

AD-A171 278

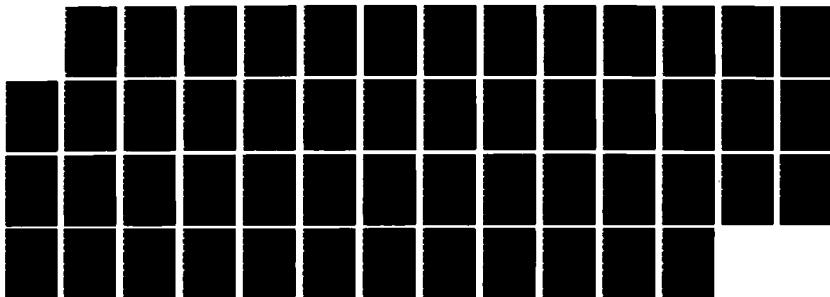
INTEGRATING COMPUTER VISION WITH A ROBOT ARM SYSEEN(U)  
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH  
J E WINGATE 1986 AFIT/CI/NR-86-127T

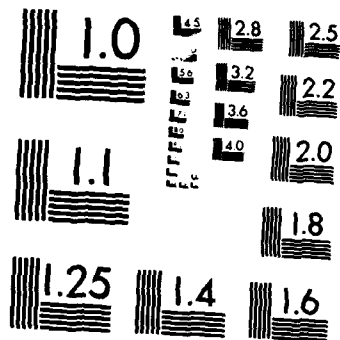
1/1

UNCLASSIFIED

F/G 17/8

ML





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS  
BEFORE COMPLETING FORM

1. REPORT NUMBER AFIT/CI/NR 86-127T		2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Integrating Computer Vision With A Robot Arm System		5. TYPE OF REPORT & PERIOD COVERED THESIS/DISSERTATION	
7. AUTHOR(s) James E. Wingate		6. PERFORMING ORG. REPORT NUMBER	
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: University of South Florida		8. CONTRACT OR GRANT NUMBER(s)	
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433-6583		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE 1986	
		13. NUMBER OF PAGES 41	
		15. SECURITY CLASS. (of this report) UNCLAS	
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) E			
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1 LYNN E. WOLAVER 13 Aug 86 Dean for Research and Professional Development AFIT/NR			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED.			

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A171 278

DTIC FILE COPY

Graduate Council  
University of South Florida  
Tampa, Florida

CERTIFICATE OF APPROVAL



Master's PROJECT

This is to certify that the Master's Project of

James E. Wingate

with a major in Computer Science and Engineering  
has been approved by the Examining Committee  
on April 10, 1986, as satisfactory for the  
Project requirement for the Master of Science degree.

Accession For	
NTIS (CRAI)	X
DTIC TAB	
Unannounced	
Justification	
By	
Distribution	
Availability	
Dist	
A-1	

Project Committee:

Rafael Perez  
Major Professor: R. A. Perez, Ph.D.

R. Turner  
Member: R. Turner, Ph.D.

Ken W. Bowyer  
Member: K. W. Bowyer, Ph.D.

INTEGRATING COMPUTER VISION

WITH

A ROBOT ARM SYSTEM

by

James E. Wingate

A graduate project submitted in partial fulfillment of the requirements  
for the degree of Master of Science  
in the Department of Computer Science and Engineering in  
the University of South Florida

May, 1986

Major Professor: R. A. Perez, Ph.D.

**COST ESTIMATION FOR REPLACEMENT OF LARGE COMPUTER  
SYSTEMS: CHANGE OF VENDORS**

**By**

**Mary C. Cobble, B.S., M.S.**

**THESIS**

**Presented to the Faculty of Trinity University  
in Partial Fulfillment of the Requirements**

**For the Degree**

**Master of Science in Computing and Information Sciences**

**TRINITY UNIVERSITY  
May 1986**

#### ACKNOWLEDGEMENT

I am deeply indebted to Dr. R. A. Perez for his valuable suggestions and criticism in his capacity as major professor and advisor. His advice, as well as that of Dr. R. Turner and Dr. K. W. Bowyer, has been very helpful in encouraging and guiding this project. I also wish to thank Mr. Joe Gomes for his cooperation and assistance in the use of the Grinnell image processing system, the VAX 11/750, the DUAL microcomputer, and the robots. Finally, I would like to thank my wife, Gloria, and my sons, James III and Joshua, for their patience and understanding during the course of this project.

## TABLE OF CONTENTS

LIST OF FIGURES . . . . .	v
ABSTRACT . . . . .	vi
<u>Chapter</u>	
I. INTRODUCTION . . . . .	1
HARDWARE CONFIGURATION AND INFORMATION FLOW . . . . .	3
ORGANIZATION . . . . .	5
II. IMAGE ACQUISITION . . . . .	7
GENERATING THE GRADIENT MAP . . . . .	8
UPLOADING THE GRADIENT MAP . . . . .	8
Transferring Data . . . . .	9
i8uss: Upload Sequential - Sequential . . . . .	9
LOCATING EDGE PIXELS . . . . .	10
DETERMINING EDGE PIXEL COORDINATES . . . . .	10
III. FEATURE EXTRACTION . . . . .	16
THE CORNER FINDING ALGORITHM . . . . .	18
LOCATION OF OBJECTS IN THE WORK SPACE . . . . .	20
Work Space Coordinates . . . . .	21
Planar Projective Mapping . . . . .	21
Computation of Centroid . . . . .	24
ORIENTATION OF OBJECTS IN THE WORK SPACE . . . . .	24
Computation of Roll Angle . . . . .	25
IV. GRASPING AND MANIPULATING THE OBJECT . . . . .	27
ROBOT INITIALIZATION . . . . .	27
COMMUNICATION BETWEEN THE VAX AND THE DUAL . . . . .	28
GRASPING THE OBJECT . . . . .	29
V. SUMMARY AND CONCLUSIONS . . . . .	30
SUGGESTIONS FOR FUTURE RESEARCH . . . . .	31
Noise Suppression . . . . .	31



Multiple Object Recognition . . . . .	32
Stereo Imaging . . . . .	32
Chain Coding . . . . .	32
REFERENCES . . . . .	33
APPENDIXES . . . . .	34
A. USER'S GUIDE . . . . .	35
B. DETERMINING THE CAMERA'S FIELD OF VIEW . . . . .	38

## LIST OF FIGURES

### Figure

1. High-level Flowchart of Robot Vision System . . . . .	4
2. Partial Gradient Map in Grinnell Image Memory . . . . .	12
3. Determining Edge Pixel Coordinates . . . . .	15
4. High-level Flowchart of Feature Extraction Algorithm . . . . .	17
5. Corner Labeling Convention . . . . .	19
6. Effect of Planar Projective Mapping . . . . .	23
7. Block Placement to Locate Camera's Field of View . . . . .	39

INTEGRATING COMPUTER VISION

WITH

A ROBOT ARM SYSTEM

by

James E. Wingate

An Abstract

Of a graduate project submitted in partial fulfillment  
of the requirements for the degree of Master of Science  
in the Department of Computer Science and Engineering in  
the University of South Florida

May, 1986

Major Professor: R. A. Perez, Ph.D.

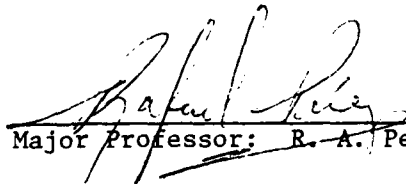
The goal of this project is to interface an image processing system to a robot arm system. Computer vision is used to compute the location and orientation of a block in the work space so that the block may be grasped and manipulated by a robot arm. A detailed explanation of how information flows from the image processor to the robot arm system is given. This flow can be broken down into three different steps.

In the first step, a gradient map from a digitized image of the work space is transferred, one line at a time, from the Grinnell image processor to its host, a VAX 11/750. As each line of the gradient map is uploaded, the coordinates of pixels on the edges of a block (edge pixels) are stored in an array. Storing the coordinates of only the edge pixels results in significant image compression.

The second step involves processing the edge pixel coordinate array to extract relevant features. The location of the corners of a block is all that is needed to compute the location of the centroid and the orientation of the block in the work space.

In step three, the centroid and roll angle information is sent from the VAX to the robot arm system host, a DUAL microcomputer. One of the robot arms is selected and appropriate commands are sent to that arm to grasp and manipulate the block.

Abstract approved:

  
Major Professor: R. A. Perez, Ph.D.

