NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

VISION-BASED PARTS FETCHING FOR A ROBOTIC ASSEMBLY CELL

by

J.P. Thomas GOGUEN

A Thesis

submitted to the School of Graduate Studies

in partial fulfillment of the requirements for the

DEGREE of MASTER of APPLIED SCIENCES

in

ELECTRICAL ENGINEERING

at the

OTTAWA-CARLETON INSTITUTE FOR ELECTRICAL ENGINEERING

May, 1988

ISBN 0-315-53775-2

ABSTRACT

This thesis concentrates upon the design and implementation of a vision-based parts presentation system with the ultimate goal of integrating the system with an assembly processor using the interface commands defined herein. Primary consideration is given to the design and construction of near real-time sensing abilities with specific concentration upon a machine vision system supported by presence and position sensors. The thesis is also concerned with the interpretation of the data gleaned from these sensors as well as its integration into a set of coherent positional parameters referenced to the assembly processor thus enabling tracking and grasping of the part.

ACKNOWLEDGEMENT

I would like to thank Dr. Emil M. Petriu for his time and patience during my tenure with the University of Ottawa. I will always cherish his friendship and will continue to be inspired by his consummate interest in every aspect of the engineering profession.

I would also like to acknowledge the friendship of the other members of our small research group who without knowing have taught me many important lessons about the people of the world we share.

Finally I would like to thank my family, for without their support, none of this would have been possible. Dad, Mom, Jill and Marc, I thank you.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

From an historical perspective self-contained automated machines have existed for over two-hundred years. From these so-called sequence machines the robot as a "reprogrammable, multi-functional manipulator designed to move material, parts, tools, or specialized devices through variable programmable motions for the performance of a variety of tasks"[1] has evolved. The robots of the past utilised little or no sensory feedback to control their movements. Today, a robot is no longer considered an independent stand-alone device but rather is incorporated as part of a computing system designed to interact with its

---

[1] From the Robot Institute of America as quoted in Arthur J. Critchlow, Introduction to Robotics, MacMillan Publishing Company, New York, N.Y., 1985, pp.8.

environment. Robotic assembly is important as it provides lower cost, higher quality, and increased productivity for a variety of industries from agriculture to manufacturing.[1]

Complex robotic systems require new, more efficient, programming methods. Efforts are currently under way to develop task-level (object oriented) programming languages. This leads to a more structured approach to the robotic assembly process identifying two distinct sub-tasks: "part fetching" and "part mating". The former, parts fetching, refers to simply identifying and grasping the part and is in general controlled in position whereas the latter, part mating, involves joining the part to the assembly and is controlled using force feedback. These distinct sub-tasks make possible the independent design of each (which essentially contributes to the reliability and further improvement of the system as a whole). This thesis deals primarily with the first sub-task, that of parts fetching using vision feedback which provides a high degree of positional accuracy. The following chapter descriptions highlight the author's contributions.

Chapter 2 details the necessary parts of the operating system specification. It involves an explanation of the task-level language of a robotic assembly cell and discusses the necessary

---

[1] Arthur J. Critchlow, Introduction to Robotics, MacMillan Publishing Company, New York, N.Y., 1985, pp. 29-34.

hardware support.

The next chapter, Chapter 3, describes a conveyor based parts presentation system, which implements the vision-based parts fetching operation.

Chapter 4 is a theoretical analysis of the vision system proposing a variable resolution windowing technique for a progressive vision system improving the performance of the positional feedback and ultimately increasing the throughput of the whole system.

The architecture of the vision hardware is described in Chapter 5. Also detailed is the need for custom hardware in order to achieve near real-time performance.

Position recovery aspects for the conveyor and associated coordinate transformations are highlighted in Chapter 6. The need for accurate position information concerning the position of the part as it moves along the conveyor demands an absolute encoding mechanism for the conveyor belt. This chapter concludes with the description of the integration of the vision and other (presence and position) sensors for the real-time updating of the positional parameters of the part.

Finally in Chapter 7 the kinematic transformations for the Rhino

SCARA robot are derived along with the development of the path control algorithms required for tracking and grasping the part. This information and the use of this robot is the final step in the parts fetching process.

CHAPTER 2 OPERATING SYSTEM SPECIFICATION

2.1 INTRODUCTION

This chapter describes language specification and structural problems of a robotic assembly cell conceived to support a task-level robotic programming language. It delineates some syntactic details of the language and discusses the hardware support required for the implementation of the language. The chapter starts with a discussion of task level languages and continues with a description of the language syntax adopted and related task planner. Then a system model used as a basis for the design is discussed along with the specification of the appropriate parts of the task level language used. The chapter concludes by restructuring the system model into two distinct functional units, parts acquisition using multi-sensor feedback

and multi-sensor parts assembly, the former being the subject of this thesis and the latter the subject of a thesis currently being written by B. Karoui, a member of the same research team.

2.2 TASK-LEVEL PROGRAMMING FOR ROBOTIC ASSEMBLY

Programming languages for robotic applications typically fall into two categories[1]: robot oriented and object oriented (task level). The difference is that robot oriented programming considers the task to be performed in terms of a series of movements made by the robot arm. So-called task-level or object oriented programming specifies a task as a series of operations to be performed on the object. This section describes the task-level language definitions used to program the architecture of the remaining chapters.

The task-level language has the advantage of being easily adapted to emulate an industrial assembly flowchart as it consists of a series of primitive actions performed upon objects. These tasks are planned off-line using a task planner

---

1 T. Lozano-Perez, "Robot Programming," IEEE Proceedings, Vol. 71, no. 7, pp. 821-841, 1983.

architecture similar to that described by Gonzalez.[1] With reference to Figure 2.1, note that the task specification is divided into two sub-tasks, object fetching and object assembly. Planners at the sub-task level generate the control routines for the sequence of assembly operations to which the objects or parts are submitted. These include:

1. **Object Presentation-** Using a parts feeder and conveyor as well a presence and vision sensors the object (part), while moving on the conveyor, is examined for integrity and position.

2. **Object Tracking and Grasping-** Using the information provided in Step 1 a robot arm tracks the object and grasps it removing it from the conveyor.

3. **Object Orientation and Relative Positioning-** Using the positional information provided by the proprioceptors in combination with the preset information concerning the position and orientation demanded for the implementation of the object mating step, the robot arm affects the object's position and orientation as required by the object

---

[1] K.S. Fu, R.C. Gonzalez, and C.S.G. Lee, Robotics: Control, Sensing, Vision, and Intelligence, McGraw-Hill Inc., U.S.A., 1987.

destination parameters.

4. Object Mating- Using a robot arm and various sensors for feedback control of the mating process the object is joined to the assembly.

Though the language is not completely specified the design and construction of the hardware requires only one command definition, that of the assembly tasks. The syntax is as follows:

<OPERVERB><OBJNAME><CONTCTREL>[<MODIF>][<CONSTR>]

Where:

<OPERVERB>::= **SIMP_PEG_IN_HOLE | PUSH_TWIST | MULT_PEG_IN_HOLE | PEG_RETAIN | SCREW | FORCE_FIT | RMV_LOC_PIN | FLIP_OVER | PROV_TEMP_SUP | CRIP_SHT | RMV_TEMP_SUP | WELD_SOLD, corresponding to the 12 assembly primitives described by Nevins[1];**

<OBJNAME>::= **any of a list of legal object or part names as pre-defined in the object model in the task specification e.g., PEG1, PEG3, BOLT, WASHER10, BLOCK8;**

---

[1] J.L. Nevins and D.E. Whitney, "Computer-Controlled Assembly," Scientific American, pp. 62-74, February, 1978.

<CONTCTREL>::= <OBJSIDE><RELTYPE><ASSEMSIDE>;

<MODIF>::= GUARDED | FREE_MOVE | etc.;

<CONSTR>::= TORQTHR .EQ. n | FORCETHR .EQ. n | etc.;

<OBJSIDE>::= list of object sides as pre-defined in the object model in the task specification e.g., PEG_3_SIDE1;

<ASSEMSIDE>::= list of object sides acceptable to the object model where the object is already part of the assembly e.g., BLOCK_8_SIDE_4;

<RELTYPE>::= AGAINST | FIT | COPLANAR; similar to the contact relations described by Popplestone et al.[1];

As noted in the description of the syntax of the assembly command, models for the objects and their assembly must be available to the task planners. These models may be defined in a number of ways depending upon the operation to be performed

---

[1] R.J. Popplestone et al., "RAPT, A Language For Describing Assemblies," Industrial Robot, Vol. 5, no. 3, pp. 131-137, 1978.

upon the object (part) and possibly the other parts to be defined. The object model must provide enough information about the object's parameters so that the assembly to be performed is possible. For example the object model used to define a set of three parts using vision identification must provide for at least one visually discernable characteristic of the parts. This could be as simple as the area of each part (provided it was different for each) to the complexity of a template of the object which is used for a best fit matching process. The object model should generally specify in this case a pre-defined reference orientation for each part (thus the previous example is good for identifying circular objects only) so that the deviation from this orientation may be measured and used when the part is ultimately oriented for assembly. The object model is generally defined in a manner that relates enough information to uniquely identify the position and orientation of the part in the simplest manner possible in an effort to simplify the computational effort required.

It may also be noted that the method of task specification and decomposition involves two relatively unrelated operations, those of object presentation using a conveyor-based parts presentation system and object assembly involving a force/torque sensor controlled assembly station. In the worst case the two sub-tasks need only communicate a demand for an object or part and subsequently the position of that part. This division is

further expounded with an examination of the system model.

2.3 THE SYSTEM MODEL

The functional model of the robotic assembly system based upon the task oriented language specification of the previous section is shown in Figure 2.2. The diagram highlights the boundaries of the projects involved, those of PARTS PRESENTATION SYSTEM, ASSEMBLY PROCESSOR, and a global TASK SCHEDULER which communicates to the ASSEMBLY PROCESSOR using the assembly command syntax defined above. Note that the TASK SCHEDULER primarily controls the ASSEMBLY PROCESSOR which in turn communicates it needs to the PARTS PRESENTATION SYSTEM.

The system operates in the manner defined in the flowchart of Figure 2.3. When the ASSEMBLY PROCESSOR accepts an instruction from the TASK SCHEDULER it issues the REQ(OBJNAME) interface command signaling the PARTS PRESENTATION SYSTEM to deliver the required part. The PARTS PRESENTATION SYSTEM uses an internal (for the purposes of this document the parts feeders are considered part of the PARTS PRESENTATION SYSTEM) REQO(n) command to request the part from FEEDER #n (OBJNAME corresponds to the index n). FEEDER #n places the part in the range of the PARTS PRESENTATION SYSTEM. The PARTS PRESENTATION SYSTEM then

brings the object to within the reach of the robot arm and, using sensory data, computes the position and checks the integrity of the part and passes the position data to the ASSEMBLY PROCESSOR using the GNT(POSOBJ) command. The ASSEMBLY PROCESSOR then completes the task previously accepted. The ASSEMBLY PROCESSOR uses the ROBSTATE to synchronize its actions with the TASK SCHEDULER. Assembly operations since they are controlled using some sensory feedback, are asynchronous operations in that the time required to perform an operation is a stochastic function. ROBSTATE can signal the failure to grasp the object sent by the presentation system as well as notifying the system in the event of the failure of an assembly operation. The interface commands and a more in depth functional description of the operation of the PARTS PRESENTATION SYSTEM comprises the better part of the next chapter.

2.4 CONCLUSION

This brief overview of the task oriented language specification and the architecture upon which it is to be implemented is provided in order that the scope of this thesis may be defined. The next chapter outlines the Parts Presentation System design and highlights the manner in which it provides some of the functions required by the task-level language.

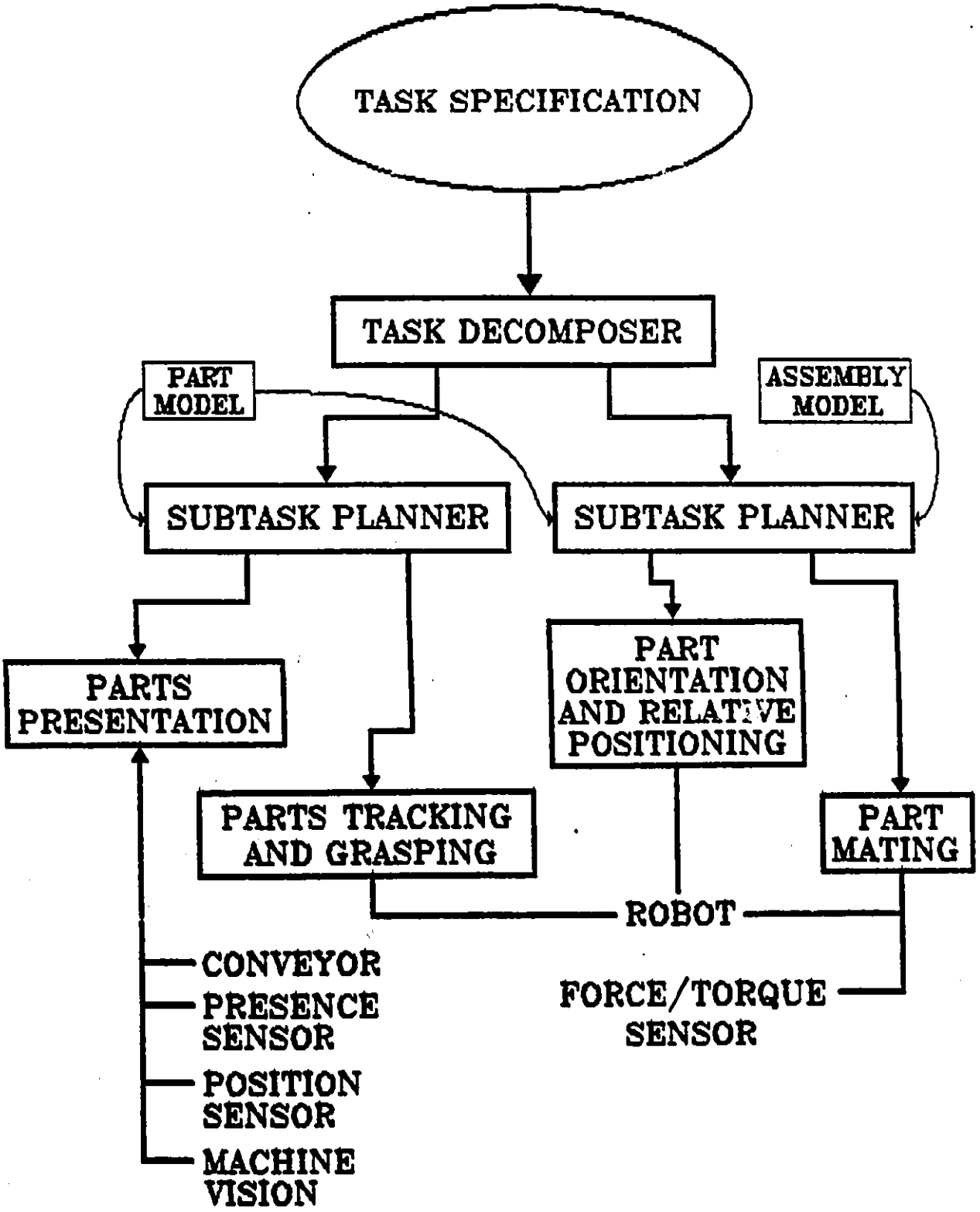


FIGURE 2.1: An illustration of the organization of the Task Planner.

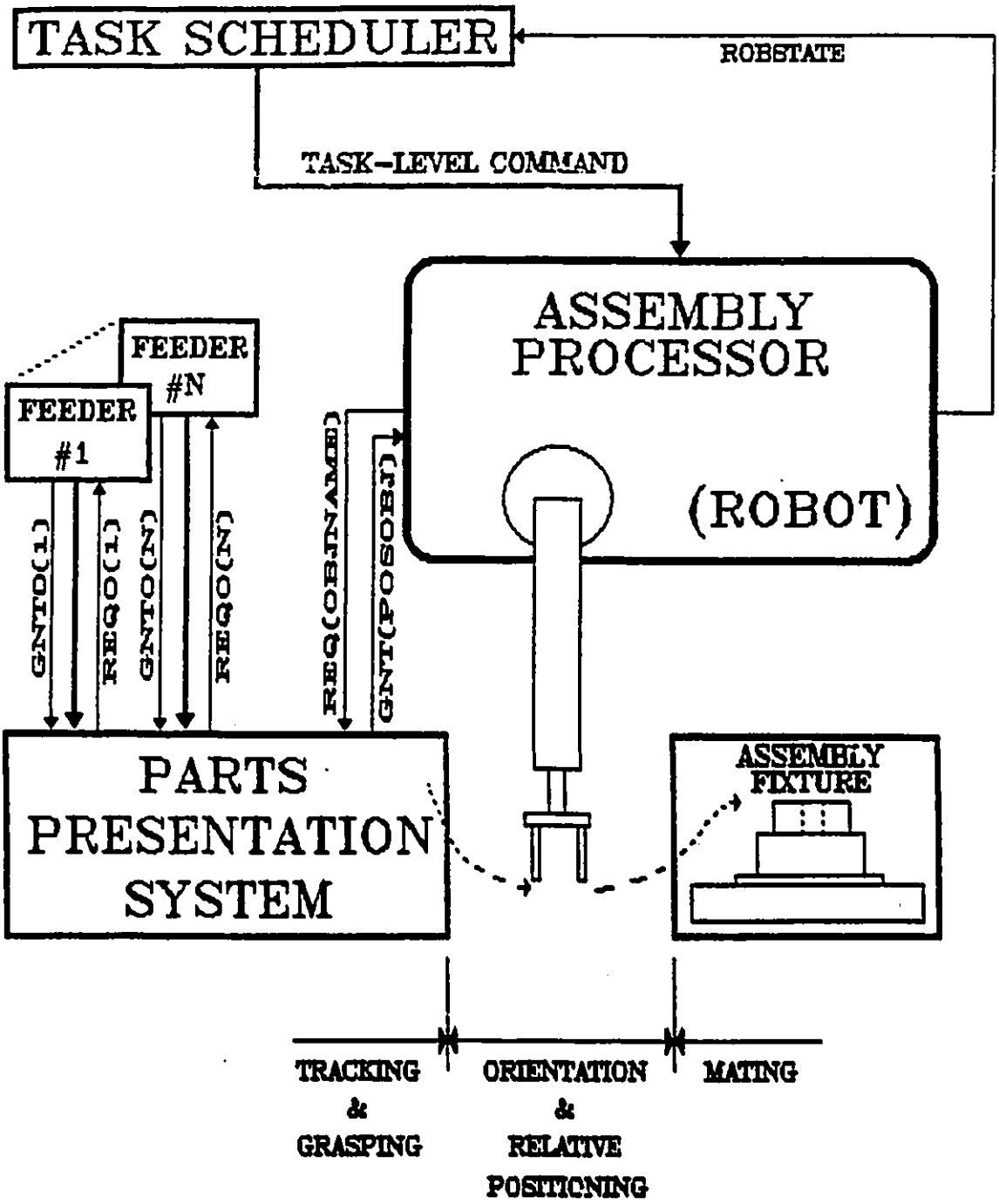


FIGURE 2.2: Functional model of the robotic assembly station.

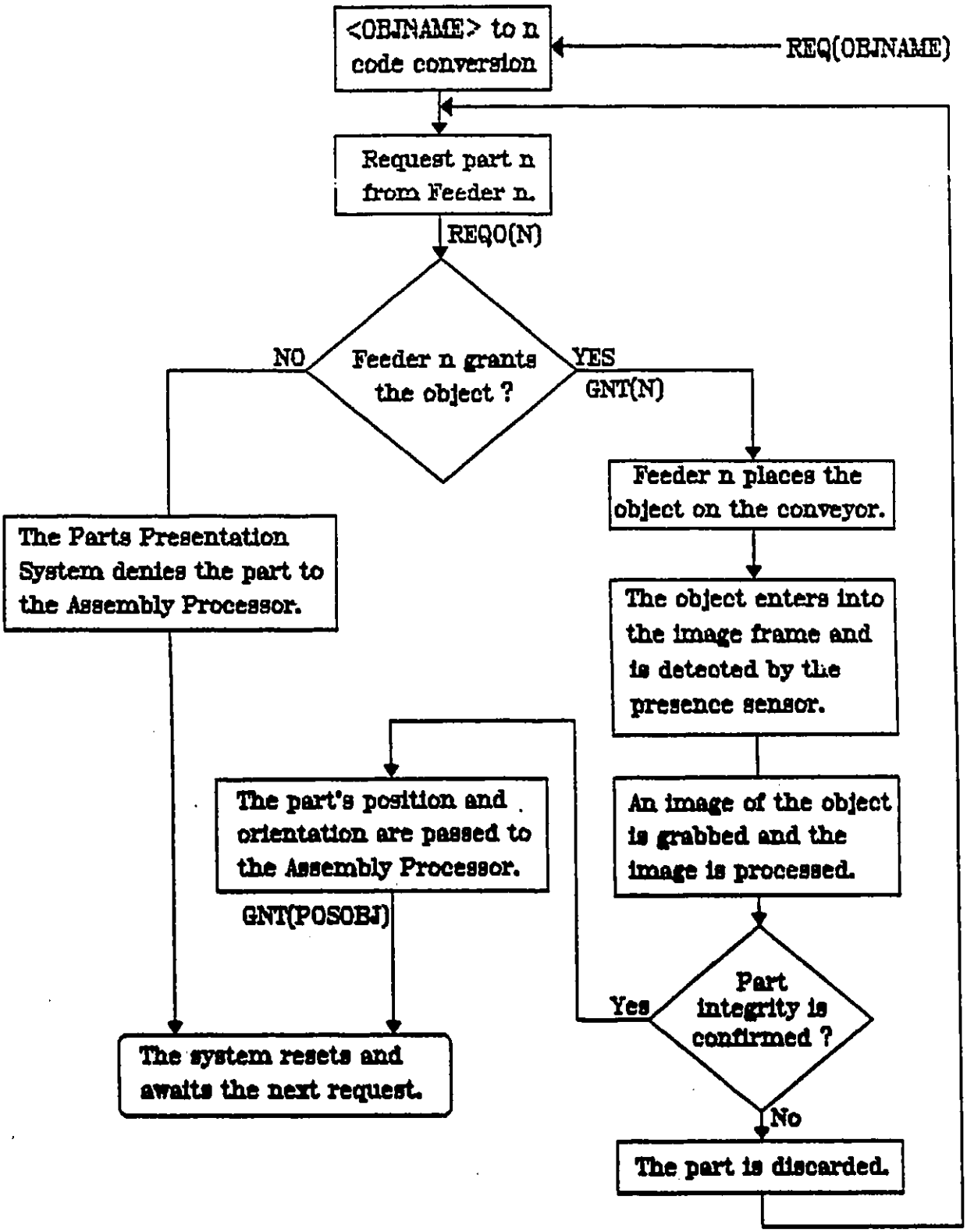


Figure 2.3: A flowchart of the operations of the Parts Presentation System.

CHAPTER 3 THE PARTS PRESENTATION SYSTEM

3.1 INTRODUCTION

As described in the previous chapter the system to be implemented consists of one or more assembly parts passing through a series of steps: acquisition, relative orientation, positioning and final mating. The system requirements call for an "intelligent" system using various forms of sensory feedback thus providing a degree of adaptability to the system. The remaining pages of this chapter are dedicated to describing a parts delivery system providing the initial acquisitioning and positioning of the parts prior to assembly in a robotic workstation.

The system is based upon a specialised conveyor system

delivering parts from a feeder to various robots along its path. The robots in turn remove desired parts from the conveyor and pass them to the appropriate workstation. A single conveyor may feed a number of robots, each of which may in turn feed several assembly stations. The concept of parts entropy as introduced by Sanderson[1] and described by Petriu et al[2]. is used to analyze the acquisition process in terms of position and orientation uncertainty.

3.2 POSITIONAL ENTROPY ANALYSIS OF THE SYSTEM

Sanderson states that positional entropy is a useful method for the description of an assembly task as a measure of the uncertainty of the position and orientation of the parts involved and he proposes that the goal of an assembly system is to "reduce the joint entropy among parts by mating them in stable configurations."[1] It is this manner that the steps involved in the operation of the parts presentation system are described.

---

[1] A.C. Sanderson, "Parts Entropy Methods for Robotic Assembly System Design," Proc. of IEEE International Conference on Robotics, Atlanta, pp. 600-607, March, 1984.

[2] E. Petriu, J.P.T. Goguen, and B. Karoui, "Multi-Sensor Robotic Tracking and Grasping," Proc. of ISA Digicom '87 Conference, Montreal, October, 1987.

The structure of the parts presentation system is illustrated in Figure 3.1 beneath a photograph of the workstation from March, 1988. It consists of a set of feeders, an indexed conveyor system with associated presence sensor and vision system, and one or more robot arms. These devices interact in order to deliver and affirm delivery of desired parts to various robotic assembly stations.

The feeders are devices which place, upon command, on the conveyor belt parts stored within. These devices require that the part conforms to a set of passive positional constraints and represent the first reduction of the parts entropy of the object to be delivered.

The conveyor moves continuously and transports the part through the image frame of vision system and ultimately delivers the part to a robot arm situated alongside or at the end of the conveyor. The conveyor is indexed providing a reference between the image frame of the vision system and the robot coordinate frame.

When the object is in the image frame it triggers the presence sensor which in turn causes the image digitizer to initiate a frame grab. As well the conveyor code at that point is recorded and converted to its corresponding absolute position on the

conveyor. The image processing system then analyses the digitized image as the part continues to travel along the conveyor. The image analysis confirms the identity of the part and provides positional information.

As the object moves within the work space of the robot arm the object, whose position has now been referenced to the robot's coordinate frame as a result of the code detection unit located opposite the robot arm, is tracked by the arm until the object reaches the point of lowest positional uncertainty i.e., when the previously recorded code is detected, the arm picks the part from the conveyor. At this point the object's positional entropy should be less than the "open/closed" hysterysis of the gripper of the robot arm.

When the object is placed within the grasp of the robot arm under the conditions stated the positional uncertainty of the object is the reduced to that of the end effector of the robot arm.

Figure 3.2 is a positional entropy analysis of the parts presentation system. It shows the theoretical results for the system with and without the use of the progressive image processing system described in the next chapter. The next paragraphs make reference to this chart but a basic explanation is in order. The figure plots position .vs. time for a part

being delivered by the conveyor belt and the robot arm at the opposite end of the conveyor belt. Note that the origin of position is considered to be the origin of the coordinate frame of the robot arm. At t_0 the robot arm is near the bottom of the graph (operating within its workspace at the end of the conveyor) and the part is dropped from the parts feeder onto the conveyor which limits its position to the shaded area at the top of the graph between t_0 and t_1. When the presence sensor detects the part, t_1, more information is known about the part and thus the boundaries of its possible positions are reduced. Using progressive vision, the low resolution image provides more positional information, t_2, and reduces the positional uncertainty and thus the bounds of the shaded region further. At this point the robot can move towards a position to intercept the part as it passes on the conveyor. Thus the position of the robot and the part move together. At t_3 the processing of the high resolution image of the part is completed and the positional uncertainty or entropy of the part is reduced to less than the compliance accepted by the end effector of the robot arm and at t_4 the robot removes the part from the conveyor.

Experiments described in the following chapters have demonstrated the performance advantages of the progressive vision system over a normal processing system. The graph of Figure 3.2 highlights two of these advantages. First, regular image processing requires a longer time for completion, t_5, and

second, whereas progressive vision provides position estimation in the middle of the vision processing, t_2, and thus allows the robot to initiate a move toward the part prior to the completion of processing, regular vision does not and therefore suffers a longer delay involved with the robot moving to the interception point, t_5 to t_6. A further explanation of each of these operations follows.

The robotic workstation is a multi-purpose operator capable of performing a variety of assembly tasks. The parts delivered to the station have, in general, been requested by the station via the software control system. As each part arrives it is subjected to a series of pre-defined assembly operations taking place at that specific workstation. Ultimately, after the incorporation of a pre-determined set of parts, an end product is arrived at and assembly concludes. The workstation, its assembly operations, and the corresponding control functions are viewed as independent processing elements with only a generalised, highly structured communications link to the parts presentation system thus effectively enabling the system to perform a variety of different assemblies using the same general delivery system.

The hardware as outlined is analogous in some ways to a computer architecture. The conveyor as a general delivery structure with distinct locations or addresses may be regarded as a bus system

for operands. The parts feeders act as input ports and the robot arms as ALU's performing on objects rather than operands. The workstations as previously stated are similar to independent processors requesting and receiving data (parts) from the system via the output devices. The system does require a form of operating system to accomplish the desired tasks in an organised fashion. This system, outlined below, provides the functions for requesting and acknowledging receipt of various parts as well as some error checking to ensure the integrity of the data (confirm the identity and location of the part).

3.3 SYSTEM CONTROL

The control software or "operating system" of the robotic assembly cell is composed in a modular manner. The assembly processor as described operate independent of the parts delivery system with intercommunications limited to a structured communications pathway. The control of the assembly processor is isolated from the control of the parts presentation system as well. This is shown in Figure 3.3 which highlights the control features of the system and the communications link between the workstation and the parts presentation system.

The assembly functions of the robotic cell are not the subject

of this thesis but require explanation in order to provide a general understanding of the associated parts presentation system. Control of the assembly stations is based upon the multi-sensor feedback execution of a set of pre-programmed assembly operations. The system is hierarchically organised and at the highest level communicates with the control system of the parts presentation system via the communications path using a single command mentioned in the previous chapter and a second optional command suggested for a multi-assembly processor system:

REQ(OBJNAME)- a request for a part identified by OBJNAME.

ACK(POSOBJ)- (optional) confirms receipt of the part at the station by sending the position of the object, POSOBJ, back to the parts presentation system essentially freeing up the location on the conveyor occupied by the part.

The parts presentation system in turn uses a single command to communicate information regarding the request initiated by the workstations:

GNT(POSOBJ)- The data ready command acknowledges the transfer of the part to the conveyor and provides the workstation with the position of the part.

Another optional command indicating the unavailability of the requested part may be used to enhance the communications or this may be incorporated as part of the GNT command where a special location POSOBJ indicates that the request for the part is denied. This sort of command would be useful in cases where the robotic workstations have alternate work that may be perfcrmed in the absence of certain parts.

