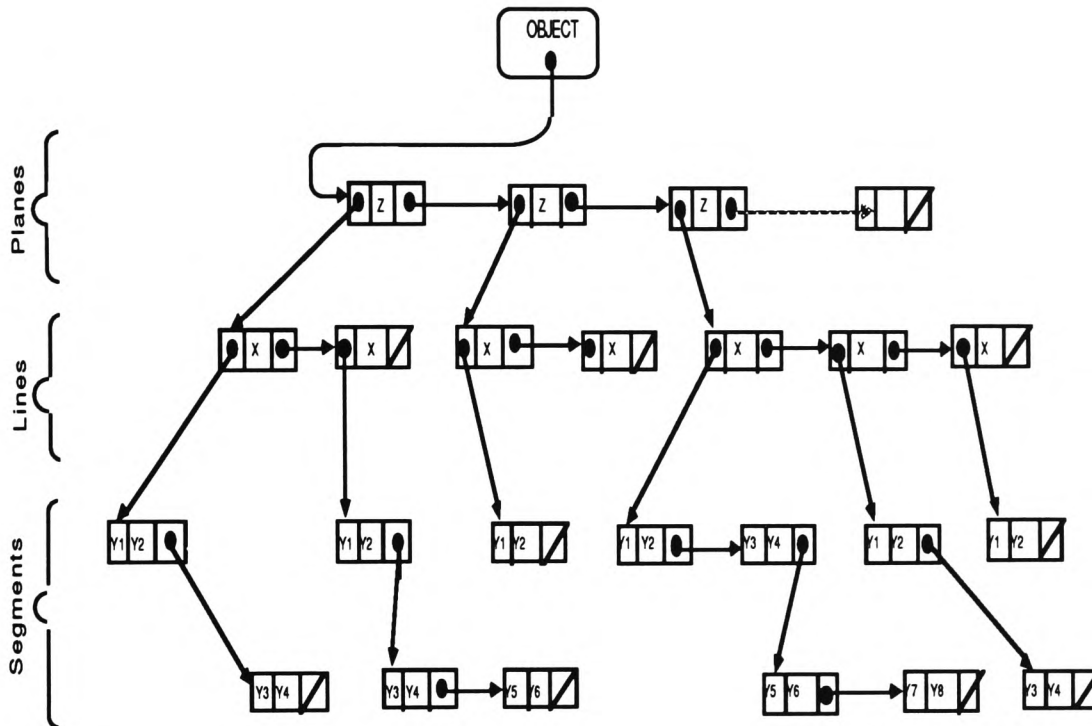


Figure 6 - Data structure schema for the volume segment representation.

In the present work, a modification of the structure proposed by Martin and Aggarwal (5) has been developed and implemented. This modified structure has been termed 'partial run-length encoding'. With partial run-length encoding a table is used to represent one of the planes (either x,y or y,z or x,z). Each entry in the table represents a column or row of voxels perpendicular to the chosen plane.

If all the voxels in the column or row are empty then the corresponding entry in the table is zero. If there are any full voxels in the column or row then the corresponding table entry is set to point to the appropriate entry in a second table holding details about full voxels. Each entry in this second table contains details related to non-empty voxels. If two or more adjacent voxels have the same details then only one table entry is required to hold all the relevant information. As an example consider how the component in figure 7 could be represented using partial run-length encoding.

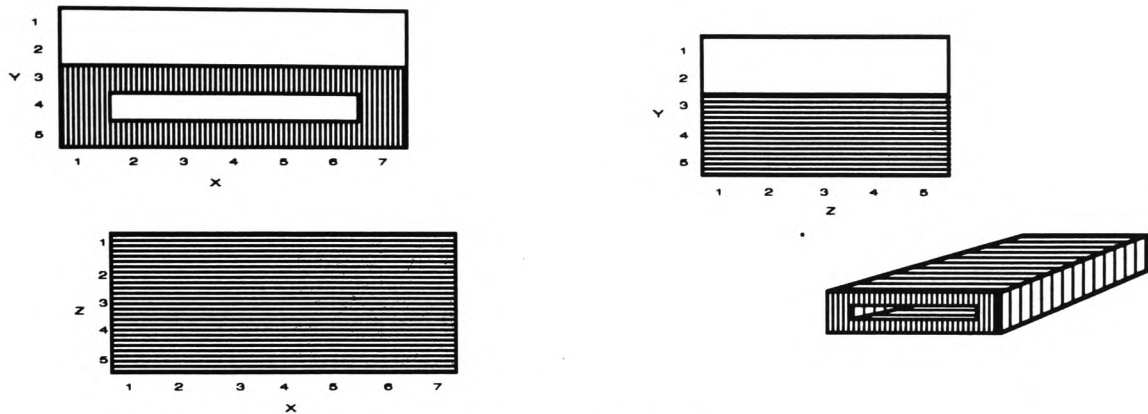


Figure 7 - Shows a component with three end on views.

1	0	0	0	0	0
2	0	0	0	0	0
3	1	2	3	4	5
4	6	8	10	12	14
5	16	17	18	19	20
	1	2	3	4	5
			Z		

Table 1 - Represents the x,y plane. Each entry in the table represents a row of voxels perpendicular to the x,y plane.

Pos	X-Start	X-End	Last	Next
1	1	7	0	0
2	1	7	0	0
3	1	7	0	0
4	1	7	0	0
5	1	7	0	0
6	1	1	0	7
7	7	7	6	0
8	1	1	0	9
9	7	7	8	0
10	1	1	0	11
11	7	7	10	0
12	1	1	0	13
13	7	7	12	0
14	1	1	0	15
15	7	7	14	0
16	1	1	0	17
17	7	7	16	0
18	1	7	0	0
19	1	7	0	0
20	1	7	0	0

Table 2 - Contains details related to non-empty voxels.

The partial run-length encoding method has been extended to provide automatic production of intersections of the projected views from the various sensors. This has been achieved by extending the second table to include a certainty factor indicating the likelihood that a voxel (or the set of voxels represented by one entry in the table) is empty or full. When information from a sensor is added to the composite 3-D model of the robot's workspace, then it is necessary to check for any intersection of the new view with those views already known. If there are intersections, then parts of the partial run-length encoding may have to be concatenated while others may become fragmented.

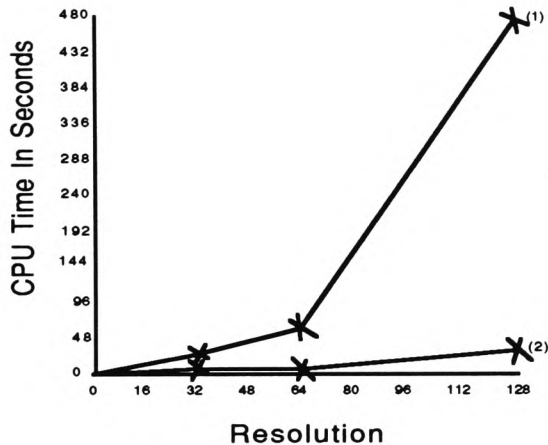


Figure 8 - [1] Time taken to build table using octree branch and twig approach.
[2] Time taken to build table using partial run-length encoding.

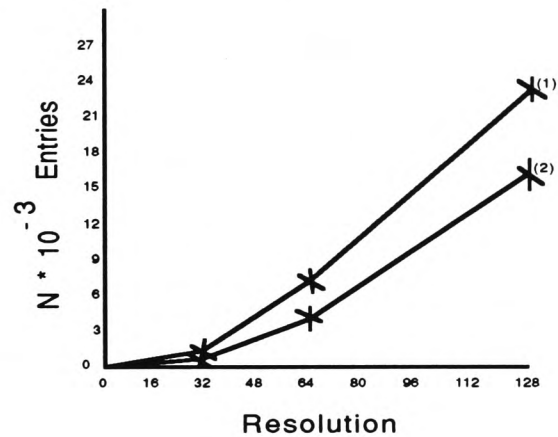


Figure 9 - [1] Table entries using Octree branch and twig approach.
[2] Table entries using partial run-length encoding.

As can be seen from figures 8 and 9 the advantage of partial run-length encoding over the octree method in both computational time and table entries is considerable. This superiority is not only in terms of computational and memory requirements but also in the ease with which it lends itself to the representation of non-regular workspace. The timings obtained clearly indicate that a partial run-length encoding data structure is well suited to the present application. The next stage in the development of a 3-D modelling system was to take the partial run-length encoding technique and adapt it to run on an array of transputers.

MODELLING SYSTEM DEVELOPED.

Once a suitable data structure had been developed and implemented the next stage was to incorporate it into the modelling system. The modelling system developed, by Ware et al (6), allows the following functions to be carried out:-

- (1) Update the model
- (2) Display views of the workspace model
- (3) Modify Workspace Characteristics
- (4) Modify Sensor Characteristics

OBJECT IDENTIFICATION.

Once components have been located within the robot's workspace the next stage is to recognise what the components are and in what position they are lying. It is assumed that the components are part of a known finite set of components. A component can be described in terms of a number of characteristics, or "features". These features can be categorised into two groups: those that represent the component's position in space and those that describe the geometry of the component. At any given time an component is at a specific location in space, it is oriented in a specific direction, and it may be moving in a certain direction. To fully define a component's position, three basic features must be determined: the component's location, its orientation, and its motion. In the present application components will be stationary and therefore only their location and orientation need to be determined.

As already stated components can be described in terms of their features, and thus to facilitate component recognition a set of describing features is produced for all components within the component set. In cases in which several component features must be measured in order to identify it, a simple factor weighting method is used to consider the relative contribution of each feature to the analysis. Each feature is compared with a standard for the goodness-of-fit measurement. Features that are known to be the most likely indicators of a match being more heavily weighted than others. A weighted total goodness-of-fit score is then determined to indicate the likelihood that the component has been correctly identified. The features used in the prototype are:-

- (1) Ratio of the length of two lines which intersect at right angles at the centre of gravity of component; one line being the axis of least moment of inertia of the component.
- (2) Ratio of perimeter length squared to component area.
- (3) Euler Number
- (4) Ratio of convex hull perimeter length squared to convex hull ratio.

As feature values are extracted from standard test components they are added to a set of component features. (Each entry in the component feature set consists of a feature type together with the mean and standard deviation of several measurement of that feature.) As components within the component set change so will the usefulness of the various features extracted. For example, if the component set contained various shaped assembly pieces then the average degree of roundness of their edges might give a good indication as to their identity. However, in another case, such as a set of various sized disks, then

knowing the average degree of roundness of edges would not help in component identification. To allow for this change in feature usefulness, each feature value must have a weighting factor calculated for it. Thus, for each set of components those features best suited to uniquely identifying components will have greatest weighting.

PARALLEL IMPLEMENTATION OF SYSTEM.

Parallel processing involves either splitting the application or process to be performed into several sub-processes and performing these on different processors concurrently, or splitting the data that is to be processed between a number of processors and executing multiple copies of the process simultaneously. It should be noted that some algorithms are inherently sequential and, even for those that are not, it is often a non-trivial task to adapt sequential algorithms for efficient execution on parallel machines. Ware and Kidner (7) found converting from sequential to parallel processing may necessitate a completely new approach, since a simple transformation of the sequential code may prove impossible. Conversely there are occasions when inherent parallelism in the problem can give excellent results with only minor changes to the sequential algorithm.

The processor chosen for the parallel implementation of the system described was the transputer. There are several chips in the transputer family, but the two most important ones are the T414 and T800, both of which rate 10 MIPS at 20 MHz. The T414 has 2K of on board RAM, whilst the more powerful T800 has 4K, as well as its own on-board floating point unit, which works in parallel with the main CPU and is capable of 1.5 Mflops. In the prototype implementation to be described, the host transputer is a T800, whilst the network transputers are T414s. Overall performance would be substantially increased if all the network transputers were substituted for T800s.

Transputers can be linked together to form a network of processors such that the workload of the application can be distributed across it. Not only does such a system offer a great deal of computing power, but its performance can be extended as required, by adding more transputers to the network. Algorithms that are to be executed on the network may be implemented in a number of high level computer languages. However, if maximum efficiency is to be achieved they should be written in OCCAM, a programming language designed specifically for the transputer.

The model of the prototype workspace was held on eight transputers, each storing the model for one eighth of the workspace. A CCD camera, connected to a transputer based frame grabber and image processing board, was used to obtain multiple views of the workspace. The transputer configuration for the prototype system is illustrated in figure 10.