Asst. Prof., Dept. of Comp. Sc. and Engineering

4/3/86  
Date of Approval

## Chapter I

### INTRODUCTION

Our ability to interact with our environment is based on sensory perception. Information about our environment is continuously collected by our sensory mechanisms and is processed by our brain. Consequently, this sensing/processing system gives us the extensive information we require to function effectively [1].

One of our senses, in particular, has extraordinary sensitivity. The human visual system can detect the location, motion, shape, size, color, and texture of objects without making any physical contact. Duplicating this sensitivity electronically, in even rudimentary form, requires an immense amount of memory and programming. However, since this highly valuable sensory input enables robots to interact very effectively with their environment, vision in robotics has become an area of intense research activity [2], and some applications have become commercially feasible.

Gonzalez and Safabakhsh divided computer vision techniques into three levels of processing: low, medium, and high-level vision [3]. To understand the relationship of these levels of processing, one could think of low-level vision as a procedure that finds the edges of an object, medium-level vision as a process that recognizes an object as a distinct entity in the work space, and high-level vision as a means of interpreting a scene and determining the interrelationships, if any

exist, among several objects (e.g., a big block next to, or near, a small block). This project implemented medium-level vision techniques to determine the location and orientation of a block in the work space. This was accomplished by augmenting and integrating the previously completed work of two fellow graduate students. Their work is briefly described below.

Majumdar [4] used low-level vision techniques to detect the edges of an object in the work space. He developed several edge detection algorithms that execute in the Grinnell image processor under the control of a program running in the host computer, a VAX 11/750. The most significant aspect of his work is the speed in which edges are detected. Using Roberts' cross operator on a 480 by 512 pixel image, an edge map is generated in approximately 300 milliseconds. However, it is important to note that, upon completion of his edge detection algorithms, the edge map remains in the image memory of the Grinnell. Thus, any application that requires the edge map for further processing, which cannot be accomplished by the Grinnell, must upload the map to the host.

Koutsourelis [5] implemented a high-level robot arm command language to provide concurrent operation, and coordination, of multiple robot arms. Simultaneous movement of the arms is obtained by creating separate processes running in parallel under the UNIX operating system [6]. This software is available in the robot arm system host computer, a DUAL System 83 microcomputer.

The results of Majumdar's edge detection algorithms provide the input for the algorithms designed and implemented in this work. These algorithms determine the location and orientation of a block in the

work space. This information is then transmitted from the VAX to the DUAL to be used as input to Koutsourelis' robot arm command language which is used to send appropriate commands to the robot to grasp and manipulate the block.

### 1.1 HARDWARE CONFIGURATION AND INFORMATION FLOW

The USF Computer Science and Engineering Robotics Laboratory (hereinafter referred to as the Robotics Laboratory) is equipped with a Grinnell 2800-32 image processing system (the Grinnell) configured as a peripheral device of the host computer, a VAX 11/750 (the VAX). A GE Model 2507 CID TV camera, mounted orthogonal to, and above, the work space, provides the video input for the Grinnell. A black and white video monitor displays the scene viewed by the camera. The camera/monitor/Grinnell/VAX configuration is often referred to as the image processing system.

A block in the work space will ultimately be grasped by one of the two MICROBOT Alpha robot arms or the Mitsubishi RM-501 robot arm. Each of these robots is connected to the DUAL System 83 MC68000-based microcomputer via an RS-232 interface. The MICROBOT/Mitsubishi/DUAL configuration is often referred to as the robot arm system.

Information flows from the work space to the robot arm system via a circuitous path (see Figure 1). After a block is placed in the work space: (1) an image is transmitted from the TV camera to the Grinnell where a gradient map is generated, (2) the gradient map is uploaded to the VAX and the coordinates of pixels on the edges of the block (edge pixels) are stored in an array, (3) relevant information is extracted from the edge pixel coordinate array and location and orientation data

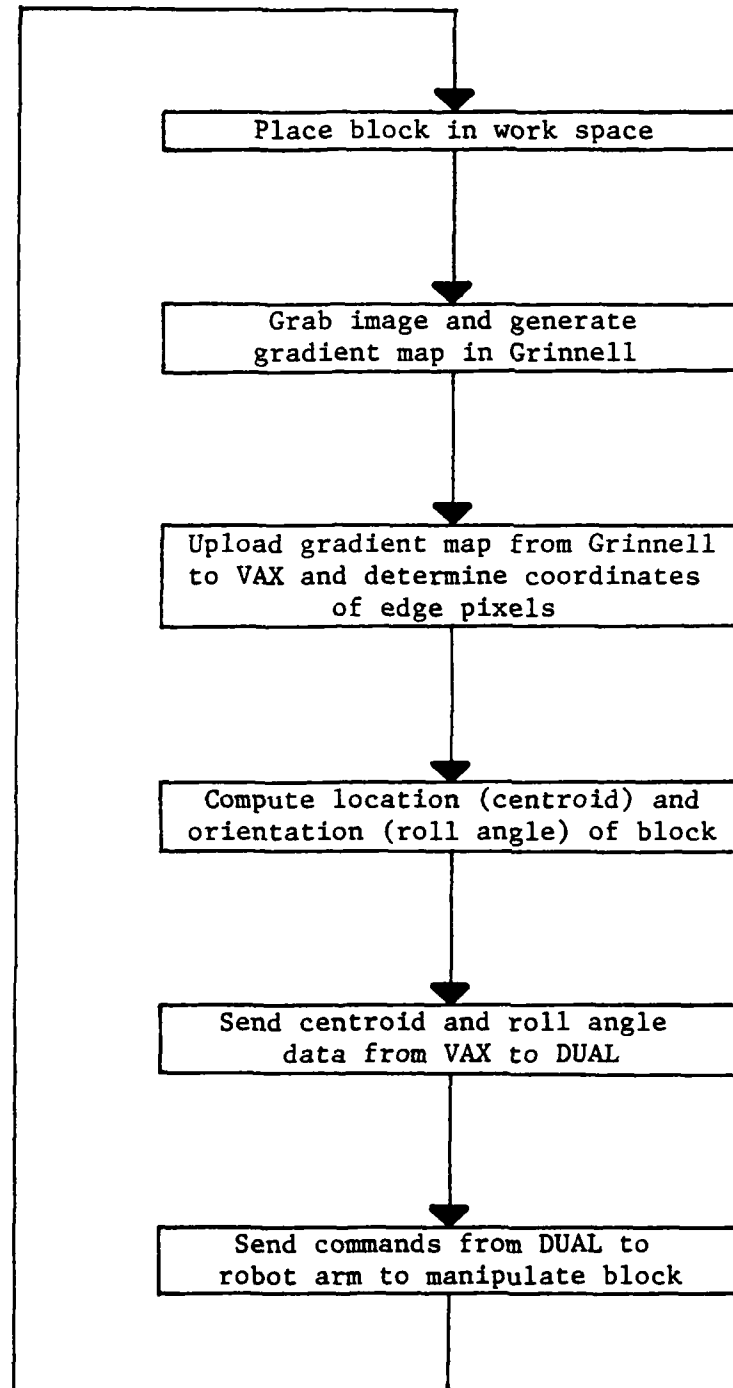


Figure 1. High-level Flowchart of Robot Vision System



is computed, (4) the location of the centroid and the roll angle is sent to the DUAL, and finally (5) one of the robots is instructed to grasp and manipulate the block. This process may be repeated as many times as is desired.

## 1.2 ORGANIZATION

Chapter II describes the procedure used to acquire and process an image of the work space. The data structures used to store lines of the image for intermediate processing (the pixel buffer) and to store the coordinates of edge pixels (the coordinate array) are described as well.

Extracting the relevant features of an object is the topic of Chapter III. To classify an object requires knowledge about certain features of the object. Some of these features are: area, centroid, perimeter length, maximum dimension, etc. [1,4]. For this project, only the location and orientation of the object are required. The location of a block can be determined by locating the four corners of the block; the orientation can then be determined from the lengths of two of the four edges between three of the four corners.

Prior to using this vision system, the robot arm system must be initialized and physically connected to the image processing system. Chapter IV describes the initialization procedure and the actions required to connect the VAX to the DUAL. Grasping and manipulating a block is also discussed along with actions taken in response to error conditions.

Chapter V provides a summary of this project and suggested topics for future research. Although this is a very simple implementation of

computer vision, this project provides a base for the development of more complex applications. Additional work could transform this simple vision system into a more powerful, and versatile, academic tool for demonstrating the use of computer vision in robotics applications.

Appendix A includes a complete User's Guide. This guide was written with the assumption that the user would be familiar with the location, and general configuration, of the various devices that form the image processing system and the robot arm system. Step-by-step instructions for using this vision system are provided.