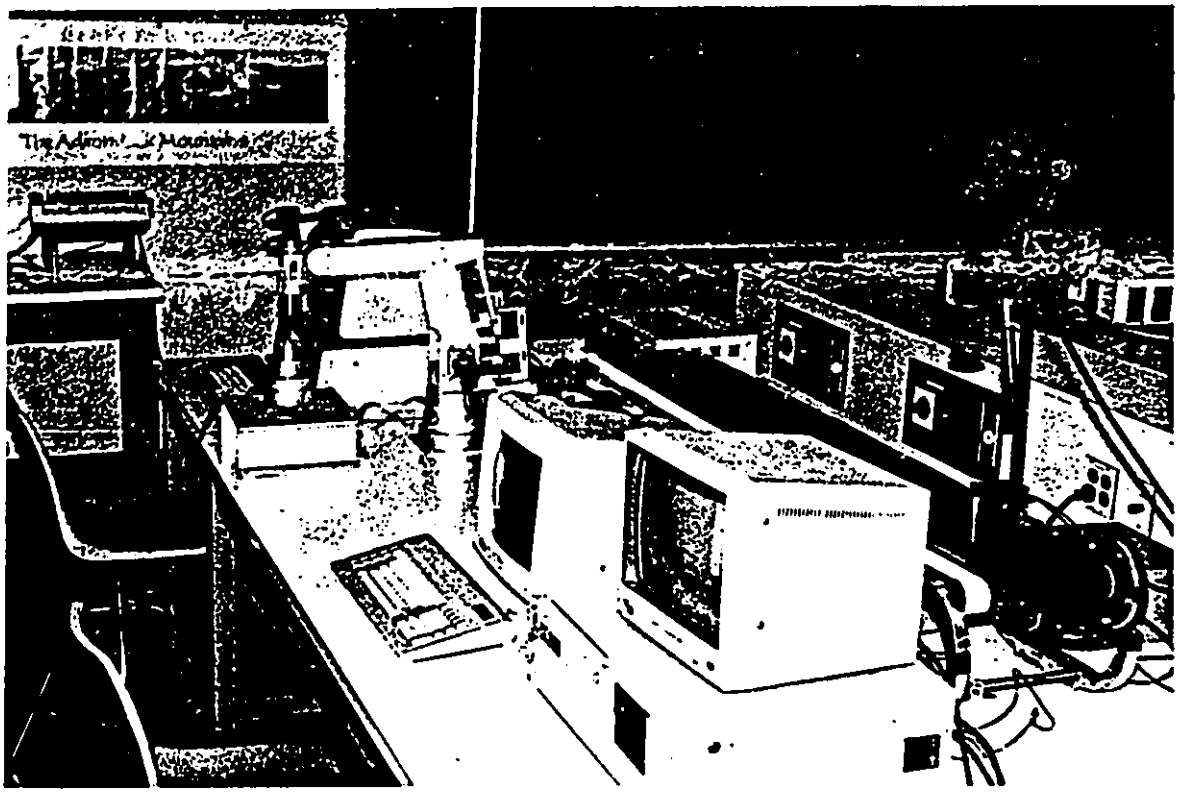
The control software of the parts presentation system waits upon requests from the individual workstations for various parts. As each is provided the software announces the transfer of the part and provides its location to the workstation which in turn acknowledges receipt of the part. The presentation system is controlled in a hierarchical manner. Each module in the lower levels of the control hierarchy is involved with the input from individual sensors. This data is subsequently interpreted at the next level generating information about the part in question. This information is interpreted and used to provide the robot arm and subsequently the robot workstation with the required positional data. The process is initiated at the highest levels of the hierarchy where communications with the

workstations takes place.

3.5 CONCLUSION

The modularity of the control system with the distinct division between the assembly station and the parts presentation system provides a natural incentive for parallelism in the operations of parts delivery and assembly. This combined with the use of multiple workstations increases the throughput of the system and leads to maximum use of the conveyor locations.

A detailed examination of the operation of the image system in conjunction with the presence sensors as well as the conveyor and associated coding follows. As well an examination of the kinematics of a SCARA type robot used as the robot arm for the part tracking and grasping is included. The interaction of each sensor is illustrated and the thesis concludes with an examination of the integrated system.



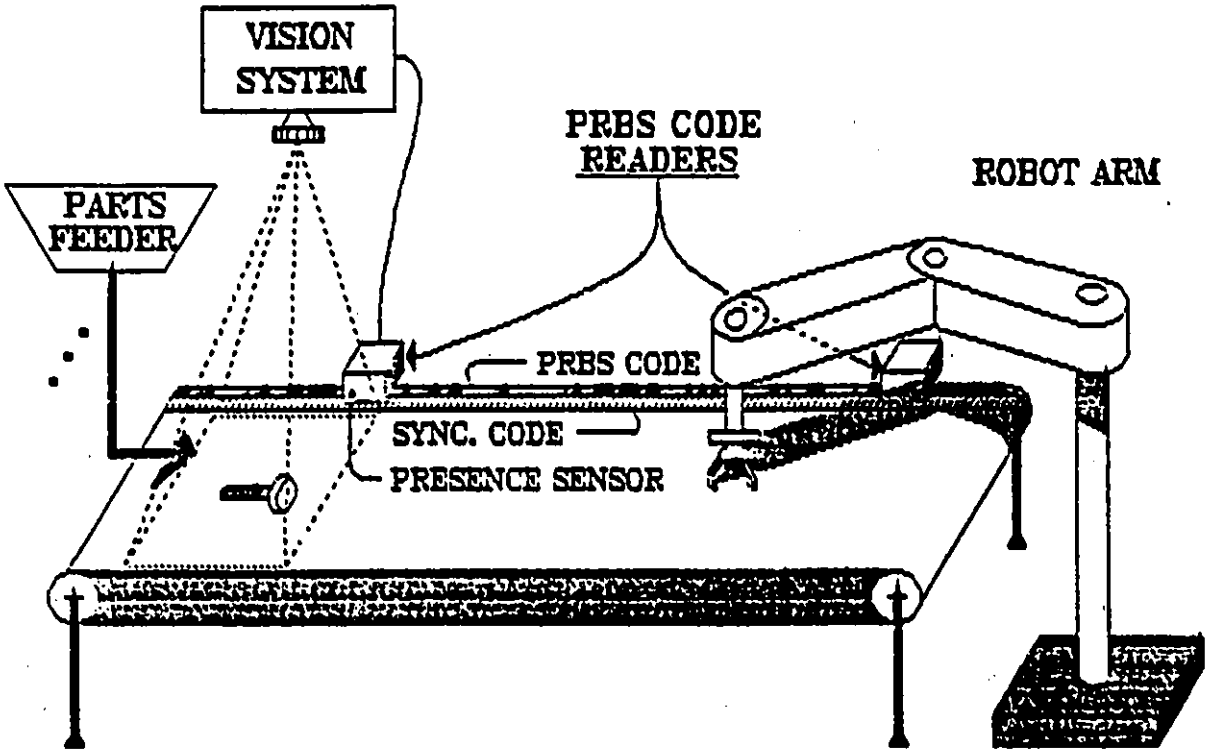


FIGURE 3.1 A photograph and an illustration of the vision-based parts delivery system for robotic assembly stations.

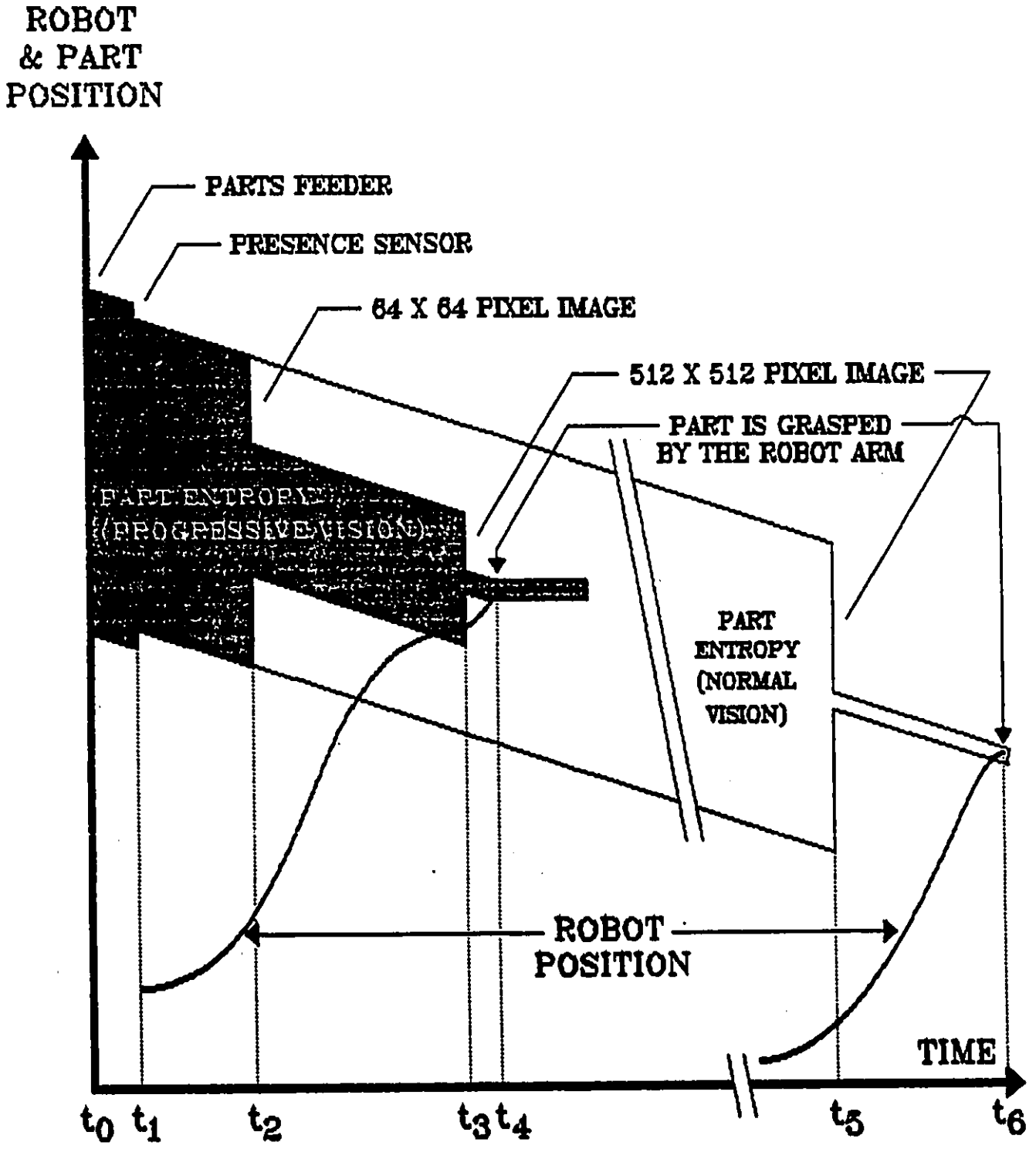


FIGURE 3.2: **A plot showing the expected positional entropy of the part being delivered using the parts presentation system of Figure 3.1.**

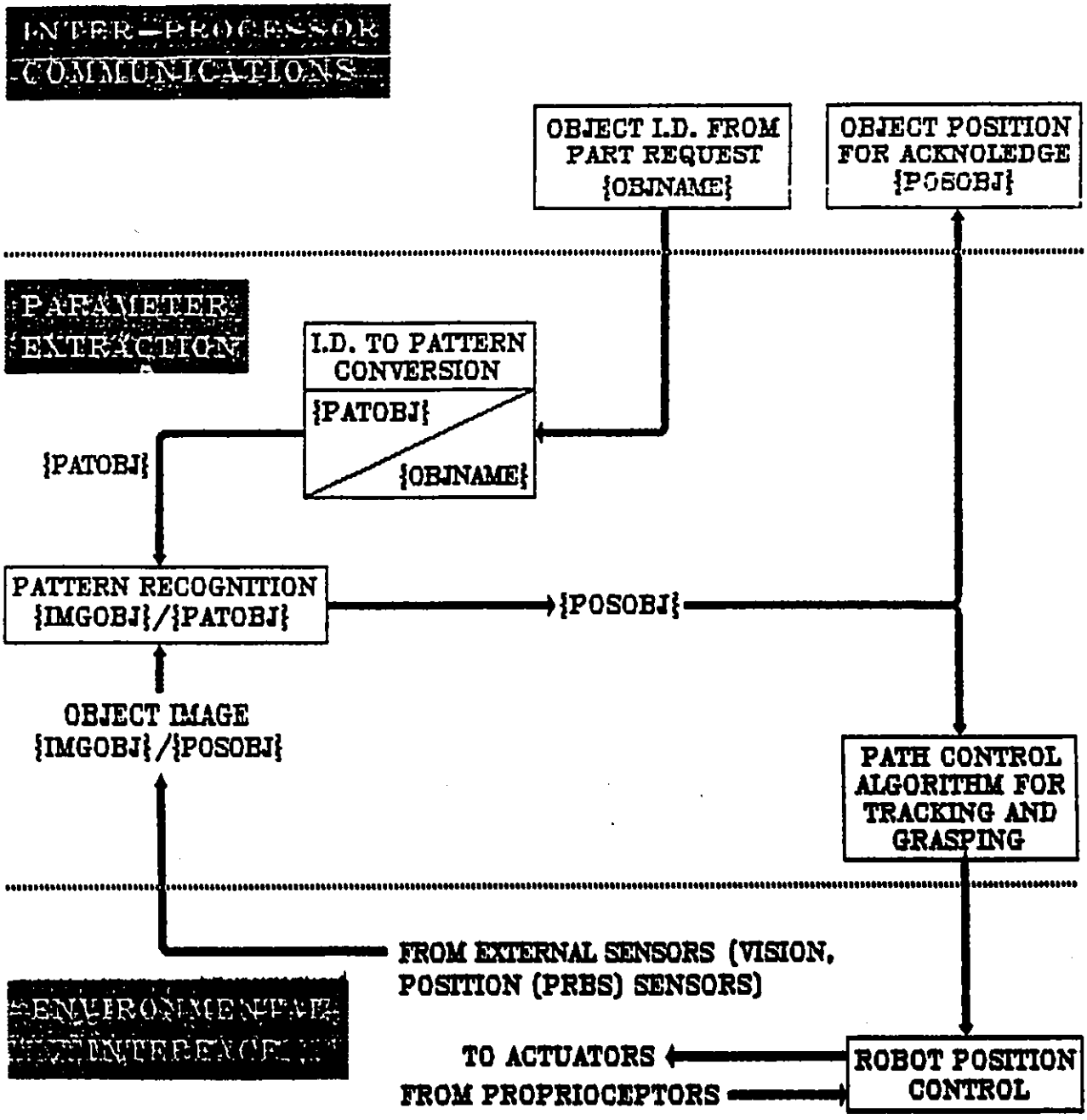


FIGURE 3.3: **A graphic depiction of the three levels of processing involved in the parts presentation systems interpretation of external sensors and communications with other processors such as those involved with assembly.**

CHAPTER 4 IMAGE WINDOWING TECHNIQUE FOR A PROGRESSIVE VISION SYSTEM

4.1 INTRODUCTION

The digital image provided by the video digitizer has a resolution of 512 X 512 pixels or over 260,000 individual pieces of information about the scene to be analyzed. The goal of image processing is to extract relevant information from this array of pixels. In order to maintain the accuracy provided by this high resolution image, during processing it is usually necessary to perform, at each stage in the information extraction process several mathematical operations involving each of the pixels. As a result simple processing of a binary image, such as calculation of moments for some target object in the image frame can require several seconds and even minutes using a micro-computer. Thus the use of a digital image for the

parts presentation system of the previous chapter is complicated by the need for the accuracy of the high resolution image and the demand for near real-time performance.

In many cases the actual regions of interest in the digital image occupy only a small percentage of the pixels that form the entire image. As a result the computer needlessly wastes a significant amount of computational time processing essentially irrelevant background pixels. The following chapter describes the concept of using a low resolution picture, created simultaneously from the higher resolution image provided by the digitizer to quickly select the region or regions of interest in the image. This so-called windowing limits the processing of the high resolution image to the regions of interest or windows defined using the low resolution picture and thus markedly improves the overall performance of the vision sensor while providing the desired accuracy. This is followed by an analysis of performance with respect to target object size in the image frame and resolution of the low resolution picture in order to determine theoretically the optimum resolution(s) of the low resolution picture(s) resulting in the fewest number of pixels to be processed and thus the shortest amount of time required to complete the information extraction process.

4.2 WINDOWING AS A METHOD OF REDUCING PROCESSING TIME

The use of the windowing technique for machine vision results in an overall reduction of the total time required for binary image processing. In order to achieve near real-time performance the algorithm assumes the simultaneous acquisition of both a high resolution binary image and a low resolution binary picture derived from this image. A search of the low resolution picture is done to define a region(s) enclosing the object(s) being examined. This region is transposed to the high resolution image and processing proceeds on the high resolution image within the bounds defined by this region or window. Using this method it is possible to reduce the number of pixels to be processed for feature extraction resulting in a significant reduction in the processing time and ultimately providing near real-time performance for many applications.

As an example consider the simple problem of locating disks as they pass through the image frame on a conveyor belt. Let's consider that a single disk may be 2 cm in diameter while the image may extend to 10 cm on each side as shown in Figure 4.1a. The area of a single disk is about 3.14 % of the area of the image. Given that the image contains an array of 512 X 512 pixels the physical resolution of the system is approximately 0.038 sq.mm per pixel. In general in order to find the disks with this accuracy most all of the 262 144 pixels must be

analyzed in some manner. However if the size of the image is reduced to say 64 X 64 pixels the physical resolution is reduced to 2.44 sq.mm per pixel and the image of the disks appears as shown in Figure 4.1b. In windowing the 4 096 pixels of the low resolution image are processed and windows encompassing the disks are defined. These are illustrated in Figure 4.1c. Then processing continues on the high resolution image but is confined to the area or pixels inside the windows. In this case a window will be a maximum of 15 pixels on each side at the low resolution or 120 pixels on each side in the high resolution. Thus in order to locate each disk with the high resolution using the windowing technique there is an overhead of 4,096 pixels per image and an additional 14,400 pixels per disk. In this case there are 3 disks shown. The total number of pixels required to be processed is less than 47,296 pixels as compared to 262,144 for processing with the high resolution alone.

This situation was simulated using an image created by the digitizer and similar algorithms for labeling and parameter extraction for each case. Using the windowing method the centroids and areas of the disks were found in less than 1 minute as compared to the 10 to 15 minutes required to process the high resolution image.

The question of what resolution is best for the low resolution picture arises when the use of the windowing technique is

considered. For example as the resolution decreases the size of the window enclosing the object grows much larger than the actual size of the object, Figure 4.2, which ultimately increases the number of pixels to be needlessly processed. However as the resolution increases, although the window encompassing the object conforms much more closely to the object, the number of pixels to be processed to find this window is much greater, Figure 4.3. An analysis of this trade-off and its relationship with the size of the object(s) in question follows as an effort to determine the requirements of the system generating the low resolution picture.

4.3 THE IDEAL LOW RESOLUTION PICTURE

The following exercise establishes a relationship between the size and the position of the object in the image with the number of pixels that must be processed in order to find the rectangular window encompassing the object in the high resolution image. The analysis considers two processing cost function definitions. The first deals with the overhead involved in finding the ideal window, one that just encloses the target object. It is essentially a count of the pixels required to define the window plus the number of pixels processed in one pass through the window used to refine the window to an ideal

size. The second contemplates cost of processing using a window generated by the low resolution image with no refinements for quantization error. These two functions are compared to each other based upon the number of passes through the windows required by the processing algorithm.

The equations examine the number of pixels processed to establish a rectangular window encompassing a circle. As a justification for this approach consider an object located somewhere in the image, Figure 4.4a. If the orientation of this object is unknown the object may be anywhere within a circle centered at the centre of area of the object. Thus for any object of unknown orientation in the worst case the object may be thought of as a circle, Figure 4.4b.

The position of the object in the image has a direct affect upon the size of the window generated from the low resolution picture. As the resolution of the image is decreased the representation of the circle (or any object) suffers from a quantization error. This error grows as the resolution decreases until all form of the circle is lost. In the best case this circle is located in one corner of the image. Thus as the resolution of the image is reduced the representation of the circle is compressed to one pixel. The window defined by this pixel will be the best window possible from this resolution. Figure 4.5a shows an 8 X 8 pixel representation of the circle.

Figure 4.5b is the 2 X 2 pixel representation of the same image. The window generated using the 2 X 2 picture is highlighted in Figure 4.5a. The size of this window and thus the number of pixels to be processed may be contrasted with the worst case that of the object located in the middle of the image, Figure 4.5c. The window generated from the 2 X 2 picture, Figure 4.5d, encloses the entire image as shown in Figure 4.5c. Therefore in order to examine the cost functions as defined, the worst case, that of a circle of varying radius centered in the middle of the image is used.

Let:

2^m X 2^m = the number of pixels in the high resolution image;

2^n X 2^n = the number of pixels in the low resolution image;

r = radius of the circle (mm);

l = length and width of the image (mm);

S_a = side length of a square window found using a resolution corresponding to an image of 2^a X 2^a pixels;

β = number of passes through the window required for the processing algorithm such that $\beta > 0$, β E I (E means element of).

For any 2^a X 2^a resolution:

$$S_a = 2 + 2*\text{TRUNC}[(r/l)*2^a] \quad , \quad r/l < 1/2 \quad ..(4.1)$$

Thus: $S_a \leq 2 + r/l * 2^{a+1}$, $r/l < 1/2$

In the worst case: $S_a = 2 + r/l * 2^{a+1}$, $r/l < 1/2$ ..{4.2}

The number of pixels required to find this window in the high resolution image may be considered as the cost of windowing C_w. This window, as previously stated is usually found by using a low resolution picture in combination with a high resolution image. However for this windowing technique to be feasible the number of pixels processed to find the window using just the high resolution image must be greater than the number processed to find the window using both resolutions. Case 1 examines the cost of finding the ideal window within the original high resolution (2^m X 2^m) image. Case 2 looks at the cost of finding the ideal window by a two-stage windowing procedure. An initial window is found using a low resolution (2^n X 2^n) picture. A high resolution representation is then considered for the initial window and the windowing procedure is repeated resulting finally in an ideal (high resolution) window. Case 3 is similar to Case 2 in that it considers the cost of windowing using 2 resolutions, however it accepts the window as defined by the low resolution image without improvement to the ideal form. Cases 2 differs from Case 3 since the algorithm of 2 has a higher initial overhead but fewer pixels to be processed within the window whereas Case 3 has a lower initial overhead but results

in more pixels needlessly processed with each pass through the window. These two cases are important because Case 2 is thought to be more efficient for processing routines that require several passes through the image frame, e.g., skeletal operations. The actual number of passes that determines which algorithm to use may be determined by comparing Cases 2 and 3. Consider:

$$\text{Case 1:} \quad C_w^1 = 2^m * 2^m \ , \ r/l < 1/2 \qquad ..(4.3)$$

$$\text{Case 2:} \quad C_w^2 = 2^n * 2^n + (S_{n,m})^2 \ , \ r/l < 1/2$$

$$\text{Case 3:} \quad C_w^3 = 2^n * 2^n + \beta[(S_{n,m})^2-(S_m)^2]$$

Where $S_{n,m}$ is side of the first stage window found using the low resolution picture but rescaled as a number of pixels in the corresponding high resolution image and S_m is side of the ideal square window in pixels of the high resolution image.

Therefore $\beta[(S_{n,m})^2-(S_m)^2]$ represents the cost of needless processing in Case 3 (resulting from the difference between the size of the actual window and the size of the ideal window).

Thus: $S_{n,m} = S_n * 2^{m-n} \ , \ r/l < 1/2$

And in the worst case:

$$C_w^2 = 2^{2*n} + (Sn * 2^{m-n})^2 \ , \ r/l < 1/2$$

$$= 2^{2*n} + [(2+2^{n+1}*r/l) *2^{m-n}]^2 \ , \ r/l < 1/2$$

$$= 2^{2*n} + 2^{2(m-n+1)} + (r/l)^2*2^{2(m+1)} + (r/l)*2^{2m-n+3} \ , \quad r/l < 1/2 \quad ..\{4.4\}$$

Whereas S_m is related to the size of the object in the high resolution image and may be found using Equation 4.2.

i.e, $$S_m = 2 + (r/l)*2^{m+1} \ , \ r/l < 1/2 \quad ..\{4.5\}$$

Thus:

$$C_w^3 = 2^{2n} + \beta[(2^{m-n+1}+(r/l)*2^{m+1})^2 - (2+(r/l)*2^{m+1})^2] \ , \quad r/l < 1/2$$

And simplifying:

$$C_w^3 = 2^{2n} + \beta[2^{2(m-n+1)}-2^2+(r/l)*2^{m+3}(2^{m-n}-1)] \ , \quad r/l < 1/2 \quad ..\{4.6\}$$

To find the ideal resolution(s) resulting in the minimum number of pixels processed to find the window, the cost, C_w, for Cases 2 and 3 must be minimized with respect to n, the resolution of the low resolution picture.

Thus to find n for Case 2 the following equation must be solved

for n given m and r/l:

$$0 = \frac{dC_w^2}{dn}$$

$$0 = 2^{2n+1}*\ln(2) - 2^{2m-2n+3}*\ln(2) - (r/l)*2^{2m-n+2}*\ln(2)$$

Simplifying: $0 = 2^{2n+1} - 2^{2m-2n+3} - (r/l)*2^{2m-n+2}$ ..(4.7)

The relationship of Equation 4.7 provides a theoretical basis for finding the resolution n that results in the fewest pixels being processes in order to find the best window circumscribing the worst case of an object somewhere in the image for the algorithm of Case 2. The number of pixels processed given a value for r/l can be found using Equation 4.4. The following is an example using the Matrox PIP 512 digitizer which provides a resolution of 512 X 512 pixels.

EXAMPLE 4.1: The highest resolution of the system is 512 X 512 pixels. The object has an r/l ratio of 1/4. Thus:

$m = 9$

$r/l = 1/4$

$$0 = 2^{2n+1} - 2^{21-2n} - (r/l)*2^{20-n} \quad ..(4.8)$$

$$0 = 2^{2n+1} - 2^{21-2n} - (1/4)*2^{20-n}$$

Simplifying: $2^{2n} = 2^{20-2n} + 2^{17-n}$

The best solution for this is n = 6. (Note that n must be an integer.) This means that using an r/l of 1/4 and a high resolution image of 512 X 512 pixels the best resolution for windowing resulting in the fewest pixels being processed is 64 X 64 pixels. Using Equation 4.4 the total cost is computed:

$$C_W{}^2 = 2^{2*6} + 2^{2(9-6+1)} + (1/4)^2 * 2^{2(9+1)} + (1/4) * 2^{18-6+3}$$

$$C_W{}^2 = 2^{12} + 2^8 + 2^{16} + 2^{13}$$

Thus:

$$C_{W2} = 78\ 080 \text{ pixels}$$

This is significant when compared to the cost of doing the same thing using only the high resolution image:

$$C_W{}^1 = 2^9 * 2^9$$

$$= 262\ 144 \text{ pixels}$$

The results of these calculations for Equation 4.4 for values of r/l from 1/2 to 1/20 were computed using the routine found in Appendix A and may be found in Table 4.1. The most notable result is the need for only one or two low resolution pictures to provide the most efficient windowing using the algorithm of Case 2 for the worst case of some object in the image frame despite the size of the object in relation to the size of the image frame. Thus following is a design for reducing the

resolution of a 512 X 512 pixel image to a 64 X 64 pixel picture, the ideal resolution for windowing objects greater in area than 1/100th of the area of the image frame.

To find n for Case 3 like Case 2 the following equation must be solved for n given m, r/l, and ß:

$$0 = \frac{dC_w^3}{dn}$$

$$0 = 2^{2n+1}*\ln(2) - ß*\ln(2)*2^{m-2n+3} - ß*(r/l)*\ln(2)*2^{2m-n+3}\ , \qquad r/l < 1/2$$

Simplifying:

$$0 = 2^{2n} - ß*2^{m-2n+2} - ß*(r/l)*2^{2m-n+2}\ , \quad r/l < 1/2 \qquad ..(4.9)$$

As for Case 2 the relationship described in Equation 4.9 provides a theoretical basis for finding the best resolution for the low resolution image resulting in the lowest processing cost in terms of pixels for the algorithm of Case 3. As an example of the use of this relationship consider the following:

EXAMPLE 4.2: The resolution of the system is again 512 X 512 pixels and the ratio of the radius of the circle to the size of the image frame is 1/4. The number of passes required is as yet unspecified. Thus:

m = 9

$$r/l = 1/4$$

$$0 = 2^{2n} - \beta*2^{20-2n} - \beta*(r/l)*2^{20-n}\ , \quad r/l < 1/2 \quad ..(4.10)$$

$$0 = 2^{2n} - \beta*2^{20-2n} - \beta*2^{18-n}$$

Simplifying: $$2^{2n} = \beta*(2^{20-2n} + 2^{18-n})$$

This relationship can be solved for the best value for n by considering various values for ß. The best solution for ß = 1 and ß = 2 is n = 6. However for $3 \leq \beta \leq 20$, ß E I the best solution is n = 7. The reason n increases in direct proportion to an increase in ß is the cumulative effect of the error of the window boundaries i.e., the window is larger than the ideal window and each pass adds to the number of pixels needlessly processed, pixels outside of the bounds of the ideal window. Thus for many passes it is desireable to have a more accurate window. The results of Equation 4.5, providing the best solution for n, have been computed using a routine found in Appendix A for various results of r/l and ß and are provided in Table 4.2.

4.4 OBSERVATIONS CONCERNING THE WINDOWING RELATIONSHIPS

The analysis of the previous section generates mathematical descriptions of two types of algorithms based upon windowing. The first, that of Case 2, attempts to find an ideal window and then processes within that window. It does so by first generating the parameters for a non-ideal window and then during the first pass through the window shrinks its boundaries to an ideal size. The second, Case 3, likewise finds a non-ideal window circumscribing the object but processes the window without shrinking it despite its greater than ideal size. These two cases will be examined further in this section in order that the choice of the better algorithm as well as the choice of the best lower resolution to be used may be accomplished.

Of primary concern to the discerning reader may be the actual solutions provided for the minimum cost functions. If one takes the time to find an integer value for n that results in a solution for the right hand side of Equations 4.7 and 4.9 nearest to 0 the cost of windowing found using Equations 4.4 and 4.6 is not necessarily the lowest cost possible. As it turns out for some situations the value of n found, representing the size of the lower resolution image, is one less or one greater than the best value for n. An example of this anomaly occurs when, for Case 3, $r/l = 1/4$ and $\beta = 3$. The value of n found using the right hand side of Equation 4.9 that results in the solution closest to 0 is $n = 6$. However the cost using $n = 6$

found using Equation 4.6 is 28,852 pixels as compared to the cost for n = 7 of 25,780 pixels. The reason for this is the non-linearity of the functions involved in computing the cost. Figure 4.6 is a plot of the costs of windowing an object with r/l = 1/4 and ß = 3 .vs. values of n from 2 to 9 for the algorithms of Cases 2 and 3. The curve is not atypical of the curves found for each set of values of n given r/l and ß.

A concern of the author's at the outset of the analysis of the windowing scheme was the relationship between the size of the object and the best lower resolution to be used. It now appears that for most values of r/l the required lower resolutions, (2^n X 2^n) are limited for 1 or 2 values. There appears to be, from Table 4.1 a direct relationship between the size of the object in the image frame and the size of the lower resolution, i.e., as the size of the object decreases the size of n also decreases. This is also reflected for Case 3 in Table 4.2 given a value for ß.

A final observation is the fact that the algorithm of Case 3 appears to be the least costly of the three cases examined. In fact for most situations studied it is in theory the best method to use although there are circumstances when the algorithm of Case 2 is more efficient.

4.5 SIMULATION OF THE WINDOWING ALGORITHM

Based upon the results of the analyses in this chapter it can be shown that the windowing method for reducing processing time without reducing the accuracy of the processing is successful from a theoretical point of view. The best windowing algorithm for most values of r/l and ß is that of Case 3 using a low resolution image of 64 X 64 pixels in conjunction with a high resolution image of 512 X 512 pixels.

The Pascal code used for the simulations may be found in Appendix B. These routines assumed the presence of both the high and low resolution images and performed a variety of functions including labeling and calculation of moments for one or more objects in one or more windows defined within the image.[1] The second routine also moved an associated robot arm to pick up the objects in the image. It should be noted that the windowing method does not limit the number of objects that may be processed. The number of objects and the spacing between them is limited only by the resolution of the high resolution image. However the speed of the processing operations was inhibited by the derivation of the low resolution image. A

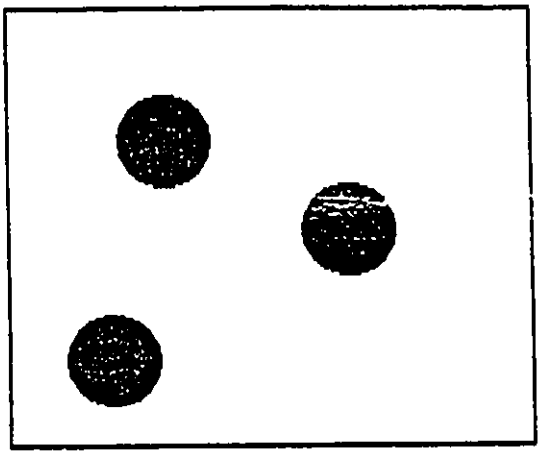
---

[1] The processing algorithms used in the simulations were adapted from W.E. Snyder, Industrial Robots: Computer Interfacing and Control, pp. 248-278, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1985.

