Design of a programmable array manipulator

Philip Ciufo
University of Wollongong, ciufo@uow.edu.au

UNIVERSITY OF WOLLONGONG
COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author.

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Recommended Citation


Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
DESIGN OF A PROGRAMMABLE ARRAY MANIPULATOR

A thesis submitted in fulfilment of the requirements for the award of the degree

Master of Engineering (Honours)

from

UNIVERSITY OF WOLLONGONG

by

P. P. Ciufo

B.E. (Hons), University of Wollongong, 1991

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

1993
Acknowledgements

Part of this work was funded from the Federal Government's Dept of Industry, Trade and Commerce under the Generic Technology component of the Industrial Research and Development Act, 1986.

I would also like to acknowledge the work of other researchers associated with this project. They are Mr N. Laszlo, Dr. P. M. Carter, Professor C. D. Cook, Dr. F. Naghdy, Dr P. Wong and S. Kii. Without their contributions, this project would not have been successful. The assistance from the Workshop Staff of the Department of Electrical and Computer Engineering at the University of Wollongong has been invaluable.

I also thank my wife, Marilyn, and my two daughters, Emma and Sophie, for the patience and support that they have given me throughout the course of this research.

Philip Paul Ciufo,
April, 1993
Design of a Programmable Array Manipulator

by

P. P. Ciufo

Submitted to the Department of Electrical and Computer Engineering in March 1993, in fulfilment of the requirements for the degree of Master of Engineering (Honours)

ABSTRACT

The Programmable Array Manipulator, or PAM, is a new technology designed to perform simultaneous manipulations on multiple objects. In its most primitive form, PAM is a matrix of actuators arranged in such a way that it may transport an object resting upon it. This transportation may be either a translation or a rotation. Such a device is currently under development at the University of Wollongong.

The purpose of this thesis is to describe the methodology and implementation of the design factors associated with the prototype development of a PAM. Since PAM is a new device many issues need to be resolved. These issues include the control system design, the actuator hardware, the hardware to control the actuators and the sensor system required for object position and orientation feedback.

The solutions to these issues form the basis of this thesis. The detailed design and the associated experimental verification is presented in full.
Contents

Declaration ii
Acknowledgments iii
Abstract iv
List of Figures viii
List of Tables x

1 Introduction 1
1.1 Programmable Array Manipulator Project 1
1.2 Array Manipulator 4
1.3 Array Manipulator Controller Design Parameters 5
   1.3.1 Modularity of PAM 5
   1.3.2 Design Criteria 6
1.4 Vision System Design Parameters 9
1.5 Summary of Thesis Aims 11
1.6 Scope of the Thesis 12
1.7 Summary 13

2 Background 15
2.1 Introduction 15
2.2 General Manipulation Systems 16
2.3 Ciliary Motion System 17
2.4 Array Manipulator Control 18
2.5 PAM Control Strategy 21
   2.5.1 Just In Time (JIT) Systems 22
2.6 Summary 24

3 Programmable Array Manipulator Hardware Design 26
3.1 Introduction 26
3.2 Single Element PAM Actuator 27
   3.2.1 Original PAM Actuator Concept 27
   3.2.2 Single Element Actuator Model 30
   3.2.3 Single Element Actuator Model Simulation 35
   3.2.4 Single Element Actuator Simulation Results 37
3.3 Multi-Element PAM Actuators 38
3.4 PAM Mechanical Modularity 42
3.5 PAM Control System Modularity 43
3.6 PAM Actuator Interface Hardware 46
   3.6.1 PAM Actuator Interface, Mark 1, Overview 46
   3.6.2 XILINX Hardware Overview 47
   3.6.3 PWM Solenoid Excitation Concept 48
   3.6.4 PWM Bit Pattern Calculation 49
   3.6.5 PWM Phase Control Philosophy 51
   3.6.6 PWM Reference Point Synchronisation 53
   3.6.7 Phase and Solenoid Selection Logic 55
   3.6.8 Alternative Actuator Interface Scheme 57
3.7 Summary 58
## List of Figures

1-1 Array Manipulator Application Example .......................... 2  
1-2 Fast Arraying of Articles for Palletising ......................... 3  
1-3 Sorting and Inspection of Meat Patties ............................. 3  
1-4 Sorting Application Sample Arrangements ......................... 6  
1-5 Example of Translational Direction Vector Allocation ............. 10  
1-6 Thesis Presentation Layout ........................................... 14  

2-1 Ciliary Motion .......................................................... 17  
2-2 Sample Palletising Task ................................................. 19  
2-3 JIT Node Relationship .................................................. 23  
2-4 Thesis Presentation Layout - Progress ............................... 25  

3-1 Single Element Prototype Side View .................................. 28  
3-2 Single Element Prototype Top View .................................. 29  
3-3 Forces Acting on a Component on a Track .......................... 30  
3-4 Single Element PAM Simulation Pseudo Code ....................... 36  
3-5 Simulated and Measured Vertical Amplitude Vs Velocity .......... 37  
3-6 Simulated and Measured Vertical Amplitude Vs Angle of Translation 38  
3-7 Vibratory PAM Mk1 Actuator Arrangement .......................... 39  
3-8 Vibratory PAM Mk2 Actuator Arrangement .......................... 40  
3-9 Vibratory PAM Ideal Solenoid Displacement ....................... 41  
3-10 Section View of 16 Actuator Module ............................... 42  
3-11 Actuation Module Plan View ......................................... 43  
3-12 Control Module Arrangement ........................................ 45  
3-13 PWM Solenoid Excitation ............................................ 49  
3-14 PWM Bit Pattern Generation Algorithm ........................... 51  
3-15 PWM Phase Control Philosophy .................................... 52  
3-16 PWM Synchronisation Logic ....................................... 53  
3-17 Synchronous Communication Decoding ............................. 56  
3-18 Thesis Presentation Layout - Progress ............................ 59  

4-1 Centralised Control Architecture ................................... 63  
4-2 "Five by Five" Manipulation Module Arrangement .................. 64  
4-3 Alternative Manipulation Module Arrangement .................... 64  
4-4 Control Layer Structure - Element Control Layer ................ 67  
4-5 Object Rotation Techniques ........................................... 69  
4-6 Control Layer Structure - Object Control Layer .................. 73  
4-7 Control Layer Interaction - Middle Three Layers ................ 74  
4-8 Control Layer Structure - Vision System Layer ................... 78  
4-9 Control Layer Structure - Command Layer ........................ 80  
4-10 Control Layer Structure - Communication Control Layer ........ 82  
4-11 Thesis Presentation Layout - Progress ............................ 83  

5-1 Control Layer Structure - Command Layer ........................ 84  
5-2 Simulation Scenario .................................................... 91  
5-3 Cost Function Illustration .......................................... 93  
5-4 Command Layer Simulation Result ................................. 96
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-5</td>
<td>First Alternative Simulation Scenario</td>
<td>104</td>
</tr>
<tr>
<td>5-6</td>
<td>Second Alternative Simulation Scenario</td>
<td>105</td>
</tr>
<tr>
<td>5-7</td>
<td>Thesis Presentation Layout - Progress</td>
<td>108</td>
</tr>
<tr>
<td>6-1</td>
<td>TTGF Block Diagram</td>
<td>114</td>
</tr>
<tr>
<td>6-2</td>
<td>Colour Look Up Table (CLUT) Function</td>
<td>115</td>
</tr>
<tr>
<td>6-3</td>
<td>Image Processing Task Block Diagram</td>
<td>117</td>
</tr>
<tr>
<td>6-4</td>
<td>Image Capture Pseudo Code</td>
<td>118</td>
</tr>
<tr>
<td>6-5</td>
<td>VRAM Physical Arrangement</td>
<td>118</td>
</tr>
<tr>
<td>6-6</td>
<td>Object Windowing</td>
<td>120</td>
</tr>
<tr>
<td>6-7</td>
<td>Boundary Tracing Pseudo Code</td>
<td>122</td>
</tr>
<tr>
<td>6-8</td>
<td>Boundary Tracing Process</td>
<td>123</td>
</tr>
<tr>
<td>6-9</td>
<td>Image Data Arrangement</td>
<td>125</td>
</tr>
<tr>
<td>6-10</td>
<td>Associate Pixel Address Calculation Pseudo Code</td>
<td>126</td>
</tr>
<tr>
<td>6-11</td>
<td>Associate Address Allocation</td>
<td>126</td>
</tr>
<tr>
<td>6-12</td>
<td>Orientation Calculation</td>
<td>129</td>
</tr>
<tr>
<td>6-13</td>
<td>Thesis Presentation Layout - Progress</td>
<td>132</td>
</tr>
<tr>
<td>7-1</td>
<td>36 Element PAM General Arrangement</td>
<td>135</td>
</tr>
<tr>
<td>7-2</td>
<td>Direction of Translation Versus Phase</td>
<td>140</td>
</tr>
<tr>
<td>7-3</td>
<td>Rotation Strategies</td>
<td>142</td>
</tr>
<tr>
<td>7-4</td>
<td>Object Position Test Layout</td>
<td>143</td>
</tr>
<tr>
<td>7-5</td>
<td>Orientation Test Scenario</td>
<td>144</td>
</tr>
<tr>
<td>7-6</td>
<td>Thesis Presentation Layout - Progress</td>
<td>147</td>
</tr>
<tr>
<td>C-1</td>
<td>Simulated Vertical Amplitude Vs Velocity</td>
<td>201</td>
</tr>
<tr>
<td>C-2</td>
<td>Simulated Vertical Amplitude Vs Angle of Translation</td>
<td>201</td>
</tr>
<tr>
<td>C-3</td>
<td>Simulated Vertical Amplitude Vs Velocity</td>
<td>202</td>
</tr>
<tr>
<td>C-4</td>
<td>Simulated Vertical Amplitude Vs Angle of Translation</td>
<td>202</td>
</tr>
<tr>
<td>C-5</td>
<td>Simulated Vertical Amplitude Vs Velocity</td>
<td>203</td>
</tr>
<tr>
<td>C-6</td>
<td>Simulated Vertical Amplitude Vs Angle of Translation</td>
<td>203</td>
</tr>
<tr>
<td>C-7</td>
<td>Simulated Vertical Amplitude Vs Velocity</td>
<td>204</td>
</tr>
<tr>
<td>C-8</td>
<td>Simulated Vertical Amplitude Vs Angle of Translation</td>
<td>204</td>
</tr>
<tr>
<td>C-9</td>
<td>Simulated Vertical Amplitude Vs Velocity</td>
<td>205</td>
</tr>
<tr>
<td>C-10</td>
<td>Simulated Vertical Amplitude Vs Angle of Translation</td>
<td>205</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1</td>
<td>Cost Functions</td>
<td>101</td>
</tr>
<tr>
<td>5-2</td>
<td>Simulation 1 Results - Multiple MM Failures</td>
<td>102</td>
</tr>
<tr>
<td>5-3</td>
<td>Simulation 2 Results - Multiple MM Failures</td>
<td>105</td>
</tr>
<tr>
<td>5-4</td>
<td>Simulation 3 Results - Multiple MM Failures</td>
<td>106</td>
</tr>
<tr>
<td>7-1</td>
<td>Natural Rotation Frequencies</td>
<td>141</td>
</tr>
<tr>
<td>7-2</td>
<td>Rotation Performances</td>
<td>142</td>
</tr>
<tr>
<td>7-3</td>
<td>Results of Static Position Test</td>
<td>144</td>
</tr>
<tr>
<td>7-4</td>
<td>Static Orientation Test Results</td>
<td>145</td>
</tr>
<tr>
<td>7-5</td>
<td>Dynamic Positioning Results</td>
<td>146</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Programmable Array Manipulator Project

The Programmable Array Manipulator research project was funded from the Federal Government's Department of Industry Trade and Commerce. Direct funding was made available through the Generic Technology component of the Industrial Research and Development Act, 1986.

The primary object of the project was to conduct research into the design and development of an intelligent, programmable, and multi-directional conveyor capable of transporting a number of objects toward individual destinations simultaneously as shown in figure 1-1.

The characteristics of the objects to be manipulated were not initially specified since there was no firm understanding of the PAM design criteria relating to object size. However, as commercial applications of PAM were studied in more detail objects with a footprint size of approximately 65 mm by 65 mm became the basis of the design and development of PAM.
The application areas initially proposed for PAM were as follows:

1. The intelligent decomposition of piles for singulation. The system would have computer vision as the main sensory device. An example of this application could be found in mail sorting.

2. Programmable arraying and packaging of produce or other inconsistent articles. For example, packing to produce a fixed weight or arraying of meat for retail presentation.

3. Intelligent inspection of produce and manufactured goods.

Such applications are illustrated in figures 1-2 and 1-3.
Chapter 1

Introduction

Fast Arraying of Articles for Cartoning or Palletising

Programmable Pattern Forming

Incoming Feed

Simultaneous Arraying

Transfer of each Formed Layer to Pallet

Incoming Meat Patties

Outgoing Stacks for Packaging

Figure 1-2 Fast Arraying of Articles for Palletising.

Visual Inspection

Reprocessing Bin

Conveyor System

Incoming Meat Patties

PAM

Outgoing Stacks for Packaging

Conveyor System

Rejects

Sorting

Stacking

Figure 1-3 Sorting and Inspection of Meat Patties
1.2 Array Manipulator

An array manipulator, in the most general sense, is a flat, table-like device with an array of actuators used to manipulate multiple objects. An actuator, in this context, is any mechanism capable of imparting a net force on an arbitrary object to produce a controlled and repeatable movement of the object. The nature of the manipulation by PAM is to move an object along a specified direction toward a target position. The motion of the object can be a translation or a rotation and it takes place in the plane of the actuators since the array manipulator does not perform three dimensional manoeuvres. This is the definitive feature of the array manipulator. Examples of objects that could be manipulated are meat cuts, or postal packages. On a smaller scale, wafers used for the manufacture of integrated circuits could be manipulated using a similar device [F5].

The array manipulator is also capable of performing simultaneous manipulations on a number of individual objects. This means that selected objects can be manipulated to individual destinations, or a common destination concurrently. This operation has a considerable speed advantage over conventional robotic manipulation.

In addition to the area of sorting and palletising, array manipulators have several advantages over special purpose machines. These advantages are due to the generic nature of the array manipulator. A special purpose machine is constructed with only a limited number of tasks in mind. Careful design may allow such a machine to cope with a variety of products but in general products would necessarily have similar dimensional properties. In contrast to this situation, an array manipulator can, in principle, handle objects with any dimensions greater than approximately six times the size of an individual actuator in the array.

The array manipulator also has disadvantages over conventional sorting devices. For example, new hardware and software feedback control schemes are necessary. The array
manipulator requires position and orientation feedback of the objects being handled. This feedback can be produced by a number of sensors. The sensor should be of the non-contact type to avoid producing any unwanted translations or rotations of the object on the array. The sensor system could be embedded in the array of actuators, or be part of the actuator mechanism. Alternatively, a camera based sensor system could be used, depending upon the size of the array.

1.3 Array Manipulator Controller Design Parameters

There are several issues that require careful investigation in order to find the most suitable form of control scheme for an array manipulator. An appropriate scheme for one application is not necessarily the best scheme for another. However, it is possible to generate a method that will work for the majority of applications. It may not be the most efficient, but it will work for all conditions.

1.3.1 Modularity of PAM

PAM is a modular device. It has a modular structure not only in the physical arrangement of the device, but also in the way that it should be controlled. For example, suppose a group of 1024 actuators, a vision system and control scheme is defined as a PAM module. In order to make a PAM for use in, say, sorting applications, an array of 4096 actuators may be required. In this instance, the PAM could be configured several ways, as demonstrated in figure 1-4.

The generic nature of the manipulator means that different applications demand different physical configurations. That is, a meat patty inspection application may require an array of 256 by 512 actuators and a sorting table may require a 512 by 512 actuator arrangement. Macroscopically, each device is really performing the same task. That task is to transport articles from one place to another, subject to some criteria. Thus the question arises
whether the control scheme for one application need necessarily be different from the other.

![Diagram of Sorting Application Sample Arrangements]

Figure 1-4 Sorting Application Sample Arrangements

The necessity of different physical configurations leads to the conclusion that an array manipulator should be a reconfigurable and modular system. Each different arrangement can be realised by combining standard modules to suit the environment it is targeted at. This characteristic is considered as a design criteria for the Programmable Array Manipulator. The PAM is modular in nature, and hence the control system devised for it should reflect and enhance this property and not detract from it.

1.3.2 Design Criteria

Initial investigations have shown that the following issues in the control strategy should be addressed.

*Control of the "in-feed" system.*

Given that the array manipulator is part of a continuous process, there will be another process prior to the array manipulator that will be feeding material onto the array.
Alternatively, upstream processes may provide a continuous feed of objects onto the array without the array having any control over this process. Clearly, the arrangement that permits the array to have an input into the feed mechanisms is superior.

**Control of the "out-feed" mechanism**

Once again, the arrangement that gives the array manipulator an input into the control of the outgoing process is desirable. Without such an ability, the control scheme could potentially become confused if items are being removed from the array without permission. This could be compensated for if the control scheme incorporated a vision component.

**Trajectory Planning**

The tasks of trajectory planning and control for an orderly or otherwise traversal of objects from their source to destination is an important aspect of the controller. If the scheme had to individually calculate and control the trajectory of every object presented to it, the computational complexity may be potentially cumbersome. There are mechanisms available to minimise such complexity. Nevertheless, some form of trajectory control needs to be implemented.

**Tracking and stock control**

The control scheme would need to know the location of an object "x" at any instant except for applications where all the objects being manipulated are identical. The inclusion of a tracking and stock control task enhances the generic nature of an array manipulator.

**Determining the Object Position and Orientation**

The object position and orientation are derived from the raw sensor data. This could be an input to the control scheme rather than a direct inclusion. A separate vision system continuously providing this input would be more desirable.
**Control of the Actuating Mechanisms**

The actuating mechanisms used to transport the objects around the grid have to be controlled. The lower level task of driving the actuators to achieve the desired directional response would be performed elsewhere. Nevertheless, the control system needs to be able to issue commands of the kind "move object x in direction y using actuators 1, 7, 8, and 11".

**Coordination of the task requirements**

This requires taking the necessary tasks and the sensory input information and translating it into physical assignments.

**Communication with any surrounding processes.**

If the array manipulator is part of a continuous process, then it may need to communicate status type information to the outside world for higher level job scheduling and so on.

The above is not an exhaustive list, but it does provide an indication of the demand on the control system for an array manipulator. When it is considered that an array manipulator can consist of several thousand actuators it can be seen that the control scheme to coordinate and control each of these actuators will require a high degree of complexity.

Ideally, the control scheme designed for PAM would be independent of the physical configuration. It would also be desirable for the proposed control scheme to be easily adapted to changes in the physical arrangement. In considering the previous variable arrangements, (figure 1-4), the unique properties of PAM would be enhanced considerably if the software and hardware for one arrangement required little or no modification for another arrangement.
It is possible to break the control system into two separate problems; one is how to control the actuators to obtain the desired translations and rotations, and the other is to develop a strategy for moving an object from position A to position B. This thesis deals with both of these issues.

1.4 Vision System Design Parameters

The array manipulator structure requires some form of feedback that can determine the position and orientation of an item being transported on the planar surface. Such feedback could come from several potential sources, including tactile sensing and vision based sensing. The incorporation of tactile sensing in the actuating mechanisms could interfere with the transportation process of the object. Additionally, the task of accommodating such a sensor and the associated wiring in PAM would be a difficult task. There would however be little scope for dealing with the problem of occluded objects.

The vision based sensing system has many advantages over the tactile device. The system would potentially be able to cope with:

1. orientation calculations,

2. inspection, and

3. tracking of items,

in a more precise and easily implemented manner.

As with most initial concepts, the specifications of the vision system as the primary sensing device of an array manipulator was not well defined. Such requirements are closely associated with the operating conditions of the array manipulator including the approximate positioning accuracy and possible object transportation velocity.
In order to realise the design parameters of the vision system, a set of performance criteria based on the results of the study of a trial PAM were established. They included:

1. maximum rotation speed 25 rpm
2. maximum translation velocity 100 mm/s

Beyond these restrictions, another performance criteria is set as a function of the style of actuator to be used in the array manipulator and possibly the hardware to be used to control the actuator. This other restriction defines the number of directions in which the object can be translated. Figure 1-5 shows how a particular object may be constrained to being driven in any one of eight directions. This will be explained in more detail in chapter 3. Nevertheless, such a restriction provides a lower limit for the position and orientation calculation from the vision system.

![Available Directions](image)

Figure 1-5 Example of Translational Direction Vector Allocation

If a vision system is devised that can provide a high degree of position and orientation measurement accuracy, then this accuracy is wasted since it is possible to move an object only in the available direction (figure 1-5).

A suitable performance requirement of the vision system could be as follows:
1. Position measurement tolerance +/- 5 mm

2. Orientation measurement tolerance +/- 5 degrees

These values are chosen to ensure reasonable performance of the array manipulator.

There is a high expectation for the image processing task with regards to execution time. The vision component of the control scheme plays an important role since it is providing the position feedback. This information needs to be supplied in near real-time in order to control the object to within the tolerance specified. Obviously, if such a processing speed cannot be met, then either the positional accuracy criterion must be relaxed, or the speed of the traversal be reduced. Alternatively, two or more positional accuracy criteria could be defined; one whilst traversing the array and a different one whilst performing the final positioning at a reduced speed.

1.5 Summary of Thesis Aims

The aims of the project are as follows:

1. To design and implement a control scheme for a programmable array manipulator. Any scheme should enhance the modularity of PAM and lend itself to reconfigurability. The scheme will be implemented using transputer based equipment and its support languages.

2. To design and implement a vision system for measuring the position and orientation of an object being manipulated by a PAM. The object will be of known geometry and physical properties. This information will be made available to the main control system for use in closed loop position control. The ability to provide this information for more than one object is also desirable. Occluded objects will not be considered in this design.
3. Design and implement the necessary hardware to allow the previous two aims to be implemented on a prototype PAM. The hardware designed should be able to allow controlled motion of an object on PAM. Such motion needs to be verified. The design must be able to deal with a potentially large number of actuators.

1.6 Scope of the Thesis

The research material and background work will be covered in chapter 2 of the thesis. A number of papers relevant to the project will be discussed.

For the project to meet its objectives, an actuator appropriate for PAM operation had to be designed. This means that the technical criteria needs to be met whilst still maintaining an acceptable cost structure. The results of the research into available actuator technologies is documented in [F5]. The search concluded with two actuator candidates, both using conventional solenoids as the active element. As a result of these investigations, several prototype PAMs were constructed.

One unit consisted of a single element (actuator) device. This was constructed in order to prove the ability of the actuator to move an object. Upon completion of this prototype, it was evident that it would be difficult to further predict the behaviour of objects on PAM with such an actuator. Consequently, a model was devised and a simulation performed in order to determine several design parameters for the next prototype.

The next device built consisted of 36 actuators, all smaller versions of the original concept. This device enabled verification of the control of objects on PAM and the determination of further design criteria for a PAM.
The third device is still being developed and consists of a 512 actuator PAM. The actuators used in this device are mechanically different from the previous two. However, the size of the actuator is much smaller.

The details of the original, single element design are presented in chapter 3. Included here is the derived model and subsequent simulation. The finer details of the actuator and hardware design for the 36 element device are discussed in chapter 3.

In the control of an array manipulator, many tasks require simultaneous execution. Chapters 4 and 5 will address these tasks in more detail and propose a multi-layered distributed control scheme. It will be shown that parallel processing capability and modularity are two features that the control system should have. In particular, chapter 5 will deal with a computer simulation of a PAM and its control scheme.

The dependence on the vision system to provide position and orientation data of the items being manipulated by the array device is a possible limitation. Since image processing is such a computationally intensive task, due to the high volume of data, it could be difficult to construct a scheme capable of satisfying the requirements of an array manipulator. Chapter 6 will present the vision system produced for the trial PAM.

The results of the experimental work will be presented in chapter 7. The vision system and control system performance are quantified and presented. These experimental results are based on the use of a trial 36 actuator PAM. The conclusions and a description of the possible future areas of research will be provided in chapter 8.

1.7 Summary

During the course of this chapter, the reader has been introduced to the concept of an array manipulator. The Programmable Array Manipulator (PAM), the device around which this thesis is based has been introduced. It was shown that PAM was a special case of an array
Manipulator capable of performing translations and rotations of objects located on its surface. The use of an array of actuators has many applications in manufacturing and material handling areas. Micro-arrays are undergoing research at the moment and as the thesis evolves, it will be seen that the control schemes for the micro-devices are not different from those which have been developed for PAM.

Since PAM is a new form of transportation device, many design parameters need to be determined. The actuator design and the interface between the actuators and the control system are two primary parameters. This thesis will take the reader through the design process involved with the complete development of a prototype PAM. Figure 1-6 describes the layout of the work to be presented.

![Figure 1-6 Thesis Presentation Layout](image)

This figure will be used throughout the course of this thesis so that the reader will understand the next step in the design process and how all the items together.
CHAPTER 2

BACKGROUND

2.1 Introduction

The main objective of this chapter is to study briefly the previous work related to PAM, particularly its control system. Since the array manipulator has not been actively researched previously, explicit references to array manipulators do not exist. However, work in some complementary areas provides some useful insights.

The basic function of PAM is to move objects in a plane to satisfy a performance criterion or a target. At a basic level, this is very similar to that of a part feeder where the final orientation of the parts is pre-defined. The main difference is that the performance expected from a PAM is much more complex and advanced than that of a conventional part feeder. In this chapter, the work conducted on a particular group of part feeders close in performance and architecture to PAM will be reviewed.

The performance criteria of the control system being developed needs to be established first. The implementation of the control system is related closely to the hardware of the PAM and so is effectively new material. The approach adopted is to present the reader with some basic concepts and ideas derived from the literature surveyed. In the course of the research performed for this project, another style of array manipulator was found. The device is undergoing research at Tokyo University's Institute of Industrial Science. The
device, referred to as a Ciliary Motion System falls very well into the category of an array manipulator.

2.2 General Manipulation Systems

The problem of parts feeding - orienting parts that are not positioned consistently - is a common problem in manufacturing and industrial automation. There exists a variety of machines in this area to perform the task of reorientation. However, in the majority of cases, the devices are designed to suit the geometry of a particular part. When the part geometry changes, the feeding or reorientation device must be redesigned. This process could take several months [G1].

The concept of a programmable parts feeder removes the need for mechanical redesign when the shape of the object that is being handled changes. For example, if an injection moulding process was part of a larger assembly line, then changing the design of the moulded piece need not necessarily bring about the redesign of the mechanical arrangement that feeds this part into the manufacturing line.

Peshkin et. al. [G2] and Singer et. al [G3] suggest that vibratory techniques, sliding and interaction with fixed fences could resolve these issues. However, both such devices still rely heavily upon knowing the geometry of the workpiece to be manipulated and designing a mechanical arrangement to suit. The work reported in [G1], however, presents a device that can reorient parts under software control. The authors use a "frictionless" parallel-jaw gripper that requires one controlled degree of freedom to orient the gripper and one uncontrolled degree of freedom to open and close the jaws. This design is considerably more versatile than the other systems. However it can only perform reorientation functions on a single object with a limited size. PAM does not suffer from such a constraint.
2.3 Ciliary Motion System

The Ciliary Motion System reported in [F5] comprises arrays of tiny hair-like strands, about half a millimetre long mounted on a silicon substrate. Each hair is made up of two different types of plastic, with a small wire placed between them. During the manufacturing process the high temperature curing causes the hairs to curl up. When current is passed through the wire sandwiched between the two plastics, the hair heats up and the differing rates of expansion cause the hair to uncurl until it is almost flat again. By repeating this action an object placed on top can be moved. This motion is illustrated in figure 2-1.