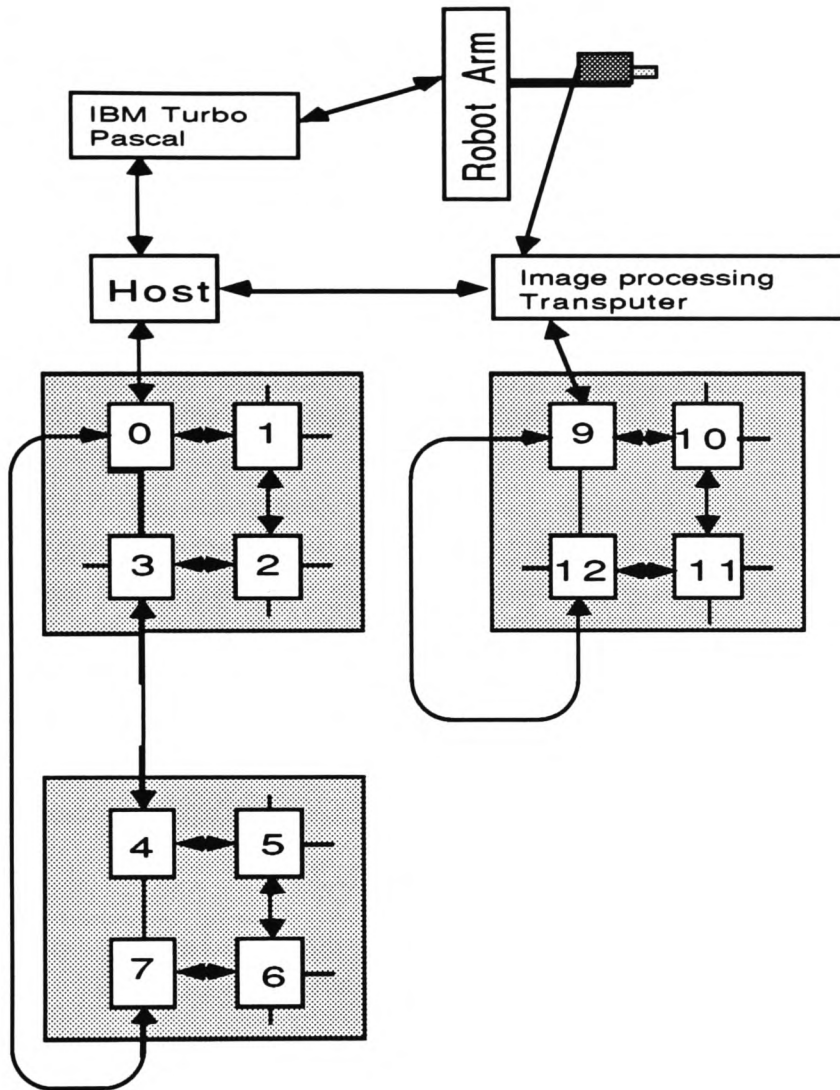


Figure 10 - Transputer Layout For Prototype System. Copies of the algorithm to maintain the workspace model are executed concurrently on processor 0-7, while processors 9-12 execute the object identification algorithms.

To enable the OCCAM-encoded processes to run concurrently on several processors, a means of sending data from processor to processor is required. To keep communication overhead to a minimum it is important that this facility sends data via the best possible route around the network and that the number of times a processor needs to communicate with other processors is optimised. It is also important to ensure that processor power is used as effectively and efficiently as possible. To help meet these criteria the following rules were developed :

1) The processors should be kept 'busy' doing useful work for as long as possible. To facilitate this, whenever data transfer is required it should be given priority over other tasks (this enables the receiving transputers to make use of the transferred information as soon as possible).

2) Data transfer between processors should be via the best possible route around the network. This is not necessarily the shortest as 'traffic jams' might develop along certain paths.

3) Data transfer should be kept at an optimum level. The optimum level of information transfer depends on two factors. First, it may be quicker to duplicate processes on more than one processor, than to send information from processor to processor. Second, if one processor is generating information to be shared by other processors in the network, then the information can be passed at one of three stages. These stages being - after all the information has been generated, after a given amount of information has been generated, as and when the information is generated.

To facilitate communication between the transputers in the network, each transputer has, in addition to the main algorithm being executed, a set of 'communication procedures'. These procedures allow the processor to receive and send data in the required manner. Data to be sent from one processor to another is packaged into a one-dimensional array. The first element indicates the destination address, the second indicates the source address, whilst the third represents the quantity of data being sent. The sending processor uses the first two pieces of information to determine the best route to the required destination. This communication packet is depicted in Figure 11.



Figure 11 - Format For Passing Data From Sensor To Model.

SYSTEM TESTING AND CONCLUSIONS.

After the system outlined had been implemented, the next stage was to test its ability to locate components within a prototype workspace. To enable testing, the algorithms running on the network were extracted. Extraction puts all the code necessary for the loading, communication and execution of algorithms into a file that is 'bootable' by a program running on host PC.

A Turbo Pascal program was written to interface the robot arm with the transputer network (the Pascal program running on the host PC). The Pascal program was responsible for driving the robot arm so that the camera repeatedly scanned the workspace until a component was detected. Once a component was detected the scanning was stopped and the orientation of the camera adjusted so that the component became located in the centre of the camera's field of view.

The 2-D image was then processed in order to remove noise and its projection through the workspace calculated. The information associated with projection was then packaged before being sent to the transputers that held the associated parts of the model.

The camera was then raised to a position vertically above the component's location. The robot moved the camera so that it scanned through an arc between the horizontal and vertical plane until the component was relocated. As with the first view the camera was adjusted automatically so that the component became located in the centre of the camera's field of view. This 2-D image was then processed as before.

The workspace model produced could now be used to determine the intersection of the two projections. This intersection enabled the three, two-dimensional views of the component to be determined. The centroid of each view was then calculated and used to give the location of the component. Running in parallel with the object location process were the procedures written to identify the component.

The testing was repeated with several components in the workspace and with additional cameras added to the system.

The system testing revealed that while on the whole the system worked well there are several areas of design which need refining. If objects within the workspace are occluded then in certain cases the intersecting views obtained from the different sensors will produce incorrect results. This can be overcome by better selection and placement of sensors, but a more satisfactory solution will be the inclusion of A.I. techniques to overcome any anomalies.

An additional problem is that the workload of the transputers is not as balanced as it might be. This leads to valuable processor power being wasted. A more dynamic processor allocation strategy will therefore be developed.

The design of the transputer makes it a cost effective processor to meet the computational requirements of such a system. The transputer allows processor power to be expanded and applied, when and where needed. For example, if it were required to double the size of the prototype work cell without loss in resolution this could easily be achieved by doubling the number of network transputers from 8 to 16. One disadvantage of using a transputer system however, is the lack of shared memory. In the object identification module, for example, it would have been simpler and more efficient to have a single copy of the image being shared by several processors.

REFERENCES.

1. Ware,J.A., Roberts,G., Davies,R.A., Miles,R., Williams J.H., 1990, "A modular sensing system for robotic control", Proceedings of the second international conference on applications of transputers, Southampton, England.
2. Ware,J.A., Roberts,G., Davies,R.A., 1991, "A user interface for robotic sensing systems: A multi-faceted approach", IEE colloquium on HCI: Issues for the factory, London, England.
3. Li,Z., and Telfer,D., 1989, "Primitive quadtrees and type code quadtree approaches for the representation of binary regions", IEE colloquium on pattern recognition for binary images, London, England.
4. Samet,H., 1984, "The quadtree and related hierarchical data structures", ACM Computing Surveys, Vol.16, No.2.
5. Martin,W.N., and Aggarwal,J.K., 1983, "Volumetric descriptions of objects from multiple views", IEEE transactions on pattern recognition and machine intelligence, VOL PAMI-5 No 2.
6. Ware J.A., Roberts G., Davies R.A., Williams J.H., 1990, "Building and storing a model of a robot's workspace using a series of 2-D images", SME VISION '90, Michigan, USA.
7. Ware J.A., Kidner D.B., "A parallel implementation of the Delaunay triangulation within a transputer environment", EGIS'91, Brussels, Belgium.

A5.3 Paper 3

Ware J.A., Roberts G., Davies R.A., Williams J.H. (1990) "A multiple camera system to facilitate object manipulation within a robot's workspace", The Second Symposium on Personal Computers in Industrial Control, Warren Spring Laboratory, Stevenage, November, pp 81-88.

A summary of the paper was also presented at the IEE colloquium on Binary Image Processing - Techniques and Applications, at The IEE, Savoy Place, London, 8th March 1991, under the title "Determining the location of components in an A.M.P. environment."

A Multiple Camera System To Facilitate Object Manipulation Within A Robot's Workspace

J.A.Ware¹, R.A.Davies², G.Roberts¹, J.H.Williams³.

This paper describes the procedure whereby the information extracted from a series of 2-D camera images, provided by one or more cameras within a robot's workspace, is used to determine the location of objects within the workspace. This information regarding object location is a prerequisite to any object manipulation that must be performed by the robot arm.

This procedure forms a single module in a transputer based multi-module sensing system. The modular approach adopted allows changes to be made to both the data acquisition hardware and software independently of the other modules.

INTRODUCTION

An object can be described in terms of a number of characteristics, or "features". These features can be categorised into two groups: those that represent the object's position in space and those that describe the geometry of the object. To facilitate the manipulation of objects by a robot within its workspace the robot controller must know how the objects are positioned in space. At any given time an object is at a specific location in space, it is oriented in a specific direction, and it may be moving in a certain direction. To fully define an object's position, three basic features must be determined: the object's location, its orientation, and its motion. In the present application objects will be stationary and therefore only their location and orientation need to be determined.

As a prerequisite to model construction three problems had to be addressed. First, the information extracted from different cameras is generally at different resolutions. Second, the representation of 3-D space requires large amounts of computer memory. Third, the production of the 3-D model, particularly when a large number of cameras are involved requires a substantial amount of processor time. The first two prerequisites were addressed using a data structure that allowed compact data storage, while the final prerequisite was met by identifying parallel aspects of the processing and implementing them on a network of processors.

DATA STRUCTURE IMPLEMENTED

A full description of the development and implementation of this data structure used to hold the workspace model is detailed by Ware et al (1) and only a brief summary is included here for completeness. The data structure developed is based on a run-length encoding technique first implemented by Martin and Aggarwal (2). Their implementation was based on splitting objects into "volume segments". Each volume segment represented part of the object where each part was parallel to the coordinate axis. This was achieved by segmenting the objects into planes parallel to the $z=0$ plane, figure 1(a). Each z -plane being further split into lines with edges parallel to the x,y axis, figure 1(b). These lines were either completely empty or contained one or more volume segments (rectilinear parallelepipeds).

1 Dept Of Mathematics and Computing, The Polytechnic Of Wales, Pontypridd, Mid Glamorgan, CF37 1DL,

2 Dept Of Computer Studies , The Polytechnic Of Wales, Pontypridd, Mid Glamorgan, CF37 1DL,

3 School Of Electrical Electronic and Systems Engineering, University Of Wales, Cardiff, CF1 3YH.

The structure then maintained an ordered pair of values for each volume segment: the values being the y-coordinates of the segment endpoints and were ordered into lists of segments having the same x-coordinate, i.e., colinear. The x-level lists were then coalesced into z-level, ordered lists by common z-coordinate, i.e., coplanar. From the top down this structure was a set of "planes" parallel to the z=0 plane, that were ordered by x-value. Each "line" comprised a set of disjoint segments that were ordered by endpoint y-values. Figure 2 shows a schematic of a volume segment structure using linked lists to order the various components.

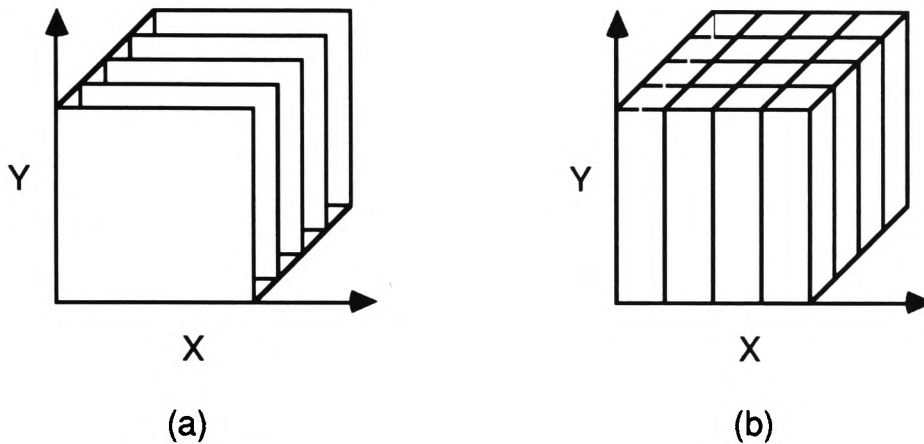


Figure 1 - (a) Object (or volume) segmented into planes
 - (b) Planes split up into lines

In a general situation the primary advantage of this structure is that the process of determining whether an arbitrary point is within the surface boundary consists of a simple search of three ordered lists: select a "plane" by z-coordinate; select a "line" by x-coordinate; and finally, check for inclusion of y-coordinate in a segment.