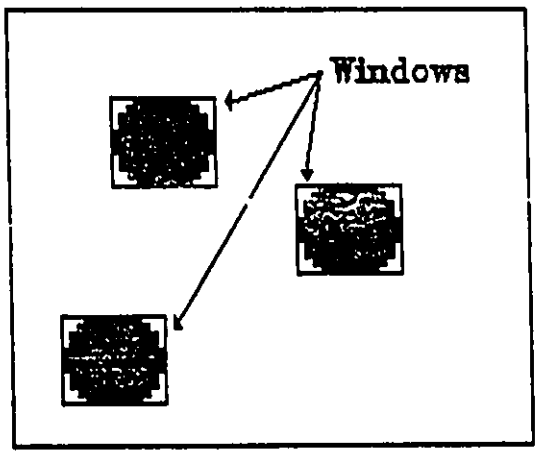
real-time derivation is required in order to achieve near real time performance of the system.

4.6 CONCLUSION

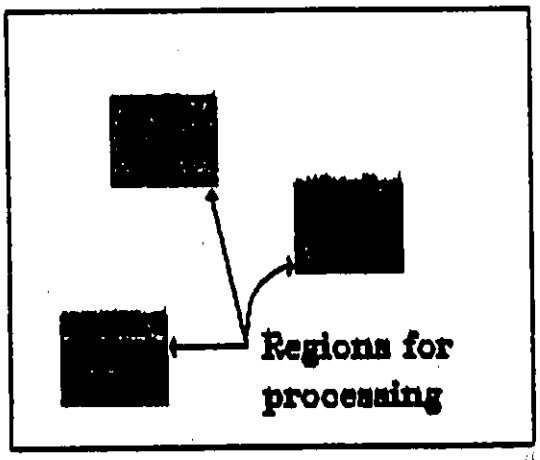
The windowing method for improving processing speed requires only one low resolution image, 64 X 64 pixels, derived from the high resolution image of 512 X 512 pixels in order to provide the most efficient use of the method for most object sizes and processing passes. This is significant as it appears to go against conventional thought concerning the efficiency of hierarchical systems. Rather than require several lower resolutions only one is really needed. However it should be noted herein that the calculation of the low resolution image from the high resolution image, an averaging and thresholding operation, in software as it was performed for the simulations requires more time in general than the actual processing for parameter extraction. This is seen as a great deficiency in the method and requires a solution in order to make the windowing method a viable option.



4.1 a: 512 X 512 pixel image of 3 disks. The image frame is 10 cm on a side and each disk is 2 cm in diameter.

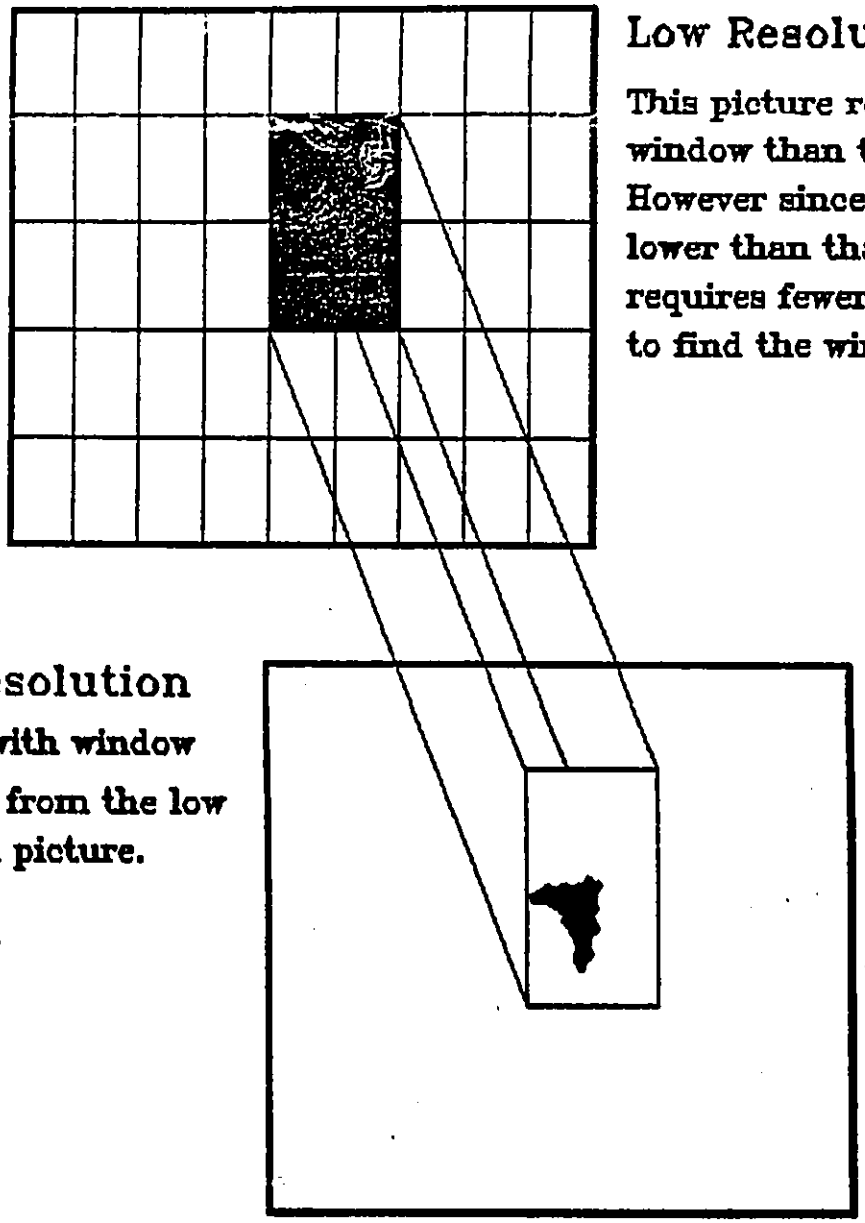


4.1 b: The same image of 4.1 a with a resolution of 64 X 64 pixels showing the windows defined by 1 pass.



4.1 c: The 512 X 512 image with the windows generated from the 64 X 64 image.

FIGURE 4.1: An illustration of the windowing process using images of 3 disks.



Low Resolution Picture

This picture results in a larger window than that of Figure 2. However since its resolution is lower than that of Figure 2 it requires fewer pixels processed to find the window.

High Resolution Image with window generated from the low resolution picture.

FIGURE 4.2: A low resolution picture of relatively low resolution and the resulting wide fit of the window surrounding the object in the high resolution image.

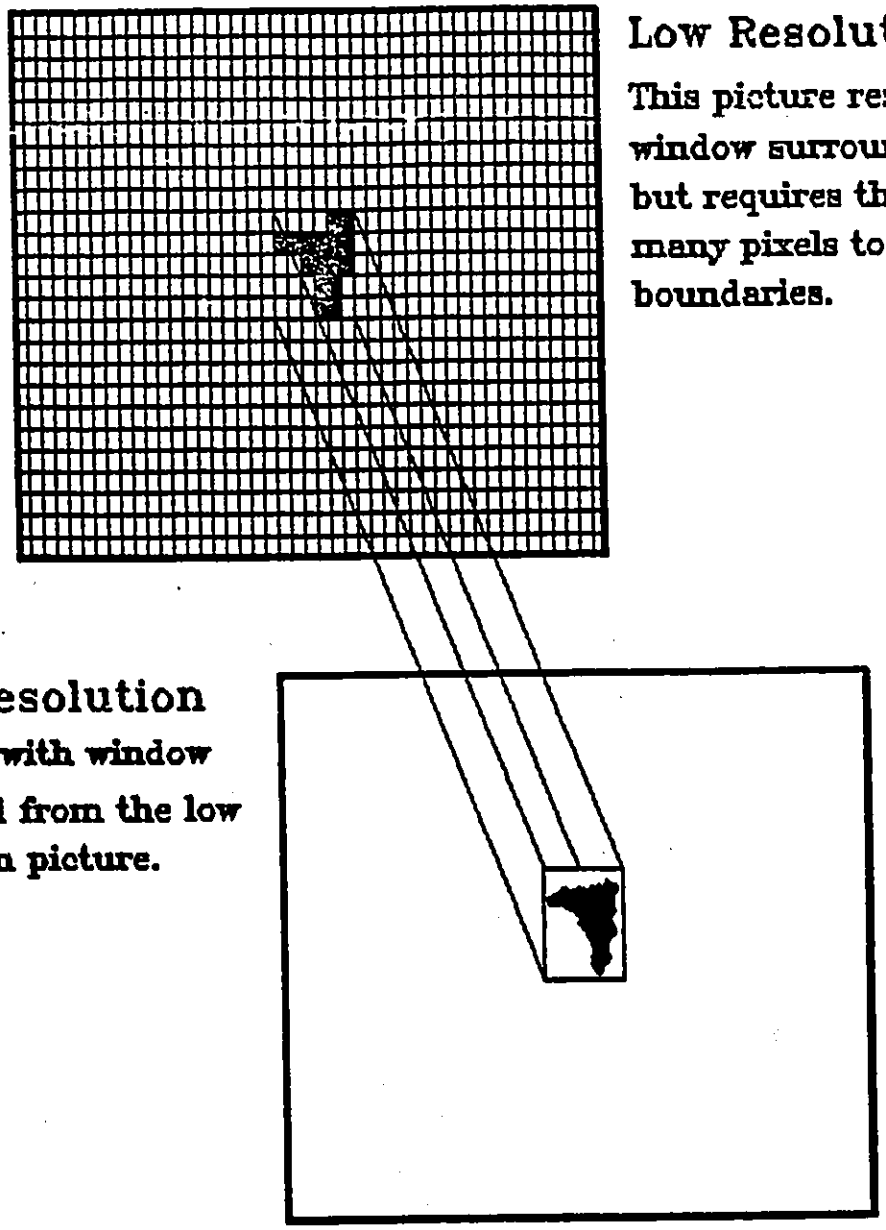
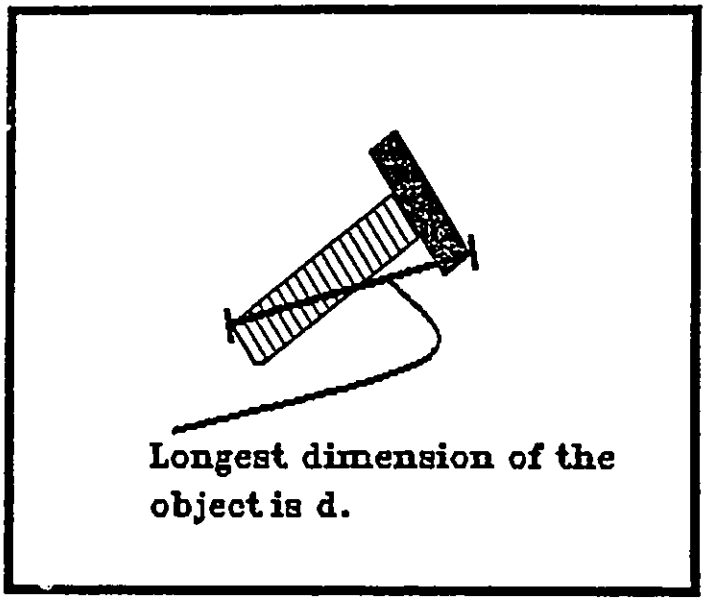
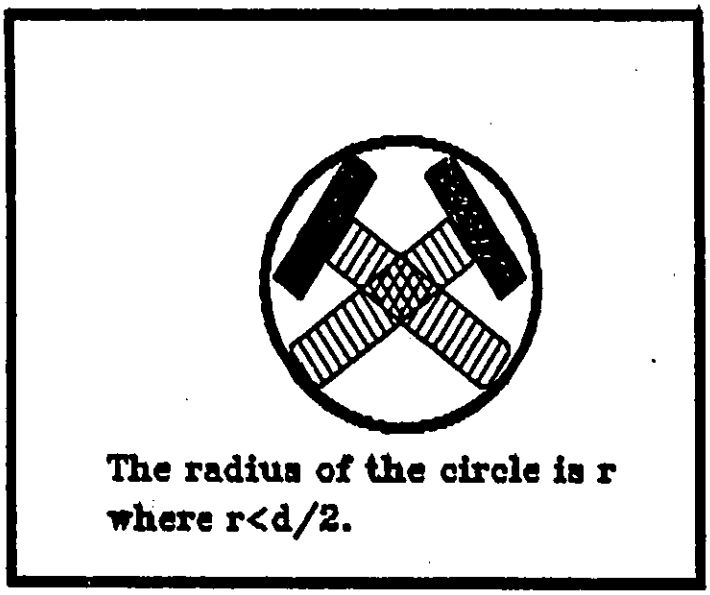


FIGURE 4.3: A low resolution picture of relatively high resolution and the resulting close fit of the window arround the object in the high resolution image.

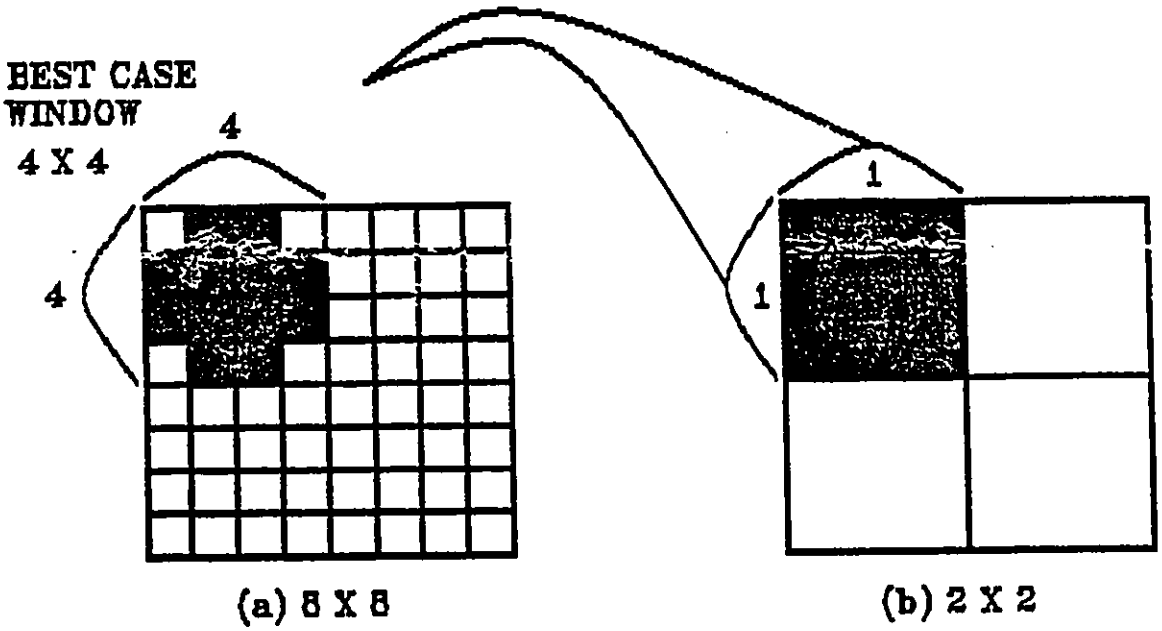


(a)



(b)

FIGURE 4.4: If the orientation of the object in (a) is unknown the object is limited to a circle whose radius is less than 1/2 the longest dimension of the object (b).



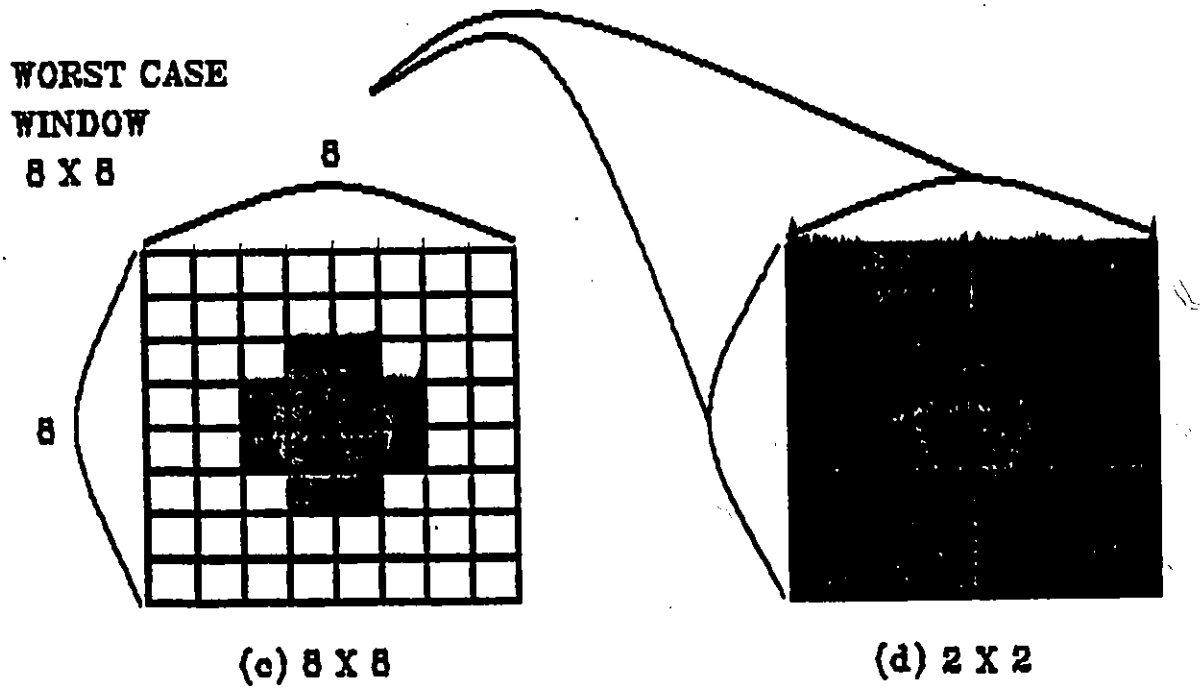


FIGURE 4.5: (a) and (b) illustrate the best case window from a 2 X 2 pixel image of the circle resulting in a window 4 X 4 pixels in size. (c) and (d) are an example of the worst case resulting in a window of 8 X 8 pixels.

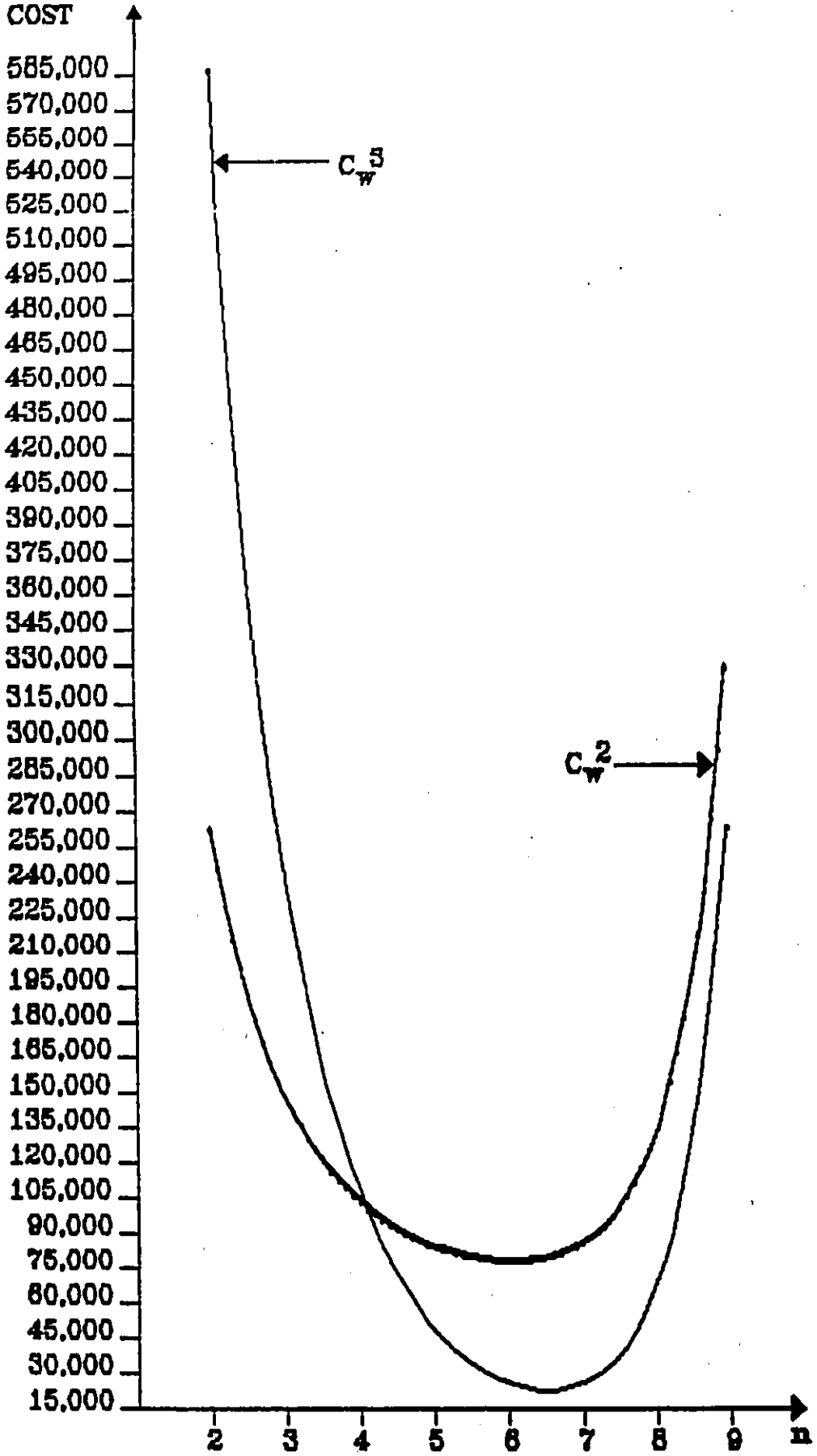

COST
585,000
570,000
555,000
540,000
525,000
510,000
495,000
480,000
465,000
450,000
435,000
420,000
405,000
390,000
375,000
360,000
345,000
330,000
315,000
300,000
285,000
270,000
255,000
240,000
225,000
210,000
195,000
180,000
165,000
150,000
135,000
120,000
105,000
90,000
75,000
60,000
45,000
30,000
15,000
C_w^3
C_w^2
2
3
4
5
6
7
8
9
n

FIGURE 4.6

r/l	COST	RES.
1/2	282 880	64 X 64
1/3	131 780	64 X 64
1/4	78 080	64 X 64
1/5	52 849	64 X 64
1/6	38 940	64 X 64
1/7	30 433	64 X 64
1/8	24 832	64 X 64
1/9	20 938	64 X 64
1/10	18 115	64 X 64
1/11	15 997	64 X 64
1/12	14 364	64 X 64
1/13	13 077	64 X 64
1/14	12 042	64 X 64
1/15	11 077	32 X 32
1/16	10 240	32 X 32
1/17	9 531	32 X 32
1/18	8 925	32 X 32
1/19	8 402	32 X 32
1/20	7 946	32 X 32

TABLE 4.1: An illustration of the results of equations 4 and 5 given a high resolution image of 512 X 512 pixels and various values for r/l.

r/l	T	COST	n	r/l	T	COST	n
1/2	1	18 684	6	1/11	1	6 955	6
1/2	2	28 792	7	1/11	2	9 813	6
1/2	3	34 996	7	1/11	3	12 672	6
1/2	4	41 200	7	1/11	4	15 530	6
1/2	5	47 404	7	1/11	5	13 389	6
1/3	1	13 905	6	1/12	1	6 737	6
1/3	2	23 715	6	1/12	2	9 379	6
1/3	3	28 852	7	1/12	3	12 020	6
1/3	4	33 008	7	1/12	4	14 661	6
1/3	5	37 164	7	1/12	5	17 303	6
1/4	1	11 516	6	1/13	1	6 554	6
1/4	2	18 938	6	1/13	2	9 011	6
1/4	3	25 780	7	1/13	3	11 469	6
1/4	4	28 912	7	1/13	4	13 926	6
1/4	5	32 044	7	1/13	5	16 384	6
1/5	1	10 062	6	1/14	1	6 396	6
1/5	2	16 069	6	1/14	2	8 696	6
1/5	3	22 055	6	1/14	3	10 996	6
1/5	4	28 454	7	1/14	4	13 296	6
1/5	5	28 972	7	1/14	5	15 598	6
1/6	1	9 127	6	1/15	1	6 140	5
1/6	2	14 157	6	1/15	2	8 423	6
1/6	3	19 188	6	1/15	3	10 586	6
1/6	4	24 219	6	1/15	4	12 750	6
1/6	5	26 924	7	1/15	5	14 913	6
1/7	1	8 444	6	1/16	1	5 884	5
1/7	2	12 792	6	1/16	2	8 184	6
1/7	3	17 140	6	1/16	3	10 228	6
1/7	4	21 488	6	1/16	4	12 272	6
1/7	5	25 481	7	1/16	5	14 316	6
1/8	1	7 932	6	1/17	1	5 658	5
1/8	2	11 788	6	1/17	2	7 973	6
1/8	3	15 604	6	1/17	3	9 912	6
1/8	4	19 440	6	1/17	4	11 850	6
1/8	5	23 276	6	1/17	5	13 789	6
1/9	1	7 534	6	1/18	1	5 457	5
1/9	2	10 972	6	1/18	2	7 786	6
1/9	3	14 409	6	1/18	3	9 631	6
1/9	4	17 847	6	1/18	4	11 476	6
1/9	5	21 285	6	1/18	5	13 320	6
1/10	1	7 215	6	1/19	1	5 278	5
1/10	2	10 334	6	1/19	2	7 618	6
1/10	3	13 454	6	1/19	3	9 379	6
1/10	4	16 573	6	1/19	4	11 140	6
1/10	5	19 692	6	1/19	5	12 901	6

TABLE 4.2: **The results of Equation 4.5 for several values of r/l and T.**

CHAPTER 5 THE VISION HARDWARE

5.1 INTRODUCTION

The necessity of performing a generalised resolution reduction by hardware becomes apparent when a software solution is attempted. Averaging the 262,144 pixels of the high resolution image to 4,096 pixels can require several seconds of processing time on a micro-computer. This of course degrades the performance of the system beyond the time savings gained using the windowing technique. A hardware implementation on the other hand can create the low resolution picture concurrent with the grabbing of the high resolution image, a process requiring only a few milli-seconds.

The following chapter describes a proposed design for the custom

hardware built as an enhancement for a Matrox PIP 512 image digitizer. The hardware is to integrated in an I.B.M. PC-XT system. This hardware will, upon command from the micro-computer, gather the data stored in the image frame buffer of the digitizer and sum 8 X 8 blocks of pixels from the 512 X 512 pixel image and subsequently store these values in a buffer on the enhancement board which is memory mapped to the micro-computer. The process essentially reduces the resolution of the 512 X 512 pixel image to 64 X 64 pixels. This corresponds to, using the notation of the previous chapter, n = 6 which is the best solution for most windowing situations.

The following sections are divided according to the differing parts of the hardware. This includes a description of the control signals used to synchronize the enhancement board with the digitizer. As well the data path on the enhancement board is illustrated with an investigation of the problems involved in the real-time computation of the low resolution image. The chapter concludes with a statement concerning the present level of functionality of an actual working model of the system described.

5.2 CONTROL SIGNALS[1]

The enhancement board requires some form of synchronization with the digitizer board in order to capture the data stored in the frame buffer of the digitizer. The control signals do such things as generate addresses for the pixels, clock the data through accumulators, and clear the 64 X 64 image frame buffer during a frame grab.

The hardware uses the following control signals which may or may not exist in some form but are easily generated from most image digitizers:

BADP: Begin Active Pixel Data is active while data on D0-D7 is valid. It is inactive during lulls in the data transfer such as during the horizontal and vertical retraces.

D0-D7: These 8 lines form the data bus used for the transfer of the pixel data.

MHS: The Master Horizontal Sync is active during

---

[1] It should be noted at the outset of this section that the description of the control signals and the timing involved in reading the PIP 512 digitizer's frame buffer is provided by Matrox Electronic Systems' PIP Video Bus Specification, Manual No. 10090-ME-00, Rev. 3, November 11, 1986.

the horizontal retrace of the master.

MVS: The Master Vertical Sync is active during the vertical retrace of the master.

PXCK: The Pixel Clock is generated by the master. It is the primary clocking signal in the computation of the low resolution picture. PXCK has a typical frequency of approximately 10 MHz.

MVSC: The vertical sync counter is enabled by the computer to initiate the resolution reduction operation. It simply counts the vertical syncs. In doing so it is used to control access to the memory between the first and third vertical syncs.

The digitizer used in this thesis did not directly provide the BAPD, PXCK, nor the MVSC signals. Rather BAPD and PXCK are generated from a combination of general timing signals. These signals include:

MCLK: The Master Clock is the main timing signal on the digitizer board. It has a frequency of approximately 20 MHz and a 50 % duty cycle.

MSIE: The Master Character Clock is another timing source with a frequency of about 1.25 MHz and a duty cycle of 62.5 %.

BK: The blanking line indicates when active video data may be found on the data lines from the digitizer.

The timing diagram of Figure 5.1 illustrates the timing involved in the generation of the BAPD and PXCK signals. These signals are constructed from the MCLK, MSIE, and BK signals using the logic depicted in Figure 5.2. Figures 5.1 and 5.2 are self-explanatory and any further description would be redundant.

There are two other timing signals required for the implementation of the resolution reduction scheme. The first, MVSC, the "MVS counter", is simply the second bit of a 4-bit binary counter clocked by MVS signal. The counter stops counting after 4 cycles of the MVS signal and is reset to 0001 binary when the computer performs a write to any of the memory-mapped addresses of the 64 X 64 pixel frame buffer on the enhancement board. Thus the MVSC is logic 0 until after a reset followed by a cycle of the MVS signal. During the second and third cycles of the MVS signal the MVSC is a logic 1. The fourth cycle of the MVS signal causes the MVSC to return to

logic 0 and the counter stops until the next reset. The purpose of this signal is to indicate the beginning and end of two consecutive image fields. This is extremely important for isolating the low resolution image buffer from the micro-computer during a frame grab. The second, the field number, FN, is the first bit of the aforementioned counter. It is a logic 1 just prior to the frame grab. As the first frame begins the FN switches to logic 0, and upon completion of the first field it returns to logic 1. This signal is used to selectively clear the 64 X 64 image frame buffer as will be shown in later sections.

With the control signals described above combined with some basic logic it is possible to control the flow of data from the digitizer through a set of accumulators to its storage in another frame buffer. In the process the resolution of the original image, 512 X 512 pixels with a width of 8 bits, is reduced to an image of 64 X 64 pixels with a width of 14 bits.

5.3 THE DATA PATH

The design of the data path on the enhancement board is in keeping with the practical realities of economy and the convenience of simplicity. However due to the format of the data flow from the digitizer board the control is somewhat

complex considering the obviously elementary operation of resolution reduction.

The pixel data coming from the digitizer appears on the data lines D0-D7 at the rate of 1 pixel/100 nanoseconds. Each 8 bit pixel must be added in turn to the other pixels of the same 8 X 8 block and the total of the 64 pixels from the 512 X 512 image make up one pixel in the 64 X 64 image. This summation process is complicated by the fact that pixels arrive row by row. As an added difficulty the rows are formatted in an interlaced scan. Thus when considering the addition of the pixels in the 8 X 8 blocks the data format dictated that the first 8 pixels arrive one after the other but the next 8 pixels begin 1 row later (less 8 pixels). However only 4 rows of the block appear consecutively. The other 4 rows are found in the next field of the interlaced scan. Using only the control signals previously defined the data is directed through the proper channels as described in the following paragraphs.