![Figure 2-1 Ciliary Motion [F5]](image)

This technique is applicable in the micro-machine domain because the thermal inertia of the cilia is sufficiently low to allow them to operate at a frequency of about 10 Hz. Hence, objects can be manipulated at a reasonable rate. However, when the physical scale is increased, problems will occur with cooling the cilia quickly to obtain reasonable conveying speeds. By interleaving rows of hairs at right angles to each other, it is possible
to control the exact position of an object anywhere on the surface of the array. Since the
device is small, it is suitable only for manipulating objects of similar dimensional
magnitudes. One potential application proposed is to use the ciliary motion system as a
transportation mechanism for silicon wafers. Such a system could position a wafer for
inspection or testing.

In large arrays with millions of cilia, trying to control each individual fibre from a central
control station would be a very complex logistical task. It would be difficult to complete
the hard wiring associated with each cilia and sensor. Therefore, one needs to incorporate
autonomy into the motion control system. The cilia are already fabricated onto a silicon
wafer, so it would be possible to incorporate computing power onto the same wafer.
Theoretically, if every hair could communicate with its neighbour, then the system could
coordinate and schedule its work in a cooperative manner. Such a concept is also
extendible for use with the Programmable Array Manipulator.

The Ciliary Motion System is seen as one of the first attempts at producing a generic array
manipulator. However, based on current technology, it can only be used on small objects,
both in physical dimension and mass. The Programmable Array Manipulator is capable of
transporting objects with a much larger dimensional footprint. PAM is seen as a first
attempt to create a generic array manipulator capable of dealing with a wide range of
objects.

2.4 Array Manipulator Control

In order to illustrate the problem, the following example will be considered. Suppose that a
PAM has a 5 by 5 module arrangement. Each PAM module has 2048 actuators and is
performing a simple palletising task whereby 5 coloured disks are being fed onto the PAM
from the left hand side, and a single module on the right hand side (say the bottom right
hand module) is to somehow obtain one of each disk and arrange them in a particular way.
The disks are then removed by some other mechanism. This scenario is depicted in figure 2-2.

From a control point of view, there needs to be some way of organising the orderly or otherwise traversal of the disks across the PAM. Additionally, each module could have its own vision system and actuator controller. Thus, the problem is how to organise all the modules to act cooperatively to achieve the aim of the process. Prior to embarking on a discussion about the control aspects of an array manipulator that has generic properties, one needs to come to terms with the requirements of such a control scheme. Chapter 1 introduced the basic jobs that a control scheme would be called on to do. For such a control scheme to be generic in nature, the tasks need to be broken into smaller, self contained jobs that can be made independent of the physical arrangement of the array. It is only this structured approach that can lead to a generic control scheme.
It is possible to break the control system into two separate problems: one is how to control the actuators to obtain the desired translations and rotations, and the other is determining the strategy to be used for getting an object from position A to position B. It is the latter problem that is extendible to a generic issue.

Once the control mechanisms have been established, the decision on how to distribute or otherwise arrange the control hardware and software must be made. Much work has been published that describes a variety of control structures in a manufacturing environment. These concepts are extendible to PAM, if one considers PAM to be a series of discrete events (perpetrated by each module).

The works presented in [E1], [E5], [E6], [E7] are particularly good discussions on various control architectures applicable to manufacturing systems. However, PAM has special needs, not entrenched necessarily by the requirements of manufacturing systems. [E1] in particular, provides a good discussion on control architectures that could be used by PAM. Such an architecture could be described as a system that fits in well with the modularity of PAM and enhances such properties.

A multi-level/multi-layer architecture is proposed by Jones and Saleh in [E1] for intelligent shop floor control. Each module within this architecture performs three functions: adaption, optimisation and regulation. In terms of the applicability of these concepts to PAM, the notion of multi-layered control is appealing. In a PAM there are many tasks that need to be performed. These tasks occur at different levels of the control hierarchy. For example, the algorithms for determining the particular movement requirements of an object on PAM are at a different level from the control of the actuators to implement this motion.

In hierarchical control structures, the following guidelines are used:

1. levels are introduced to reduce complexity, responsibility and authority,
2. each level has a distinct planning horizon, and

3. control resides at the lowest level.

In terms of multi-layer control, [E7] uses a three layer model for an intelligent control system. The three layers are execution, coordination and organisation. It will be seen that a combination of the hierarchical and multi-layer control architectures can be applied to PAM.

2.5 PAM Control Strategy

The type of control strategy required for PAM can also be found in systems where a number of devices, irrespective of their sizes, are required to operate on material which is flowing through them. The synchronisation of the operation and scheduling of them has similar characteristics to the control of a PAM.

One such example is a production line with its associated concept of "kanban". The concept of kanban relates, in the first instance, to a scheduling system in a manufacturing environment. This, at first sight may seem rather remote from the control scheme for an array manipulator. However, if one considers a modularised array manipulator as a series of material handling processes, the connection becomes more apparent. A simple but concise description of kanban is found in [E2].

In [E2], the author describes the use of kanban in scheduling and stock arrangement in a manufacturing environment by way of an example. As parts are made, they go into a readily identifiable container, such as a coloured box or tray. Each container has its own card stating the product, where it will go, how many parts are required to make up a complete container, and how many cards in a set, eg. a card could be number 1 of 5. When a work cell completes a container, it goes off, with its card, to a storage area, where its customer, or the next user, can select it. When a container load of parts is used up, the card
(or bin if it is a specialised part) is sent back to the cell responsible for the manufacture of that part. The cards are displayed on a board, for example, and when a preset trigger point is reached, the cell will commence manufacturing the part once again.

If each module of the PAM is modelled as a single process in an overall manufacturing strategy, it would be possible to apply the kanban idea. If the PAM function itself was the overall manufacturing process, then each module needs to contribute towards achieving the aim of this process. By using the kanban principle, one is able to organise the individual PAM modules in such a way that there is a high degree of cooperation between the modules such that the PAM operates in a cohesive manner. In this way, the need for a centralised controller is removed. This is a highly desirable quality for a generic PAM control scheme.

2.5.1 Just In Time (JIT) Systems

In JIT systems material flows only when it is pulled from a downstream process. That is, material flow is on an as-needed basis and not on an as-available basis. In classical JIT systems, kanbans are used to produce the required pull control mechanism. However, this may not necessarily be the case. Kanban is, in a sense, a physical realisation of control information requirement for demand-pull to be accomplished. In the case of a PAM control scheme, the concept of kanban can only exist in software. In this instance, the software is providing the pull control structure. In [E3], the authors describe a demand-pull mechanism control structure. In order to describe the structure, two hypothetical nodes are defined:

1. A puller is the node to which material is pulled.

2. A source node is the node from which the material is pulled.
For each puller node, a control signal is required to activate the pulling of material from its source node. Note that no comparable control structure exists for push systems, since material is always pushed as far as possible, as soon as possible. In order to take this concept further, additional explanation is required [E3].

Consider two JIT modules, A and B, operating in tandem. In [E3], the authors define a JIT module as a component of a system with a set of input and output buffers and some processing between these two buffers. At each buffer, a mechanism is established that enables it to pull material from other parts of the system (sources). The output from A serves as the input to B. A pull control signal is sent to module A’s output buffer from B’s input buffer. Since A has now supplied material to B, its output buffer develops the need for some additional material. Thus, A’s output buffer sends a signal to A’s input buffer to authorise the pulling of material to itself. This concept can be extended to an arbitrarily complex network of JIT modules. The authors claim that two conditions must be met:

1. Inter-module material flow occurs between the output buffer at the source and the input buffer at the puller. The flow is controlled by the puller modules input buffer.
2. Intra-module material flow, which occurs between a module's input and output buffer is controlled by the module's output buffer.

These control mechanisms are, of course, purely software strategies. Module B's unsatisfied input buffer demands must go to module A's output to seek satisfaction. In providing this service to module B, A's output buffer becomes unsatisfied. A's output buffer will now signal the input buffer for additional material. A's input buffer then proceeds to pull material through the manufacturing process to satisfy the input buffer's demand. This is the basic demand-pull mechanism. Such logic is applicable to that of a modular programmable array manipulator, such as PAM. The above mechanisms of demand-pull not only lend themselves to suitability as a control structure for PAM, but they also permit the distribution of the control system.

The need for a central control scheme is removed since each PAM module can have control over its own input and output. When many modules are combined to make a PAM, the control is therefore distributed and the modularity of the PAM is preserved. This demand-pull forms the basis of the control scheme developed for PAM.

2.6 Summary

This chapter has presented the areas of research covered in establishing the work for the PAM project. It covered the area of manufacturing automation as it is applicable to PAM. It is interesting to note that many large and complex manufacturing tasks can be compared to the control system required by PAM. This comparison is possible if one organises the jobs into smaller modules of processing and then establishes a cooperative framework for these jobs to act within. Such a framework is used in the manufacturing automation field.

The use of PAM as a part feeding device is a relatively new concept for the project. The initial intention of PAM was that of an intelligent conveying device to be used in
manufacturing systems. However, further research has revealed additional applications for PAM in the area of part feeding and manipulation. There is a need for reorienting parts received from an injection moulding device, for example, prior to their being fed into the manufacturing process. Attempts have been made to construct such a device [G1], [G2], but neither design has the flexibility provided by PAM.

Figure 2-4 Thesis Presentation Layout - Progress

This concludes the discussion on the theoretical control issues of PAM (figure 2-4). Chapter 3 will now present the hardware details associated with the development of a prototype PAM.
CHAPTER 3

PAM HARDWARE DESIGN

3.1 Introduction

As discussed previously, the programmable array manipulator is a device capable of manipulating objects in two dimensions. The actuator mechanisms required to perform such manipulations cannot be easily created. Potentially, a device which can push an object from its present position along a desired direction is complicated and expensive. On the other hand, the actuator required for PAM has to be simple and inexpensive as a large PAM could use several thousand of them.

The system used for the prototype PAM is based on the principles of vibratory motion. Actuators of this style are inexpensive and are simple to manufacture. The main disadvantage of such a mechanical arrangement is the electronic complexity of the hardware required to generate the control signal to the actuators. This complexity is caused by the generic nature of PAM which in practice is a multi-directional vibratory feeding device. This is a radical change from conventional part feeders which are generally uni-directional.

A number of trial PAM's have been developed during this project. Two of the devices are based on vibratory feeders of different types. During this chapter the principles behind the operation of each design will be briefly explained.
In order to understand the mechanisms associated with a particular actuator design, a model of the single element actuator device was derived. The main reason for this derivation was the realisation that experimental results alone would not provide a clear indication of the design parameters for a miniaturised version of such an element.

This simple model of the vibratory actuator is presented. It is the core of a computer simulation written to enhance the design of miniaturised versions of PAM actuators. The mechanical and electrical components of the trial PAM and the interface circuitry for its control will then be discussed and the design procedure of its elements will be described.

The mechanical systems described in this chapter were designed by the PAM research project group collectively.

3.2 Single Element PAM Actuator

The introduction stated that the actuation mechanism for the programmable array manipulator is based on the concept of electromagnetic vibratory motion. The use of vibro-feeding is well entrenched in many manufacturing systems. Typical applications include packaging, mixing, batching of bulk materials. These applications, however, can only produce movement in a single direction. The primary actuating device in these systems is an electromagnet which is supplied with an oscillating voltage. The armature of the electromagnet is attached to a bowl, or tray of some sort that then vibrates at the same frequency that the coil is excited with. This vibratory motion produces movement of the objects sitting in the bowl or tray.

3.2.1 Original PAM Actuator Concept

In order to determine the suitability of a vibratory style of actuator to the PAM project, the first prototype actuator that was constructed was a large single element actuator. This was necessary to prove that directional control could be obtained using vibratory
style actuators. In applications using vibratory feeding the direction of movement is singular. For the PAM, this would have to be changed for multi-directional motion.

The unit consists of a circular plate, approximately 270 mm in diameter suspended by three solenoids. This entire mechanism is driven in a circular orbit by a modified commercial orbital sander. Figures 3-1 and 3-2 illustrates this actuator concept.

![Diagram](image)

Figure 3-1 Single Element Prototype Side View

The plate orbits in the horizontal plane and is made to travel vertically, sinusoidally, with the same frequency as the plate orbit and with a phase locked relative to the plate orbit. The vertical motion produces frictional forces between the plate and the object resting on it which vary with the angle of the orbit. The net result is motion of an object in a straight line along the surface of the plate. By varying the phase of the vertical motion relative to the orbital motion, the direction of travel can be varied.
Using this prototype, experiments were conducted which showed that objects could be conveyed at speeds of up to 60 to 80 mm/s. However, it was difficult to predict the upper limit of these speeds since the limit using the prototype was a fraction of the tangential speed of the plate orbit.

Other design parameters that were needed in order to miniaturise the actuator with a high degree of confidence are not easily determined experimentally. This is primarily due to physical constraints. Such parameters include:

1. radius of orbit,
2. coefficient of static friction, and
3. amplitude of oscillation.

To try and understand (and thus predict) the speed and direction of movement of objects on the plate, a model and computer simulation of the system was devised. The results of this simulation were then compared to actual results for known properties. Since it is possible to vary parameters such as those listed above easily in the simulation, the design of the smaller actuator could be performed with a higher degree of confidence.
3.2.2 Single Element Actuator Model

The simple model used for the single element actuator is based on a model used for linear vibratory motion [G4]. When considering the mechanics of linear vibratory motion, it is convenient to represent the conveying track as a straight track which is inclined at a small angle to the horizon.

![Figure 3-3 Forces Acting on a Component on a Track [G4]](image)

The forces shown in this diagram are referred to as the external forces. These are the forces that will be experienced by an object when it is not moving on the track. If the track is excited sinusoidally, so that it oscillates forward and upward, the normal force per unit mass between the component and the track is given by [G4]:

\[
N = g\cos\theta - \omega^2 A_v \sin\omega t
\]  

(3.1)

where:

- \(N\) is the normal force per unit mass between the component and the track,
- \(g\) is the gravitational constant of acceleration, 9.81 m/s\(^2\),
- \(\theta\) is the angle of the track inclination,
- \(\omega\) is the angular frequency of vibration, rad/s, and
- \(A_v\) is the normal track vibration amplitude.
In the case of the single element PAM model, this value becomes:

\[ N = g - \omega^2 A_v \sin \omega t \]  

(3.2)

since \( \theta = 0^\circ \).

The frictional force per unit mass \( F_F \) acting on the component is given in [G4] as:

\[ F_F = \pm \mu_s N \]  

(3.3)

where:

\( \mu_s \) is the coefficient of static friction.

It is clear that if \( F_F \) becomes equal to \( |\mu_s| \) sliding between the component and the track will occur. In the case of PAM, this will be a driving force. Now, consider the mechanisms associated with the above system if the angle of inclination is zero. This will match the PAM case closely.

Another important consideration in the conveying of an object by PAM is the point at which the object leaves the plate. If the normal force becomes zero or less, then the object will leave the plate.

That is, the object leaves the plate when:

\[ g < \omega^2 A_v \cos \omega t \]  

(3.4)

When the object loses contact with the plate, it will have a particular velocity, \( v_1 \) and height, \( s_1 \). If \( \omega t_1 \) is the angular position of the PAM when the object leaves the PAM, then these values are defined by:

\[ v_1 = -\omega A_v \cos \omega t_1 \]  

(3.5)

\[ s_1 = A_v \cos \omega t_1 \]  

(3.6)
When the object leaves the track, it will travel in free flight until time \( \omega t_2 \) when it lands on the plate. Without solving for time \( \omega t_2 \), it is possible to calculate the point at which the object lands, since the simulation to be performed will be discrete rather than continuous. This is achieved by iteratively calculating the height of the object at each time interval of the simulation. Once the height of the object is less than the height of the plate, then it has landed. The object height is given by:

\[
s_f = s_1 + v_1 t_f - 0.5gt_f^2 \tag{3.7}
\]

where:

- \( s_f \) = height of object during flight, and
- \( t_f \) = accumulated flight time of object.

The height of the plate, \( s_p \) is defined by:

\[
s_p = A_v \cos \omega t \tag{3.8}
\]

Thus when \( s_f < s_p \) the object will have landed on the plate. Upon landing, the object will have a velocity given by:

\[
v_L = \mu (gt_f - v_1 - \omega A_v \cos \omega t) \tag{3.9}
\]

If a time interval is defined, \( t_{\text{step}} \), that represents the period of the orbit during which the behaviour of the object is to be calculated, then we can apply this time value to equation 3.10. This will result in the calculation of a velocity "impulse". The magnitude of this velocity impulse is:

\[
v_{\text{imp}} = 0.5\mu_s(g - \omega^2 A_v \sin \omega t)_{\text{step}} \tag{3.10}
\]

When the object lands, the net velocity is the sum of the velocities given by equations (3.9) and (3.10).

\[
v_{\text{net}} = v_L + v_{\text{imp}} \tag{3.11}
\]
If the velocity imparted on the object by the action of the modulated friction is not sufficient to move the object, the velocity of the object is the same as the velocity of the plate. The theory deviates from the linear vibratory theory, at this point, since the plate is actually travelling in an orbital path.

In order to simplify the mathematics, the orbital frequency is split into $x$ and $y$ components. If the plate is orbiting with angular frequency $\omega$, then the $x$ and $y$ velocities are given by:

$$v_{x\text{plate}} = \omega \cos(\omega t) \quad (3.12)$$

$$v_{y\text{plate}} = -\omega \sin(\omega t) \quad (3.13)$$

It is worthwhile noting that the value of $\omega$ in these equations is the same as the $\omega$ that is exciting the vertical motion of the plate. This is a key factor in controlling the object on the single element PAM; the frequency of the vertical oscillation is locked to the orbital frequency.

The magnitude of the plate velocity is calculated as:

$$v_{\text{plate}} = \sqrt{(v_{x\text{plate}})^2 + (v_{y\text{plate}})^2} \quad (3.14)$$

In order to determine if the object will move, the relative velocity between the plate and object should be determined. Define:

$$v_{x\text{obj}} = x \text{ component of the object velocity,}$$

$$v_{y\text{obj}} = y \text{ component of the object velocity,}$$

$$v_{x\text{rel}} = \text{relative } x \text{ velocity component, and}$$

$$v_{y\text{rel}} = \text{relative } y \text{ velocity component, such that}$$

$$v_{x\text{rel}} = v_{x\text{plate}} - v_{x\text{obj}} \quad (3.15)$$
\[ v_{yrel} = v_{yplate} - v_{yobj} \]  

(3.16)

Now, calculating the relative velocity magnitude, \( v_{relmag} \):

\[ v_{relmag} = \sqrt{v_{xrel}^2 + v_{yrel}^2} \]  

(3.17)

If the impulse velocity magnitude (eqn 3.10) is greater than the relative velocity magnitude, then the object cannot move and the velocity of the object is the same as the plate velocity relative to the object.

\[ \text{ie. } v_{imp} = v_{relmag} \]  

(3.18)

Now, the \( x \) and \( y \) components of the object velocity impulse can be determined:

\[ v_{xobjimp} = v_{xrel} \times \frac{v_{imp}}{v_{relmag}} \]  

(3.19)

\[ v_{yobjimp} = v_{yrel} \times \frac{v_{imp}}{v_{relmag}} \]  

(3.20)

From this result, a new \( x \) and \( y \) component of object velocity can be calculated, as can the distance travelled by the object during the \( t_{step} \) interval.

\[ v_{xobj} = v_{xobj} + v_{xobjimp} \]  

(3.21)

\[ v_{yobj} = v_{yobj} + v_{yobjimp} \]  

(3.22)

\[ s_{xobj} = s_{xobj} + 0.5 \times v_{xobj} \times t_{step} \]  

(3.23)

\[ s_{yobj} = s_{yobj} + 0.5 \times v_{yobj} \times t_{step} \]  

(3.24)

where:

\[ s_{xobj} = \text{distance travelled by object in the } x \text{ direction during the } t_{step} \text{ interval, and} \]

\[ s_{yobj} = \text{distance travelled by object in the } y \text{ direction during the } t_{step} \text{ interval.} \]
This completes the derivation of the model used to predict the behaviour of an object on the single element PAM actuator. The model assumes the following:

1. the motion of the orbiting plate is unaffected by the presence of the object,

2. the coefficient of static friction is constant,

3. the shape of the object has no effect on the behaviour, and

4. after the object lands, it does not bounce.

Assumption 3 essentially limits the object being modelled to that of an object with a flat base.

### 3.2.3 Single Element Actuator Model Simulation

In determining the mechanisms that cause an object to move on the single element PAM, there are a number of different forces and conditions applied to the object that need to be considered. The use of the simulation package allows the study of the object behaviour on the single element PAM beyond that which is experimentally measurable due to mechanical limitations. The simulation package also contributes to the understanding of the model derived above.

In order to understand the simulation, first consider the pseudo-code for the simulation program as presented in figure 3-4.

Initialise variables
Prompt user for:
- plate orbital frequency, Hz
- radius of plate orbit, mm
- coefficient of static friction between plate and object
- initial x component of velocity, cm/s
- initial y component of velocity, cm/s
- angular resolution for each simulation step, degrees
- total simulation time
Convert user data to SI units
Set up the screen for plotting the results of the simulation
While(actual simulation time < total simulation time)
Resolve the orbital frequency to x and y velocity components (eqn 3.12, 3.13)
Calculate the relative velocity between plate and object
Calculate the friction between plate and object (eqn 3.3)
If(friction < 0)
   If(air flag is set)
      If(first air flag set)
         Calculate object vertical displacement (eqn. 3.6)
         Calculate object vertical velocity (eqn. 3.5)
         Remember time this occurs
         Clear first air flag
      Endif
   Friction = 0
   Calculate object flight time
   Calculate object height based on flight time (eqn. 3.7)
   Calculate plate height (eqn. 3.8)
   If(object height < plate height)
      Object has landed
      Set land flag
      Clear air flag
   Endif
Endif
Endif
Calculate the x and y velocity components imparted on the object by the frictional force
If(land flag is set)
   Calculate velocity impulse to object when it lands
   Clear land flag
Endif
If(frictional force is not sufficient to move object)
   Object velocity = plate velocity relative to object
Endif
Accumulate the velocity of the object
Plot displacement of the object path on the screen
Plot position of the plate on the screen
Increment simulation time
End

Figure 3-4 Single Element PAM Simulation Pseudo Code

The next section provides a discussion of the results obtained from the simulation. In particular, the results that compare the simulated performance with the experimental one are presented. The source code for the simulation software is presented in Appendix C.
3.2.4 Single Element Actuator Simulation Results

The model and the above simulation produced results that were in agreement with the experimental trials using the single element actuator. This was an encouraging result since it meant that the performance of an object on the PAM could be predicted with a high degree of confidence.

The single element actuator under consideration has the following physical properties:

1. Coefficient of static friction between surface and test object 0.50
2. Radius of orbit 0.75 mm

The amplitude of the oscillation could be varied by adjusting the tension of the springs responsible for the suspension of the orbiting plate (figure 3-1). The following figures show both the results of the simulation and the results obtained experimentally for the above scenario.

![Vertical Amplitude Vs Velocity (r=0.75, u=0.5)](image)

Figure 3-5 Simulated and Measured Vertical Amplitude Vs Velocity
Additional results are given in Appendix C.

In order to determine the velocities capable of being produced, further simulations were performed. These simulations were performed for different orbital frequencies with various static friction coefficients, orbit radii and vertical amplitudes of oscillation. From this work, it was determined that higher orbital frequencies for a fixed orbital radius lead to directional instability. This trend can be observed in figures C-2, C-4, C-6, C-8 and also in figure C-10. Thus the expected operating frequency of such a PAM would be in the order of 20 Hertz.

The velocity of the object across the plate is controlled by manipulating the radius of the orbit, rather than the orbital frequency. The radius of orbit was chosen to be the same as that for the single element actuator. The prime reason for this decision was the mechanical drive system devised for the prototype. The simulation also reveals that there is little effect from the coefficient of static friction on the speed and direction.

3.3 Multi-Element PAM Actuators

In the first prototype multi-element PAM, the primary actuating device is the solenoid, the arrangement of which is shown in figure 3-7.
The action of this solenoid is simple. As soon as the solenoid is energised the armature pulls in to the minimum air gap and consequently raises the tile. Release the solenoid, and the spring pulls the armature back down to the balanced position. The stroke of the solenoid can be adjusted by the pre-load and level adjustment screw at the base of the armature.

![Figure 3-7 Vibratory PAM Mk1 Actuator Arrangement](image)

This is the arrangement style used in the 36 element trial array. The larger 512 element array under construction uses a different technique. This method is shown in figure 3-8.

In this design, the solenoid is held in place at its shoulder. The rest of the solenoid, from the body to the end of the armature, is housed in a pressurised chamber. The action of the air pressure on the rubber diaphragm is used to pull the solenoid down to the lowered position. This has advantages over the previous arrangement because the
'spring' rate can be varied by regulating the air pressure. The flow of air around the solenoid provides a mechanism for improved heat transfer.

Figure 3-8 Vibratory PAM Mk2 Actuator Arrangement

In both cases, the solenoid arrangement is mounted on a flat plate which undergoes a circular orbital motion with a frequency of approximately 25 revolutions per second at a radius of 0.5 mm. The solenoid is excited with a pulse width modulated (PWM) signal that attempts to force the solenoid armature to follow a sinusoidal displacement with the
same period as the orbital frequency of the mounting plate. Figure 3-9 demonstrates the type of solenoid displacement that is being sought.

Figure 3-9 Vibratory PAM Ideal Solenoid Displacement

By forcing the solenoid to follow such a displacement, the objects that are in contact with the tile at the top of the solenoid are vibrated up and down at the same frequency as the orbit of the plate. In doing so, the viscous friction between the PAM surface and the object is modulated between a minimum and maximum value. When the tile is accelerating upwards, the viscous friction will reach the maximum value. It is during this period that the object will experience an acceleration which forces it to follow the orbital path that the tile is moving along. The tile will then accelerate downwards and the object will stop following this path since the viscous friction is reduced considerably during this phase. It is this repetitive action which causes the object to move along the desired path.

Now it is possible to control the direction of the resultant force imparted on these objects by controlling the phase of the peak solenoid displacement with respect to the orbit of the plate. The reference point of the plate orbit is always the same and is defined by a proximity style limit switch arrangement.
Therefore, in order to be able to control the direction of movement the object being manipulated will follow, it is necessary to devise hardware that can produce a relative phase varying pulse width modulated (PWM) signal, under the command of the PAM control system.

3.4 PAM Mechanical Modularity

In order to devise a modular PAM arrangement, the mechanical design of PAM is broken down into modules. The trial PAM with 512 elements is made up of 32 Actuation Modules, or AM's. Each AM has 16 solenoids. Figure 3-10 shows a sectioned view of the actuation module.

![Figure 3-10 Section View of 16 Actuation Module](image)

Note that each module is a self contained unit comprising 16 actuation elements. Each group of 4 Actuation Modules has a driver circuit board associated with it for powering the solenoids. The entire PAM is made up from combinations of these modules. The combination of 32 of these modules makes up what is referred to as a Manipulation Module. Figure 3-11 shows a plan view of the 16 actuator module.
3.5 PAM Control System Modularity

The idea of PAM as a modular reconfigurable conveying system enhances the generic capabilities of the device. In chapter 1, PAM was described as a modular device. This was justified because the mechanical design of the PAM allowed it to be put together like a jig-saw puzzle. That is, modules containing 'x' actuators could be combined to form a single cohesive PAM. In order to do this, the structure of the control system also needs to be arranged in a modular fashion. The hardware and software of the control system should be modular in the same way that the mechanical structure is.

The modern control system is typically digital, which requires a piece of software to run on a processor. In PAM, the processors controlling each manipulation module must be loosely coupled and communicate with each other. Conventional microprocessors are usually interfaced to each other through an ad-hoc interface which generally increases the complexity of the software, design development and maintenance.

In order to avoid this problem, the transputer has been chosen as the computing platform of the PAM control system. Using the transputer, the issue of processor to processor communication is taken care of systematically through the serial links of the transputer. This link is not only transparent to the programmer but built into the programming languages developed for the transputer.
Having decided to use the transputer, the distribution of the control scheme onto a transputer array needs to be addressed. The number of transputers required and the topology by which they would be organised should also be determined. Other factors that may influence the physical arrangement are the availability of commercial transputer hardware to perform the tasks expected of it, and the ability to connect the network as the designer would like. The T800, T400 generation of transputers are constrained to 4 serial communication links. This may influence the number of processors and the topology.