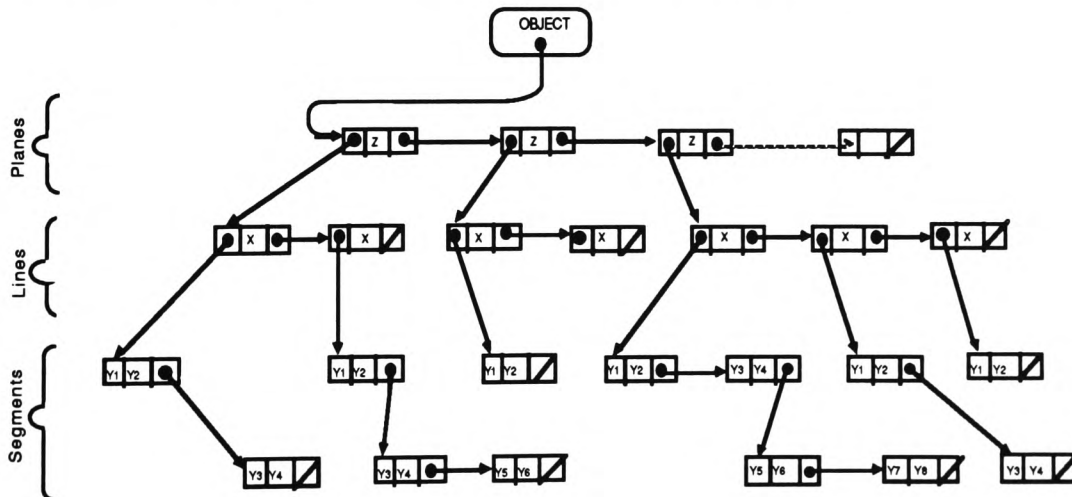


Figure 2 - Data structure schema for the volume segment representation.

In the present work, a modification of the structure proposed by Martin and Aggarwal has been developed and implemented. This modified structure has been termed 'partial run-length encoding'. With partial run-length encoding a table is used to represent one of the planes (either x,y or y,z or x,z). Each entry in the table represents a column or row of voxels perpendicular to the chosen plane.

If all the voxels in the column or row are empty then the corresponding entry in the table is zero. If there are any full voxels in the column or row then the corresponding table entry is set to point to the appropriate entry in a second table holding details about full voxels. Each entry in this second table contains details related to non-empty voxels. If two or more adjacent voxels have the same details then only one table entry is required to hold all the relevant information. As an example consider how the object in figure 3 could be represented using partial run-length encoding.

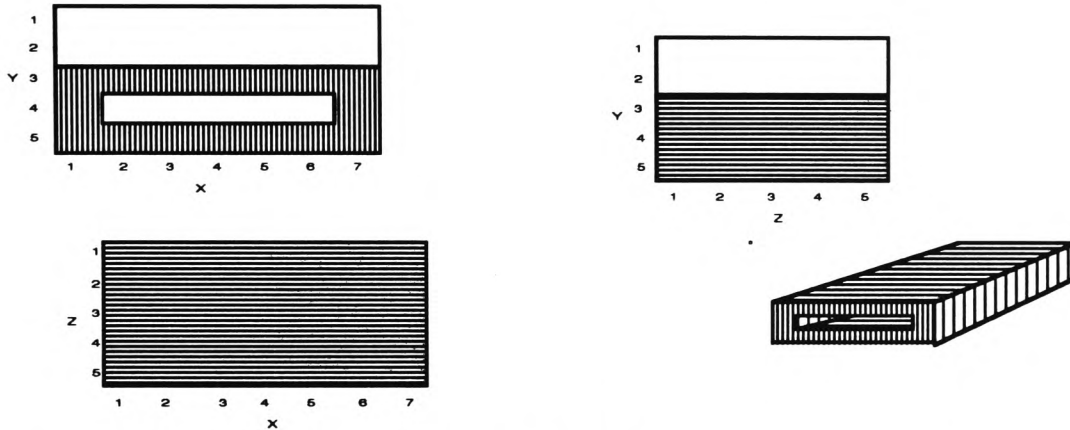


Figure 3 - Shows an object with three end on views.

	1	0	0	0	0	0
	2	0	0	0	0	0
Y	3	1	2	3	4	5
	4	6	8	10	12	14
	5	16	17	18	19	20
		1	2	3	4	5
						Z

(1)

Pos	X-Start	X-End	Last	Next
1	1	7	0	0
2	1	7	0	0
3	1	7	0	0
4	1	7	0	0
5	1	7	0	0
6	1	1	0	7
7	7	7	0	0
8	1	1	0	0
9	7	7	0	0
10	1	1	0	11
11	7	7	10	0
12	1	1	0	13
13	7	7	12	0
14	1	1	0	15
15	7	7	14	0
16	1	1	0	17
17	7	7	16	0
18	1	7	0	0
19	1	7	0	0
20	1	7	0	0

(2)

Table 1 - Represents the x,y plane. Each entry in the table represents a row of voxels.
 2 - Contains details related to non-empty voxels.

The partial run-length encoding method has been extended to provide automatic production of intersections of the projected views from the various cameras. This has been achieved by extending the second table to include a certainty factor indicating the likelihood that a voxel (or the set of voxels represented by one entry in the table) is empty or full. When information from a camera is added to the composite 3-D model of the robot's workspace then it is necessary to check for any intersection of the new view with those views already known. If there are intersections then this will lead to the possibility of parts of the partial run-length encoding needing to be concatenated while other parts will become fragmented.

MODELLING OF PROTOTYPE

Once a suitable data structure had been developed to hold the workspace model the next stage was to implement it for a prototype workspace. The workspace, which is rectangular (figure 4 shows the dimensions of the prototype workspace), is made of white plastic so that a good visual contrast exists between the cell and the objects placed within it.

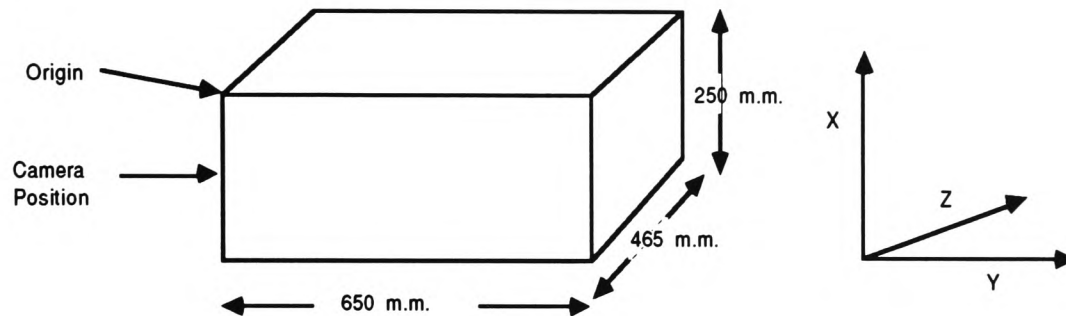


Figure 4 - The prototype workspace.

Initially the views of the cell required to produce the model were obtained using a CCD camera mounted on the end of a SCARA robot arm. The robot arm gave great flexibility in positioning the camera in relation to the workspace enabling multiple views to be obtained.

Although, in the first instance, only a single camera was used to obtain the required views, additional cameras were added later. Each time a new camera is added to the system or a characteristic of an existing camera is changed (such as the focal length of the camera) the camera has to be calibrated. The following describes the procedure for calibrating a camera.

Camera Calibration The calibration is carried out by placing a black disk of known radius at right angles to the camera. Using a disk ensures that there are no edges to be aligned with the camera. The distance between the camera and disk must equal the maximum distance between objects and the camera. An image is obtained and the optimum threshold calculated. After image thresholding, the number of pixels per millimetre in both the x^I and y^I (where x^I and y^I denote the coordinate system of the camera) directions are calculated. These values are subsequently used when calculating the dimensions of unknown objects from their imaged silhouettes.

The calibration procedure assumes that the camera is uniform in resolution over its imaging surface and that the silhouette obtained is a parallel projection of the object. The validity of the former assumption can be verified from the documentation supplied with camera. The latter assumption results in the calculated bounding volume of some objects being larger than they actually are. The degree of this inaccuracy in measurement depends on the distance of the object from the camera and the angle of view of the camera. This inaccuracy in measurement is decreased when subsequent views are added.

Calculation Of Projection Through Workspace It has been stated earlier that a volumetric representation of an object can be obtained by projecting several two dimensional views of the object through the workspace and finding their intersection. These projections are calculated using the following procedure.

The position of the camera relative to some fixed reference point must be calculated. Position refers to both camera location (x, y, z coordinates) and orientation (angles it makes with the planes of the coordinate system). In this case under consideration two angles are sufficient to describe the orientation of the camera. They are the angle the camera makes with the side elevation (that is the rotation) and the angle it makes with the plan (that is the dip).

The final part of the process is to calculate the image projection through the workspace as follows. Assume a ray of light is passed parallel to the side and top planes and at an angle of ninety degrees to the front plane. This beam, of length Z_1 units, passes through the camera coordinates X, Y and Z.

Now if the beam of light is moved through an angle 'a' in the horizontal plane and then through an angle 'b' in the vertical plane, a point relative to the camera centre which the beam passes through can be calculated (see figure 5). If only part of the beam is considered (measurement Z_1), and the magnitude of the section under consideration is varied then a series of points which the beam passes through can be calculated. If the above process is repeated for each pixel in the image then the series of lines obtained by joining the calculated points forms the projection through the workspace.

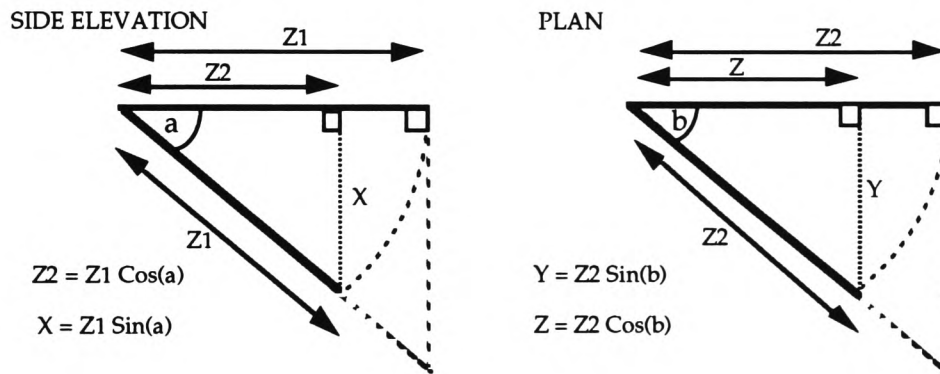


Figure 5 - Position through workspace given by X, Y and Z.

MODELLING SYSTEM DEVELOPED

Once a suitable data structure had been developed and implemented the next stage was to incorporate it into the modelling system. The modelling system developed, by Ware et al (3), allows the following functions to be carried out:-

- (1) Update the model
- (2) Display views of the workspace model
- (3) Modify Workspace Characteristics
- (4) Modify Sensor Characteristics

CONFIGURATION OF COMPUTER NETWORK The model of the prototype workspace was held on eight transputers, each storing the model for one eighth of the workspace. A CCD camera, connected to a transputer based frame grabber and image processing board, was used to obtain multiple views of the workspace. The transputer configuration for the prototype system is illustrated in figure 6.

To facilitate communication between the transputers in the network each transputer has, in addition to the main algorithm being executed, a set of 'communication

procedures'. These communication procedures allow the processor to receive and send data in the required manner. Data to be sent from one processor to another is packaged into a one-dimensional array. The first element in the array indicates the destination address, the second the source address and the third the quantity of data being sent. The sending processor uses the first two pieces of information to determine the best route to the required destination.