Figure 5.3 illustrates the design of the data path and Figure 5.4 contains the timing information of interest. As stated the system is based on the pixel clock signal which is divided by clocking a counter with PXCK in order to generate various control inputs. The system uses an interlaced memory and redundant accumulators. This is done to reduce the speed requirements of the memory at the cost of increasing the control

complexity. In this manner the system essentially doubles the time allotted for read and write operations allowing for the use of less expensive components in return for a slight increase in control complexity.

As an explanation of the operation of the hardware consider the generation of a 64 X 64 pixel resolution picture of the 512 X 512 pixel image stored in the frame buffer of the digitizer. This begins with a command from the micro-computer to the hardware to create the low resolution picture. This is simply a matter of performing a write to any of the 4096 memory locations on the enhancement board. This write operation, previously described, loads 0001 binary into the counter that generates the MVSC and FN signals. The next vertical retrace and the activation of the BAPD signal indicates the beginning of either the odd or even frame corresponding to the start of the image and data from the 512 X 512 pixel frame buffer of the digitizer board appears on D0-D7.(Note that since the image is to be averaged to 64 X 64 it is not necessary to know the actual frame number but rather only that 2 consecutive frames be included in the computation of the low resolution picture.)

Each of the interleaved memories stores 2048 of the pixels in the image. Channel 1 stores the odd numbered columns (1,3,5,...,63) and Channel 2 the even columns (2,4,6,...,64). With the activation of the BAPD signal and the start of the data

flow from the digitizer alternate groups of 8 pixels are added in the accumulators (A1) and (A2) i.e., the first 8 pixels are accumulated in (A1), the next 8 in (A2), the third 8 in (A1) and so on. As data is being summed alternately in the accumulators the value stored in the memory, either (M1) or (M2), at the corresponding pixel address is latched to the buffer, (B1) or (B2). The adders, (S1) and (S2) combine the results of the accumulators with the pixel sum from the memories. This sum is then stored in the memories at the current address. By controlling memory access 4 groups of 8 consecutive pixels are accumulated and stored in memory for each field resulting in 8 groups of 8 pixels, the 8 X 8 blocks, being summed and stored in their respective locations in the 64 X 64 pixel frame buffer. These operations are not entirely concurrent but rather occur in the following order:

1.a) 8 pixels are summed in accumulator (A1).

b) The data stored at the current address in memory (M1) is read and temporarily stored in buffer (B1).

c) The previous result from accumulator (A2) is summed with the data in buffer (B2) via adder (S2) and is stored in its corresponding location in memory (M2).

2.a) 8 pixels are summed in accumulator (A2).

b) The data stored at the current address in memory

(M2) is read and temporarily stored in buffer (B2).

c) The result from accumulator (A1) is added to the data in buffer (B1) via adder (S1) and is stored in memory (M1).

3. Steps 1 and 2 are repeated for all 262,144 pixels of the high resolution image.

The interleaved fashion of the dual accumulator and memory stages providing independent paths for odd and even pixels serves to double the time allowed for accessing the memories. A single path for data flow was seen to require close to the same amount of control as the plan described above but would have resulted in unrealistic performance requirements of the memories. (One such scheme required a read/write cycle of less than 50 ns.) The interleaved scheme above requires a read/write cycle performance of less than 400 ns.

A description of the control functions involved follows. Later sections are devoted to addressing modes and the interface with the micro-computer.

Each of the accumulators (A1) and (A2) are clocked on the rising edge of the PXCK*[1]. These accumulators are made to add

[1] An asterisk after the acronym of a signal e.g., BAPD, indicates that the signal has been logically inverted e.g., BAPD* = not(BAPD).

alternate sums of 8 pixels through a mutually exclusive inhibition of the summation process i.e., (A1) is inhibited when (PXCK/16)*[1] is a logic 0 and (A2) is inhibited when (PXCK/16) is a logic 0. (A1) is cleared both (PXCK/8) and (PXCK/16) are logic 1 whereas (A2) is cleared when (PXCK/8) and (PXCK/16)* are both logic 1.

The contents of the memories (M1) and (M2) are read during the operations of (A1) and (A2) respectively. The data from (A1) is latched by buffer (B1) on the rising edge of (PXCK/8)* as is data from (A2) by (B2). However the buffers only latch valid data on every other rising edge (see Figure 5.4). It is necessary to clear these buffers at the beginning of a new 8 X 8 block of pixels in order to essentially clear the frame buffer of the data from the previous image. The beginning of the 8 X 8 blocks is indicated when bits 0 and 1, the 2 least significant bits of the row address and the field number FN are all logic 0. When this occurs (B1) and (B2) are set to 0 rather than the value at that memory location. Finally note that the memories are isolated from the forward flow of the data path by three state buffers (3SB's). The 3SB's are always in the high impedance state except when the accumulated data is to be written to the memories. This happens for (M1) when (PXCK/4) is

---

[1] (PXCK/n) indicates that the pixel clock signal is divided by n where n = 2,4,8,... This means that the frequency of the signal PXCK is divided by n and that the period of the signal is multiplied by a factor of n.

logic 0, (PXCK/8) is logic 1, and MVSC is a logic 1. A write occurs on (M2) when (PXCK/4) is a logic 0, (PXCK/8) is a logic 0, and MVSC is a logic 1. By keeping the 3SB's in the high impedance state when MVSC is a logic 0 the data path is completely isolated from the memory except during a frame grab. As will be shown in the following sections the data and address buses of the micro-computer are isolated from the memories when MVSC is a logic 1. This prevents bus contention problems and allows separated access to the memories.

5.4 ADDRESSING MODES AND THE USER INTERFACE

There 64 X 64 pixel memory on the enhancement board is memory mapped to the micro-computer as well as being accessible to the enhancement board's data path during a frame grab. This duality is accomplished by isolating the data path from the memory's data and address buses except during a frame grab when the memory is isolated from the computer's data and address lines. As noted in the previous section access to the memory is controlled or arbitrated by the state of the MVSC signal.

Figure 5.5a highlights the employment of the 64 X 64 pixel frame buffer as part of the memory map of the micro-computer. As shown the memory occupies 4096 addresses from 80,000 hex to

80,FFF hex. These addresses pass through a transparent 3SB which is switched to the high impedance state during a frame grab preventing the micro-computer from accessing the address lines of the memories. Of interest is the way the memories both receive the same address from the micro-computer during a read. The data lines are channelled through a transparent 3SB which is selected or placed in the transparent state according to the value of the least significant bit of the address (logic 0 indicates the even memory, logic 1 the odd) provided that the address decoding indicates an address within the range specified.

Figure 5.5b illustrates the access to the memory during a frame grab. Addresses are generated using 2 sets of counters, one for the rows and one for the columns. These addresses are latched alternately by a set of 3SB as they are generated. The addresses reflect the position of the data that is being added in the active accumulator. The addresses need to be temporally stored in the buffer since the final add and write operations occur when the next 8 pixels are being accumulated. These buffers are in the high impedance state except during t frame grab.

The user need only to read any address of the frame buffer on the enhancement board to initiate a frame grab. After a period of approximately 31.88 ms during which memory access from the

micro-computer is blocked, the data is ready to be processed and may be read directly, pixel by pixel, row on row from the memory locations. The data is 8 bits wide. The actual width of a byte of memory at any location in the frame buffer is 16 bits of which only the least significant 14 are relevant. (Each location has accumulated 64 or 2^6 8 bit values. Thus the maximum value after summation is $2^6 * 2^8 = 2^{14}$.) The 8 bits passed to the micro-computer are bits 7 through 14 (where bit 1 is the least significant and bit 16 is the most significant). This data may be read without processing in which case the value reflects the numerical average of the 64 pixels summed. If a machine vision effect is desired the values can be made to reflect a number of things. For example if the thresholding mechanism sets the pixel values at either 40 hex indicating that the pixel is 'on' or 00 hex meaning 'off', then the value accumulated will reflect the number of pixels in the high resolution image that are set to 40 hex or turned on. A further thresholding operation can be used to decide how many pixels of the high resolution image must set to 40 hex in order to turn on the low resolution pixel.

As a final note consider that to perform the windowing operation the pixels in the low resolution image were turned on if any pixel in the corresponding 8 X 8 block is turned on. The system described meets the demands of high speed thus increasing the performance effects of the windowing operation and provides the relevant information for the operation. As well the hardware in

no way limits the number of objects allowed in the image frame at any one time and is versatile in that it may be used for a number of other applications and is not limited to being simply a part of the windowing operation.

5.5 CONCLUSIONS

The most important conclusion that can be drawn from this exercise is that in order to do practical real-time image processing one must generally look to a hardware solution or dedicate a large processing unit to this end because of the huge amount of information involved. The performance of the system is improved by up to 25 times using the windowing approach in combination with the hardware generating the image. Finally note that unlike other machine vision systems which use the windowing approach this method allows for filtering and labeling operations on both levels and in so doing it can process several objects in the same image even when packed tightly together.

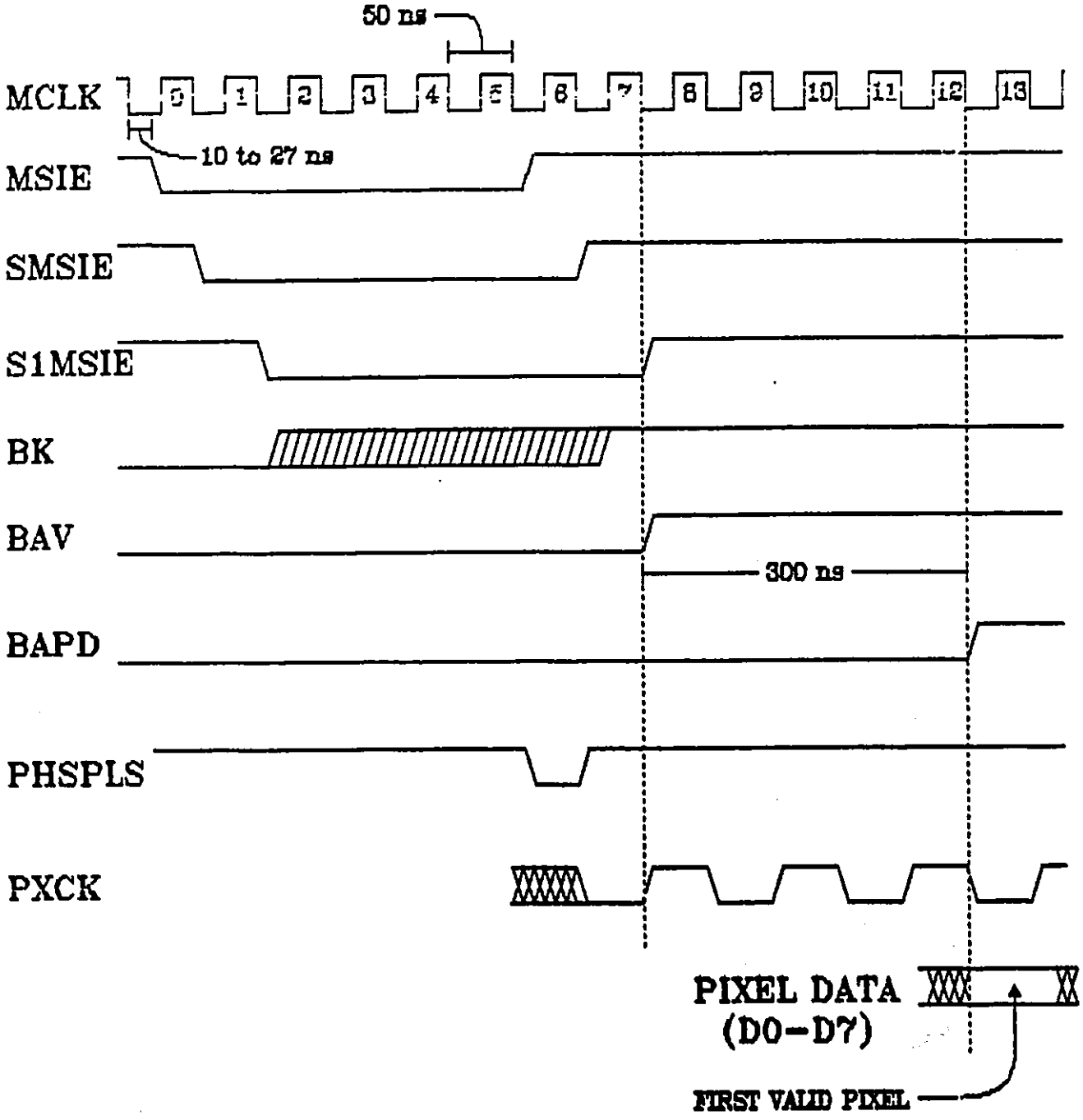


FIGURE 5.1: **A timing diagram illustrating the relationship between the various control signals. Note: PHSPLS is a logical combination of the MSIE and the synchronized MSIE signal, SMSIE, used to generate the phase of the PXCK signal.**

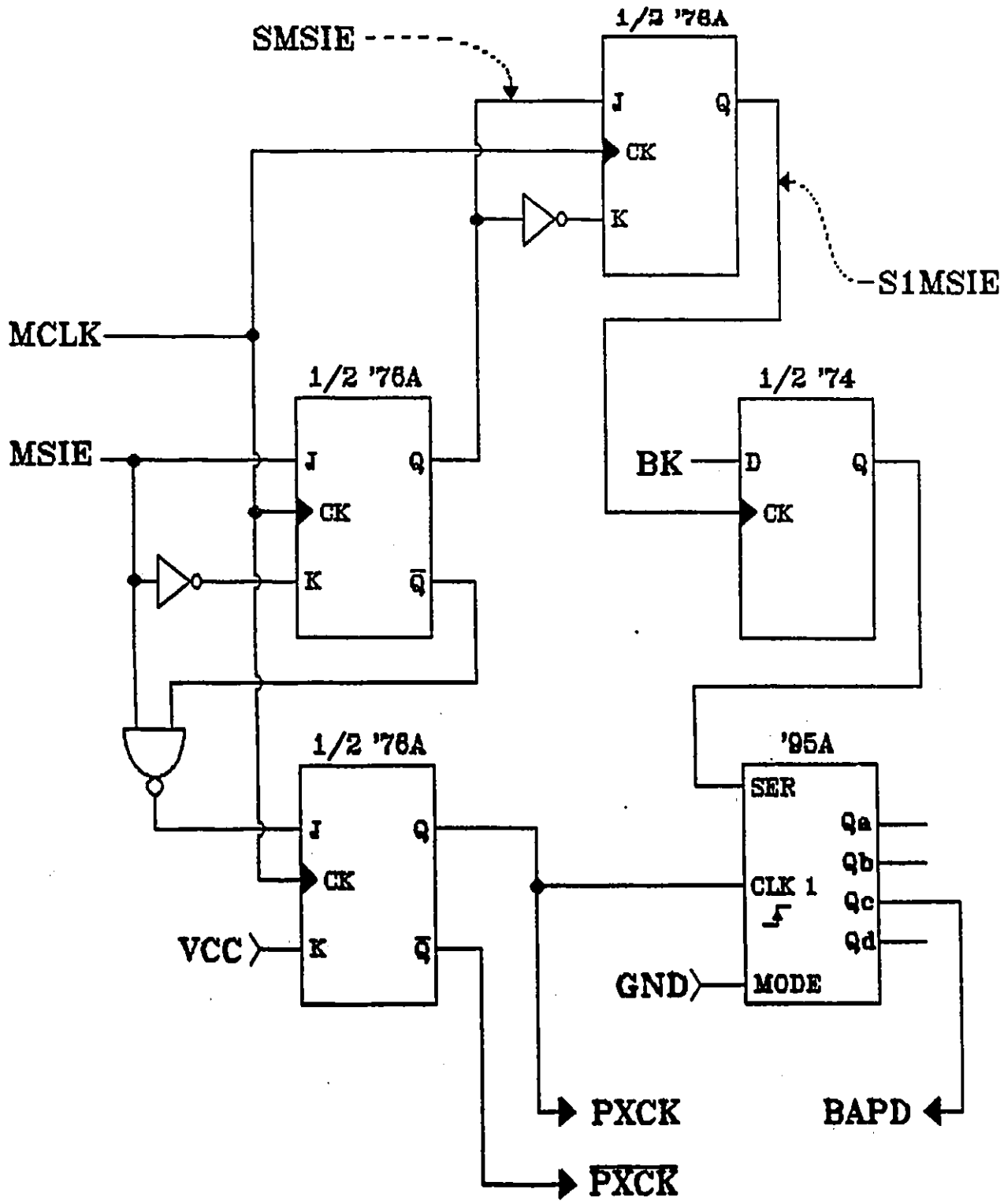


FIGURE 5.2: **A depiction of the logic circuitry used to generate the BAPD and PXCK signals. Note that SMSIE and S1MSIE are intermediate signals, not provided, but required for the BAPD and PXCK signals.**

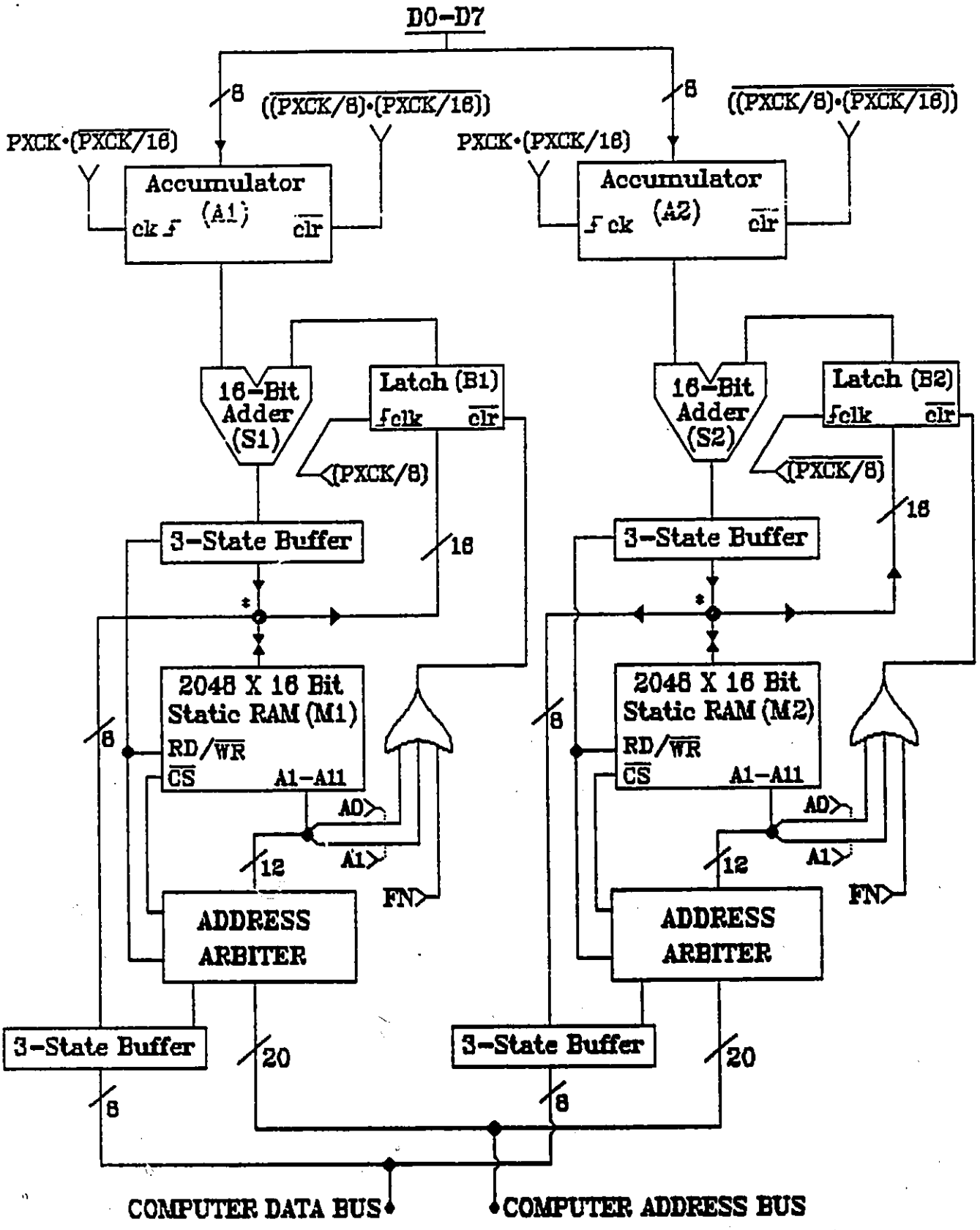


FIGURE 5.3: Flow chart highlighting the data flow through the interleaved memories.

* Division by 64 is accomplished by truncating the least significant 8 data bits at this point.

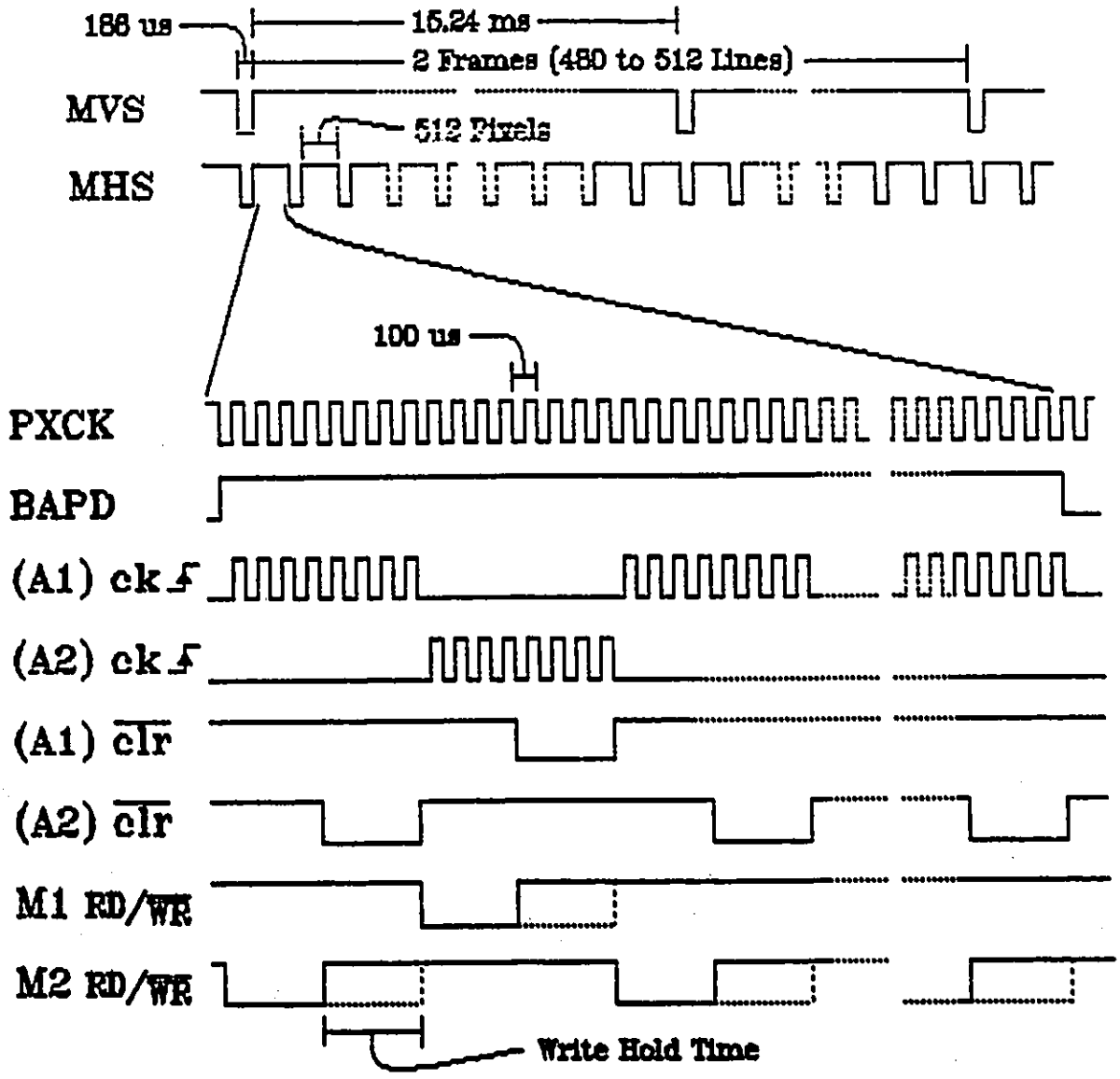


FIGURE 5.4: **An illustration of the timing signals involved in converting the 512 X 512 pixel image from the digitizer to a 64 X 64 pixel image.**

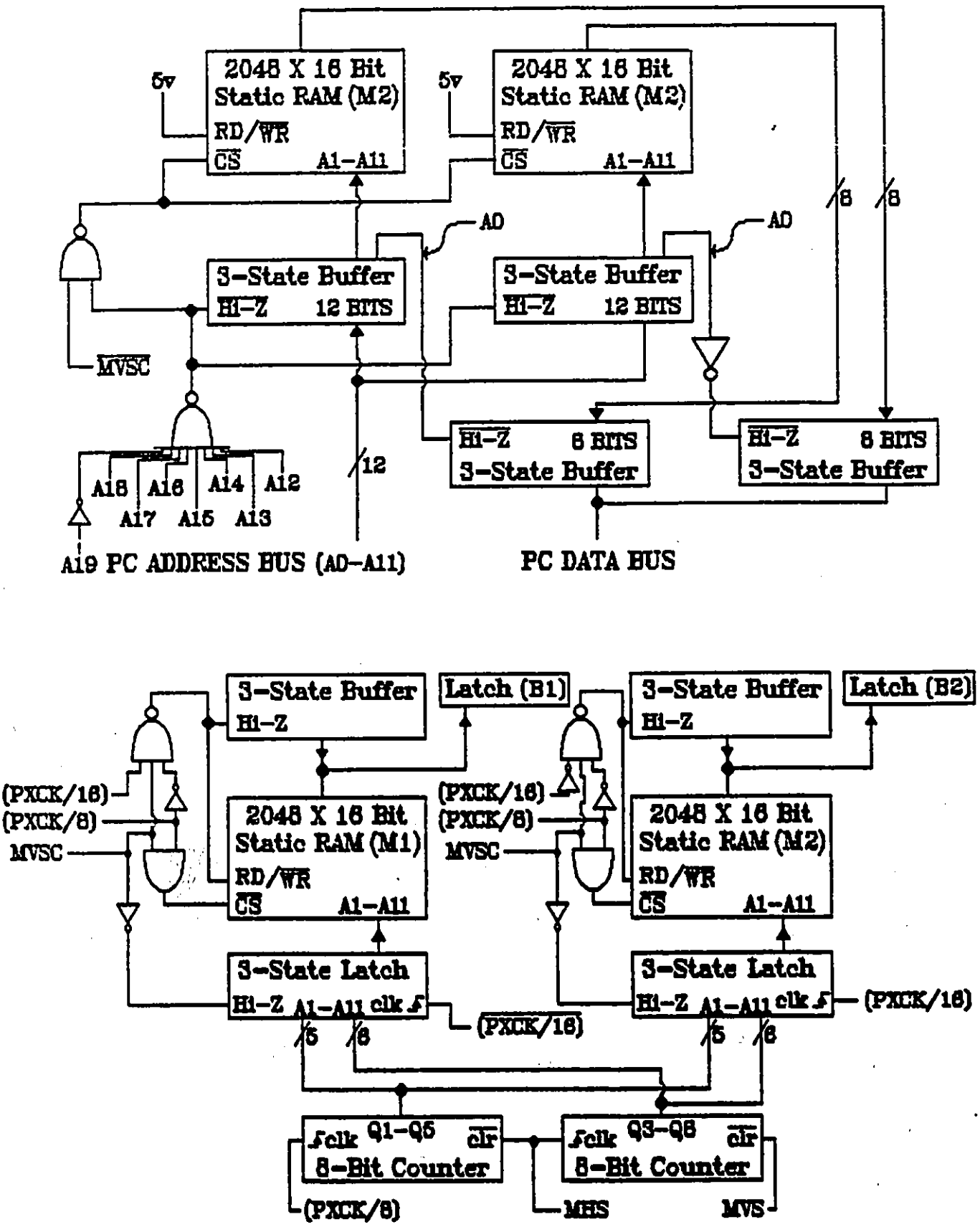


FIGURE 5.5: a) TOP: Memory access for the micro-computer.
b) BOTTOM: Memory Access for the data path.

CHAPTER 6 THE ABSOLUTE ENCODING AND COORDINATE TRANSFORMATIONS FOR THE CONVEYOR SYSTEM

6.1 INTRODUCTION

Proper robotic tracking and grasping of parts delivered by the conveyor systems is possible only if the position of the parts is continuously monitored by the robot's controller. Presence, conveyor position, and vision sensors are used concurrently to measure the required part position. More coordinate systems are involved and consequently some coordinate transformations have to be implemented.

The transformation of the coordinate system of the vision system to the coordinate system of the robot occurs in two phases. The first is a linear displacement along the length of the conveyor belt which moves the image frame from the point at which the

part was detected to a preset location within the coordinate frame of the robot. The second consists of a coordinate system transformation relating the preset image frame position to the origin of the coordinate frame of the robot. These transformation phases are illustrated in Figure 6.1. The purpose of this chapter is to describe in detail the methods used to perform these transformations and essentially relate the appearance and position of a part at one end of the conveyor belt to the position of a robot arm at the other end.

There are a number of ways that the image frame may be shifted from the area of the parts feeders to a place adjacent to the robot arm requesting the part depicted in the image. In an effort to maintain the accuracy provided by the image analysis it is desireable to utilize a method providing the absolute position of the image frame on the conveyor. Also it is necessary to be able to anticipate the arrival of the image frame and thus the part to within the coordinate frame of the robot arm in order that the part may be tracked and grasped. In the following sections a method utilizing the unique properties of pseudo-random binary sequences will be described that provides the absolute position of the image frame when a part is detected in the range of the vision system by the presence sensor and subsequently the position of the image frame at any point along the conveyor belt.

The transformation of the image frame coordinates into the coordinate system of the robot arm may be reduced to the identification of the same three points in each coordinate system. This calibration may be automated and needs only to be performed once during the initialization of the system. The process will be described in later sections and an algorithm for automating the transformation will be highlighted.

6.2 ALGORITHM FOR COORDINATE TRANSFERENCE

The methodology for sensing a part on the conveyor and interpreting the part's location found from using the vision system to the coordinate frame of the robot arm then finally tracking the part with robot arm and ultimately grasping the part is a multi-step process. In previous chapters it was shown that by using the vision system the location of the part within the vision system could be quickly derived. In the next chapters it will be shown that the robot may be made to track and grasp an object given its location(s) relative to the robot's own reference coordinate system. Thus some algorithm for transferring the coordinates referenced to the image frame to coordinates referenced to the robot is necessary.

As stated the transformation is a two phase operation, the first

based upon coded locations on the conveyor and the second a geometric transform. The part's location is identified by and undergoes the following steps:

1. A presence sensor indicates that a part is within the image frame of the vision system.

2. Given the detection of the object the system awaits the confirmation of the next known location on the conveyor then initiates an image frame grab and records the identifying code of the conveyor location (Section 6.3).

3. The image is processed as is the code identifying the address.

4. The location of the part, gleaned from the image, is transformed into the coordinates relative to the robot given that the image frame is in a preset location within the reach of the robot arm (Section 6.4).