Appendix B describes the procedure used to determine the location of the camera's field of view on the xy plane of the work surface. It is important to note that the position of the camera fixes the location of the work space! Should the position of the camera change, the location of the work space will change as well. This procedure may be used to reestablish the location of the work space.

## Chapter II

### IMAGE ACQUISITION

The TV camera is mounted orthogonal to, and approximately 27 inches above, the surface of the table on which the robots have been placed. A black and white video monitor displays the scene viewed by the camera. To implement this vision system, it was necessary to determine the field of vision (i.e., "visual" work space) of the robots. Consequently, we had to determine the location, and size, of the area of the table viewed by the camera. Thus, determining the robots' work space is analogous to determining the camera's field of view. A procedure to do this is given in Appendix B.

When a robot has been placed relative to the work space, its arm is situated such that part of it appears within the camera's field of view. Prior to analyzing a scene, the robot is instructed to move to the side to keep its arm from appearing within the image.

When instructed to "grab an image," the Grinnell accepts a frame of data from the camera. An analog-to-digital converter in the Grinnell converts the analog signal into a gray-scale digital image. This image is then stored in one of the Grinnell's twelve 512 by 512 pixel image memory banks. The intensity of each pixel in the image is represented by an 8-bit value. A value of 0 represents minimum intensity, or black-level; a value of 255 represents maximum intensity, or white-level.

Algorithms to detect the edges of objects in the work space have

been implemented in the Grinnell [4]. These algorithms are implemented using several methods discussed in the literature [1,3,4,7,8]. The concept of using a gradient operator to detect the edges of an object in the work space is briefly reviewed in the following section.

## 2.1 GENERATING THE GRADIENT MAP

To isolate the edges of an object, a gradient map is generated by applying a gradient operator to every pixel in the image. The operator serves to indicate the presence, and magnitude, of local discontinuities in intensity within the image. The magnitude of the gradient values of edge pixels will be large, but less than 255; whereas, the magnitude of the gradient values of all other pixels will be close to zero.

Majumdar designed and implemented algorithms that use both Sobel's 3 by 3 operator and Roberts' cross operator. Although it provides more complete edge segments, Sobel's operator takes more than three times longer to generate a gradient map than does Roberts' operator [4]. For this application, the quality of edges is not a major concern. The corner finding algorithm developed for this project, and described in 3.1, is not sensitive to the quality of edge segments - it simply attempts to locate corner points. Thus, Majumdar's algorithm that implemented Roberts' cross operator was chosen for this particular application.

## 2.2 UPLOADING THE GRADIENT MAP

As a special purpose computer with a bit-slice ALU and a pipelined video processor, the Grinnell was designed for high-speed graphics and image processing applications. Due to the special purpose nature of this architecture, we have yet to find ways to implement conventional

programming logic in the Grinnell. Consequently, for any processing of this type, it is necessary to upload the gradient map to the VAX.

#### 2.2.1 Transferring Data

The Grinnell is configured as a peripheral device of the VAX. The Intelligent Host Interface in the Grinnell is connected to the Host Resident Adaptor Board in the VAX through a 16-bit parallel data cable. Although the time required to physically transfer data is on the order of microseconds, system overhead for each I/O operation is on the order of milliseconds. To upload the gradient map from the Grinnell requires 480 transfers of 512 16-bit gradient values (the high-order eight bits of each value are padded with zeroes). Thus, a very large percentage of the time required to upload the gradient map is due to system overhead resulting from I/O operations between the Grinnell and the VAX.

To effect the transfer of data, the applications programmer uses existing software routines, the Primary Interface Primitives. These primitive functions define the lowest level of interaction between a program running in the VAX and the Grinnell device driver. The primitive used to upload the gradient map is described below.

#### 2.2.2 i8uss: Upload Sequential - Sequential

This function transfers a block of data from the Grinnell to the VAX. Data is accessed in the Grinnell at sequentially increasing addresses and is placed into sequentially increasing addresses in the VAX. Refer to [4] for a more detailed explanation of this, and other, primitive functions.

The following is an example of uploading one line of the gradient map from a Grinnell image memory bank to a buffer in the VAX:

```

/* Create Buffer */

unsigned short pixel_buffer[512];

/* Address of Line 5 - Image Card 1 (I01) */

up_addr = 0200052000L;

/* Upload Line 5 */

i8uss(GrinnellFD, pixel_buffer, up_addr, 512);

```

The programs designed and implemented for this project were written in C. Refer to [9] for more information on this programming language.

### 2.3 LOCATING EDGE PIXELS

Lighting is a very important factor in image processing work. Its effects vary with the shape, color, texture, and location of objects in the work space. Because of this, we provided a mechanism to allow the user to select the value above which a pixel is considered to be an edge pixel. Edge pixels are located by applying this global intensity threshold to every pixel in the gradient map. Pixels with intensity value above the threshold are edge pixels - they are "on the edges" of the object; and, pixels with intensity value below the threshold are not edge pixels. Occasionally, extraneous pixels are encountered in the gradient map. These pixels are usually the result of incorrect lighting adjustment or are caused by some other type of noise. Refer to Appendix A, steps 8 and 9, for more information on lighting adjustment and threshold selection.

### 2.5 DETERMINING EDGE PIXEL COORDINATES

Because the focal point of the lens in the TV camera is behind the lens, all objects in the work space appear in an inverted orientation

both on the video monitor and in the Grinnell's image memory. An object placed in the lower-left area of the work space would appear in the upper-right area of both the monitor and the Grinnell's memory. Thus, the coordinates of pixels in the gradient map do not correspond to the physical position of the object in the work space. In order to reorient the pixel coordinates of the image, the gradient map is processed in reverse order. Instead of starting with the pixel in the upper-left corner and proceeding in the normal top-down, left-to-right order, we start with the pixel in the lower-right corner and process the map in bottom-up, right-to-left order.

An illustration of a simplified image in gradient map form is given in Figure 2. In this image, edge pixels are those pixels with gradient value greater than 70. Note that, in an actual image, the edges would be more than one pixel wide and there would be clusters of pixels at the vertices of the edges. Although there are routines in the VAX for thinning edges, it was not necessary to use these routines in this application because the inaccuracies caused by wider edges were negligible when finding corner locations.

The gradient map is uploaded to the VAX, one line at a time, under the control of a for-loop. The for-loop index,  $y$ , is initialized to 479 and is decremented by 1 until the index is equal to -1 (480 iterations). As each line is uploaded, the gradient values are temporarily stored in a 512-word buffer, the pixel buffer. The for-loop index,  $y$ , serves as the line number of the line currently in the buffer. Thus, the map is uploaded in bottom-up order.

A for-loop is also used to process the gradient values in the pixel buffer. The for-loop index,  $x$ , is initialized to 509 and is decremented

			3	3	3	3	3	3	3	3	3	5
		0	0	0	1	1	1	1	1	1	...	1
0	...	8	9	0	1	2	3	4	5	6		1
0												
.												
.												
.												
126		4	3	5	16	41	68	35	4	3		
127		1	2	6	13	52	132	47	17	2		
128		5	4	3	58	124	8	129	65	15		
129		3	21	55	121	13	5	9	130	29		
130		13	46	117	17	3	11	129	69	13		
131		62	119	18	6	17	123	59	28	9		
132		19	66	103	14	105	39	24	8	5		
133		11	25	62	99	63	16	9	6	3		
134		2	1	19	48	51	32	8	4	2		
.												
.												
.												
479												

Figure 2. Partial Gradient Map in Grinnell Image Memory



by 1 until the index is equal to 2. Thus, lines from the gradient map are processed in right-to-left order. Also, note that the values in the two right-most, and two left-most, locations of the pixel buffer are not processed. This was done intentionally due to noise at the boundary of the image.

Each gradient value in the buffer, except those noted above, is compared to the global intensity threshold. Gradient values less than the threshold are ignored. A gradient value greater than the threshold indicates that an edge pixel has been located. The coordinates of this pixel are stored in the edge pixel coordinate array.

If it were necessary to store the entire image, a 480 by 508 word matrix would be required. This would occupy 243,840 words of memory! All that is really needed, however, is to store the (x,y) coordinates of the edge pixels. A structured array of 1000 elements is used to store the coordinates of edge pixels from the images of objects used in this project. Each element of the edge pixel coordinate array contains two values - the horizontal position of the edge pixel and the vertical position of the edge pixel, the x, and y, components of the edge pixel coordinate pair, respectively. Using this storage scheme results in significant image compression.