The image processing to be performed for the PAM is also based on the transputer. The transputer module allocated for this task is a Transtech TTGF-4 TRAM. The image processing is a time consuming task, and hence the TTGF is not realistically capable of performing other tasks. The interface to the PAM actuators will be via one of the transputer communication links. In the current design, a further processing TRAM is used for the control strategy and a third TRAM is to be used for the actuator control and coordination. A fourth TRAM, one dedicated to graphics output, is used for debugging the image processing tasks.

The above arrangement is referred to as a control module, or CM. The inclusion of three transputers in the control module means that the task of networking such modules is possible without additional hardware. Of course if the processing requirement increases, then adding further processors is a trivial task. The arrangement of the Control Module is illustrated in figure 3-12.

The number of available links of the transputer is currently limited to four and is a constraint on the network topology of the CM's. Each Control Module can be surrounded by other modules and still must maintain the required connectivity. Each of the outgoing links, Link 0 to Link 3, are used for networking with other control modules. This network constitutes a considerable capital investment, if the hardware purchased is a standard commercial product. For the purpose of research and
development, it could be considered more convenient to purchase such products to reduce development time. However, as the project evolves, then design and development of custom hardware could well become more attractive.

For example, the frame grabber TRAM consists of a T805 transputer, a real time image digitiser and colour look up table (for driving a video output signal), 4 MByte of DRAM and 1 MByte of VRAM (video RAM). It is also capable of receiving 3 video inputs. To reduce the cost, a frame grabber could be used in conjunction with three modules rather than one. The flexibility of the transputer allows this approach. The TRAM, however, needs to perform image processing for three separate images, and communicate with three processes. This takes too much time for it to be a real time application. A customised design of the frame grabber, however, can provide a more appropriate system for PAM.

In a commercial application of PAM, it would certainly be more desirable to build custom hardware to perform the task required. This would reduce cost significantly and
could possibly lead to a more compact hardware system. Instead of using TRAM's, a purpose built circuit board, with all the necessary hardware on board could be designed.

3.6 PAM Actuator Interface Hardware

There are two different drive circuits designed for the 36 element and 512 element trial PAM's. Each type, although similar in action, is very much different in the way a PWM signal is derived and subsequently applied to the solenoid. The hardware used for interfacing to the 36 element array is to be presented. This was a concept proving device since the previous method used in the single element PAM was designed using analog circuitry. It had to be established that the digital design devised would provide directional control of objects on PAM as desired. Secondly, the communication between the PAM control system and the PAM actuators had to be fast and be able to cope with potentially thousands of actuators.

The technique of establishing the bit pattern to produce a PWM signal that will force the solenoid to produce a sinusoidal displacement pattern will also be presented.

3.6.1 PAM Actuator Interface, Mark 1, Overview

The conceptual design of the circuit used to drive the solenoids in the 36 element PAM is presented here. The detailed circuit design is far too long to be included in the main body of this report, and is therefore given in Appendix A.

In order to produce an appropriate actuator motion, two factors need to be controlled by the control system:

1. the shape of the displacement versus time characteristic, and

2. the relative phase of the minimum solenoid air gap with respect to the orbit of the PAM.
The shape of the displacement characteristic is important since the actuator should not throw the object being manipulated into the air. Hence, the motion of the solenoid armature should have a gentle acceleration rather than a jerky movement caused by simply energising the solenoid through a fixed voltage. In practice, it is desirable that the armature tip follow a sinusoidal pattern, although this is quite difficult to achieve. This hardware is to communicate to the transputer network via a synchronous serial communication link.

The design of the circuit was implemented digitally using a programmable logic approach. The one used for this work is manufactured by XILINX. This technology was chosen because of the ease of design modification. The high gate count supported by this hardware is an advantage, since there is a considerable amount of logic associated with the task at hand. With such an amount of logic, realisation of the design on anything but programmable logic would necessitate the use of an unrealistic number of components for what is essentially a prototype device. Such a large number of logic components would also require several printed circuit boards and associated connections thus making the prototype expensive. The design was implemented on a two layer printed circuit board using 5 programmable logic IC's and several smaller logic devices.

3.6.2 XILINX Hardware Overview

XILINX programmable logic is based on a Logic Cell Array (LCA) architecture which is their own proprietary design and nomenclature [H2]. The architecture is similar to that of other gate arrays, with an interior arrangement of logic blocks and surrounding input and output resources. Interconnect switches occupy the region between the logic blocks and input/output resources.

The LCA device is a program driven logic device, like a microprocessor. The functions of the LCA logic blocks, input blocks, output blocks and their interconnection are controlled by a user configurable program stored in on-chip memory. The program can
be loaded automatically on command, as for a microprocessor, from an external memory device such as an EPROM, or it can be downloaded via a serial communication link from a host machine. The devices can be reprogrammed repeatedly without the need for erasure. The device is loosely based on SRAM technology and as a consequence, the program memory is lost when the device is powered down. To allow for this, the devices have a power down state, which allows battery back-up to be used. Alternatively, configuration via an EPROM is used.

The process of designing logic to be downloaded to the logic devices is based on the schematic capture approach. In our case, the ViewLogic Workview package was used. Design was a question of using the correct logic macro components and connecting it all together as one would for conventional logic. The circuit diagram for this logic is included in Appendix A. Inspection of these drawings will reveal that they are conventional circuit diagrams, using specialised symbols. Once the logic has been designed, it undergoes an automated translation process in readiness for downloading to the logic arrays.

The outputs from the chip do not have a high current driving capability. Each output is limited to 5 volts @ 4 mA. The largest chip used by this project, the XC3090PG175 has 145 available I/O lines. The logic gate equivalent is 9000 logic NAND gates.

3.6.3 PWM Solenoid Excitation Concept

As with most digital systems, the control of the voltage or current in a device, such as a solenoid, is implemented using the principle of Pulse Width Modulation (PWM). In order to be able to control the displacement versus time characteristic of the solenoid actuator, one needs to be able to control the current. In this particular application the solenoid is excited with a PWM version of a sinusoidal voltage that causes a sinusoidal response.
Basically, PWM is nothing more than the high frequency switching of a voltage applied to a load. In the case of the solenoid actuator, a bit pattern is applied to the base of a power transistor that is driving the solenoid.

![PWM Solenoid Excitation](image)

The basic arrangement of the solenoid driver circuit is shown in figure 3-13. The key to the control system hardware is the generation of the bit pattern to drive the base of the transistor.

### 3.6.4 PWM Bit Pattern Calculation.

The bit pattern required for the solenoid driver circuit needs to be a binary representation of the desired voltage waveform. In this instance, the desired waveform is a sinusoid. An algorithm was developed which can produce a bit pattern of fixed length, for any particular waveshape.

To simplify the process of generating the bit pattern, a short C program was. The program is not highly automated, since for each waveshape the user wishes to digitise, the function is entered manually in the source. This was regarded as sufficient for this work. In order to explain the algorithm used, the flow diagram of the algorithm is produced in figure 3-14.
The function that is to be synthesised is evaluated at 256 different points. For example, if a sinusoidal waveform was to be digitised in this fashion, the function that would be evaluated would be:

\[ 0.5 \times \cos \left(2 \times x \times \frac{\pi}{256}\right) \]

where \( x \) ranges in value from 0 to 256 and \( \pi \) is 3.1412\ldots.

The full source code for the program is included in Appendix B. The output of this example is also included in the Appendix.
3.6.5 PWM Phase Control Philosophy

The direction of an object motion on PAM is determined by the phase of the bit pattern which is controlled with respect to a reference point on the orbit. This reference position is measured by a proximity limit switch that senses a small steel block attached to the orbiting plate of the PAM.
In order to illustrate the problem consider the simple case of a square wave bit pattern exciting the solenoids. The bit pattern could be:

00000000111111110000000011111111 (i)

The bit stream is 16 bits long and is applied to the solenoid once every orbit of the PAM. The reference point for the bit pattern is the left most significant bit. If the above bit pattern represents the pattern for 2 consecutive rotations, then the bits would be shifted right to left such that the frequency of the bit shift is equal to 1/16th of the orbit frequency. From this information, we can see that the solenoid will be excited by a '0' for half of the orbit period followed by a '1' for the rest of the orbit.

Now, consider the following bit pattern:

00001111111100000000111111110000 (ii)

The pattern has the same number of 1's and 0's, but they occur at different places in the pattern. Thus the phase of the solenoid excitation with respect to the start of the bit is different from the previous case. This is illustrated in figure 3-15. It is assumed that the solenoid has a perfect response. The solenoid is being excited by bit pattern (i) as shown in figure 3-15(a). The peak of the solenoid response occurs at a particular point in time. When the solenoid is excited by bit pattern (ii), the response of the solenoid changes with respect to the reference point of the orbit, as shown in figure 3-15(b).

Figure 3-15 PWM Phase Control Philosophy
Therefore, by changing the point at which the bit pattern is first applied to the solenoid, different solenoid responses can be generated. Thus, for a 16 bit PWM pattern it is possible to generate 16 different phase responses. Due to the relatively slow response of the solenoid it is difficult to differentiate between the response when the bit pattern is started shifted only one or two bits to the left.

3.6.6 PWM Reference Point Synchronisation

In the example given in the previous section, the reference point for the bit pattern shift frequency was known. In reality, however, the pulses for the synchronisation of the PWM signals come from a limit switch mounted on the PAM itself. Furthermore, the speed of the orbit of the PAM is not necessarily constant, since there is no closed loop speed regulation of the drive system. A method had to be devised such that the bit shift frequency is calculated each orbit of the PAM.

In order to explain the operation of the logic further, consider the following diagram:

Figure 3-16 Synchronisation Logic
The signal labelled GCLK, (a), is a 1 MHz clock signal derived from an external 8 MHz crystal. It clocks a 17 bit counter, that is reset on the falling edge of a pulse from the proximity limit on the PAM orbit mechanism. This limit switch signal, IPROX, is first conditioned by additional logic to provide a single pulse with a width of 1μs at each rising edge of the signal (b). It is also synchronised to the rising edge of a clock pulse. The least significant 8 bits of this counter are ignored. The next 8 bits are shifted into a first-in-first-out register (c) on the rising edge of the IPROX pulse. The 17 bit counter is reset on the falling edge of this pulse. This operation effectively loads the FIFO register with the number of bit shifts required, in a single IPROX period, to successfully shift a 256 bit pattern by 256 bits.

Consider further the logic behaviour if a fixed 25 Hz signal was used instead of the IPROX signal. If GCLK was a 1 MHz clock, then the counter will reach 1000000/25 (40000) counts before being reset. The FIFO register will be loaded with 40000/256 on the rising edge of the IPROX pulse. This value is therefore the bit shift rate for a 256 bit PWM signal, and it would be the value observed at the output of the FIFO register.

The GCLK signal also clocks an 8 bit counter, (d). The output of this counter is then compared to the value in the FIFO register (in fact this occurs one IPROX cycle later), and if the same, then a single pulse is generated, labelled SRCLK (e). This shifts the 256 bit pattern by one bit elsewhere in the logic. Based on the example just given, the frequency of SRCLK would be 40000/256 pulses per second. The SRCLK signal is then used to shift the bit pattern used to excite the solenoids.

The bit pattern is reloaded once every revolution of the PAM orbiting plate, regardless of the current status of the shifted word. The bit pattern is stored on the XILINX chip as a series of 256 D type flip flops, with the D input tied to either a 1 or a 0 as appropriate.
3.6.7 Phase and Solenoid Selection Logic

The method used to vary the phase of the excitation applied to the solenoid has been presented. This logic is implemented on a single XC3090 XILINX chip. The output of this chip is 32 lines of PWM signals providing 32 different phases of bit patterns. These bit patterns can then be selected by further logic for use in driving the solenoids. This additional logic is distributed between four XC3090 chips, each one being responsible for 9 solenoids. This logic must comprise two distinct blocks:

1. an interface with the control system, and
2. logic for selecting the phase and solenoid and the appropriate output.

The control system has the ability to choose any of 32 different phase possibilities and applies it to any one of 36 solenoids. The control system communicates to the phase and solenoid selection logic via a 5 MBit/sec synchronous data communication link. The format of the transmitted word is; 2 start bits (both logical 1's), 8 data bits, and 1 stop bit (logical 1). Hardware was designed to interface with the transputer control network to provide this particular protocol. It is the responsibility of the control software to select the phase and solenoid requirement separately and transmit the data via this communication link.

The serial communication decoding hardware is simply a series of self-resetting shift registers. The data transmitted by the control system contains all the information required for decoding the message. The message is 3 bytes of data, and when it is combined with start and stop bits, there will be 33 bits of data transmitted synchronously.
Referring to figure 3-17, (a), (b), (c), and (d) are 8 bit shift registers that clock the data presented to it by one bit to the left on the falling edge of each clock pulse. When the first start bit arrives at Q7 of register (d), it means that all data has been received by this logic except for the last stop bit. At this point, the first start bit is then "ANDED" with the communication clock produce a DATAVALID signal (h) one clock period in length. On the falling edge of this clock pulse, the last stop bit is shifted through and all four shift registers are cleared by a D type flip flop (e). This flip flop is then cleared on the next falling edge of the clock.

It is worth noting that the clock signal is a continuous 5 MHz clock. It does not stop and start with the data. The data is synchronised to the clock edges at the transmission end.

When the DATAVALID signal goes high, the data is decoded further by additional demultiplexing logic. Recall there are two pieces of information to be extracted. One is the phase that a solenoid is to use and the other is the solenoid number. There are a total of 32 phases to choose from, so 5 bits of data are required to encode this information. There are 36 solenoids to select from, so 6 bits of data are required. Even though each decoder is responsible for 9 solenoids, the control system is not aware of the
segregation of this logic. It is merely sending out a phase selection command to a particular solenoid.

It is now possible for this data to be demultiplexed by extra logic. This logic is four, 3 to 8 demultiplexers and a 2 to 4 demultiplexer. As shown in the schematic SOLCON3.1 in Appendix A, a single signal line is produced out of this block of decoding logic. This signal line is the selected PWM phase. Each decoding block is also associated with solenoid selection logic. This logic only requires to decode which solenoid is being addressed. This is achieved by boolean expressions using the 5 solenoid select bits from the received data.

3.6.8 Alternative Actuator Interface Scheme

The hardware described in the previous section allows 32 different directions of movement. The design has several drawbacks:

1. it is not a cost effective solution, and

2. it takes up too much room.

The reasons behind these conclusions are worthy of further discussion, as one aim of this work is to produce a cost effective actuator for PAM. For the purpose of research and development, the hardware devised is adequate and acceptable. Expansion of this design, however, to control a PAM with 512 actuators is practically difficult. The amount of logic hardware required for such a controller would be enormous. If each group of 9 solenoids required a decoder/driver logic block, then this would involve using some 57 XILINX XC3090 integrated circuits. These items cost approximately $250.00 each. Obviously this cost cannot be justified. In addition to this direct cost, there is also the expense of the printed circuit board design and manufacture. An estimate of the cost of the interface hardware is $45.00 per actuator which is commercially quite unreasonable. A cheaper, equivalent solution is required.
This solution is provided by using high speed processing and digital logic. The generation of a PWM bit pattern is possible using software and a microprocessor. It is necessary for the processor to have the bandwidth to be able to generate the PWM patterns at the appropriate rate. This design is currently being developed for the 512 actuator PAM. It utilises four digital signal processors, four gate array chips and a microcontroller. The microcontroller is used to boot the code in the digital signal processor and the gate array.

There are significant cost savings in using this approach. Instead of having multiple boards to accommodate the interface logic, it can all be done on a single board. This single board has the digital signal processors, programmable logic, microcontroller and EPROM. Additional circuit boards are then required for the power transistor drive of the solenoids. It is estimated that a cost of about $8.00 per actuator will be achieved with the new design.

3.7 Summary

This chapter has described in detail the hardware designed to control the actuator elements of the PAM. The hardware is suited to a particular style of actuator, the vibratory element. The motion imparted on an object being manipulated by PAM is based on the principle of vibratory motion, as found in many part feeding applications in manufacturing. The ability to position and orient the object is made possible by controlling the relationship between the armature stroke modulation of the solenoid and the orbit of the PAM plate mechanism.

The actuator control hardware communicates with the transputer control system using one of the four available serial communication links of the transputer. This application extends the concept of communicating sequential processors as applied to parallel processing tasks.
The notion of PAM as an reconfigurable, modular device has been further enhanced by the use of transputers in the control system. The construction of a control module, means that the hardware has been established that will allow the development of a modular control strategy for a PAM. Thus the modularity has been extended to the maximum possible extent in the hardware design; both mechanically and in the control hardware.

Figure 3-18 presents the progress of the work presented so far. Now that the actuator and the hardware to control the actuator has been designed, it is possible to commence work on the control system. The next chapter deals with this issue. The control devised will need to control an object on the 36 element PAM. However, the system design should be generic so that it can deal with different sized PAMs.
CHAPTER 4

PAM CONTROL SCHEME

4.1 Introduction

This chapter will describe the control scheme philosophy for the PAM device. The system devised for PAM could be configured in several different designs and a discussion about various control topologies will precede the main discussion about the system devised for PAM. There is an underlying design principle associated with any control scheme for PAM and that is to enhance the modularity of the device.

PAM is capable of performing many functions. Accordingly, the control can be separated into several levels of tasks; what to do and how to do it. The manipulator can be configured to be a simple conveyor, or a complicated parts feeding and reorienting device, or some combination of these. The controller should be able to deal with any task presented to it.

Recall that the Programmable Array Manipulator is physically constructed from combinations of Manipulation Modules (MM's). Each MM has a particular number of actuation elements capable of imparting the motion on the object being manipulated. Each MM also has a Control Module (CM) assigned to it. It is the responsibility of the CM to implement the necessary control actions of the actuators to create the desired object movement.
The control scheme will first be presented as a theoretical formulation, then the mechanics associated with the implementation of these concepts will be discussed. The task expected of a PAM involves the control of an array of actuation elements. There are several issues to consider, each of which could be described as being at a different level of the control. There are also several possible topologies of control schemes that could be applied to PAM. The reasons why one of these schemes was chosen as the candidate for the control are given.

4.2 Centralised Control Structure

One such scheme could be based on what could be termed as a Centralised Control System. In such a scheme, one control module would be allocated as a Central Controlling Unit (CCU), or System Controller. It is then the task of this module to coordinate and control all of the actuators associated with the PAM.

The implementation of a centralised control architecture seems to present a scheme of reasonable simplicity upon initial investigation. However, this is not necessarily the case. Many factors need careful thought before a conclusion may be drawn. The task list presented earlier in chapter 1 is a non-trivial list. Some of the tasks required of the control scheme have large computational burdens. In particular, the high speed vision system for providing the position and orientation of multiple objects would be in itself a burden. The tolerable processing overhead would of course be a function of the required accuracy and the speeds of the objects as they traversed the array.

Consider the case of a large array of work cells. In order to maintain measurement accuracy for the vision system, more than one camera would be required. Clearly, there is a limit on the positional measurement accuracy that is associated with:

1. the resolution of the camera and

2. the viewing area of the camera.
Thus, if the array of work cells were to cover a large area, then additional processors would be required just for the image processing.

In much the same way that the vision system becomes a computational burden, so does a large tracking system. The array actuator controller and indeed the entire control process grows proportionally in complexity. Eventually, the processing power may not be able to execute the task in real time. As a result an additional processing unit should be added to deal with the demands of the control system. The inclusion of multiple processing units now changes the hierarchy of the control scheme. At the start, it was suggested that a single centralised controller could be used for the scheme. Now, due to the computational burden of the control scheme, several processors would be required. This strategy adds a further level of complexity to the control of the array of work cells. This additional level is the interaction between the two processors.

As an object traverses the array of work cells, it will pass from the control of one processor to that of the other. At the boundary, the two processors must interact to allow a smooth transition. To maintain the central control scheme, a master processor must be introduced to coordinate and control the subordinate units. There are now three controllers, one which is the master plus two slaves.

Let us assume the level of complexity of the array of work cells grows much further and multiple processors are required to implement the control scheme. There is now yet a further level of complexity added to the control scheme. The question of interprocessor communication is no longer a trivial case as it was with two processors. In the multiple case, there is very likely a need for an upper hierarchy that can coordinate all the control units. In this situation the communication of the processors with the CCU becomes an important issue.

In the work reported by [F7] a serialised and multiplexed communication system is proposed for a system of distributed micro-systems. The concept of a high speed
communication network is suggested as a method for solving the problems associated with networking multiple processors. Such an arrangement is illustrated in figure 4-1.

![Centralised Control Architecture](image)

Figure 4-1 Centralised Control Architecture [F7]

Nevertheless this technique does not resolve the problem of multiple processors communicating with multiple sensors and actuators cooperatively.

There is another scenario which can also highlight the limitation of a centralised control scheme. Consider an area consisting of a 5 by 5 grid of work cells. At one end of the grid, the input side, a feed system delivers several differently coloured disks. Such an arrangement is shown in figure 4-2. The task is to organise these disks into a predefined pattern on a single work cell at the other end of the grid, the delivery side. The pattern would, for example, involve the ordering of the disks into some desired colour scheme. At the delivery end of the grid, there is some other mechanism for removing the disk arrangement such that they are passed on to the next process.
The disks are being delivered as a result of some other process upstream. There is an orderly delivery of the disks so that there is an even distribution of colours. The only control the grid has over this process is to stop and start it.

Figure 4-2 "Five by Five" Manipulation Module Arrangement

Suppose that the 5 by 5 grid of work cells is to be extended, or mechanically reconfigured so that the number of objects required on the array rises to 8. It is obvious that the previous control strategy cannot cope with this new situation and the rules of the control scheme should be re-authored.

Figure 4-3 Alternative Manipulation Module Arrangement

This is of course a poor method of dealing with what is essentially a simple change. Additionally, this will impair the flexibility of the system. It is possible to take into
account every possible combination of operating conditions given a fixed mechanical
arrangement. It is, however, extremely difficult to compensate for every possible
change in mechanical configuration. Any attempt to do this will be a waste of valuable
computing resources.

The main disadvantage of the centralised control architecture is its inflexibility which
puts a constraint on the expansion of the system. In addition, the complexity of a
centralised control scheme becomes excessive when a large number of objects and
manipulator elements is considered.

4.3 Distributed Control Architecture

An alternative to the centralised control architecture is a distributed system. Such a
scheme provides a degree of flexibility that enhances the modularity of the
Programmable Array Manipulator. The platform allows easy expansion and
reconfiguration with little effort. If the control software is structured appropriately, then
no changes need to be made when reconfiguration or resizing takes place.

Such a control scheme adopted for PAM can be best referred to by distributed,
cooperative parallel control. The scheme is distributed since there are multiple
processors (and processes) of equal status, none being master or slave. The cooperative
term reflects the nature of the control scheme. The control architecture is structured so
that the control modules may interact in a way that no need for a central planning
process exists. The scheme is parallel, because it has several processes occurring
simultaneously.

The proposed control scheme has many similarities with the robotics system referred to
as the Distributed Manipulation Environment [F4]. According to (Naghdy and
Strickland), such an environment has the following features:

1. modularity: the controller is formed by the integration of a number of modular
   and simple components,
2. simplicity of design, and

3. simplicity of maintenance and development.

The distributed, parallel cooperative control scheme certainly fits into the mould of the Distributed Manipulation Environment. It is this style of control scheme that is applicable for the PAM. All the desirable qualities can be created by adopting such an architecture.

4.4 Hierarchical Distributed Control Structure

The scheme designed for controlling the PAM has a hierarchical, multi-layered structure. The layering is a result of a number of tasks required to be performed in parallel whilst controlling the PAM. The most difficult part of this structure is the control software that is responsible for the planning and coordination of the manipulated objects. The basis for this control operation is the demand-pull mechanism commonly found in manufacturing automation. The other tasks that the control is responsible for are much the same as those proposed in chapter 1.

The various layers of the devised control scheme are as follows:

1. Element Control Layer, or Actuator Control Layer

2. Object Control Layer,

3. Vision System Layer,

4. Command Layer, and

5. Communication Layer.

The layered approach of the control automatically introduces a structure in the program design which means that a review of certain aspects of the control scheme need not necessarily mean a rewrite of the entire code.
4.4.1 Element Control Layer

The Element Control Layer (Element Layer) is responsible for the interface between the Object Control Layer and the PAM elements. It is the lowest level of the control hierarchy, but it has an important role. Figure 4-4 shows how this layer fits in with the proposed hierarchy.

![Control Layer Structure - Element Control Layer](image)

Figure 4-4 Control Layer Structure - Element Control Layer

It is necessary to have a layer at this level to isolate the control command requirements from the next level. The object control layer is not necessarily "interested" in the type of PAM it is interfacing to. The PAM could possibly consist of one of many different actuator styles. Although in this instance there is only one type of actuator to interface to, this could be changed in the future should a new style of actuator be developed.

The knowledge required by the Element Layer is the commands necessary to operate an actuator according to the instructions of the object layer. For example, the object layer requires a set of actuators to move an object in a particular direction. Hence the object layer issues the appropriate command. The element layer takes this command and translates it into a command that the control hardware can respond to. The research performed so far demonstrates the value of the layered, or modularised, control approach. For example, the 36 element array uses different actuators to the 512 element array. The change, however minor, means that one software module can simply replace an existing module instead of trying to modify a much larger code segment.
The element layer must transmit the following messages:

Message 1: Solenoid Number, and

Message 2: Phase Number.

For the 36 element array, the first message can be contained within one byte, since there are only 36 solenoids. However, in the 512 element case, two bytes need to be transmitted. The synchronous format of the communication protocol requires that all three bytes of data be transmitted without time gaps. This is a function of the hardware, but the layer needs to be aware of this limitation. To deal with it requires the assembly of a complete message prior to transmission. This is opposed to sending the solenoid number then waiting to transmit the phase number.

The phase number describes the desired direction for the motion of the object. In both cases, the number of phases is limited to 32 and thus the data can be transmitted in a single byte. For a different type of actuator, say binary with simple on and off control, the element control layer will translate the message into a somewhat different format. The object control layer, however, need not know the difference between the two styles of actuator. It is only concerned with moving an object in a particular direction and not the mechanisms required to do this.

4.4.1.1 Translation to Actuator Control Messages

This task is highly actuator dependant and complex. There are many actuators suitable for use in a programmable array manipulator, and each requires a special control strategy. There has only been one type of actuator used in this work so far and the relationships between element control and object behaviour have been well established.

The element layer receives a message from the object control layer instructing it on the number of the actuator and the type of motion. The message, hence can have one of the following forms:
1. *make the elements bounded by* \((x_1, y_1)\) *and* \((x_2, y_2)\) *move an object in the direction* \(X\).

2. *make the elements bounded by* \((x_1, y_1)\) *and* \((x_2, y_2)\) *rotate an object counterclockwise.*

3. *make the elements bounded by* \((x_1, y_1)\) *and* \((x_2, y_2)\) *stop.*

In all cases, the two coordinates specified refer to the top left hand corner and the bottom right hand corner respectively.

In order to execute message 1, the element layer has a look up table that indicates the appropriate phase to apply in order to get the desired direction of motion. The movement of the object is open loop at this layer. That is, the element layer is a dumb layer in that it can only do what is requested of it. In order to make an object traverse the PAM, only the elements under the object are activated. Thus as the object moves, the elements being used will change. It is the responsibility of the next layer to track the object and issue the correct boundary and direction.