5. The codes as they pass by the robot are analyzed in anticipation of the codes just prior to the target code identifying the location of the image frame on the conveyor i.e., the point at which the frame was

grabbed in step 2. These preliminary codes are used to track the part as it is conveyed to the target position in front of the robot arm.

6. Upon detection of the target code at the robot arm, indicating that the image frame now occupies the preset location used for the geometric transformation, the object is grasped and removed from the conveyor belt.

Note that prior to the target code being identified at the robot arm and the part being grasped by the arm the codes detected are directly related to a linear displacement of the part along the conveyor belt from the position at which it is ultimately grasped. This relationship is used to anticipate the arrival of the part and to track it to the position at which it is grasped.

The next section describes the means by which the part is detected and the image frame is tracked as it moves down the conveyor belt to a preset position in the coordinate frame of the robot. The coordinate transformation from the 2-D image space to the 2-D plane perpendicular to the grasping height in the robot's coordinate frame is required. Each of these coordinate systems are Cartesian. The transformation from the Cartesian space of the robot's coordinate frame into the robot's joint coordinate system is discussed for a SCARA type robot in

the next chapter.

6.3 PART DETECTION AND LINEAR DISPLACEMENT

Ideally, the conveyor belt would move continuously at a constant speed. The translation of the image frame from one position on the conveyor to another would be directly related to a displacement in time based upon the velocity of the belt. However, as the weight of parts differs and the power to the motor is not strictly regulated, the speed of the conveyor varies somewhat in practice. An alternate solution is to code positions on the conveyor belt and use the codes to uniquely identify these locations.

Typically, encoding involves either using an incremental encoding method which essentially allows the position to be generated in a cumulative manner based upon the previous positions, or encoding may be performed in an absolute manner. Incremental encoding suffers from a cumulative error degradation but is usually much more economical. Absolute encoding provides a unique identity for each location and as a result it avoids the propagation of errors. The cost of this is fairly high since the code is recorded transversely across the path. Consequently, the number of positions, N, is related to the

number of code tracks, t, as a power of 2:

$$N = 2^t$$

In order to provide 128 different locations on the conveyor, using the typical absolute encoding methods, 7 bit encoding is required, i.e., 7 code tracks must be read and interpreted at each location. This can be fairly costly in terms of hardware and real-estate upon which to record the tracks. To overcome the problems of the classical absolute encoding while retaining the benefits the system described below uses an adaptation of the absolute linear encoding method described for position transducers[1] and mobile robotics[2]. According to these methods, the position encoding is done in an absolute manner but longitudinally instead of transversely thus requiring only one code track. The system designed for the conveyor belt works with 7-bit pseudo-random binary codes providing 127 uniquely identified locations and occupying only two tracks, one for coding and one for synchronization.

The encoding method relies upon the properties of pseudo-random binary sequences (p.r.b.s.'s) generated by direct module-two

---

1 E.M. Petriu, "Absolute-Type Position Transducers Using a Pseudo-Random Encoding," IEEE Transactions on Instrumentation and Measurement, vol. IM-36, no. 4, December, 1987.

2 E.M. Petriu, "Absolute Position Recovery For Automated Path Guided Vehicles", Conference Proceedings Robots 12, Detroit, Michigan, June, 1988.

feedback n-bit shift registers. One such property is that each n consecutive bits form a unique pattern.[1] It is this unique pattern that identifies each location on the conveyor belt. In order to generate the 7 bit p.r.b.s. the following algorithm is applied to a binary seed value of 000 0001:

$$X(0) = X(7) \oplus X(3) \qquad ..(6.1)$$

Where: $X(n) =$ n^{th} bit of the 7-bit feedback shift register for code generation.

A complete listing of the values generated using Equation 6.1 and the seed from above along with their corresponding so-called "natural" positions[2] is included in Appendix C. The sequence is applied to the conveyor beside a synchronization track used to indicate the point at which the data bits are valid. The bits are read sequentially with every 7 consecutive bits forming a unique pattern identifying a single location on the conveyor even though situated at any given location there is only one bit of the 7 bit address. Reading of the pseudo-random n-tuples

---

[1] V.D.T. Davies, System Identification for Self-Adaptive Control, London: Wiley-Interscience, 1970, Chapter 3, pp. 44-88.

[2] Natural positions may be thought of in terms of the normal way of labeling consecutive positions with consecutive natural numbers, e.g., 1,2,3,... whereas p.r.b.s. are not incremental from position to adjacent position e.g., p.r.b.s. code 3 may correspond to a natural position of 5 whereas the natural position 6 may have a p.r.b.s. code of 13.

requires a single code reading head in combination with the synchronization reading head. The synchronization signal marks exactly the location identified by the code track. It is by triggering on the edge of this sync that provides the positional accuracy demanded by the system in keeping with the high degree of accuracy of the coordinates found using the vision system. Figure 6.2 illustrates the tracks and associated absolute positions as well as the code pickups.

The code is read sequentially as the conveyor passes beneath the code pickups located near the projection of the image frame on the conveyor belt as shown in Figure 6.2. The optical sync pickup is used as a clocking signal that latches the code bits picked up by another optical detector serially through a shift register. Each sequence of 7 bits contained within the shift register represents the code identifying the position corresponding to the leading edge of the sync bit. The micro-computer can read these bits as well as the status of the sync bit. The pseudo-random 7-tuple assembled in the shift register does not have a format that is convenient for further use. Another code conversion is required. This code conversion may be performed using a look-up table based upon the values given generated using Equation 6.1. The table provides both forward and reverse translations thus allowing the system to identify any coded location on the conveyor belt as well as locate the codes for adjacent positions on the conveyor belt prior to

detection. This is necessary for the tracking and grasping operations.

Also portrayed in Figure 6.2 are the positions of an infrared transmitter and receiver used to detect a part moving along the conveyor belt. The infrared beam is located just inside the image frame projection on the conveyor belt. Note that it must be located a minimum distance from the edge of the image frame in order that the vision system will pick up the complete image of the part. This minimum distance corresponds to the maximum distance between two consecutive sync pulses. The length of the conveyor used for our system is approximately 325 cm. Using a 7 bit encoding provides for 127 locations with a distance between sync pulses of approximately 2.559 cm (given that they are evenly distributed). As a part on the conveyor belt passes through the beam the micro-computer awaits the next sync pulse. Upon detection of the leading edge of this sync bit the micro-computer initiates a frame grab and reads the 7 bit code from the shift register. The micro-computer now has a digital image to process and also has the code corresponding to the absolute position of the image projection on the conveyor belt required for steps 1,2, and 3, of the algorithm presented in section 6.2. This absolute position may be regarded as a "base address" of the image relative to the conveyor reference system.

There are two possible ways in which the coding may be

interpreted to displace the projection of the image frame from beneath the camera to a position within the reach of the robot arm. It is possible to accurately measure the distance required for the part or image frame to travel in order to reach a position in the range of the robot arm as a number of the positions identified on the conveyor belt. Then it would only be necessary to continue to monitor the 7 bit codes until the code corresponding to the specified distance from the position of the target code. However, although this method requires only one code reader for the entire conveyor it demands that the distance between the sync pulses be uniform and known and as well that the shape of the leading edge for each sync pulse be uniform. Alternatively a second code reader with optical sync pickup can be deployed at a preset location in front of the robot arm. In this case it is only necessary to ensure that the sync pick-ups are identical i.e., that they both have the same threshold and thus trigger at the same point along the leading edge of the sync pulse. The system adaptation uses a code reader at each position. This is seen as being more accurate for the reasons previously stated as ultimately allowing for a dedicated processing unit at the robot arm to control the tracking and grasping functions separate from the parts feeder and image processing processor.

The design of the infrared presence sensor and optical pick-ups to be used in conjunction with the image system is illustrated

in Figure 6.3. The design of the optical pick-ups located at the output of the conveyor is shown in Figure 6.4. The data generated using these circuits is read by the micro-computer using a set of I/O ports. Note that it is possible providing that the digitizer used may be slaved to an external device, to initiate the frame grab directly upon detection of the sync pulse following the detection of a part by the presence sensor. This has not been implemented with the Matrox PIP 512 digitizer however the circuit provides for interfacing with the appropriate logic if this level of automation is desired. Figures 6.3 and 6.4 are similar in that each circuit uses two TIL 139 optical transceivers to detect the code and sync bits on the conveyor belt. These sync detector clocks the data from the code detector through an 8 bit serial to parallel shift register. The 7 LSB's assembled in parallel represent the 7 bit PRBS code for each location. These may be passed to the computer via an addressable 3-state latch or transparent buffer. The circuit of Figure 6.3 provides for a slightly different operation. Prior to the transfer of a part from the parts feeder to the conveyor the computer resets the circuit in Box #1 by clearing the J-K flip-flop using Output 1. The Q output of the flip-flop is then a logic 0 and thus so is Input 8. Input 8 is monitored by the computer and a frame grab is initiated when the status of Input 8 changes to a logic 1. This occurs when the part on the conveyor breaks the infrared beam of the presence detector followed by a low to high transition of the

sync signal. This changes the state of the Q output to a logic 1 and locks the data from the shift register into the 3-state latch. This data is the PRBS code representing the address of the image frame just grabbed. Thus box #1 provides a signal indicating when the image frame is to be grabbed and also saves the coded location of the frame grabbed.

Using the codes from the conveyor belt part after part may be transferred to the robot arm. To be noted is that a variety of methods, both hardware and software may be incorporated to detect any errors in the reading of the code. These include the monitoring of present and previous codes at each point to ensure that they represent consecutive natural locations on the conveyor. If the codes are read continually in this manner it is possible to detect any errors in the reading of the code bits and to correct for them until the shift register, after 7 consecutive synchronization pulses, purges the bit in error. Another method involves checking the next bit in advance using the bit generation function. This may easily be implemented in hardware if desired. Of note is that errors if they occur are recoverable and correctable without extraordinary measures or a resetting of the system to a known configuration. Start-up is as simple as reading 7 consecutive bits correctly. This is considered to be far superior to most other methods of linear encoding.

By encoding the conveyor belt in the manner described and correctly interpreting these codes it is possible to displace the image frame linearly along the length of the conveyor belt to a position within the coordinate frame of the robot. The system using multiple stations, one dedicated to the imaging system and the others to robot arms along the length of the conveyor belts provides for easy implementation of a more distributed processing architecture.

6.4 THE COORDINATE SYSTEM TRANSFORMATION

The transfer of coordinates from the 2-D Cartesian space of the image frame located in a preset position within the 2-D Cartesian coordinate space associated with the robot arm involves both a translation of origins and possible rotation of axis. The situation to be examined is illustrated in Figure 6.1 which shows the image frame after its displacement from beneath the camera to a position within the reach of the robot executed according to the coding algorithm of the previous section. Much as it would be convenient to align the axis of the image frame with the axis of the robot's coordinate frame, it would require extreme (and expensive) positioning control for both the camera and robot. The translation and rotation is not a difficult problem requiring only that the system analyze three reference

points prior to the start of actual operation. From the analyses of these three controlled points the functions relating the two coordinate systems may be derived as shown below.

Consider Figure 6.1 which depicts the coordinate systems of from a more mathematical point of view where O is the origin of image frame whose axis are represented by X & Y and O' is the origin of the robot's coordinate frame with axis U & V. The origin O of the X-Y system may be represented as a point (u_0,v_0) relative to the U-V system. Points (x,y) in the X-Y system are represented as points (u,v) in the U-V system using the following relationships[1]:

$$u = l_1*x + l_2*y + u_0 \qquad ..(6.2)$$

$$v = m_1*x + m_2*y + v_0 \qquad ..(6.3)$$

where: l_1,m_1; l_2,m_2 are the direction cosines of the X,Y axis relative to the U,V axis.

The direction cosines can be computed using two reference points that occur in both coordinate frames i.e., given two different values for x and y corresponding to known values for u and v it is possible to solve for l_1 and l_2 using Equation 6.2 provided u_0 is known. Similarly m_1 and m_2 may be computed from Equation

[1] Freely adapted from Murray R. Spiegel, Mathematical Handbook of Formulas and Tables, McGraw-Hill Inc., 1968.

6.3 given v_0. However the point (u_0,v_0) is typically unknown as well. Therefore a third reference point is required and Equations 6.2 and 6.3 are solved as two sets of three equations with three unknowns. These values remain constant while the system is in operation.

The computation of the relationship between the two Cartesian coordinate systems needs only to be performed once given stability in the positions of the reference frames. In other words provided that the camera, conveyor, and robot arm(s) are solidly mounted the system can be initialized once upon start-up and then during regular parts presentation operations the system can translate any coordinates in the vision frame to the robot frame via the relationships given in Equations 6.2 and 6.3 using the constants calculated off-line during initialization.

6.5 CONCLUSION

Ultimately the goal of a parts presentation system is to deliver requested parts, accompanied by their positional parameters with a desired degree accuracy, for further robotic handling. In order to provide the accuracy demanded by robotic assembly systems some elaborate methods of measurement are required.

The use of a vision system as described has been demonstrated to provide very accurate information about an object within its image frame. Furthermore, vision can check the integrity and identity of various objects using a variety of means such as template matching and chain codes.[1] Note that it is not within the scope of this thesis to present a series of image processing algorithms to be used for parameter extraction but rather to provide an architecture for a vision-based parts presentation system fitting meeting the requirements of the task-level language outlined in Chapter 2. The system allows the user to specify and design the internal routines desired for parameter extraction given a specific experiment.

The absolute position recovery described for the conveyor system allows the system to maintain the accuracy even when the object is displaced from the image frame after a frame grab. The absolute encoding of the conveyor also provides the possibility of tracking the object as it moves along the conveyor belt.

---

[1] Dana H. Ballard and Christopher M. Brown, Computer Vision, Prentice-Hall Inc., Englewood Cliffs, New Jersey, (1982).

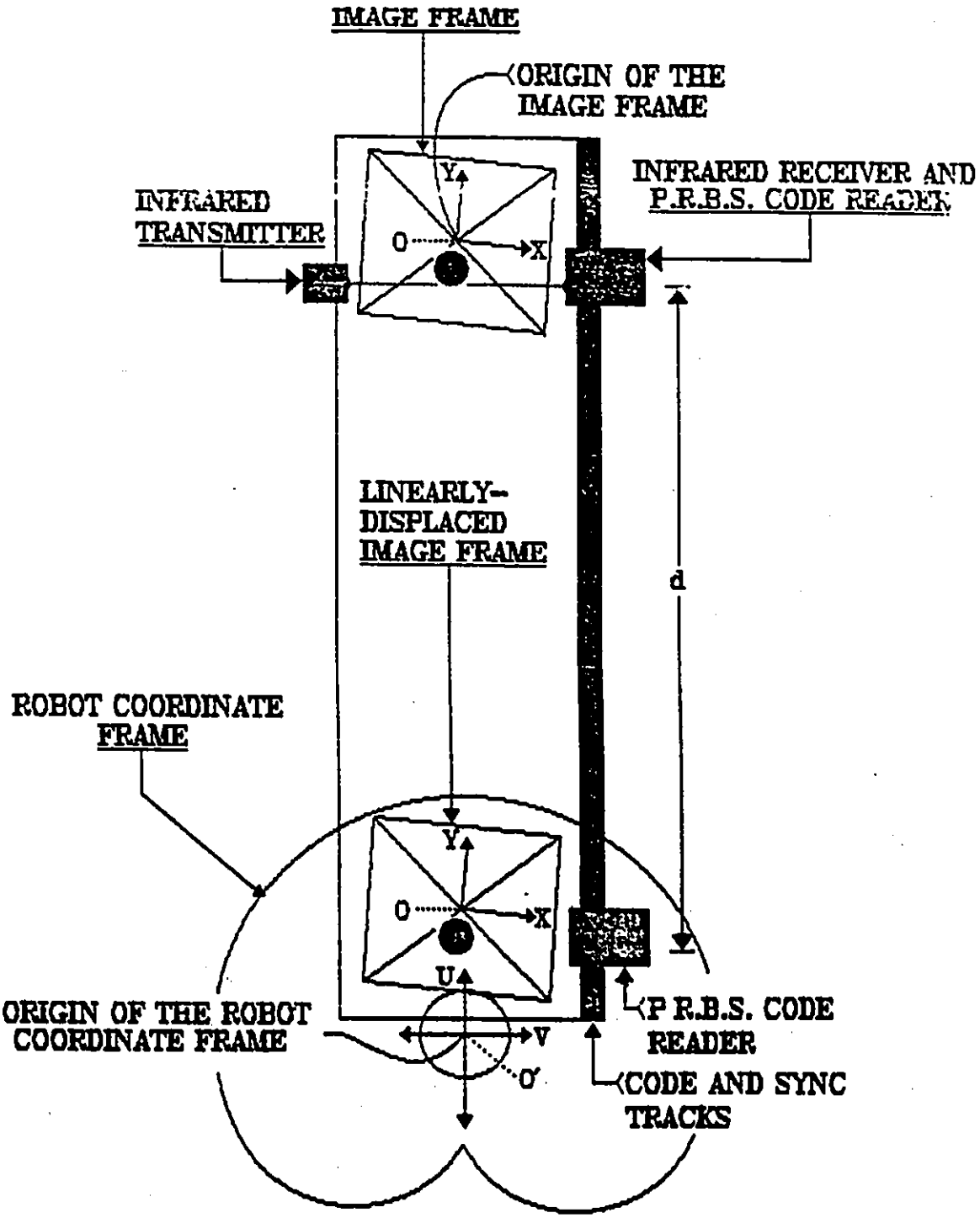
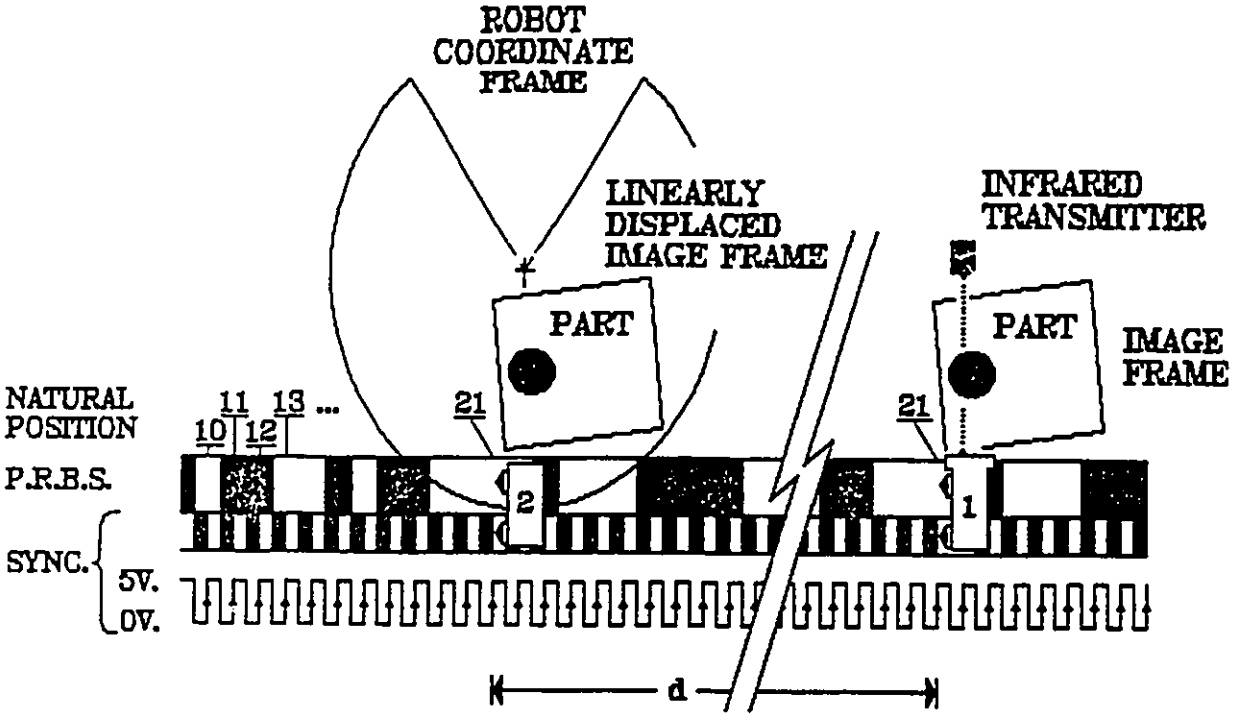


FIGURE 6.1: An illustration of the coordinate transformations required. The part is detected when it breaks the beam. The image frame is displaced a distance d, and the object is related to the robot after the displacement and the image analysis.



BOX #1. Optical code reader and infrared receiver. This is situated towards the top of the image frame. When the part is detected, BOX #1 awaits the next positive going edge of the SYNC (a black to white transition) then initiates a frame grab and records the address detected on the PRBS code track.

BOX #2. Optical code reader. This is situated adjacent to a robot arm. The addresses detected are used to anticipate the arrival of the part, i.e., to displace the image frame linearly along the conveyor belt.

FIGURE 6.2: A graphic depiction of the code readers and the displacement of the image frame accomplished through the interpretation of the PRBS code.

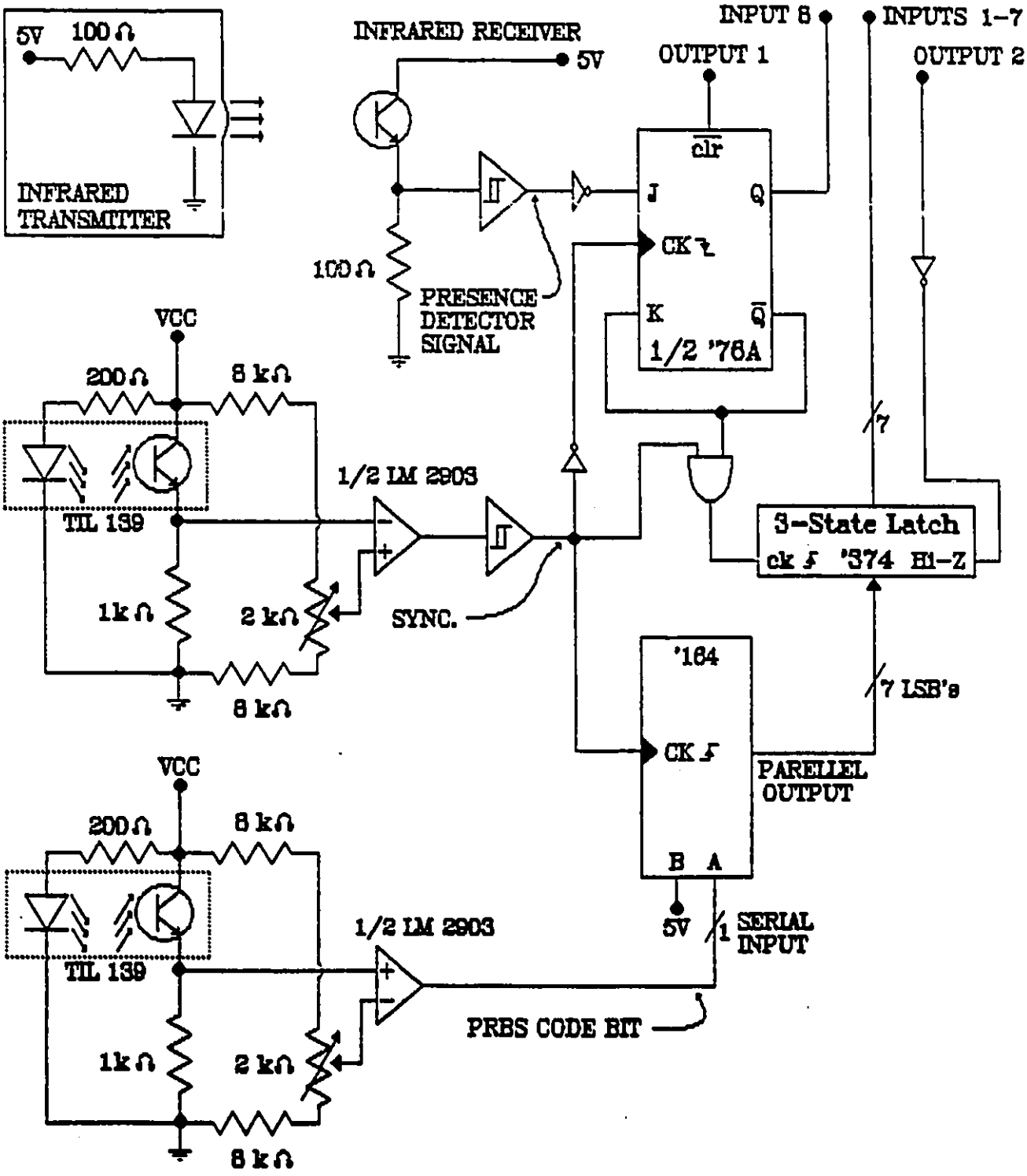


FIGURE 6.3: **A schematic showing the circuit of Box #1 from Figure 6.2 which is associated with the image system.**

Note: Inputs and outputs are associated with an I/O port in the computer.

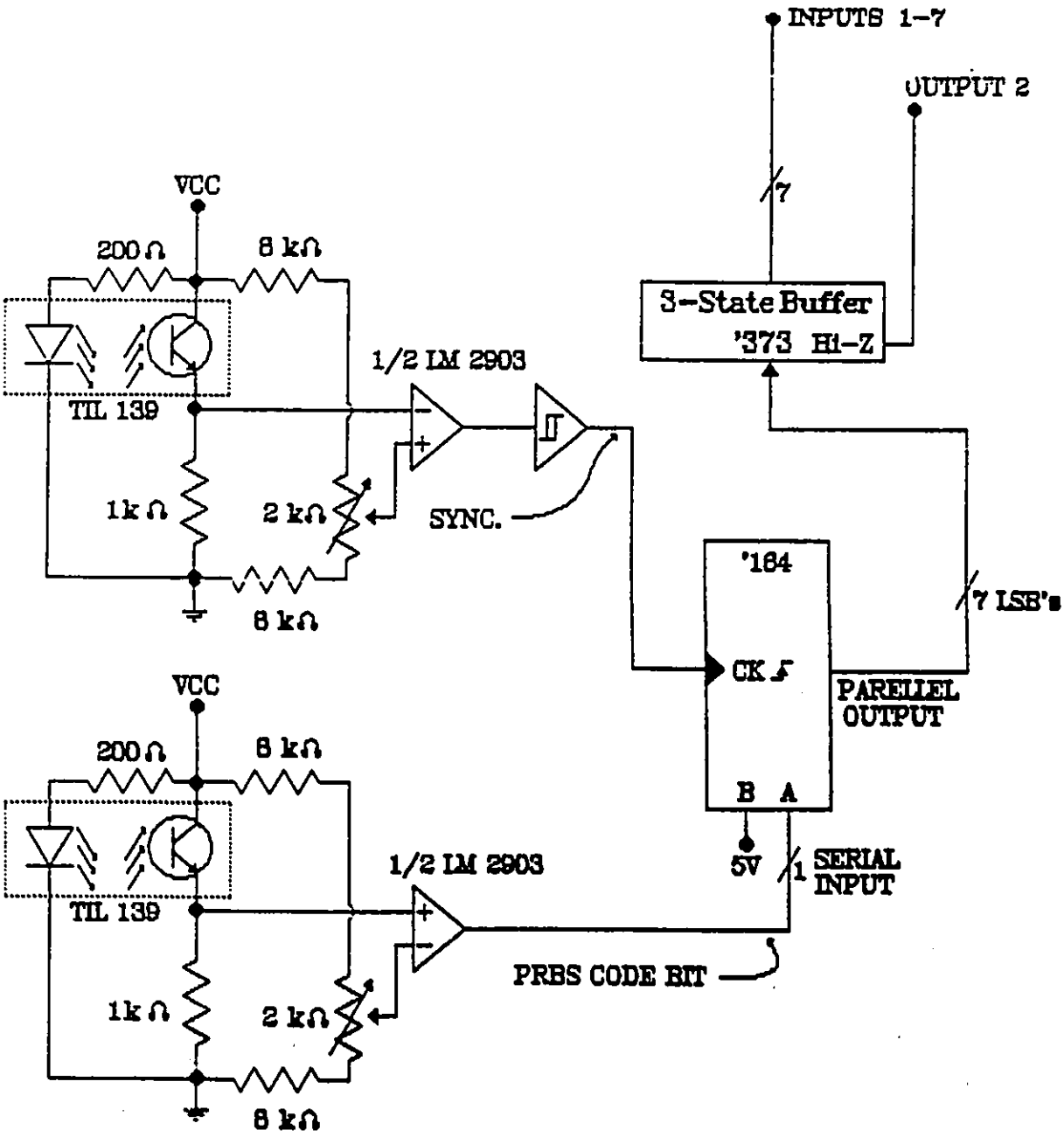


FIGURE 6.4: **A schematic showing the circuit of Box #2 from Figure 6.2 which is associated with a robot arm.**

Note: Inputs and outputs are associated with an I/O port in the computer.

CHAPTER 7 ROBOTIC TRACKING AND GRASPING

7.1 INTRODUCTION

The final step in the Parts Presentation System's operation is the situation of the part into a known position within the reach of the robot arm. In some cases this is completed when the conveyor moves the part into the pre-determined position in front of the robot arm at the opposite end of the conveyor if the robot arm is part of the Assembly Processor. However, we believe that using a single robot arm attached to the Parts Presentation System to feed parts form the conveyor to a set on Assembly Processors located adjacent to the robot arm is more efficient in terms of time and control. This allows for the use of a specialized robot designed for easy control in one plane simplifying the tracking and grasping operations while allowing

another arm of a different configuration to perform the assembly operation.

Thus this chapter describes the kinematic equations for a SCARA type robot arm employed specifically for part tracking and grasping. This includes a definition of the robot's joint space and the rcbot's Cartesian coordinate frame as well as the formulation of the relationships between points defined in each system, the forward and inverse kinematic equations.

This robot arm will be made to move through a series of points until the part is finally grasped by the end-effector. These points together form a line parallel to the direction of the conveyor belt and the movement of the end-effector along this line at the speed of the conveyor belt corresponds to the tracking phase. When the code sent to the robot arm is the same as that for the image frame the grippers close about the part and it is removed from the conveyor belt. This corresponds to the last reduction in the positional entropy for the part and it then is handed to the Assembly Processor.

7.2 THE FORWARD KINEMATIC TRANSFORMATIONS

The robot for which these transforms have been designed is Rhino Robots Inc. XR Series SCARA Robot illustrated in Figure 7.1. It provides for a minimum resolution of 1 mm and a repeatability of 1 mm with a maximum payload of 1 lb. The SCARA (Selective Compliance Assembly Robot Arm) robots are designed with at least one translational joint, in this case providing a vertical movement, and multiple rotating joints. The robot used has 4 degrees of freedom, three for the placement of the end-effector and a wrist rotation for turning the end-effector. The translational movement is up/down allowing for the selection of a horizontal plane and two rotating joints providing the movement on that plane. Figure 7.2 illustrates the action of these joints with respect to a Cartesian coordinate system originating at the base of the robot.

The kinematic transforms may be rigorously defined using the notation of Paul[1], as explained by Snyder[2], however for the robot examined herein this is considered unnecessarily bulky since there are only 4 degrees of freedom to be examined.

---

[1] R.P. Paul, Robot Manipulators, Cambridge, Mass., MIT Press, 1981.

[2] Wesley E. Snyder, Industrial Robots: Computer Interfacing and Control, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1985.

Rather it is much easier to comprehend the kinematic relations as a set of geometric transformations related algebraically.

Consider first the forward kinematic equations which describe the position and orientation of the end effector in Cartesian space in terms of the positions of the joints of the robot arm. The most obvious observation from examination of Figure 7.2 is that the horizontal (X-Y) plane occupied by the end effector is simply a function of the position of the prismatic joint located near the end effector. Furthermore the position in that plane is a function of the angles of the first two rotational joints, $\tilde{a}_A$[1] and $\tilde{a}_B$ as illustrated in Figure 7.3, a topside view of the robot arm in a variety of configurations. Finally the orientation of the gripper is generated by a rotation $\tilde{a}_C$ and is a function of all three angles, $\tilde{a}_A\tilde{a}_B\tilde{a}_C$.