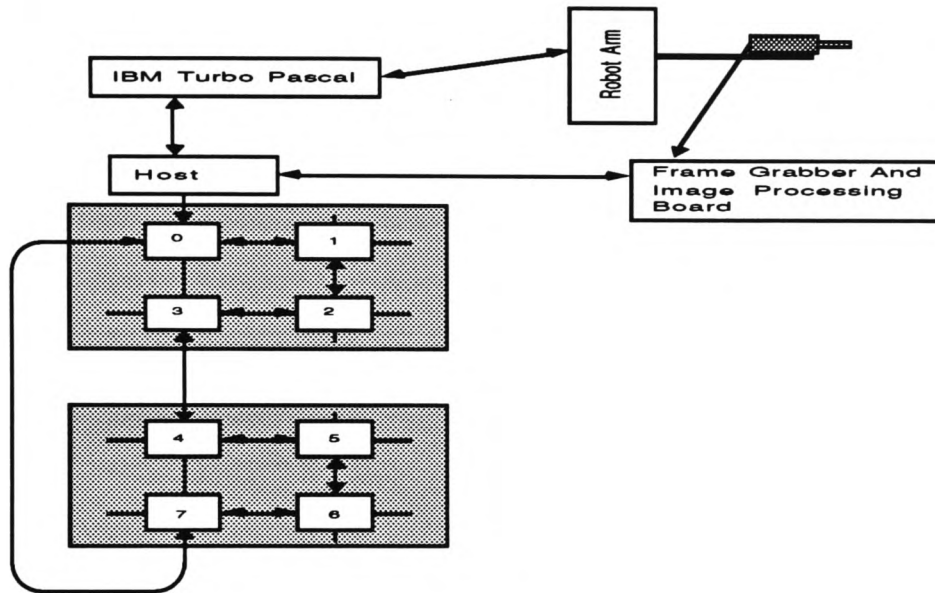


Figure 6 - Transputer Layout For Prototype System. Copies of the algorithm to maintain the workspace model are executed concurrently on processor 0-7.

SYSTEM TESTING

After the system outlined had been implemented, the next stage was to test its ability to locate objects within a prototype workspace. To enable testing, the algorithms running on the network were extracted. Extraction puts all the code necessary for the loading, communication and execution of algorithms into a file that is 'bootable' by a program running on host PC.

A Turbo Pascal program was written to interface the robot arm with the transputer network (the Pascal program running on the host PC). The Pascal program was responsible for driving the robot arm so that the camera repeatedly scanned the workspace until an object was detected. Once an object was detected the scanning was stopped and the orientation of the camera adjusted so that the object became located in the centre of the camera's field of view.

The 2-D image was then processed in order to remove noise and its projection through the workspace calculated. The information associated with projection was then packaged before being sent to the transputers that held the associated parts of the model.

The camera was then raised to a position vertically above the object's location. The robot moved the camera so that it scanned through an arc between the horizontal and vertical plane until the object was relocated. As with the first view the camera was adjusted automatically so that the object became located in the centre of the camera's field of view. This 2-D image was then processed as before.

The workspace model produced could now be used to determine the intersection of the two projections. This intersection enables the three, two-dimensional views of the object within the workspace to be determined. The centroid of each view was then calculated and used to give the location of the object.

CONCLUSIONS AND FUTURE WORK

The implementation of the partial run-length encoding data structure on an array of transputers proved, on the whole, to be an effective means of representing the workspace. It was found however, that confusion can arise in locating objects in certain circumstances. As an example consider the plan view of a workspace containing three objects.

If three views are taken then it is possible to end up with three intersecting views, figure 7(a). Using the premise that 'the more views intersect in a given area the greater the likelihood that an object exists in that area', a wrong conclusion would be drawn.

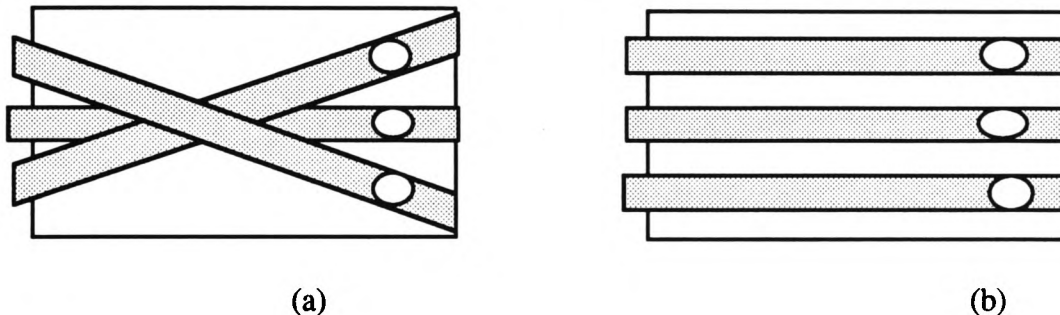


Figure 7 - (a) Using the intersection of views to determine the position of objects can lead to incorrect conclusions being made.
 (b) Using the intersection of views to determine the position of objects can also produce very good results.

This anomaly in the location procedure is something that requires further work. A possible solution might be the introduction of a rule based system for choosing the position from which views should be taken.

Introduction Of A Rule Based System Obviously in the example quoted, figure 7(a) is the worse possible case and it is equally likely that three views would have resulted in a more correct conclusion being deduced (for example figure 8(b) - the best possible case).

The problem illustrated by figure 8(a) is likely to be reduced (although it could conceivably get worse) if more views were taken from different positions. This solution however, gives rise to another problem that is as damaging to the viability of the system as the one it solves.

As the number of views increases so the time to obtain those views and add their projection to the data structure also increases. The number of view that can be held within the data structure is also restricted by the limited memory on the individual processors. The solution to these problems can be stated simply enough "do not waste time obtaining views that are not required, but obtain those views that will be of greatest benefit when analysing the workspace".

If a human were asked to analyse the workspace then, depending on the workspace layout, they would know intuitively which view to take. The problem that needs to be solved therefore becomes "how can human intuition be imputed to the computer?". One solution might be to introduce a rule based system for determining which view should be obtained next. An example rule might be:-

(rule) If two views intersect then a third view should be obtained that intersects the first two at their intersection.

In practice if two views were obtained of the workspace as in figure 8(a) which intersected then a third view that met the criteria of the above rule would make clear that there was no object at the intersection of the two views. However, if the same rule was applied to the workspace shown in figure 8(b) then it can be seen that the error in conclusion is compounded. It is clear that the choice of rules is an important factor that might prove valuable research material.

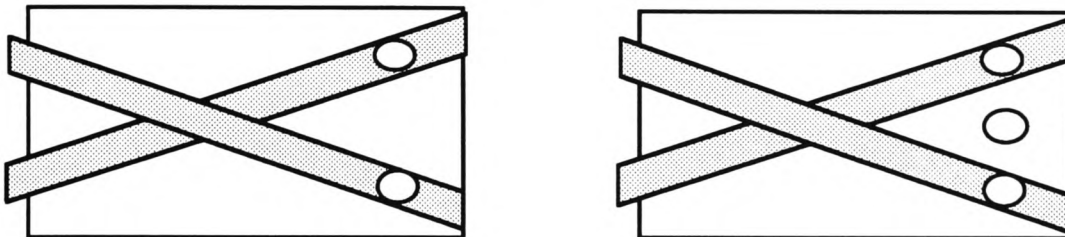


Figure 8 - (a) Using a third view at the intersection of the first two views will produce a more accurate model of the workspace.
 (b) Using a third view at the intersection of the first two views will produce a less accurate model of the workspace.

REFERENCES

- (1) Ware, J.A., Roberts, G., Davies, R.A., Williams, J.H., "The development and implementation of a modelling system for a robot's workspace", VISION '90, Society of Manufacturing Engineers, Michigan, USE, November 1990.
- (2) Martin, W.N., and Aggarwal, J.K., "Volumetric descriptions of objects from multiple views", IEEE transactions on pattern recognition and machine intelligence, VOL PAMI-5 No 2, March 1983, pp 150-157.
- (3) Ware, J.A., Roberts, G., Davies, R.A., Miles, R., Williams, J.H., "A modular sensing system for robotic control", The second international conference on applications of transputers, Southampton University, July 1990, pp 78-85.
- (4) Chien, C.H., and Aggarwal, J.K., "Model Construction and Shape recognition from occluding Contours", IEEE Transactions on pattern analysis and machine intelligence, Vol 11, No. 4, April 1989, pp 372-389.

A5.4 Paper 4

"A user interface for robotic sensing systems: A multi-faceted approach", Colloquium on HCI: Issues For The Factory, The IEE, Savoy Place, London, 21'st February 1991.

A User Interface For Robotic Sensing Systems: A Multi-Layered Approach

J.A.Ware¹, R.A.Davies², G.Roberts¹.

Abstract

This paper outlines the user interfaces to a modularised sensing system designed to locate and recognise objects within a robot's workspace. The interface has been divided into three layers each designed for different levels of user operation. At the lowest level the user is concerned with the hardware and software employed to extract information from the workspace. The middle level deals with the way in which the extracted information is presented to the higher level users. At the highest level the interface is responsible for allowing the user to deal with the extracted information in the required way. A description of the three layers is given together with the philosophy behind their design and implementation.

[1] **Introduction** Robots are generally seen as anthropomorphic, combining human qualities of intelligence, mobility and manipulative ability. This perception of a robot is one that has been propagated by the writers and directors of fictional films. In reality, most robots currently employed in the manufacturing industry, fall far short of being the intelligent robot that resembles a human. An intelligent robot uses sensory perception to detect changes in the work environment or work condition and, by its own decision making faculty, proceeds with its operations accordingly.

One of the main problems facing the would be user of intelligent robots is the difficulty of interfacing the robot with modern sensor technology. For the greater part, the degree of integration between the software and hardware employed in sensing systems means that the addition of extra sensors requires, at the very least, substantial changes to both hardware and software.

Research at the Polytechnic Of Wales [1,2 and 3] has led to the development, in prototype form, of a system that contains the facility to add extra or to change existing sensors without requiring major changes to hardware and software. This is achieved by designing the robotic system as a set of independent modules. In particular the hardware, software and the user-interface have been modularised and each can be modified independently of the other two.

This modularity means that the sensors that provide the information about the work cell are independent of the algorithms that make use of the information. That is, the sensors have no knowledge as to when or how the information they provide is to be used. Similarly, the algorithms that make use of the data have no knowledge as to the provider of the data.

-
1. Dept Of Mathematics and Computing, The Polytechnic Of Wales, Pontypridd, Mid Glamorgan, CF37 1DL,
 2. Dept Of Computer Studies , The Polytechnic Of Wales, Pontypridd, Mid Glamorgan, CF37 1DL.

[2] Overview Of System Implemented As a prerequisite to object identification their location within the robot's workspace must be determined. In the system to be described, the location of objects is achieved by building a model of the workspace. The model is produced using information provided by sensors located within the workspace. This model is then used to determine the location of objects within the workspace [4].

Once objects have been located the identification process can begin. Objects can be described in terms of their features and thus to facilitate object recognition a set of describing features is produced for all objects within the object set. The features used in the prototype are:-

- (1) Ratio of the length of two lines which intersect at right angles at the centre of gravity of object; one line being the axis of least moment of inertia of the object.
- (2) Ratio of perimeter length squared to object area.
- (3) Euler Number
- (4) Ratio of convex hull perimeter length squared to convex hull ratio.

[3] Hardware And Software Modules The majority of algorithms designed to interpret and analyse the information provided by the sensors, although well tried and tested, are rather slow in execution. In order for these algorithms to be executed in real time they have been modified to facilitate implementation using parallel processing techniques. Parallel processing involves either splitting the process to be performed into several subprocesses and performing these on different processors concurrently, or splitting the data that is to be processed between a number of processors and executing multiple copies of the process simultaneously.

It should be noted that some algorithms are inherently sequential and even for those that are not it is often a non-trivial task to adapt sequential algorithms for efficient execution on parallel machines. Baker and Harp [5] found when moving from sequential to parallel processing that *often a completely new approach is needed, and a simple transformation of the old code proves impossible. Conversely there are occasions when inherent parallelism in the problem can give excellent results with only minor changes to the sequential algorithm.*

The processor chosen for implementation of the sensing system was the Transputer. Transputers can be linked together to form a network and the workload of the application distributed across it. Not only does such a system offer a great deal of computing power, but its performance can be extended as required, by adding more Transputers to the network. Algorithms that are to be executed on the network may be implemented in a number of high level computer languages. However, if maximum efficiency is to be achieved they should be written in OCCAM, a language designed specifically for programming the transputer.

While OCCAM has much the same constructs as the majority of high level languages, one novel feature is the built in requirement for fixed indentation (considered by many less experienced programmers to be unnecessary). The feature is useful in the development of RUM code, that is code which is Readable, Usable and Maintainable.