Rotating an object is more complex. Figure 4-5 shows two methods of rotating an object using PAM. The arrows in this figure represent the direction of the force being experienced by the object as a result of the action of the actuators of PAM.
Both techniques will rotate an object in the clockwise direction. It is important to note that there is a minimum number of elements under an object required to successfully rotate it. From the experimental work performed to date, this number is in the range 12 to 16. The reasoning that it takes a certain number of actuators to rotate an object can be justified in several ways. Firstly, if there was only one element driving the object, then the rotation would rely on the object remaining stationary with respect to the top of the PAM at one point in order for a rotation to occur. This cannot be guaranteed. Secondly, the driving action of the solenoids does not occur simultaneously. That is, if one solenoid is being excited by phase X and another by phase Y, then they are reaching the peak of their amplitude at different times. In this way, a rotation is effectively a series of overlapping, straight line translations.

The more elements under the object, the more successful the rotation is likely to be. The problem with the technique on the left hand side of figure 4-5 is that it will tend to cause the object to translate as well as rotate. This situation is improved using the technique described by the right hand side of figure 4-5. The best method is one which has two sets of forces acting on the object; one set causing the object to rotate and a second keeping the object in the same position. This is the method used for the trial PAM. There could be occasions when it would be desirable to be able to translate and rotate the object simultaneously. This case has not been considered.

The methodologies adopted for rotating an object on PAM were derived from experimental work. A dynamic model of a multi-element PAM is in the process of being developed so that the behaviour of an object on a PAM can be better understood. This is preferable to simply using trial and error through experimentation.

In order for the control system to communicate to the actuator hardware, a high speed communication was designed. This introduced an interesting problem to the transputer control hardware since the interface is not intelligent. That is, the communication need
only be from the control system hardware to the actuator hardware. Section 4.4.1.2 describes the problem and the method used to solve it.

4.4.1.2 Communication with Serial Hardware

Both Occam and Parallel C languages are designed to deal with communicating sequential processes [A3] when two tasks, or processes exchange data. Thus both languages are not designed to deal with this special case of a process communicating with an external piece of hardware. This is the fault of the compilers used and not the language itself.

The hardware associated with the PAM control system is based on a collection of TRAnspeuter ModuleS (TRAMS) located on a motherboard. This motherboard is installed in a PC and a special software package called a "server" is executed to open a gateway between the PC’s operating system and the transputer network. The server is responsible for downloading the code to the correct TRAM and then "booting" the TRAM. That is, configuring the transputer to start executing the code. When these TRAMS are mounted on the motherboard each communication link from the TRAM needs to be configured.

Each TRAM (in this case) has four INMOS links. The motherboard can accommodate up to 10 TRAMS. The motherboard is constructed such that communication link 2 of TRAM 'n' is hardwired to communication link 1 of TRAM 'n+1'. The remaining 3 links are then connected to an IMSC004 Crossbar Switch. This device allows any one of its 32 inputs be connected to each other. The device is analogous to a telephone exchange. Thus, one could connect TRAM 3, communication link 3 to TRAM 7, communication link 0 via the crossbar switch. In the case of the element layer, it is required that element layer link output to the edge connector of the motherboard so that the communication to the outside world can be established.
This is a minor task since there is a configuration language available for the motherboard that allows the crossbar switch to be configured as one desires. The problem, however, arises when one tries to link a process to a hardware device with no intelligence. The software does not allow this to occur. Deadlock can occur if a process tries to communicate with a process that does not exist. As far as the Occam and Parallel C packages used in this project are concerned, the element control hardware is a dumb process and thus cannot communicate with a task running on a transputer.

In order to allow such a link to occur, it is necessary to trick the software into believing that the element control layer is communicating with a legal task. This can be achieved by taking advantage of the three levels of software associated with task allocation in a transputer environment. These three levels could be described by:

1. task allocation onto processors,
2. configuration of inter-process communication via virtual channels, and
3. configuration of the physical channels.

Since it is possible for many tasks to execute on a single transputer, they must communicate with each other via virtual channels. That is, communication links within the transputer hardware and software. If the tasks are spread amongst several processors, then the communication channels are physical links. On the other hand, if two processes are mapped onto one processor and the communication link is assigned to a physical link, it is possible to physically tap into this communication link.

This approach is adopted for the element layer communicating with the element control software. The element control layer communicates to a dummy task, that uses minimal memory but does nothing. The link between the two tasks is then mapped into a physical channel. This channel is then configured by another software package (the link configuration software) to be linked to the edge connector of the motherboard. In doing so the one-way communication of the element control layer can be diverted to the
element control hardware. Since PAM is not sending any information back, dead lock will not occur. A time out will occur if the correct communication hand shake does not take place.

4.4.2 Object Control Layer

The object control layer (object layer) issues commands to the element layer in order to position and orient an object under its control. Figure 4-6 shows how this layer fits into the control scheme. Interaction with the command layer is also required for the higher level planning task. The object layer receives information from the two layers above it; the vision and command layer. This effectively places these two higher tasks at the same level. However, the distinction into two layers is made for reasons to be justified later. The interaction with the vision layer is intense since it is the vision which provides the necessary feedback for the object control to behave correctly.

![Control Layer Structure - Object Control Layer](image)

One of the other task requirements of this layer is the tracking process. This process is associated with knowing where an object is at any one time and how many of these objects exist. In order to fully understand the operation of the object layer, one needs to understand the commands it is receiving from the upper levels. It is not necessary to know (at this stage) how these commands are derived, but simply to know what form they take.
Chapter 4  PAM Control Scheme

The command layer will send a message to the object layer of the form "move an object from \((x_1, y_1)\) to \((x_2, y_2)\)". These coordinates represent the location of the centroid of the object. In order for the command layer to issue such a control signal, it too must know the position and orientation of all the objects currently being controlled. Hence there is a strong interaction between the three middle layers.

4.4.2.1 Object Control Layer - Tracking Task

The task of tracking the objects in the view of the vision system is a complex one. At present, the code can only deal with two objects. This is not a limitation of this layer, instead it is a shortcoming of the vision system currently available to the project. In order to perform this task, the layer has to execute several assignments.

1. Receive information about the position and orientation of the objects and relevant object numbers from the vision layer.

2. Identify any new objects in this list. This is achieved by detecting an object in a previously unknown position. The list of known objects is updated to include the latest position and orientation information.
3. If a new object is identified from 2, then it must be added to the object list currently being used. A unique object number is allocated to this item (derived from the current task time) and this number is passed on to the vision layer. When the vision layer sends new information at a later time, it will be transmitted along with the identification number. This makes the task of sorting the objects less intensive.

4. An updated stock list is sent to the command layer. The command layer needs this information so that it can interact with the surrounding manipulation modules.

This layer exists on the "basic control" TRAM of the control module. The ability of this layer to reason whether an object is new or old is an important function. As an object traverses the PAM, or even the manipulation module, it will appear in the field of view of the vision system and some time later it will disappear from view. It is this feature that can be exploited in order to identify the item in question. Removal of an object from PAM does not produce any confusion.

4.4.2.2 Object Control Layer - Object Control Task

The other major task of the object control layer is the object control. This task is considerably more complex than the tracking task. The object control must also receive information about the position and orientation of the objects being manipulated so that it can "close" the control loop.

It is not possible to compare the position and orientation control problem of a programmable array manipulator with that of a robot axis. There is virtually no explicit control over the velocity and acceleration of the object on a PAM unlike that of a robot arm. It is possible to control the velocity of an object by limiting the number of active elements under the object or by modulating the on and off times of the actuators in a
PAM Control Scheme

Chapter 4

PWM style. This is a difficult task to perform since at present a model is not available. The control scheme at this level is therefore currently very simple.

Similarly, the control scheme is different from conventional model-based control systems. The control signal can only direct the element layer to drive an object in a particular direction. It has no knowledge or feedback of the forces applied to the object by the actuators. The control strategy is a rule based scheme that can learn how the manipulation of an object can be controlled. A model based system could improve the control scheme, however, object stall for example, or irregular object behaviour will always be difficult to deal with. This is true for all schemes with feedback.

As mentioned before, the basic control command signals for the object layer are received from the command layer. These commands arrive in the form of:

"pass an object to the northern neighbour with a particular orientation".

In order to issue such a command, several exchanges of messages between higher layers will take place. As shown in figure 4-6, the command layer knows all the necessary information about all the objects currently being manipulated by the PAM from the object layer. It can therefore generate the necessary command to be issued to the object layer.

The control strategy devised currently is very basic. If the control command asks that an object move from point A to point B, the control will assume a motion along a straight line. The translation and rotation are performed separately and sequentially, translation being first. The control algorithm is a series of actions, rather than a continuous time varying operation. The activities of the object control layer are:

1. Exchange command messages with the command layer.
2. Separate the translation and rotation component from each other.
3. Calculate position error, \( p_{\text{error}} = \frac{p_{\text{ref}} - p_t}{b} \),

where:
\[ p_{error} = \text{position error}, \]
\[ p_{ref} = \text{position reference}, \]
\[ p_{fb} = \text{position feedback}. \]

4. If the position error is still large, then the translation is not complete. Issue a command to the element layer accordingly.

5. If the position error is acceptable, issue a command to the element layer to stop the object.

6. Calculate the orientation error, \( o_{error} = o_{ref} - o_{fb} \),

where:
\[ o_{error} = \text{orientation error}, \]
\[ o_{ref} = \text{orientation reference}, \]
\[ o_{fb} = \text{orientation feedback}. \]

7. If the orientation error is still large, then the orientation is not complete. Issue a command to the element layer accordingly.

8. If the orientation error is acceptable, issue a command to the element layer to stop the object.

4.4.3 Vision Layer

The vision scheme developed so far for the system in conjunction with the control strategy is quite simple. It can only deal with two objects in the field of view. If more than two objects are in the view of the camera, then only information (in the first instance) on the first two objects found (the two closest to the top of the frame) will be calculated. The position of the vision layer within the control hierarchy is in the middle, as illustrated in figure 4-8. This layer exchanges information with layers both above and below.
The tasks of the vision layer consist of determining the position and orientation of an item in its field of view and a partial tracking of these objects. This position and orientation is determined with respect to a known point in the PAM frame. The partial tracking task relates to the ability of this layer to return the position and orientation information with an object identification tag attached. This ID is uniquely established by the Object Control Layer and communicated to the vision layer. It is then the responsibility of the vision layer to maintain the correct object tag. These tasks are non ideal and relate only to the small vision system developed for the prototype PAM. A full implementation of PAM would require a vision system with a superior processing capability. Chapter 6 describes the mechanics of the vision layer in greater detail. This chapter is only concerned with the task level attributes.

4.4.3.1 Position and Orientation Task

This task is virtually self explanatory. It must be capable of accurately determining the position and orientation of the objects in the field of view. The task sequence is as follows:

1. Capture an image into memory.

2. When the image capture is complete, then:
   - if this is the first frame to be processed, then establish the position and orientation of the objects in the field of view (up to a maximum of two).
- if this is a subsequent frame, use a window of the captured image to determine the new position and orientation of a previously established object. This technique involves processing a portion of the captured frame that one would expect the established objects to be located within. It assumes a limited translational movement from one frame to another and could be prone to errors should this movement exceed the prediction. This is however unlikely to occur. If it does occur, the logic can sense a change in the objects’ image characteristics and can adjust the window accordingly.

- determine if any new objects have entered the frame (if only one object is currently being "tracked").

3. Attach the position and orientation information to an object identity, if it exists. If it is a new object, pass on only the relevant data and expect a reply from the control layer for the object identity.

4. If an object identity does not arrive from the object layer, then continue regardless since the control layer will deal with the problem.

The mechanics of the position and orientation calculation are based on the concept of moment invariants [D3] and boundary tracing. These concepts are well established. The boundary tracing algorithm was developed from first principles for the PAM project.

4.4.4 Command Layer

This layer is arguably the most important layer within the control architecture. It is responsible for determining the tasks that the PAM module must perform. In this instance, PAM does not necessarily refer to a complete PAM, but rather the Control Module & Manipulation Module pair. The multiple occurrence of these modules create a PAM. The command layer communicates its requirements to the surrounding control modules via the communication layer. Figure 4-9 shows the arrangement of the command layer within the control system structure. A more complete description of the tasks and theory behind the Command Layer is found in chapter 5.

It is possible for the command layer to survive without the communication layer. However, this higher strata provides an interface should a multiplexed communication system becomes necessary. It also enhances the modularity of the control scheme.
Chapter 4  

PAM Control Scheme

The command layer consists of several tasks. Since this layer is the interface of the manipulation module to the rest of the PAM, it determines the task of its associated MM. For instance, it is this layer that issues the command to the object control layer to move an object from one point to another with a particular orientation. The coherence of various components of PAM, mechanically, electrically and in control, as a single device is essential for its operation. Without this coherent structure, a PAM cannot perform the overall function expected from it by the other processes interfaced to PAM in a production line.

Consider PAM as a part feeder in a system. In this case the command layer is responsible for the coordination of its MM's to perform the part feeding task. There are three basic movements expected from PAM:

1. move from one location to another with no reorientation, or

2. reorient an object without translation, or

3. translate from one location to another whilst reorienting.

The command layer takes enquiries from surrounding MM's and translates them into task for each MM. For example, an MM physically residing to the left of a particular module might send a message asking for an item with a given orientation. If this MM has such an item and it can supply it, then the command layer will respond accordingly.
and then proceed with the mechanisms required to complete this task. The command layer, hence, consists of tasks that:

1. allow the manipulation module to interact with surrounding modules,
2. translate global task requests into local jobs,
3. translate local tasks into global requests, and
4. interact with surrounding processes.

The last task (task 4) interfaces the MM to the incoming and outgoing processes which are usually found in a manufacturing environment.

The algorithms used in this layer are based on the demand-pull strategy often encountered in manufacturing systems. These algorithms and a computer simulation of the behaviour of the command layer interacting with other modules are presented in chapter 5.

4.4.5 Communication Layer

The communication layer is the gateway for the command layer into the surrounding processes. Figure 4-10 shows the communication layer at the highest level of the control system. In many cases, it is not necessary for this layer to exist. For example, when the command layer of one MM is attempting to communicate with the same layer of another MM, direct communication via the transputer asynchronous communication link can take place. However, if the command layer needs to interface to some other medium, then this layer can perform the necessary translation and control tasks.
The code and mechanics for this layer have not been finalised and so this section is only really a conceptual discussion of how this layer should operate. In the proposed control module arrangement, there are four possible outgoing communication links distributed around several processors. As a result, the command layer cannot always directly communicate to a surrounding control module. In this case, a communication layer can perform the task of a message router. Since it knows the layout of the communication network, its task can be distributed amongst several processors and it can provide the necessary single channel interface to the command layer.

4.5 Summary

During the course of this chapter, the concepts of various control schemes applicable to PAM have been presented. The implication of the possible schemes is that in order to optimise the modularity of PAM, there is a limited choice of topologies. The scheme chosen as the candidate is best described as a hierarchical distributed control structure. The concept of a centralised control architecture, whilst suitable for small PAM’s, does not lend itself to expansion easily. There are many issues relating to the control of PAM that have not been discussed in this chapter. These issues, including collision avoidance, parallel manipulation of multiple objects and so on, are all tasks which can be incorporated easily into the chosen control architecture.
The use of the transputer as the control processor has been extensively justified by this chapter. Without this device, the modular approach would not be as easy to achieve. Control schemes for PAM should be as modular as possible. It makes the task of reorganisation much simpler should a change be required. As shown in this chapter, a centralised control architecture designed for PAM is essentially ineffective when the physical PAM configuration changes.

As shown in figure 4-11, the project has presented material up to and including part of the work associated with the control scheme. The rest of the control scheme philosophy is presented in chapter 5. The key to the cooperative performance of the MM's lies in the software of the Command Layer. The way in which this has been implemented for PAM is the subject of discussion in the next chapter.
CHAPTER 5

PAM CONTROL SCHEME - COMMAND LAYER

5.1 Introduction

The command layer is the second highest level of the proposed control scheme for the Programmable Array Manipulator. It was briefly summarised in chapter 4 and was described as the most important layer in the control scheme topology. Figure 5-1 describes the Command Layer as the fourth layer of the control architecture.

![Figure 5-1 Control Layer Structure - Command Layer](Figure 5-1)

The reason for the importance of the command layer lies, of course, in the duties it is to perform. This layer is the interface of the manipulation module it is responsible for to the rest of the world. The rest of the world for a MM is the other neighbouring MM's. It allows the individual MM's to interact in such a way that the PAM appears to be a single, uniform device. The modularity of PAM has been emphasised throughout the
course of this thesis; thus it would not be desirable for each MM to act as a single, isolated item.

The principle behind the command layers ability to act cooperatively with other control modules is that of demand-pull. Such a concept is found in manufacturing system automation. During the course of this chapter, the concept of demand-pull will be described, as will the connection of this concept to the PAM control philosophy. In order to determine the suitability of this scheme to PAM, a simulation of a 25 MM PAM has been performed. The results of this simulation are also presented. Finally, a summary will be given.

5.2 Philosophy Behind the Command Layer

In chapter 2, the notions of demand-pull, kanban and just-in-time control were introduced. These ideas all complement each other when it comes to implementing them in a control system. Demand-pull is a mechanism by which a process can obtain all the material it requires.

It was decided earlier that a PAM is no more than a device that can reposition and reorient objects based on some criteria. It has also been established that the individual command layers of the MM's that make up a PAM need to cooperate with each other in order to work effectively. Demand-pull is the mechanism which allows this to occur.

5.2.1 Demand-Pull Mechanism

Demand pull is a method for a downstream process to get objects it requires from an upstream process. In a computer controlled environment, demand-pull is nothing more than a particular software strategy. In a manufacturing environment, demand-pull methodologies may result from the implementation of other control techniques, such as kanban.
To understand what demand-pull means, a control structure must be introduced that allows the effect to occur. The control structure devised allows the material to be pulled regardless of whether it is required immediately, or at a later time. It is assumed that the process being carried out by PAM is repetitive and continuous. Thus it will be performing the same task over and over again. The control structure is based on the role that an MM is capable of performing:

1. A *customer* manipulation module is an MM which requires an object. It wants this object in order to fulfil a task it is performing. This task could be initiated by a process external to PAM, or it could be as a result of another MM.

2. A *supplier* manipulation module is an MM which attempts to satisfy the demands of the customer MM.

Each type of MM can be further classified depending upon the location of the module with respect to the PAM overall task. These classifications are also based on the role of the MM:

1. A *target customer* is a manipulation module which terminates the PAM process. It is the MM that the objects being manipulated will finally arrive at prior to their removal and further processing on other machines.

2. A *primary supplier* is the first MM on the PAM to receive an object. Such an MM would have control over the process that supplies these objects. The supply mechanism in this case could be a conveyor, or a vibratory feeder.

In a large PAM, the role of supplier and customer is interchangeable. In some instances a MM could be a customer, and at other times it could be a supplier. These dual roles are important, since it is the identity of the MM which determines what controls the pull mechanism. Each customer MM provides a control signal to the surrounding supplier MM's. Note that these surrounding supplier MM's are only classified as suppliers with respect to the customer MM. There is nothing stopping them from being classified as
customers for their own purposes. This control signal is the desire to obtain a particular object at a given orientation. Such a desire is a result of the overall task that the PAM is performing. Such a task is delivered to PAM via the target customer(s).

In order to illustrate the pull mechanism further, consider the interchange between a customer MM and a supplier MM. Let the two MM's be defined as MM\textsubscript{a} and MM\textsubscript{b}, such that MM\textsubscript{a} is the supplier and MM\textsubscript{b} the customer. Each MM must keep a local record, or buffer, of the objects it has available and the items that it demands. As objects move onto a particular MM, they will appear in that MM's input buffer. As the objects move off the MM, the objects disappear from this buffer. A supplier PAM will also keep a record of requested objects. When a customer requests an object from a supplier, that request is stored in the supplier MM's order buffer. As the supplier provides this component to any customer, it disappears from this buffer. Observe that the supplier can supply the object to a different MM, rather than the one that placed the order initially.

The control pulse is provided by the customer MM. This control pulse takes the form of a message from one command layer to another. This message could be written as:

"please supply object X at orientation \( \theta \)."

In this example, such a message would originate from MM\textsubscript{b}. If MM\textsubscript{a} can satisfy the demand of MM\textsubscript{b} it does so. It would thus respond with an affirmative or negative reply. If the reply is negative and MM\textsubscript{b} cannot receive the item from anywhere else, then it must place an order for the object with the neighbouring supplier MM's. In the example there is only one MM that is able to provide objects to MM\textsubscript{b}, however, in a real world example, MM\textsubscript{b} could obtain items from up to four suppliers.

The order is then placed by MM\textsubscript{b} with MM\textsubscript{a} for an object. As far as the control scheme of MM\textsubscript{a} is concerned, it has no memory of which MM wants this item. There is no memory of the owner of an order for a particular item. All that MM\textsubscript{a} knows is that it
requires an object, X. The net effect of this forgetful supply scheme is that the entire PAM will eventually have multiple objects being moved around towards the target customer. This ensures that the target MM will be able to obtain an item within a reasonable time.

Now that MM₁ has an order to fulfil, it is no longer a supplier for the object it seeks. Since it is MM₁ that wants the object in question, it must now behave as a customer in order to obtain it. The process is now repeated with MM₁ acting as the source of the pull control pulse. Eventually, the pull control pulse will be satisfied when the object requested can be supplied without delay. The item can then back-track to the target customer.

5.2.2 Scalable Demand Pull Mechanism

The above philosophy need not apply to the command layer mechanisms. It also has the ability to be used by the Object Control Layer. If each manipulation module was devised from multiple manipulation sub modules, then application of this pull mechanism would be simplified. This is an appealing feature since it could reduce the code development time.

There is, however, another result of this strategy that is more appealing than simply reducing the code development time. This is the simplicity of the concept. Since there are only simple algorithms used to determine which object should go where on the PAM, or on the MM itself, there are no tracking decisions to be made. The MM essentially looks after itself without having to make difficult and complex decisions. When an object is demanded, it is supplied if it is possible otherwise it will be ordered just in case it will be asked for again. In a highly repetitive assembly or material handling arrangement the likelihood of a item being requested more than once is very high.
An additional advantage in adopting this simplified control approach is found upon investigating the nature of the messages being exchanged between adjacent MM's. Could the messages be replaced by binary logic? This would simplify the hardware structure by replacing the communication hardware with a standard logic interface. The use of shared memory between two adjacent MM's would simplify the interface even further. It would mean that the software would not have to deal with communication with its neighbours at all, since it would have all the information available to it in the shared memory. The communication mechanism therefore becomes nothing more than reading and writing to shared memory.

In the case of transputer hardware, this is of little consequence since the communication is essentially a DMA into the neighbouring control hardware. However, since the communication layer is distributed amongst the processors in a control module, shared memory or digital interfaces could replace several multiplexing tasks, thus simplifying the logic further.

5.3 Simulation of Command Layer.

In order to determine the behaviour of the PAM using the above control philosophy, a simulation was performed. There are a variety of behavioural properties that might be found if a complete simulation was written. There are, however, some aspects of the PAM control that cannot be simulated. Such aspects do not relate entirely to its control, but rather to the way the objects on a PAM behave. Collisions, jamming and so on are the sort of problems that a simulation cannot deal with.

The focus of the simulation is on the way that different manipulation modules interact whilst executing the demand-pull strategy. It does not attempt to determine the calculation or communication overhead of such a scheme.

The simulation was initially to be performed on a network of transputers. Instead, for portability and graphical reasons, the simulation was switched to a personal computer.
It is not necessary to separate the tasks into individual concurrent processes, since real
time is not an issue. Thus it is quite reasonable to perform this simulation in a PC
environment, using symbols (program elements) as process parameters. The code is
provided in Appendix E.

5.3.1 Simulation Scenario

The simulation is based on a five by five square arrangement of manipulation modules.
The MM's on the left most side of the PAM are primary suppliers of coloured disks.
These disks are sized such that only five disks may be located on a MM at any one time.
The supply of the disks onto these primary supply MM's can be applied in two ways:

1. assume continuous flow of product, or

2. assume a finite number of available disks.

In fact, both instances were looked at.

The MM located in the right "lower" corner is the target customer. The aim of the
exercise is to satisfy the demands of the target customer. It is programmed to array five
disks, one of each colour into a predefined pattern. The arraying of the disks is not part
of the simulation. This scenario is depicted in figure 5-2.

The simulation starts with the target customer requesting an object from its neighbours.
The one limitation imposed on the control is that objects can only be supplied to a MM
immediately from the left, below or above. An object cannot traverse a diagonal path
when being passed from one MM to another. This is not a major deviation from
impositions that may be set in place in the real world. An object crossing a diagonal
boundary would require the coordination of the elements of four manipulation modules.
The object would also find itself in a position where it is not really "owned" by one
unique MM.
Thus the tracking task may also be complicated. Such an exercise in coordination could prove difficult, unlike the crossing of a vertical or horizontal boundary.

The problems associated with this situation are beyond the scope of this simulation, since they could be regarded as an implementation issue and not a philosophical one. Since the demand-pull scheme is a new concept for PAM (which is itself also a new concept) it is this scheme that is to be looked at.

The types of issues under investigation in this simulation are:

1. The nature of the message passing between MM's. This requires trying to understand what message content is needed in order to execute the demand-pull strategy. This would also establish the viability of using a digital interface, rather than a message passing one.

2. Would it be possible to create a "traffic jam" of objects on the PAM? If the demand-pull mechanisms try to route all of the items through a single MM, then the other units would not be used efficiently and one could reason that either the
size of the array was incorrect, or the control scheme is not performing satisfactorily.

3. Would it be possible to have objects wander around aimlessly? Since there is no overall planning and control, it might happen that an object or group of objects may find themselves wandering around the PAM without ever reaching the target customer.

4. Since the target customer is the master of the PAM, the way that the demands of this MM are propagated throughout the MM network need to be established. Are there rules that need to be applied to the ordering mechanisms, such as priority? Would it be needed to prevent objects from being ordered at high quantities? Should the message passing be bi-directional between customer and supplier at all times?

5. In this simulated example, there is a general flow of product from the left hand side of the PAM to the right hand side (refer to figure 5-2). Should product be allowed to move from right to left? Additionally, should objects be allowed to backtrack over MM's that an object has already been through?

6. Will the transfer of product from the primary suppliers to the target customer be inefficient? Since there is no overall planning strategy, could it possibly mean that an object takes too long to get from the supply MM to the target MM. Judgement of such a condition could be heuristic unless a minimum cost function is established. A cost function is based on the minimum distance the object needs to travel to get from the supply side to the target MM. The function is evaluated using the straight line motion restriction.
7. In a real world application, it is possible that a complete MM could stop working due to a hardware failure. Alternatively, some elements may stop working correctly and the module could shut itself down until it is repaired (a future enhancement perhaps). If this was to occur, in what way would the demand-pull mechanism be affected? Would there be an effect on the efficiency of the strategy? The effect of MM breakdown could be simulated for the multiple failure situation. Is it possible to fail a certain number of MM's, or a certain pattern of MM's that could lead to a situation where the demand-pull strategy fails?

8. In some cases, two MMs could supply the same object. This situation should be prevented and mechanisms established to deal with this situation.

The results of the simulation with respect to these and other issues are presented later in this chapter. It is important to learn the behaviour of the demand-pull control scheme since it would be an expensive task to implement it in a real system.
Chapter 5  PAM Command Layer

5.3.2 Simulation Algorithms

An important principle for the demand-pull algorithm is the way in which the demand signal is distributed through the network. There are two techniques that can be used for the propagation of this control signal. The first method uses a one-time pulse and the second a cyclical scheme.

5.3.2.1 One-time Demand Pulse

The use of this technique involves the target customer issuing a single order for the items it requires. In such a scheme, the neighbouring manipulation modules then interrogate the customer MM (in this case it would be the target customer) to determine if the object they hold in stock is required.

5.3.2.2 Cyclical Demand Pulse

The cyclical demand pulse is, as the name implies, based on a periodic request for items by the target customer to fulfil its demands. Such requests are issued each time the complete message has been propagated throughout the PAM network. The target customer knows when this has occurred by the number of counts passed on the global synchronisation bus.