The coordinates of an edge pixel are stored as follows: the x component of the coordinate pair becomes equal to 511 minus the value of the for-loop index, x, and the y component becomes equal to the value of the for-loop index, y, the line number. Thus, by processing the gradient map in reverse order and by using the for-loop indices in this manner, the coordinates of the edge pixels from the gradient map are reoriented so that they correspond to the physical position of the

object in the work space.

Figure 3 illustrates the process of determining the coordinates of edge pixels. In Figure 3(a), line 130 of the gradient map (from Figure 2) has been uploaded and stored in the pixel buffer. With an intensity threshold value of 70, the gradient values in locations 310 and 314 of the buffer qualify as edge pixels. In Figure 3(b), the coordinates of these pixels are stored in the edge pixel coordinate array. Note that, to reorient the coordinates of edge pixels in the buffer, the buffer is processed in right-to-left order. Thus, the coordinates of the edge pixel at location 314 were stored in the array before the coordinates of the edge pixel at location 310. This figure also shows the contents of the coordinate array after the entire gradient map has been uploaded and processed.



### Chapter III

#### FEATURE EXTRACTION

The shape of objects recognized by the vision algorithms designed for this project was intentionally restricted. The reason for doing this was to facilitate the initial implementation of computer vision in the Robotics Laboratory. The feature extraction algorithm was specifically designed to determine the location and orientation of either a rectangular block or a cube. Hereinafter, the term "block" is used to refer to either a rectangular block or a cube.

Restricting the shape of objects to be recognized reduces the number of relevant features that must be determined. Because we are working with blocks, we need to extract only two features from an image of the work space. These are the location and the orientation of the block. The location of a block can be determined by locating its four corners. The orientation of this block can be determined from the relationship that exists between the lengths of any two perpendicular edges between any three corners.

This chapter describes the procedures that were designed and implemented to compute the location and orientation of a block in the work space. The effect of planar projective mapping is discussed as is the algorithm that was designed to compensate for its effects. Figure 4 provides a high-level flowchart of the feature extraction algorithm.

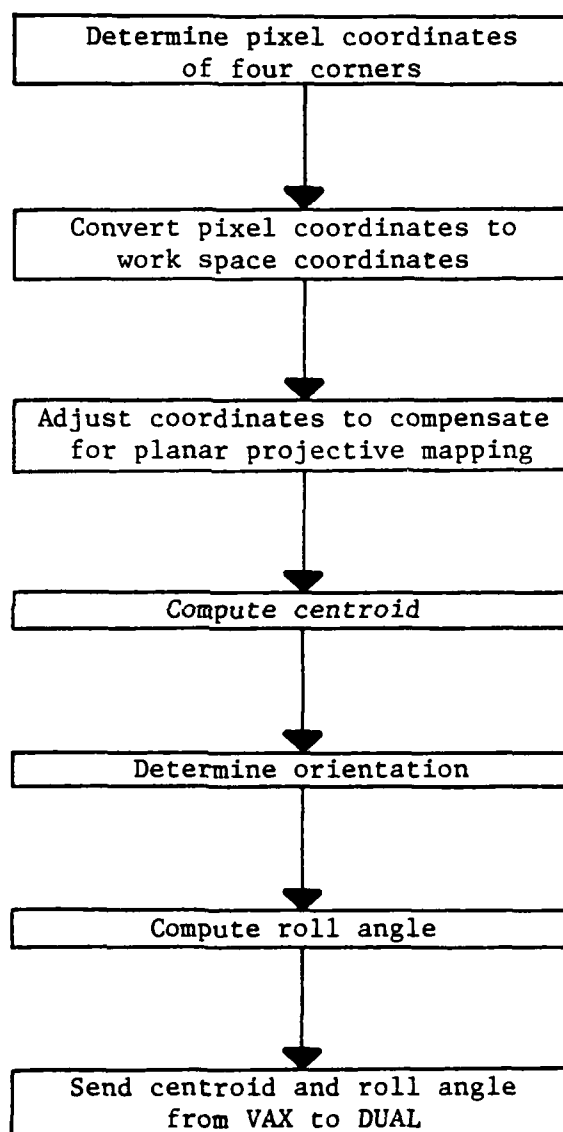


Figure 4. High-level Flowchart of Feature Extraction Algorithm

### 3.1 THE CORNER FINDING ALGORITHM

The basic assumption made in this algorithm is that, regardless of the orientation of an object in the work space, the first edge pixel encountered represents a corner of the object and the last edge pixel encountered also represents a corner. Thus, the first and last pair of coordinates in the edge pixel coordinate array represent two of the four corners of the object in the work space.

Figure 5 illustrates the corner labeling convention used in this project. In Figure 5(a), the inverted orientation of a block is shown as it would look in the Grinnell's image memory. Figure 5(b) shows the reoriented pixel coordinates of the four corners of this block.

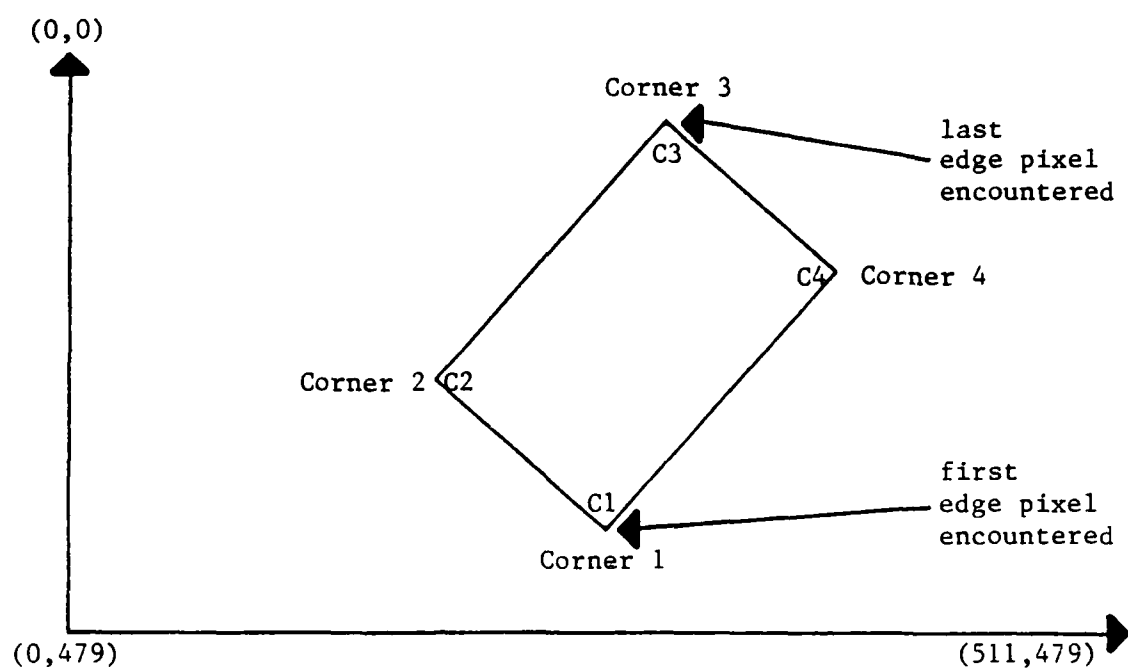
The goal of the corner finding algorithm is to determine the pixel coordinates of the four corners of a block. A detailed description of this algorithm is given below. Refer to Figures 5(a) and 5(b) for corner locations.

Step 1: From the edge pixel coordinate array, select the first and last coordinate pair as the coordinates for C1 and C3, respectively.