In general transputers are well suited to the task at hand. There are however, a number of drawbacks in employing them. Firstly, connecting transputers to form an array of communicating processors is not a trivial task. Secondly, as OCCAM is a highly specialised language, only those that program the transputer regularly will be able to implement the algorithms quickly and efficiently. These difficulties in using the transputer can be reduced if a distinction is made between the people that set up the system and those that use it on a day-to-day basis. During the development of the prototype system it was therefore decided that there was a requirement for a multi-layered user interface where people with differing tasks have different views of the system. A task-oriented user interface was therefore constructed.

[4] Task-Oriented User Interface The most frequent user of the system will be the operator responsible for teaching it to recognise new objects and controlling the location and recognition of objects within the robot's workspace. These functions requires no changes to the way information about the objects is obtained from the sensors or presented to the user. At a different level and on a less frequent basis it might become necessary to modify the way in which the information extracted from the sensors is presented to and subsequently used by the operator. It is feasible, for example, that a change be required in the way in which results are presented.

In addition to the two levels of operation already stated there will also be times when the characteristics of the sensing system will require changing. This will occur when sensors are either removed from or added to the system. All three tasks will normally be undertaken at different time intervals and by people with different skills and knowledge about the system.

It is helpful to list the most common functions of the system along with the frequency at which the task is carried out. The main tasks can be grouped as follows (table 1):-




<p>Add objects to a set. Remove objects from a set. Load a set into the system from disc. Save a new or amended set onto disc. Identify object. Clear set. Update the model. Display views of the workspace model. Modify workspace characteristics.</p>		<p>A selection of these functions will be required whenever the system is used.</p>
<p>Change the way in which information is used or presented.</p>		<p>On an irregular basis, whenever the presentation of the information on the use to which it is put is changed.</p>
<p>Add or Remove sensors. Modify sensor characteristics.</p>		<p>When the system is set up or the configuration of the sensors is altered.</p>

Table 1 - Shows the main tasks and the frequency with which they are carried out.

From table 1 it can be seen that the tasks fall naturally into three layers:-

(i) **Layer 1** The lowest level is where sensors are either added to or removed from the system. If an extra sensor is to be connected then an extra transputer has to be added to the host PC. Conversely, if a sensor is to be removed then this requires the removal of a transputer board from the host PC. Whenever a board is added or removed then the transputers in the network will require rewiring and the OCCAM source code recompiled. This is a relatively trivial task for someone with technical expertise.

Each sensor employed within the system is connected to a transputer that preprocesses any data received before passing it on to the system's interface. The transputers used to interface with the sensors are programmed using OCCAM via the Transputer Development System (TDS). The TDS need only be used when a sensor is added or removed from the system as all other changes can be effected outside the TDS at a higher level.

The TDS consists of a plug-in transputer board and development software which runs on the transputer board. This combination provides a complete, self-contained development environment in which programs can be developed, compiled and executed. Programs can also be developed and compiled on the TDS to execute on a network of transputers, the code being loaded on to the network from the TDS. In this case the combination of transputer board and PC is referred to as the 'host computer', and the transputer network is known as the 'target system'. Finally, as is probably more realistic for most applications, programs can be developed for 'stand-alone' execution on transputers independently of the TDS.

The principal interface to the system is the editor. As soon as the system starts up the user is placed in an editing environment, and all program editing, compilation and execution can be carried out within that environment, by the use of a set of function keys. Instead of having a special command language to the operating system to manage the filling system, file operations occur automatically as a result of certain editor operations. There is also a set of 'utility' function keys which may be assigned to different functions during a session.

The editor interface is based on a concept called 'folding'. The folding operations allow the text currently being entered to be given a hierarchical structure ('fold structure') which reflects the structure of the program under development.

Just as a sheet of paper may be folded so that portions of the sheet are hidden from view, the folding editor provides the ability to hide blocks of lines in a document. A fold contains a block of lines which may be displayed in two ways: open, in which case the lines of the fold are displayed between two marker lines (called creases), or closed, in which case the lines are replaced by a single marker line called a fold line.

To create a fold the user inserts creases around the text to be folded; the fold is closed automatically when the second crease is made. Any text may be placed on the fold line to indicate what the fold contains; this text is called the 'fold header'. A fold may be removed, so that its contents are once again placed in sequence with the surrounding lines. Folds may contain text

lines and also fold lines; therefore folds can be nested. Folds can be nested to a maximum depth of 50. A folding editor allows the code to be viewed at different levels giving a hierarchical view of the code.

Once a suite of programs have been written using the TDS editor they can be extracted. Extraction puts all the code necessary for the loading, communication and execution of processes into a file that is 'bootable' by a program executing on the host PC [6].

The TDS enables the most common programming requirements (compiling, linking, editing and debugging) to be performed by the use of function keys. These function keys provide an effective and efficient user interface to OCCAM and the network for the regular TDS user. Unfortunately for those less frequent users there are a lot of key combinations to remember. A help screen is provided to overcome this problem but it is rather vague and difficult to understand. The main drawback with using the TDS is that it relies on the user 'remembering' rather than 'recognising' what is to be done.

(ii) **Layer 2** The middle level is where changes can be made in the way information provided by the sensors is used. An example of the type of changes that can be made at this level consider the following scenario; "A system has been set up to locate and recognise objects from a given object set. If an object is located, but is found not to be part of the set, then the object is removed from the workspace and ignored. Management have however decided that it would be advantageous to keep track of the frequency with which these unidentified objects have to be removed." The changes to facilitate this amendment would be made at this middle level.

The design philosophy underpinning this middle layer was that it should be understandable to someone with no knowledge as to how the sensor technology had been implemented. A Turbo Pascal program was therefore written to interface the transputer network with the outside world. This enabled the user to interface with the network via a more widely used and understood language.

(iii) **Layer 3** The highest layer provides the interface for the operator, who needs no programming or technical expertise. The operator is responsible for teaching the system to recognise new object sets and for controlling the system throughout the working day.

The interface consists of a series of hierarchical and interconnected menus which allow the user to perform the required operation, or display the required information, or if appropriate list further options. The advantage of using menus is that they act as an aid to memory. It is a well established fact that *"menus incorporate the desirable, cognitive feature of recognition, rather than recall, of the item or command needed. The user need only select an option to set the correct programs in motion. This eliminates the need to memorise cryptic, complex system commands to accomplish a set of actions. _ _ _ Menus enable the user to gain immediate use and productivity on a system. However, they must be responsive, address the needs of the user, and provide quick results."* [7]

The menus in the interface were designed to:-

- 1) minimise the number of times a user has to interface with the system,
- 2) minimise the learning requirements of the user,
- 3) minimise the use of user manuals,
- 4) guide users through the required task,
- 5) ensure that required operations are carried out in the correct order,
- 6) maximise clarity and control.

Menus have been arranged into hierarchical groups which the designer has thought to be sensible, comprehensible and convenient. The wording of the menu options is such that the user should have a clear understanding of what will happen when a given path is taken. However, should a user require the order and wording to be changed then these can easily be implemented via the Turbo Pascal editor.

The number of options in each menu varies between four and eight. This is consistent with the findings of Miller [8]. Miller's work examined the way in which users responded to a variety of different hierarchical menu systems. From his experiments he concluded that the best performance was obtained with four to eight options per menu and the lowest error rate with eight selections per menu.

[5] Conclusions And Future Work

The splitting of the user-interface into layers (a diagrammatical representation of this is shown in figure 1) has resulted in a system that has several advantages over conventional systems:-

- (i) The underlying technology is revealed to the user on a 'need to know basis'. This leads to a steeper learning curve for the majority of users. Table 2 acts as a summary of the main tasks in each of the three layers and shows the personnel who perform them.
- (ii) The highest layers can be modified quickly and easily to suit individual requirements.
- (iii) Layers can be modified and updated independently of the rest of the system.

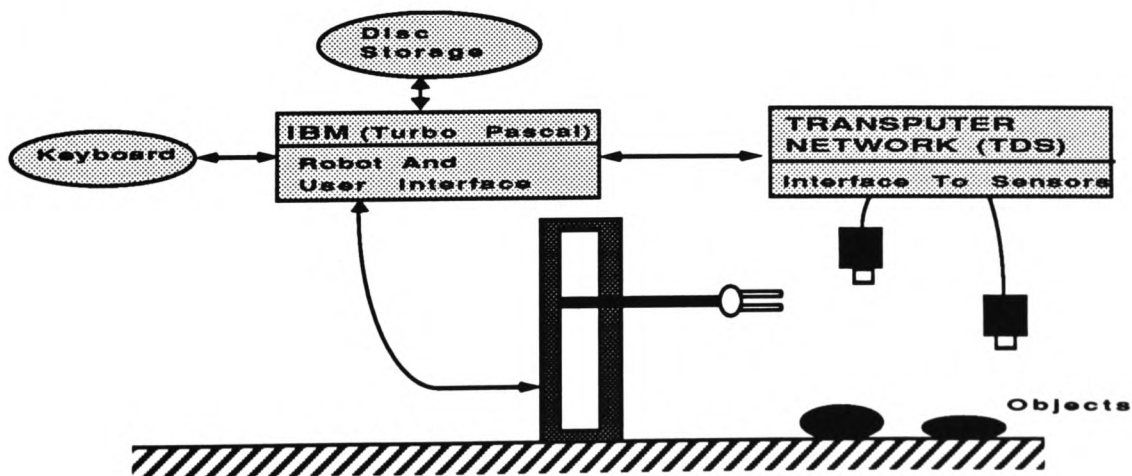


Figure 1 - A diagrammatical overview of the system.

<p>Add objects to a set. Remove objects from a set. Load a set into the system from disc. Save a new or amended set onto disc. Identify object. Clear set. Update the model. Display views of the workspace model. Modify workspace characteristics.</p>		<p>An operator, who needs no programming or technical expertise.</p>
<p>Change the way in which information is used or presented.</p>		<p>Pascal programmer.</p>
<p>Add or Remove sensors. Modify sensor characteristics.</p>		<p>OCCAM programmer with technical expertise.</p>

Table 2 - Shows the main tasks and the personnel who perform them.

It is envisaged that in the future additional layers will be added, in particular:-

- (i) The system described can be used to locate and recognise objects in a variety of robotic workspaces. These workspaces may range from a flat workbench to a room containing permanent fittings. At present the dimensions of the workspace are entered via the keyboard. The next stage in the evolution of the prototype is to enable the workspace model to be created interactively via a CAD system.
- (ii) A layer that will take the information provided by (i) and automatically calculate the optimum position for locating sensors.

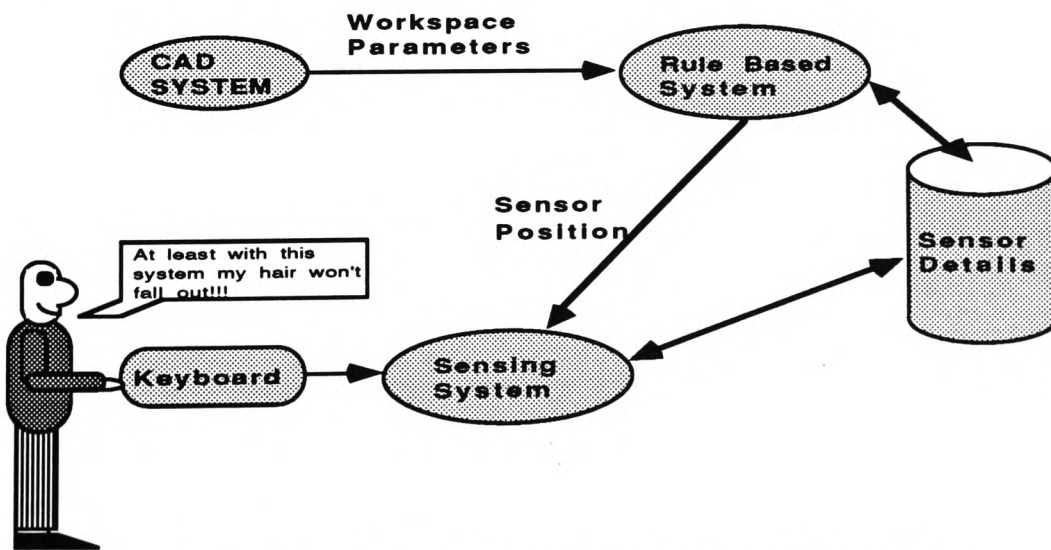


Figure 2 - Shows how additional layers can be added to the system.