Consider the position of the end effector in the X-Y plane defined by the position of the of the prismatic joint. The end of any link n in a 2-dimensional Cartesian space with origin at the centre of rotation of joint n is defined as:

$$X_n = d_n * \text{Cos}(\tilde{a}_n)$$
$$Y_n = d_n * \text{Sin}(\tilde{a}_n)$$

---

[1] Note ã refers to the Greek letter alpha.

Where: d_n is the length of link n;

$\bar{a}_n$ is the angle of rotation of joint n;

X_n is the X coordinate of the end link n defined in a 2 dimensional Cartesian space with origin through joint n;

Y_n is the Y coordinate of the end of link n in the previously described Cartesian space;

Note that the distances of each link, the angles of rotation allowed for each joint and the travel permitted in the movement of the prismatic joint for the Rhino SCARA robot are provided in Table 7.1 which includes other specifications of interest.

For joint A the origin of its Cartesian space is the same as that for the robot arm. The origin of the Cartesian space of joint B is at the end of link A. It is thus displaced from the origin of the robot arm by X_A in the X direction and Y_A in the Y direction. Thus the end of link B may be defined in terms of the Cartesian space of the robot arm as:

$$X_R = X_A + X_B$$
$$Y_R = Y_A + Y_B$$

Where: X_R is the X coordinate of the end effector related to the robot arm's coordinate space;

Y_R is the Y coordinate of the end effector related to the robot arm's coordinate frame.

Thus: $X_R = d_A * \text{Cos}(\tilde{a}_A) + d_B * \text{Cos}(\tilde{a}_A + \tilde{a}_B)$ ..{7.1}

$Y_R = d_A * \text{Sin}(\tilde{a}_A) + d_B * \text{Sin}(\tilde{a}_A + \tilde{a}_B)$ ..{7.2}

And: $Z_R = Z_C$ ..{7.3}

Using Equations 7.1, 7.2 and 7.3, given the angles of rotation of joints A and B along with the position of the prismatic joint, joint C it is possible to calculate the position of the robot arm with respect to the robot's 3-dimensional Cartesian space. The orientation of the end effector is a rotation, as shown in Figure 7.3, about the Z- axis at the end point of joints A and B. This may be defined with respect to the X-axis of a 2-dimensional Cartesian coordinate system located at that point parallel to the robot's coordinate frame as shown in Figure 7.4 as:

$$^{D}\tilde{a}_R = \tilde{a}_D + \tilde{a}_A + \tilde{a}_B \text{ ,} \qquad \text{..{7.4}}$$

Thus the position of the robot arm may be found with respect to the robot's coordinate system and the orientation of the robot arm may be considered as a rotation about this point subject to an offset caused by the displacement which in turn may be computed using Equation 7.4.

Equations 7.1 through 7.4 define the forward kinematic transformations used for a SCARA type robot arm. Though useful for the initialization procedure where it is necessary to relate 3 arbitrary points to the position of the robot arm these relations are otherwise of very little concern in the operation of the robot as the main problem in robotics applications is not the interpretation of the joint positions as a point in some pre-defined Cartesian space but rather the problem is the reverse. It is very important to be able to compute the joint positions as a function of some Cartesian point. This is also a more difficult operation because there is a many to one relationship that exists between points in joint space and points in Cartesian space. That is, a Cartesian point may be accessed by a number of configurations of the robot's joints thus there is no unique solution for the inverse kinematic transformations. The actual number of configurations possible is directly related to the number of rotational joints in the robot. In the next section the inverse kinematic transformations are examined with special emphasis upon further defining the coordinate systems of the robot arm in order to reduce the solution of the inverse kinematics to a unique set of joint positions.

7.3 THE INVERSE KINEMATIC EQUATIONS

As a result of the coordinate system definitions of the previous chapter as well as the method of describing the transformations the inverse kinematic relationships are surprisingly simple in form for our robot. Given one restriction in the formation of the joint positions it is possible to establish a one to one ratio between the points in Cartesian space and those of the joint space. Thus a unique solution is possible for the inverse kinematics for every point within the reach of the robot arm. This restriction will be derived in the following paragraphs along with a loose graphical proof of the one to one relationship established and then the inverse kinematic equations growing out of this restricted system will be formulated. These equations are designed as the best for the situation of the robot picking a part from the conveyor belt with no obstructions in its motion with the possible exception of a height limitation affecting only the prismatic joint.

The multiple joint positions resulting in a single Cartesian position come as a result of the two rotational joints allowing for rotations in the X-Y plane only. The Z axis is defined uniquely by the position of the prismatic joint. However there are also multiple solutions in joint space for some desired orientations in Cartesian space as highlighted in Figure 7.5.

Note that there are at least two solutions for most of the points in the Cartesian space. In order to find a unique solution for inverse kinematic transformations the following restriction is applied. Consider that for each point in Cartesian space defined by a positive X_R value the value of $\tilde{a}_A$ is restricted to a positive value. Similarly if X_R is a negative value so too must $\tilde{a}_A$. Thus every position Cartesian space has a unique solution in joint space. This applies to position only. For orientation the rotation of joint D is restricted to a clockwise rotation from its origin, positive $\tilde{a}_D$, when X_R is positive and counter-clockwise, negative $\tilde{a}_D$, when X_R is negative. This results in a unique solution for orientation as shown in Figure 7.6.

These restrictions have not been chosen in an arbitrary manner. They are peculiar to the Rhino SCARA robot with its characteristic operating envelope as shown in Figure 7.7. As an explanation consider that each rotational joint has certain constraints preventing operation through 360^{o}. As a result points close to the origin of the X-Y plane may be reached using only one configuration of the joints. The restrictions applied above offer possible joint space solutions to all points within the operating envelope. As well the configuration is not affected by the travel of the object on a path parallel to the X_R axis. The solutions are stable for all points along the path. The only detrimental effect of these restrictions is the

reduced versatility of the arm in avoiding obstacles in the operating range. If for example a partial barrier of some sort exists in the 1st quadrant (positive X, positive Y) then points in the second quadrant are inaccessible as the barrier with obstruct the movement of the arm in the first quadrant. These points would otherwise be accessible, because of the multiple configurations provided by the multiplicity of unrestricted joint space solutions. This is not a problem in this case since there are such obstructions.

The solutions are now fairly simple to derive. Using the coordinate system conventions of the previous section consider a point defined in Cartesian space (X_R, Y_R, Z_R). As previously stated:

$$Z_C = Z_R \qquad ..(7.5)$$

In order to find the values for $\tilde{a}_A$, $\tilde{a}_B$, and $\tilde{a}_D$, with reference to Figure 7.8, the case of $X_R \geq 0$ the solution for the angles of a triangle with sides of known length must be computed. The hypotenuse of this triangle is line segment joining the origin with the point $P(X_R, Y_R)$. The length, b is found using the following relationship:

$$b = [(X_R)^2 + (Y_R)^2]^{\frac{1}{2}}$$

And the angle this vector makes with respect to the X_R axis, Q_R

is found using:

$$Q_R = \text{Tan}^{-1}[Y_R/X_R] \ , \ X_R <> 0$$
$$= 90^o \ , \ X_R = 0$$

The other two sides of this triangle are formed by the segments of the robot arm between joints A and B and Joints B and C. Conveniently for the Rhino SCARA these lengths are equal further simplifying the solutions. First the semi-perimeter, s of the triangle is found[1]:

$$s = 1/2(a + b + c)$$

Where:

a is the length of the segment connecting joints B and C;

c is the length of the segment connecting joints A and B.

The interior angle A of the triangle is found using:

$$A = \text{Sin}^{-1}\{[2/(b*c)]*[s*(s-a)*(s-b)*(s-c)]^{\frac{1}{2}}\}$$

Angles B and C are found in a similar manner. From Figure 7.8

---

[1] The equations for the triangular solutions have been adapted from Murrey R. Spiegal, Mathematical Handbook of Formulas and Tables, McGraw-Hill, USA, 1968.

it is clear that $\tilde{a}_A$ is simply a function of A and Q_R:

$$\tilde{a}_A = Q_R - A \text{ degrees} , Y_R \geq 0 \qquad ..(7.6)$$

and $\tilde{a}_B$ is a function of B:

$$\tilde{a}_B = 180 - B \text{ degrees} , Y_R \geq 0 \qquad ..(7.7)$$

Finally the orientation of the end effector or the grippers is specified as a fourth coordinate, an angle referenced to an axis parallel to the X_R axis through the endpoint of the robot arm (through joints C and D). This angle ${}^R\tilde{a}_D$ is related to the angle of rotation of the joint D, $\tilde{a}_D$ as:

$$\tilde{a}_D = {}^R\tilde{a}_D - (\tilde{a}_A + \tilde{a}_B) \qquad ..(7.8)$$

Equations 7.5 through 7.8 are collectively the inverse kinematic transformations for the restricted operating range of the Rhino SCARA robot when $Y_R \geq 0$. When $Y_R < 0$ there are two differences which result from the fact that the interior angles of the triangle are always positive even when they exist in a negative quadrant since they are not referenced to the coordinate system. As a result Equations 7.6 and 7.7 must be adapted to reflect the proper condition i.e.,

$$\tilde{a}_A = Q_R + A \text{ degrees} , Y_R < 0 \qquad ..(7.9)$$

and: $\tilde{a}_B = B - 180 \text{ degrees}, \; Y_R < 0$ ..(7.10)

Thus Equations 7.5 through 7.10 specify completely the inverse kinematic transformations required in order to position the Rhino SCARA robot in the restricted Cartesian space previously defined. These equations are used to determine the appropriate angles through which the joints of the robot arm must rotate to arrive at a position specified in Cartesian space.

The joints themselves move in discrete steps of known size. The step sizes for each joint are also provided in Table 7.1. The result of this type of motion is that the number of Cartesian points is limited since the number of joint configurations is finite. More importantly generally speaking, the robot cannot actually be configured to a distinct Cartesian position but rather only an approximation of this position. There exists a definable error between the desired position of the arm and the actual position into which it moves. This error is bounded at all times by value of one half the resolution of each joint and may be calculated using the forward kinematic transformations. For example Table 7.1 indicates that joint A has a resolution of 1/8.5 degrees per step of rotation. Similarly joint B's resolution is 1/4.4 degrees per step; joint C, 1/7.1 mm per step; and joint D, 1/4.4 degrees per step. The worst error in the positioning of the robot's end effector, $^{E}Z_R$, at some Z_R is found using Equation 7.3:

$$Z_R = Z_C$$

The error in the position of Z_C is bounded as follows:

$$0 \leq {}^{E}Z_C \leq 0.5 * 1/7.1$$

Thus : $${}^{E}Z_R \leq 0.0704 \text{ mm}$$

Similarly using Equations 7.1 and 7.2 the errors in the X_R and Y_R coordinates are found as:

$$0 \leq {}^{E}\tilde{a}_A \leq 0.5 * 1/8.5 \text{ degrees}$$

And: $$0 \leq {}^{E}\tilde{a}_B \leq 0.5 * 1/4.4 \text{ degrees}$$

Thus when the arm is fully extended:

$${}^{E}X_R \leq 457.8 - [228.6 * \text{Cos}(1/17) + 228.6 * \text{Cos}(1/17 + 1/8.8)] \text{ mm}$$

$${}^{E}X_R \leq 0.60 \text{ mm}$$

And:

$${}^{E}Y_R \leq 228.6 * \text{Sin}(1/17) + 228.6 * \text{Sin}(1/17 + 1/8.8) \text{ mm}$$

$${}^{E}Y_R \leq 0.923 \text{ mm}$$

Finally the error for the hand rotation, ${}^{ER}\tilde{a}_D$, is:

$${}^{ER}\tilde{a}_D \leq {}^{E}\tilde{a}_D + {}^{E}\tilde{a}_A + {}^{E}\tilde{a}_B \text{ degrees}$$

$${}^{ER}\tilde{a}_D \leq 0.286 \text{ degrees}$$

These values provide the knowledge required to formulate an attempt at grasping a part as they provide the necessary tolerances which must be observed when moving the end effector near the part. Also, the actual path traced by the end effector only approximates the desired path. This is not a problem as

long as the compliance allowed by the end effector is greater than the path error which in the case of the Rhino robot used implies that the grippers must be open enough that the space enclosed is greater then the width of the part plus the tracking error.

7.4 PATH CONTROL

There exist a variety of methods used to guide the end effector along a specific path. However most of these methods rely upon some form of control over the speed of the joints. This is not possible with the Rhino SCARA controlled using the Rhino XR controller. Snyder's algorithm[1] for joint interpolated motion provides a basis for what is referred to as joint interpolated control which essentially provides timed point to point transitions in the positioning of the end effector. An adaptation of his algorithm presented herein ha been designed for the case of the Rhino SCARA tracking a part along a conveyor belt.

First the operator specifies a minimum allowable path deviation

---

[1] Wesley E. Snyder, Industrial Robots: Computer Interfacing and Control, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985, pp. 193-197.

for the robot arm. As well the starting and ending points must be provided. Typically the path deviation allowed will be approximately 1mm in either the X_R or Y_R directions. This is the advertised resolution of the Rhino and has been confirmed in the previous section. Also the maximum deviation of ${}^{D}\tilde{a}_R$ is chosen to be 0.5 degrees. Given these values and those of the endpoints of the robot's travel in Cartesian space the joint positions are calculated for each of these two points. The midpoint of the two endpoints is determined in joint space. The forward kinematic transformation is computed for this midpoint using the values determined from the midpoint of joint space. The Cartesian point determined is compared to the actual Cartesian midpoint determined from the two Cartesian endpoints. If the difference in the two midpoints is less than the specified allowable deviations then the robot is driven from point to point. If not then the segment is quartered and the calculations are repeated for each quarter. This continues until the bounds of the path's intermediate points are within the allowable deviations. At this point the robot is driven via the intermediate points from the specified starting point to the specified endpoint. According to Snyder this algorithm converges rapidly i.e., the deviation is typically reduced by a factor of four with each bisection of the segment.

In terms of part tracking the robot's end effector will be made to drive along a line segment of approximately 10 cm in length

parallelling the direction of the conveyor belt and the X_R axis with the end point located at the point where the part is to be removed from the conveyor belt. The speed of the motion is controlled by monitoring the conveyor belt codes which change about every 2.5 cm. Figure 7.9 is a flowchart of the tracking and operation employing the point to point path control described above and an example of a program using the kinematic transformations may be found in Appendix D.

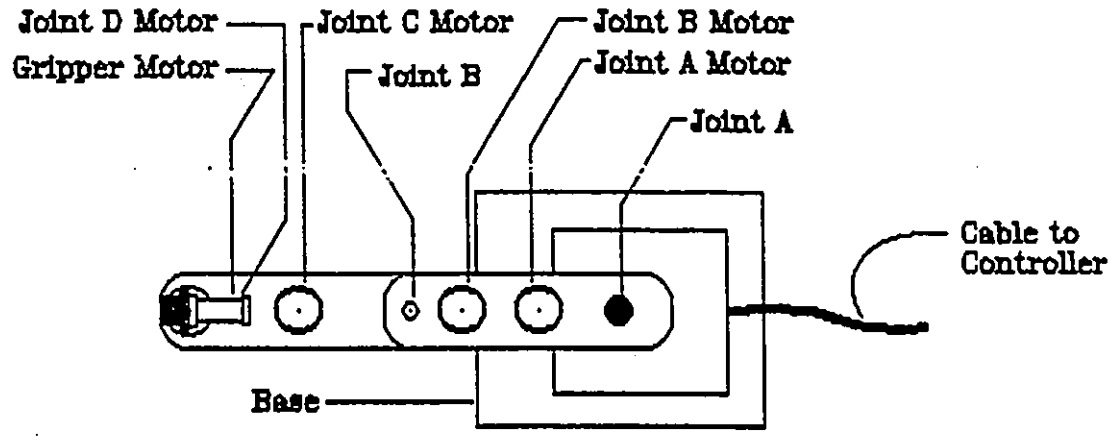
7.5 CONCLUSION

Using the equations derived in the previous sections the robot may be moved from point to point in Cartesian space. The joint speed of the individual joints is not directly controllable and thus if a path is to followed at a specified rate the end effector must be driven through a set of intermediate points with associated time delays at each point as well as a restriction in the deviation from the path allowed.

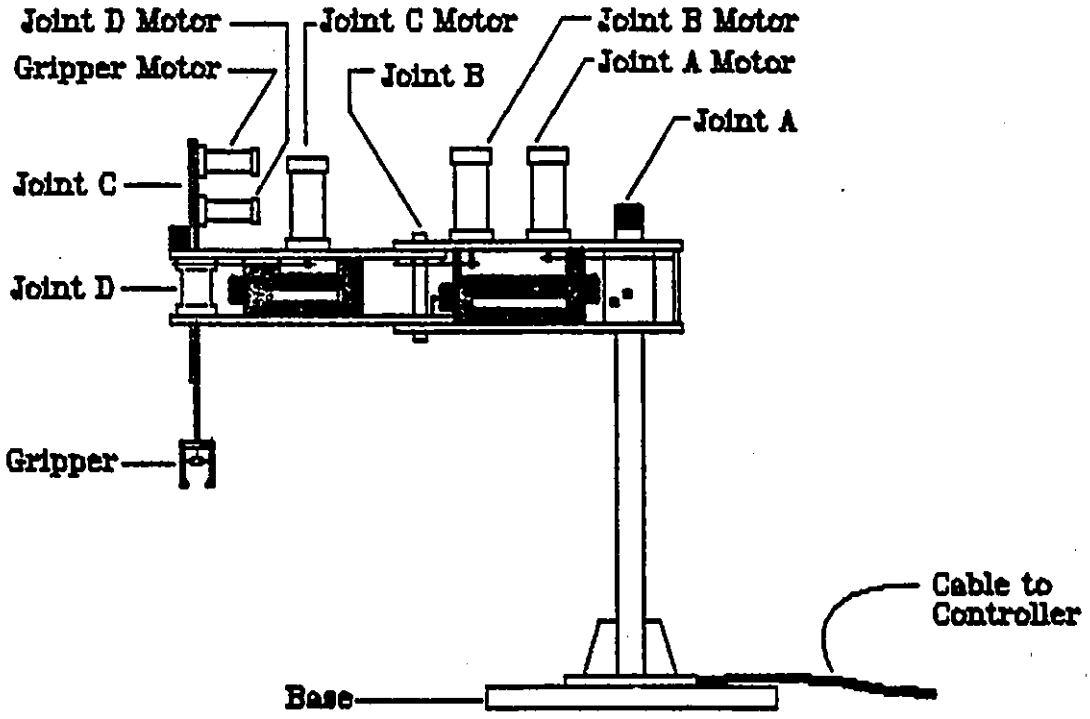
The kinematic relations as presented herein reflect the geometric relationships between the Rhino SCARA robot's restricted joint space and a Cartesian coordinate system with origin located near the base of the robot arm. It is to be noted that while the equations derived herein are not in a more

formal format they provide the basis for any formal description of the system and yet do not require the more complex functions demanded by matrix notation. This simplifies the computations especially since there are only 4 degrees of freedom, one of which is a translation as opposed to a rotation.

It is left to the system user to design an alternate control routine if desired and in fact experimentation with different control systems as well as with alternate restrictions upon the joint space is encouraged.



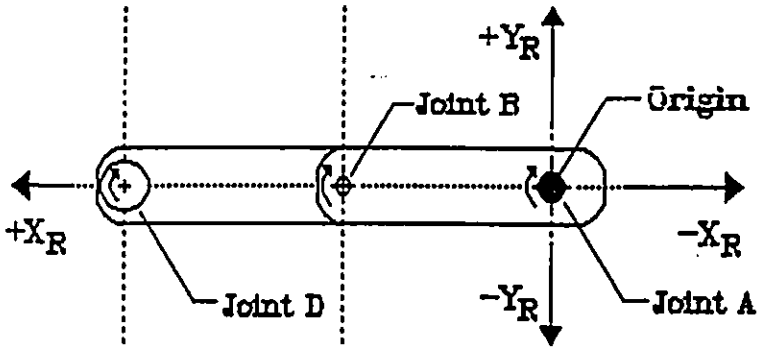
TOP VIEW



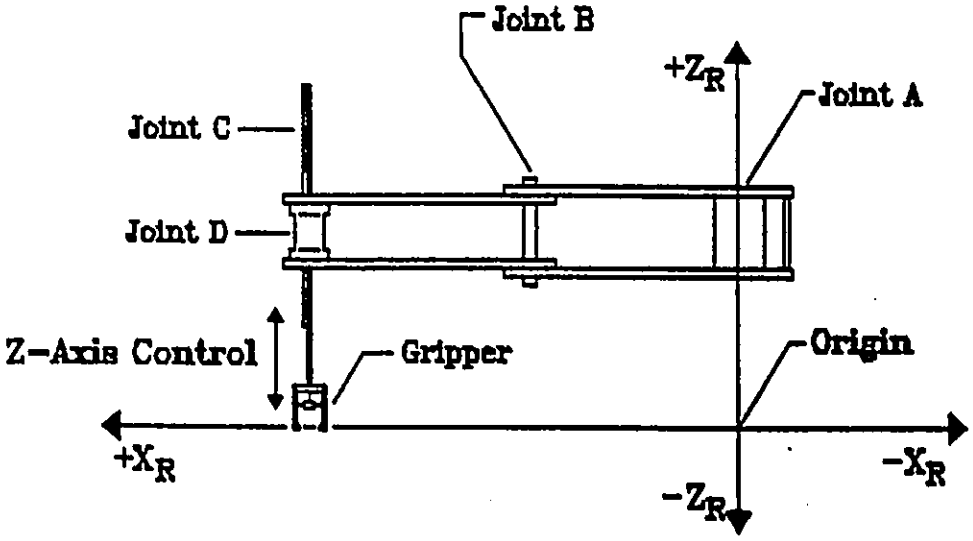
SIDE VIEW

FIGURE 7.1: The Rhino SCARA Robot

X-Y Planar Motion



TOP VIEW



SIDE VIEW

FIGURE 7.2: **The actions of the various joints with respect to the robot's Cartesian coordinate space.**

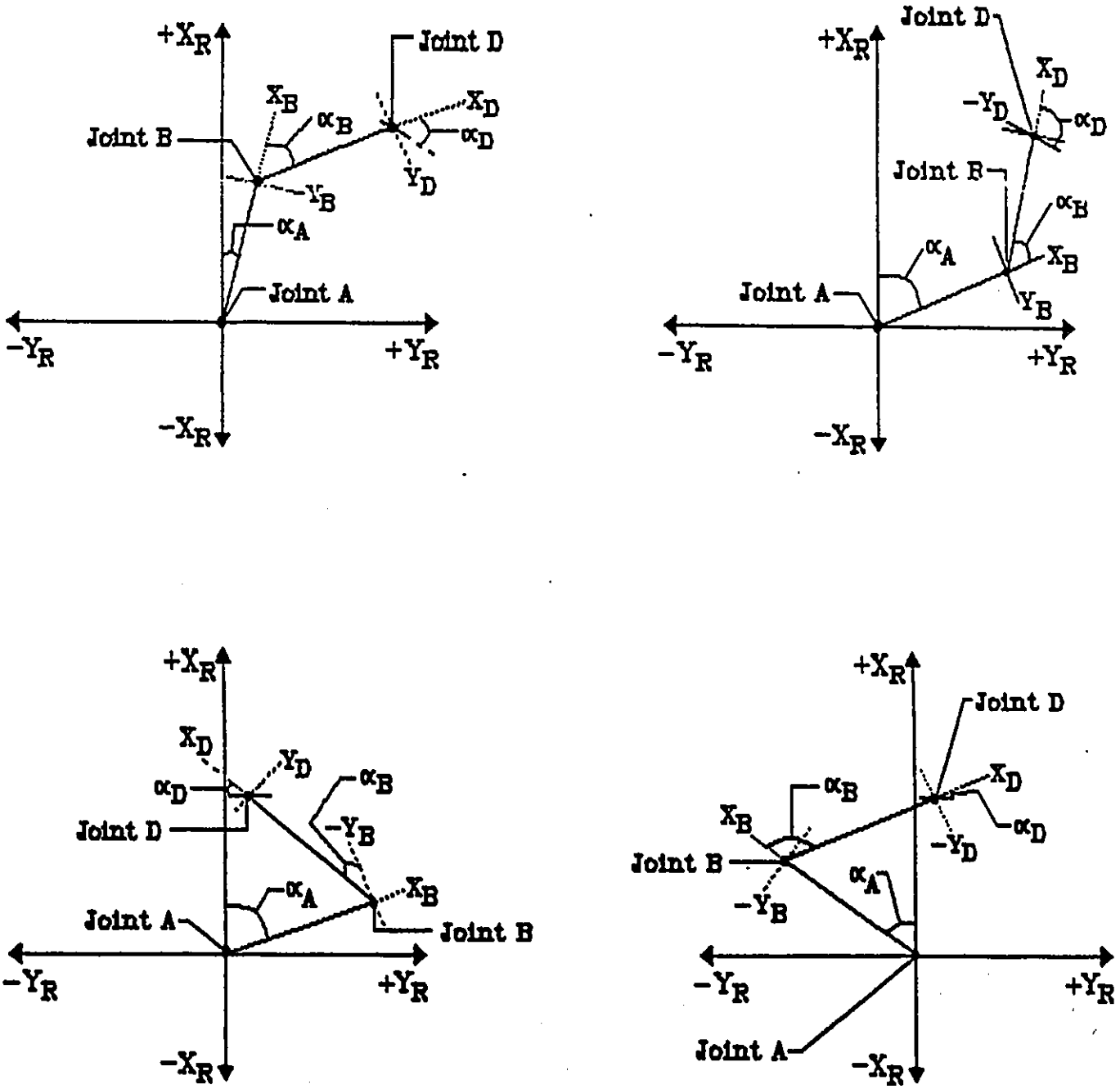
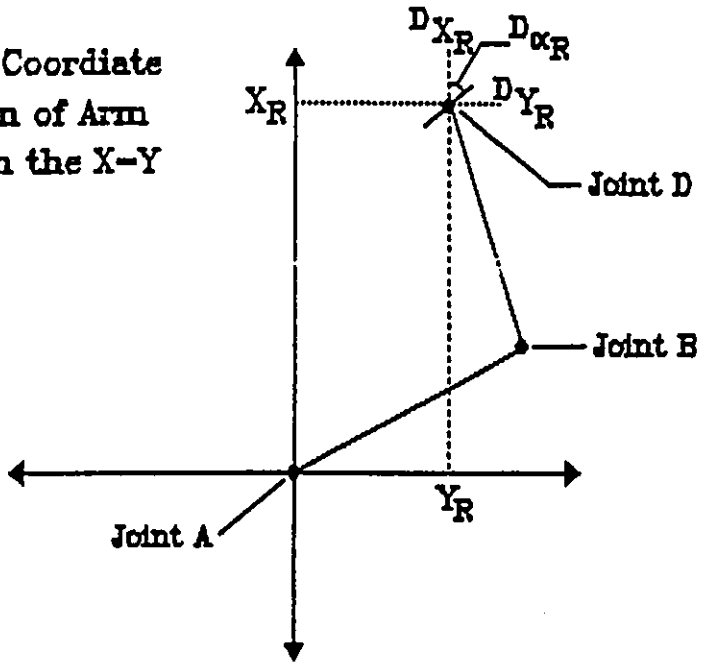


FIGURE 7.3: Various configurations of the robot arm in the X_R-Y_R plane. Note that the top configurations and the bottom configurations access the same Cartesian point despite different joint positions.

Cartesian Coordiate Description of Arm Position in the X–Y Plane.



Robot Joint Coordinate Description of the Arm Position in the X–Y Plane.

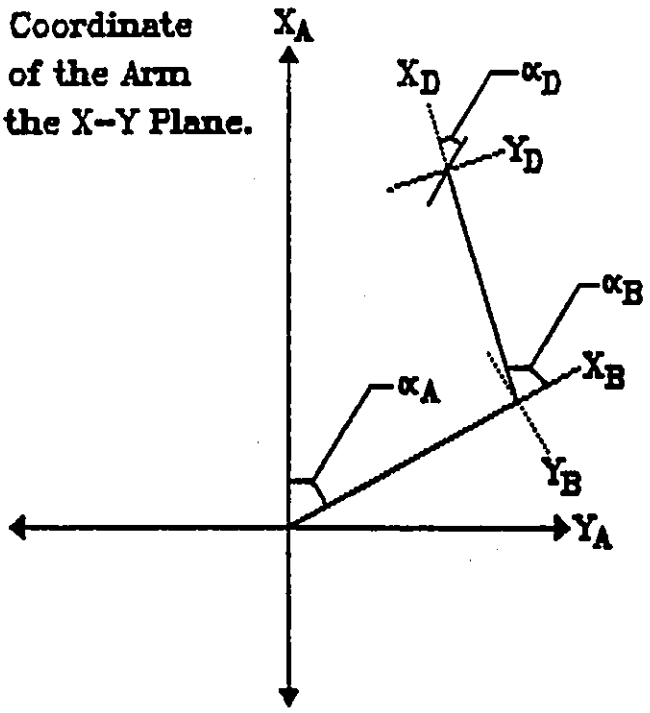
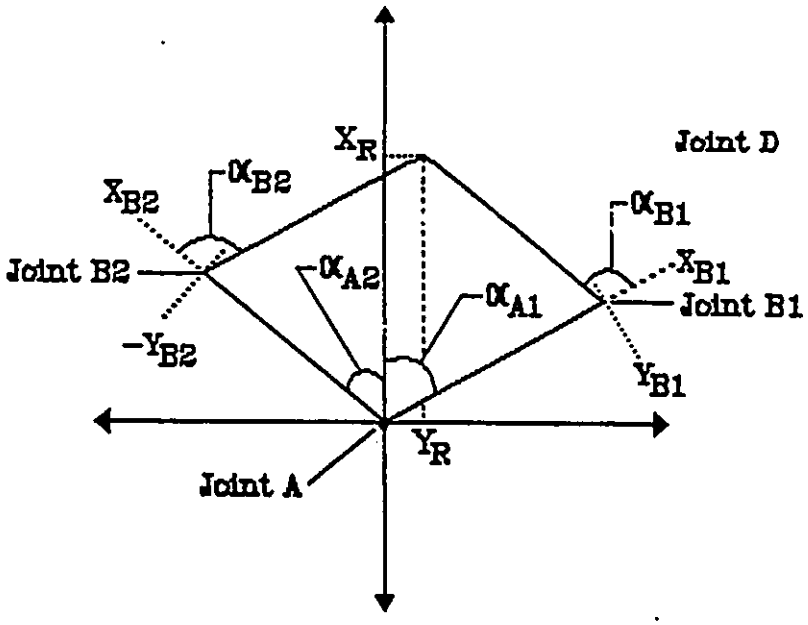


FIGURE 7.4: An illustration of the Cartesian and robot joint spaces defined with respect to end effector position and orientation.



The diagram on the left illustrates the two ways in which the arm may access a single Cartesian point in the X–Y plane. This is due to the rotating action of joints A and B.

Due to the symetry of the shape of the gripper and the rotating action of joint D the gripper may be driven through two different angles in order to achieve the same orientation with respect to a local Cartesian coordinate system parallel to the robot's Cartesian space.