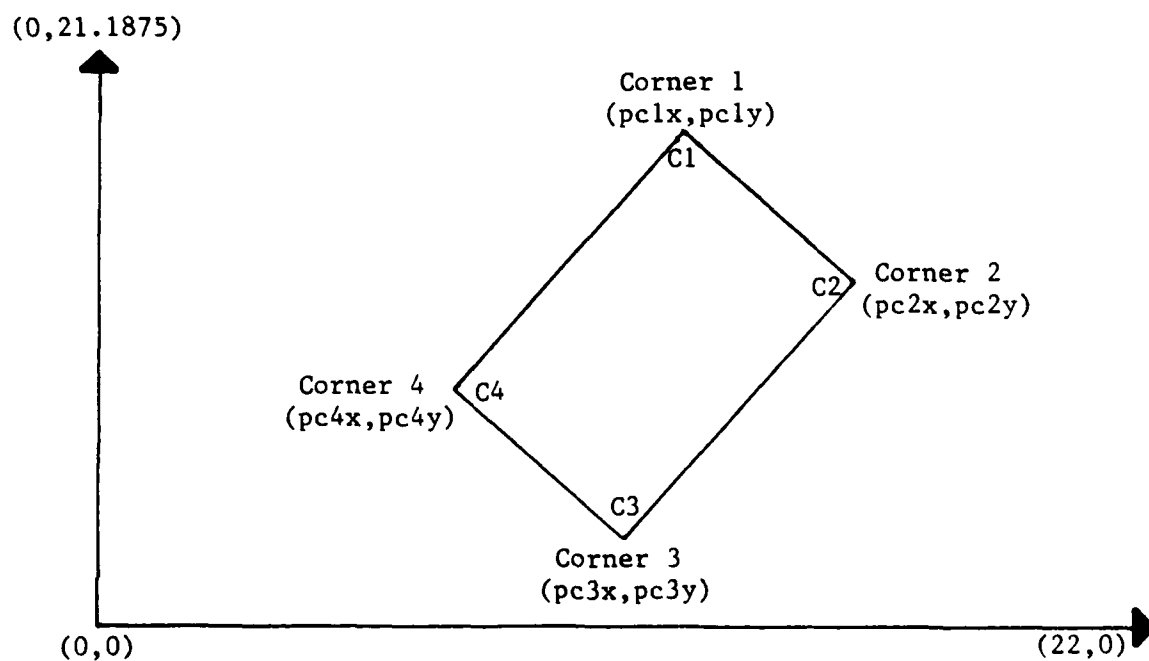
Step 2: Search the edge pixel coordinate array for the coordinate pair with the maximum x component. Select this coordinate pair as the coordinates for C2.

At this point, the coordinates of C1, C2, and C3 have been found. Now, we know that the object in the image is either a rectangular block or a cube; therefore, the edge between C1 and C2 should be perpendicular to the edge between C2 and C3. However, in actual practice, this was simply not the case!

In an actual image, there are clusters of edge pixels around the



(a) Corners of a Block in Grinnell's Image Memory



(b) Corners of a Block in the Work Space

Figure 5. Corner Labeling Convention

corners of an object. From the cluster of edge pixels around C1, we arbitrarily let the coordinates of the first edge pixel be the coordinates of C1; and, from the cluster around C3, we arbitrarily let the coordinates of the last edge pixel be the coordinates of C3. From the cluster of edge pixels around C2, we arbitrarily selected the coordinates of the edge pixel with the maximum x component as the coordinates of C2. Thus, the arbitrary selection of these three points is the reason the edge between C1 and C2 is not perpendicular to the edge between C2 and C3. Furthermore, the coordinates of C4 are projected, based on the relationship of C2 to C1 and C3; and, because the coordinates of C2 are inaccurate, the coordinates of C4 will be inaccurate as well. Finally, subsequent computations will use the coordinates of C3 and the coordinates of either C2, or C4, depending on the orientation of the block, to determine the robot's roll angle. Should the coordinates of C2 be used as selected in Step 2, the roll angle would be adversely affected. So, the coordinates of C2 must be adjusted. A procedure to do this is given below.

Step 3: Place the center of a 9 by 11 matrix at C2. Beginning with the point (pc2x-5, pc2y+4), determine the coordinates of the point in this matrix that yields the "best fit" for a right angle between C1 and C3. Select the coordinate pair of this point as the coordinates for C2.

Step 4: Compute the coordinates of C4 as follows:

$$pc4x = pc3x - (pc2x - pclx);$$

$$pc4y = pc3y - (pc2y - pcly);$$

### 3.2 LOCATION OF OBJECTS IN THE WORK SPACE

The camera's field of view on the xy plane of the work surface was



located using the procedure in Appendix B. The horizontal length of the work space is 22 inches and the vertical length is 21.1875 inches. To convert image coordinates (pixels) to work space coordinates (inches), the pixel coordinates of the four corners of a block are divided by a scale factor equal to the number of pixels per inch along the x, and y, axis, respectively.

### 3.2.1 Work Space Coordinates

The scale factor along the x axis,  $s_x$ , is computed as follows:

$$\begin{aligned} s_x &= \text{Horizontal Resolution} / \text{Horizontal Distance} \\ &= 512 \text{ pixels} / 22 \text{ inches} \\ &= 23.272727 \text{ pixels per inch} \end{aligned}$$

The scale factor along the y axis,  $s_y$ , is computed as follows:

$$\begin{aligned} s_y &= \text{Vertical Resolution} / \text{Vertical Distance} \\ &= 480 \text{ pixels} / 21.1875 \text{ inches} \\ &= 22.654867 \text{ pixels per inch} \end{aligned}$$

In the equations below, the pixel coordinates of  $C_1$ ,  $p_{clx}$  and  $p_{cly}$ , are converted to work space coordinates,  $i_{clx}$  and  $i_{cly}$ . Note that this must be done for the other three corners as well.

$$\begin{aligned} i_{clx} &= p_{clx} / s_x; \\ i_{cly} &= p_{cly} / s_y; \end{aligned}$$

### 3.2.2 Planar Projective Mapping

Accurately locating an object in the work space is affected by the use of a wide angle lens in the camera of the image processing system. Due to the effect of planar projective mapping (i.e., mapping a 3-dimensional object onto a 2-dimensional plane) [10], an object that is placed near the periphery of the work space appears to be distorted.

Objects that are closer to the optical axis have less distortion than objects that are farther from the optical axis. Thus, a significant portion of the work space is affected by this mapping phenomenon.

To compensate for this effect, an algorithm was developed to adjust the "projected" corner coordinates so that they more accurately reflect the physical location of the object in the work space. A detailed description of this algorithm is given below. Refer to [10] for a more complete treatment of this rather complex problem.

In the following steps, the coordinates of a single "projected" corner, (pcx,pcy), are used to show the effect of planar projective mapping. Refer to Figure 6(a) for Steps 1 thru 4 and to Figure 6(b) for Steps 5 thru 8 below.

Step 1: Compute the distance between (oax,oay,0) and (pcx,pcy,0):

$$dx = pcx - oax;$$

$$dy = pcy - oay;$$

$$d\_oa\_pc = \text{sqrt}(\text{pow}(dx,2.) + \text{pow}(dy,2.));$$

Step 2: Compute the ratio of the legs of the large triangle:

$$\text{ratio} = d\_oa\_pc / oaz;$$

Step 3: Compute the distance between (oax,oay,h) and (cx,cy,h):

$$d\_oa\_c = (oaz - h) * \text{ratio};$$

Step 4: Compute the distance between (cx,cy,0) and (pcx,pcy,0):

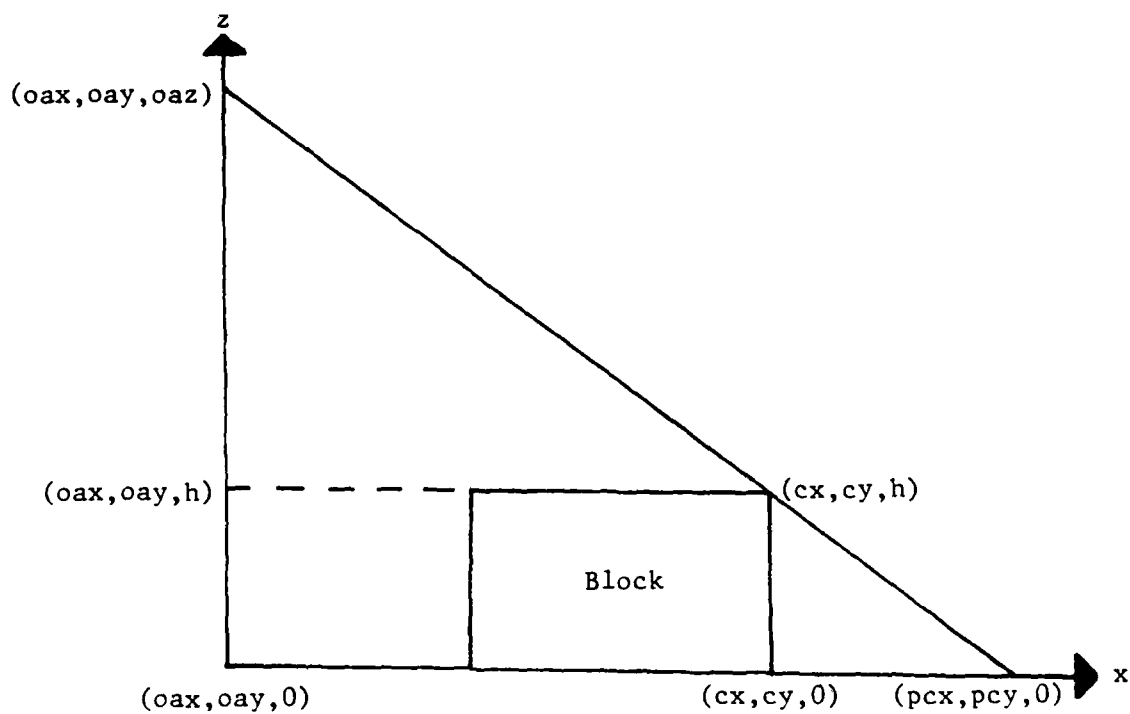
$$d = d\_oa\_pc - d\_oa\_c;$$

Step 5: Compute the distance between (oax,oay,0) and (pcx,oay,0):

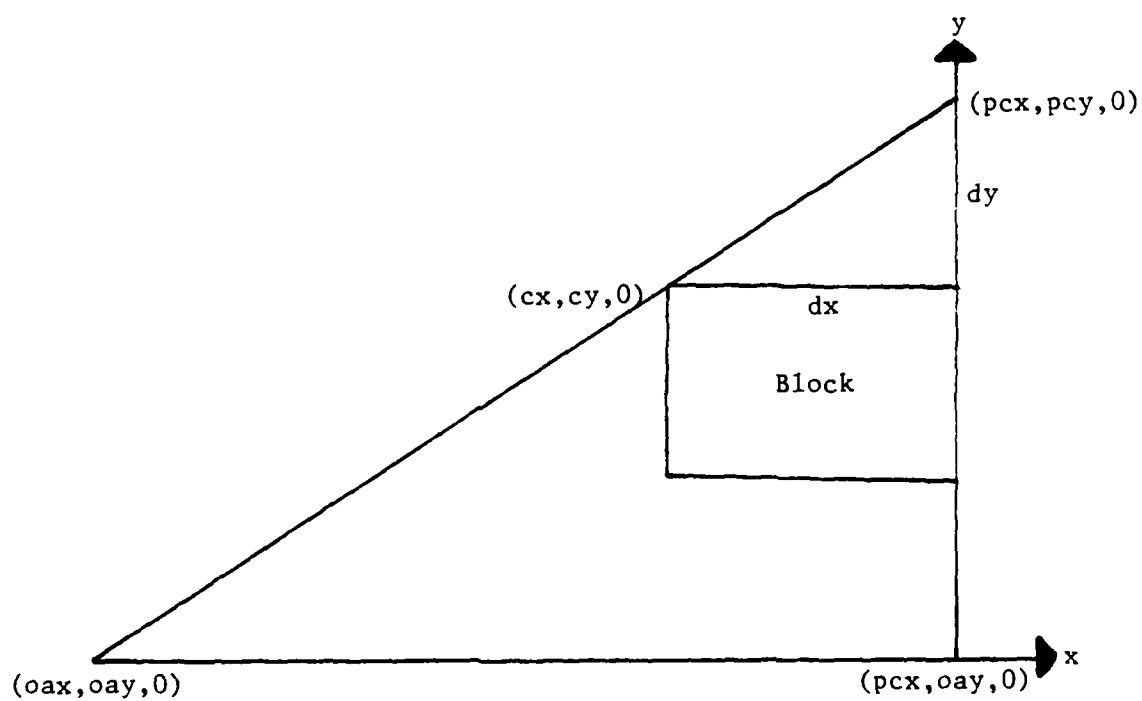
$$d\_pcx = oax - pcx;$$

Step 6: Compute adjustment for x component:

$$dx = (d * d\_pcx) / d\_oa\_pc;$$



(a) View Along Z Axis



(b) View Above XY Plane

Figure 6: Effect of Planar Projective Mapping

Step 7: Compute the distance between (pcx,oay,0) and (pcx,pcy,0):

$$d\_pcy = oay - pcy;$$

Step 8: Compute adjustment for y component:

$$dy = (d * d\_pcy) / d\_oa\_pc;$$

Step 9: Adjust the "project" work space coordinates:

$$cx = pcx + dx;$$

$$cy = pcy + dy;$$

These nine steps must be accomplished for each of the four corners of the block.

### 3.2.3 Computation of Centroid

Typically, the best location for a robot to grasp an object of uniform mass is at the center of mass, or centroid. For the objects used in this project, the computation of the centroid is trivial. The centroid of a block, or cube, is located at the midpoint of either diagonal and inside the block, or cube, a depth equal to one-half the height. The equations below yield the coordinates of the centroid.

$$\text{centroid\_x} = (ic1x + ic3x) / 2.;$$

$$\text{centroid\_y} = (ic1y + ic3y) / 2.;$$

$$\text{centroid\_z} = h / 2.;$$

where, ic1x, ic1y = work space coordinates of corner 1,

ic3x, ic3y = work space coordinates of corner 3,

and h = height of block.

### 3.3 ORIENTATION OF OBJECTS IN THE WORK SPACE

In order to grasp an object in the work space, the robot must be instructed to roll its gripper such that the fingers of the gripper are parallel to the longest edges of the object. Before the roll angle

can be computed, the orientation of the object must be determined. This is because the roll angle computation uses the coordinates of C3 and the coordinates of either C2, or C4, depending on the orientation.

The algorithm developed to determine the orientation (i.e., which corners should be used in the roll angle computation) is described below. Refer to Figure 5(b) for corner locations. Note that the first letter of the corner coordinate names has been changed from "p" to "i" to reflect the conversion from pixels to inches.

Step 1: Compute the length of the edge between C1 and C2:

$$dx = ic1x - ic2x;$$

$$dy = ic1y - ic2y;$$

$$l\_c1\_c2 = \text{sqrt}(\text{pow}(dx,2.) + \text{pow}(dy,2.));$$

Step 2: Compute the length of the edge between C2 and C3:

$$dx = ic2x - ic3x;$$

$$dy = ic2y - ic3y;$$

$$l\_c2\_c3 = \text{sqrt}(\text{pow}(dx,2.) + \text{pow}(dy,2.));$$

Step 3: If  $l\_c1\_c2$  is less than  $l\_c2\_c3$ , then use C2 and C3 to compute the roll angle.

Step 4: If  $l\_c1\_c2$  is greater than  $l\_c2\_c3$ , then use C4 and C3 to compute the roll angle.

Step 5: If  $l\_c1\_c2$  is equal to  $l\_c2\_c3$ , then the object is a cube. Use C2 and C3 to compute the roll angle.

### 3.3.1 Computation of Roll Angle

The standard convention for the sign of an angle measured in a clockwise direction is negative; and, the sign of an angle measured in a counterclockwise direction is positive. However, for the robot's

gripper to roll in the proper direction relative to the orientation of the object in the work space, this sign convention must be reversed. Thus, clockwise angles will have a positive sign and counterclockwise angles will have a negative sign.

The roll angle of the robot's gripper is computed as follows:

Step 1: If  $l_{c1\_c2}$  is less than, or equal to,  $l_{c2\_c3}$ , then

$$dy = ic2y - ic3y;$$

$$dx = ic2x - ic3x;$$

$$\theta = - \operatorname{atan}(dy/dx);$$

Step 2: If  $l_{c1\_c2}$  is greater than  $l_{c2\_c3}$ , then

$$dy = ic4y - ic3y;$$

$$dx = ic4x - ic3x;$$

$$\theta = - \operatorname{atan}(dy/dx);$$

Step 3: Special case: When  $l_{c1\_c2}$  is equal to  $l_{c2\_c3}$ , the object is a cube. If  $\theta$  is greater than 45 degrees, some "roll time" can be saved by having the robot grasp the cube from the other direction. In this case,  $\theta$  is complemented as follows:

$$\theta = 90 + \theta;$$

Step 4: Convert  $\theta$  from radians to degrees as follows:

$$\theta = \theta * (180/\pi);$$

The image processing system is now ready to send the location and orientation information to the DUAL so that one of the robot arms may be selected to grasp and manipulate the block.

## Chapter IV

### GRASPING AND MANIPULATING THE OBJECT

Having determined the location (centroid) and orientation (roll angle) of the block, the image processing system has nearly completed its task as sufficient data is now available to enable the robot to grasp and manipulate the block. First, however, the robot must be placed at a location close enough to the work space so that it can reach the block; then, the coordinate system of the robot must be synchronized with the coordinate system of the work space. That is, the robot must be "told" where it has been placed; otherwise, it does not "know" where it is in respect to the origin of the work space, and, the robot will not be able to grasp the block.

At this point, a program running in the DUAL will accept the location and orientation data from the VAX. Then, using commands from Koutsourelis' robot arm command language, this program will instruct the robot to fetch the block.

#### 4.1 ROBOT INITIALIZATION

To enable the arm to reach as much of the work space as possible, the home position of the robot's gripper was chosen as (0,10.5,0) in the work space coordinate system. Initialization routines in the robot arm command language assume the gripper has been placed at (5,0,0) in the robot coordinate system. Therefore, the origin of the robot's

coordinate system is located at  $(-5, 10.5, 0)$  relative to the origin of the work space  $(0, 0, 0)$ . Thus, whenever the robot is instructed to move to a point relative to the origin of the work space, the x component of the location is increased by 5 inches and the y component is increased by 10.5 inches. Synchronization of the robot's coordinate system with the work space coordinate system is thus achieved. Whenever the robot is instructed to move relative to its own coordinate system, it is not necessary to adjust the coordinates as the arm moves to the  $(x, y, z)$  location relative to its origin of  $(0, 0, 0)$ .

#### 4.2 COMMUNICATION BETWEEN THE VAX AND THE DUAL

Communication of data from the VAX to the DUAL is accomplished as a simple port-to-port transfer of data. Ports for both computers are located on the patch panel in the Robotics Laboratory. Prior to using this vision system, the ports must be physically connected by using an RS-232 serial data cable with DB-25 connectors on each end. Also, the baud rate for the DUAL port must be reset to 9600. This port is used to send data to a printer and the baud rate is usually set to 600. Detailed instructions for connecting the VAX to the DUAL can be found in Appendix A.

The program running in the DUAL accepts location and orientation data from the VAX by issuing an `fscanf(vaxport, ...)`, from C, or a `readln(vaxport, ...)`, from PASCAL. Of course, the "file" assigned to the port must have been opened prior to making the I/O request. After issuing the `fscanf`, or `readln`, the UNIX I/O handler will wait until the request has been satisfied (i.e., when the VAX writes data to the port) before resuming execution of the program. Communication from the DUAL



to the VAX is performed in a similar manner.

#### 4.3 GRASPING THE OBJECT

After receiving the location of the centroid and the roll angle, the program running in the DUAL sends a series of "moveto" commands [5] to the robot arm. These commands cause the arm to grasp and manipulate the object. A typical sequence of arm movements during the process of grasping a block is given below.

- Move 1: Move 3 inches above the (x,y) coordinate of the centroid with a roll angle of theta degrees and open the gripper.
- Move 2: Move down to the z coordinate of the centroid.
- Move 3: Close the gripper.
- Move 4: Lift the block 3 inches above the work space.
- Move 5: Move the block to an arbitrary location.
- Move 6: Lower the block to the surface of the table and open the gripper.
- Move 7: Return to the home position and wait until instructed to fetch another block.

If the block is placed in the work space such that the robot arm cannot reach the centroid of the block, the program controlling the robot sends an appropriate error message to the user's terminal and instructs the robot arm to return to the home position and wait until instructed to fetch another block.

## Chapter V

### SUMMARY AND CONCLUSIONS

The major objective of this project was to interface the image processing system (i.e., camera/monitor/Grinnell/VAX) to the robot arm system (i.e., MICROBOT/Mitsubishi/DUAL). The goal was to demonstrate the ability to use computer vision to locate and manipulate a simple object in the robot's work space.

To achieve this goal, numerous procedures and algorithms were designed and implemented. A procedure was developed to determine the location of the camera's field of view (i.e., the work space) on the robot's work surface. A data transfer algorithm was implemented to upload a gradient image from the Grinnell to the VAX. The corner finding algorithm was used to locate the four corners of a block in the work space. An algorithm was also developed to compensate for the distortion resulting from planar projective mapping. Additionally, an algorithm to determine the orientation of a block in the robot's work space was implemented. This was required in order to determine which corners to use in computing the roll angle at which the robot's gripper must be placed to grasp the block. Finally, a procedure was implemented to enable the transfer of location and orientation data from the VAX to the DUAL.

The time required from when the camera grabs the image to when the robot arm grasps the block is approximately 23 seconds due to the

I/O overhead between the Grinnell and the VAX. Thus, a vision system such as this would not find acceptance in industrial applications. For academic purposes, however, the excessive I/O time provides the student with an opportunity to experience, firsthand, the effect of slow data transfer on a computer vision system. It underscores the need for, and importance of, high-speed communication between a vision processor and its host computer, especially for robots operating in an environment where they must analyze changing scenes in real time. The communication and computational requirements of computer vision place a tremendous demand on an image processing system. There have been estimates made that processor speeds on the order of 1 to 100 billion operations per second will be required to solve some of the current problems in image processing [11].

### 5.1 SUGGESTIONS FOR FUTURE RESEARCH

Now that the goal of interfacing the image processing system to the robot arm system has finally been demonstrated, additional research projects related to computer vision could be undertaken. Some ideas for future research related to the use of computer vision with multiple, coordinated robot arms are given below.

#### 5.1.1 Noise Suppression

In its present form, the corner finding algorithm is extremely fragile in that any noise in the image will cause the algorithm to fail (i.e., the coordinates of the corners will be incorrect). To make this a more robust, fault-tolerant algorithm, additional steps should be added to ignore isolated pixels and to suppress other noise in the image.

### 5.1.2 Multiple Object Recognition

The corner finding algorithm was designed and implemented to locate the four corners of a single object, either a rectangular block or a cube, in the robot's work space. It would be very desirable to extend the capability of this algorithm so that it could identify more than one object at a time. With more than one object in the work space, the user's applications programs would encounter several new and challenging problems (e.g., interpreting a scene to determine the interrelationships among multiple objects so as to prevent colliding with one object while attempting to grasp another).

### 5.1.3 Stereo Imaging

In this project, the robot can "see" only the xy plane of the work space. The height of an object in the work space is input to the vision algorithms from the keyboard. A second camera aimed at the z axis could be used to determine the height of an object, or objects, in the work space. Also, objects that may appear to be a particular shape when viewed from the perspective of the overhead camera may turn out to be an entirely different shape when viewed from the perspective of the camera along the z axis. Hence, objects could be identified much more accurately with stereo images.

### 5.1.4 Chain Coding

Objects used in this project were very simple, geometrically symmetrical objects of uniform mass. Identifying and locating more complex shapes would, indeed, make this a more powerful vision system. Implementation of chain coding in the host computer would allow the vision system to accomodate complex, or irregularly shaped objects.

## REFERENCES

1. Critchlow, A. J., Introduction to Robotics, New York: Macmillan Publishing Company, 1985, pp. 115-379.
2. Coiffet, P., Robot Technology, Vol. 2: Interaction with the Environment, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983, pp. 103-227.
3. Gonzalez, R. C., and Safabakhsh, R., "Computer Vision Techniques for Industrial Inspection and Robot Control," - A Tutorial Overview, pp. 300-24.
4. Majumdar, G., "Implementation of Edge Detection Algorithms on a Special Purpose Image Processor," M.S. thesis, University of South Florida, Tampa, Florida, December, 1985.
5. Koutsourelis, D. I., "Coordinating Robot Arm Motion with UNIX," M.S. project, University of South Florida, Tampa, Florida, December, 1985.
6. Kochan, S. G., and Wood, P. H., Exploring the UNIX System, Hasbrouck Heights, New Jersey: Hayden Book Company, 1984.
7. Levialdi, S., "Edge Extraction Techniques," in Fundamentals in Computer Vision, pp. 117-44, edited by O. D. Faugeras, Cambridge: Cambridge University Press, 1983.
8. Rheinboldt, W., gen. ed., Computer Science and Applied Mathematics, New York: Academic Press, Inc., 1979, Computer Image Processing and Recognition, by E. L. Hall.
9. Kochan, S. G., Programming in C, Hasbrouck Heights, New Jersey: Hayden Book Company, 1983.
10. Graham, J. H., and Kennedy, T. G., "Shape Correction for Digital Images Formed by Wide Angle Lenses," IEEE 1984 International Conference on Robotics, pp. 122-29.
11. Delp, E. J., et. al., "Parallel Processing for Computer Vision," Robot Vision, Azriel Rosenfeld, Ed., Proc. SPIE 336 (1982), pp. 161-67.

## APPENDIXES

## Appendix A

### USER'S GUIDE

This guide was written with the assumption the user is familiar with the location, and general configuration of the various devices that comprise the image processing system and the robot arm system. To have a robot grasp a block in the work space, the following steps should be accomplished.

1. Turn on the robot to be used.
2. Place the robot at the desired location outside the perimeter of the work space.
3. Use the Teach Control Box to place the gripper at (5,0,0) with the gripper closed and just barely touching the work space. The gripper should be parallel to the base of the robot arm.
4. At the patch panel, disconnect the cable from the port labeled ttyd2. This cable is for the printer (/dev/lp) connected to the DUAL.
5. Connect one end of the DUAL/VAX cable to the port labeled ttyd2 and connect the other end to the port labeled VAX (Direct). The latter is actually port ttyl0 on the VAX; and, this port DOES NOT go through the USF Dataswitch.
6. Turn on the video monitor.
7. Turn on the Terak monitor.

8. Turn on the lights around the work surface and adjust the intensity until very little glare is visible on the video monitor.
9. Log on to the VAX (i.e., vax2 or unix) and execute the program findobject by entering the following command:  

```
    /usr/robotics/wingate/programs/findobject nnn
```

where nnn is an intensity threshold value between 0 and 255. Experimental results have shown that a threshold value in the range of 40 - 70 yields the best edges.
10. Log on to the DUAL and set the baud rate of ttyd2 to 9600 by entering the following command:  

```
    stty 9600 > /dev/ttyd2    (or, stty 9600 > /dev/lp)
```
11. Execute the program getobject by entering the following command:  

```
    /a/wingate/getobject
```
12. When prompted to do so, press the "mode" button on the robot's Teach Control Box.
13. Place a block in the work space.
14. When prompted to do so, enter the height of the block placed in the work space.
15. The robot will pick up the block and set it down outside the work space. To repeat this operation, respond 'y' to the prompt on the DUAL terminal. To exit, go to step 17.
16. Place a block in the work space and press 'return' on the DUAL terminal. Steps 14 thru 16 may be repeated as many times as is desired.
17. Respond 'n' to the prompt on the DUAL terminal. This will



terminate program findobject in the VAX and program getobject in the DUAL.

18. Reset the baud rate of ttyd2 to 600 by entering the following command:

stty 600 > /dev/ttyd2 (or, stty 600 > /dev/lp)

19. Log off the DUAL terminal.
20. Disconnect the DUAL/VAX cable and plug the gree end of the printer cable back into the port labeled ttyd2.
21. Log off the VAX terminal.
22. Turn off the Terak monitor.
23. Turn off the video monitor.
24. Turn off the lights around the work surface.
25. Turn off the robot.

## Appendix B

### DETERMINING THE CAMERA'S FIELD OF VIEW

To grasp an object, the robots must be placed relative to the xy axes of the work space. Before the robots can be placed, the location of the work space must be determined. Determining the location of the work space is analogous to determining the location of the camera's field of view. It is important to note that whenever the camera's position is changed the location of the work space is changed as well!

To determine, or redetermine, the location of the camera's field of view, the following procedure should be used. Refer to Figure 7(a) for block placement in the work space and Figure 7(b) for corresponding positions on the Terak monitor (i.e., a block placed near position 1 in the work space will appear near position 1 on the Terak monitor).

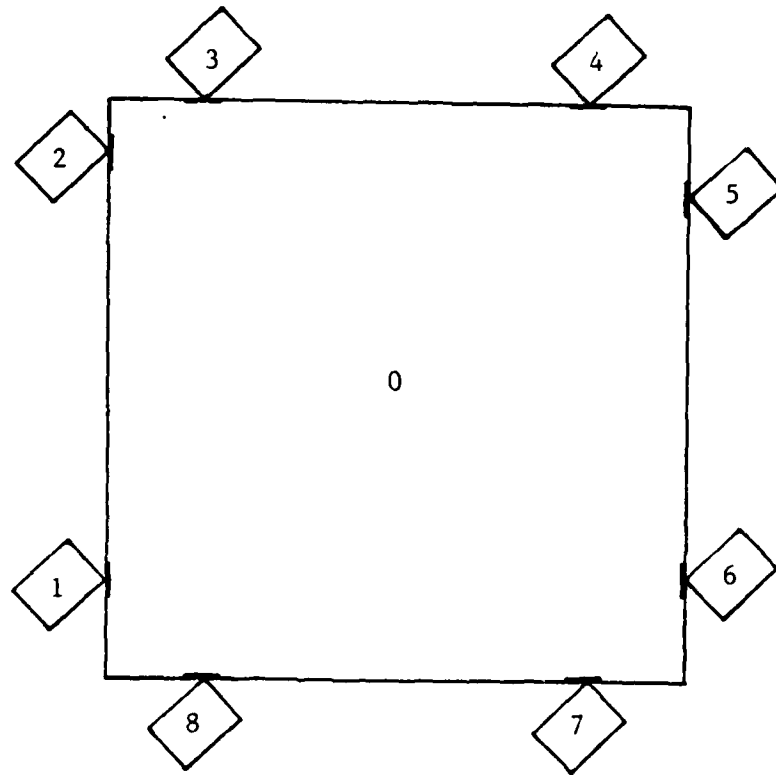
1. If applicable, remove the tape, or other material, used to mark the boundary of the old work space.
2. Log on to the VAX (i.e., vax2 or unix).
3. Change directories by entering the following command:

```
cd usr/robotics/wingate/programs
```

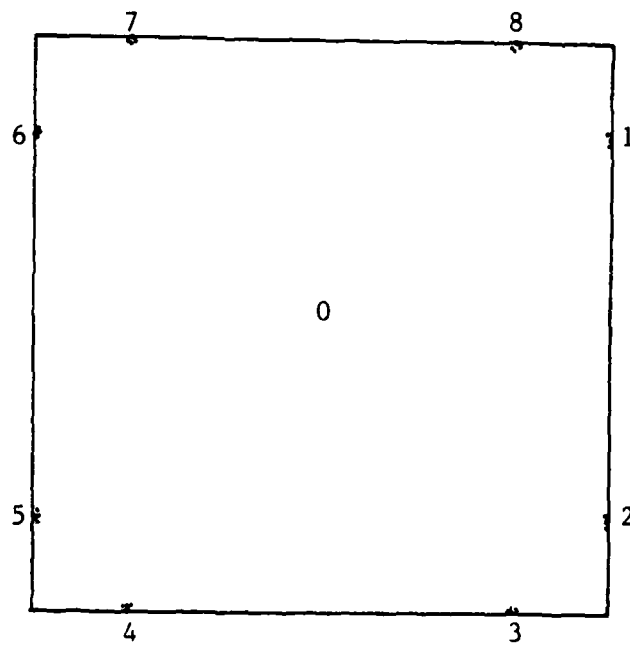
4. Perform steps 6 thru 8 in the User's Guide at Appendix A.
5. Place a block near location 0 in the work space and run the program view by entering the command:

```
view nnn
```

where nnn is an intensity threshold value between 0 and 255.



(a) Work Space



(b) Terak Monitor

Figure 7. Block Placement to Locate Camera's Field of View

Experimental results have shown that a threshold value in the range of 40 - 70 yields the best edges. Observe the Terak monitor and, if necessary, adjust the threshold value or the position/intensity of the lights, or both, and rerun the program. Repeat this step until you are satisfied with the quality of the edges, then go to the next step.

6. Place a corner of the block near position 1 in the work space, rerun the program, and observe the Terak monitor. The goal is to repeatedly move the block, run the program, and observe the monitor until the corner of the block is just barely visible near position 1 of the Terak monitor. Barely visible means just that - if more than 2 - 3 pixels are visible, the block should be moved again. This step requires a lot of patience!
7. When you are satisfied that you have located the boundary of the camera's field of view at this position, place a piece of tape, or some other marker, at this location.
8. Repeat steps 6 and 7 for positions 2 thru 8 of the work space observing the corresponding positions on the Terak monitor each time.
9. After all eight positions have been marked, place a strip of tape on the surface of the table such that it just barely touches the outside of the markers placed at positions 1 and 2. Repeat this step, using the appropriate positions, to form the other three sides of the work space. Trim any excess tape to form neat corners. The work space is inside this rectangle.
10. When this paper was written, the length of the x axis was 22

inches and the length of the y axis was 21.1875 inches. If the new length of the x axis is less than 21.75 inches or greater than 22.25 inches, or, if the new length of the y axis is less than 20.9375 inches or greater than 21.4375 inches, then the variables for the lengths of the axes in the findobject-series of programs should be updated. Variables hd, and vd, for the horizontal distance and vertical distance, respectively, should be changed to reflect the new values. The programs should then be recompiled and relinked.

END

10-86

DTIC