[6] References

1. Ware J.A., Roberts G., Davies R.A., Miles R., Williams J.H., "A modular sensing system for robotic control", The second international conference on applications of transputers, Southampton University, July 1990.
2. Ware J.A., Roberts G., Davies R.A., Williams J.H., " Building and storing a model of a robot's workspace using a series of 2-D images", VISION '90, Society of Manufacturing Engineers, Michigan, USA, November 1990.
3. Ware J.A., Roberts G., Davies R.A., Williams J.H., "A multiple camera system to facilitate object manipulation within a robot's workspace", The second symposium on personal computers in industrial control, Warren Spring Laboratory, Stevenage, November 1990.
4. Ware J.A., Roberts G., Davies R.A., Williams J.H., "The location of components in an automatic manufacturing process", Colloquium on computer image processing and plant control , at The IEE, Savoy Place, London, 18'th May 1990.
5. Baker,S.A., and Harp,J.G., "Parallel processing on transputer arrays for the recognition of objects in infrared images", Colloquium on 'Transputers for image processing applications', IEE, Savoy Place, London, February 13'th 1989.
6. Inmos, 1987, "Transputer development system", Prentice Hall International (UK) LTD.
7. Hodgson G.M., Ruth S.R., "The use of menus in the design of on-line systems: a retrospective view", SIGCHI Bulletin, Volume 17, Number 1, July 1985.
8. Miller A.R., "MENU: An easy way to simplify CP/M", Interface Age, Nov 83, pp 138-141.

A5.5 Paper 5

Ware A., Kidner D. (1991) "Parallel implementation of the Delaunay triangulation within a transputer environment", The Second European Conference on Geographical Information Systems (EGIS '91), Brussels, pp. 1199 - 1208.

PARALLEL IMPLEMENTATION OF THE DELAUNAY TRIANGULATION WITHIN A TRANSPUTER ENVIRONMENT

Andrew Ware⁺ & David Kidner^{*}

⁺Department of Mathematics and Computing

^{*}Department of Computer Studies

The Polytechnic of Wales

Treforest, Mid Glamorgan, Wales, U.K. CF37 1DL

e-mail : JAWARE or DBKIDNER@UK.AC.POW.GENVAX

ABSTRACT

Digital terrain modelling by triangulated irregular networks (TINs) is adaptive to surface variability, as only critical features are incorporated within the data structure. One such method for constructing a TIN is the Delaunay triangulation, an efficient technique which produces unique solutions using any random arrangement of points. As TINs are computationally expensive to construct from large numbers of points, recent research has focused on the use of optimal parallel algorithms in their implementation. The unique characteristics of the Delaunay triangulation lend themselves to parallel implementation on a transputer network. A transputer is a microcomputer with its own local memory which when networked to other transputers enables algorithms to run concurrently on a number of processors. This paper outlines the application of an array of transputers using algorithms encoded in OCCAM to implement a parallel version of the Delaunay triangulation. Major considerations of hardware and software implementation include the relationship between number of transputers used and algorithm efficiency, and a comparison between parallel and sequential algorithms. Experimental results support the conclusion that use of parallel algorithms running in a transputer environment, greatly enhances the computational efficiency of triangulation.

1. INTRODUCTION

Digital terrain modelling addresses the problem of characterising the Earth's surface by either numerical or mathematical representations of a finite set of terrain measurements. The regular rectangular grid is the most commonly used DTM, due to its simplicity, implicit coordinates, application efficiency and the widespread availability of data in this format. However, their inability to adapt to terrain variability and the likelihood of data redundancy has led to the development of alternative methods. One such DTM is the triangulated irregular network or TIN (Peucker et al, 1978), which utilises 'surface-specific' points (eg. peaks, pits, passes, ridges and channels) to form a network of triangular facets.

Many different triangulation algorithms have been defined for the surface representation of an arbitrary data set, in an attempt to satisfy a number of criteria used to determine a 'good' triangulation (Gold, 1979). McCullagh (1987) states that a triangulation should have the properties of stability, equilaterality and non-intersection for some applications, such as contouring, where an arbitrary triangulation may not be acceptable. The Delaunay triangulation meets these requirements and has been extensively used as a basis for surface modelling (De Floriani, 1987).

As DTMs are more widely used within GIS, the need has arisen for national data bases of terrain at fine resolutions. As a result, DTM and GIS data structures and their associated algorithms must evolve to handle the vast amounts of data that will become available. However, for a large number of data points, a TIN can be highly inefficient in storage and for search and retrieval operations (De Floriani, 1987). Hence the advantages of the TIN are in danger of being compromised by extensive data volumes. Whilst the surface-specific points are adaptable to surface roughness with no 'data redundancy', the representation of the TIN topology may incur extensive storage overheads (Kidner & Jones, 1991).

The problem of database storage overheads for a TIN can be alleviated by storing only the vertices from which the triangulation topology can be regenerated at the application stage using an appropriate algorithm (Kidner & Jones, 1991). However, for this and the generation of other static TINs, the triangulation process can be computationally expensive. Hence, the application of parallel processing for triangulating a set of points has recently received more widespread attention (Merks, 1986; El Gindy, 1986, 1990). The Delaunay triangulation is particularly amenable to a parallel implementation, due to its uniqueness characteristic (ie. for any set of points, the triangulation is singular). This paper details a prototype parallel implementation of the Delaunay triangulation and addresses some of the hardware and software considerations.

2. THE DELAUNAY TRIANGULATION

The Delaunay triangulation of a set of points is considered the 'best', since the triangles formed are as equiangular as possible and the longest sides of the triangles are as short as possible, thus avoiding thin and elongated facets. The resulting triangulation is unique, such that irrespective of starting point, the network of triangles formed will always be the same. This triangulation is the dual of the Thiessen polygons (Voronoi diagram, Dirichlet tessellation or Wigner-Seitz cells), in which any location on the plane is assigned to the polygon containing the nearest data point (Gold, 1979). Three lemmas can be distinguished which globally and locally define a Delaunay triangulation (Lee & Schachter, 1980) :

Lemma 1 : For any triangulation of N nodes, B of which are on the boundary (convex hull), there are $2N-B-2$ triangles and a total of $3N-B-3$ edges (Euler's Theorem).

Lemma 2 : Two vertices form a Delaunay edge, if and only if there exists a circle passing through the vertices that does not contain any other vertex.

Lemma 3 : Three vertices form a Delaunay triangle if and only if its circumcircle does not contain any other point in its interior.

From Lemma 2 and 3, a simple algorithm can be defined for the construction of the Delaunay triangulation, in which the properties of Lemma 1 are implicitly incorporated. The algorithm used in this study is based upon that of McCullagh & Ross (1980). The search of the neighbours of a vertex (Thiessen neighbours) proceeds in a clockwise direction around that point. For any known neighbour, the next neighbour is located as the vertex for which a circumcircle passes through the three points, with no other point inside the circle (Lemma 3). This is accomplished with a search circle passing through the vertices of the known edge, checking for an inscribed point in a clockwise direction. If no points are found, the size of the circle is increased, whilst if more than one point is located, the Thiessen neighbour which has the largest angle subtended from the known edge is selected.

This algorithm is satisfactory for 'arbitrary' located data, but since more and more TIN data are derived from regular grid DTMs, the degree of arbitrariness may be insufficient for a consistent triangulation. The regular coordinates of the vertices may cause degeneracies to occur, such that four or more points may lie on a circumscribing circle, each subtending the same angle from the known edge. This is equivalent to four or more Thiessen polygons meeting at one point. An example of this occurs for the four vertices of a regular grid square. By the above definitions, both diagonals are valid edges, thus causing an overlapping or intersection of triangles. If the edges are calculated from every vertex, the algorithm will produce a degenerate triangulation. McCullagh & Ross (1980) suggest selecting the point closest to the known neighbour providing it is not also closest to the other vertex of the known edge. However, this is not sufficient for a degeneracy of the above type. Instead a local decision rule is required which takes into account the nature of the surrounding points, since an arbitrary choice of edge may cause large elevation errors. For example, the diagonals of the four points forming a grid square could represent either a ridge or a channel, but the true feature can only be determined from examining the local terrain.

For a parallel implementation, there will be several initial points (one per processor), from which the local triangulations are calculated. It is thus essential that the algorithm employed adheres to the criteria set out by McCullagh (1987), namely that of stability, equilateralness and non-intersection of triangles. This will require the triangulation to be unique for any starting point and is the most important implementation consideration. McCullagh & Ross (1980) have shown, for the Delaunay triangulation, that the resulting network of triangles is unique, irrespective of starting point.

3. PARALLEL PROCESSING IN A TRANSPUTER ENVIRONMENT

The procedure for Delaunay triangulation may be considered rather convoluted and thus slow in execution. This is primarily due to the search time required to locate the Delaunay edges or Thiessen neighbours of each point (see Section 4). In order for the triangulation to be executed in real time, a system has been developed to facilitate implementation using parallel processing techniques. Parallel processing involves either splitting the application or process to be performed into several sub-processes and performing these on different processors concurrently, or splitting the data that is to be processed between a number of processors and executing multiple copies of the process simultaneously. It should be noted that some algorithms are inherently sequential and, even for those that are not, it is often a non-trivial task to adapt sequential algorithms for efficient execution on parallel machines. Ware et al (1990) found converting from sequential to parallel processing may necessitate a completely new approach, since a simple transformation of the sequential code may prove impossible. Conversely there are occasions when inherent parallelism in the problem can give excellent results with only minor changes to the sequential algorithm.

The processor chosen for the parallel implementation of the Delaunay triangulation was the transputer¹. There are several chips in the transputer family, but the two most important ones are the T414 and T800, both of which rate 10 MIPS (million instructions per second) at 20 MHz. The T414 has 2K of on board RAM, whilst the more powerful T800 has 4K, as well as its own on-board floating point unit, which works in parallel with the main CPU and is capable of 1.5 Mflops (million floating point operations per second). In the prototype system described below, the host transputer is a T800, whilst the network transputers are T414s. Overall performance would be substantially increased if all the network transputers were substituted for T800s.

Transputers can be linked together to form a network of processors such that the workload of the application can be distributed across it. Not only does such a system offer a great deal of computing power, but its performance can be extended as required, by adding more transputers to the network. The network transputers are available as a fixed number per printed circuit board (PCB). For the prototype systems discussed in this paper, there are four transputers per PCB. Algorithms that are to be executed on the network may be implemented in a number of high level computer languages. However, if maximum efficiency is to be achieved they should be written in OCCAM, a programming language designed specifically for the transputer.

The initial prototype system was implemented on a network of four transputers arranged to form a bidirectional loop (Figure 1). The network is connected to a host transputer which is responsible for reading the data from disk before packaging it to be sent to the network. The whole system is housed in an IBM PC and has access to the host computer disk and input/output system.

To enable the OCCAM-encoded processes to run concurrently on several processors, a means of sending data from processor to processor is required. To keep communication overhead to a minimum it is important that this facility sends data via the best possible route around the network and that the number of times a processor needs to communicate with other processors is optimised. It is also important to ensure that processor power is used as effectively and efficiently as possible. To help meet these criteria the following rules were developed :

- 1) The processors should be kept 'busy' as soon as possible and for as long as possible.
- 2) Data transfer between processors should be via the best possible route around the network. This is not necessarily the shortest as 'traffic jams' might develop along certain paths.
- 3) Data transfer should be kept at an optimum level to achieve maximum efficiency, such that when transfer is required it should be given priority over other tasks (this enables the receiving transputers to make use of the transferred information as soon as possible).

¹ The word transputer is a composite of transistor and computer. The name reflects its design, in that it consists of a processor, memory and communications facilities built onto a single chip.

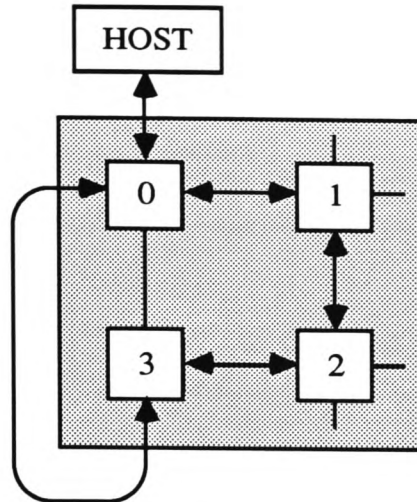


Figure 1 - Prototype System of Four Transputers.
 (N.B. The flexible link between transputers 0 and 3 allows future expansion of the network.)

The optimum level of information transfer depends on two factors. First, it may be quicker to duplicate processes on more than one processor, than to send information from processor to processor. Second, if one processor is generating information to be shared by other processors in the network, then the information can be passed at one of three stages. These stages being :

- (i) after all the information has been generated,
- (ii) after a given amount of information has been generated,
- (iii) as and when the information is generated.

Whilst this option depends on the system being implemented, the correct choice of approach for information passing is crucial to the efficient execution of parallel processes. When making this decision, the system designer should bear in mind rule (1). Timings have shown, that in the prototype system described, option (ii) proved to be the best choice.

To facilitate communication between the transputers in the network, each transputer has, in addition to the main algorithm being executed, a set of 'communication procedures'. These procedures allow the processor to receive and send data in the required manner. Data to be sent from one processor to another are packaged into a one-dimensional array. The first element indicates the destination address, the second indicates the source address, whilst the third represents the quantity of data being sent. The sending processor uses the first two pieces of information to determine the best route to the required destination. This communication packet is depicted below in Figure 2.

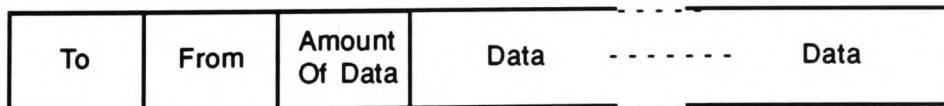


Figure 2 - Format For Passing Data Around the Network.

Each transputer in the network therefore has the following processes executing concurrently :

- (a) A 'get info' process to receive data from both the clockwise and anticlockwise transputers.
- (b) A multiplexer (Multi-Clock) to send data to the next clockwise transputer. This data may have been processed by the sending transputer, or the sending transputer may be acting as a link in the chain. (The multiplexer is required to collect data from the 'get info' and 'process request' routines and pass it on via a single connection).
- (c) A multiplexer (Multi-AClock) to send data to the next anticlockwise transputer. Again this data may have been processed by the sending transputer, or the sending transputer may be acting as a link in the chain.

In addition, the first transputer in the network has the facility to communicate with the host transputer, via the Multi-Host multiplexer. Figure 3 shows the processes that run concurrently on the first transputer, while the other transputers in the network have all but the Multi-Host process running concurrently on them.

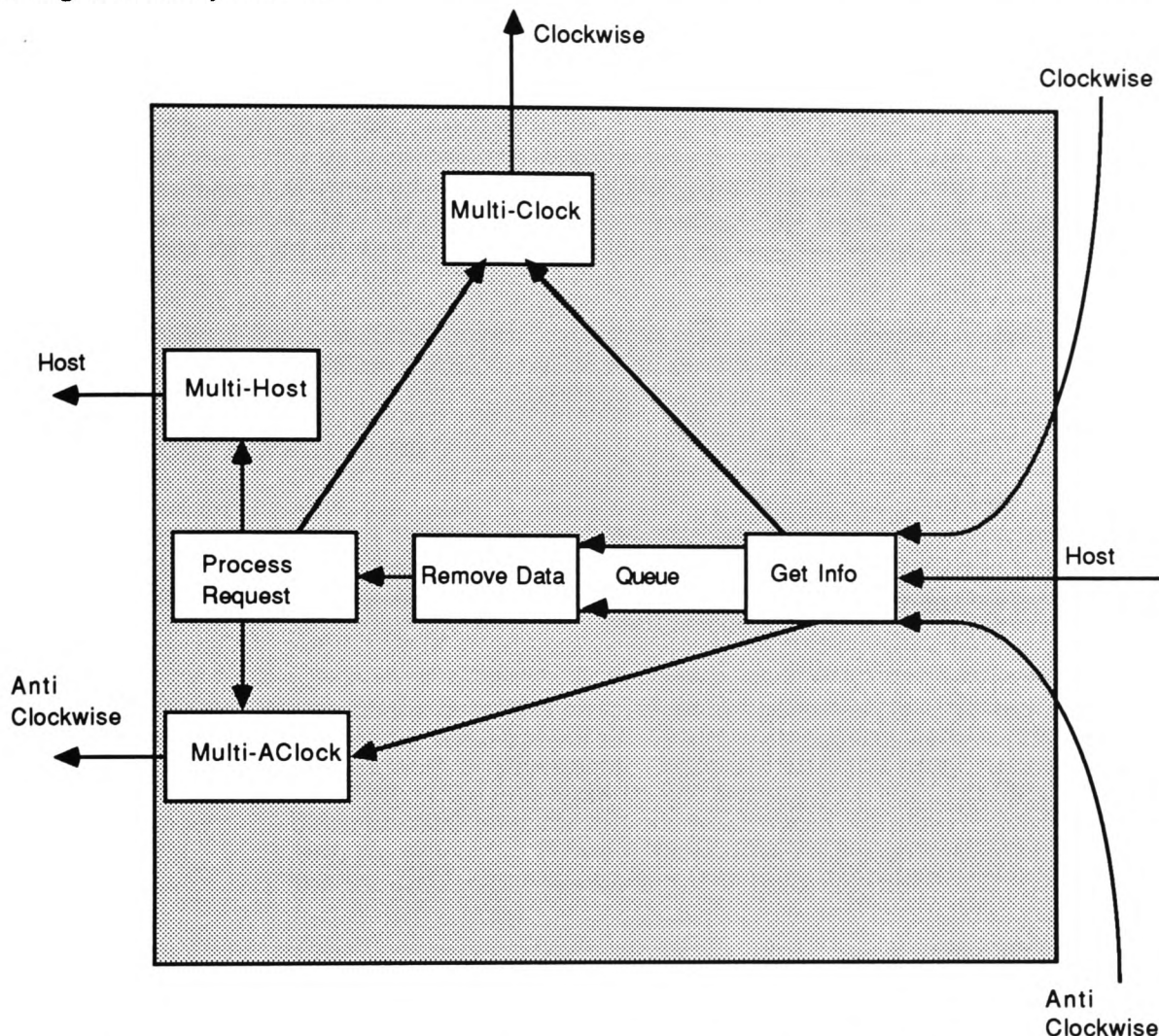


Figure 3 - Diagrammatical View Of The Processes Running Concurrently On The First Processor.

4. IMPLEMENTATION DETAILS

The initial process of most triangulation algorithms is to calculate the convex hull of the data. For a parallel implementation, this must be accomplished for the complete data set, rather than the convex hull of each processor subset. In this prototype model, the vertices of the triangulation are derived from a regular grid DTM. As such, the boundary of the complete data region forms a rectangle, so the convex hull does not pose a problem. In addition, the boundary points are implicitly identified from their x or y coordinates. For a model such as this, the boundary points are not treated any different from interior points. At the triangulation stage, however, the boundary points enable the search for Thiessen neighbours to stop at the boundary with the 'outside world'.

The spatial search for data points is organised in a grid overlay or 'box-sort' auxiliary data structure (McCullagh & Ross, 1980; Kidner & Jones, 1991). This enables an efficient search for Thiessen neighbours to be limited to the most likely candidates. A grid overlay is used to spatially address all the points within each grid cell, provided that the data have been sorted by x and y coordinates. However, if the vertices are derived from a regular grid, the data can be written to a file in this order, without sorting. A further consideration in this parallel implementation is the relationship between the grid structure and the data subset for each processor. The main advantage of the grid is its simple and very efficient, direct access to points within a grid cell. Therefore, the partitioning of

the vertices between processors should correspond with the nature of the spatial addressing of points. Since the data are organised within a grid, it follows that the assignment of processors should correspond to a number of grid cells, organised either in long, rectangular strips or in square blocks of cells, such as a quadtree.

For the prototype implementation, the rectangular strips were chosen, due to their greater flexibility. Strips of grid cells, corresponding to a row or column of the grid overlay provides a simple allocation of the data to processors, whilst maintaining an efficient spatial search for triangulation. The vertices in each processor strip can be accessed sequentially for the calculation of Thiessen neighbours, without searching at each stage for the next point within the subset. For this implementation, blocks of strips are assigned to each processor, since every processor is likely to access a number of strips. This also allows distinct data sets to be assigned to each processor, such that the complete data set need not be copied to each transputer. This would have been necessary with narrow strips, since a processor would not know in advance what strip it would be assigned and neighbouring strips would also have to be copied, since edges will connect points in neighbouring strips. The number of strips assigned to each processor is calculated such that the number of points is distributed as equally as possible, within the confines of the overlaying grid.

5. RESULTS AND CONCLUSIONS

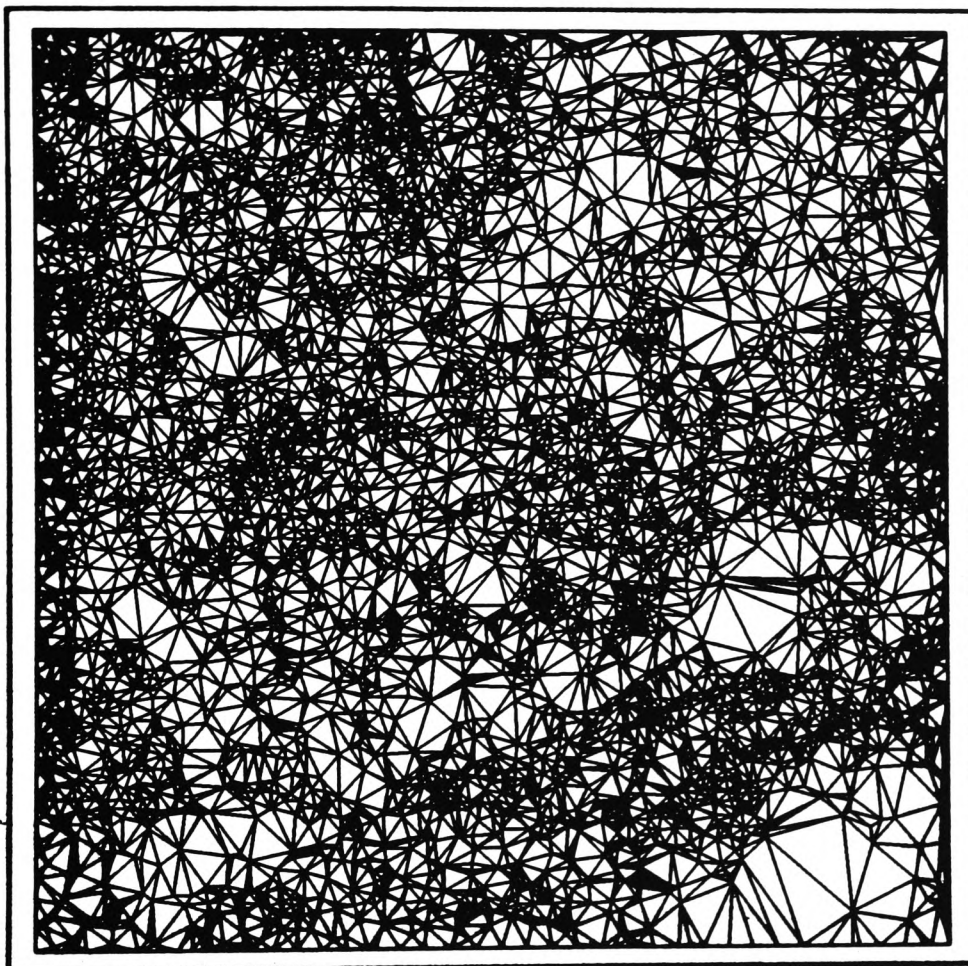
The test data set for the prototype model represents a 20 x 20 kilometre region of the South Wales valleys (National Grid Reference ST08/09/18/19). The vertices were initially derived by a dynamic triangulation, such that the TIN was constrained to a maximum error tolerance of 15 metres. The network consists of 8278 vertices (309 of which are on the boundary), 16,245 triangles and 24,522 edges (Figure 4).

For the prototype implementation of a network of four transputers, initial testing has shown that a significant reduction (approaching a quarter of the time for one processor) in processor time can be achieved when triangulating for a given set of points. Thus for this implementation, time decreases in direct proportion to the number of processors. Figures 5 & 6 illustrate the stages of the triangulation at 1/12 th and 1/6 th of the time taken for one processor. In effect, the first (top) strip can be considered to be the equivalent stage for just one processor in the same time allocation. A comparison of the TINs with the original in Figure 4, shows that the triangulation at each stage is identical. This proves that the Delaunay triangulation is unique, irrespective of initial starting point.