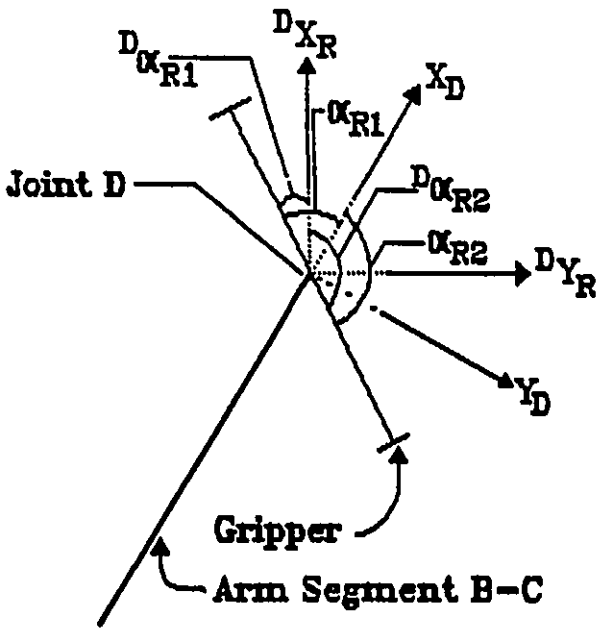
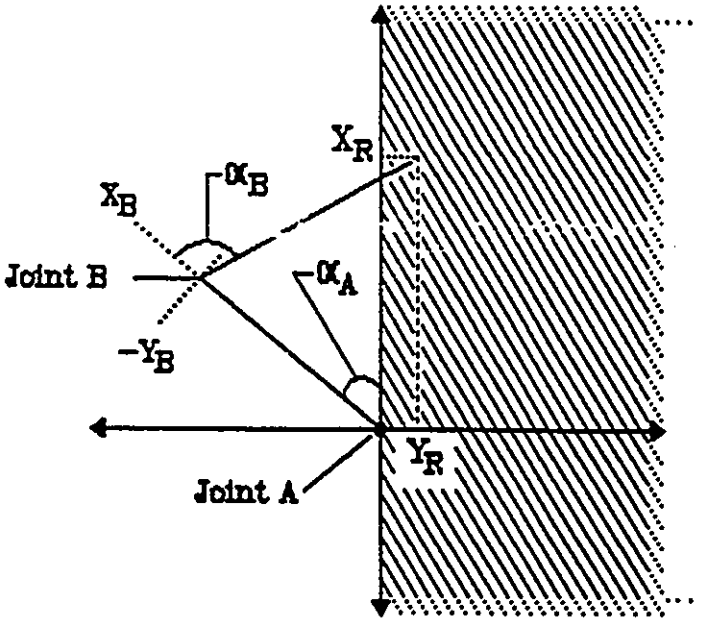
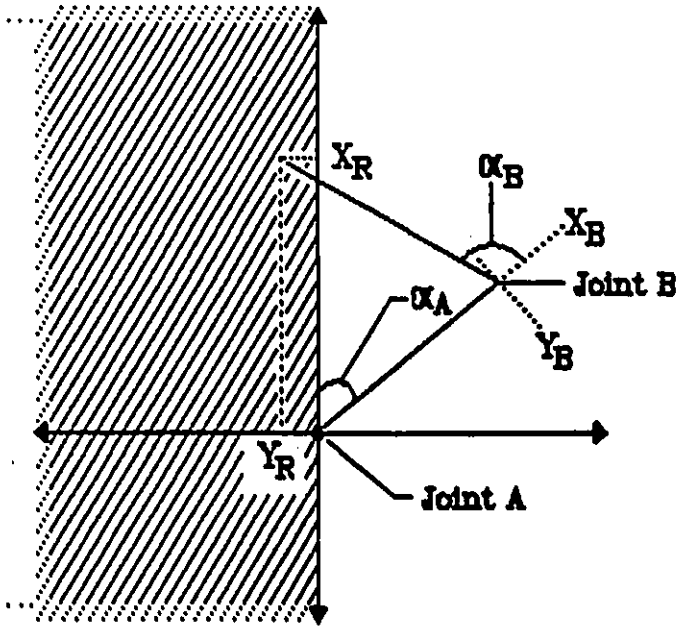


FIGURE 7.5: Diagrams showing the many to one relationship that exists between points in the robot's joint space and the robot's Cartesian space.

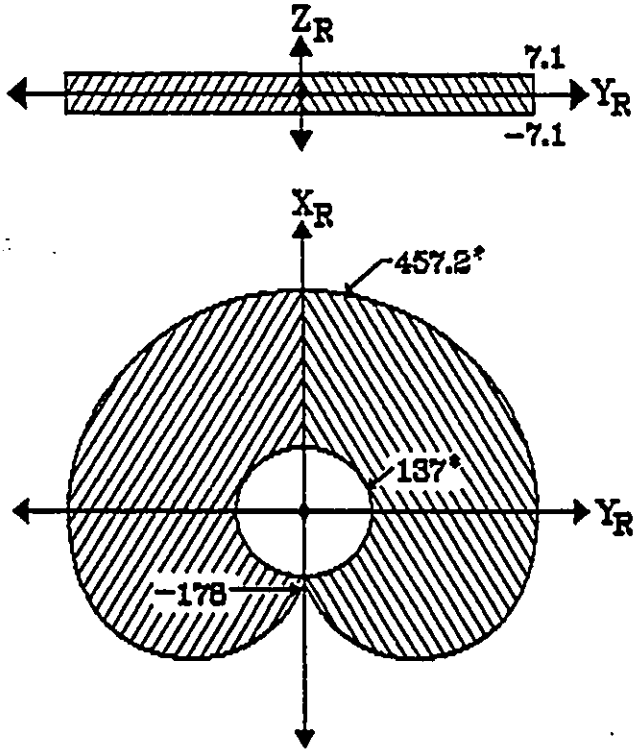
When Y_R is positive α_R must be positive. Thus to access the shaded region the elbow must be bent to the right as shown.



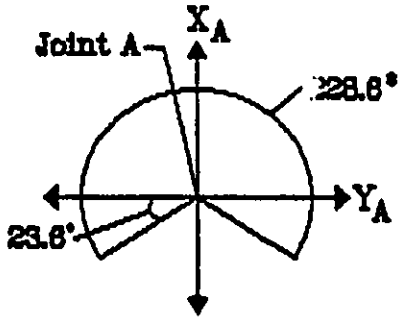


With Y less than zero α_B must also be negative. This restriction permits only one combination of joint angles for accessing a point in the 2-D Cartesian space.

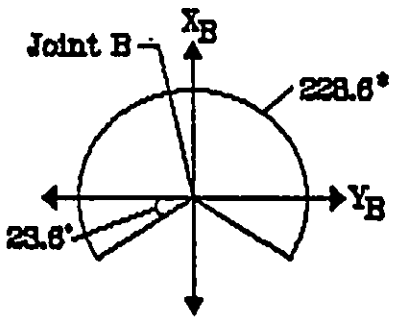
FIGURE 7.6: Restricting the joint angles as shown limits the number of joint space solutions for a given Cartesian point to one.



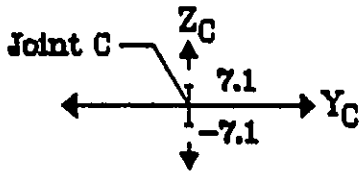
The diagrams to the left illustrate the operating envelope of the Rhino SCARA robot. Below are diagrams depicting the motion permitted for each of the three joints involved in the positioning of the end effector. The orientation of the end effector is controlled by a rotation about the Z-axis.



Arm Segment
A-B



Arm Segment
B-C



Arm Segment
C-Gripper

FIGURE 7.7: The pictures highlight the working range of the Rhino SCARA robot and associated joints.

Note: All dimensions are in mm. An asterisk (*) indicates a radial measurement.

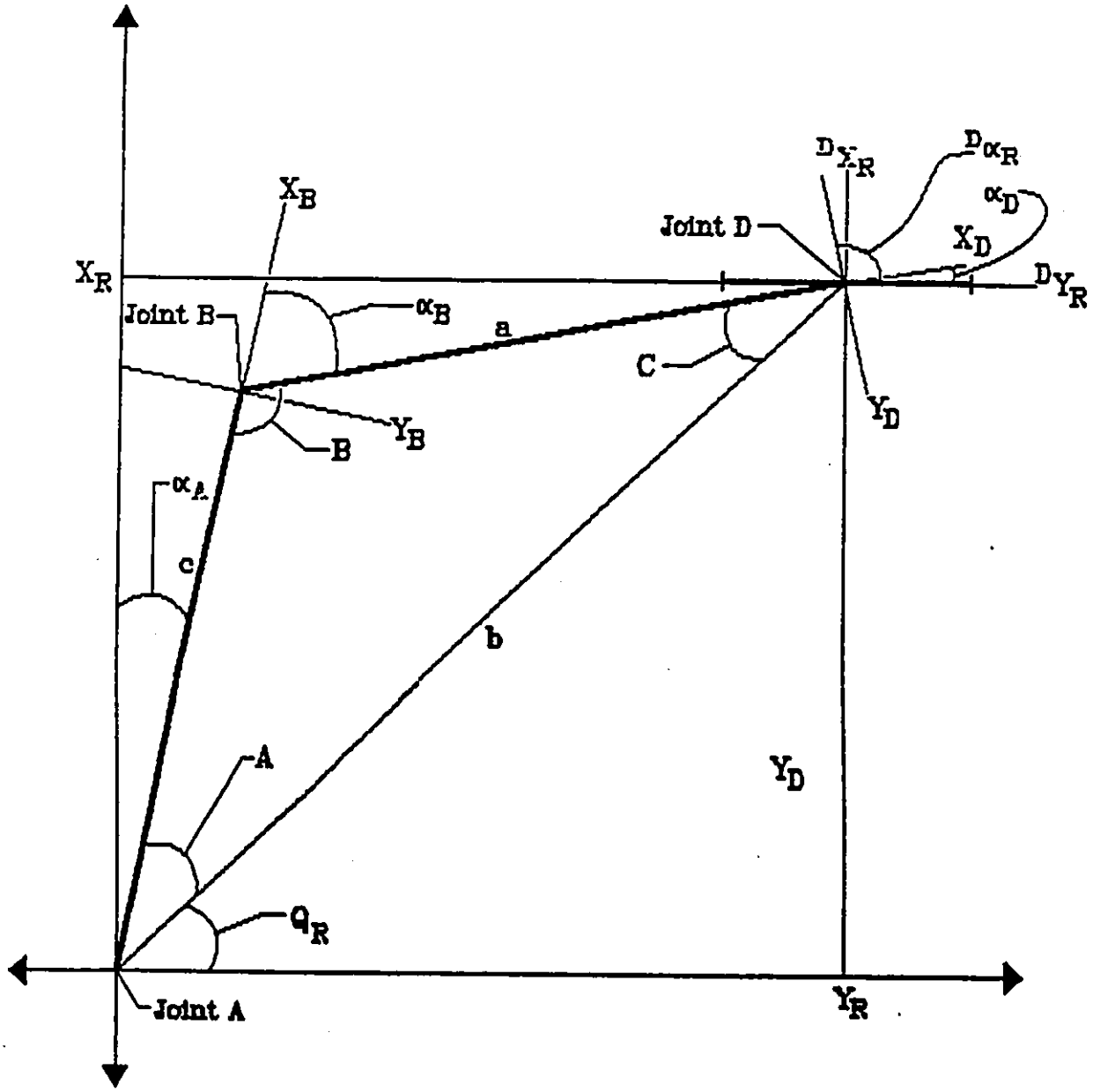


FIGURE 7.8: **An illustration of the geometrical relations involved in the derivation of the forward and inverse kinematic transformations.**

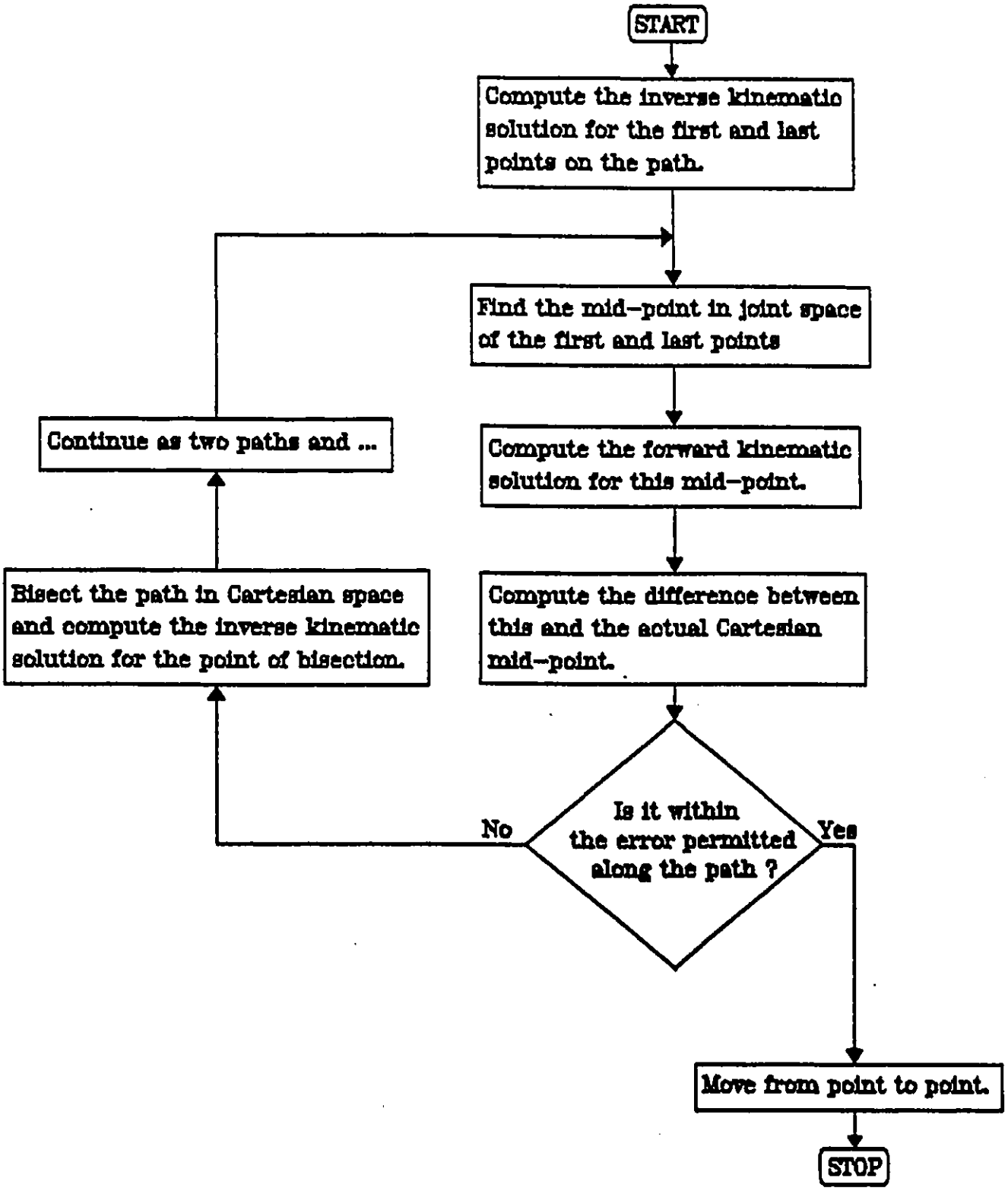


FIGURE 7.9: A flowchart illustrating the path control algorithm used for moving the robot with the part along the conveyor adapted from Snyder.

Joint	A	B	C	D
Resolution	1/8.5 deg.	1/4.4 deg.	1/7.1 mm	1/4.4 deg.
Type	Rotational	Rotational	Prismatic	Rotational
Function	Shoulder Rotation in X-Y plane.	Elbow Rotation in X-Y plane.	Z-axis control of gripper.	Orientation of gripper.
Range of Motion	224 deg.	228 deg.	82 mm	290 deg.

Length:	
A to B (d_A):	228.6 mm
B to C (d_B):	228.6 mm

Resolution:	1 mm
Speed:	203.2 mm/s Extended
Repeatability:	1 mm
Payload:	0.45 kg
Weight:	5.45 kg

TABLE 7.1: Various specifications of the Rhino SCARA Robot used to determine the kinematic transformations and the path tracking algorithm.

CHAPTER 8 CONCLUSION

The robotic assembly architecture currently under development provides the framework for the implementation of a task-level programming language. The windowing design tool and hardware developed in this thesis contributes to the improvement of the real-time parameters of the robot assembly system architecture without requiring costly high speed equipment.

The approach taken in this thesis, the use of "off the shelf" micro-computers and image acquisition systems upgraded by specialized hardware resulting from a detailed system analysis, may enable the propagation of the use of robotics to domains previously unapproachable. These systems require modifications in some cases to achieve the performance requirements desired. In some cases careful analysis of the situation provides for

enhanced algorithms reducing the otherwise sometimes extreme demands made of the equipment. In this thesis just such modifications and analyses provided near real-time parameter extraction from captured images. This information was used to locate and pick up an object in the image field using a robot arm controlled by the same micro-computer.

This thesis has modeled the parts presentation sub-task of a robotic assembly station currently under construction at the University of Ottawa. This model was based upon a task level language specification outlined herein. This model was analysed for theoretical performance as compared to current acquisition systems and simulated using aviliable equipment. Improvements in terms of vision based parameter extraction are also analysed for performance enhancement.

Chapter 4 provides a methodology for reducing the burden of calculation demanded by high resolution vision systems for a number of binary vision situations. The requirement of a lower resolution image derived from the high resolution image may be solved using an adaptation of the design of Chapter 5. These improvements allow for the use of a vision system providing accurate positional feedback information based upon a standard relatively inexpensive micro-computer.

The parts delivery operation uses a conveyor belt to transfer

parts from the vision system to the assembly platform. This allows the vision system, typically one of the most expensive elements of a system of this kind, to capture and analyze images for not just one robotic assembly platform but for many. With this transference comes the need to maintain the accuracy of the positional information gleaned from the image analysis as the part moves along the conveyor belt. Assuming the object remains stationary with respect to the conveyor belt (this is not unexpected nor unreasonable since the conveyor will move at a relatively constant speed) the position of the part is a function of the position of the conveyor between the point of image frame acquisition and the point at which the part is removed from the conveyor by a robot arm. This is controlled by coding positions on the conveyor using an absolute encoding system based upon the properties of pseudo-random binary sequences. Absolute encoding is far superior to incremental encoding in that any reading errors that occur are easily detected and corrected.

Finally, the thesis provides a robotic arm path control methodology for the tracking and grasping operations that remove the part from the conveyor as well as a description of the geometry and kinematic equations for a Rhino SCARA robot arm.

This thesis leaves the implementation of various structures described herein as a cohesive unit to another researcher who

will ultimately join the two sub-tasks, parts acquisition and assembly, under a single task manager. As stated the parts grasping operation based upon the information retrieved from the vision system has been demonstrated and is remarkably efficient. As well the pseudo-random binary encoding scheme has been used to successfully control a mobile robot, thus allowing for path recovery during deviations from a pre-determined pattern, in an un-related experiment.

The most important feature of the parts acquisition system and shared by the complete assembly station is the versatility and adaptability of the system. Robots are by definition non-specific, capable of performing a variety of tasks. So too should be robotic assembly systems if they are to be used in the rapidly changing manufacturing fields. This system is capable of performing a variety of assembly processes with a large number of pre-defined parts. Finally the system is intelligent in that it is capable of responding to environmental changes which modestly illustrates the capabilities of a feedback controlled robotic assembly station.

APPENDIX A PROGRAMS FOR COMPUTATION OF WINDOWING COSTS

```
PROGRAM CASE2(OUTPUT);
{
This program computes the cost of windowing using the algorithm of
Case 2.  It computes the costs for each resolution from n = 2 to n = 9
using values of r/l from 1/2 to 1/20.
}

var l,p,res,lres:integer;
    lcost,cost:real;

FUNCTION EXPO(BASE:real;POWER:integer):REAL;
{
Computes the value of a number, BASE, raised to the power of an
integer, POWER.
}

var i:integer;
    expon:real;

begin
expon:=1;
for i:=1 to power do begin
   expon:=base*expon;
   end;
expo:=expon;
end; {EX}


begin
for l:= 2 to 20 do begin
    lcost:=10.0E9;
    for res:= 4 to 9 do begin
{
Case 2 cost function:
}
   cost:=expo(2,2*res)+expo(2,20-2*res)+expo(2,20)/(l*l)+expo(2,21-res)/l;
   writeln(output,res);
      if (cost<lcost) then begin   {  Find the lowest cost for a variety }
        lcost:=cost;               {  of resolutions.                     }
        lres:=res
        end;

      end;{res}
   write(lst,l);
   write(lst,'  ');                {  Print out the values of r/l and     }
   write(lst,lcost);               {  the best cost and the best          }
   write(lst,'  ');                {  resolution.                         }
   writeln(lst,lres);
 end;
end.
```

```
PROGRAM CASE3(OUTPUT);

{
This program computes the cost of windowing using the algorithm of
Case3.  It computes costs for each resolution from n = 2 to n = 9
using values of r/l from 1/2 to 1/20 and number of passes from 1 to 20.
}

var l,p,res,lres:integer;
    lcost,cost:real;


FUNCTION EXPO(BASE:real;POWER:integer):REAL;
{
Computes the value of a number, Base to the power of an integer, POWER.
}
var i:integer;
    expon:real;

begin
expon:=1;
for i:=1 to power do begin
   expon:=base*expon;
   end;
expo:=expon;
end; {EX}


begin

for l:= 2 to 20 do begin
  for p:= 1 to 20 do begin
    lcost:=10.0E9;
    for res:= 4 to 9 do begin

{Case 3 cost function:
}
cost:=expo(2,2*res)+p*expo(2,20-2*res)-4*p+(p/l)*expo(2,21-res)-(p/l)*expo(2,12
;

      if (cost<lcost) then begin { Find the lowest cost for a }
        lcost:=cost;             { variety of resolutions.    }
        lres:=res
        end;

      end;{res}

   write(lst,l);                 { Print the values of r/l,   }
   write(lst,'  ');               { p and the best cost and    }
   write(lst,p);                 { the best resolution.       }
   write(lst,'  ');
   write(lst,lcost);
   write(lst,'  ');
   writeln(lst,lres);
   end;
 end;
end.
```