When the target customer requests an object from its neighbour, the neighbour then has to decide whether or not it can supply the request. If the item is available, then the response is obvious. However, if the item is not available, then the target customer will place an order with the supplier MM. When the next synchronisation pulse occurs, these MM's will then attempt to obtain the items ordered from their neighbours. Thus the demand pulse will propagate through the array. Once the correct number of synchronisation pulses has occurred, the target can issue another demand pulse. The correct number of synchronisation pulses is simply the maximum number of MM's in the physical path between the target and the primary supplier. In the simulation example this number is 8.
Recall that the nature of the demand-pull mechanism is that when a supplier receives a request for an item, it will then always want that item regardless of subsequent requests. It is not until the item is received by this MM that the demand is erased. The reasoning behind the cyclical demand strategy is a result of the forgetful nature of the demand-pull strategy. If the flow of pulses stop, then no supplier MM will know who to give the items to.

This cyclical demand pulse is the basis of the simulation of the command layer. In the simulation, the message is propagated via an interrogation of a large array of information about the stock and the status of each of the manipulation modules. Once the control pulse has been distributed through the network, the responses or actions to this pulse occur. Such an action involves the moving of an object from one MM to another. Recall that it is only the command layer that is being investigated and not the element control layer. In real life, the effect of the element layer would first need to be evaluated. This could also be done by simulation if the control strategy were known.

5.3.3 Result of Command Layer Simulation

Prior to a discussion of the results, a presentation of the actual simulation is given. In figure 5-4, a scenario consisting of five pairs of objects on the left hand side of the PAM are to be arranged at the target customer. The arrangement requires one of each item to be placed at the correct position on the target MM. Each frame demonstrates the movement of the objects after each demand pulse. It is assumed that as soon as the target has all 5 disks, they are removed and a new batch is required as the process continues.
5.3.3.1 Message Passing Requirements

Earlier, it was proposed that the message passing could be replaced by straightforward digital logic hardware. This would appear to be unlikely, since the simulation showed an exchange of message involving data is required. The idea of shared memory between MMs would be more appropriate.
Since the message passing algorithm requires the demand pulse to be propagated throughout the entire PAM for each cycle, then a simple digital interface will not be sufficient. Although it is possible to implement it, an additional level is still required for this message mechanism. The message itself is simple and requires no more than an inquiry for the availability of a stocked item. Prior to the exchange of messages, the software needs to know some information about the MM it is about to communicate with. Such information can be held in a configuration data structure for each MM. This structure contains the following boolean information:

1. the presence of a neighbour,
2. the status of each neighbour - either functional or non-functional,
3. indication of primary supplier neighbour,
4. indication of target customer neighbour

Such a list means that the MM can make sensible decisions about the messages that should be issued. For example, it would be unnecessary for a MM to attempt to communicate with a northern neighbour if it did not exist. Such a configuration can be written at compile time, or it could be dynamically configured. The second process may be performed by an automated message exchange process at the start of the PAM cycle. The early configuration by the programmer based on the current physical arrangement seems more appropriate. The only operation that needs to be dynamic is the functional indication.

5.3.3.2 "Traffic Jam" Occurrence

In the simulations performed no such phenomena appeared. However, there is a good reason for this. Each array is allowed to keep a certain number of items. This is a reasonable restriction, since in real life it would also be wise in order to limit the number of items that can be physically located on a manipulation module. Since each
MM has a limited physical dimension, it is not possible to control an excessive number of items without severe vision related problems. Problems of occlusion, items colliding with each other and so on could occur. In the results given above, each MM sees an object passing through it. Hence there does not seem to be any under-use of the MM's.

The result is somewhat different, however, when several MMs are placed out of service. The outcome is not that of a traffic jam but rather an increase in the number of movements required to translate an object from the starting MM to the target customer. The result is quantified in section 5.3.3.6.

5.3.3.3 Loss of Direction Occurrence

The situation of aimless wandering arises when an object moves from one MM to another without ever reaching the target customer. The wandering may eventually lead the item to arrive at the target, but the efficiency of the path is very low. The very nature of the demand-pull mechanism is such that an object will always migrate towards the source of the demand-pull pulse. Deviation by one or two MMs on its way to the target customer can be observed. This does not constitute an aimless wandering.

This was the expected result. As stated already, there is no reason why the object should move in an unpredictable manner since the methodology used for the propagation of the demand pulse throughout the network of MM's is regular and highly repeatable.

5.3.3.4 Demand Pulse Propagation

The mechanism for providing the control pulse in a simulated environment is certainly less complex than that required for a real world PAM. The simulation is closer to that of a shared memory communication method rather than a message passing technique. It is worth noting that the simulation code was written in such a manner, that the message passing is duplicated by argument passing from one function call to the other.
This leads to a series of sequential processes rather than concurrent ones. In a real PAM, would there be any concurrency at this level of the control architecture? The answer is essentially no since the demand pulse must pass from one MM to the other and return before the source MM can make a decision about what to do next. The same applies for MM's downstream from the target customer that are attempting to satisfy its demands. For this reason, although the actual method used to simulate the control pulse propagation is different from the real implementation, the outcome is essentially the same.

There is very little complexity associated with the message passing rules. The only message that is passed is the request for a product followed by an affirmative or negative response. If the response is negative, the requesting MM will place an order for that item with the downstream MM. If it has multiple neighbours, each neighbour will receive an order. In order to effectively implement this demand-pull, each MM needs to keep a list of different items. This list includes

1. a list of ordered items,
2. a list of stock items, and
3. a list of orders.

Item 1 is required to prevent excessive ordering of the same item. Item 3 is different from 1 in that it reflects the number of MM's that have placed an order for a particular item. Item 2 is required in order to know if the MM can satisfy the demand of its customer.

5.3.3.5 Right to Left Motion

The simulation presented is not really the correct platform to test the notion of right to left motion of an object. In this instance the target MM is located in the bottom right hand corner of an imaginary PAM. As a result, there is no reason for an object to move
right to left, unless a module ceases to be functional while the PAM is performing the task asked of it. If, for example, the target customer was located centrally and the primary suppliers distributed around the periphery, then it would be necessary to allow the object to move in any given direction.

Either way, breakdown or different physical configuration, there is no reason to disallow the right to left motion of an object. In reality, the issue that needs to be clarified is the number of back-tracking steps allowed to occur in the event of a module failure, given the various possible configurations. The simulation showed that a limit was not required on the back-tracking steps for the simulated PAM since there was no limit imposed and the results were satisfactory. Alternatively, a different task may also require a different approach to this issue, although this seems unlikely.

5.3.3.6 Cost Function

One of the most important performance measures for the control strategy is the efficiency by which the objects are conveyed from one region of the PAM to another. As stated earlier, there is a cost function which describes the minimum number of moves required by an object to traverse the PAM. In the simulation case, each object has a different minimum cost value since they start at different positions relative to the target array. The following table illustrates the various cost functions for the items in figure 5-2.

<table>
<thead>
<tr>
<th>ITEM</th>
<th>COST FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5-1 Cost Functions
By observation of figure 5-2, the above table is easily justified. For example, item 5 has to move 4 manipulation modules to the right in order to arrive at the target customer. The lowest cost value for the other objects are evaluated in the same way. The value of this parameter is a function of the task that the PAM is being asked to perform. If, for example, the target customer is located in the centre of the array then obviously the cost values would be different.

The inactive MM's are numbered in figure 5-2. The table does not present the resultant cost function for the case when no MM's are inactive, since the minimum cost function was met.

<table>
<thead>
<tr>
<th>Item</th>
<th>No. of Inactive MM's</th>
<th>Cost Function</th>
<th>Actual Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>First Object</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>1(MM 1)</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2(MM 1 &amp; 2)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3(MM 1, 2 &amp; 3)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5-2 Simulation Results - Multiple MM Failures

The results of the simulation indicate that the proposed demand-pull concept operates efficiently. Each object was translated from its primary supplier to the target customer in the minimum number of movements. In the case when a MM is out of service, the cost function changes. In such an instance, the demand-pull mechanism works well. Table 5-2 shows the results of various simulations with different numbers of MM's out
of service. Each is compared to the cost function given the same number of inactive MM's.

From Table 5-2, the performance of the demand-pull mechanism is reduced when there are a number of failures present. The worst case performance resulted in an object having to move to the target customer via two additional MM's. This is not a disastrous result; however there is room for improvement. There is some scope for a better outcome if the method by which the demand-pull control pulse is propagated through the network is varied. In all cases, the control signal propagation was not modified from the original case where there were no failed MM's.

5.3.3.7 Multiple Supplier Case

In this instance, there needs to be a handshake between the suppliers and the customers. When the customer MM requests an item from its neighbour, at that instance a positive or negative response will be received. Even though two (or more) neighbours could possibly supply the required object, it will be supplied by the MM that first responds to the request. The simulation could not deal with this scenario since it is a real time issue. Instead, if two or more neighbours had the required object, a random selection of the supplier was made. This worked effectively.

5.3.3.8 Alternative Scenarios

The results of the previous simulation showed that the control philosophy devised could produce desirable results. However, that was for one scenario. In this section, several different PAM configurations are considered. The first alternate scenario is a PAM that is 3 MM's high and 7 MM's long. This scenario is depicted in figure 5-5.
Figure 5-5 First Alternative Simulation Scenario

Rather than providing a frame by frame analysis as figures 5-4A & B did, the results of the simulation are provided by tabulating the actual cost functions of each item against the result of the simulation.

The failed MM's correspond to those numbered in figure 5-5.
### Table 5-3 Simulation 2 Results - Multiple MM Failures

The second alternative scenario is essentially the same as the original simulation, except that the target customer has been relocated. Figure 5-6 illustrates the scenario.

![Diagram](image-url)

**Figure 5-6 Second Alternative Simulation Scenario**
Once again, rather than providing a frame by frame analysis, the results of the simulation are provided by tabulating the actual cost functions of each item against the result of the simulation. The failed MM's correspond to those numbered in figure 5-6.

<table>
<thead>
<tr>
<th>Item</th>
<th>No. of Inactive MM's</th>
<th>Cost Function</th>
<th>First Object</th>
<th>Second Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1(MM 1)</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2(MM 1 &amp; 2)</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3(MM 1, 2 &amp; 3)</td>
<td>7</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>11</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-4 Simulation 3 Results - Multiple MM Failures

The performance of the control system under the various simulation scenarios was satisfactory. At most, there was a deviation by two MM's from the optimum cost function. This result implies that the designed control strategy works well for PAM.

5.4 Extrapolation of the Simulation into the Real World

Having established that the demand-pull mechanism works in an orderly and predictable manner, the real test would be the application of the philosophy into the real PAM application. Such a test would involve the fabrication of a PAM of similar dimension to that used in the simulation. However, it was stated earlier that it might be possible to use the same control ideas for the control of objects within the boundary of a manipulation module itself. Instead of regarding the MM as a single device, it could be
treated as a combination of actuation modules, AM's. The task of the MM is then to move an object from one AM to another given an initial and final position and orientation setpoint.

By choosing this method, the same concept of reduced planning overheads could be realised. Since the hardware was not available at the time of writing to test this result, it cannot be reported. However, based on the excellent results of the simulation, there is no obvious reason why this compact strategy would not work. Similarly, the use of the demand-pull strategy for the overall PAM coordination and control would appear to be a good choice.

5.5 Summary

This chapter has presented to the reader the concept of demand-pull as applied to PAM. This concept is used in many planning schemes, particularly those in manufacturing systems. The idea was extended and used in the coordination and control of a multi-manipulation module PAM. In order to prove the usefulness of this philosophy, several simulations were performed.

The results of these simulations provided insight into the sort of mechanisms required to make the demand-pull control work for a PAM. It assisted in determining the message content for exchanges between control layers of neighbouring MM's and the level of information that each individual layer needed to keep track of. The idea of shared memory for providing the communication channel was also proposed. However, since the hardware has been developed for the use of the transputer serial communication facilities, this would be the obvious method to use.

The overall coordination of the demand-pull control pulse is synchronised to a control pulse that is seen by all modules simultaneously. Since all modules know their position with respect to the rest of the PAM, they can easily reference themselves correctly.
The demand-pull concept removes the necessity of a central planning and coordinating process. The details of implementing such a control would create many problems. By adopting the demand-pull approach and distributing the tasks to the manipulation modules, the overall modularity of PAM has been greatly enhanced. Extending the control mechanism developed to that associated with the object control layer would produce a much simpler control, even at the lower level.

This concludes the presentation of the control system. In order to implement this scheme at a lower level, the object layer requires the correct position and orientation feedback. The vision system developed for the prototype PAM is the subject of the next chapter.
CHAPTER 6

PAM VISION SYSTEM

6.1 Introduction

The integration of a vision system with the other levels of control is an essential task for a programmable array manipulator. In the case of a PAM which uses actuators that do not drive the objects directly, the need for a reliable and fast vision system is very high. It is possible to envisage actuators that can drive an object under open loop control. In this instance, it would be likely that a dead reckoning philosophy would go a long way to providing a satisfactory level of performance. This is not the case with the vibratory element style of actuator as it would be extremely difficult to obtain such a performance level in open loop conditions. Thus a vision system would be able to provide the correct object position and orientation information.

This chapter provides the details for the vision system developed for the prototype PAM. A discussion of the hardware used by the system is provided as is the performance criteria that the system was designed to operate under. The techniques used for extracting the position and orientation information are also presented. It is important to keep in mind that one of the key issues with the vision system is the processing speed.
6.2 Vision System Background Research

The vision system to be devised for PAM is not entirely different from those developed previously for inspection systems. It does not, however, need a complete vision and inspection system as would be required for a commercial version of PAM. The prototype system should provide position and orientation feedback for the object being manipulated. In the first instance, this could be developed for the single object case, and then developed further for multiple objects.

In order to provide the necessary background to develop such a scheme, areas of investigation centred around the inspection and assembly topics. The investigation revealed many useful ideas. The papers best suited to the needs of PAM are [D3], [D4], [D5], [D6] and [D7].

The work reported by [D3] was published many years ago and established some of the basic ideas behind the extraction of data for image processing. The theory of two-dimensional moment invariants for planar geometric figures was the main thrust of the paper. A method for recognition of visual patterns and characters independent of position, size and orientation was presented. This is important for PAM, since the orientation of objects in the field of view are not known, prior to processing. The techniques discussed for calculating such moment invariants, are also capable of being applied in a manner that allows the system to calculate the centroid of the object under investigation. This is useful, as it provides a means for tracking the object on a PAM independent of the orientation.

In order to perform calculations on the image data for an object and establish certain characteristics, it is necessary first to determine where the boundary of the object lies in the image. Boundary tracing is one of the fundamental tools used in shape analysis. A boundary is the orderly (sequential) representation of the object boundary pixels. It is therefore necessary to trace the boundary of the object, so that the coordinates of the points in the boundary can be established. However, applying it to image data such that
it works quickly and reliably is another issue. The techniques discussed in [D5], [D6] and [D7] all provide a useful base for establishing a method for performing boundary tracing tasks.

The method described in [D7] lends itself well to parallel processing. This is achieved by splitting the image to be processed into several smaller regions, each of which is separately processed. This technique, although desirable, is difficult to implement quickly on a smaller network of processing elements. In the case of PAM, where there is a limited number of processors, the parallelism cannot be exploited to the extent that is becomes faster to implement a more simplified technique to a single processor.

The boundary tracing algorithm used for the PAM project is sufficiently different from other techniques to justify calling it original and is described in section 6.4.4. The method is simple, as most tracing algorithms are, and robust.

The reset of this chapter is dedicated to the development of the vision feedback system for the prototype PAM. The performance criteria established for the scheme are presented along with the hardware used and the software devised.

6.3 Vision System Requirements

The goal of the vision system task is to recognise objects of unknown shape and position. The idea of recognising objects does not involve actually identifying an object to be a chicken leg, or a floppy diskette, for example. Instead, the vision system should be able to determine a set of unique parameters for each object presented to it that can be used by the control system to perform closed loop position control.

Such a parameter list also leads to the question of how the control system going to use this information. Obviously if PAM is performing a palletisation of a set of chicken legs, it somehow needs to recognise the objects as chicken legs and not an unknown object. The same could be said of any other object that PAM has to deal with. These issues are well beyond the scope of this thesis and are not addressed here. It has been
assumed that the object the PAM is to deal with, and indeed the control scheme, is to be an object of known dimension and properties, such as a round disk, or a square block. In this way, the vision and control system principles can be proven without the need for a higher level of intelligence such as that required for a visual recognition system.

6.3.1 Performance Criteria

In order to realise the design parameters of the vision system, a set of performance criteria based on the results of the study of a trial PAM were defined. They include:

1. maximum rotation speed 25 rpm
2. maximum translation velocity 100 mm/s

Tolerances required to provide adequate performances for PAM were defined as:

1. Position measurement tolerance +/- 5 mm
2. Orientation measurement tolerance +/- 5 degrees

These values are parameters that will result in reasonable performance of the array manipulator. Based on the above rules, the performance expectation of the vision processing system would be in the order of 0.1 of a second per processing iteration. This time is obtained by considering the distance an object would move based on the maximum translational velocity and position accuracy requirement:

\[
\text{time} = \frac{\text{distance}}{\text{speed}},
\]

\[
\text{time} = \frac{10\text{mm}}{100\text{mm/s}}, \text{ or}
\]

\[
\text{time} = 0.1\text{s}
\]
This is a high expectation for an image processing arrangement. Obviously, if such a processing speed cannot be met, then either the positional accuracy criterion must be relaxed, or the speed of the traversal reduced. Alternatively, two or more position accuracy criteria could be defined; one whilst traversing the array and a different one whilst performing the final positioning at a reduced speed.

6.3.2 Vision System Hardware

In many ways, the hardware platform for the vision system to be used by the PAM was chosen when the transputer was selected as the main processing element. Thus, the hardware needs to connect with the other processing modules with a minimum of fuss. The image processing hardware should therefore be based on the transputer also.

The TRAM selected for the image processing function is the Transtech TTGF module. The frame grabber TRAM consists of a T805 transputer, a real time image digitiser and colour look up table (for driving a video output signal), 4 MByte of DRAM and 1 MByte of VRAM (video RAM). Associated with the TRAM are the usual link services, three video inputs and a single video output. The hardware associated with the image digitising functions are memory mapped into the transputers' address space and is therefore accessible to the software.

The hardware has a major shortcoming in that it is not possible for captured images to be directed to the video output. The task of writing image processing software is made easier if feedback of the image stored in memory is available. It is useful, for example, to be able to draw a line or even a box on the image and subsequently display this image. The TTGF hardware does not support this feature. The video output available from this TRAM is simply the digitised video after the digitising process but before the VRAM.
An important feature of the TTGF is the hardware associated with the image digitising process. The Bt251 image digitiser has a built in colour look up table (CLUT). This look up table allows the user to manipulate the grey scale values prior to the digitising process. This has many advantages, particularly when dealing with grey level images. In many image processing applications, the amount of information contained in a 256 grey level image means that the processing time can easily become excessive. Reducing the grey level image to a binary image, for example, is often used to reduce the information bandwidth. However, for a large amount of image data, this can be a time consuming task. The ability to do this task in hardware via a colour look up table removes this computational overhead.

The CLUT acts like a programmable bandpass filter, followed by an clamping stage. It uses the pixel grey level as an address input into a table of colour values. For an eight bit pixel value, there will be 256 entries in the table. Once the address has been determined, the CLUT will then look at the contents of the location addressed by the pixel value and output the value contained in this location. For example, if the grey level 133 is presented to the CLUT, the hardware will look in location 133 of its table. This location could, for example, contain the value 0. Thus the output of the CLUT for

![TTGF Block Diagram](image-url)
an input of 133 is 0. It is easy to see how it is possible to reduce a 256 grey level image to a binary image in real time using this hardware.

![CLUT Diagram](image)

**Figure 6-2 Colour Look Up Table (CLUT) Function**

6.3.3 Camera Resolution

When devising a vision system, it is important to also consider the camera resolution. This parameter will influence the accuracy of the measurement being performed. The image processing hardware samples the image data in object space into a finite number of spatial, two dimensional data points which are called pixels. Each pixel is assigned an address in the frame store, or VRAM, and a quantised value which can vary from 0 to 255. This quantised value is called the grey level. In the case of the TTGF hardware, this grey level can be manipulated via the look up table as described in the previous section.

To determine the accuracy requirement, one needs to consider the relationship between pixels and actual real world dimensions. For example, if a camera with a 500 by 500 pixel resolution is used to view an area 500 by 500 millimetres, then assuming the pixel is square, each pixel would be representative of a 1 by 1 millimetre area. This is only a simplified rule and it may not be representative of the smallest dimension the vision system can observe. The nature of the task, contrast associated with the detail to be detected, and positional repeatability are important factors that will also contribute to the size of the smallest observable detail.
In trying to relate the many different criteria to PAM, the task was complicated by the fact that the size of the PAM was not known when the vision system was put together. The actuator type and the positional accuracy of the actuator were also unknown factors. An estimate of an 800 by 600 millimetre PAM was made and a positional accuracy of 1 mm was the target. A single 756 by 581 square pixel camera was chosen for the task.

6.4 Image Processing

The vision system developed for the prototype is based around the transputer hardware described in the earlier sections. The task required of this vision system is to determine the position and orientation of an object in the field of view. This information is then passed on to the object control layer. Where applicable, this information will be attached to an object identity that originated from the object control layer. The rest of this section is devoted to the discussion of the algorithms associated with determining the position and orientation of an object.

6.4.1 Macroscopic View of the Problem

The task of the vision system has already been defined many times - that is to find the position and orientation of an arbitrarily placed object on the PAM surface. This data is then to be used by the other control layers. In order to fulfil this requirement, the processes required for fulfilling this task need to be defined. To this end, the image processing task is described in figure 6-3.

Each of these sub-tasks involves additional computations. The following sections will describe in more detail each of the jobs. The task of tracing the boundary of the object is the most involved. The image capture task is performed by manipulating the hardware appropriately. It is useful to understand this process of image capture, since the method of storing the data in memory is an integral part of the image processing task.
6.4.2 Image Capture Process

The task required for the capture of the image is described by the pseudo-code in figure 6-4. The three important items to note here are the memory assignment for the captured image and the event counter. Assigning memory for the captured image is important since the location of the data has to be known in order to do the processing. The user would have two options at this point; assign normal RAM as the area to put the image into, or use the video RAM (VRAM). If the user chooses the normal RAM to store the captured image, care must be taken not to overlap data with code in memory. The most obvious choice would be to assign an area of VRAM for the image to be stored in.

Assign source for video synchronisation signal
Assign video digitising look up table entries
Assign video output look up table entries
Assign voltage levels for black and white
Assign source for video input
Define video digitising granularity
While(task_is_required)
    Assign memory for image
    Commence image capture
    While(number of events is less than 39)
        do_nothing
    End
    Image capture complete
    Process image
End

Figure 6-4 Image Capture Pseudo Code

The storage of the image in VRAM is not necessarily contiguous. That is, if a 512 by 512 pixel image was stored into memory, each pixel would not be located into a contiguous block of memory. The VRAM is arranged as an area capable of storing up to 1024 by 1024 pixels of data. Figure 6-5 describes the memory arrangement.

Figure 6-5 VRAM Physical Arrangement

Thus when this smaller frame is stored in memory, only the pixels in a single row are contiguous.
The second point to note is the use of the event counter. The transputer has an external mechanism called an EVENT that allows hardware to interrupt the processor. It is not dealt with by the processor in the normal interrupt fashion, but rather it is handled as another serial communication channel. In this way, the parallel or concurrent nature of the program is not disturbed. The external image digitising hardware signals an event to the transputer hardware at every 16th line of capture. Since a composite video frame comprises 625 lines, then after 39 events (39 * 16 = 624) the full frame has been captured. This ensures the program execution time is minimised, in terms of the digitisation process.

The third and final issue relates to the entries for the look up table used in the digitising process. It was mentioned earlier that an image with 756 by 581 pixels, each representative of 256 grey levels is a tremendous amount of information. Consequently, an excessive amount of processing time is often required to deal with this amount of information. The inclusion of a colour look up table allows the amount of information presented to the image processing algorithms to be reduced by a significant amount. The look up table values would therefore need to be selected carefully in order to obtain the desired effect, without degrading the image quality.

The desired effect is to highlight the edges of the objects being manipulated relative to the PAM itself. In order to assist in this process, several other mechanisms are available and are also employed. These include:

1. contrast enhancement by using a black surface on the PAM,
2. improvement in contrast by using lighting, and
3. using the maximum camera aperture.

The combination of the above with the fine tuned use of the look up table entries provides the desired effect. The digitised image becomes essentially a binary image.
with consistent edge data. Variation in this data is acceptable, provided it is within a certain limit. That limit is dependent upon the positional accuracy achieved.

6.4.3 Image Scanning Process

The process of scanning the image to find the object is closely related to the boundary tracing problem. In order to speed up the execution time for the overall image processing task, the location of the object is recalled from frame to frame. When the position and orientation of the object is found for the first time, it is also enclosed by a bounding rectangle. That is, the object is windowed with respect to the rest of the frame. This process is illustrated in figure 6-6. The size of this window is determined from the maximum and minimum values of the pixels along the boundary of the object, plus some arbitrary value. In this way, the entire image need not be scanned whilst trying to process the data for a particular object. Only the previously calculated window need be processed.

The issue is complicated when multiple objects are in view and when new objects are entering the field of view. This object flow problem has not been addressed extensively for this project, since the system need only track a single object or several unoccluded single objects on a PAM and not consider the case of the object entering and leaving the manipulator.
6.4.4 Boundary Tracing Process

The process of tracing the boundary of an object is the essential task of the image processing. It allows the calculation of many parameters associated with the object being manipulated. In particular, these calculations allow the position and orientation control to operate in a closed loop mode.

The boundary tracing process is simplified because of a reduction in grey levels during the image capture process. From the boundary information, the features of position and orientation are to be determined. The theory of two dimensional moment invariants for planar geometric figures provides the mechanism for determining these features.

Another feature which improves the processing time for an image is the pixel resolution of the object being analysed. If an object has a perimeter length of say 300 pixels, it will take longer to trace the boundary compared to an object with say 25 pixels. The vision system can be designed so that the viewing area is selected such that the positional accuracy can be met without providing an excessive pixel resolution. There is no use being able to calculate the position to within 0.5 mm when only 5 mm is sufficient.

The boundary tracing algorithm, is presented in its entirety (in terms of the code written) in Appendix D. The reader can peruse the code and all the functions associated with it. The algorithm is, of course, essentially useless without additional rules that can utilise the boundary information extracted. The entire vision scheme is too large to be included in the appendix.

It is always easier to understand code if it is presented in block diagram or pseudo-code format. The boundary of the object is placed in a two dimensional array. The first element represents the x pixel coordinate and the second the y pixel coordinate. The root pixel is, in the first instance, the start of the boundary of the object. After the root pixel is determined, the neighbouring pixels are analysed. If a pixel is found that indicates the boundary is extending to this area, this pixel then becomes the root pixel
for the next analysis. The process continues until the boundary returns to the first root pixel, or it exceeds the number of expected points in the boundary. Figure 6-7 presents the pseudo code for the boundary tracing algorithm.

Assign variables for use by the boundary tracing function
Determine the coordinate of the object root pixel
Assign root pixel coordinate to the first member of boundary array
Determine the physical addresses of 8 neighbours of the root pixel
Assign address of first search position to be neighbour 0
While(not exceeding number of boundary points)
  While(not at root pixel)
    While(analysing neighbouring pixels)
      If(neighbour > threshold)
        Update position and define new root pixel
        Assign new neighbour addresses
        Enter new member into boundary array
        Accumulate x and y pixel coordinates
        Repeat for next neighbour
      End
    End
    Non-continuous boundary, false start, try again
  End
  Return error from function
End
Back at root pixel - end of process
Calculate centroid of object
Return from function
End
Tracing incorrect boundary - try again
Return error from function

Figure 6-7 Boundary Tracing Pseudo Code

In order to understand the process more clearly, a series of frames are presented showing the steps of the boundary tracing process.
It is worth noting that the boundary tracing process also performs the calculations required to establish the centroid of the object being traced. Establishing the points of the boundary is an integral part of the centroid calculation.

An important variation of the boundary tracing process is that the algorithm used does not always start the search for the next pixel in the boundary from the 0th neighbouring pixel. Instead the algorithm commences the search at a position based on the location of
the pixel found in the previous iteration. In figure 6-8, the search for the next pixel in
the second iteration would commence at neighbour 1 and in the fourth iteration, the
search would commence at neighbour 2. The rule used states "commence the search in
the new iteration at the neighbour 1 location counter-clockwise from the root pixel
found in the previous iteration".

By using this technique, the algorithm tends to look for pixels in the direction the
boundary is being traced and this improves the iteration time.

6.4.4.1 Pixel Addressing

In order to perform fast access to the image, the code calculates the address of the pixel
in memory. This is the reason why understanding the placement of the image data into
VRAM is important. The organisation of image data is also used by the function that
allocates the addresses of the neighbouring pixels of the root pixel. A general rule for
accessing the i'th pixel if the j'th line captured is:

\[
\text{pixel address} = \text{base address} + [(j - 1) \times 1024 + (i - 1)]
\]

The base address is, of course, the location in memory of the first pixel to be captured.
In determining the location in memory of the pixels to be investigated, consider the
following method.

An image frame consists of \( r \) rows and \( c \) columns. Let the location of the root pixel be
defined by \((r_r, c_r)\). This defines the position of this pixel in much the same way that one
would define a location in a cartesian coordinate system. For example, a location of
\((3,5)\) would mean the pixel is located on the third row, fifth column.

The image data can be dealt with as though it were an array as illustrated in figure 6-9.
This array could be called \( \text{image}[r \times c] \). The location of the root pixel would then be
defined as:

\((r_r \times c + c_r)\)'th element of the matrix, or
Thus if \textit{image} was defined as a pointer that referenced the starting physical address of the image data, then \textit{image[305]} would be the address of the root pixel in the above diagram. In the case of the TTGF hardware, the starting address of the image data in VRAM could be 0x80800000 (hex) and therefore the address of the pixel located at row 3, column 5 of a 500 by 100 frame would be 0x80800131 (hex). This pixel could be used in the code either by a pointer that points directly to this address, or the 305'th element of an array whose 0'th element is located at 0x80800000.

It is now possible to define the addresses of the 8 neighbours of this pixel quite easily. Each of these addresses can be an entry in another array such that the array entry holds the address of the neighbour in question. That is, entry 2 of the neighbour array, for example, would have the address of the neighbour 2. An array, \textit{associate[8]}, could be defined to hold these addresses. The pseudo-code in figure 6-10 is used to assign the addresses of the neighbouring pixels to the elements of this new array.
\[
\text{pixel\_row} = \text{pixel\_row\_reference} \times \text{number\_of\_columns} \\
\text{previous\_pixel\_row} = \text{pixel\_row} - \text{number\_of\_columns} \\
\text{next\_pixel\_row} = \text{pixel\_row} + \text{number\_of\_columns} \\
\text{pixel\_column} = \text{pixel\_column\_reference} \\
\text{previous\_pixel\_column} = \text{column\_reference} - 1 \\
\text{next\_pixel\_column} = \text{column\_reference} + 1 \\
\text{associate}[0] = \text{image}[\text{previous\_pixel\_row} + \text{pixel\_column}] \\
\text{associate}[1] = \text{image}[\text{previous\_pixel\_row} + \text{next\_pixel\_column}] \\
\text{associate}[2] = \text{image}[\text{pixel\_row} + \text{next\_pixel\_column}] \\
\text{associate}[3] = \text{image}[\text{next\_pixel\_row} + \text{pixel\_column}] \\
\text{associate}[4] = \text{image}[\text{next\_previous\_pixel\_row} + \text{pixel\_column}] \\
\text{associate}[5] = \text{image}[\text{next\_pixel\_row} + \text{previous\_pixel\_column}] \\
\text{associate}[6] = \text{image}[\text{pixel\_row} + \text{previous\_pixel\_column}] \\
\text{associate}[7] = \text{image}[\text{previous\_pixel\_row} + \text{previous\_pixel\_column}] \\
\]

Figure 6-10 Associate Pixel Address Calculation Pseudo Code

Diagrammatically, the above references are illustrated in figure 6-11.

Figure 6-11 Associate Address Allocation

This variation in boundary tracing simplifies the boundary tracing process and improves the execution time. Instead of dealing with the data as a two dimensional array, it is dealt with as a linear block of data. Rather than calculating the new address of each
associate every time it is to be analysed, it needs only to be calculated once, each time
the new root pixel is established. Analysis of the surrounding pixels then becomes a
matter of incrementing an array to point to the next pixel in the search. It is important
when allocating these addresses that the boundary of the image data is preserved. If it is
not, then the code may reference data outside of the image frame.

The full source code for these routines are included in appendix D with the boundary
tracing code.

6.4.5 Position and Orientation Calculation

The location of the centroid of the object with respect to the boundary of that object is
invariant under rotation and translation in a two dimensional system of axes. In the case
of the chicken leg as illustrated in figure 6-6, regardless of the orientation of the leg in
the image plane, the centroid of the drumstick is always in the same position. Literature
published in this area of image processing will often refer to moments of area whilst
discussion blob feature analysis. In general, the moments of the blob are calculated on
binary images using all pixels contained within the boundary of the blob. However,
using only the boundary of the object, the position of the centroid can still be
calculated. Other parameters such as orientation, area, major and minor axes etc cannot
be determined. Thus, there is the question of establishing the orientation of an object
that is otherwise unknown to the vision system. This will be discussed later.

Suppose the coordinate (with respect to the image plane) of each of the pixels located
on the boundary of the object is known. This can be represented by a two dimensional
matrix, \( B(x_n, y_n) \), where \((x_n, y_n)\) represent the coordinates of the boundary points and \(n\)
represents the total number of points in the matrix. The coordinate of the centroid of the
object is then given by [D3]:

\[
x_c = \frac{\sum_{0}^{n} x_n}{n} \quad y_c = \frac{\sum_{0}^{n} y_n}{n}
\]
where:

\[ x_c \] is the x coordinate of the centroid with respect to the image plane, and

\[ y_c \] is the y coordinate of the centroid with respect to the same plane.

Strictly speaking, the centroid of an object describes the centre of mass of that object and should therefore take into account the entire object and not just the boundary data. For example, the object may have a hole located off centre that influences the location of the centre of mass. By using the boundary pixels only, the object is assumed to be solid.

In order to determine the orientation of the object, there are two methods available that will produce satisfactory results. One method is to use the moment invariant theory with information about the entire object and not just the outer boundary. The other is to place a feature mark, easily identifiable, on the top surface of the object. Typically, this mark would be black or non-reflective. The latter option can also be implemented if the object being analysed has features already available. For example, a hole located in the object could be used in some circumstances.

In order to find the orientation, there are two uses of the boundary tracing procedure. One traces the outer boundary of the object and the other traces the boundary of the feature mark as shown in figure 6-12. By determining the angle of a line between the centroid of the object and that of the feature with respect to an axis of the image plane the orientation is easily calculated.
Using this technique, it is possible to assist in the processing time by careful placement of the feature mark. Since the geometry of the object being manipulated is known and the centroid of this object is also known, then by placing the feature mark close to the object centroid it is possible to fix a window size and position to search for the feature. Once the feature is found within this window, it is then possible to create a sub-window such that during the next frame, the search area for the feature mark is reduced further. However, since the area is small anyway, this method provides little improvement in processing time. The feature itself can be made relatively small so that the number of pixels located on the boundary is small. This will provide assistance in processing speed.

This technique will work for manufactured items. Other methods to determine the orientation of an object need to be established for items outside this category.

6.5 Vision System Performance

The execution performance of the vision system is important since it will largely determine the positional accuracy that the control system will be able to obtain. The
processing time for each image is dependent upon a number of properties. It can be said generally that:

1. the processing time is scaled by the number of boundary points located on an object,

2. without the windowing techniques (mentioned earlier), the processing time is scaled by the position of the object within the frame, and

3. blurring, or extremely noisy edges present a major problem.

Recall that the original performance criteria set down was 100 milliseconds per frame of data. Stated in a more precise manner, an update of the position and orientation of the object being manipulated is required every 100 milliseconds.

The vision scheme developed for the prototype 36 element PAM provides an update for the position and orientation of a single object to the object control layer every 155 milliseconds. This is outside the tolerance specified originally. The implication of this result needs to be considered.

The original estimation of transportation velocities was made using a simulation of the single element vibratory PAM. This simulation showed controlled motion at speeds up to 100 mm/s. However, as is presented in chapter 7, the results of work performed using the 36 element prototype PAM indicate that velocities in the order of 60 to 80 mm/s are more reasonable for a multiple element PAM. In this case, given the same positional accuracy, a processing speed of 167 to 125 milliseconds is acceptable. In this case, the vision scheme provides the performance required. If the same position tolerance is required (+/- 5 mm), then the maximum translation velocity is 65 mm/s. Alternatively, if the positional tolerance is relaxed, higher velocities may be used. Chapter 7 presents details of the experimental verification of the vision system performance.
The scheme provided for the prototype PAM need not necessarily be the final scheme. As is outlined in chapter 8, there are a variety of performance improvement measures that may be undertaken that could remove the processing bottleneck of the vision scheme.

6.6 Summary

During the course of this chapter, basic image processing techniques have been presented. In many image processing applications, little or no material exists that can assist designers in selecting hardware options for the task at hand. Much of it is common sense and experience based rules. Some of these rules have been presented.

The concept of boundary tracking has been presented as has the algorithm designed for the PAM vision system. It is a generic algorithm that would have application in many other areas. The use of the location of the boundary points for calculation of the object centroid is a quick method for tracking the position of the object. Even though in some cases the true centroid is not determined, it provides a reference point relative to the boundary of the object that is invariant under translation and rotation.

Although these techniques are well established, a method was presented that speeds up conventional processing algorithms. This method involves direct addressing of image pixels and looking for boundary pixels in the direction the boundary is moving, rather than starting from a fixed point. A short-cut method for determining orientation was justified by the highly complex nature of a generic method.
This concludes the design and implementation work associated with the PAM project. Details of the concept of PAM, the actuators used and the interface hardware designed have all been given. Finally, in this chapter, details of the vision system required to make the PAM work were described in this chapter. Chapter 7 provides details of the experimental work performed in order to verify the design presented.
CHAPTER 7

EXPERIMENTAL RESULTS

7.1 Introduction

The control and vision scheme provides the software necessary to move an object around a PAM in a repeatable and reliable manner. The hardware to provide these mechanisms has been designed, built and proven correct. The principle of the vibratory element PAM was tested on a single element prototype, then miniaturised and extended to a thirty-six element prototype PAM. Up until this chapter, little discussion has been made about the way in which an object behaves on a PAM, i.e., the way it rotates or the way it translates.

This chapter seeks to provide the experimental results of the various software strategies used to control translation and orientation. The performance parameters of the trial PAM and the best way to move an object on the PAM is experimentally found. A classical approach would be to model the mechanics of the PAM and use this in the control of the translational and rotational processes. This modelling is currently under research. However, the immediate need to prove the functional usefulness of the PAM requires a practical approach. This approach seeks to establish rules via a trial and error methodology. The shortfall of such an approach is that is cannot provide an explanation for the behaviour of an object on the PAM. It does, however, produce experimental results which can be later verified by dynamic modelling.
7.2 36 Element Test Manipulator

Much of the mechanics of the element actuator was presented in chapter 3. Recall that the element consists of a single solenoid with a tile of dimension 15 by 15 mm placed at the top of the solenoid slug, or armature (refer to figure 3-4). The simple actuating motion is the vertical movement of the solenoid armature. The upward motion is initiated by energising the solenoid. That is, allowing current to pass through the winding of the solenoid. The downward motion is provided by springs that are compressed during the upward stroke. Thus, when the solenoid is de-energised, the solenoid armature returns to the position set by the spring adjustment. Such an arrangement is a very simple design mechanically, yet it is difficult to adjust and keep the system maintained in adjustment.

7.2.1 Mechanical Arrangement

The entire assembly of 36 actuators is mounted on a plate that orbits at a variable frequency. The drive mechanism is provided by a modified orbital sander device. Figure 7-1 is a sectioned view of the trial 36 element PAM.

The mechanical arrangement was, as predicted, difficult to maintain. Throughout the life of the prototype, the spring tensions were readjusted several times. Even now, the top surface of the PAM is uneven and causes problems when attempting to translate objects.

The drive system, not shown in figure 7-1, consists of a modified commercial orbital sanding device. The orbit provided by the sander is not circular, but slightly elliptical. This has little effect on the way an object can be translated and rotated.
Figure 7-1 36 Element PAM General Arrangement

(reproduced with the permission of Mr N. Laszlo)
7.2.2 Electrical Interface

The details of the interface designed to drive this array are provided in chapter 3. Although the drive signals to the solenoid are a PWM style, it is the current in the solenoid windings that is responsible for the movement of the solenoid armature. It was determined that the solenoid excitation has a significant effect on the way that an object is transported on the PAM.

For example, by exciting the solenoids by a simple square wave with a 50% duty cycle, the object would tend to bounce on the PAM. The acceleration that the solenoids were experiencing was so high that the object resting on the surface would be tossed into the air. This is not satisfactory, since the object needs to remain in contact with the surface in order for controllable motion to occur.

In another instant, the solenoids were presented with a PWM version of a triangular wave, with the same half-cycle period as the PAM orbit period. Instead of the object being thrown into the air as for the square wave case, a gentle vibration was produced. This achieved the desired effect of controllable motion. This style of excitation was marginally better than the sinusoidal technique.

It is difficult to quantify which technique was the best. In all cases, the direction of translation of the object could be controlled. In the case of the square wave excitation, the problem was the deviation from the straight line translation under open loop conditions. However, in the case of the triangular and sinusoidal excitation, both techniques produce satisfactory results. Straight line motion was observed in both instances; however the triangular excitation was considered superior because the audible noise level of the PAM was reduced.

These results are only valid for the tested style of actuator. An alternative actuation mechanism may produce different results. Thus it could be concluded that the excitation
is best optimised through trial and error means, since the actual excitation originally predicted did not result in the best motion effects.

7.3 Object Control

In the control of an object on PAM, it is important to know how an object will move on the surface of the PAM, given a particular phase of excitation. Without this knowledge, it is not possible to control the motion of the object across the PAM surface. The experimental work is an important factor in trying to develop a model. By understanding the behaviour of an object, given a particular set of phases, the modelling can be refined.

7.3.1 Control of Translation

In the experimental work associated with the control of object translation, several fundamental relationships were to be investigated. These are:

1. Translational velocity for various objects,
2. Directional stability relative to orbital frequency,
3. Repeatability of direction of translation for different object properties, and
4. Directional resolution.

The property 1 is to be investigated in order to establish if the model developed for the single element PAM and the subsequent simulation of the single element actuator (refer to section 3.2) were correct for the multi-element case. The property (2) is also related to the fact that the simulation indicated that there will be a maximum orbital frequency which, if exceeded, would result in the loss of directional stability.

The property (3) is important since if the PAM is to be capable of manipulating multiple objects, of different physical properties, then they should all respond the same way. If there was excessive variation in the object response, then more complex object
identification techniques would be required. Alternatively, an adaptive control algorithm would need to be implemented. Such an algorithm would modify the look up table parameters for directional control based on the real time measurement of the object response on the PAM.

The property (4) would reveal whether or not the idea of PWM phase control is successful. Without the angular resolution, the interface would need to be redesigned until an acceptable resolution was obtained.

The objects tested were:

1. an aluminium disk, 12 mm thick and 75 mm in diameter, and mass 223 g,

2. a steel disk, 15 mm thick, 75 mm diameter and mass 777 g,

3. a paper notepad, 75 mm square and mass 43 g, and

4. a roll of insulation tape, 65 mm diameter, 12 mm thick and mass 61 g.

7.3.1.1 Results of Velocity Trials

The mass of the objects had little effect on the speed of translation. This statement does, however, require some qualification. The distance travelled was small since the array only allows approximately 60 mm of straight line travel. Since this distance was small, there was no way of knowing whether the object would continue to accelerate had it been allowed to travel further. This did not appear to be the case; however a larger PAM is required to validate the result fully.

The tests performed indicate a speed of 30 to 40 mm/sec at a rotational frequency of 23 Hertz. There was a significant amount of uncertainty in this measurement due to the PAM itself. The uneven surface of the PAM meant that the object would not always travel in a straight line at a continuous speed. The object would often deviate around high areas of the PAM, or stall momentarily when a high spot was reached.
The important factor demonstrated was the consistent velocity despite the variation in object parameters.

7.3.1.2 Results of Directional Stability Tests

The simple result of these tests is that there is no directional instability that can be measured up to an orbital frequency of 35 Hertz. This can be explained by two reasons. Firstly, the prototype PAM is too small to measure the directional stability. The distance that the objects can translate is too short to state, with a high degree of certainty, that such stability exists. However, over the distances travelled of approximately 60 mm, the translation along straight lines was observed.

The other reason is because the PAM cannot orbit faster than 35 Hertz. Above this speed, the drive motor current becomes excessive and an overload occurs.

The conclusion reached for this prototype is that there is no directional instability when operating the PAM at an acceptable orbital frequency.

7.3.1.3 Directional Resolution

This experiment was performed to determine if the hardware designed for the control of the PAM was working as required. The designed allowed for up to 32 different directions of translation.

The following diagram demonstrates the phase versus direction characteristics at an orbital frequency of 23 Hertz using the Aluminium disk and the Steel disk as the objects. All directions are referenced to the direction obtained when exciting the solenoids with phase 1 (of 32).
The test also determined that there was little effect on the direction of translation for different masses of object. This is observed by inspection of figure 7-2. The Aluminium disk and the Steel disk have considerably different masses. Despite this difference, the direction of translation versus phase characteristic is similar.

One oversight in the design was the inability to stop the object. In the original design, 32 directions were allowed for. This was the way the circuit board was designed. In order to stop the object, the solenoids must be without any excitation. In order to allow for this condition, phase 0 was converted to the zero excitation phase. As a result, there are only 31 available directions, with the difference between phase 31 and phase 1 being twice that of any other unit phase difference.

7.3.2 Control of Rotation

In the experimental work associated with the control of object rotation, several fundamental relationships were to be investigated. These are:
1. Rotational velocity for various objects,

2. Rotational stability relative to orbital frequency, and

3. Repeatability of direction of rotation for different object properties.

These properties are of equal importance as the translational properties since the nature of PAM and its applications demands rotational accuracy.

It is important to note that there is a tendency for the object to rotate when the solenoids are not excited. The rotational velocity for the object is different for different objects. In all cases, the direction of rotation was opposite to the direction of orbit rotation. The following figure lists the objects and their natural rotation frequencies. The natural rotation is in the same direction as the direction of orbit. When the PAM is reversed, the natural rotation frequency also is reversed. Table 7-1 quantifies this effect.

<table>
<thead>
<tr>
<th>Object</th>
<th>Natural Rotation*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aluminium Disk</td>
<td>0.12 CCW</td>
</tr>
<tr>
<td>Steel Pulley</td>
<td>0.05 CCW</td>
</tr>
<tr>
<td>Paper Notepad</td>
<td>0.19 CCW</td>
</tr>
<tr>
<td>Insulation Tape</td>
<td>0.25 CCW</td>
</tr>
</tbody>
</table>

* revolutions per sec

Table 7-1 Natural Rotation Frequencies

When attempting to rotate an object on a PAM, it is desirable to impart a rotation without inducing a translation. This improves the usefulness of PAM. In order to do this, several different techniques were tried, many of which introduced a translation, whilst still successfully rotating the object.

In describing the method used for rotating the objects, consider figure 7-3. Each arrow represents a particular phase of solenoid excitation that would make the object translate in the given direction.
Due to the natural rotation of an object, it would be expected that the object could be rotated faster in the CCW direction. However this is in fact not the case. The best CCW rotation performance is achieved for the CW case. Table 7-2 lists the best clockwise and counter-clockwise rotation performances.

<table>
<thead>
<tr>
<th>Object</th>
<th>Best CW Performance</th>
<th>Best CCW Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aluminium Disk</td>
<td>0.40</td>
<td>0.15</td>
</tr>
<tr>
<td>Steel Pulley</td>
<td>0.45</td>
<td>0.19</td>
</tr>
<tr>
<td>Paper Notepad</td>
<td>0.27</td>
<td>0.18</td>
</tr>
<tr>
<td>Insulation Tape</td>
<td>0.40</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 7-2 Rotation Performances (revolutions per sec)

There is a principle that relates the number of elements in contact with the object with the ability to rotate the object. Prior to the experimental stage, it was known that there would be a connection with this ratio and the rotational performance. As a result of the experimental work, it has been determined that for flat based objects, the number of PAM elements required to rotate the object satisfactorily is of the order of 12 to 16. There does not appear to be this minimum number associated with the translational aspects of PAM. However, improved performance is achieved with more elements.
7.4 Vision System Performance

7.4.1 Static Conditions

In order to determine the accuracy of the vision system, a series of static tests was performed. The test involved the placement of an object at a series of known positions and a series of known orientations. The vision system was then used to calculate the position and orientation of the object. Instead of using the trial PAM, the static test was performed using an object placed on a piece of cardboard. This allowed the vision system to be tested over a greater range of values than otherwise would be possible.

Figure 7-4 demonstrates the layout of the position test series.

![Diagram of object position test layout](image)

Figure 7-4 Object Position Test Layout

The results of the test are quantified in Table 7-3. The results demonstrate the effectiveness of the vision system under static conditions. The desired accuracy of the position measurement was +/- 5mm and the results indicate that this accuracy has been met.
Table 7-3 Results of Static Position Test

The next static test involved the placement of the same object at various orientations. The method of using a feature mark to calculate orientation was employed. Figure 7-5 illustrates the test scenario.

Table 7-4 quantifies the results of this test. The values of the orientation are given in degrees with respect to the horizontal axis.
Table 7-4 Static Orientation Test Results

<table>
<thead>
<tr>
<th>Position</th>
<th>Actual Orientation</th>
<th>Vision Orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>43</td>
</tr>
<tr>
<td>43</td>
<td>90</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>135</td>
<td>132</td>
</tr>
<tr>
<td>5</td>
<td>180</td>
<td>176</td>
</tr>
<tr>
<td>6</td>
<td>225</td>
<td>230</td>
</tr>
<tr>
<td>7</td>
<td>270</td>
<td>268</td>
</tr>
<tr>
<td>8</td>
<td>315</td>
<td>312</td>
</tr>
</tbody>
</table>

These results clearly demonstrate the ability of the vision system devised for PAM to operate within the desired tolerance whilst measuring static objects.

7.4.2 Dynamic Conditions

When the objects move, the problems do not arise from the vision algorithms, but rather from the hardware. A moving object will cause blurring of the image if the aperture or exposure time is too long. By shortening the exposure time, the blurring can be removed, but the contrast is reduced. In the case of PAM, it was necessary to introduce additional lighting so that sufficient contrast could be maintained whilst keeping the exposure time high.

It is difficult to quantify the performance of the vision scheme under dynamic conditions because the objects are moving and therefore cannot have the position measured accurately. However, since there is no blurring of the image, then the processing software cannot tell the difference between the static and dynamic case. The real issue under dynamic conditions is the processing time and its impact on the position control algorithms.

In order to determine the accuracy of the position and vision control under dynamic conditions, a series of tests was devised that called for the transportation of the object to a particular position coordinate. Upon arrival at this position, the solenoids were de-
energised and the PAM was manually stopped. This was reasonable since there is no mechanism for the control scheme to control the orbit of the PAM.

Once the PAM was stopped, the position of the object was measured and compared to the desired position. Table 7-5 presents the results of this test. The results are given in millimetre coordinates. In all cases the starting position of the object for the test was at the 80,60 coordinate.

<table>
<thead>
<tr>
<th>Trial Number</th>
<th>Desired Position</th>
<th>Actual Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40,40</td>
<td>44,45</td>
</tr>
<tr>
<td>2</td>
<td>80,40</td>
<td>80,45</td>
</tr>
<tr>
<td>3</td>
<td>60,60</td>
<td>64,63</td>
</tr>
</tbody>
</table>

Table 7-5 Dynamic Positioning Results

The tests performed are far from conclusive since they only represent a translation of approximately 40 mm (straight line). However, despite the short traversal distance, the object was able to be located at the desired position within the tolerance previously assigned.

7.5 Summary

This chapter has provided the reader with the performance characteristics of the 36 element PAM prototype. The conveying velocities achieved are less than that obtained from the single element prototype. This appears to be due mainly to the reduced orbit diameter of the 36 element prototype.

The rotational frequency was never considered in the original performance specification. Additionally, there was some doubt as to the ability to perform rotations in a controlled manner. This was proven incorrect. There are many ways to make an object rotate on a PAM. It is more difficult to stop the rotation rather than start it due to
the tendency to rotate naturally. The rotation is observed as a natural phenomenon and must be compensated for in the position and orientation control scheme.

The success of the directional control hardware was established by experimentally verifying the direction of translation for a variety of excitation phases. The directional resolution was verified to be 31. Although originally designed to be 32, one direction had to be used as the zero vector to stop the object from moving.

Overall, the results obtained in rotating and translating an object were very encouraging. The PAM can deliver satisfactory performance, based on the original design criteria. The strategies developed and tested during the experimental phase form part of the software associated with the Object Control Layer of the control system.
As can be seen from figure 7-6, the main body of the report has been completed. This chapter has demonstrated the effectiveness of the hardware devised to control the direction of translation of an object on PAM. It has also shown that the control software is capable of translating and object in the desired direction. The ability of the vision system to measure position and orientation of the object to within the specified tolerance completes the effectiveness of the control scheme.
CHAPTER 8

CONCLUSION

8.1 Introduction

The final chapter of this thesis is concerned with summarising the result of the work performed for the project. It provides a generalised discussion of this work and recommendations for future research activities. Since the work performed was essentially for a prototype PAM, much more work is required before a commercially viable product can be produced. The bulk of this work is in the vision processing aspect of the control scheme. The fundamental control philosophy based on demand-pull mechanisms has been proven to work under simulated conditions.

8.2 Multi-Layer Distributed Control Architecture

The selection of the multi-layered distributed control architecture was justified by the need to keep the modularity of PAM. This feature has been designed from the mechanical arrangement right through to the control scheme. A centralised control architecture suffers from many disadvantages including:

1. limited expandibility,

2. difficult to expand and modify,

3. possible computational overhead burdens, and

4. non-modular structure.
On the other hand, the distributed control architecture provides many advantages. The most important advantage is the ability to maintain a highly modular, loosely coupled series of processes for the control scheme. In this way, the PAM would have virtual limitless expansion capabilities without overextending the control scheme.

The proposed layering of the control system is a sound practice. It modularises the code into workable tasks without overextending the depth of knowledge required to deal with the software. If instead of layering the task, a single large program was written, then the task of debugging is simplified. Additionally, since the processor used is the transputer, the task of mapping the control onto the array would be difficult with a single layer program. Layering the software makes the job of processor/process allocation simpler. It also creates a more structured environment where changes and additions are far easier to implement.

The conceptual control system based on the demand-pull mechanism has been shown to be effective by the simulation. The design removes the need for a centralised command and control supervisor. Each control module is responsible for the movement of objects on the corresponding manipulation module. The demand-pull mechanism idea is scalable and could also be used for the control of the object within a single MM. The synchronisation of the demand pull pulse to the event link of the transputer removes the need for a complex network of communication to propagate these messages.

The simulation of the control philosophy provided valuable insight into the data that needs to be exchanged between control modules for the mechanism to work correctly. The fears of possible traffic jams of material due to the demand-pull philosophy were abated by the simulation. Instead, the product flow appears to be logical and ordered.

Although the command layer could not be implemented in full, the other layers have. In particular, the object control and vision layers were proven to be effective in controlling the position and orientation of an object on the surface of PAM. Apart from the obvious mechanical problems relating to the design of the prototype, much knowledge
was obtained about the methodology of dealing with PAM. Since the idea of this style of programmable logic manipulator is new, little was known about the mechanisms required to produce the style of manipulations desired. The software implemented provided the ability to rotate and translate the test objects.

During the course of the control development, the principal that relates the number of elements in contact with the object with the ability to drive the object was established. Prior to the experimental stage, it was known that there would be a connection with this ratio and the rotational and translational performance. However, the values associated with this connection were not well known. Consequently, it has been established that for flat based objects, the number of PAM elements required to rotate the object satisfactorily is in the order of 12 to 16. There does not appear to be this minimum number associated with the translational aspects of PAM. However, better performance can be achieved with more elements.

The design of the scheme, being highly modular, lends itself to reconfiguration easily. That is, a PAM made up of 5 by 5 MM's is easily extended into a 5 by 6 arrangement, for example, without major software revisions. Instead, only an update of the configuration data in each control module is required. This is of great benefit as it enhances the generic value of PAM considerably.

The use of programmable logic added flexibility to the prototyping stage of the project. It was possible to test various bit pattern excitation schemes and to establish the best PWM pattern to obtain the desired object response. The design for the PWM scheme to drive the excitation of the solenoids is unique and interesting. The design had to be implemented digitally which presented a challenge. The design established, although not cost effective, provided the required interface and driving logic in one.

The aim of designing and implementing a control scheme that was modular and reconfigurable was satisfied. The design was based on the transputer and its support languages.
8.3 PAM Vision System

The vision system designed for the PAM prototype is a simple but effective scheme. A simple boundary tracing algorithm was designed and implemented. This allowed the use of the theory of moment invariants for planar geometrical figures to be used to determine the position of the object. The test object, being a circular disk, also required orientation to be determined. The same technique of boundary scanning was used to determine the position of a feature mark on the disk itself. The orientation was then measured using the angle relationship between lines drawn through the feature mark centroid and the object centroid.

The performance of the vision system was satisfactory with regard to the determination of the position and orientation information. However, the execution time was not as fast as was desired. Improvement of the execution time could be made if a high level of parallelism was introduced. Alternatively, there are commercial products available to fill the needs of such vision systems that have highly specialised hardware. Although this prospect is far too expensive for a prototype, a commercial application of PAM could find it acceptable. Nevertheless, the image processing system established for PAM allowed the use of vision for feedback in a position and orientation controller to be proved for PAM.

8.4 Future Work

There is a tremendous amount of research which needs to be continued in order for PAM to eventually become a commercially viable product. The prototype designed for PAM is far from the desirable unit one would expect in commercial applications. The new 512 element mark 2 prototype seeks to address some of the problems. This design, however, needs to be proven effective before it could be considered commercially. Thus, much of the work already performed in the area of object control on the mark 1 prototype needs to be duplicated for the mark 2 version.
The area of vision is perhaps the greatest challenge faced by PAM. Sorting and palletising of highly irregular objects such as meat cuts, fish, chicken legs etc is a difficult task. It is compounded by the fact that there are potentially large numbers of these items to be dealt with simultaneously. The use of PAM as a parts feeder would potentially reduce the workload of such a vision scheme since the parts would be regular, predictable and of known dimension.

Another area to be developed is in the establishment of a predictive model of object motion that would be suitable for embedding into the control system. Such a model would reduce the amount of feedback required from array sensors during automated operation.

8.5 Conclusion

Chapters 1 and 2 introduced the idea of an array manipulator, whilst in chapter 3 the design principles behind the hardware of such a device were presented. The conceptual control scheme was described in chapter 4. The scheme is based on the idea of demand pull. The modularity of the scheme is a vital factor that enhances the use of the scheme for a PAM controller. Chapter 5 described in detail the mechanisms of demand pull with respect to the PAM idea. The mechanisms were simulated and much was learnt about the information and communication requirements of this scheme.

Chapter 6 presented the simple vision system established for the prototype PAM. Although the bandwidth of the system was not as high as expected, it still allowed reasonable performance of the PAM under partially closed loop conditions. Chapter 7 offered the results of the experimental work performed to establish the viability of PAM as a material handling device.

The basic aims of the thesis have been satisfied. At the start of the project, the design of the PAM was essentially unknown and hence it was difficult to set a strict design goal to be met. There was no knowledge about the way an object would behave on a PAM,
or about the design of the actuators themselves. During the course of the project, the research team established a new and exciting mode of vibratory transport.

This new mode meant that vibratory feeding need not necessarily be omnidirectional as it had been previously. The fundamental ideas from the single element PAM actuator were modelled and it was established that directional control coupled with stability could be attained. The idea was miniaturised and the first multi-element prototype was built. A second, more refined, device is currently under construction and will further develop the concept of PAM further.
BIBLIOGRAPHY

A. Parallel Processing


B. Transputer General


C. Transputer Hardware


Bibliography


D. Vision Systems


E. Control Of Manufacturing Systems


F. Mobile and Modular Robotics


G. Part Feeding and Orientation


H. General


APPENDIX A

XILINX SCHEMATIC DIAGRAMS

This appendix contains the schematic diagrams of the hardware used to drive the solenoids of the 6 by 6 prototype PAM. Not all of the schematics have been included, mainly for reasons of reducing the amount of drawings presented. Since much of the design is repetitive, these instances are noted and subsequently left out.

The first set of drawings represent the logic for the circuits that accept the serial information from the communication link and use it to drive the solenoids with the requested phase.
CLB

SELECT[3:0]  SELECT0  SELECT1  SELECT2  SELECT3  SELECT

EQUATE=F=(~A*~B*~C*~D*E)

BASE=F  CONFIG=X:Y

A B C D E

DI EC K RD
EQUATE = F = (A* B* C* D* E)

CLB

SELECT[3:0]
SELECT0
SELECT1
SELECT2
SELECT3
SELECT
Title  
SOLD1.1

<table>
<thead>
<tr>
<th>Size</th>
<th>A4</th>
<th>Revision</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drawn</td>
<td>PPC</td>
<td>Sheet</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of</td>
<td>16</td>
</tr>
<tr>
<td>Design</td>
<td>PPC</td>
<td>Date</td>
<td>31/8/92</td>
</tr>
</tbody>
</table>
EQUATE-F = (\neg A \cdot B \cdot \neg C \cdot \neg D \cdot E)
EQUATE - F = (A*B*C*D*E)

CLB

SELECT[3:0]  SELECT0  SELECT1  SELECT2  SELECT3  SELECT

BASE = F
CONFIG = X: F

X

Y
Title: SOLD3.1

<table>
<thead>
<tr>
<th>Size</th>
<th>A4</th>
<th>Revision</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drawn</td>
<td>PPC</td>
<td>Sheet</td>
<td>8 of 16</td>
</tr>
<tr>
<td>Design</td>
<td>PPC</td>
<td>Date</td>
<td>31/8/92</td>
</tr>
</tbody>
</table>

Appendix A
EQUATE-F := (~A*~B*C*~D*E)
SELECT[3:0] SELECT0
SELECT1
SELECT2
SELECT3
SELECT

BASE=F CONFIG=X:F

EQUATE=F=(A\sim B*C\sim D*E)
EQUATE-F = (~A*B*C*D*E)
CLB

SELECT[3:0] SELECT0
SELECT1
SELECT2
SELECT3
SELECT

BASE-F
CONFIG=X:F

EQUATE-F= (A*B*C*-D*E)
CLB

SELECT [3:0] SELECT0 SELECT1 SELECT2 SELECT3 SELECT

BASE-F
CONFIG=X:F

EQUATE-F= (~A*~B*~C*D*E)
The next series of drawings represent the logic that produces the PWM bit patterns, for use by the previous logic. The drawing PH1.1 is 1 of 32 drawings, all very similar in appearance. The 0's at the D1 input of the FDMRD symbols represent the bit pattern logic values to be used. The other drawings in the sequence show the combination of 1's and 0's used in the generation of this bit pattern. The actual bit pattern used is shown in appendix B.
<table>
<thead>
<tr>
<th>Title</th>
<th>FIFOREG.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>A4</td>
</tr>
<tr>
<td>Revision</td>
<td>A</td>
</tr>
<tr>
<td>Drawn</td>
<td>PPC</td>
</tr>
<tr>
<td>Sheet</td>
<td>5 of 43</td>
</tr>
<tr>
<td>Design</td>
<td>PPC</td>
</tr>
<tr>
<td>Date</td>
<td>1/9/92</td>
</tr>
</tbody>
</table>
GCOMP

\[ \text{Diagram of a circuit involving XOR gates and an AND gate.} \]
APPENDIX B

PWM BITSTREAM CALCULATION SOURCE CODE

B.1 Source Code

BITGEN.C

Program to generate a Pulse Width Modulation bitstream from a formula which is assigned to function_value() below. The bitstream which is generated can represent signals up to a maximum value of 1.0 and down to a minimum value of 0.0. If the function doesn't start at the value of 0.0 there will be an initial period while the bits are all 1s or all 0s while it heads towards the initial value of the function. If the function changes at more than 1.0 per bit then the bit stream will not be able to keep up with the function. Use a smaller step size if this happens.

*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.1412
int bits[256];
double function_acc = 0;
double function_value = 0;
double bitsum = 0;
int x, word;
div_t result;
FILE *fp;

main()
{
    for(x = 0; x < 256; x++) bits[x] = 0; /* clear array */
    fp = fopen("c:\qc2\master\bitstrm.txt", "wt"); /* open file to write */
    fprintf(fp, "PWM BITSTREAM FOR SINE WAVE TO PAM (256 BITS TOTAL)\n\n");
}
fprintf(fp,"\nWord 0: (Function Value at first bit = 0.0000) \[\n\]
for(x=0; x<256; x++) { /* bitstream evaluation main loop */
    /* WRITE FUNCTION HERE */
    function_value = 0.5*(1.0-cos(2.0*((double) x)*PI/256.0));
    function_acc = function_acc + function_value; /* accumulate */
    bitsum = bitsum + bits[x]; /* ditto */
    if(bitsum < function_acc - 0.5)
        bits[x+1]=1;
    else
        bits[x+1]=0;
    word = x/8; /* tidies file formatting */
    result = div(x,8);
    if(word)
        if(result.rem == 0)
            fprintf(fp,"\nWord %d: (Function Value at first bit =
        \n0.0625) [00001000]
            fprintf(fp,"%d",bits[x]);
    }
    fprintf(fp,"\n\n");
    fclose(fp);
}

B.2 Sample Program Output

PWM BITSTREAM FOR SINE WAVE TO PAM (256 BITS TOTAL)
Word 0: (Function Value at first bit = 0.0000) [00000000]
Word 1: (Function Value at first bit = 0.0625) [00001000]
Word 2: (Function Value at first bit = 0.1250) [00000100]
Word 3: (Function Value at first bit = 0.1875) [00100001]
Word 4: (Function Value at first bit = 0.2500) [00010001]
Word 5: (Function Value at first bit = 0.3125) [00100001]
Word 6: (Function Value at first bit = 0.3750) [10100101]
Word 7: (Function Value at first bit = 0.4375) [00101010]
Word 8: (Function Value at first bit = 0.5000) [10101010]
Word 9: (Function Value at first bit = 0.5625) [11010101]
Word 10: (Function Value at first bit = 0.6250) [10110110]
Word 11: (Function Value at first bit = 0.6875) [11011011]
Word 12: (Function Value at first bit = 0.7500) [10111011]
Word 13: (Function Value at first bit = 0.8125) [11011111]
Word 14: (Function Value at first bit = 0.8750) [10111111]
Word 15: (Function Value at first bit = 0.9375) [11111111]
Word 16: (Function Value at first bit = 0.9922) [01111111]
Word 17: (Function Value at first bit = 0.9297) [11111111]
Word 18: (Function Value at first bit = 0.8672) [01111110]
Word 19: (Function Value at first bit = 0.8047) [11110111]
Word 20: (Function Value at first bit = 0.7422) [01110110]
Word 21: (Function Value at first bit = 0.6797) [11011011]
Word 22: (Function Value at first bit = 0.6172) [01101010]
Word 23: (Function Value at first bit = 0.5547) [11010101]
Word 24: (Function Value at first bit = 0.4922) [01010101]
Word 25: (Function Value at first bit = 0.4297) [00101001]
Word 26: (Function Value at first bit = 0.3672) [01001001]
Word 27: (Function Value at first bit = 0.3047) [00100010]
Word 28: (Function Value at first bit = 0.2422) [00100001]
Word 29: (Function Value at first bit = 0.1797) [00001000]
Word 30: (Function Value at first bit = 0.1172) [00000100]
Word 31: (Function Value at first bit = 0.0547) [00000000]
APPENDIX C

SINGLE ELEMENT PAM SIMULATION

C.1 Source Code

A simulation program for the single element vibratory actuator.

```c
#include <stdio.h>
#include <graph.h>
#include <math.h>
#include <float.h>
#include <stdlib.h>
#include <bios.h>
#include "vibsim.h"

main()
{
    int key;
    rerun = 1;
    while(rerun == 1){
        _clearscreen(_GCLEARSCREEN);
        input_user_data();
        initialise_variables();
        setup_screen();
        while(t < total_time){
            calc_plate_vel();
            calc_rel_vel();
            calc_friction();
            calc_imp_vel();
            update_vel_and_disp();
            plot_path();
            plot_plate();
            t = t + time_step;
        }
    }
    _settextposition(2,1);
```
void input_user_data(void) {
    
    printf("PROGRAM TO PLOT DISPLACEMENT OF OBJECTS ON THE PAM VIBRATORY TEST RIG\n\n");
    printf("Please enter the following information when requested:\n");
    printf("The orbital frequency of the plate (Hz): ");
    scanf(\"%f\",&f);
    printf("The radius of the plate orbit (mm): ");
    scanf(\"%f\",&rmm);
    printf("The coefficient of friction between the plate and the object: ");
    scanf(\"%f\",&u);
    printf("The maximum vertical displacement of the plate (mm): ");
    scanf(\"%f\",&avmm);
    printf("The initial X velocity of the object (cm/s): ");
    scanf(\"%f\",&x_vel_objcm);
    printf("The initial Y velocity of the object (cm/s): ");
    scanf(\"%f\",&y_vel_objcm);
    printf("The angular resolution required for each simulation step (degrees): ");
    scanf(\"%f\",&dtheta);
    printf("The total time the simulation is to run for (s): ");
    scanf(\"%f\",&total_time);
}

void initialise_variables(void) {

    w = 2.0 * pi * f;
    r = rmm * 0.001;
    av = avmm * 0.001;
    x_vel_obj = x_vel_objcm * 0.01;
    y_vel_obj = y_vel_objcm * 0.01;

time_step = dtheta/(360.0 * f);

/* Initialise time and displacement of object */

t = 0.0;
x_dis_obj = 0.0;
y_dis_obj = 0.0;

/* Initialise program flow parameters */

first_air_flag = 1;
air_flag = 0;
land_flag = 0;

}/****************************************************************
* — setup_screen —
* Set up graphics and plotting parameters
****************************************************************/

void setup_screen(void) {

    plot_widthcm = 5.0;        /* The distance represented (in cm) from the
                              * top to the bottom of the screen. Change
                              * this when required. */

    plot_width = plot_widthcm * 0.01;

    scrn_width = 247.0;        /* The height and width of the physical screen
                              * so that x and y distances will appear equal */

    scrn_height = 180.0;

    screen_lhs = 320.0 - (320.0 * scrn_height / scrn_width);
    screen_rhs = 320.0 + (320.0 * scrn_height / scrn_width);

    _setvideomode(_VRES16COLOR);

    /* Place text on screen */

    _settextposition (1, 47);
    _outtext("X-Y PLOT OF OBJECT ON ORBITAL RIG");

    _settextposition (3, 68);
    printf("f = %5.2f Hz", f);

    _settextposition (4, 68);
    printf("r = %5.2f mm", r);

    _settextposition (5, 68);
    printf("u = %6.3F", u);

    _settextposition (6, 67);
printf("av = %5.2f mm",avmm);

_settextposition (8, 68);
printf("At time t= 0:");

_settextposition (10, 62);
printf("x vel = %5.2f cm/s",x_vel_objcm);

_settextposition (11, 62);
printf("y vel = %5.2f cm/s",y_vel_objcm);

_settextposition (1, 28);
printf("y = %4.1f cm",plot_widthcm);

_settextposition (40, 27);
printf("y = -%4.1f cm",plot_widthcm);

_settextposition (17, 0);
printf("x = -%4.1f cm",plot_widthcm);

_settextposition (17, 70);
printf("x = %4.1f cm",plot_widthcm);

_settextposition (1, 0);
printf("TIME = s");

_setcolor(2); /* Draw grid lines for plot */
_moveto (320,479);
_lineto (320, 0);
_moveto (screen_lhs, 240);
_lineto (screen_rhs, 240);
_moveto (310, 0);
_lineto (330, 0);
_moveto (310,479);
_lineto (330,479);
_moveto (screen_lhs, 230);
_lineto (screen_lhs, 250);
_moveto (screen_rhs, 230);
_lineto (screen_rhs, 250);
_moveto( 320,240); /* Set plotting at centre of screen */
_setcolor(3);
}

/**************************************************************************/

--- calc_plate_vel ---

* Assumes the plate travels in a circular orbit, travelling clockwise with the orbit
* initially at 12 o'clock.
* 
**************************************************************************/

void calc_plate_vel(void){
    x_vel_plate = w * r * cos(w * t);
    y_vel_plate = w * r * -1.0 * sin(w * t);
Appendix C

197

void calc_rel_vel(void) {
    x_vel_rel = x_vel_plate - x_vel_obj;
    y_vel_rel = y_vel_plate - y_vel_obj;
}

void calc_friction(void) {
    if(air_flag == 1) {
        if(first_air_flag == 1) {
            first_obj_height = av * cos(w * t);
            first_vert_vel = -1 * w * av * sin(w * t);
            first_air_time = t;
            first_air_flag = 0;
        }
        friction = 0;
        air_time = t - first_air_time;
        obj_height = first_obj_height + first_vert_vel * air_time - 0.5 * g * air_time * air_time;
        plate_height = av * cos(w * t);
        if(obj_height < plate_height) {
            _setcolor(3);
            air_flag = 0;
            land_flag = 1;
        }
    }
    else {
        friction = u * (g - av * w * w * cos(w * t));
    }
}
if( friction < 0.0 ) { /* Check for object leaving plate */
    _setcolor(4);
    air_flag = 1;
    first_air_flag = 1;
}

/*---------------------------------------------*/
/* --- calc_imp_vel --- */
/*
* This function uses the magnitude of the frictional force and the relative direction
* of the velocity between the object and the plate to calculate velocity imparted to
* the object by the frictional force over one timestep. Linear acceleration is assumed
* over one timestep. The "if" statement is to check whether the frictional force is
* sufficient to move the object. If not, then the object is stuck to the plate surface
* and the object's velocity must be the same as that of the plate relative to the object.
* 
* On landing, an additional velocity impulse is given to the object equal to u times the
* vertical velocity of the object wrt the plate.
* 
* ---------------------------------------------*/

void calc_imp_vel(void) {
    impulse_vel_mag = 0.5 * friction * time_step;

    if (land_flag == 1) {
        impulse_vel_mag = impulse_vel_mag + u * (g * air_time - first_vert_vel - w * av *
            sin(w * t));
        land_flag = 0;
    }

    rel_vel_mag = sqrt( x_vel_rel * x_vel_rel + y_vel_rel * y_vel_rel);
    if( impulse_vel_mag < rel_vel_mag)
        impulse_vel_mag = rel_vel_mag;

    x_vel_impulse = x_vel_rel * impulse_vel_mag / rel_vel_mag;
    y_vel_impulse = y_vel_rel * impulse_vel_mag / rel_vel_mag;
}

/*---------------------------------------------*/
/* --- update_vel_and disp --- */
/*
* Add the velocity due to the frictional impulse to the object's original velocity.
* Also add the new displacement vector to the object's original displacement.
* 
* ---------------------------------------------*/

void update_vel_and_disp(void) {
    x_vel_obj = x_vel_obj + x_vel_impulse;
    y_vel_obj = y_vel_obj + y_vel_impulse;
x_dis_obj = x_dis_obj + 0.5 * x_vel_obj * time_step;
y_dis_obj = y_dis_obj + 0.5 * y_vel_obj * time_step;
}

/*******************************************************************************/
*                           — plot_path —
*                           
* This function plots the displacement of the object on the screen. The values of
* locx and locy are automatically converted to integer type by multiplying by an integer.
* 
* scrn_height/scrn_width ensures the x and y distances appear equal on the screen.
* 
*******************************************************************************/

void plot_path(void) {
  locx = 320 + (scrn_height / scrn_width) * (x_dis_obj * 640 / plot_width);
  locy = 240 - (y_dis_obj * 480 / plot_width);

  _lineto (locx, locy);

  _settextposition (1,7);
  printf("%6.3f",t);
}

/*******************************************************************************/
*                           — plot_plate —
*                           
* This function plots the position of the plate on the screen
*                           
*******************************************************************************/

void plot_plate(void) {
  col = _getcolor();
  _setcolor(0);
  _setpixel(temp_locx, temp_locy);
  _setcolor(2);
  _setpixel(320, temp_hy);
  _setcolor(6);
  locy = 240 - (av * cos(w * t) * 240 / plot_width);
  temp_hy = locy;
  _setpixel(320, locy);
  _setcolor(5);
  locx = 320 + (scrn_height / scrn_width) * (sin(w * t) * r * 320 / plot_width);
  locy = 240 - (r * cos(w * t) * 240 / plot_width);
  temp_locx = locx;
  temp_locy = locy;
  _setpixel(locx, locy);
  _setcolor(col);
}

/*******************************************************************************/
*                           — finish —
*                           

void finish(void){
    _setvideomode(_DEFAULTMODE);
}

C.2 Results

Presented in this section are graphs of vertical amplitude versus velocity and vertical amplitude versus direction of movement for various orbital frequencies. The values were obtained by running the simulation software. The results obtained form a close correlation with those obtained experimentally, for those values corresponding to the PAM itself. The trends, based on the frequency of orbit in particular have a good likeness to the real values.

From this work, it was seen that higher orbital frequencies for a fixed orbital radius lead to directional instability. Thus the expected operating frequency of such a PAM would be around 20 to 25 Hertz. The velocity of the object across the plate is controlled by manipulating the radius of the orbit, rather than the orbital frequency. The simulation also reveals that there is little effect from the coefficient of static friction on the speed and direction.
Appendix C

Vertical Amplitude Vs Velocity (r=0.75, u=0.25)

![Graph showing vertical amplitude vs velocity with different frequencies.]

Figure C-1 Simulated Vertical Amplitude Vs Velocity

Vertical Amplitude Vs Angle (r=0.75, u=0.25)

![Graph showing vertical amplitude vs angle with different frequencies.]

Figure C-2 Simulated Vertical Amplitude Vs Angle of Translation
Vertical Amplitude Vs Velocity ($r=0.75$, $u=0.35$)

![Graph](image)

Figure C-3 Simulated Vertical Amplitude Vs Velocity

Vertical Amplitude Vs Angle ($r=0.75$, $u=0.35$)

![Graph](image)

Figure C-4 Simulated Vertical Amplitude Vs Angle of Translation
Appendix C

Vertical Amplitude Vs Velocity ($r=0.75$, $u=0.5$)

![Graph showing vertical amplitude vs velocity with lines for 10Hz, 20Hz, and 30Hz.]

Figure C-5 Simulated Vertical Amplitude Vs Velocity

Vertical Amplitude Vs Angle ($r=0.75$, $u=0.5$)

![Graph showing vertical amplitude vs angle with lines for 10Hz, 20Hz, and 30Hz.]

Figure C-6 Simulated Vertical Amplitude Vs Angle of Translation
Appendix C

Vertical Amplitude Vs Velocity (r=1.5, u=0.35)

Figure C-7 Simulated Vertical Amplitude Vs Velocity

Vertical Amplitude Vs Angle (r=1.5, u=0.35)

Figure C-8 Simulated Vertical Amplitude Vs Angle of Translation
Appendix C

Vertical Amplitude Vs Velocity (r=3.0, u=0.35)

Figure C-9 Simulated Vertical Amplitude Vs Velocity

Vertical Amplitude Vs Angle (r=3.0, u=0.35)

Figure C-10 Simulated Vertical Amplitude Vs Angle of Translation
APPENDIX D

CONTROL SYSTEM SIMULATION

D.1 Source Code

/*----------------------------------------SIM7.H----------------------------------------*/
/* SIM7.H contains the global definitions used by SIM7.C */
/*--------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <dos.h>
#include <bios.h>
#include <ctype.h>
#include <math.h>
#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
#define BROWN 6
#define WHITE 7
#define DARK_GREY 8
#define LIGHT_BLUE 9
#define LIGHT_GREEN 10
#define LIGHT_CYAN 11
#define LIGHT_RED 12
#define LIGHT_MAGENTA 13
#define LIGHT_BROWN 14
#define LIGHT_WHITE 15
#define MAX_OBJECTS 2
#define INITIAL_STOCK 5
int u,v,w,x,y,z; /* global work variables, for indexing etc */
int item; /* index for arrayed stock data*/
char buffer[80]; /* buffer for output character function using _outtext()*/
char *alphabet[26]= /* global work variables, for indexing etc */
    /* index for arrayed stock data*/
    /* buffer for output character function using _outtext()*/
    ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q",
    "R","S","T","U","V","W","X","Y","Z"];

int customer_x, customer_y; /* array to identify the defined customer (one only) */
int customer_line; /* line number of customer array */
int step=0; /* step counter for the number of iterations 
             * the main control programme has executed */

int hit0, hit1, hit2; /* counter if neighbouring array has item in stock*/
int choice; /* dud to check any result of a function call*/
int result; /* used to display stock taken off customer */
int stock[5];
int next_cell[25][2]= /* dud to check any result of a function call*/
    [4,4,4,3,3,4,2,4,3,4,2,1,4,2,3,3,2,4,1,0,4,1,3,2,2,3,1,4,0,0,3,
    1,2,2,1,3,0,0,2,1,1,2,0,0,1,1,0,0,0];

typedef enum {false, true} boolean;

struct element
{
    boolean north_nbor; /* 1 when an item is on order */
    boolean south_nbor; /* 1 when array is a supplier */
    boolean east_nbor; /* 1 when array is a customer */
    boolean west_nbor; /* 1 when array is a source for an object */
    boolean functional;
    boolean object;
    boolean order; /* 1 when an item is on order */
    boolean supplier; /* 1 when array is a supplier */
    boolean customer; /* 1 when array is a customer */
    boolean source; /* 1 when array is a source for an object */

    int source_item; /* item the array is a source for, if it is a source */
    int stock_list[5]; /* eg --> stock_list[3] contains the number of item 3's */
    /* the element has */
    int order_list[5]; /* ditto */
    int ordered_list[5]; /* 1 when item already ordered*/
    int new_stock[5]; /* items to be ordered after end of scan */
};

struct element pam[5][5];
struct dostime_t dtime;
div_t divres;
/*----------------------------------------SIM7.C----------------------------------------*/

* SIM7.C is a new approach to the problem.
* *----------------------------------------*/

#include "sim7.h"
main()
{
   int result;
   int key;
   int not_fInished=1; /*loop test for space bar hit*/
   int keep_going=1; /*loop test for main loop*/

   _clearscreen(_GCLEARSCREEN);
   init_pam();
   result=identify_customer();
   draw_screen();
   update_step_display();
   update_display();

   if(!result){
      printf("Error, no customer defined, programme halted...
");
   exit(0);
   }

   _dos_gettime(&dtime);
   srand(dtime.hsecond);

   while(keep_going){
      while(not_fInished){
         while(!kbhitO){
            iterate_process();
            result=check_empty();
            if(result==0)
               break;
            step++;
         }

         not_fInished=0;
      }
      _settextposition(30,3);
      printf("Programme stopped");
      keep_going=0;
   }
   _setvideomode(_DEFAULTMODE);
}

/**************************************************************************************/
*  --- iterate_process ---
* * This is the primary function of the program. It will loop around testing
* all the PAM arrays for orders/stock etc. It starts at the bottom LH
corner and works backwards. This works OK for now because this is
the location of the customer, however this would not be so fine if it was
located elsewhere.

iterate_process()
{
    int loop;
    if(step==1)
        send_order();
        update_display();
    }
else{
    for(loop=0; loop<25; loop++)
        x=next_cell[loop][0];
        y=next_cell[loop][1];
        if(pam[x][y].order & & (pam[x][y].functional==true))
            for(z=0; z<5; z++)
                if(pam[x][y].source_item != z)
                    /* three variables which increment upon successfully
                    * finding a neighbour with stock for the queried
                    * array's order list */
                    hit0=0;
                    hit1=0;
                    hit2=0;
                    w=0;
                    if((pam[x][y].north_nbor=true)
                        & & (pam[x][y].order_list[z] >0)
                        & & (pam[x][y].stock_list[z] < MAX_OBJECTS)
                        & & (pam[x+1][y].stock_list[z] >0)
                        & & (pam[x+1][y].customer==false))
                        hit0++;
                        if((pam[x][y].west_nbor=true)
                            & & (pam[x][y].order_list[z] >0)
                            & & (pam[x][y].stock_list[z] < MAX_OBJECTS)
                            & & (pam[x-1][y].stock_list[z] >0)
                            & & (pam[x-1][y].customer==false))
                            hit1++;
                            if((pam[x][y].south_nbor=true)
                                & & (pam[x][y].order_list[z] >0)
                                & & (pam[x][y].stock_list[z] < MAX_OBJECTS)
                                & & (pam[x+1][y].stock_list[z] >0)
                                & & (pam[x+1][y].customer==false))
                                hit2++;
                                if(hit0==1 & & hit1==1 & & hit2==1){ /* all 3 neighbours have stock
                                    * so choose supplier */
                                    choice=flip30; /* choose supplier */
                                    if(choice==0) /* north */
                                        get_from_north(x,y,z);
                                        if(choice==1) /* west */
                                            get_from_west(x,y,z);
if(choice=2) /* south */
    get_from_south(x,y,z);
}

if(hit0==1 && hit1==1 && hit2==0){ /* n & w neighbour..*/
    choice=flip20; /* choose supplier */
    if(choice==0) /* north */
        get_from_north(x,y,z);
    if(choice==1) /* west */
        get_from_west(x,y,z);
}

if(hit0==0 && hit1==1 && hit2==1){ /* s & w neighbour..*/
    choice=flip20;
    if(choice==0) /* south */
        get_from_south(x,y,z);
    if(choice==1) /* west */
        get_from_west(x,y,z);
}

if(hit0==1 && hit1==0 && hit2==1){ /* n & s neighbour..*/
    choice=flip20;
    if(choice==0) /* north */
        get_from_north(x,y,z);
    if(choice==1) /* south */
        get_from_south(x,y,z);
}

if(hit0==1 && hit1==0 && hit2==0) /* n neighbour..*/
    get_from_north(x,y,z);

if(hit0==0 && hit1==1 && hit2==0) /* w neighbour..*/
    get_from_west(x,y,z);

if(hit0==0 && hit1==0 && hit2==1) /* s neighbour..*/
    get_from_south(x,y,z);

if(hit0==0 && hit1==0 && hit2==0){ /* no luck..place orders */
    if(pam[x][y].north_nbor==true) { /* order with north */
        if(pam[x-1][y].customer==false &&
            pam[x-1][y].functional==true) {
            if(pam[x-1][y].source_item != z) {
                if(pam[x][y].ordered_list[z]==false &&
                    pam[x][y].stock_list[z]==0) {
                    pam[x-1][y].order_list[z]=H-;
                    pam[x-1][y].order=true;
                }
            }
        }
    }
}
if(pam[x][y].west_nbor=true){ /* order with west */
    if(pam[x][y-1].customer=false
        && pam[x][y-1].functional=true)
        if(pam[x][y-1].source_item != z){
            if(pam[x][y].ordered_list[z]==false
                && pam[x][y].stock_list[z]==0){
                pam[x][y-1].order_list[z]++;
                pam[x][y-1].order=true;
                w++;
            }
        }
    }
    if(pam[x][y].south_nbor=true){ /* order with south */
        if(pam[x+1][y].customer=false
            && pam[x+1][y].functional=true){
            if(pam[x+1][y].source_item != z){
                if(pam[x][y].ordered_list[z]==false
                    && pam[x][y].stock_list[z]==0){
                    pam[x+1][y].order_list[z]++;
                    pam[x+1][y].order=true;
                    w++;
                }
        }
    }
    if(w>0)
        pam[x][y].ordered_list[z]=true;
}
Appendix D

212

--- clear_customer ---

* This function clears all items from the customer array
* stock list and updates another array that is used to display
* how much stock has been received by the customer.
*

```c
void clear_customer()
{
    int x = customer_x;
    int y = customer_y;
    for (int z = 0; z < 5; z++) {
        pam[x][y].stock_list[z] = 0;
        stock[z]++;
    }
    draw_objects(x, y);
}
```

--- get_from_north ---

* Function is used to get a stock item from the north neighbour.
* Input is the array address of the receiving module.
* The function massages global variables, so no return value.
*
```
void get_from_north(int row, int col, int item)
{
    if (pam[row][col].stock_list[item] < 9) {
        pam[row][col].new_stock[item]++;
        pam[row][col].order_list[item]--;
        pam[row][col].ordered_list[item] = false;

        /*if (pam[row-1][col].supplier==false)* /

        pam[row-1][col].stock_list[item]--;
        move_object(row-1, col, 2, item);
        draw_grid();
    }
}
```

--- get_from_west ---

* Function is used to get a stock item from the west neighbour.
* Input is the array address of the receiving module.
* The function massages global variables, so no return value.
*
```
void get_from_west(int row, int col, int item)
{
    if (pam[row][col].stock_list[item] < 9) {
```
pam[row][col].new_stock[item]++;  
pam[row][col].order_list[item]--;  
pam[row][col].ordered_list[item]=false;

/*if(pam[row][col-1].supplier==false)*/  
pam[row][col-1].stock_list[item]--;  
move_object(row, col-1, 1, item);  
draw_grid();
}

/***********************************************************/
* --- get_from_south ---
* Function is used to get a stock item from the south neighbour.
* Input is the array address of the receiving module.
* The function massages global variables, so no return value.
* 
/***********************************************************/

get_from_south(int row, int col, int item)
{
   if(pam[row][col].stock_list[item]<9) {
      pam[row][col].new_stock[item]++;  
pam[row][col].order_list[item]--;  
pam[row][col].ordered_list[item]=false;

      /*if(pam[row+1][col].supplier==false)*/  
pam[row+1][col].stock_list[item]--;  
move_object(row+1, col, 3, item);  
draw_grid();
   }
}

/***********************************************************/
* --- check_order_lists ---
* This function checks all arrays for items on order. If an
* array has nothing on order, the order element is reset.
* 
/***********************************************************/

check_order_lists()
{
   for(x=0; x<5; x++) {
      for(y=0; y<5; y++) {
         if(pam[x][y].order_list[0]==0 && pam[x][y].order_list[1]==0 &&  
pam[x][y].order_list[2]==0 && pam[x][y].order_list[3]==0 &&  
pam[x][y].order_list[4]==0  
         pam[x][y].order=false;
      }
   }
}
/******************************************k
**
* — send_order —
* Sends and order for items into the system and sets the
* customer order bit.
*
*/

send_order()
{
x=customer_x;
y=customer_y;

for(z=0; z<5; z++)
{
    pam[x][y].order_list[z]=1;
    pam[x][y].ordered_list[z]=0;
}
pam[x][y].order=true;
}

/******************************************k
**
* — int flip3 —
* Choose between 1 of 3 choices.
* Return value is the choice to be made.
* 0=choice 0, 1=choice 1, 2=choice 2
*
*/

int flip3()
{
    /*_dos_gettime(&dtime);
    divres = div(dtime.hsecond,3);*/

    /*srand(step);*/
    divres = div(rand0,3);
    if(divres.rem==0)
        return(0);
    if(divres.rem==1)
        return(1);
    else
        return(2);
}

/******************************************k
**
* — int flip2 —
* Choose between 1 of 2 choices.
* Return value is the choice to be made.
* 0=choice 0, 1=choice 1.
*
*/

int flip2()
{
    /*_dos_gettime(&dtime);
    divres = div(dtime.hsecond,2);*/
/*srand(step);*/
divres = div(rand(),2);
if(divres.rem==0)
    return(0);
else
    return(1);

--- int check_satisfied ---
* Checks if the customer has received one of each item.
* If it has, return value 1 else 0.

int check_satisfied()
{
    int empties;
    empties=0;
    for(x=0; x<5; x++){
        for(y=0; y<5; y++){
            for(z=0; z<5; z++){
                empties+=pam[x][y].stock_list[z];
            }
        }
    }
    return empties;
}

update_stock_list()
{
    for(x=0; x<5; x++)
        for(y=0; y<5; y++)
for(z=0; z<5; z++){
    pam[x][y].stock_list[z] += pam[x][y].new_stock[z];
    pam[x][y].new_stock[z]=0;
}

update_display()
{
    for(x=0; x<5; x++){
        for(y=0; y<5; y++){
            draw_objects(x,y);
        }
    }
}

update_step_display()
{
    _settextposition(30,14);
    printf("%d",step);
    _settextposition(30,30);
    printf("Stock Removed %d %d %d %d %d",
            stock[0],stock[1],stock[2],stock[3],stock[4]);
    /*pam[3][4].stock_list[0],pam[3][4].stock_list[1],
    pam[3][4].stock_list[2],pam[3][4].stock_list[3],
    pam[3][4].stock_list[4]);*/
}

move_object(int x, int y, int direction, int object)
{
    int x_offset;
    int y_offset;
    short colour;
    int x_start;
    int y_start;
    int x_inc;
    int y_inc;
    int box=12;

    switch(object){
    case(0):
        colour=WHITE;
        y_start=74;
        x_start=14;
        break;

    case(1):
        colour=LIGHT_BLUE;
        y_start=54;
        x_start=24;
        break;

    case(2):
        colour=RED;
y_start=44;
x_start=44;
break;

case(3):
colour=DARK_GREY;
y_start=34;
x_start=64;
break;

case(4):
colour=GREEN;
y_start=14;
x_start=74;
break;

default:
colour=BLUE;
break;
}

if(direction==1){
x_offset=0;
x_inc=0;
y_inc=2;
for(y_offset=0; y_offset<80; y_offset+=y_inc){
    _setcolor(BLUE);
    _ellipse(_GFILLINTERIOR,
        y*80+y_start+y_offset,x*80+x_start+x_offset,
        y*80+y_start+box+y_offset,x*80+x_start+box+x_offset);
    _setcolor(colour);
    _ellipse(_GFILLINTERIOR,
        y*80+y_start+y_offset+y_inc,x*80+x_start+x_offset+x_inc,
        y*80+y_start+box+y_offset+y_inc,x*80+x_start+box+x_offset+x_inc);
    if(pam[x][y].stock_list[object]>0)
        _ellipse(_GFILLINTERIOR,
            y*80+y_start,x*80+x_start,
            y*80+y_start+box,x*80+x_start+box);
}
}

if(direction==2){
y_offset=0;
x_inc=2;
y_inc=0;
for(x_offset=0; x_offset<80; x_offset+=x_inc){
    _setcolor(BLUE);
    _ellipse(_GFILLINTERIOR,
        y*80+y_start+y_offset,x*80+x_start+x_offset,
        y*80+y_start+box+y_offset,x*80+x_start+box+x_offset);
    _setcolor(colour);
Appendix D

if(pam[x][y].stock_list[object]>0)
    _ellipse(_GFiLLINTERIOR,
        y*80+y_start+y_offset+y_inc,x*80+x_start+x_offset+x_inc,
        y*80+y_start+box+y_offset+y_inc,x*80+x_start+box+x_offset+x_inc);
    
    if(pam[x][y].stock_list[object]>0)
        _ellipse(_GFiLLINTERIOR, 
            y*80+y_start,x*80+x_start, 
            y*80+y_start+box,x*80+x_start+box);
    
if(direction==3) {
    y_offset=0;
    x_inc=-2;
    y_inc=0;
    for(x_offset=0; (abs(x_offset)<80); x_offset+=x_inc){
        _setcolor(BLUE);
        _ellipse(_GFiLLINTERIOR,
            y*80+y_start+y_offset,y*80+x_start+x_offset,
            y*80+y_start+box+y_offset,y*80+x_start+box+x_offset);
        
        _setcolor(colour);
        _ellipse(_GFiLLINTERIOR,
            y*80+y_start+y_offset+y_inc,x*80+x_start+x_offset+\inc,
            y*80+y_start+box+y_offset+y_inc,x*80+x_start+\box+x_offset+\inc);
    
    if(pam[x][y].stock_list[object]>0)
        _ellipse(_GFiLLINTERIOR, 
            y*80+y_start,x*80+x_start, 
            y*80+y_start+box,x*80+x_start+box);
    
    if(pam[x][y].stock_list[object]>0)
        _ellipse(_GFiLLINTERIOR, 
            y*80+y_start,x*80+x_start, 
            y*80+y_start+box,x*80+x_start+box);
        
        _setcolor(WHITE);
        draw_grid();
    
    draw_objects(int x, int y)
    
    if(pam[x][y].stock_list[0]){
        _setcolor(BLUE);
        _ellipse(_GFiLLINTERIOR,y*80+74,x*80+14,y*80+86,x*80+26);
        _setcolor(WHITE);
        _ellipse(_GFiLLINTERIOR,y*80+74,x*80+14,y*80+86,x*80+26);
    } else{
        _setcolor(BLUE);
        _ellipse(_GFiLLINTERIOR,y*80+74,x*80+14,y*80+86,x*80+26);
    }
customer
 customer:
 )
) ( Pam[n][V] [Customer = true]
 ) ( + + + v c:
 ) ( + + + n > 0
 ) ( + + + n > 0
 )

 } in indentity - customer
 )

 )

 )

 )

 else

 )

 )

 )

 )

 else

 )

 )

 }
return(true);
}
}
return(false);
}
draw_screen(void)
{
if(_setvideomode(_VRES16COLOR)=-1){
    printf("Error, Video mode is not available\n");
    _setvideomode(_DEFAULTMODE);
    return;
}
_setbkcolor(_BLUE);
_setcolor(WHITE);
_settextcolor(WHITE);
draw_grid();
}
draw_grid()
{
    int x;

    for(x=10; x<411; x+=80){
        _moveto(10,x);
        _lineto(410,x);
        _moveto(x,10);
        _lineto(x,410);
    }
}
init_pam(void)
{
    pam[0][0].north_nbor=false;
    pam[0][0].south_nbor=true;
    pam[0][0].east_nbor=true;
    pam[0][0].west_nbor=false;
    pam[0][0].functional=true;
    pam[0][0].object=true;
    pam[0][0].order=false;
    pam[0][0].supplier=true;
    pam[0][0].customer=false;
    pam[0][0].source=true;

    pam[0][1].north_nbor=false;
    pam[0][1].south_nbor=false;
    pam[0][1].east_nbor=true;
    pam[0][1].west_nbor=true;
    pam[0][1].functional=true;
    pam[0][1].object=false;
    pam[0][1].order=false;
    pam[0][1].supplier=false;
    pam[0][1].customer=false;
    pam[0][1].source=false;
pam[1][1].customer=false;
pam[1][1].source=false;

pam[1][2].north_nbor=true;
pam[1][2].south_nbor=false;
pam[1][2].east_nbor=true;
pam[1][2].west_nbor=true;
pam[1][2].functional=true;
pam[1][2].object=false;
pam[1][2].order=false;
pam[1][2].supplier=false;
pam[1][2].customer=false;
pam[1][2].source=false;

pam[1][3].north_nbor=true;
pam[1][3].south_nbor=false;
pam[1][3].east_nbor=true;
pam[1][3].west_nbor=true;
pam[1][3].functional=true;
pam[1][3].object=false;
pam[1][3].order=false;
pam[1][3].supplier=false;
pam[1][3].customer=false;
pam[1][3].source=false;

pam[1][4].north_nbor=true;
pam[1][4].south_nbor=false;
pam[1][4].east_nbor=false;
pam[1][4].west_nbor=true;
pam[1][4].functional=true;
pam[1][4].object=false;
pam[1][4].order=false;
pam[1][4].supplier=false;
pam[1][4].customer=false;
pam[1][4].source=false;

pam[2][0].north_nbor=true;
pam[2][0].south_nbor=true;
pam[2][0].east_nbor=true;
pam[2][0].west_nbor=false;
pam[2][0].functional=true;
pam[2][0].object=true;
pam[2][0].order=false;
pam[2][0].supplier=true;
pam[2][0].customer=false;
pam[2][0].source=true;

pam[2][1].north_nbor=true;
pam[2][1].south_nbor=false;
pam[2][1].east_nbor=true;
pam[2][1].west_nbor=true;
pam[2][1].functional=true;
pam[2][1].object=false;
pam[2][1].order=false;
source=true
customer=false
supplier=false
order=false
object=true
functional=true
west-neighbor=false
east-neighbor=true
south-neighbor=false
north-neighbor=false

object=true
functional=true
west-neighbor=false
east-neighbor=true
south-neighbor=false
north-neighbor=false
pam[2][3].source=false;
pam[2][3].customer=false;
pam[2][3].supplier=false;
pam[2][3].order=false;
pam[2][3].object=false;
pam[2][3].functional=true;
pam[2][3].west-door=true;
pam[2][3].east-door=true;
pam[2][3].south-door=true;
pam[2][3].north-door=true;

pam[2][2].source=false;
pam[2][2].customer=false;
pam[2][2].supplier=false;
pam[2][2].order=false;
pam[2][2].object=false;
pam[2][2].functional=true;
pam[2][2].west-door=true;
pam[2][2].east-door=true;
pam[2][2].south-door=true;
pam[2][2].north-door=true;

pam[2][1].source=false;
pam[2][1].customer=false;
pam[2][1].supplier=false;
pam[3][1].order=false;
pam[3][1].supplier=false;
pam[3][1].customer=false;
pam[3][1].source=false;

pam[3][2].north_nbor=true;
pam[3][2].south_nbor=false;
pam[3][2].east_nbor=true;
pam[3][2].west_nbor=true;
pam[3][2].functional=true;
pam[3][2].object=false;
pam[3][2].order=false;
pam[3][2].supplier=false;
pam[3][2].customer=false;
pam[3][2].source=false;

pam[3][3].north_nbor=true;
pam[3][3].south_nbor=false;
pam[3][3].east_nbor=true;
pam[3][3].west_nbor=true;
pam[3][3].functional=true;
pam[3][3].object=false;
pam[3][3].order=false;
pam[3][3].supplier=false;
pam[3][3].customer=false;
pam[3][3].source=false;
pam[3][4].north_nbor=true;
pam[3][4].south_nbor=false;
pam[3][4].east_nbor=false;
pam[3][4].west_nbor=true;
pam[3][4].functional=true;
pam[3][4].object=false;
pam[3][4].order=false;
pam[3][4].supplier=false;
pam[3][4].customer=false;
pam[3][4].source=false;

pam[4][0].north_nbor=true;
pam[4][0].south_nbor=false;
pam[4][0].east_nbor=true;
pam[4][0].west_nbor=false;
pam[4][0].functional=true;
pam[4][0].object=true;
pam[4][0].order=false;
pam[4][0].supplier=true;
pam[4][0].customer=false;
pam[4][0].source=true;

pam[4][1].north_nbor=true;
pam[4][1].south_nbor=false;
pam[4][1].east_nbor=true;
pam[4][1].west_nbor=true;
pam[4][1].functional=true;
pam[4][1].object=false;
pam[4][1].order=false;
pam[4][1].supplier=false;
pam[4][1].customer=false;
pam[4][1].source=false;

pam[4][2].north_nbor=true;
pam[4][2].south_nbor=false;
pam[4][2].east_nbor=true;
pam[4][2].west_nbor=true;
pam[4][2].functional=true;
pam[4][2].object=false;
pam[4][2].order=false;
pam[4][2].supplier=false;
pam[4][2].customer=false;
pam[4][2].source=false;

pam[4][3].north_nbor=true;
pam[4][3].south_nbor=false;
pam[4][3].east_nbor=true;
pam[4][3].west_nbor=true;
pam[4][3].functional=true;
pam[4][3].object=false;
pam[4][3].order=false;
pam[4][3].supplier=false;
pam[4][3].customer=false;
pam[4][3].source=false;

pam[4][4].north_nbor=true;
pam[4][4].south_nbor=false;
pam[4][4].east_nbor=false;
pam[4][4].west_nbor=true;
pam[4][4].functional=true;
pam[4][4].object=false;
pam[4][4].order=false;
pam[4][4].supplier=false;
pam[4][4].customer=true;
pam[4][4].source=false;
for(u=0; u<5; u++){
    for(v=0; v<5; v++){
        for(w=0; w<5; w++){
            pam[u][v].stock_list[w]=0;
            pam[u][v].order_list[w]=0;
            pam[u][v].ordered_list[w]=0;
            pam[u][v].new_stock[w]=0;
            if(pam[u][v].source)
                pam[u][v].source_item=u;
            else
                pam[u][v].source_item=-1;

        }
    }
}

for(u=0; u<5; u++){
    pam[u][0].stock_list[u]=INITIAL_STOCK;
}
}
APPENDIX E

BOUNDARY TRACING SOURCE CODE

E.1 Source Code

```c
#include <dos.h>
#define TRUE 1
#define FALSE 0

int int boundary_trace(char *frame, int r, int c, int threshold, int *boundary_array,
                        int max_points, int start_x, int start_y, int end_x, int end_y,
                        int mark_boundary, int *xavg, int *yavg)
{
    int root_pixelx=start_x;
    int root_pixely=start_y;
    int next_search_element[8];
    int associate[8];
    int row = 0;
    int column = 0;
    int next_pixel_found;
    int loop;
    int index;
    int loop1;
    int first_index;
    int second_index;
    int count;
```
int not_end_of_track = TRUE;
int no_object_flag;
int num_boundary_points = 0;
int boundary_ptr = 0;
int row_accumulate = 0;
int col_accumulate = 0;
int result;

no_object_flag = find_root_pixel(frame, r, c, threshold, &root_pixelx, &root_pixely, end_x, end_y);

if (no_object_flag == 1) {
    calc_associate_address(associate, root_pixelx, root_pixely, frame, r, c, start_x, start_y, end_x, end_y);
    num_boundary_points = 0;
    boundary_array[boundary_ptr++] = root_pixelx;
    boundary_array[boundary_ptr++] = root_pixely;
    next_search_element[0] = 1;
    for (loop = 1; loop <= 7; loop++)
        next_search_element[loop] = 0;
    row = root_pixelx;
    column = root_pixely;
    second_index = 0;
    while (not_end_of_track) {
        next_pixel_found = FALSE;
        loop = TRUE;
        index = second_index;
        while (loop) {
            if (next_search_element[index] == 1) {
                loop1 = TRUE;
                first_index = index;
                count = 0;
                while ((loop1 == TRUE) && (count < 8)) {
                    change_direction(&first_index, second_index, 2);
                    if (associate[first_index] > threshold) {
                        result = new_position(first_index, &row, &column);
                        result = calc_associate_address(associate, row, column, frame, r, c, start_x, start_y, end_x, end_y);
                        next_search_element[index] = 0;
                        next_search_element[second_index] = 1;
                        loop1 = FALSE;
                    }
                    first_index++;
                    count++;
                }
            }
            if (count == 8)
                next_pixel_found = FALSE;
            else
                next_pixel_found = TRUE;
            loop = FALSE;
        }
        index++;
    }
    if (next_pixel_found) {
if((row != root_pixelx) || (column != root_pixely))
    if(mark_boundary)
        frame[(row*c)+column]=255;
        row_accumulate+=row;
        col_accumulate+=column;
        num_boundary_points++;
        boundary_array[boundary_ptr++] = row;
        boundary_array[boundary_ptr++] = column;
    }
else
    not_end_of_track = FALSE; /* signify back at root pixel */
}
else
    not_end_of_track = FALSE; /* Unable to find another pixel */

if(num_boundary_points >= max_points) /* stop array overrun */
    not_end_of_track = FALSE;
}

if (row == root_pixelx || column == root_pixely)
    if(mark_boundary)
        frame[(row*c)+column]=255;
        row_accumulate+=row;
        col_accumulate+=column;
        num_boundary_points++;
        boundary_array[boundary_ptr++] = row;
        boundary_array[boundary_ptr++] = column;
    }

/* no_object_flag */

if(no_object_flag)
    {
        *xavg=row_accumulate/num_boundary_points;
        *yavg=col_accumulate/num_boundary_points;
        return(num_boundary_points);
    }
else
    return(0);
}

/* int find_root_pixel */

int find_root_pixel( char *frame, int r, int c, int threshold, int *startx, int *starty, int endx, int endy)
Appendix E

```c
int root_found = 0;
int ptr;
int x;
int y;

for(x=*startx; x<endx; x++){
    for(y=*starty; y<endy; y++){
        ptr=x*c+y;
        root_found=((frame[ptr]>threshold) && (frame[ptr+1]>threshold));
        if(root_found){
            *startx=ptr/c;
            *starty=ptr-(*startx*c);
            return(root_found);
        }
    }
}
return(root_found);

/*
 * int calc_associate_address
 */

int calc_associate_address( int *associate, int row, int column, char *frame, int r, int c,
        int startx, int starty, int endx, int endy)
{
    int pixel_row, previous_pixel_row, next_pixel_row;
    int pixel_column, previous_pixel_column, next_pixel_column;

    pixel_row = (row * c);
    previous_pixel_row = pixel_row - c;
    next_pixel_row = pixel_row + c;
    pixel_column = column;
    previous_pixel_column = pixel_column - 1;
    next_pixel_column = pixel_column + 1;
    associate[0] = (int) frame[previous_pixel_row + pixel_column];
    associate[1] = (int) frame[previous_pixel_row + next_pixel_column];
    associate[2] = (int) frame[pixel_row + next_pixel_column];
    associate[3] = (int) frame[next_pixel_row + next_pixel_column];
    associate[4] = (int) frame[next_pixel_row + previous_pixel_column];
    associate[5] = (int) frame[pixel_row + previous_pixel_column];
    associate[6] = (int) frame[next_pixel_row + previous_pixel_column];
    associate[7] = (int) frame[previous_pixel_row + previous_pixel_column];

    if(row<startx){
        associate[0] = 0;
        associate[1] = 0;
        associate[2] = 0;
    }
    if(row>endx){
        associate[3] = 0;
    }
}
```
associate[4] = 0;
associate[5] = 0;
}

if(column<starty){
    associate[5] = 0;
    associate[6] = 0;
    associate[7] = 0;
}

if(column>endy){
    associate[1] = 0;
    associate[2] = 0;
    associate[3] = 0;
}
return(1);
}

int change_direction(int *first_index, int *second_index, int diff_factor )
{
    int index ;

    index = *first_index ;
    switch ( index )
    {
    case (0):
        *first_index = index;
        *second_index = 6;
        break;
    case (1):
        *first_index = index;
        *second_index = 7;
        break;
    case (8):
        *first_index = 0;
        *second_index = 6 ;
        break;
    default:
        *first_index = index ;
        *second_index = index-diff_factor;
        break ;
    }
    return(1);
}

/*-----------------------------
 * int new_position
 */

/* -----------------------------
 * --- int change_direction ---
 */
int new_position(int pixel_direction, int *row, int *column)
{
    switch (pixel_direction) {
    case (O):
        --(*row);
        break;
    case (1):
        --(*row);
        ++(*column);
        break;
    case (2):
        ++(*column);
        break;
    case (3):
        ++(*row);
        ++(*column);
        break;
    case (4):
        ++(*row);
        break;
    case (5):
        ++(*row);
        --(*column);
        break;
    case (6):
        --(*column);
        break;
    case (7):
        --(*row);
        --(*column);
        break;
    }
    return (1);
}