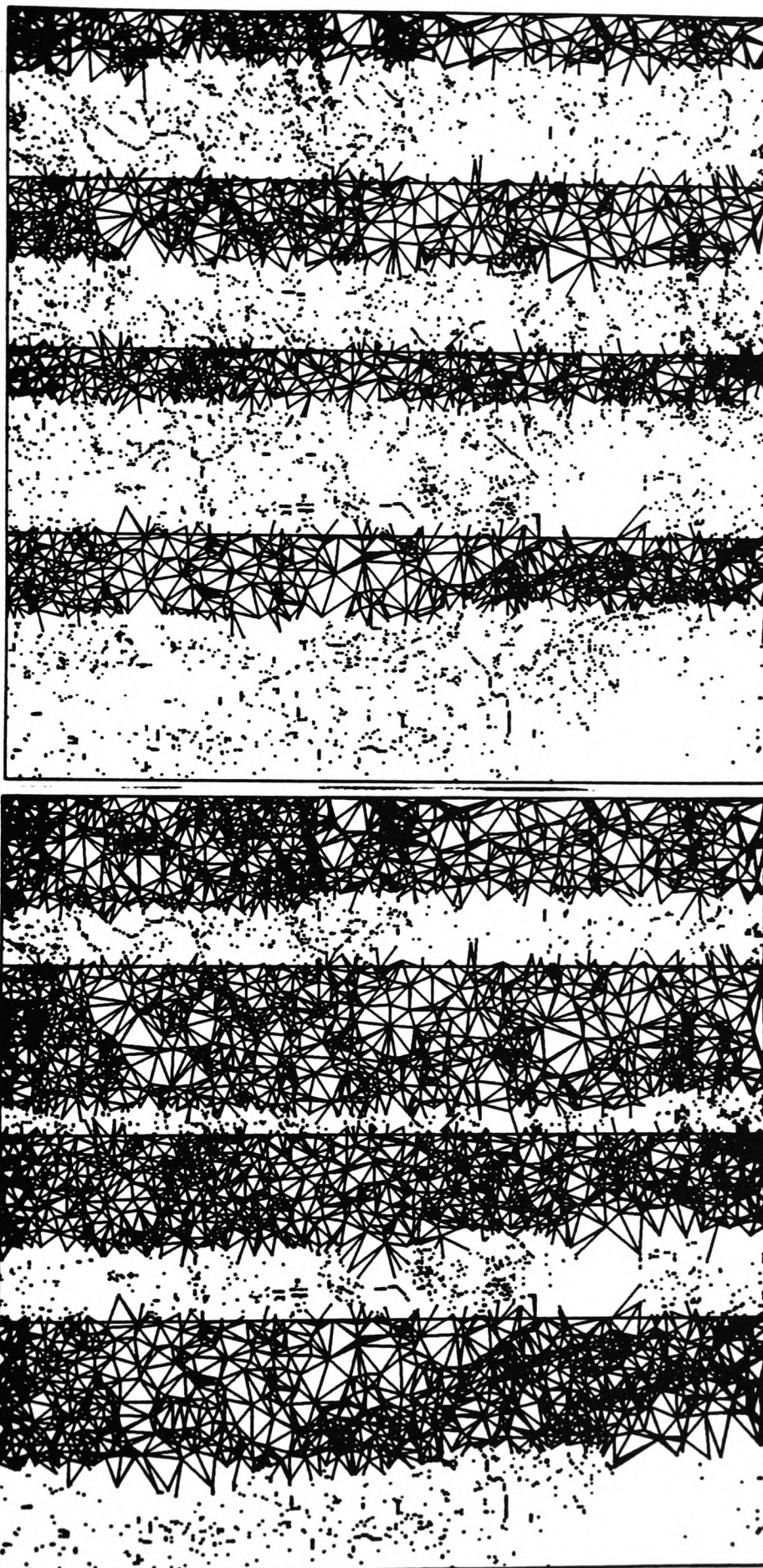
The bidirectional loop of four transputers can easily be extended to facilitate an even faster turn around rate (see work by Ware et al). The prototype implementation was therefore extended to a network of eight (2 PCBs of four transputers) and 16 (4 PCBs) transputers. The resulting TINs at 1/12 th and 1/24 th of the time taken for one processor are illustrated in Figures 7 & 8. The former of these figures is therefore directly comparable with Figure 5 after the same elapsed CPU time. It should be noted however that the law of diminishing returns will come into force as the size of the network is increased. This is due to the increased communication overheads incurred when passing information around the network. Another factor which will affect the performance of the system is the partitioning of the data. Despite strips of approximately equal numbers of vertices being assigned to each processor, the time taken for each processor to calculate the Thiessen neighbours will be different. This variation, whilst small as a proportion of total time, has a tendency to be more significant as the number of processors increases. Since the effective total time is equivalent to the time of the last processor to finish, any significant variance could cause a noticeable decrease in performance, or 'processor redundancy'.

Any variation in time for the processors is due to the calculation of each vertex's Thiessen neighbours and more specifically the search time required to find these edges. An equal number of points per processor however, will not necessarily produce an equal number of edges. This will be variable, depending upon the spatial distribution of the data. Thus search time is not directly related to the number of vertices, but rather the number of edges. In the examples of Figures 5-7, for four and eight processors, the variation in number of calculated Thiessen neighbours (or edges) is over 1000, for a very small variation in vertices. This can be seen in Figure 6, where the second processor appears closest to finishing. Hence, an ideal partitioning of the data should be related to the number of edges per processor. However, this cannot be determined before triangulation has been completed, so an alternative criterion for data partitioning is required. This could possibly be accomplished by examining the density of point sampling within each possible processor strip. For the triangulation

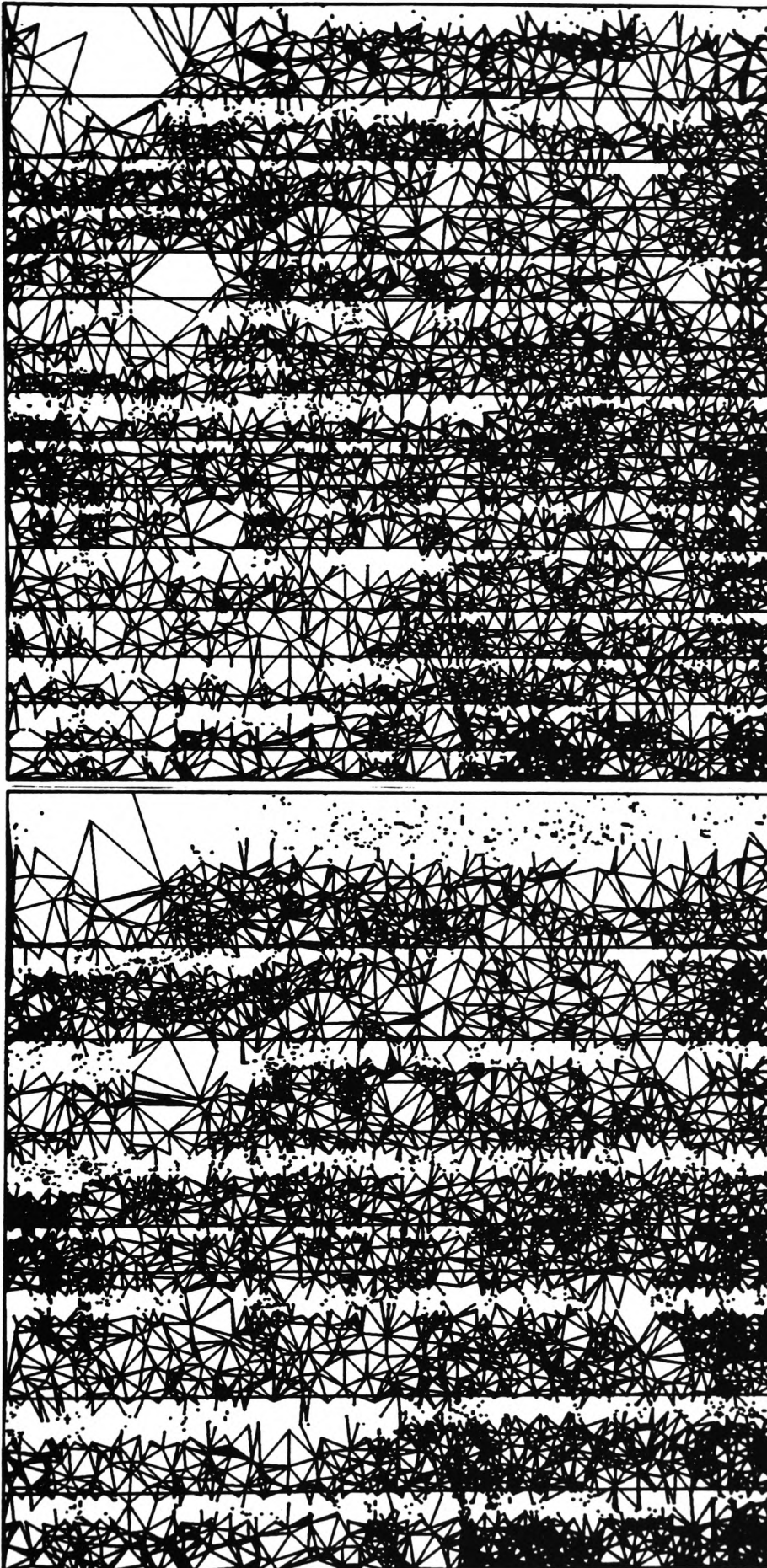
algorithm described in this paper, it is clear that the search time for each possible Thiessen neighbour is related to the number of points within each search circle. Hence, the searching algorithm should be more computationally efficient for sparser data.



*Figure 4 - Triangulated Irregular Network for Test Data Set ST08
(8278 Vertices, 309 Boundary Points, 16245 Triangles and 24522 Edges)*



Figures 5 and 6 - ST08 TIN Using Four Processors at 1/12 th and 1/6 th of CPU Time of One Processor



Figures 7 and 8 - ST08 TIN Using 8 and 16 Processors at 1/12 th and 1/24 th of CPU Time of One Processor

6. REFERENCES

- De Floriani, L. (1987) "Surface Representations Based on Triangular Grids", *The Visual Computer*, Vol.3, No.1, Feb., pp.27-50.
- El Gindy, H. (1986) "An Optimal Speed-Up Parallel Algorithm for Triangulating Simplicial Point Sets in Space", *Int. Journal of Parallel Programming*, Vol.15, No.5, Oct., pp.389-398.
- El Gindy, H. (1990) "Optimal Parallel Algorithms for Updating Planar Triangulations", *Proc. of the 4th Int. Symposium on Spatial Data Handling, Zurich*, Vol.1, July 23-27, pp.200-208.
- Gold, C.M. (1979) "Triangulation-Based Terrain Modelling - Where Are We Now?", *Proc. of Auto Carto IV*, Vol.2, Nov.4-8, pp.104-111.
- Heller, M. (1990) "Triangulation Algorithms for Adaptive Terrain Modeling", *Proc. of the 4th Int. Symposium on Spatial Data Handling, Zurich*, Vol.1, July 23-27, pp.163-174.
- Kidner, D.B., Jones, C.B. (1991) "Implicit Triangulations For Large Terrain Databases", *Proc. of the Second European Conference on Geographical Information Systems*.
- Lee, D.T., Schachter, B.J. (1980) "Two Algorithms For Constructing a Delaunay Triangulation", *Int. Journal of Computer and Information Sciences*, Vol.9, No.3, pp.219-242.
- McCullagh, M.J., Ross, C.G. (1980) "Delaunay Triangulation of a Random Data Set for Isarithmic Mapping", *The Cartographic Journal*, Vol.17, No.2, Dec., pp.93-99.
- McCullagh, M.J. (1987) "Digital Terrain Modelling and Visualisation", *Proc. of a Short Course in 'Terrain Modelling in Surveying and Civil Engineering'*, Univ. of Surrey, Apr.7-9, and Univ. of Glasgow, Sept.1-3, 27 pages.
- Merks, E. (1986) "An Optimal Parallel Algorithm For Triangulating A Set of Points in the Plane", *Int. Journal of Parallel Programming*, Vol.15, No.5, Oct., pp.399-411.
- Peucker, T.K., Fowler, R.J., Little, J.J., Mark, D.M. (1978) "The Triangulated Irregular Network", *Proc. of the Digital Terrain Models (DTM) Symposium*, May 9-11, pp.516-540.
- Ware J.A., Roberts G., Davies R.A., Miles R., Williams J.H. (1990) "A Modular Sensing System For Robotic Control", *The Second Int. Conference On Applications Of Transputers*, Southampton University, July, pp 78-85.

ACKNOWLEDGEMENTS

This work has been partially funded by the Royal Signals and Radar Establishment (R.S.R.E), Ministry of Defence, Procurement Executive, Malvern, Worcestershire, 56England.