APPENDIX B SIMULATIONS OF THE WINDOWING ALGORITHM

```
PROGRAM PTEST1(input,output);
{
This is a test of the windowing operation using the algorithm of Case 2
optimized to reduce the number of calculations required in order to reduce
the window to the ideal size.  This routine assumes the existence of a 512
X 512 binary pixel image of some object with a high signal to noise ratio and
a 64 X 64 binary pixel image of the same image frame as the 512 X 512 image.
The routine computes the area and center of area for all pixels that are "on".

Note:  This routine will also compute the values of area and center of area
       using only the 512 X 512 image.
}

type charfile = file of char;

var x1,x2,y1,y2,method:integer;
      area,x_pos,y_pos,f_pos:real;
      fname64,fname512:string[25];
      lp:charfile;
      fp:file;


PROCEDURE WINDOW(var min_x,min_y,max_x,max_y:integer;var fp:file);
{
This procedure computes the window that circumcribes the object in the
64 X 64 pixel image and computes the coordinates of that window as positions
in the 512 X 512 image.
}

  type pic_64_array =  array [0..63,0..63] of char;

  var  pixel:          char;
       x,y:            integer;
       pic_64:         pic_64_array;
begin

blockread(fp,pic_64,32);

for y:=0 to 63 do begin
  for x:=0 to 63 do begin

    if (ord(pic_64[y,x]) < 127) then begin
      if (x>max_x) then max_x:=x;
      if (x<min_x) then min_x:=x;
      if (y>max_y) then max_y:=y;
      if (y<min_y) then min_y:=y;
    end;{if}
  end;{y}
end;{x}

if not(min_x < 2 ) then min_x:=min_x*8
  else min_x:=0;
if not(min_y < 2 ) then min_y:=min_y*8
  else min_y:=0;
if not(max_x >62) then max_x:=max_x*8
  else max_x:=511;
if not(max_y >62) then max_y:=max_y*8
  else max_y:=511;
end;{WINDOW}
```

```
PROCEDURE REFINE_TOP(var min_x,min_y,max_x:integer;var lp:charfile);
{
This procedure, one of four similar procedures refines the top boundary
of the window using an 8 neighbourhood operation.
}

var pixel  : char;
    refine : boolean;
    x      : integer;
    f_pos  : real;
begin

x:=min_x;
f_pos:=(min_y-1)/1*512+x;
seek(lp,f_pos);
refine:=false;

while (not(x>max_x) and not(refine)) do begin

  read(lp,pixel);
  if (ord(pixel)>127) then x:=x+1
  else begin
    min_y:=min_y-1;
    refine:=true;
    end;

  end;{while}

if (refine) then refine_top(min_x,min_y,max_x,lp);

end;{REFINE_TOP}


PROCEDURE REFINE_LHS(var min_x,min_y,max_y:integer;var lp:charfile);
{
This procedure, one of four similar procedures refines the left hand
vertical boundary of the window using an 8 neighbourhood operation.
}
var pixel  : char;
    refine : boolean;
    y      : integer;
    f_pos  : real;
begin

y:=min_y;
refine:=false;

while (not(y>max_y) and not(refine)) do begin

  f_pos:=y/1*512+min_x-1;
  seek(lp,f_pos);
  read(lp,pixel);
  if (ord(pixel)>127) then y:=y+1
  else begin
    min_x:=min_x-1;
    refine:=true;
    end;

  end;

if (refine) then refine_lhs(min_x,min_y,max_y,lp);

end;{REFINE_LHS}
```

```
PROCEDURE REFINE_RHS(var min_y,max_x,max_y:integer;var lp:charfile);
{
This procedure, one of four similar procedures refines the right hand
vertical boundary of the window using an 8 neighbourhood operation.
}
  var pixel  : char;
      refine : boolean;
      y      : integer;
      f_pos  : real;
begin

  y:=min_y;
  refine:=false;

  while (not(y>max_y) and not(refine)) do begin

    f_pos:=y/1*512+max_x+1;
    seek(lp,f_pos);
    read(lp,pixel);
    if (ord(pixel)>127) then y:=y+1
    else begin
      max_x:=max_x+1;
      refine:=true;
      end;

    end;

if (refine) then refine_rhs(min_y,max_x,max_y,lp);

end;{REFINE_RHS}


PROCEDURE REFINE_BOT(var min_x,max_x,max_y:integer;var lp:charfile);
{
This procedure, one of four similar procedures refines the bottom
boundary of the window using an 8 neighbourhood operation.
}
  var pixel  : char;
      refine : boolean;
      x      : integer;
      f_pos  : real;
begin

  x:=min_x;
  refine:=false;
  f_pos:=x+(max_y+1)/1*512;
  seek(lp,f_pos);

  while (not(x>max_x) and not(refine)) do begin
    read(lp,pixel);
    if (ord(pixel)>127) then x:=x+1
    else begin
      max_y:=max_y+1;
      refine:=true;
      end;

    end;

if (refine) then refine_bot(min_x,max_x,max_y,lp);

end;{REFINE_BOT}
```

```
PROCEDURE REFINE_WINDOW(var min_x,min_y,max_x,max_y:integer;
                        var lp:charfile);
{
Refines the window to the ideal size using the previous four procedures
to compute the optimum sides of the window.
}

begin

refine_top(min_x,min_y,max_x,lp);
refine_lhs(min_x,min_y,max_y,lp);
refine_rhs(min_y,max_x,max_y,lp);
refine_bot(min_x,max_x,max_y,lp);

end;{REFINE_WINDOW}



PROCEDURE SEARCH(var area,x_pos,y_pos:real;
                     min_x,min_y,max_x,max_y:integer;
                 var lp:charfile);
{
This computes the number of pixels turned "on" within the window boundaries
and also the coordinate around which they are centered.  It does this by
examining each of the pixels in the image within the window.
}

var x,y:integer;
    pixel:char;
    f_pos:real;

begin

for y:=min_y to max_y do begin

  f_pos:=y/1*512+min_x;

  seek(lp,f_pos);

  for x:=min_x to max_x do begin

    read(lp,pixel);

    if (ord(pixel)<127) then begin

      area:=area+1;
      x_pos:=x_pos+x;
      y_pos:=y_pos+y
      end;{if}

  end;{x}
end;{y}
```

```
x_pos:=round(x_pos/area);
y_pos:=round(y_pos/area);

end;{SEARCH}


begin {MAIN PROGRAM}

x1:=63;
x2:=0;
y1:=63;
y2:=0;
area:=0;
x_pos:=0;
y_pos:=0;

clrscr;
writeln(output);
writeln(output,'Progressive Image processing Test');
writeln(output);
writeln(output,'This program calculates the number of black pixels ');
writeln(output,'in a binary picture in two ways.');
writeln(output);
write(output,'Enter 1 for regular, 2 for progressive: ');
readln(input,method);
writeln(output);
write(output,'Enter the name of the 512 X 512 picture: ');
readln(input,fname512);
assign(lp,fname512);
reset(lp);
writeln(output);
if(method=2) then begin
  write(output,'Enter the name of the 64 X 64 picture: ');
  readln(input,fname64);
  writeln(output);
  assign(fp,fname64);
  reset(fp);
  window(x1,y1,x2,y2,fp);
  refine_window(x1,y1,x2,y2,lp);
  search(area,x_pos,y_pos,x1,y1,x2,y2,lp);
  end
else begin
  search(area,x_pos,y_pos,0,0,511,511,lp);
  end;
  writeln(output);
  write(output,'The area is ');
  write(output,area);
  writeln(output,' pixels.');
  write(output,'The centroid is ');
  write(output,x_pos);
  write(output,' on the X-axis and ');
  write(output,y_pos);
  writeln(output,' on the Y-axis.');
  writeln(output)
end.{PTEST1}
```

```
PROGRAM PTEST2(input,output);
{
This program is similar to PTEST1 in that it uses the algorithm of Case 2
to compute the window that limits the processing for an object in a high
resolution image.  However this program does not require the high signal to
noise ratio that PTEST2 needs since it is able to compute windows for any
number of objects within the image frame provided that they are separately
discerable in the high resolution image.  As well the program provides a
limited interface with a SIR 1 educational robot and has performed picking
and placing of a set of disks in the image frame.

Note:  This program assumes the existance of a high resolution image and its
       low resolution counterpart.  This creates a performance problem in that
       the computation of the low resolution image is time consuming.

  const object_area:real = 1700;
        error       :real =  500;
        shoulder_max:integer = 1150;
        shoulder_min:integer =  380;
        elbow_max:integer    =  450;
        elbow_min:integer    =  670;
        center_x:integer     =  255;
        center_y:integer     =  255;
        conversion:real      =  0.6;
        distance_y:integer   =  450;
        min_d:integer        =  350;
        max_d:integer        =  400;

  type charfile       = file of char;
       window_pointer = ^window_element;
       window_element = record
                          min_x:integer;
                          min_y:integer;
                          max_x:integer;
                          max_y:integer;
                          next :window_pointer;
                          end;
       pixel_pointer  = ^pixel_element;
       pixel_element  = record
                          x    :integer;
                          y    :integer;
                          next :pixel_pointer;
                          end;

  var x1,x2,y1,y2,method      :integer;
      area,x_pos,y_pos,f_pos  :real;
      window_sill             :window_pointer;
      fname64,fname512        :string[25];
      lp                      :charfile;
      fp                      :file;
```

```
{
The following four procedures are standard stack operations designed for
the particular data types involved.
}

PROCEDURE PUSH_PIXEL(pixel:pixel_pointer;var object_tos:pixel_pointer);

  begin
  pixel^.next:=object_tos;
  object_tos:=pixel
  end;

PROCEDURE POP_PIXEL(var x,y :integer; var tos:pixel_pointer);

  begin
  x:=tos^.x;
  y:=tos^.y;
  tos:=tos^.next
  end;

PROCEDURE PUSH_WINDOW(window:window_pointer;var window_sill:window_pointer);

  begin
  window^.next:=window_sill;
  window_sill:=window
  end;

PROCEDURE POP_WINDOW(var min_x,min_y,max_x,max_y:integer;
                     var window_sill:window_pointer);

  begin
  min_x:=window_sill^.min_x;
  min_y:=window_sill^.min_y;
  max_x:=window_sill^.max_x;
  max_y:=window_sill^.max_y;
  window_sill:=window_sill^.next
  end;
```

```
PROCEDURE MAKE_WINDOWS(var window_sill:window_pointer;var fp:file);

{
This procedure computes the windows surrounding objects located in the
64 X 64 pixel image.  It uses 8 connectivity to determine the windows and
will define a number of windows corresponding to the number of discernable
regions in the 64 X 64 pixel image.  These regions are first identified
using a labeling operation.  It then corrects these values for coordinates
for the 512 X 512 pixel image.
}

type   pic_64_array =  array [0..63,0..63] of char;

var    lable,i,j,x,y,next_x,next_y,addx,addy : integer;
       pic_64              : pic_64_array;
       pixel               : pixel_pointer;
       object_tos          : pixel_pointer;
       window              : window_pointer;

begin

blockread(fp,pic_64,32);
lable:=1;

for y:=0 to 63 do begin
  for x:=0 to 63 do begin


    if (ord(pic_64[y,x]) = 0) then begin
      pic_64[y,x]:=chr(lable);
      new(window);
      window^.max_x:=0;
      window^.max_y:=0;
      window^.min_x:=63;
      window^.min_y:=63;
      new(object_tos);
      object_tos:=nil;
      new(pixel);
      pixel^.x:=x;
      pixel^.y:=y;
      push_pixel(pixel,object_tos);

      while not(object_tos = nil) do begin
        pop_pixel(next_x,next_y,object_tos);

          for i:= -1 to 1 do begin
            for j:= -1 to 1 do begin
              addx:=next_x+j;
              addy:=next_y+i;
              if (addx > 63) then addx:=63
                else if (addx < 0) then addx:=0;
              if (addy > 63) then addy:=63
                else if (addy < 0) then addy:=0;
```

```
            if (ord(pic_64[addy,addx]) = 0) then begin
              pic_64[addy,addx]:=chr(lable);
              new(pixel);
              pixel^.x:=addx;
              pixel^.y:=addy;
              push_pixel(pixel,object_tos);
              end;{if}
            end;{j}
          end;{i}

      if (next_x>window^.max_x) then window^.max_x:=next_x;
      if (next_x<window^.min_x) then window^.min_x:=next_x;
      if (next_y>window^.max_y) then window^.max_y:=next_y;
      if (next_y<window^.min_y) then window^.min_y:=next_y;


      end;{while}

    if not(window^.min_x < 2 ) then window^.min_x:=window^.min_x*8
      else window^.min_x:=0;
    if not(window^.min_y < 2 ) then window^.min_y:=window^.min_y*8
      else window^.min_y:=0;
    if not(window^.max_x >62) then window^.max_x:=window^.max_x*8
      else window^.max_x:=511;
    if not(window^.max_y >62) then window^.max_y:=window^.max_y*8
      else window^.max_y:=511;

    push_window(window,window_sill);

    lable:=lable+1;
    end;{if}
  end;{y}
end;{x}

end;{MAKE_WINDOWS}


{
The following five procedures optimize the windows for size using an
8 neighborhood operation.
}
```

```
PROCEDURE REFINE_TOP(var min_x,min_y,max_x:integer;var lp:charfile);

var pixel  : char;
    refine : boolean;
    x      : integer;
    f_pos  : real;

begin

x:=min_x;
f_pos:=(min_y-1)/1*512+x;
seek(lp,f_pos);
refine:=false;

while (not(x>max_x) and not(refine)) do begin

  read(lp,pixel);
  if (ord(pixel)>127) then x:=x+1
  else begin
    min_y:=min_y-1;
    refine:=true;
    end;

  end;{while}

if (refine) then refine_top(min_x,min_y,max_x,lp);

end;{REFINE_TOP}


PROCEDURE REFINE_LHS(var min_x,min_y,max_y:integer;var lp:charfile);

var pixel  : char;
    refine : boolean;
    y      : integer;
    f_pos  : real;
begin

y:=min_y;
refine:=false;

while (not(y>max_y) and not(refine)) do begin

  f_pos:=y/1*512+min_x-1;
  seek(lp,f_pos);
  read(lp,pixel);
  if (ord(pixel)>127) then y:=y+1
  else begin
    min_x:=min_x-1;
    refine:=true;
    end;

  end;

if (refine) then refine_lhs(min_x,min_y,max_y,lp);

end;{REFINE_LHS}
```

```
PROCEDURE REFINE_RHS(var min_y,max_x,max_y:integer;var lp:charfile);

  var pixel  : char;
      refine : boolean;
      y      : integer;
      f_pos  : real;
begin

  y:=min_y;
  refine:=false;

  while (not(y>max_y) and not(refine)) do begin

    f_pos:=y/1*512+max_x+1;
    seek(lp,f_pos);
    read(lp,pixel);
    if (ord(pixel)>127) then y:=y+1
    else begin
      max_x:=max_x+1;
      refine:=true;
      end;

    end;

if (refine) then refine_rhs(min_y,max_x,max_y,lp);

end;{REFINE_RHS}


PROCEDURE REFINE_BOT(var min_x,max_x,max_y:integer;var lp:charfile);

  var pixel  : char;
      refine : boolean;
      x      : integer;
      f_pos  : real;
begin

  x:=min_x;
  refine:=false;
  f_pos:=x+(max_y+1)/1*512;
  seek(lp,f_pos);

  while (not(x>max_x) and not(refine)) do begin
    read(lp,pixel);
    if (ord(pixel)>127) then x:=x+1
    else begin
      max_y:=max_y+1;
      refine:=true;
      end;

    end;

if (refine) then refine_bot(min_x,max_x,max_y,lp);

end;{REFINE_BOT}
```

```
PROCEDURE REFINE_WINDOW(var min_x,min_y,max_x,max_y:integer;
                        var lp:charfile);

begin
refine_top(min_x,min_y,max_x,lp);
refine_lhs(min_x,min_y,max_y,lp);
refine_rhs(min_y,max_x,max_y,lp);
refine_bot(min_x,max_x,max_y,lp);
end;{REFINE_WINDOW}


PROCEDURE MOVE_TO(var x1,y1,x2,y2:integer);
{
This routine executes a simple form of the inverse kinematic transformations
for the SIR-1 robot.  It drives the robot to a position near the location of
the object within the window defined by x1,y1,x2, and y2.  This occurs prior
to the completion of the high resolution processing.  This routine is used
when the robot is slow and as it allows the robot to start moving sooner.
}
var x,y                              :real;
    base,shoulder,elbow,gripper :integer;
    base_angle,distance,ratio   :real;
    stb,sts,ste,stg             :string[3];
    outp                        :string[30];

begin

writeln(aux,'H');
x:=(x2-x1)/2+x1;
x:=(x-center_x)*conversion;
y:=(y2-y1)/2+y1;
y:=(y-center_y)*conversion + distance_y;
base_angle:= arctan(x/y)*180/(2*3.1415926);
distance:= sqrt(x*x+y*y);
ratio:= (distance-min_d)/(max_d-min_d);

if not(ratio>1) then begin
  elbow:=0;{ round(elbow_min-ratio*(elbow_min-elbow_max))-200}
  shoulder:=round(shoulder_min+ratio*(shoulder_max-shoulder_min));
  base:=round(base_angle/0.24);
  gripper:=175;
  str(base,stb);
  str(shoulder,sts);
  str(elbow,ste);
  str(gripper,stg);
  outp:='M'+' '+stb+','+sts+','+'0'+','+'0'+','+'0'+','+stg;
  writeln(aux,outp);
  x1:=shoulder;
  y1:=base;
  x2:=elbow
  end

else begin
  writeln(output,'OBJECT OUT OF RANGE');
  x1:=0;
  y1:=0;
  x2:=0
  end;
end;{MOVE_TO}
```

```
PROCEDURE PICK_UP(x_pos,y_pos:real;x1,y1,x2:integer);
{
This procedure also moves the SIR-1 robot but this time the coordinates
corresponds to the center of area of the object, x-pos and y-pos,
and the robot moves to the location, descends upon the object, picks
it up, and then places it back down again.

var x,y                          :real;
    base,shoulder,elbow,gripper :integer;
    base_angle,distance,ratio    :real;
    stb,sts,ste,stg              :string[3];
    outp                         :string[25];

begin

x:=x_pos;
x:=(x-center_x)*conversion;
y:=y_pos;
y:=(y-center_y)*conversion + distance_y;
base_angle:= arctan(x/y)*180/(2*3.1415926);
distance:= sqrt(x*x+y*y);
ratio:= (distance-min_d)/(max_d-min_d);

if not(ratio>2) then begin
  elbow:= round(elbow_min-ratio*(elbow_min-elbow_max)-10);
  shoulder:=round(shoulder_min+ratio*(shoulder_max-shoulder_min)+100);
  base:=round(base_angle/0.24-y1);
  gripper:=375;
  outp[1]:='M';
  outp[2]:=' ';
  str(base,stb);
  str(shoulder,sts);
  str(elbow,ste);
  str(gripper,stg);
  outp:='M'+' '+stb+','+sts+','+'0'+','+'0'+','+'0'+','+stg;
  writeln(aux,outp);
  outp:='M'+' '+'0'+','+'0'+','+ste+','+'0'+','+'0'+','+'0';
  writeln(aux,outp);
  outp:='M'+' '+'0'+','+'0'+','+'0'+','+'0'+','+'0'+','+'-275';
  writeln(aux,outp);
  outp:='M'+' '+'0'+','+'-400'+','+'-200';
  writeln(aux,outp);
  base:=y1-base;
  shoulder:=-shoulder;
  elbow:=-elbow;
  str(base,stb);
  str(shoulder,sts);
  str(elbow,ste);
  outp:='M'+' '+stb+','+sts+','+ste;
  writeln(aux,outp);
  outp:='M'+' '+'0'+','+'0'+','+'0'+','+'0'+','+'0'+','+'100';
  writeln(aux,outp);
  writeln(aux,'H');
  end

else begin
  writeln(output,'OBJECT OUT OF RANGE');
  end;

end;{PICK_UP}
```

```
PROCEDURE SEARCH(var area,x_pos,y_pos:real;
                     min_x,min_y,max_x,max_y:integer;
                 var lp:charfile);
{
Using the coordinates provided by the windowing operations this routine
computes the area and center of area for the object in the window.  These
coordinates are used to move the robot to pick up the object.
}

var x,y   :integer;
    pixel :char;
    f_pos :real;

begin

area:=0;
x_pos:=0;
y_pos:=0;

for y:=min_y to max_y do begin
  f_pos:=y/1*512+min_x;
  seek(lp,f_pos);

  for x:=min_x to max_x do begin
    read(lp,pixel);

    if (ord(pixel)<127) then begin
      area:=area+1;
      x_pos:=x_pos+x;
      y_pos:=y_pos+y
      end;{if}

  end;{x}

end;{y}

if(area=0) then area:=area+1;

x_pos:=round(x_pos/area);
y_pos:=round(y_pos/area);

end;{SEARCH}


begin  {MAIN PROGRAM}

fname512:='d:present.pic';
assign(lp,fname512);
reset(lp);
fname64:='d:present.p64';
assign(fp,fname64);
reset(fp);
new(window_sill);
window_sill:=nil;
make_windows(window_sill,fp);
```

```
while not(window_sill=nil) do begin
  pop_window(x1,y1,x2,y2,window_sill);
  refine_window(x1,y1,x2,y2,lp);
  move_to(x1,y1,x2,y2);
  search(area,x_pos,y_pos,x1,y1,x2,y2,lp);
  writeln(output);
  write(output,'The area is ');
  write(output,area);
  writeln(output,' pixels.');
  write(output,'The centroid is ');
  write(output,x_pos);
  write(output,' on the X-axis and ');
  write(output,y_pos);
  writeln(output,' on the Y-axis.');
  writeln(output);

  if not((area > object_area+error) or (area < object_area-error)) then
     pick_up(x_pos,y_pos,0,0,0);

  end;{while}

end.{MAIN PROGRAM}
```

APPENDIX C THE PSEUDO-RANDOM 7-TUPLES AND THEIR CORRESPONDING NATURAL POSITIONS

NATURAL POS.	PRBS CODE	NATURAL POS.	PRBS CODES
1	000 0001	33	111 1100
2	000 0010	34	111 1000
3	000 0100	35	111 0001
4	000 1001	36	110 0011
5	001 0010	37	100 0111
6	010 0100	38	000 1110
7	100 1001	39	001 1101
8	001 0011	40	011 1011
9	010 0110	41	111 0110
10	100 1101	42	110 1100
11	001 1010	43	101 1000
12	011 0100	44	011 0001
13	110 1001	45	110 0010
14	101 0011	46	100 0101
15	010 0111	47	000 1010
16	100 1111	48	001 0100
17	001 1110	49	010 1001
18	011 1101	50	101 0010
19	111 1011	51	010 0101
20	111 0111	52	100 1011
21	110 1110	53	001 0111
22	101 1100	54	010 1111
23	011 1000	55	101 1111
24	111 0000	56	011 1110
25	110 0001	57	111 1101
26	100 0011	58	111 1010
27	000 0111	59	111 0101
28	000 1111	60	110 1010
29	001 1111	61	101 0101
30	011 1111	62	010 1010
31	111 1111	63	101 0100
32	111 1110	64	010 1000

NATURAL POS.	PRBS CODE	NATURAL POS.	PRBS CODES
65	101 0000	97	100 0001
66	010 0001	98	000 0011
67	100 0010	99	000 0110
68	000 0101	100	000 1101
69	000 1011	101	001 1011
70	001 0110	102	011 0110
71	010 1101	103	110 1101
72	101 1011	104	101 1010
73	011 0111	105	011 0101
74	110 1111	106	110 1011
75	101 1110	107	101 0111
76	011 1100	108	010 1110
77	111 1001	109	101 1101
78	111 0011	110	011 1010
79	110 0111	111	111 0100
80	100 1110	112	110 1000
81	001 1100	113	101 0001
82	011 1001	114	010 0011
83	111 0010	115	100 0110
84	110 0101	116	000 1100
85	100 1010	117	001 1001
86	001 0101	118	011 0010
87	010 1011	119	110 0100
88	101 0110	120	100 1000
89	010 1100	121	001 0001
90	101 1001	122	010 0010
91	011 0011	123	100 0100
92	110 0110	124	000 1000
93	100 1100	125	001 0000
94	001 1000	126	010 0000
95	011 0000	127	100 0000
96	110 0000	1	000 0001

APPENDIX D PROGRAM FOR CONTROLLING THE RHINO SCARA ROBOT IN CARTESIAN SPACE

```
{$D-} {Enable I/O buffering}

PROGRAM RHINOXYZ(input,output);

type steps  = record
              wrist    :integer;
              shoulder :integer;
              elbow    :integer;
              z        :integer;
              end;

     angles = record
              wrist    :real;
              shoulder :real;
              elbow    :real;
              z        :real;
              end;

     posit  = record
              x        :real;
              y        :real;
              z        :real;
              gripper  :integer;
              end;

     length = record
              shoulder :real;
              elbow    :real;
              end;

const max_steps    :steps  = (wrist:640;shoulder:950;elbow:500;z:290);{steps}
                             {This refers to the number of steps allowed for
                              each joint from the hard home position in one
                              direction.}

      lengths      :length = (shoulder:228.6;elbow:228.6);{mm}
                             {This is the length of each joint.}

      step_size    :angles = (wrist:4.4;shoulder:-8.5;elbow:4.4;z:7.1);
                             {This is the ratio of steps to degrees for each
                              rotational joint or steps to mm for the
                              prismatic joint.}

      radtodeg     :real   = 57.29579;
      pi           :real   = 3.1415926;


var  next_dest    :posit;
     next_angle   :angles;
     last_dest    :steps;
     last_gripper:integer;
     dest_step    :steps;
     movesteps    :boolean;
     moveangles   :boolean;
```

```
    done         :boolean;
    true_angle   :angles;
    i:integer;

FUNCTION ARCSIN(u:real):real;{This computes the arcsine of a real number u
                              provided that -1 < u < 1.}

begin

if (u*u = 1) then begin
  if (u>0) then arcsin:=pi/2
  else arcsin:=-1*pi/2;
  end
else arcsin:=arctan(u/sqrt(1-u*u));

end;


PROCEDURE FIND_ANGLES(next_dest:posit;var next_angle:angles;
                                      var moveangles:boolean);

{Given the desired Cartesian point this function computes the necessary
joint angles to move the robot to that point.  Ths is essentially the inverse
kinematic transformation for the robot.}

var a,BB,CC,p,s,t : real;

begin

a:=sqrt(next_dest.x*next_dest.x+next_dest.y*next_dest.y);

if(next_dest.x=0) then p:=pi/2
else p:=arctan(next_dest.y/abs(next_dest.x));

s:=0.5*(a+lengths.shoulder+lengths.elbow);

if ((a>s) or (lengths.shoulder>s) or (lengths.elbow>s)) then moveangles:=false
else begin
  moveangles:=true;
  t:=sqrt(s*(s-a)*(s-lengths.shoulder)*(s-lengths.elbow));
  BB:=arcsin((2*t)/(a*lengths.elbow));
  CC:=arcsin((2*t)/(a*lengths.shoulder));

  next_angle.elbow:=radtodeg*(BB+CC);
  next_angle.shoulder:=90-radtodeg*(p+BB);
  next_angle.wrist:=radtodeg*(p-CC);

  if (next_dest.x<0) then begin
    next_angle.elbow:=-1*next_angle.elbow;
    next_angle.shoulder:=-1*next_angle.shoulder;
    next_angle.wrist:=-1*next_angle.wrist;
    end;

  next_angle.z:=next_dest.z;
  end;
```

```
end;


FUNCTION CHECK_MOVE(dest_step:steps):boolean;{When provided with the desired
                                              number of steps for a move this
                                              function checks the values
                                              against the maximum number of
                                              steps permitted for each joint.}


begin

check_move:=true;

if ((abs(dest_step.shoulder)>max_steps.shoulder)
   or(abs(dest_step.elbow)>max_steps.elbow)
   or(abs(dest_step.z)>max_steps.z)
   or(abs(dest_step.wrist)>max_steps.wrist))
   then check_move:=false;

end;


PROCEDURE MOVE_RHINO(dest_step:steps);{Given a legal number of steps to move
                                       this routine converts these values to a
                                       string of the format required by the
external robot controller.  This routine effectively causes the robot to move.
It should be noted that the Rhino XR controller requires specialised routines
to enable the robot to move through a number of steps greater than 95.  These
are incorporated herein.}


var   moves        :steps;
      rsvp         :char;
      strs         :string[10];
      stre         :string[10];
      strz         :string[10];
      strw         :string[10];

begin

while not((dest_step.shoulder=0)and(dest_step.elbow=0)
        and(dest_step.z=0)and(dest_step.wrist=0)) do begin


if (dest_step.shoulder<>0) then begin

strs:='E?';
write(aux,strs);
read(aux,rsvp);
moves.shoulder:=95-ord(rsvp)+32;

if (dest_step.shoulder>0) then begin

   if(dest_step.shoulder<moves.shoulder) then begin
```

```
        moves.shoulder:=dest_step.shoulder;
        end;

    str(moves.shoulder,strs);
    strs:='E+'+strs;
    dest_step.shoulder:=dest_step.shoulder-moves.shoulder;
    end

else begin

    if(abs(dest_step.shoulder)<moves.shoulder) then begin
      moves.shoulder:=-1*dest_step.shoulder;
      end;
    str(moves.shoulder,strs);
    strs:='E-'+strs;
    dest_step.shoulder:=dest_step.shoulder+moves.shoulder;
    end;

strs:=strs+chr(13);
write(aux,strs);

end;

if (dest_step.elbow<>0) then begin

stre:='D?';
write(aux,stre);
read(aux,rsvp);
moves.elbow:=95-ord(rsvp)+32;

if (dest_step.elbow>0) then begin

    if(dest_step.elbow<moves.elbow) then begin
        moves.elbow:=dest_step.elbow;
        end;

    str(moves.elbow,stre);
    stre:='D+'+stre;
    dest_step.elbow:=dest_step.elbow-moves.elbow;
    end

else begin

    if(abs(dest_step.elbow)<moves.elbow) then begin
      moves.elbow:=-1*dest_step.elbow;
      end;
    str(moves.elbow,stre);
    stre:='D-'+stre;
    dest_step.elbow:=dest_step.elbow+moves.elbow;
    end;

stre:=stre+chr(13);
write(aux,stre);

end;
```

```
if (dest_step.wrist<>0) then begin

strw:='B?';
write(aux,strw);
read(aux,rsvp);
moves.wrist:=95-ord(rsvp)+32;

if (dest_step.wrist>0) then begin

    if(dest_step.wrist<moves.wrist) then begin
        moves.wrist:=dest_step.wrist;
        end;

    str(moves.wrist,strw);
    strw:='B+'+strw;
    dest_step.wrist:=dest_step.wrist-moves.wrist;
    end

else begin

    if(abs(dest_step.wrist)<moves.wrist) then begin
      moves.wrist:=-1*dest_step.wrist;
      end;
    str(moves.wrist,strw);
    strw:='B-'+strw;
    dest_step.wrist:=dest_step.wrist+moves.wrist;
    end;

strw:=strw+chr(13);
write(aux,strw);

end;

if (dest_step.z<>0) then begin

strz:='C?';
write(aux,strz);
read(aux,rsvp);
moves.z:=95-ord(rsvp)+32;

if (dest_step.z>0) then begin

    if(dest_step.z<moves.z) then begin
        moves.z:=dest_step.z;
        end;

    str(moves.z,strz);
    strz:='C+'+strz;
    dest_step.z:=dest_step.z-moves.z;
    end

else begin

    if(abs(dest_step.z)<moves.z) then begin
```

```
      moves.z:=-1*dest_step.z;
      end;
    str(moves.z,strz);
    strz:='C-'+strz;
    dest_step.z:=dest_step.z+moves.z;
    end;

strz:=strz+chr(13);
write(aux,strz);

end;

end;{while}

end;


PROCEDURE MOVE_GRIPPER(var newgrip :integer;  {NOTE: The gripper according to}
                       var lastgrip:integer); {the specifications is supposed}
                                              {to be open/close.  However it is
                                               not.  Thus the gripper may be
                                               moved through steps like the
                                               other joints.}

var strg      :string[10];
    gripmoves :integer;
    rsvp      :char;

begin

if(newgrip>60)then newgrip:=60;
if(newgrip<0) then newgrip:=0;

newgrip:=newgrip-lastgrip;
lastgrip:=newgrip+lastgrip;

while not(newgrip=0) do begin

  strg:='A?';
  write(aux,strg);
  read(aux,rsvp);
  gripmoves:=95-ord(rsvp)+32;

  if(newgrip>0) then begin

    if(newgrip<gripmoves) then gripmoves:=newgrip;
    str(gripmoves,strg);
    strg:='A+'+strg;
    newgrip:=newgrip-gripmoves;
    end

  else begin

    if(abs(newgrip)<gripmoves) then gripmoves:=-1*newgrip;
    str(gripmoves,strg);
```

```
    strg:='A-'+strg;
    newgrip:=newgrip+gripmoves;
    end;

  strg:=strg+chr(13);
  write(aux,strg);
  end;

end;


PROCEDURE GET_COORDINATES(var next_dest:posit;var done:boolean);

{This procedure prompts the user for the destination Cartesian coordinates.
In the conveyor control system this information would be provided by the
interpretation of the various positional sensors.}

var rsvp   :char;

begin

clrscr;
writeln(output);
writeln(output,'           CARTESIAN COORDINATE POSITIONING SYSTEM');
writeln(output);
writeln(output,'  This program demonstrates a position control sytem for the');
writeln(output,'RHINO  Robot.  The program requests as user input a point in');
writeln(output,'cartesian coordinates as illustrated in the documentation.');
writeln(output,'The program converts the point into a set of angles,');
writeln(output,'compares these with the robot''s joint ranges and moves the');
writeln(output,'robot into position if possible.');
writeln(output);
write(output,'MOVE ROBOT ? (y/n): ');
readln(input,rsvp);
if ((rsvp='Y')or(rsvp='y')) then begin
  done:=false;
  writeln(output);
  write(output,'ENTER X COORDINATE (in mm): ');
  readln(input,next_dest.x);
  write(output,'ENTER Y COORDINATE (in mm): ');
  readln(input,next_dest.y);
  write(output,'ENTER Z COORDINATE (in mm): ');
  readln(input,next_dest.z);
  write(output,'ENTER GRIPPER POSITION (0 to 60): ');
  readln(input,next_dest.gripper);
  end
  else done:=true;

end;


PROCEDURE FIND_STEPS(next_angle:angles; var dest_step:steps;
                                        var last_dest:steps;
                                        var movesteps:boolean);
```

```
{When the angles have been computed then these values must be converted to
an integer number of steps through which the joint is to travel.  This is why
the robot may only approximate a Cartesian point and may not necessarily move
exactly to the desired point.  This routine also considers the present
position of the robot and computes the number of steps required to move from
the present position to the new one.  NOTE WELL:  Position is considered as a
number of joint steps rather than a set of joint angles.  This is the more
accurate method since the joints cannot move to the exact angles specified.}

begin

dest_step.shoulder:=round(next_angle.shoulder*step_size.shoulder);
dest_step.elbow:=round(next_angle.elbow*step_size.elbow);
dest_step.wrist:=round(next_angle.wrist*step_size.wrist);
dest_step.z:=round(next_angle.z*step_size.z);

movesteps:=check_move(dest_step);

if(movesteps) then begin
  dest_step.shoulder:=dest_step.shoulder-last_dest.shoulder;
  last_dest.shoulder:=dest_step.shoulder+last_dest.shoulder;
  dest_step.elbow:=dest_step.elbow-last_dest.elbow;
  last_dest.elbow:=dest_step.elbow+last_dest.elbow;
  dest_step.wrist:=dest_step.wrist-last_dest.wrist;
  last_dest.wrist:=dest_step.wrist+last_dest.wrist;
  dest_step.z:=dest_step.z-last_dest.z;
  last_dest.z:=dest_step.z+last_dest.z;
  end;

end;



begin  {MAIN PROGRAM}

{The main program initializes various variables and uses the preceeding
functions and procedures to move the robot to Cartesian points within the
operating space of the robot.}

last_dest.shoulder:=0;
last_dest.elbow:=0;
last_dest.z:=0;
last_dest.wrist:=0;

dest_step.shoulder:=0;
dest_step.elbow:=0;
dest_step.z:=0;
dest_step.wrist:=0;
done:=false;

get_coordinates(next_dest,done);

while not(done) begin

 next_dest.x:=150;
```

```
next_dest.z:=0;
next_dest.gripper:=0;

for i:=0 to 20 do begin
  next_dest.y:=400 - i*10;

  find_angles(next_dest,next_angle,moveangles);


  if (moveangles) then begin

    find_steps(next_angle,dest_step,last_dest,movesteps);

    if (movesteps) then begin

  write(lst,' ');
  write(lst,next_angle.shoulder);
  write(lst,' ');
  write(lst,next_angle.elbow);
  write(lst,' ');
  write(lst,next_angle.wrist);

  write(lst,' ');
  write(lst,last_dest.shoulder);
  write(lst,' ');
  write(lst,last_dest.elbow);
  write(lst,' ');
  write(lst,last_dest.wrist);

  write(lst,' ');
  write(lst,dest_step.shoulder);
  write(lst,' ');
  write(lst,dest_step.elbow);
  write(lst,' ');
  write(lst,dest_step.wrist);

  true_angle.shoulder:=last_dest.shoulder/step_size.shoulder;
  true_angle.elbow:=last_dest.elbow/step_size.elbow;
  true_angle.wrist:=last_dest.wrist/step_size.wrist;
  write(lst,' ');
  write(lst,true_angle.shoulder);
  write(lst,' ');
  write(lst,true_angle.elbow);
  write(lst,' ');
  writeln(lst,true_angle.wrist);

  move_rhino(dest_step);
      move_gripper(next_dest.gripper,last_gripper);
      end
    else begin
      writeln(output);
      writeln(output,'*** ERROR:  Desired coordinates out of range. ***');
      write(output,'            <RETURN> to continue.');
      readln(input);
      end;{if}
```

```
    end

  else begin
    writeln(output);
    writeln(output,'*** ERROR:  Desired angles out of range. ***');
    write(output,'             <RETURN> to continue.');
    readln(input);
    end;

 get_coordinates(next_dest,done);

  end;{while}

  end;{x}
end.{MAIN}
```

BIBLIOGRAPHY

Ballard, D.H., and C.M. Brown, Computer Vision, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982.

Critchlow, A.J., Introduction to Robotics, Macmillan Publishing Company, U.S.A., 1985.

Fu, K.S., R.C. Gonzales, and C.S.G. Lee, Robotics: Control, Sensing, Vision, and Intelligence, McGraw-Hill Inc., U.S.A., 1987.

Goguen, J.P. Thomas, and E.M. Petriu, "Progressive Image Resolution Techniques For Real-Time Image Processing", Proceedings of Montech '87, Montreal, 1987.

Goguen, J.P. Thomas, "A Specialised Architecture For Progressive Image Processing", SME Vision '88, Detroit, Michigan, 1988.

Gonzales, R.C., and P. Wintz, Digital Image Processing, Addison-Wesley Publishing Company, U.S.A., 1977.

Granlund, G.H., and H. Knutsson, "Hierarchical Processing of Structural Information in Artificial Intelligence", Proceedings of the ICASS, Paris, 1982.

Hendrickson, Tom, and Harprit Sandhu, Rhino SCARA Owner's Manual, Ver. 1.00, Copyright Rhino Robots, Champaign, Illinois, August 14, 1986.

Hendrickson, Tom, and Harprit Sandhu, XR-3 Robot Arm and Mark III 8 Axis Controller Owner's Manual, Ver. 3.00, Copyright Rhino Robots, Champaign, Illinois, November 1, 1986.

Hirzinger, G., "Robot Systems Completely Based on Sensory Feedback", IEEE Transactions on Industrial Electronics, Vol. IE-33, No. 2, May, 1986.

Knowlton, K., "Progressive Transmission of Grey-Scale and Binary Pictures By Simple, Efficient, and Lossless Encoding Schemes", Proceedings of IEEE, Vol. 68, No. 7, July, 1980.

Larin, David J., "Vision's Next Steps", Manufacturing Engineering, December, 1986.

Lozano-Perez, T., "Robot Programming", Proceedings of the IEEE, Vol. 71, No. 7, 1983.

Nevins, J.L., and D.E. Whitney, "Computer-Controlled Assembly", Scientific American, February, 1978.

Petriu, E.M., "Absolute-Type Position Transducers Using a Pseudorandom Encoding", IEEE Transactions on Instrumentation and Measurement, Vol. IM-36, No. 4, December, 1987.

Petriu, E.M., J.P.T. Goguen et al., "Multi-Sensor Robotic Tracking and Grasping", Proceedings of ISA Digicom '87, Montreal, 1987.

Petriu, E.M., J.P.T. Goguen et al., "Robotic Assembly Architecture For a Task-Level Programming Language", Proceedings of SME Robots 12, Detroit, Michigan, 1988.

Sanderson, A.C., "Parts Entropy Methods For Robotic Assembly Systems Design", Proceedings of IEEE International Conference on Robotics, Atlanta, Ga., 1984.

Saradis, G.N., "Control Performance as an Entropy: An Integrated Theory For Intelligent Machines", Proceedings of the IEEE Conference on Robotics, Atlanta, Ga., March, 1984.

Shin, K.G., and S.T. Malin, "A Structured Framework for the Control of Industrial Manipulators", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-15, No. 1, Jan./Feb. 1985.

Snyder, W.E., Industrial Robots: Computer Interfacing and Control, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1985.

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA