

Volume I

THE INTEGRATION OF VISUAL AND TACTILE SENSING FOR
THE DEFINITION OF REGIONS WITHIN A ROBOT WORKCELL

by

Edward Christopher De Meter

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Industrial Engineering and Operations Research

APPROVED:

M. P. Deisenroth, Chairman

M. S. Jones

R. G. Leonard

June 11, 1986

Blacksburg, Virginia

**THE INTEGRATION OF VISUAL AND TACTILE SENSING FOR
THE DEFINITION OF REGIONS WITHIN A ROBOT WORKCELL**

by

Edward Christopher De Meter

M. P. Deisenroth, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

Vision systems are widely used in robot workcells for sensory feedback. The resolution of a vision system is usually good enough to locate an object so that it can be grasped, but not good enough to accurately locate an insertion hole. Tactile probes are used to accurately locate objects. However, they require a data base containing the approximate location of an object in order to be used effectively.

This thesis presents the development of a robot workcell which utilizes a vision system and tactile probe to identify, locate, and orientate two types of circuit board fixtures. The vision system approximately locates the corner points of each fixture in the robot workcell. The tactile system uses the data base created by the vision system to conduct a tactile search for each fixture and to accurately define the coordinates of each corner point. After a fixture is accurately located, a region (sub-coordinate system) is defined about the fixture. The location of each insertion hole within a fixture is defined relative to the region and the robot subsequently inserts the tactile probe into each hole.

The vision system developed can define any two dimensional object and can locate the corner points of any straight edged object, whose adjacent sides have an included angle greater than 90 degrees. The tactile system is self calibrating and has a repeatability of 0.009 inches.

A probe insertion error analysis was conducted on the system. The average probe insertion error for the system was determined to be 0.0337 inches. In addition, it was determined that probe insertion error increases with the distance between a hole and the origin of its defining region, and that the major source of probe insertion error is the robot language's (AML/E Verion 4.0) inability to accurately define points within a region.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Michael P. Deisenroth for his endless patience and vigilance. Without his encouragement, many of the concepts brought forth in this thesis would not have been born. The past year and a half under his leadership have been both an honor and a pleasure, and I look forward to pursuing my Doctoral degree under him in the coming academic year.

I would also like express thanks to my friends, Larry and Radha, whose comradery through the good times and the bad times have made graduate school an experience I will always cherish.

TABLE OF CONTENTS

1.0 INTRODUCTION	1
1.1 Problem Statement	1
1.2 Objective	3
1.3 Research Phases	4
1.3.1 Main Program Development	6
1.3.2 Vision Subroutines Development	8
1.3.3 Communication Subroutines Development	8
1.3.4 Tactile Sensor Design	9
1.3.5 Tactile Sensor Subroutines Development	10
1.3.6 Probe Insertion Subroutines Development	11
1.3.7 User Information Subroutines Development	12
1.3.8 System Analysis	12
2.0 LITERATURE REVIEW	13
2.1 Systems Designed for Identification and Approximate Location	13
2.2 Systems Designed for Accurate Location	17
3.0 METHODOLOGY	20
3.1 System Overview	20
3.2 Main System Software	21
3.3 Vision System	24
3.3.1 Vision System Hardware	27
3.3.2 Vision System Software	29

3.3.2.1	Vision Hardware Interface	29
3.3.2.2	Object Boundary Identification and Tracking	30
3.3.2.3	Object Boundary Definition	30
3.3.2.4	Circuit Board Fixture Identification and Positioning .	31
3.4	Tactile System	32
3.4.1	Tactile System Hardware	32
3.4.2	Tactile System Software	36
3.5	Probe Insertion Software	39
3.6	User Information Software	40
3.6.1	GRAPH1	41
3.6.2	GRAPH2	42
3.6.3	GRAPH3	42
3.7	Communication System	43
3.7.1	Communication System Hardware	43
3.7.2	Communication System Software	44
3.7.2.1	Communication Protocol	44
3.7.2.2	EXEC	48
3.7.2.3	CNTRL	49
3.7.2.4	WRITEP	52
3.7.2.5	WRITER	55
3.7.2.6	READP	57
3.7.2.7	READR	58
3.7.2.8	RSTATUS	60
3.7.2.9	RDIDO	63
3.7.2.10	PUTWAIT	66

3.7.2.11	Supporting Subroutines	69
4.0	VISION SOFTWARE LIBRARY	75
4.1	Object Boundary Identification and Tracking	75
4.1.1	Gray Scale to Binary Conversion	75
4.1.2	Boundary Continuation, Birth, and Death	76
4.1.2.1	DFBND	86
4.1.2.2	BNDESCN	91
4.1.2.3	BNDBTH	92
4.1.2.4	BNDDTH	93
4.1.2.5	Supporting Subroutines	96
4.2	Object Boundary Definition	99
4.2.1	Boundary Mapping	100
4.2.1.1	LBDINT and RBDINT	108
4.2.1.2	RTECHG	109
4.2.1.3	RETE	109
4.2.1.4	CSWTCH	110
4.2.1.5	PSWTCH	110
4.2.1.6	RTEPLN	111
4.2.2	Corner Point Definition	111
4.2.2.1	PNTDEF	115
4.2.2.2	CLNUP1 and CLNUP2	117
4.2.2.3	RTESTT	117
4.3	Circuit Board Identification and Orientation	118
5.0	Tactile System Software Library	123

5.1	Tactile Probe Calibration	123
5.1.1	ERCALC	125
5.1.2	TOUCH	127
5.1.3	ANGCLC and ANGSUB	129
5.1.4	TTOUCH	129
5.2	Tactile Search and Region Definition	131
5.2.1	TACTIL	135
5.2.2	RECOR	138
5.2.3	VECDEF	138
6.0	Probe Insertion Software	141
6.1	AINsert and BINSert (IBM AT)	144
6.2	AINsert and BINSert (IBM 7545 Controller)	147
7.0	System Analysis	149
7.1	Execution Time	149
7.2	Tactile System Repeatability	151
7.3	Probe Insertion Error	152
8.0	Conclusions and Recommendations	157
	LIST OF REFERENCES	159
	Vita	161
	Appendix A. Main System Software	162
	Appendix B. Vision System Software	167

Appendix C. Tactile System Software	277
Appendix D. Probe Insertion Software	305
Appendix E. User Information Software	312
Appendix F. Communication System Software	320
Appendix G. Vision Hardware Interface Software	367
Appendix H. Tactile System Hardware	376
Appendix I. SYSTEM DATA	380

LIST OF ILLUSTRATIONS

Figure 1. Circuit Board Fixtures in Robot Workcell	5
Figure 2. Software Structure and Distribution	7
Figure 3. Diagram of Robot Workcell	22
Figure 4. Main System Flowchart	25
Figure 5. Main System Flowchart (continued)	26
Figure 6. Vision System Hardware	28
Figure 7. Tactile Probe	35
Figure 8. EXEC Transmission Sequence	50
Figure 9. CNTRL Transmission Sequence	53
Figure 10. WRITEP Transmission Sequence	56
Figure 11. READP Transmission Sequence	59
Figure 12. RSTATUS Transmission Sequence	64
Figure 13. RDIDO Transmission Sequence	67
Figure 14. RDIDO Transmission Sequence (cont'd)	68
Figure 15. PUTWAIT Transmission Sequence	70
Figure 16. Descriptive Array Elements Used For Boundary Definition	79
Figure 17. Descriptive Array Elements Used For Boundary Definition (cont'd)	80
Figure 18. Descriptive Array Elements Used For Boundary Definition (cont'd)	81
Figure 19. Object Types	82
Figure 20. Parent Object	83
Figure 21. Boundary Continuation Equations	87
Figure 22. Boundary Birth	88
Figure 23. Boundary Termination	89

Figure 24. Area and Moment Equations 90

Figure 25. Descriptive Array Elements Used for Boundary Tracking 101

Figure 26. Descriptive Array Elements Used for Boundary Tracking
(cont'd) 102

Figure 27. Clockwise and Counter Clockwise Scanning 104

Figure 28. Boundary Routing Example 105

Figure 29. Boundary Vector Analysis 116

Figure 30. Descriptive Arrays Used for Circuit Board Positioning 119

Figure 31. Corner Point Ordering 122

Figure 32. Error Compensation Equations 126

Figure 33. Descriptive Arrays Used for Tactile Search and Region De-
finition 132

Figure 34. Fixture Defined Region 136

Figure 35. Region Modification 137

Figure 36. Vision and Robot Reference Systems 139

Figure 37. Region Sub-Coordinate System 142

Figure 38. Probe Insertion Pattern for an "A" Type Fixture 145

Figure 39. Probe Insertion Pattern for an "B" Type Fixture 146

Figure 40. Insertion Point Error 156

1.0 INTRODUCTION

In the past fifteen years, a substantial amount of research has been conducted toward the integration of sensory feedback (predominantly visual and tactile sensing) with robot control. The driving force behind this research has been the fact that a robot with sensory feedback has far more flexibility than one without.

As an example, consider a simple workcell in which a robot is assigned the task of picking up a part from a conveyor. A robot with sensory feedback can be programmed to identify, locate and pickup a part irregardless of its position and orientation. On the other hand, a robot without sensory feedback can only be programmed to move to a point along the conveyor and to grasp at an object that is hopefully the correct part, in the correct position and orientation.

1.1 PROBLEM STATEMENT

The majority of research in sensory feedback has been toward the application of computer vision in the identification and location of objects and regions within a robot workcell. Efforts have ranged from simply identifying two-dimensional objects from a contrasting background to identifying and locating three-dimensional objects from a bin of similar objects.

In any of these applications, the two most important characteristics of the vision system are its field width and resolution. The field width and resolution of a vision system are a function of its field of view and the dimensionality of the pixel array that represents that field of view. The relationship is such that for a given pixel array size, the greater the field of view, the greater the field width, and the poorer the resolution. Likewise, the smaller the field of view, the smaller the field width, but the greater the resolution. Thus there exists an inverse relationship between the field width and resolution of a vision system.

As a result of this inverse relationship, vision systems have predominantly been used to identify and coarsely locate objects within a large field of view. The level of accuracy obtained with these systems has typically been good enough to grasp an object. Vision systems have also been used in applications where the accurate location of a boundary such as a weld seam has been the sole criteria. However, for most situations in which high locational accuracy has been needed, tactile probes have been used.

Tactile probes are robot end effectors which provide an electrical signal when contact is made with an object. Depending upon manufacturer, their accuracy ranges from .001 inches to .0001 inches. In normal use, a robot controller will obtain an approximate location of an object's boundary from its data base and then move the tactile probe to that location. It will then use the tactile probe in coordination with a boundary detection subroutine to accurately locate the object's boundary.

Though tactile probes are very accurate, they are never used without a preconceived location of an object's boundary. However, in many applications it is desirable to have a robot/sensory feedback system which has the characteristic accuracy of a tactile probe, yet the flexibility to locate an object anywhere within a robot workcell and not just at a preconceived location.

Such a system can be developed through the integration of visual and tactile sensing with a robot. This system would use vision to identify and coarsely locate an object within a robot workcell. It would then utilize a tactile probe in coordination with the data base created by the vision system to accurately locate the object.

1.2 OBJECTIVE

The objective of this research was to integrate a separate processing system, a visual sensor, and a tactile sensor with an IBM 7545 robot in order to identify, and accurately locate two types of circuit board fixtures (refer to Figure 1) within the workcell and to perform a probe insertion routine on each fixture identified.

To accomplish this objective, the research involved the creation of a system which would do the following:

1. Dynamically calibrate the tactile probe in order to develop a set of error compensation parameters,

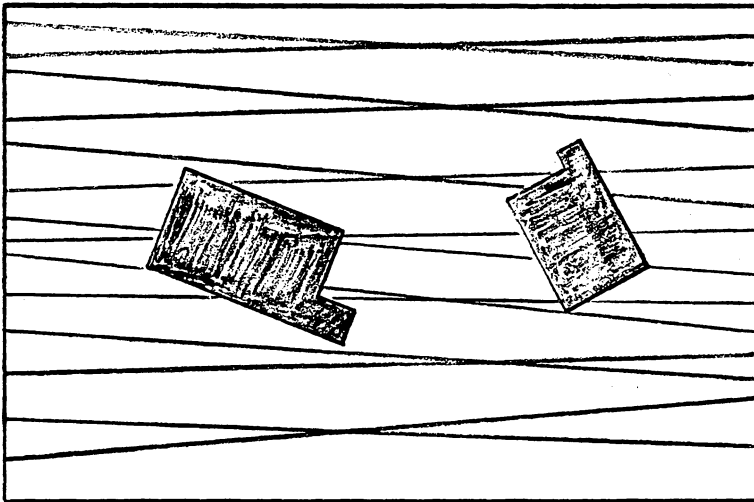
2. Utilize the vision system to create a data base on each object within the system's field of view, irregardless of its position and orientation,
3. Use the data base to distinguish (by shape and size) two distinct circuit board fixtures from each other and/or from any other extraneous objects within the field of view of the vision system,
4. Use the data base to locate the corners of either fixture within the robot workcell,
5. Use the located corner points as the basis for a tactile probe search for the coordinates of the fixture's corner points,
6. Use the defined corner points of the fixture as the reference points for a region (sub-coordinate system) within the robot workcell, and
7. Perform a fixture dependent, tactile probe insertion routine within the defined region.

1.3 RESEARCH PHASES

The research involved in this thesis involved both hardware and software development. Hardware development entailed the creation of a workcell platform and a tactile probe. Software development entailed the creation of FORTRAN and assembly language routines for the main processing system (IBM AT) and AML/E routines for the IBM 7545 robot controller.

The research was broken down into the following eight phases:

Workcell Platform



IBM 7545 Robot

Figure 1. Circuit Board Fixtures in Robot Workcell

1. Main Program Development,
2. Vision Subroutines Development,
3. Communication Subroutines Development,
4. Tactile Sensor Design,
5. Tactile Sensor Subroutines Development,
6. Probe Insertion Subroutines Development,
7. User Information Subroutines Development, and
8. System Analysis.

1.3.1 Main Program Development

The main system program, shown in Figure 2, resides in the IBM AT and coordinates the operation of the system by executing the following functions:

1. Calls upon tactile sensor subroutines to calibrate the tactile probe,
2. Calls upon vision subroutines to visually identify and locate circuit board fixtures within the robot workcell,
3. Calls upon tactile sensor subroutines to accurately define the corner point locations of the fixtures, and
4. Calls upon probe insertion subroutines to conduct probe insertion routines on the fixtures.

The main system program was written in FORTRAN.

IBM AT
Main Program (FORTRAN)
Vision Subroutine Library(FORTRAN)
Tactile Sensor Subroutine Library (FORTRAN)
Probe Insertion Subroutine Library (FORTRAN)
User Information Subroutine Library (FORTRAN)
Communication Subroutine Library (Assembly, Intel 8088)
Vision Subroutine Library (Assembly, Intel 8088)
Graphics Subroutine Library (Assembly, Intel 8088)

IBM 7545 Controller
Tactile Sensor Subroutine (AML/E)
"A" Type Probe Insertion Subroutine (AML/E)
"B" Type Probe Insertion Subroutine (AML/E)

Figure 2. Software Structure and Distribution

1.3.2 Vision Subroutines Development

Vision subroutines are accessed by the main system program and were created to do the following:

1. Initiate the vision system,
2. Grab a picture frame and perform a gray-scale to binary conversion,
3. Use Boundary Continuation techniques to define objects within the picture frame,
4. Use Boundary Tracking techniques to determine the location of the corner points for each object, and
5. Identify and determine the orientation of any fixture within the field of view.

The vision subroutines were written in FORTRAN and they call upon a library of available assembly language routines to control the vision hardware.

1.3.3 Communication Subroutines Development

Communication subroutines were created to facilitate communication between the IBM AT and the IBM 7545 controller via a RS-232 interface. The subroutines do the following:

1. Receive either data or AML/E commands from a calling routine,

2. Convert the data or commands into the required control code specified for Host Communications for the AML/E language,
3. Imbed the data or command code into a transmittable information package as specified for Host Communications for the AML/E language,
4. Send the information package via a RS-232 interface to the robot controller using the handshaking procedures and protocol specified for Host Communications for the AML/E language, and
5. Receive information packages from the robot controller and decode the packages so that the data can be passed to the calling routine.

Since FORTRAN does not have the capability of interacting with the serial interface in the needed fashion, the communication subroutines were written in 8088 assembly language.

1.3.4 Tactile Sensor Design

Commercially available tactile sensor probes have accuracies ranging from .001 inches to .0001 inches and cost from \$1000 upwards. Due to its high commercial cost and the exploratory nature of the research, a tactile probe and gripper interface were designed and built in house.

The IBM 7545 robot has a positional repeatability of 0.002 inches [6]. Thus it was desirable to develop a tactile probe which when used with the

IBM 7545 robot would provide a locational repeatability close to 0.002 inches.

The tactile system developed consisted of a simple metallic probe connected directly to the robot end effector interface, and with an electrical connection to a robot input port. The workcell platform is covered with a grid of conductive wires which are connected to ground along with the ground side of the robot input port. Thus any time a metallic fixture is placed on the platform, it obtains a ground potential.

The tactile probe is connected electrically to a robot input port. When the probe makes contact with a fixture, the ground potential at the fixture is sensed, causing the input port to be pulled low. The robot controller is able to determine the edge of a fixture by periodically moving the robot arm along a vector and checking the status of the input port.

1.3.5 Tactile Sensor Subroutines Development

Tactile sensor subroutines reside in both the IBM AT and the IBM 7545 controller and work in coordination with one another to accurately locate the corner points of the circuit board fixtures.

The tactile sensor subroutines that reside in the IBM AT were written in FORTRAN to do the following:

1. Repeatedly call upon the tactile routine in the 7545 controller to obtain four calibration points for the tactile probe,

2. Compute the error compensation parameters for the tactile probe,
3. Accept approximate fixture corner point locations from the main program,
4. Repeatedly call upon the tactile routine in the 7545 controller to determine the location of two points on each of two adjacent sides of the fixture, and
5. Use the located points to compute the boundary points of the region defined by the fixture.

The tactile sensor subroutine residing in the IBM 7545 controller was written in AML/E to accurately locate a point on the side of an object by commanding the robot to move the tactile probe along a vector until contact is made with an object at the tip of the probe.

1.3.6 Probe Insertion Subroutines Development

Probe Insertion subroutines reside in both the IBM AT and the IBM 7545 controller and work in coordination with one another to insert the tactile probe into the patterned holes in each fixture.

The subroutines residing in the IBM AT were written in FORTRAN to transmit the regional boundary points to the subroutines in the 7545 controller, and to run an insertion operation on each fixture in the workcell.

The subroutines residing in the 7545 controller were written in AML/E to command the robot to move to each drill hole in a fixture (defined by a region) and to insert and withdraw the tactile probe.

1.3.7 User Information Subroutines Development

User information subroutines were written in FORTRAN so that the main program could interactively communicate with the user of the program. Communication involved informing the user of the number of objects and fixtures identified in the workcell as well as the current operation in progress. The subroutines called upon a library of assembly language graphics routines.

1.3.8 System Analysis

Experimentation was conducted on the robot/sensory system to determine its execution time, the repeatability of the tactile system, and the probe insertion error incurred when defining the side of an object with two or three contact points. An analysis was run on the data collected, the described parameters were computed, and areas of improvement were identified.

2.0 LITERATURE REVIEW

Over the past three years, there has been a substantial amount of research toward the integration of robot control systems with visual and/or tactile sensing systems. This section will review some of the major developments in sensory feedback systems that were or are being designed either to:

1. Identify and approximately locate objects within a robot workcell, or
2. Accurately locate an object or region within a robot workcell.

2.1 SYSTEMS DESIGNED FOR IDENTIFICATION AND APPROXIMATE LOCATION

Kohno, Horino, and Isobe [8] integrated a vision system with a two arm robot system. The vision system consisted of a standard TV camera (one producing a RS-170 signal) connected to a HIDIC-80 process computer. The robot system consisted of two 6 degree-of-freedom (D.O.F) articulated robots.

Each robot was controlled by a hierarchical system of two HMCS 6800 microprocessors. The main processor handled all interfaces with peripheral units, including the vision system. It also handled all mathematical computations including the transformation manipulator displacement to robot joint angles. The slave processor provided control over the robot's 6 DC drive systems.

A special robot programming language was developed for the system. The language included robot motion commands, program flow of control commands, math functions, and peripheral input/output commands.

The vision system, upon command, grabbed an image of the view area. The picture frame was then converted from a gray-scale image to a binary image using a simple thresholding technique. The thresholding was accomplished through a hardware image processor.

The objects, which were to be picked up by the robot, had circular targets placed on them. Thus the vision system searched for the binary images of these circular targets in the binary pixel array. Once a target was identified and located within the array, the coordinates were sent to the main robot processor for conversion to robot joint angular displacements. Care had to be taken not to have two targets within the vision area in order to avoid improper recognition of the targets. This system successfully located toy train components within the robot workcell and assembled the components into a train.

Nehr and Martini [12] interfaced a vision system with a Siemens 4 D.O.F. robot which was controlled by a Siemens PR 310 minicomputer. The vision system consisted of a 4096 (64 X 64) photodiode array sensor interfaced with a Siemens PR 310 minicomputer.

The vision system grabbed a picture frame, converted it to a binary image, and then extracted a set of M features from the image. These features

were then compared to another set of reference features stored in the vision system's memory. When a match occurred, the minicomputer proceeded to determine the cartesian coordinates of the object's area centroid. It then extracted a set of orientation features from the image. Again, these features were compared to a series of reference orientation features stored in the vision system's memory.

When a match was made, the Cartesian coordinates of the object's grasp points were computed, and then converted into the robot's joint angles. These angles were then transmitted to the robot controller.

This system was capable of identifying and locating single two-dimensional objects placed at any orientation on a conveyor. When an object was positively identified and located, the robot was commanded to pick up the object from the conveyor and to set it down at a predefined location.

Amat, Casals, and Llarío [2] integrated a robot with a vision system that performed essentially all of its image processing and feature analysis through hardware.

The vision system grabbed a picture frame and then subtracted a normal picture frame (stored as reference in memory) from it to extract an object's features from the background. The object's features were then subjected to a mathematical enhancement procedure which reconstructed any boundary lost during feature extraction.

An image rectangle was created from the boundary of the object. From this rectangle, the centroidal location and orientation of the object was determined. The grasp coordinates of the object were then transmitted to the robot controller, which performed the necessary coordinate transformations.

This system was capable of identifying and locating two-dimensional objects placed at any orientation on a conveyor. Since nearly all of the vision processing was done in hardware, the system was capable of completely processing a picture frame within a 400 millisecond time span.

Kelly, Birk, Martins, and Tella [7] implemented a robot/vision system capable of locating and picking up a cylindrical object from a bin of similar objects.

The robot was a 6 D.O.F. articulated robot with a parallel jaw gripper. The vision system consisted of a computer and a standard TV camera mounted over the workcell. The vision software grabbed an image of the bin's contents and converted the grey-scale image to a binary image using a simple thresholding technique. Clusters within the binary image underwent a shrinking operation in which the total number of pixels in each cluster was reduced. This process essentially eliminated all but the largest clusters.

The centroid, major and minor axes as well as the moments of inertia were computed for the largest cluster. Depending upon the ratio of the major

axis to the minor axis, the vision system could determine whether the cylinder was on end, inclined, or flat. The gripper coordinates were selected based on this criteria.

The authors further improved the accuracy of the vision system by using the actual line of sight vector for the computation of position of each pixel in the array. This involved using a second order polynomial equation for the calculation of distances between pixels.

Didocha, Lyons, and Thompson [4] are currently integrating a tactile sensory pad with a vision system to recognize overlapping parts. The tactile array in use is a Lord's model LTS 100 and the vision system is an Automatix AV-4.

The tactile array is to be used to identify the parts that are on the bottom and to extract features of the top part. The vision system is used to extract features of the entire blob. By combining the features extracted from both systems the authors intend to identify and located the various overlapping parts.

2.2 SYSTEMS DESIGNED FOR ACCURATE LOCATION

Hewit and Jones [5] created a vision/robot system that welded spacers on to heat exchanger platens. The robot used was an ASEA IRB6, while the vision system consisted of a CCD camera linked to a LSI-11/03 minicomputer. The LSI-11 was linked to the robot controller via a RS-232 interface.

The main system program, which was written in FORTRAN, was executed by the LSI-11. Using CAD data, the LSI-11 guided the robot to the space joint to be welded. The camera, which was mounted along side the welder, then grabbed a close up picture frame (256 X 256 pixel array) of the area to be welded.

Sharp increases in the gray level within the pixel array signified the region to be welded. The LSI-11 computed the Cartesian coordinates of the initial point of the seam and then converted them into the proper robot joint angles. These angles were subsequently transmitted to the robot controller, which then initiated a welding subroutine.

More recently, Lam, Moore, and Snell [9] integrated a tactile probe with a robot to test printed circuit boards. The robot system used was a Intelledex 605, 6 D.O.F. articulated robot. ROBOT BASIC was used as the robot command language. The printed circuit board testing system was a Teradyne L2000. Besides performing logic tests, the Teradyne system also controlled the entire test/robot system.

The tactile probe served two functions. First, it performed as a probe which was used to accurately locate the tooling pins on which the circuit board was mounted. It then served as a circuit testing probe which made contact with the circuit board at various points.

The tactile probe maintained four optical sensors. Two sensors were used for force feedback at the end of the probe. One was used for determining

when the probe was centered on a contact point, and the last one was used for detecting the edges of the tooling pins.

The Teradyne tester initiated the operation sequence by sending the robot/probe to each of the three tooling pin locations. The probe would be used to detect the edges of the tooling pin. The tester would subsequently determine the average of the X and Y coordinates of the edges to determine the center of the pin.

After determining the exact location of each of the three tooling pins, the tester defined the location of the printed circuit board within the robot workspace. It subsequently used CAD data to send the test probe to various locations on the printed circuit board to conduct tests.

Both the program residing in the robot controller and the main program residing in the Teradyne tester utilized communication subroutines that were written in 8088 assembly language.

3.0 METHODOLOGY

This chapter describes the methodology used in this research. The following items are discussed:

1. System Overview,
2. Main System Software,
3. Vision System,
4. Tactile System,
5. Probe Insertion Software,
6. Graphics Software, and
7. Communication System.

3.1 SYSTEM OVERVIEW

The robot workcell, shown in Figure 3, consists of the following hardware:

1. IBM 7545 Robot/Controller,
2. IBM Personal Computer AT,
3. RCA RS-170 Camera,
4. PCVISION Frame Grabber,
5. Workcell Platform (Built In House), and
6. Tactile Probe (Built In House).

The robot/sensor system does the following:

1. Creates a feature data base on any straight edged object within the field of view of the vision system,
2. Identifies, locates and orientates any one of two types of circuit board fixtures,
3. Accurately locates every 0.055" hole on each circuit board fixture within the field of view, and
4. Inserts the tip of the tactile probe into every hole on each circuit board fixture.

The system software works interactively with the user so that the system can be directed to either grab and process a new image of the workcell platform, perform an insertion operation on the fixtures identified from the previous sensory search, or return the robot to the home position and end program execution.

3.2 MAIN SYSTEM SOFTWARE

The main program was written in FORTRAN and can be referred to in Appendix A. The flowchart of the main system program, Figure 4, shows the sequence of operations performed by the system. The initial operation performed by the main program is to calibrate the tactile probe. This is accomplished by calling the tactile subroutine, ERCALC, which develops the error compensation parameters for the tactile probe. The information subroutine, GRAPH1, is called by ERCALC to instruct user of the probe calibration operation in process.

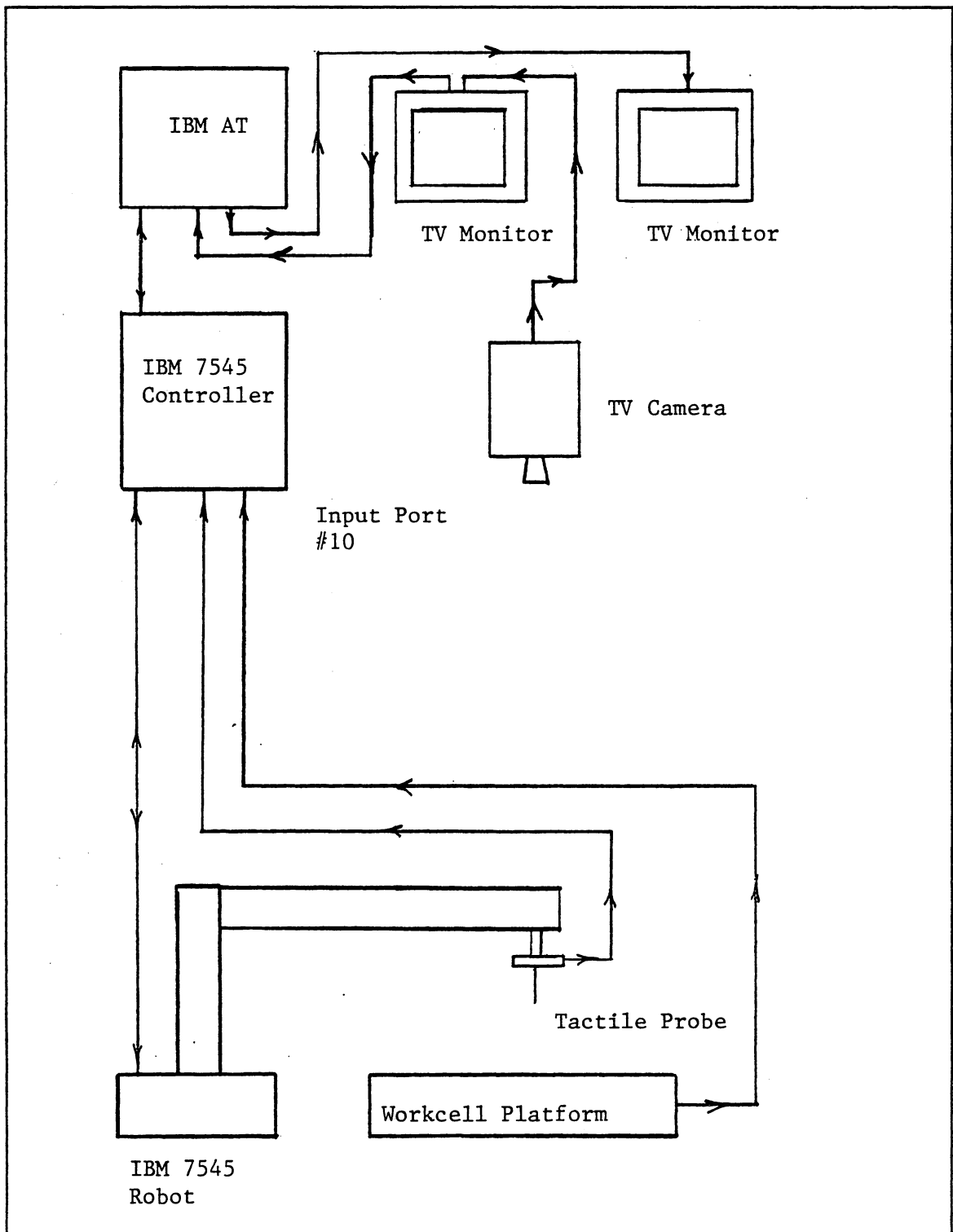


Figure 3. Diagram of Robot Workcell

The main program then initializes a group of variables, which are to be used by the vision subroutines, and calls GRAPH1 to instruct the user of the vision system's pending operation. Subsequently, the main program calls upon the vision subroutine, FRMGRB, to initialize the vision system hardware and to grab an image of the workcell platform. The image is then converted from a gray-scale image to a binary image through the call of CNVRT.

The vision subroutine, DFBND, is called next in order to define all object boundaries in the binary image. FNSTAT is then called to compute the area and centroid for each object in the image. The location of each object corner point is then determined through the call of PNTDEF. The identification, location, and orientation of every "A" type and "B" type fixture in the image is determined through the call of the subroutines ASEROR and BSEROR respectively.

The information subroutine, GRAPH2, is called to instruct the user on the number of objects observed by the vision system, and the number of "A" type and "B" type fixtures identified and located. GRAPH2 instructs the user to strike any key on the keyboard to initiate the tactile search for any fixtures in the workcell.

After program execution is returned from GRAPH2, the main program calls the tactile subroutine, TACTIL, to conduct a tactile search for any "A" type and/or "B" fixtures. If no fixtures have been identified by the vision system, then TACTIL will not initiate a search. Instead it merely

returns execution to the main program. GRAPH1 is called by TACTIL to instruct the user of the tactile search operation in progress.

Once the boundaries of all the fixtures in the workcell platform have been defined, the insertion subroutines, AINSERT and BINSERT are called to perform probe insertion routines on all "A" type and "B" type fixtures respectively. If no fixtures have been identified by the vision system, then no insertion routines are performed. GRAPH1 is called by both AINSERT and BINSERT to instruct the user of the probe insertion operation in progress.

The graphics subroutine, GRAPH3, is called next to instruct the user to either enter a "1" on the keyboard to reinitiate the insertion operation, a "2" to process another image of the workcell platform, or a "3" to return the robot to the home position and end program execution. If the user decides to end program execution, then the communication subroutine, EXEC, is called to return the robot to the home position and program execution is ended. Otherwise execution is branched to the appropriate section of the main program.

3.3. VISION SYSTEM

The vision system developed in this research is capable of creating a feature data base on any two-dimensional, straight edged object, whose

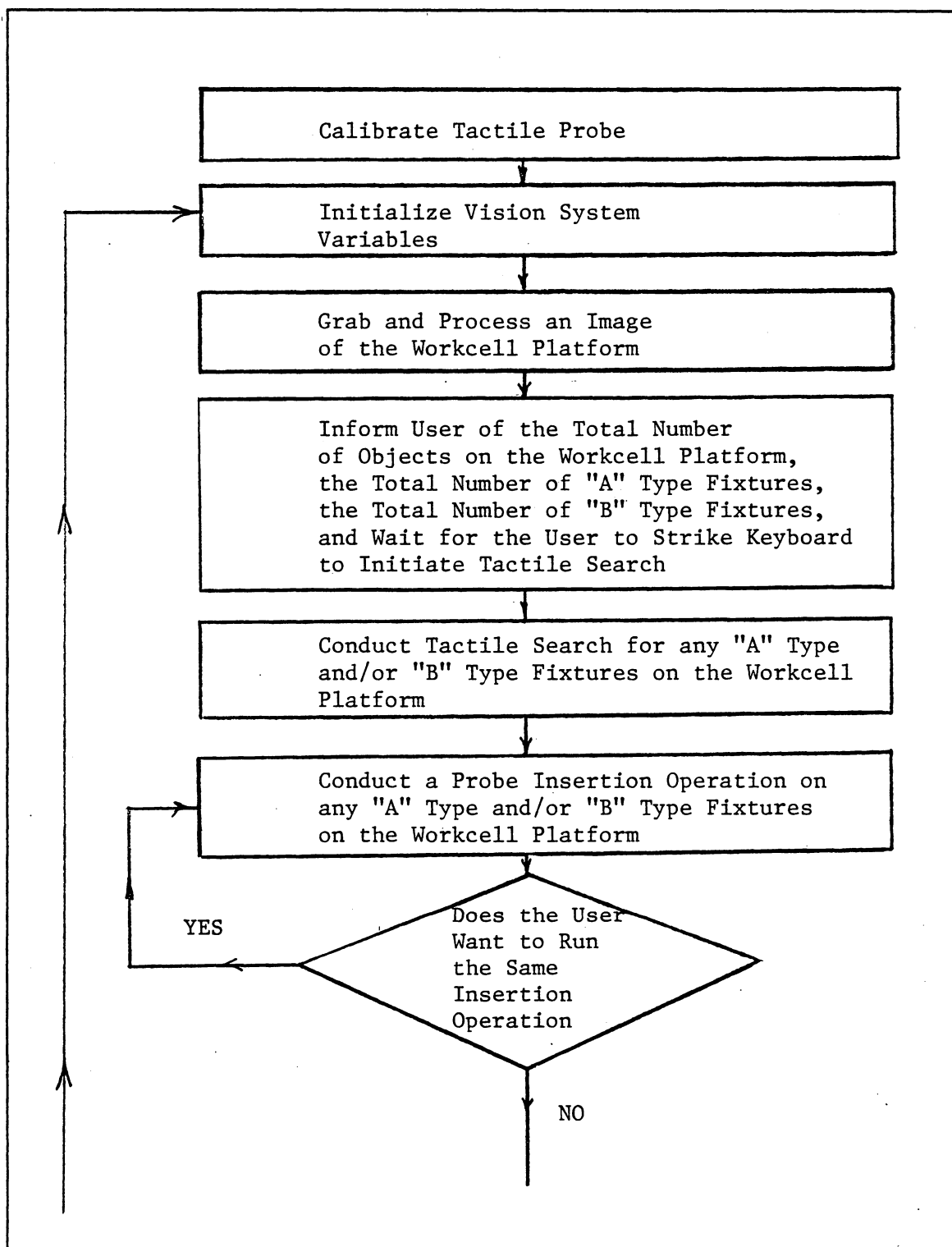


Figure 4. Main System Flowchart

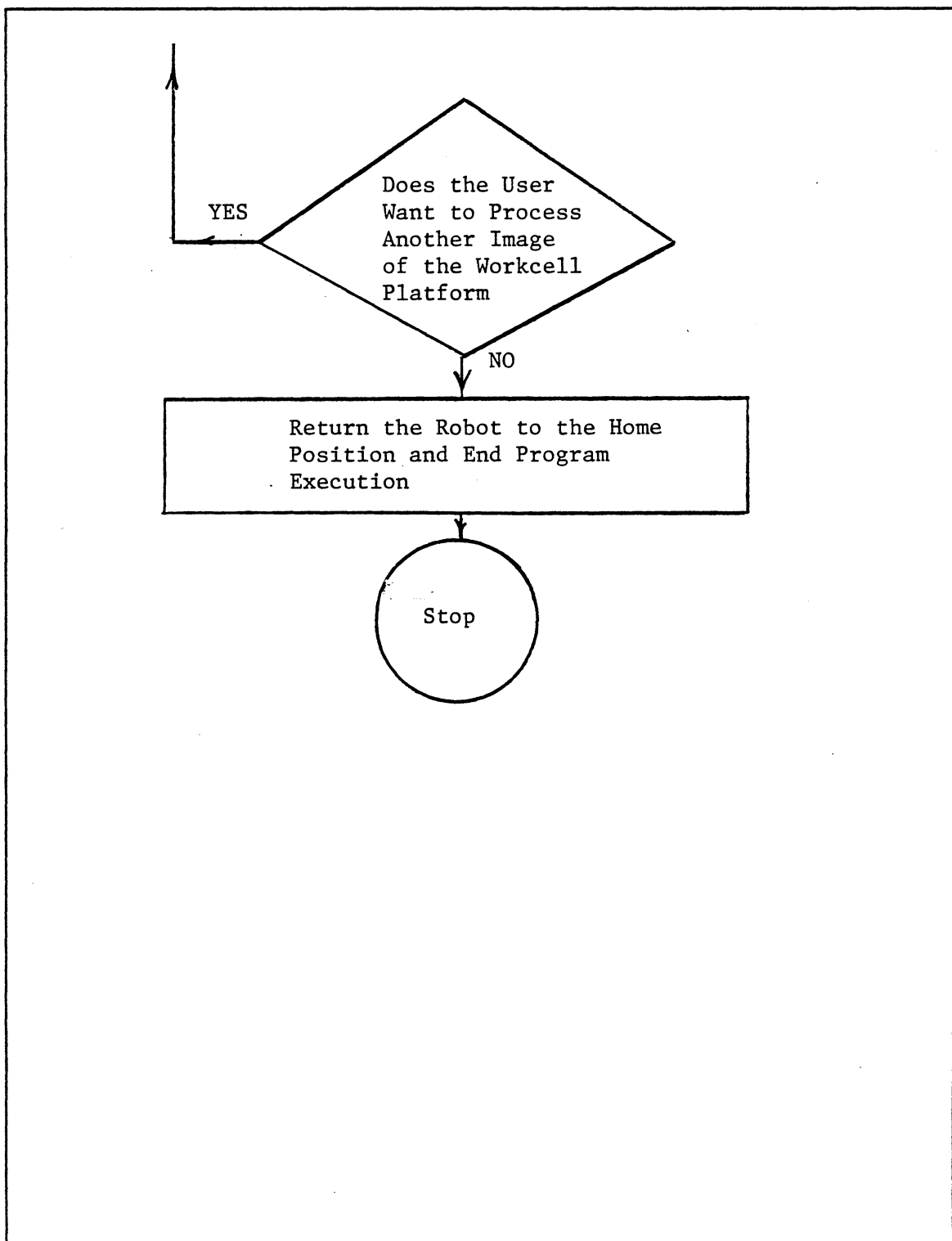


Figure 5. Main System Flowchart (continued)

adjacent sides have a included angle greater than 90 degrees. This section describes the hardware and software that make up this system.

3.3.1 Vision System Hardware

The vision system hardware, Figure 6, consists of the following:

1. RCA RS-170 Camera,
2. PCVISION frame grabber system,
3. Two PANASONIC TV Monitors, and
4. IBM AT.

The camera is used to constantly transmit a RS-170 video signal to the frame grabber board and to one of the TV monitors. The frame grabber board connects directly into the main bus of the IBM AT, and consists of 256 kbytes of memory, a flash A/D converter, and a D/A converter. On command, the A/D converter digitizes the RS-170 video signal and stores the gray scale values into a 480 X 512 array of memory, which is broken into four 64 kbyte segments. Subsequently, through the use of a look up table and a D/A converter, the frame grabber board sends a conditioned signal to the other TV monitor.

The IBM AT has access to the frame grabber memory, and through software processes the image data.

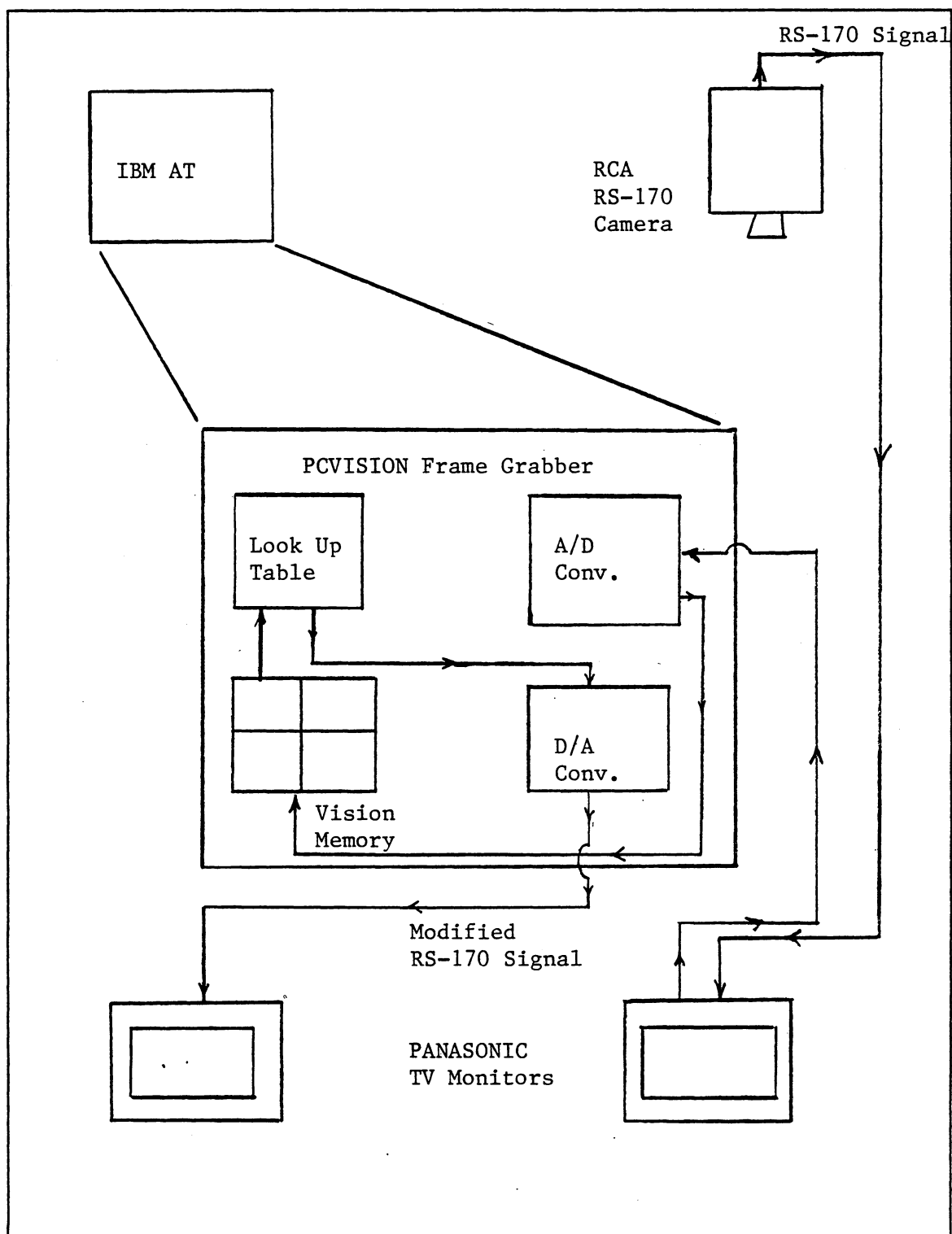


Figure 6. Vision System Hardware

3.3.2 Vision System Software

The vision system software resides in the IBM AT and was written in FORTRAN. The software can be divided into the four following segments:

1. Vision Hardware Interface,
2. Object Boundary Identification and Tracking,
3. Object Boundary Definition, and
4. Circuit Board Fixture Identification and Positioning.

Each of these segments is discussed in the following sub-sections.

3.3.2.1 Vision Hardware Interface

Command access to the frame grabber system is conducted through the vision subroutine, FRMGRB, which can be referred to in Appendix B. FRMGRB accesses three assembly language subroutines, VINIT, FREEZE, and RAMP, which were written by Dr. Michael P. Deisenroth and which can be referred to in Appendix G. FRMGRB first calls upon VINIT to initialize the frame grabber board. FREEZE is then called upon to digitize the RS-170 signal and store the image into memory. Finally, FRMGRB calls upon RAMP to set up the look up table for the board's D/A converter, and then returns execution to the main program.

3.3.2.2 Object Boundary Identification and Tracking

After storing the digitized image into memory, the vision software then converts the gray scale data into binary data through the use of a simple thresholding technique. A "Boundary Continuation" method is used to detect and track object boundaries within the image. Each row of the image array is scanned for the birth, continuation, or death of an object boundary.

An object can either be classified as a body or a hole. Objects can have both parent objects and offspring objects, which result from boundary births and boundary merges. For example, when a hole is born, its parent is its surrounding body; and when the left boundary of one body merges with the right boundary of another body, the latter body becomes the parent of the former.

Through use of the Boundary Continuation method, the vision software creates a data base which describes the boundaries of every object within the image array as well as its parent/offspring relationships. A much more detailed description of this method and the associated software is given in Chapter 4.

3.3.2.3 Object Boundary Definition

Once the initial vision data base has been created from the binary image, the vision software then utilizes a "Boundary Tracking" method to define the corner points of each object within the binary image array. Corner

points are determined by picking three subsequent points along an object's boundary. The three points define two boundary vectors. A vector analysis is then conducted to determine if the second point of the trio is a corner point.

To locate the corner points of an object, the vision software scans along the object's boundary and uses the vector analysis to detect the presence of a corner point. The two main tasks of this software segment are to determine the routing along an object's boundary and to conduct the vector analysis in order to detect the object's corner points.

Through the use of the Boundary Tracking method, the vision software creates a data base which stores the location of each object corner point identified in the binary image array. A more detailed description of this method and the associated software is given in Chapter 4.

3.3.2.4 Circuit Board Fixture Identification and Positioning

The last major task of the vision software is the identification and positioning of fixtures within the image array. This segment of code scans through the feature data base of every object in the image array. If an object has six corner points, and an area and perimeter that are within bounds of either an "A" type or "B" type fixture, then it is considered to be one of these respective fixtures. The parameter limits for each fixture were determined by processing three images of each, computing the average and standard deviation, and computing the 3 SIGMA limits.

The sequential relationship among an object's corner points is always the same, irregardless of the object's orientation. Thus, if just one unique corner point can be identified within a data base, then all the corner points can be identified by following the sequencing relationship among the points.

Once an object has been identified as a fixture, the vision software then locates the unique corner point by checking the distance between each corner point and the object's centroid.

The execution of this software segment creates a data base which identifies objects as fixtures and which identifies and locates every fixture corner point. A more detailed description of this software is given in section Chapter 4.

3.4 TACTILE SYSTEM

The tactile system developed in this research is a self-calibrating system with a positional repeatability of 0.009 inches. This section describes the hardware and software that make up this system.

3.4.1 Tactile System Hardware

The tactile system hardware consists of the following:

1. IBM 7545 Robot and Controller,
2. Tactile Probe,

3. Workcell Platform, and
4. Circuit Board Fixtures.

The IBM 7545 robot serves as the positioning device for the tactile system and has a positional repeatability of 0.002 inches. It receives its position coordinates from the executing software in the controller.

The IBM 7545 controller performs two functions in the tactile system. Its primary task is to command the robot to move to various locations. Its secondary task is to check for an input port being high, indicating that the tactile probe has made contact with the fixture. An input port is high when it sees a ground potential between one of its nodes and ground.

The tactile probe, Figure 7, consists of a "Face Plate" and a needle. The Face Plate, which can be referred to in Appendix H, mates with a "Lower Body Interface" at the end of the robot's Z axis shaft and has a hole, which houses the probe needle. The needle is held in place by two set screws.

The Lower Body Interface is connected electrically to Input Port #10 on the controller. Through metal to metal contact, the probe needle is also connected to Input Port #10.

The Face Plate was built using IEOR shop facilities. The probe needle is a simple, truncated sewing needle. The Lower Body Interface was de-

signed by Peter Mertens [11] and was also built using IEOR shop facilities.

The Workcell Platform, which can be referred to in Appendix H, was built using IEOR shop facilities and consists of a 36" X 30" X 6" wooden platform covered on the surface by a grid of wire.

The grid of wire is connected electrically to the ground side of a 5 volt power supply. When a metallic fixture is placed on the platform, it obtains a ground potential through metal to metal contact with the grid. Thus when the probe needle makes contact with a fixture, current flows through the input port, causing it to see ground potential.

In addition, a calibration block is attached by screws to the top surface of the platform. This object is used as the calibration surface for the tactile probe.

The circuit board fixtures, which can be referred to Appendix H, are made of steel plate and are of two varieties. The type "A" fixture is the smaller of the two and has a pattern of five 0.055 inch holes drilled through its top surface. The type "B" fixture has a pattern of nine 0.055 inch holes drilled through its top surface.

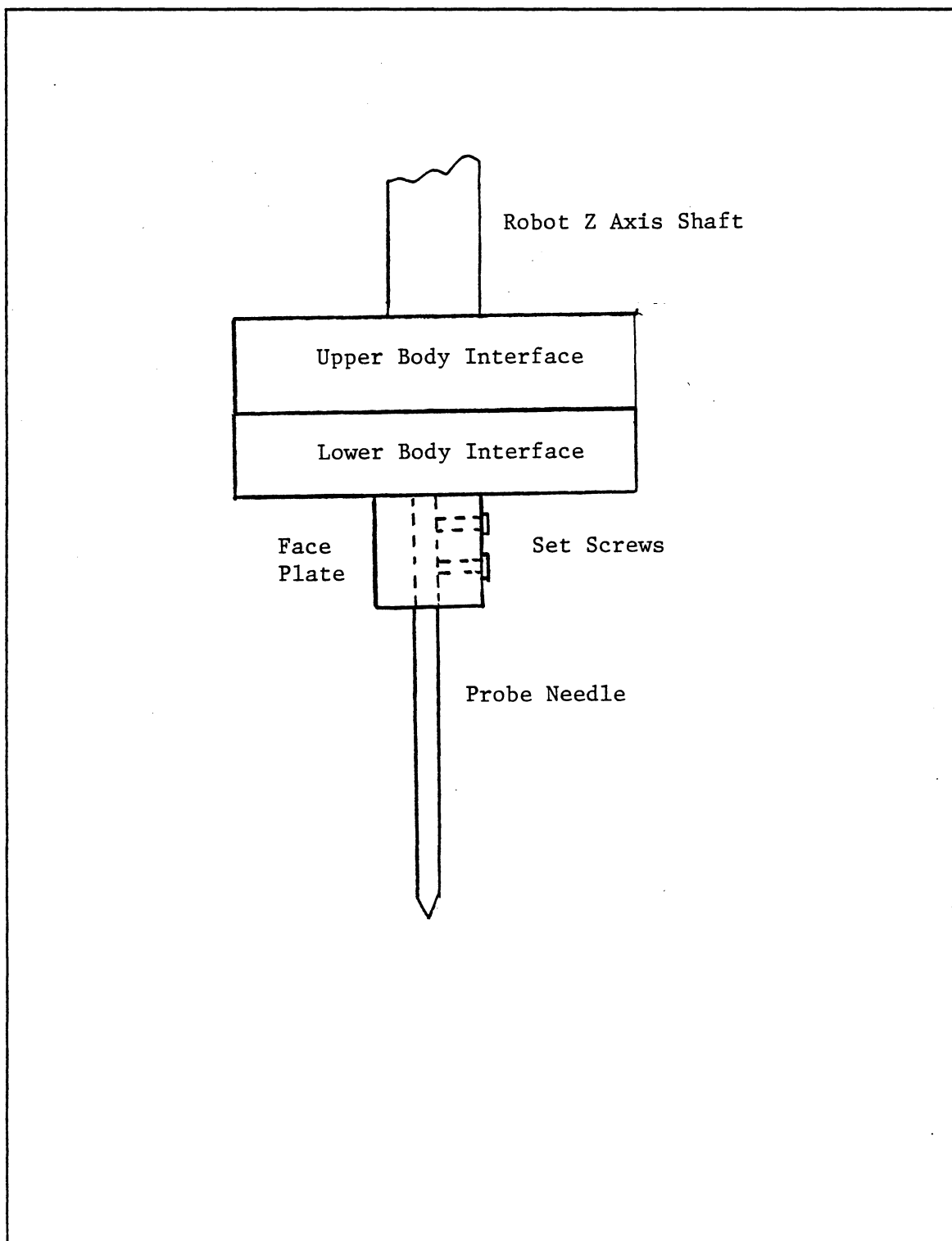


Figure 7. Tactile Probe

3.4.2 Tactile System Software

The tactile system software developed in this research reside in both the IBM AT and the IBM 7545 controller and have a hierarchial relationship. The software written for the IBM AT acts as the master program. It computes points along a search vector, downloads the points to the slave program in the controller, commands the slave program to execute, and accepts the point of contact from the slave program upon completion of execution. This software is written in FORTRAN.

The software written for the IBM 7545 controller acts as the slave program. It accepts a group of points from the master program in the IBM AT and commands the robot to move to each point until contact is made with an object at the tip of the probe. This software was written in AML/E.

Through this hierarchial relationship, the software do the following:

1. Calibrate the Tactile Probe,
2. Perform Tactile Searches, and
3. Define Regions.

The initial task of the tactile software is to calibrate the tactile probe by determining the direction and magnitude of the probe's eccentricity. This is accomplished by touching the probe against the two normal surfaces of the calibration block while the robot shaft is rotated at 0 degrees and 180 degrees. The data collected from this operation is used in the

computation of the magnitude and direction of the probe's eccentricity. These parameters are in turn used in a positional error compensation equation, which is used to eliminate positioning error during tactile searches.

The tactile software's second major task is to conduct searches for fixture sides. This is accomplished by doing the following:

1. Extracting three corner points of a fixture from the vision data base,
2. Computing two side vectors defined by the three points,
3. Computing two vectors that are normal to the side vectors,
4. Commanding the robot to conduct tactile searches along the normal vectors, until two points along each side have been obtained,
5. Using the acquired points to compute two new side vectors which more accurately define the sides of the fixture,
6. Computing two vectors that are normal to the new side vectors, and
7. Commanding the robot to conduct tactile searches along the normal vectors, until two points along each side have been obtained.

The two pass search technique is used to overcome the locational inaccuracy of the vision base corner points and to maximize the power of the error compensation system.

In addition to the error compensation techniques discussed above, it should be noted that diameter of the probe at the point of contact is also accounted for in the location of a fixture's side. A tactile search only concludes when contact is made between the very tip of the probe needle and the side of the fixture. This quality of the search technique essentially eliminates any error due to variation of probe diameter at the point of contact.

The final task of the tactile software is to compute the coordinates of the corner points of the rectangular region defined by the fixture. This is accomplished by doing the following:

1. Using the four points obtained from the tactile searches to mathematically define two lines,
2. Finding the intersection of the two lines in order to define the origin (lower left point) of the region,
3. Computing two vectors that originate from the origin and which follow the two defined lines,
4. Adding the vectors to the origin to define the two corner points (upper left and lower right corner points) which along with the origin define the axes of the region, and
5. Adding the first vector to the lower right region point to define the upper right region point.

This segment of code creates a data base which defines a region for every fixture observed by the vision system.

The error compensation techniques, region defining techniques and associated tactile software are discussed in greater detail in Chapter 5.

3.5 PROBE INSERTION SOFTWARE

The probe insertion software developed in this research reside in both the IBM AT and the IBM 7545 controller and have a hierarchial relationship. The software written for the IBM AT acts as the master program. It down loads region corner points to the slave routine in the controller and commands the routine to conduct a probe insertion routine. This software was written in FORTRAN.

The software written for the IBM 7545 controller acts as the slave program. It accepts region corner points from the master routine and then commands the robot to insert the tactile probe into each hole defined within the fixture defined region. This software was written in AML/E. Through this hierarchial relationship, the executing software do the following:

1. Define a region representing a fixture on the workcell platform,
2. Compute the location (in terms of absolute coordinates) of every 0.055 inch hole in the fixture using the relative hole locations within the defined region, and
3. Command the robot to insert the tip of the tactile probe into every fixture hole, following a prescribed sequence.

The pattern of holes into which the probe is to be inserted is unique to each fixture type.

A region is a coordinate system defined within the absolute coordinate system of the robot. A region in AML/E is defined by four points in the robot workcell and by the arbitrary distance assigned between each succeeding corner point. By defining the corner points of a fixture to be the corner points of a region, every point on the surface of the fixture can be reached by commanding the robot to move to a point relative to the regional axes.

The major thrust of this concept is that if the locations of a set of holes are known relative to a fixture's corner points, then the holes can easily be located by simply defining the corner points of the fixture as the corner points of a region within an AML/E program. A more thorough discussion of regions and the probe insertion software is given in Chapter 6.

3.6 USER INFORMATION SOFTWARE

The user information software developed in this research resides in the IBM AT and was written in FORTRAN. The purpose of this software is to interactively communicate with the user to do the following:

1. Inform the user of the current operation in progress,

2. Inform the user of how many objects were detected by the vision system and how many of them were "A" type and "B" type fixtures, and
3. Input the user's desire to either conduct another insertion operation, grab and process another image of the workcell platform, or return the robot to the home position and end program execution.

The graphics software is broken into three main subroutines: GRAPH1, GRAPH2, and GRAPH3, and can be referred to in Appendix E. All of the subroutines call upon a library of assembly language graphics routines written by Drew Snader [13]. The following subsections will describe these three subroutines.

3.6.1 GRAPH1

GRAPH1 accepts the value stored in LINNUM from a calling routine and prints the corresponding message to the CRT screen. A list of the values of LINNUM with their corresponding messages is given below:

1. IMAGE DATA IS BEING PROCESSED
2. PROBE SEARCH FOR TYPE "A" OBJECT IS BEING CONDUCTED
3. PROBE SEARCH FOR TYPE "B" OBJECT IS BEING CONDUCTED
4. INSERTION OPERATION FOR TYPE "A" OBJECT IS BEING CONDUCTED
5. INSERTION OPERATION FOR TYPE "B" OBJECT IS BEING CONDUCTED
6. PROBE CALIBRATION IS BEING CONDUCTED

GRAPH1 prints a message to the screen by first calling CLS, which clears the CRT screen and sets the foreground and background colors. The position of the cursor on the screen is then set by calling LOCATE. Finally the message is printed on to the screen by sequentially assigning each character in the message to the variable, CHAR, and subsequently calling WRCHAT, which transmits the ASCII character to the screen and increments the cursor.

3.6.2 GRAPH2

GRAPH2 accepts the total number of objects detected by the vision system (BDCNT), the total number of "A" type fixtures (ANUM) identified by the vision system, and the total number of "B" type fixtures (BNUM) identified by the vision system from the calling routine and prints an appropriate message on the CRT screen.

In addition, GRAPH2 also waits for the user to strike any key on the keyboard before returning to the calling routine. GRAPH2 uses the same procedure as GRAPH1 to print the message to the screen. Subroutine, INKEY, is used to input the ASCII character from the keyboard.

3.6.3 GRAPH3

GRAPH3 prints the following message to the CRT screen:

ENTER:

1. TO REINITIATE INSERTION PROCEDURE

2. TO DETECT AND LOCATE NEW OBJECTS
3. TO END PROGRAM

It then waits for the user to enter an appropriate response on the keyboard. If the user enters any other character, the program will continue to prompt the user for anyone of the three characters. The integer value of the entered character is stored in the variable, KEY, and is returned to the calling routine. GRAPH3 prints to the screen and waits for a keyboard entry in the same fashion as GRAPH2.

3.7 COMMUNICATION SYSTEM

The communication system developed in this research permits full communication between an IBM PC/AT and an IBM 7545 controller. This section describes the hardware and software that make up this system.

3.7.1 Communication System Hardware

The communication hardware consists of a RS-232 interface between the IBM AT and the IBM 7545 controller, which uses the following protocol:

1. Full Duplex Transmission,
2. Baud Rate = 4800,
3. Parity = Even,
4. Data Bits = 7, and
5. Stop Bits = 2.

3.7.2 Communication System Software

The communication software library, which resides in the IBM AT, was written in assembly language (Intel 8088) and can be accessed by a program written in PROFESSIONAL FORTRAN. This sub-section discusses the communication protocol used between the IBM PC and the IBM 7545 as well as the individual subroutines in the library.

3.7.2.1 Communication Protocol

The information in this section was obtained from the AML/E Version 3.0 Users Manual [3].

The communications protocol used by the IBM PC and the IBM 7545 controller is a transaction based protocol with the host controlling all transactions. A communication transaction consists of the following:

1. An Identifier,
2. A Record, and
3. An Optional collection of identifiers, records, requests, and responses.

Transactions are initiated by the IBM PC transmitting an identifier to the controller. After the transaction has been started, the identifiers are appended to the beginning of each record.

When communicating with the controller, the IBM PC can send anyone of the following single character identifiers:

1. Read (R),
2. Execute (X),
3. Data(D),
4. Control (C), and
5. End (E).

The R identifier is used to initiate a communication sequence in which the IBM PC (host) reads program variables from the controller. The X identifier is used to initiate a communication sequence where the host commands the controller to perform an action such as returning the robot to the home position. The D record is used to initiate the transmission of values from the host to program variables in the controller or vice versa. The C identifier is used to initiate a communication sequence in which the host controls the execution of a program by the controller. The E identifier is used to signify the end of a data transmission.

A record, which is shown below:

ID	BC	DATA	CS	Cr	Lf

contains an identifier (ID), byte count (BC), data, check sum (CS), and terminator (Cr and LF). The identifiers were discussed above. The byte count contains two hexadecimal digits representing the byte length of the data. Its value can range from 01 to 10 (1 to 16 decimal).

All data to be transmitted by the communication system is represented in either fixed point or AML/E floating point format. Fixed point format involves four bytes of data stored in two's complement notation.

AML/E floating point format, which can be seen below, is the format in which AML/E stores real numbers.

X	XXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Mantissa	Exponent	Fractional Mantissa
Sign bit	-64 to 63	0, or 0.5 to 1

The high order bit is the mantissa sign bit. The mantissa is positive if the sign bit is 0 and negative if the sign bit is 1. The next seven bits represent the exponent in two's complement notation. The remaining 24 bits are the fractional mantissa. The mantissa must be 0 or between 0.5 and 1. Every bit in the mantissa is a negative power of 2.

FORTTRAN floating point format, which can be seen below, is different from the AML/E version.

X	XXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Sign Bit	Exponent	Fraction
	0 to 255	Eight Decimal Digits

The mantissa, which is stored in the highest byte, has a value which ranges from 0 to 255. The actual exponent is determined by subtracting a bias of 127 from the mantissa. The fraction, which is stored in the remaining three bytes, is the decimal number on the right side of the

decimal point. A number is obtained by concatenating the following forms:

Sign. 1. fraction* 2** Actual Exponent

Any transmission of a real number from the host to the controller first requires the conversion of the number from FORTRAN floating point format to AML/E floating point format. Likewise, any real number transmitted from the controller to host has to be converted from AML/E format to FORTRAN format before it can be used by a FORTRAN program.

The check sum is the modulo 256 of all the hex digit pairs in the data bytes. It is not counted as part of the data.

The controller uses special ASCII characters to control transactions.

These characters are the following:

1. Ack
2. Cr
3. Eot
4. Lf
5. Nak
6. Nul
7. Xoff
8. Xon

A detailed explanation of these characters can be found in the AML/E Version 3.0 Users Manual.

The following sub-sections describe the subroutines available in the communication software library.

3.7.2.2 EXEC

The communication subroutine, EXEC, which can be referred to in Appendix F, performs a remote function related to the operator control panel. The calling sequence for EXEC is:

```
CALL EXEC (CODE,ERROR)
```

where:

CODE (INTEGER*2) can be one of the following:

- 11 - Return Home,
- 12 - Recall Memory,
- 13 - Reset Error,
- 20 - Auto,
- 22 - Start Cycle,
- 23 - Stop Cycle,
- 24 - Stop and Mem,
- 25 - Step,
- 31 - Application 1,
- 32 - Application 2,
- 33 - Application 3,
- 34 - Application 4,
- 35 - Application 5.

and ERROR (INTEGER*2) contains the appropriate error return code:

- 0 - Communication was successfully completed,
- 1 - An EOT was encountered, communication was not successful, and
- 2 - A NAK was encountered, communication was not successful.

EXEC transmits a typical execution command, shown in Figure 8, by first calling COMINIT to initialize the serial communication port. COMOUT is then called to transmit an "X" to the controller. Subsequently, XNFIN is called to wait for the transmission of an Xon and Xoff from the controller. The variable, ERROR, is checked for the error return code. If the transmission is successful, then routine execution is continued. If the error code reveals an Eot, then a "1" is stored in ERROR and execution is returned to the calling routine. If the error code reveals a Nak, then a "2" is stored in ERROR and execution is returned to the calling routine.

The value stored in CODE is loaded into the register, DS, and DEC16OUT is called to transmit the appropriate ASCII characters to the controller. EXEC then calls IDEND to transmit a CR and Lf to the controller.

If CODE equals 11, 24, or 23, then EXEC calls XNFIN to wait for an Xon and Xoff from the controller. Otherwise, it calls upon ACKIN to wait for an ACK. In either case, it checks the return error code and sets the value of ERROR before returning execution to the calling routine.

3.7.2.3 CNTRL

The communication subroutine, CNTRL, which can be referred to in Appendix F, controls the execution of an application program. The calling sequence for CNTRL is:

```
CALL CNTRL (CODE, ERROR)
```

where:

HOST	CONTRL.	EXPLANATION
"X" -----		The host sends the controller an X identifier to indicate the start of an execute transaction.
----- Xoff		The controller responds with an initial Xoff.
----- Xon		The controller is able to accept the X record.
"23"CrLf -----		The host sends the stop cycle operand to the controller followed by a Cr and Lf to indicate the end of the record.
----- Xoff		The host sends an Xoff, which indicates it received the command.
----- Xoff		The controller sends an Xoff 25 seconds until the stop cycle command is executed.
----- Xon		The controller sends an Xon to indicate the completion of the stop cycle command.

Figure 8. EXEC Transmission Sequence

CODE (INTEGER*2) can be one of the following:

- 1 - Suspend Program Execution,
- 2 - Restart Program Execution,
- 4 - Execute Until Next Terminator Is
Encountered, and
- 20 - Reset Controller.

and ERROR (INTEGER*2) is the same as described in
EXEC.

CNTRL transmits an execution command, Figure 9, by first calling COMINIT to initialize the serial communication port. COUT is then called to transmit a "C" to the controller. Subsequently, XNFIN is called to wait for the transmission of an Xon and Xoff from the controller. The variable, ERROR, is checked for the error return code. If the transmission is successful, then routine execution is continued. If the error code reveals an Eot, then a "1" is stored in ERROR and execution is returned to the calling routine. If the error code reveals a Nak, then a "2" is stored in ERROR and execution is returned to the calling routine.

The value stored in CODE is loaded into the register, DS. If the value is equal to 20, then DEC16OUT is called to transmit the appropriate ASCII characters to the controller. IDEND is subsequently called to send CR and Lf. Afterwards, EXEC calls on ACKIN to wait for the controller to send an Ack. It then checks the return error code and sets ERROR before returning program execution to the calling routine.

If the value in CODE does not equal to 20, then the value is loaded into the variable, CBUFF, and CHAROUT is repeatedly called until the appro-

priate ASCII characters are transmitted to the controller. XNFIN is then called to wait for the controller to send an XON and Xoff. The return error code is checked and the value in ERROR set before program execution is returned to the calling routine.

3.7.2.4 WRITEP

The subroutine, WRITEP, which can be referred to in Appendix F, transmits the X, Y, Z, and R values of a point into four consecutive controller memory locations. The calling sequence for WRITEP is:

```
CALL WRITEP (VARSTRT, X, Y, Z, R, ERROR)
```

where:

VARSTRT (INTEGER*2) is the starting variable number,

X (REAL) is the X coordinate of the point,

Y (REAL) is the Y coordinate of the point,

Z (REAL) is the Z coordinate of the point,

R (REAL) is the R coordinate of the point, and

ERROR (INTEGER*2) is the same as described in EXEC.

WRITEP transmits a point to the controller, Figure 10, by first calling COMINIT to initialize the serial communication port. The values in VARSTRT, X, Y, Z, and R are then read into the variable, VARNUM, and the buffer, PBUFF, respectively.

COUT is called to transmit a "C" to the controller. Subsequently, XNFIN is called to wait for the transmission of an Xon and Xoff from the con-

HOST	CONTRL.	EXPLANATION
"C" -----		The host sends the controller a C identifier to indicate the start of a control transaction.
-----	Xoff	The controller responds with an initial Xoff.
-----	Xon	The controller is able to accept the C record.
"1"CrLf -----		The host sends the suspend program operand to the controller, followed by a Cr and Lf to indicate the end of the record.
-----	Xoff	The host sends an Xoff which indicates that it received the command.
-----	Xoff	The controller sends an Xoff every 25 seconds until the suspend program command is executed.
-----	Xon	The controller sends an Xon to indicate the completion of the suspend program command.

Figure 9. CNTRL Transmission Sequence

troller. The variable, ERROR, is checked for the error return code. If the transmission is successful, then routine execution is continued. If the error code reveals an Eot, then a "1" is stored in ERROR and execution is returned to the calling routine. If the error code reveals a Nak, then a "2" is stored in ERROR and execution is returned to the calling routine.

DEC16OUT is called to transmit "80" to the controller. Subsequently IDEND is called to transmit Cr and Lf. ACKIN is then called to wait for an Ack from the controller. The error return code is again checked for the transmission status, and the action taken is the same as described previously.

The starting variable number is loaded into the variable, DDBUF, and the representative ASCII characters are sent to the controller through the call of DRECOUT. ACKIN is then called to wait for an Ack from the controller, and the error return code is subsequently checked for the status of the transmission.

The lower two bytes of the value read in from X is loaded from PBUFF into REALW1, while the upper two bytes of the value are loaded into REALW2. WRITEP then calls CNVRT86Z80 to transform the value from FORTRAN floating point format to AML/E floating point format. Subsequently, DRECOUT is called to transmit the associated D record to the controller, and ACKIN is called to wait for the controller's acknowledgment of the record. The error return code is then checked for the success or failure of the transmission and the appropriate action is taken.

The sequence of operations described above is repeated for the values stored in Y, Z, and R. WRITEP then calls EGOUT to send an "E" record to the controller, indicating the successful completion of the transaction. This is followed by the call of ACKIN and the checking of the error return code before program execution is returned to the calling routine.

3.7.2.5 WRITER

The communication subroutine, WRITER, which can be referred to in Appendix F, transmits a real value into controller memory. The calling sequence for WRITER is:

```
CALL WRITER (VARSTRT, VAL, ERROR)
```

where:

VARSTRT (INTEGER*2) is the starting variable number,

VAL (REAL) is the value to be sent to the controller, and

ERROR (INTEGER*2) is the same as described in EXEC.

The manner in which WRITER transmits a real value to the controller is identical to the sequence of operations performed by WRITEP with the exception that WRITER reads in and transmits one real value as opposed to four.

HOST	CONTRL.	EXPLANATION
"C" -----		The host sends the controller a C identifier to indicate the start of a control transaction.
----- Xoff		The controller responds with an initial Xoff.
----- Xon		The controller is able to accept the C record.
"80"CrLf -----		The host sends the write variable operand to the controller followed by a Cr and Lf to indicate the end of the record.
----- Ack		The controller sends an acknowledgment of the record.
"D040010000818"CrLf		The host sends a D record with the identifier, byte count, starting variable number, the number of variables to be sent and the terminator.
----- Ack		The controller sends an acknowledgment of the record.
"EG"CrLf -----		The host sends an E record, which contains the identifier (E), the op-code (G), and the terminator.
----- Ack		The controller sends an acknowledgment of the E record.

Figure 10. WRITEP Transmission Sequence

3.7.2.6 READP

The communication subroutine, READP, which can be referred to in Appendix F, reads the X, Y, Z, and R values of a point from four consecutive controller memory locations. The calling sequence for READP is

```
CALL READP (VARSTRT, X, Y, Z, R, ERROR)
```

where:

VARSTRT (INTEGER*2) is the starting variable number,

X (REAL) is the X coordinate of the point,

Y (REAL) is the Y coordinate of the point,

Z (REAL) is the Z coordinate of the point,

R (REAL) is the R coordinate of the point, and

ERROR (INTEGER*2) is the same as described in EXEC.

READP reads a point from the controller, Figure 11, by first calling COMINIT to initialize the serial communication port. The value representing the starting variable number is loaded into the variable, VARNUM.

ROUT is called to send an "R" to the controller. XNFIN is subsequently called to wait for the controller to send an Xon and a Xoff. The variable, ERROR, is checked for the error return code. If the transmission is successful, then routine execution is continued. If the error code reveals an Eot, then a "1" is stored in ERROR and execution is returned to the calling routine. If the error code reveals a Nak, then a "2" is stored in ERROR and execution is returned to the calling routine.

If the transmission is successful, then READP then calls DEC16OUT to transmit an "80" to the controller. This is followed by the call of IDEND to transmit the terminators, Cr and Lf. Subsequently, READP calls ACKIN to wait for the acknowledgement of the record from the controller. The error return code is again checked and the appropriate action is taken.

If the previous transmission has been successful, then READP loads the starting variable number and the number of variables to be read, 4, into the variable DDBUFF. DRECOUT is then called to transmit the corresponding D record to the controller. DRECIN is subsequently called to read in the D record containing the variables, and to store the values in DDBUFF. The error return code is then checked and the appropriate action is taken.

The upper and lower two bytes of each real value stored in DDBUFF are sequentially loaded into REALW1 and REALW2, converted from AML/E floating point format to FORTRAN floating point format by CNVRTZ8086, and loaded into the variable, PBUFFW.

The real values are then loaded into the return variables, X, Y, Z, and R before program execution is returned to the calling routine.

3.7.2.7 READR

The communication subroutine, READR, which can be referred to in Appendix F, reads a real value from controller memory. The calling sequence for READR is:

HOST	CONTRL.	EXPLANATION
"R" -----		The host sends the controller a R identifier to indicate the start of a control transaction.
----- Xoff		The controller responds with an initial Xoff.
----- Xon		The controller is able to accept the R record.
"80"CrLf -----		The host sends the read variable operand to the controller, followed by a Cr and LF to indicate the end of the record.
----- Ack		The controller sends an acknowledgment of the record.
"D040010000818"CrLf		The host sends a D record with the identifier, byte count, starting variable number, the number of variables to be sent, and the terminator.
-----D-Record		The controller responds with a D record containing the variables.
Ack -----		The host acknowledges the D record.
----- "EG"CrLf		The controller sends a E record which contains the identifier (E), the op-code (G), and the terminator.
ACK -----		The host sends an acknowledgment of the E record.

Figure 11. READP Transmission Sequence

CALL READR (VARSTRT, VAL, ERROR)

where:

VARSTRT (INTEGER*2) is the starting variable number,

VAL (REAL) is the value to be sent to the controller, and

ERROR (INTEGER*2) is the same as described in EXEC.

The manner in which READR reads a real value from the controller is identical to the sequence of operations performed by READP with the exception that READR reads in one real value as opposed to four.

3.7.2.8 RSTATUS

The communication subroutine, RSTATUS, reads the machine status, reject status, or current instruction address from the controller. The calling sequence for RSTATUS is:

CALL RSTATUS (CODE, VAL1, VAL2, ERROR)

where:

CODE (INTEGER*2) is one of the following:

- 0 - Read Current Instruction Address,
- 1 - Read Machine Status, and
- 2 - Read Reject Status.

VAL1 (INTEGER*2) is a return value containing the following if CODE is equal to 01:

- 4 - Data Error,
- 6 - AML/E Error,
- 8 - Transmission Error,
- 16 - Over Time Error,
- 32 - Over Run Error,
- 64 - Power Failure, and

128 - Servo Failure.

VAL2 (INTEGER*2) will contain the additional information if VAL1 is equal to 6:

- 10 - Part Number Too Small For Pallet,
- 11 - Part Number Too Large For Pallet,
- 20 - Invalid Index For A Group, and
- 30 - Communications Not Established.

VAL2 (INTEGER*2) will contain the additional information if VAL1 is equal to 4:

- 0 - No Data Error Present,
- 1 - Bus Error,
- 2 - Memory Test Error,
- 17 - Arithmetic Error,
- 18 - Programming Error,
- 19 - Invalid Op Code,
- 20 - Invalid Data,
- 21 - Invalid Port Number,
- 23 - Stack Error,
- 24 - Address Error, and
- 64 - Point Out Of Workspace.

VAL1 (INTEGER*2) will contain the following if CODE is equal to 2:

- 16 - Record Format Error,
- 21 - Invalid Port Number,
- 32 - Undefined Record,
- 48 - Improper Start Up Sequence,
- 49 - C Record, But No Application,
- 64 - Point Out Of Workspace,
- 80 - Insufficient Memory,
- 81 - Invalid Robot Type,
- 82 - Double Select Error,
- 83 - Invalid Application Number,
- 84 - C Record Format Error,
- 96 - Invalid Identifier Send Before N Record,
- 112 - Xoff Time Out,
- 116 - Invalid Data,
- 117 - Too Much Data,
- 118 - Too Little Data,
- 128 - Manipulator Power Off,
- 160 - Xoff Time-Out State,
- 161 - Can Not Accept Command (Xon State),
- 162 - Communication Operation Not Allowed, and
- 170 - Controller Has Gone Off Line.

and ERROR (INTEGER*2) is the same as described in EXEC.

RSTATUS reads the status of the controller, Figure 12, by first calling COMINIT to initialize the serial communication port.

ROUT is called to transmit an "R" to the controller. Subsequently, XNFIN is called to wait for the transmission of an Xon and Xoff from the controller. The variable, ERROR, is checked for the error return code. If the transmission is successful, then routine execution is continued. If the error code reveals an Eot, then a "1" is stored in ERROR and execution is returned to the calling routine. If the error code reveals a Nak, then a "2" is stored in ERROR and execution is returned to the calling routine.

The value stored in CODE is loaded into variable, CBUFF, and CHAROUT is called to transmit the appropriate ASCII characters to the controller. RSTATUS then calls IDEND to transmit a CR and Lf to the controller.

If the value in CODE equals 1, then RSTATUS calls DRECIN to read in the D record transmitted by controller, and to store the data into DBUFF. The error return code is then checked to determine if the transmission was successful, and the appropriate action is taken. The upper byte of the data is then checked for meaningful information. If it contains valid data, then it is loaded into VAL2 and the lower byte of data is loaded into VAL1. If it does not contain valid data, then only the lower byte of data is loaded into a return variable.

If the value in CODE is equal to 0 or 2, then RSTATUS performs the same operations described before except it automatically loads the data into VAL1 and ignores the upper byte of the data since it is known in advance to contain no meaningful information.

When program execution is returned to the calling routine, the appropriate data is stored in VAL1, VAL2, and ERROR.

3.7.2.9 RDIDO

The communication subroutine, RDIDO, which can be referred to in Appendix F, reads the status of one of the 16 DI or DO ports from the controller.

The calling sequence for RDIDO is:

```
CALL RDIDO (PORTNUM, STATUS, ERROR)
```

where:

PORTNUM (INTEGER*2) is the desired port number:

- 1 - 16 For Input Ports 1 - 16, and
- 17 - 32 For Output Ports 1 - 16.

STATUS (INTEGER*2) is the status of the port:

- 1 - On, and
- 2 - Off.

and ERROR (INTEGER*2) is the same as described in EXEC.

RDIDO reads the status of a DI/DO port from the controller, Figure 13, by first calling COMINIT to initialize the serial communication port.

HOST	CONTRL.	EXPLANATION
"R" -----		The host sends the controller a R identifier to indicate the start of a control transaction.
----- Xoff		The controller responds with an initial Xoff.
----- Xon		The controller is able to accept the R record.
"08"CrLf -----		The host sends the read address operand to the controller followed by a Cr and Lf to indicate the end of the record.
-----"D02dddd"CsCrLf		The controller responds with a D record containing, the byte count, two bytes of data, the check sum, and the terminator. The data is the current instruction address.
Ack -----		The host acknowledges the D record.
----- "EG"CrLf		The controller sends a E record, which contains the identifier, (E), the op-code (G), and the terminator.
ACK -----		The host sends an acknowledgment of the E record.

Figure 12. RSTATUS Transmission Sequence

ROUT is called to transmit an "R" to the controller. Subsequently, XNFIN is called to wait for the transmission of an Xon and Xoff from the controller. The variable, ERROR, is checked for the error return code. If the transmission is successful, then routine execution is continued. If the error code reveals an Eot, then a "1" is stored in ERROR and execution is returned to the calling routine. If the error code reveals a Nak, then a "2" is stored in ERROR and execution is returned to the calling routine.

DEC16OUT is called next to transmit "10" to the controller. IDEND is subsequently called to send Cr and Lf. RDIDO then calls DRECIN to read in the D record containing the status information. The first two bytes of data represent the status of the input ports, while the second two bytes represent the status of the output ports. Each bit in a data byte represents a port. RDIDO then checks the error return code to insure proper transmission.

In addition to the status of the original 16 DI/DO ports, the controller also sends the status of an optional set of DI/DO points. To follow communication protocol, RDIDO calls DRECIN to read in the additional D record. However, the data, which is meaningless, is dumped into garbage variables.

After reading in the additional D record, RDIDO then loads the desired port number into the register, AX. The status of the port in question is determined by first locating the bit in the data which represents the

status of the port. The data is then extracted by performing an "AND" operation on the byte of data with the proper data mask.

When program execution is returned to the calling routine, the port status is stored in STATUS and the communication status is stored in ERROR.

3.7.2.10 PUTWAIT

The communication subroutine, PUTWAIT, waits for the execution of a PUT command by the IBM 7545 controller. The calling sequence for PUTWAIT is:

```
CALL PUTWAIT (ERROR)
```

where:

ERROR (INTEGER*2) is the same as described in EXEC.

PUTWAIT begins execution, Figure 15, by calling COMINIT to initialize the communication hardware. XNOUT is then called to transmit an Xon to the controller, putting the controller in the Xon state. DRECIN is then called to read in the first D record, which contains the starting variable number, the number of variables to be sent, etc.

The variable, ERROR, is checked for the error return code. If the transmission is successful, then routine execution is continued. If the error code reveals an Eot, then a "1" is stored in ERROR and execution is returned to the calling routine. If the error code reveals a Nak, then a "2" is stored in ERROR and execution is returned to the calling routine.

HOST	CONTRL.	EXPLANATION
"R" -----		The host sends the controller a R identifier to indicate the start of a control transaction.
----- Xoff		The controller responds with an initial Xoff.
----- Xon		The controller is able to accept the R record.
"10"CrLf -----		The host sends the read DI/DO status operand to the controller followed by a Cr and Lf to indicate the end of the record.
----"D04ddddddd"CsCrLf		The controller responds with a D record containing the, byte count, two bytes of data, the check sum, and the terminator. The data is the status of the original input and output ports
Ack -----		The host acknowledges the D record.
--- "D04ddddddd"CsCrLf		The controller responds with a D record containing the, byte count, two bytes of data, the check sum, and the terminator. The data is the status of the optional input and output ports
Ack -----		The host acknowledges the D record.

Figure 13. RDIDO Transmission Sequence

----- "EG"CrLf The controller sends a E
 record, which contains the
 identifier, (E), the op-code
 (G), and the terminator.

ACK ----- The host sends an
 acknowledgment of the
 E record.

Figure 14. RDIDO Transmission Sequence (cont'd)

DRECIN is called again to read in the second D record, which contains variable information. Since the purpose of this routine is to determine when the controller reaches a pre-determined point in its execution and not to read in the transmitted values, the data contained in the D record is placed in a garbage variable.

The error return code is checked once again and its value is placed in the return variable, ERROR, before program execution is returned to the calling routine.

3.7.2.11 Supporting Subroutines

This sub-section briefly describes all of the supporting subroutines in the communication software library. All of the routines can be referred to in Appendix F.

DRECOUT constructs a D record by assembling the D identifier, byte count, data, and check sum. It then sends the information package through the serial port to the IBM 7545 controller, followed by Cr and Lf. The data to be transmitted is stored in the variable, DBUFF, before entry into the subroutine. DRECOUT calls upon DOUT, IDEND, CHECKSUM, and CHAROUT.

DRECIN receives a D record from the 7545 controller. It determines whether or not the record is correct by computing the check sum of the record and comparing it to the transmitted value, and then transmits either an Ack or Nak to the controller depending upon the record's status.

HOST	CONTRL.	EXPLANATION
XON -----		The controller must be in the Xon state.
----- "D0502vvvvvnnnn"CsCrLf		When the application program encounters the PUT command, it sends a D record containing the op-code (02), starting variable number, the number of continuous variables, followed by Cr and LF.
Ack -----		The host acknowledges the D record.
----- D-Record		The controller sends a D record with the value for each of the variables.
Ack -----		The host acknowledges the D record.
----- "EG"CrLf		The controller sends an E record, which signifies the end of the transaction.
Ack -----		The host acknowledges the E record.

Figure 15. PUTWAIT Transmission Sequence

It then waits for an E record from the controller and sends an Ack when it arrives.

If the variable, DSET, is set equal to 1 upon entry to the subroutine, DRECIN will not wait for an E record, but instead will send either a Nak or Ack to the controller before returning program execution to the calling routine. DRECIN calls upon COMINCK, CHARIN, CHECKSUM, COMOUT, and ACKOUT.

CNVRT86Z80 converts a value stored in FORTRAN floating point format to a value stored in AML/E floating point format. The number to be converted is stored in the variable, REALW, and upon exit the converted number is stored in the variable, RBUFF. CNVRT86Z80 converts a value by determining which powers of 2 add up to equal the number to be converted, and by setting the appropriate bits in the mantissa and fraction bytes of the converted number. This routine utilizes the 8087 math co-processor and calls upon SETBIT.

CNVRTZ8086 converts a real number stored in AML/E floating point format to a value stored in FORTRAN floating point format. The number to be converted is stored in the variable, RBUFFW, upon entry into the subroutine. Upon exit, the converted number is stored in REALW. CNVRTZ8086 converts a value by checking each bit in the fraction bytes of the value to be converted, and adding the appropriate power of 2 to the converted value. This subroutine utilizes the 8087 co-processor and calls upon EXPADD.

CHECKSUM computes the modulo 256 of the data stored in DBUFF. Upon exit, the variable, CHKSUM contains the computed value. This subroutine utilizes the 8087 co-processor and calls upon no subroutines.

DEC16OUT converts the value stored in the register, DX, into its appropriate ASCII characters, and then sends the characters through the serial communications port to the IBM 7545 controller. This routine calls upon COMOUT, and was taken from the "Blue Book of Assembly Routines for the IBM PC&XT" [10].

CHAROUT converts the byte of data stored in the variable, CBUFF, into its appropriate ASCII characters and transmits the characters to the IBM controller through the serial communications port. This subroutine calls upon COMOUT.

CHARIN converts the ASCII characters stored in the register, AX, into their corresponding byte value, and stores this value in the variable, CBUFF. This routine does not call upon any other subroutines.

SETBIT sets a bit in the variable, RBUFF, to the value 1. The position of the bit set is stored in the variable, BITSET, upon entry into the subroutine. This routine does not call upon any other subroutines.

EXPADD receives a byte of the mantissa stored in AML/E format and converts the value into one stored in FORTRAN floating point format. Upon exit, the converted value is stored in St (0) of the 8087 math co-processor,

which is utilized in this subroutine. This routine does not call upon any other subroutines.

ROUT, COUT, DOUT, IDEND, XNOUT, ACKOUT, and EGOUT send the characters "R", "C", "D", ("Cr" and "Lf"), "Xon", "Ack", and ("E" and "G") respectively to the IBM 7545 controller through the serial communication port. Each subroutine calls upon COMOUT.

XNFIN waits for the IBM controller to send an Xoff and Xon to the host computer through the serial communication port. If the controller sends either an Eot or Nak, an error signal will be stored in the variable, ERROR. Otherwise, ERROR will be equal to zero upon exit. This subroutine calls upon COMINCK.

ACKIN waits for the controller to send an Ack to the host computer through the serial communication port. If the controller sends either an Eot or Nak, an error signal will be stored in the variable, ERROR. Otherwise, ERROR will equal to zero upon return to the calling routine. This subroutine calls upon COMINCK.

COMINIT initializes COM0 of the serial communications port with the following configuration:

1. Baud Rate = 4800,
2. Parity = Even,
3. Data Bits = 7, and

4. Stop Bits = 2.

This subroutine calls upon BIOS interrupt 14h and was taken from the referred book of assembly language routines.

COMINCK checks for data in the serial communication port input buffer. If the buffer is full, then the 8088 flags are set to non-zero and the input data is stored in the register, AL. If the buffer is empty, then the flags are set for zero status. This subroutine was taken from the referred book of assembly language routines.

COMOUT transmits the ASCII character stored in the register, AL, to the controller through the serial communication port. This subroutine calls upon BIOS interrupt 14h and was taken from the referred book of assembly language routines.

4.0 VISION SOFTWARE LIBRARY

The vision software library is comprised of the four following segments:

1. Vision Hardware Interface,
2. Object Boundary Identification and Tracking,
3. Object Boundary Definition, and
4. Circuit Board Fixture Identification and Positioning.

This chapter discusses the theory and software associated with the latter three. All software discussed can be referred to in Appendix B.

4.1 OBJECT BOUNDARY IDENTIFICATION AND TRACKING

Before entry into this segment of code, the vision subroutine, FRMGRB, is called to create a gray scale pixel array representation of the workcell platform. Each pixel in the array has a value ranging from 0 (black) to 255 (bright white). The objective of this software segment is to convert the array from a gray scale format to a binary format and to locate, identify, track, and obtain statistical data on object boundaries within the array.

4.1.1 Gray Scale to Binary Conversion

The conversion of the pixel array from a gray scale image to a binary image is accomplished through the use of a simple thresholding technique. The gray scale value of each pixel array element is checked and compared

to a threshold value. If the gray scale value is greater than or equal to the threshold value, then it is changed to 255 (bright white). If the gray scale value is less than the threshold value, then it is changed to 0 (black). In this research, the threshold value is 60. This value was chosen because it conceptually produced the best binary image during a trial and error analysis of threshold values.

The vision subroutine, CNVRT, performs the described conversion on the pixel array. CNVRT calls the assembly routine, VRDROW, to sequentially store each line of the pixel array into the variable, SCNROW. It then performs a binary conversion on each element within SCNROW.

To reduce the effects of shadowing on the binary image, every pixel within the first 50 lines of the array or beyond the 450th line are given a value of 255. In addition, every pixel within the first 50 columns or beyond the 400th column are given a value of 255.

After converting each element within a line, CNVRT calls upon the assembly routine, VWRROW, to store the elements in SCNROW into the original pixel array. The assembly vision routines, VWRROW and VRDROW were written by Dr. Michael P. Deisenroth and can be referred to in Appendix G.

4.1.2 Boundary Continuation, Birth, and Death

Object boundaries are identified and tracked using the Boundary Continuation method, which involves sequentially scanning each line of the bi-

nary pixel array and simultaneously updating the following descriptive arrays:

1. OLDROW,
2. NEWROW,
3. BTHROW,
4. DTHROW,
5. PARENT,
6. BNDTRK,
7. AREA,
8. XMOMNT, and
9. YMOMNT.

The array, OLDROW, Figure 16, describes the location and object number of every left and right boundary identified in the previous scan row. NEWROW describes the location and object number of every left and right boundary identified in the current scan row. BTHROW identifies which boundaries are continuations from the previous scan row and which boundaries were born in the current scan row. DTHROW identifies which boundaries are continuations from the previous scan row and which boundaries have terminated in the current scan row. PARENT describes which objects are parents and which objects are their respective offspring. BNDTRK describes the position of every object boundary at each scan line. If an object's boundary does not exist, then BNDTRK contains the value, 514, for the object's boundary position. AREA lists the area of every iden-

tified object. XMOMNT lists the X coordinate of every identified object; and YMOMNT lists the Y coordinate of every identified object.

An object's left boundary is determined by the transition of a white pixel to a subsequent black pixel. An object's right boundary is determined by the transition of a black pixel to a subsequent white pixel.

An object, Figure 19, can either be a body or a hole. A body's left boundary is always at a lesser column position than its right; and a hole's left boundary is always at a greater column position than its right.

A parent object, Figure 20, is an object which is comprised of a single defined object and a group of objects which either merged with the defined object or terminated within the object. An example is a body which has a hole within its boundaries, and which is merged with another body.

An object with terminated boundaries is considered complete, while an object with non-terminated boundaries is considered undefined. Thus an object which covers the last row of the pixel array would be considered undefined unless it was externally defined as complete.

To determine whether an object's boundaries continue from the previous scan line to the current scan line, the continuation criteria equations, shown in Figure 21, are applied to every left and right boundary iden-

OLDROW (2, 100, 2)

1 - Left Bnd.	1 - 1st Bnd.	1 - Column # In In Pixel Array
2 - Right Bnd.	2 - 2nd Bnd.	2 - Object # Of Boundary
	3 - 3rd Bnd.	
	.	
	.	
	100 - 100th Bnd.	

NEWROW (2, 100, 2)

1 - Left Bnd.	1 - 1st Bnd.	1 - Column # In In Pixel Array
2 - Right Bnd.	2 - 2nd Bnd.	2 - Object # Of Boundary
	3 - 3rd Bnd.	
	.	
	.	
	100 - 100th Bnd.	

BTHROW (2, 100, 2)

1 - Left Bnd.	1 - 1st Bnd.	1 - Equals: 0 - New Bnd.
2 - Right Bnd.	2 - 2nd Bnd.	1 - Cont. Bnd.
	3 - 3rd Bnd.	2 - Equals: Cont. Bnd. #
	.	in
	.	OLDROW
	.	
	100 - 100th Bnd.	

Figure 16. Descriptive Array Elements Used For Boundary Definition

DTHROW (2, 100, 2)

1 - Left Bnd.	1 - 1st Bnd.	1 - Equals: 0 - Dead Bnd.
2 - Right Bnd.	2 - 2nd Bnd.	1 - Cont. Bnd.
	3 - 3rd Bnd.	2 - Equals: Cont. Bnd. #
	.	in
	.	NEWROW
	100 - 100th Bnd.	

PARENT (20, 10)

1 - 1st Parent	1 - Equals Object # of Parent
2 - 2nd Parent	2 - Equals Object # of 1st Offspring
3 - 3rd Parent	3 - Equals Object # of 2nd Offspring
.	.
20 - 20th Paren	10 - Equals Object # of 9th Offspring

BNDTRK (2, 40, 480)

1 - Left Bnd.	1 - 1st Object	1 - Equals Bnd.
2 - Right Bnd.	2 - 2nd Object	Position
	3 - 3rd Object	at Line #
	.	1
	.	.
	40 - 40th Object	.
		480 - Equals Bnd.
		Position
		at Line #
		480

Figure 17. Descriptive Array Elements Used For Boundary Definition
(cont'd)

```

AREA (40                                     )

    1 - Equals Area of 1st Object
    2 - Equals Area of 2nd Object
    3 - Equals Area of 3rd Object
    .
    .
    40 - Equals Area of 40th Object

X MOMNT (40                                 )

    1 - Equals X Coord. of 1st Object
    2 - Equals X Coord. of 2nd Object
    3 - Equals X Coord. of 3rd Object
    .
    .
    40 - Equals X Coord. of 40th Object

Y MOMNT (40                                 )

    1 - Equals Y Coord. of 1st Object
    2 - Equals Y Coord. of 2nd Object
    3 - Equals Y Coord. of 3rd Object
    .
    .
    40 - Equals Y Coord. of 40th Object

```

Figure 18. Descriptive Array Elements Used For Boundary Definition
(cont'd)

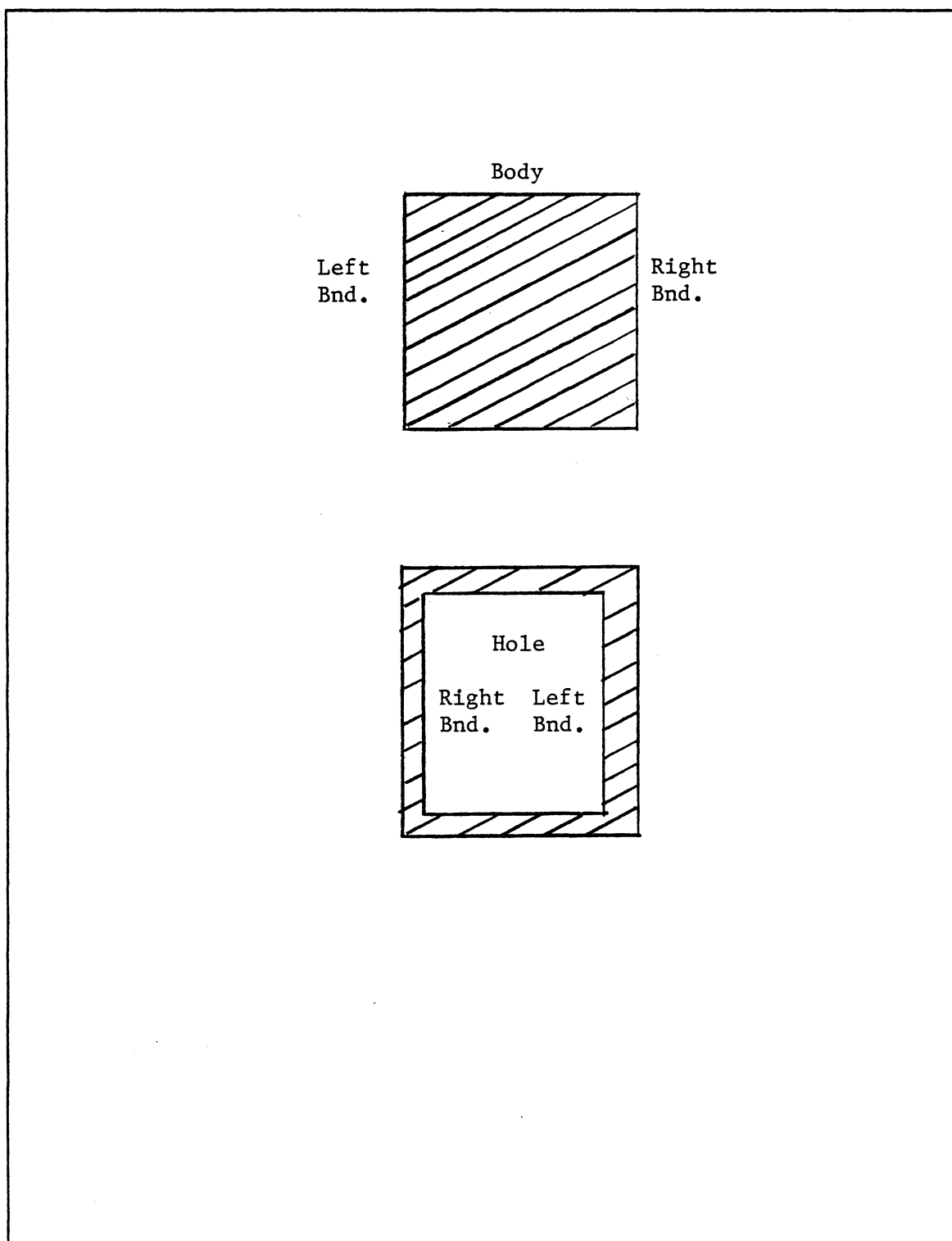
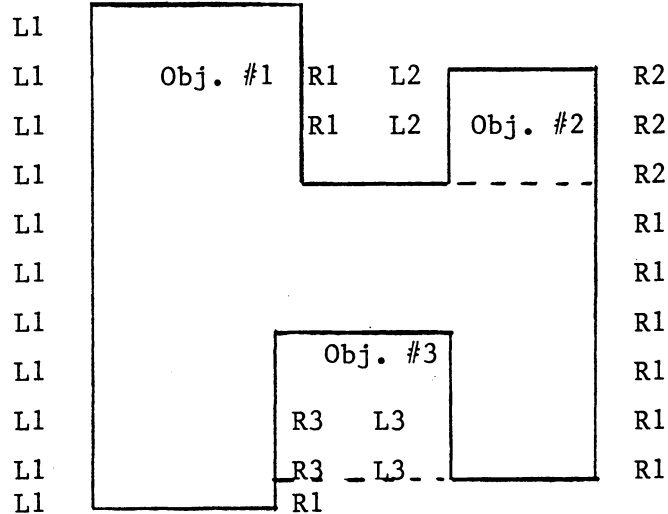


Figure 19. Object Types



<u>Object #</u>	<u>Object Type</u>
1	Body
2	Body
3	Hole

Parent Object = Obj. #1

Offspring Objects = Objs. #2 & #3

L1 = Left Bnd. of Obj. #1
 R1 = Right Bnd. of Obj. #1
 L2 = Left Bnd. of Obj. #2
 R2 = Right Bnd. of Obj. #2
 L3 = Left Bnd. of Obj. #3
 R3 = Right Bnd. of Obj. #3

Figure 20. Parent Object

tified in OLDROW and NEWROW. These equations were developed by Agrawala and Kulkarnu [1].

The birth of an object's boundary, Figure 22, along a scan line is determined by scanning through BTHROW and identifying column positions which contain 0's. Two row elements in a column containing 0's represent the birth of a body, while a right hand column element containing 0 followed by a left hand column element containing 0 and a right hand column element containing 1 represent the birth of a hole.

When a body is born, it is considered its own parent. When a hole is born, the surrounding body is considered its parent.

The termination of an object's boundary, Figure 23, along a scan line is determined by scanning through DTHROW and identifying column positions which contain 0's. Two row elements in a column which contain 0's represent either the termination of a hole, the merging of two body boundaries, or the merging of a body with a surrounding hole. A right hand column element containing 0 followed by a left hand column element containing 0 and a right hand column element containing 1 represent either the termination of a body, the merging of two hole boundaries, or the merging of a hole with a surrounding body.

When two or more bodies merge, the left most body becomes the parent of the other bodies. Likewise, when two or more holes merge, the left most hole replaces the surrounding body as the parent of the other holes. If

a body merges with a surrounding hole, the hole becomes the parent of the body. Finally, when a hole merges with a surrounding body, no parental updating is needed since the body was assigned as the hole's parent when the hole was born. In all of the cases described above, the merging boundary of the parent is always extended to the continuing boundary of the offspring.

The area and centroid of every object within the binary pixel array are determined by computing the incremental area and area moments of an object for one scan line, using the equations shown in Figure 24. The total area of an object is computed by sequentially integrating the incremental area of the object over all of the scan lines in the pixel array. The centroid of an object is computed by sequentially integrating the incremental X area moment and Y area moment of the object over all of the scan lines in the pixel array, and then dividing the values by the object's total area.

Due to the method of image data collection, the length represented by the X axis in the vision system is different from the length represented by the Y axis. The ratio of the X:Y representative lengths is called the "Aspect" ratio. This ratio is taken into account in all computations involving the binary pixel array. The vision system in this research has an Aspect ratio equal to 1.3156. This value was determined experimentally.

All bodies have positive areas, while holes have negative areas. In addition, the total area of a parent object is the sum of the areas of the defined parent object and all of its offspring. Likewise, the centroid of a parent object is computed by adding up the area moments of the defined parent object and all of its offspring and then dividing the values by the parent's total area.

The subroutines used to identify and track object boundaries are described in following sub-sections.

4.1.2.1 DFBND

The vision subroutine, DFBND, is called upon to implement the Boundary Continuation method on the pixel array and to compute the area and centroid of every object identified.

DFBND sequentially reads each row of the binary pixel array by calling VRDROW. Each row is scanned for the black to white and white to black pixel transitions which indicate an object's boundaries. The array, NEWROW, is simultaneously updated.

When a boundary is detected, DEFND begins to copy NEWROW from the previous scan line into OLDROW of the current scan line. In addition, the vision subroutine, BNDSCN, is called to apply the Boundary Continuation criteria

$$\begin{array}{cccc}
 \dot{X}_{R_i}(k-1) & \dot{X}_{L_i}(k) & \dot{X}_{R_i}(k) & \dot{X}_{L_i}(k+1) \\
 \\
 \dot{X}_{R_j}(k-1) & \dot{X}_{L_j}(k) & \dot{X}_{R_j}(k) & \dot{X}_{L_j}(k+1)
 \end{array}$$

Left Boundary: For $X_{L_i}(k)$ to continue to $X_{L_j}(k)$, then

$$X_{R_j}(k-1) < X_{L_i}(k) < X_{R_j}(k), \text{ and (1)}$$

$$X_{R_i}(k-1) < X_{L_j}(k) < X_{R_i}(k). \quad (2)$$

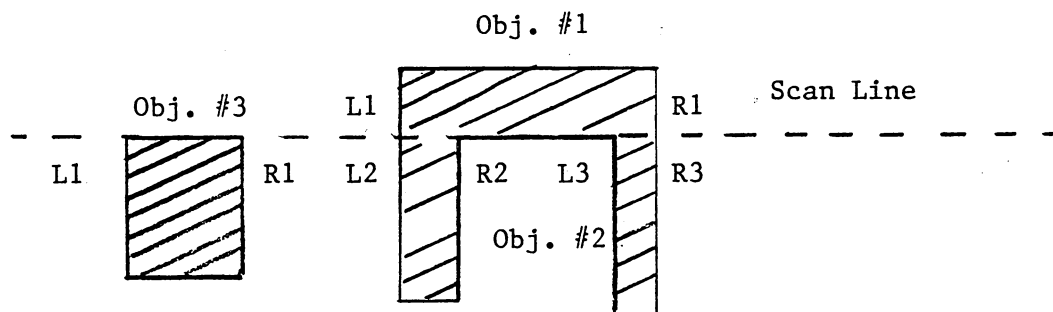
Right Boundary: For $X_{R_i}(k)$ to continue to $X_{R_j}(k)$, then

$$X_{L_j}(k) < X_{R_i}(k) < X_{L_j}(k+1), \text{ and (3)}$$

$$X_{L_i}(k) < X_{R_j}(k) < X_{L_i}(k+1). \quad (4)$$

Where: X_L = Left Boundary, and
 X_R = Right Boundary.

Figure 21. Boundary Continuation Equations



L1 = Left Bnd. #1
 R1 = Right Bnd. #1
 L2 = Left Bnd. #2
 R2 = Right Bnd. #2
 L3 = Left Bnd. #3
 R3 = Right Bnd. #3

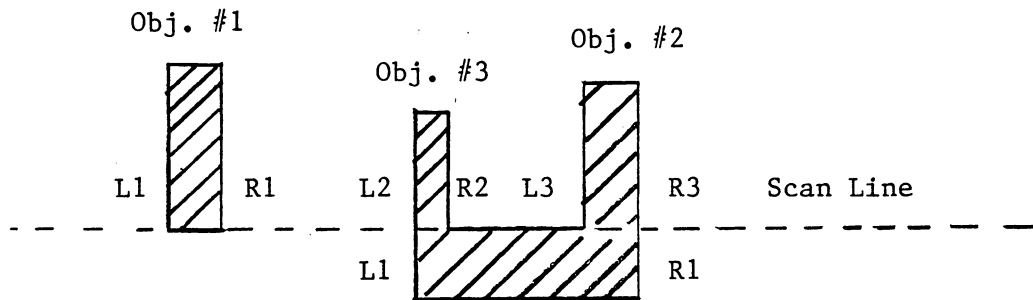
BTHROW (*, *, 1)

Left	0	1	0
Right	0	1	0

BTHROW (*, *, 2)

Left	*	1	*
Right	*	*	1

Figure 22. Boundary Birth



L1 = Left Bnd. #1
 R1 = Right Bnd. #1
 L2 = Left Bnd. #2
 R2 = Right Bnd. #2
 L3 = Left Bnd. #3
 R3 = Right Bnd. #3

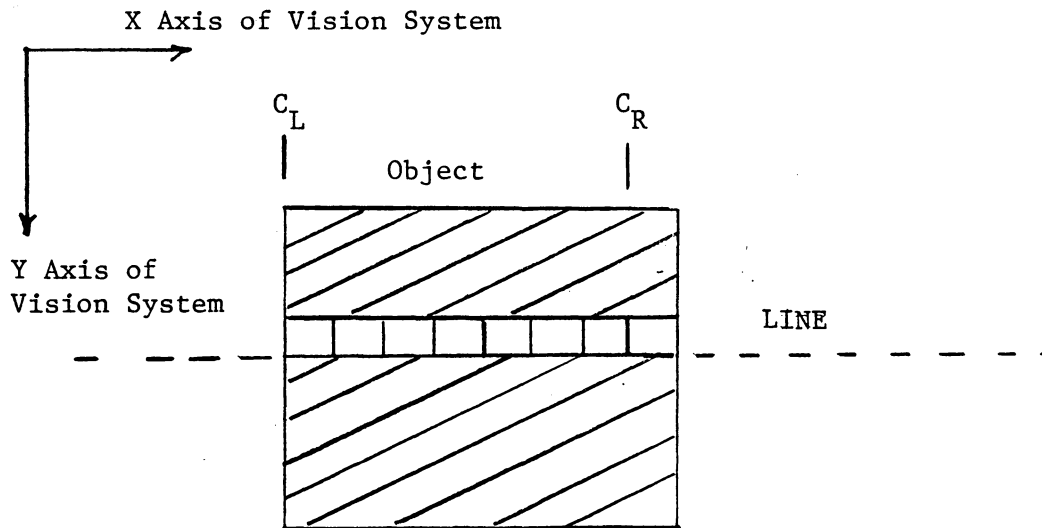
DTHROW (*, *, 1)

Left	0	1	0
Right	0	0	1

DTHROW (*, *, 2)

Left	*	1	*
Right	*	*	1

Figure 23. Boundary Termination



$$IAREA = ((C_R - C_L + 1) \times FLD^2) / (ARATIO \times 512 \times 480) \quad (5)$$

$$IXM = (((C_R + C_L) \times 0.5) - 1) \times FLD / 512 + FLD / (2 \times 512) \quad (6)$$

X IAREA

$$IYM = ((LINE \times FLD) / (ARATIO \times 480) + FLD / (ARATIO \times 2 \times 480)) \quad (7)$$

X IAREA

Where: IAREA = Incremental Area (inches²) of the Object,
 IXM = Incremental X Moment (inches²) of the Object,
 IYM = Incremental Y Moment (inches²) of the Object,
 C_R = Column Number of the Object's Right Boundary,
 C_L = Column Number of the Object's Left Boundary,
 LINE = Line Number of Scan Line,
 FLD = Width of Field of View (inches), and
 ARATIO = Aspect Ratio of Vision System.

Figure 24. Area and Moment Equations

equations to OLDROW and NEWROW to determine which object boundaries continue. NEWROW, BTHROW, DTHROW, and BNDTRK are simultaneously updated.

After DFBND determines which boundaries continue along a scan line, it then calls upon BNDBTH to determine which object boundaries have been born and upon BNDDTH to determine which object boundaries have died. These subroutines update BNDTRK and PARENT. In addition, they also update the descriptive arrays, BNDRTE, LBDTRK, and RBDTRK, which are used in succeeding boundary definition routines and which are discussed in a later section.

The vision subroutine, STSCAN, is called to compute the incremental area and area moments of each object identified in the scan line and to add these values to the object's total area and area moment.

When DEFBND reaches the last line to be scanned, it then terminates all bodies whose boundaries are undefined. In addition, it calls upon the vision subroutine, LCNSRT, to merge all undefined holes with their surrounding bodies.

4.1.2.2 BNDSCN

The vision subroutine, BNDSCN, is called to apply the Boundary Continuation criteria equations to the boundaries identified in OLDROW and NEWROW. By applying these equations, BNDSCN determines which boundaries in NEWROW are continuations of boundaries identified in OLDROW. In addition, the

arrays BTHROW and DTHROW are updated by having their columns, which reflect continuing boundaries, set equal to 1.

4.1.2.3 BNDBTH

The vision subroutine, BNDBTH, is called to determine which boundaries described in BTHROW are newly born. BNDBTH first scans through BTHROW searching for right hand column elements which contain 0 and which are succeeded by a right hand column elements which contain 1 and by a left hand column element which contains 0. This column sequence indicates the birth of a hole.

Upon finding a hole birth, BNDBTH calls upon PARFND to enter the hole's surrounding body as a parent in the array, PARENT, and to enter the hole as an offspring. In addition, the descriptive array, BNRTE, along with other minor descriptive arrays are updated. The right hand and left hand columns containing 0 are subsequently set to 1, and BNDBTH begins its search again from the first column of BTHROW.

After completing its search for hole births, BNDBTH then searches BTHROW for column elements which both contain 0. This indicates the birth of a body. Upon finding a body birth, BNDBTH calls upon PARFND to enter the body as its own parent in PARENT. In addition, the descriptive array, BNRTE, along with other minor descriptive arrays are updated. BNDBTH then continues the search from the subsequent column position.

Upon completing its search for body births, BNDBTH returns program execution to BNDSCH.

4.1.2.4 BNDDTH

The vision subroutine, BNDDTH, is called to determine which boundaries described in DTHROW are newly terminated. BNDDTH first copies DTHROW into the array, BUFROW, since DTHROW is drastically altered during its initial scan. It scans through DTHROW searching for right hand column elements which contain 0 and which are succeeded by a right hand column element which contains 1 and by a left hand column element which contains 0. This column sequence indicates the termination of a hole, the merging of two bodies, or the merging of a body with its surrounding hole.

If the terminating boundary belongs to a body, then BNDDTH scans back toward the first column of DTHROW, searching for the right boundary of a surrounding hole or the continuing left boundary of a body. If the right boundary of the hole is found first, then BNDDTH updates BNDTRK and calls upon PARFND to assign the hole to the array, PARENT and to assign the merging body as its offspring. In addition, any bodies in between the hole's right boundary and the body's left boundary, which have both right and left merging boundaries, are also assigned as offspring to the hole. These bodies are referred to as terminating bodies. Finally the right boundary of the original merging body is reassigned as the right boundary of the merging hole.

If the continuing left boundary of a body is found first, then BNDDTH updates BNDTRK, extends the right boundary of the continuing body to the right boundary of the merging body, and calls upon PARFND to assign all terminating bodies as offspring to the body with the continuing left boundary.

In addition to the parental/offspring assignment described above, BNDDTH also updates a group of minor descriptive arrays, and sets all columns in DTHROW that are associated with the terminating bodies equal to 1. It also calls upon the vision subroutines, RBDINT, LBDINT, and RTECHG to update the arrays, RBDTRK, LBDTRK, and BNDRTE.

If the terminating boundary belongs to a hole, then BNDDTH scans back toward the first column in DTHROW, searching for the right boundary of the hole, the extending right boundary of a body, or the continuing left boundary of a body.

If the first boundary found is the right boundary of the same hole, then BNDDTH simply terminates the hole, and updates the associated descriptive arrays. If the first boundary found is the extending right boundary of a body, then BNDDTH terminates the body, extends the left boundary of the hole to the left boundary of the terminated body, and updates the associated descriptive arrays. If the first boundary found is the continuing left boundary of a body, then BNDDTH terminates the body and all terminating bodies in between, extends the left boundary of the hole to the

left boundary of the terminated body, and updates the associated descriptive arrays.

After BNDDTH updates the descriptive arrays associated with one set of object merges or terminations, it then resumes its scan of DTHROW from the first column. After BNDDTH has completed its search of DTHROW for the array patterns described above, it then modifies BUFROW in order to mask out the object boundaries which have all ready been processed.

BUFROW is then scanned and the boundary positions of each body described by the array are recorded. Each body identified by the previous scan is checked for simple body termination, which is indicated by a 0 in the body's left and right column elements. If a simple body termination is found, then the associated descriptive arrays are updated.

BNDDTH then makes two double scans within the boundaries of all non-simple-terminating bodies.

In the initial double scan, BNDDTH searches for either the merging of the left side of the object with the right side of a hole or the merging of two or more holes. In either case, upon finding an object merge, BNDDTH extends the boundary of the parent object, updates the associated descriptive arrays and continues its search until there are no other similar merges to be found. The vision subroutines, XXSCH, OBCHK, RX2SCH, BBCHK, X2SCH, and HLETRM are utilized in the initial double scan.

In the second double scan, BNDDTH determines whether the right boundary of the body merges with the right boundary of a hole. If a merging exists, then the right boundary of the body is extended to the right boundary of the merging hole and the associated descriptive arrays are updated. The vision subroutines, RX2SCH, OBCHK, BBCHK, and HLETRM are utilized in the second double scan.

After processing every body in BUFROW, program execution is returned to DEFBND.

4.1.2.5 Supporting Subroutines

This sub-section briefly describes each of the subroutines which support DEFBND, BNDSCH, BNDBTH, and BNDDTH.

PARFND is called to enter a parent object and its offspring into the array, PARENT. PARFND first determines if the parent object is entered in the first column of PARENT. If it is not, then PARFND enters the parent object in the first column and enters the offspring in the subsequent column in the same row. If it is, then PARFND simply enters the offspring in the subsequent column.

XXSCH is called to search through BUFROW for a left hand column element equal to 1 and a right hand column element greater than or equal to 1. The search is conducted from the column number stored in the variable,

MARK5, to the last column. If the array pattern is found, then the variable, XXFLG, is set equal to 1 and MARK5 indicates the column position.

RX2SCH is called to search through BUFROW for a left hand column element greater than or equal to 1 and a right hand column element equal to 1. The search is conducted from the column number stored in MARK5 back to the first column. If the array pattern is found, then the variable, RX2FLG, is set equal to 1 and MARK5 indicates the column position.

LX2SCH is called to conduct a search through BUFROW for a left hand column element equal to 1 and a right hand column element greater than or equal to 1 or a left hand and a right hand column element that are equal to 0. The search is conducted from the column number stored in MARK5 to the last column. If either array pattern is found, then the variable, LX2FLG, is set equal to 1 and MARK5 indicates the column position.

OBCHK is called to check whether the column number stored in MARK5 represents a body boundary in BUFROW. OBCHK compares the column number with column numbers recorded as body boundaries. If a match is found, then the variable, OBFLG, is set equal to 1.

BBCHK is called to search through BUFROW for a left hand and right column element equal to 0. The search conducted between the column numbers stored in MARK5 and the variable, MARK2. If the array pattern is found, then the variable, BBFLG, is set equal to 1 and the variable, MARK4, indicates the column position.

HLETRM is called to terminate a series of merging holes and to assign them to their parent object in PARENT. If the variable, HLEFLG, is equal to 1 upon entry into the subroutine, then HLETRM determines whether the parent object is entered in the first column of PARENT. If it is not, then the object is entered into the first column.

If HLEFLG is equal to 0 upon entry into the subroutine, then HLETRM terminates all holes described in BUFROW between the column numbers stored in the variables, MARK6 and MARK7. Each merging hole is assigned as an offspring to the parent object in PARENT. The associated descriptive arrays are updated and the subroutines, RBDINT, LBDINT, and RTECHG are called to update the arrays, RBDTRK, LBDTRK, and BNDRTE.

If HLEFLG is equal to 1, then all of the operations described previously are conducted. In addition, the subroutine, PARCHG, is called to find the original parent of each terminating hole and to eliminate the hole from the parent's group of offspring.

PARCHG finds the original parent of each terminating hole by scanning through the first column of PARENT. When the parent is found, PARCHG proceeds to scan through the row for the position of the terminating hole. When the hole's column position is found, then every succeeding offspring is set back one column position. This results in the elimination of the hole from the parent's group of offspring.

STSCAN is called to compute the incremental area and incremental X and Y moments of each undefined object and to add these values to the object's total area and X and Y moments. STSCAN uses the equations shown in Figure 24 in its computations.

FNSTAT is called to compute the total area and centroid of each parent object. FNSTAT first scans through PARENT and adds each offspring's area, X area moment, and Y area moment to its parent's area and X and Y moments. It then computes each object's centroid by dividing the object's total X and Y moments by its total area. Upon exit, AREA, XMOMNT, and YMOMNT have been updated.

4.2 OBJECT BOUNDARY DEFINITION

This segment of software uses the Boundary Tracking method to identify and locate the corner points of every object defined in the binary pixel array. The Boundary Tracking method involves two operations. The first operation maps a route around an object's boundary. The subsequent operation uses the mapped route to scan the object's boundary. During this scan, a vector analysis is performed on points along the boundary to determine the existence of corner points. This analysis assumes that the object has straight edges and that the positive angle between sides is greater than 90 degrees.

4.2.1 Boundary Mapping

The mapping of an object's boundary involves the creation of the following descriptive arrays:

1. LBDTRK,
2. RBDTRK,
3. BNDRTE, and
4. ROUTE.

LBDTRK, shown in Figure 25, describes a group of parent objects and their offspring, whose right boundaries merged with the parent objects. RBDTRK describes a group of parent objects and their offspring, whose left boundaries merged with the parent objects. BNDRTE describes the scanning direction, clockwise or counter clockwise, around an object's boundary; and ROUTE describes the complete route schedule for navigating around an object's boundary.

The clockwise scanning, Figure 27, of an object indicates that the left boundary is scanned in an upwards direction while the right boundary is scanned in a downwards direction. The counter clockwise scanning of an object indicates that the left boundary is scanned in a downwards direction while the right boundary is scanned in an upwards direction.

A body is normally scanned in a counter clockwise direction and a hole is normally scanned in a clockwise direction. However, when a object

LBDTRK (20, 10, 2)

1 - 1st Parent	1 - Parent	1 - Equals
2 - 2nd Parent	2 - 1st Offspring	Obj. #
3 - 3rd Parent	3 - 2nd Offspring	2 - Equals
.	.	Line #
.	.	of
20 - 20th Parent	10 - 9th Offspring	Merge

RBDTRK (20, 10, 2)

1 - 1st Parent	1 - Parent	1 - Equals
2 - 2nd Parent	2 - 1st Offspring	Obj. #
3 - 3rd Parent	3 - 2nd Offspring	2 - Equals
.	.	Line #
.	.	of
20 - 20th Parent	10 - 9th Offspring	Merge

BNDRTE (40)

1 - 1st Object's Route Dir.	- Equals:	0 - CCW
		1 - CW
2 - 2nd Object's Route Dir.		
3 - 3rd Object's Route Dir.		
.		
.		
40 - 40th Object's Route Dir.		

ROUTE (200, 5)

1 - 1st Scan	1 - Equals Start.	Line #
2 - 2nd Scan	2 - Equals Stop.	Line #
3 - 3rd Scan	3 - Equals Scan Dir.:	0 - Up
.		1 - Down
.	4 - Equals Object #	
200 - 200th Scan	5 - Equals Boundary:	0 - Left
		1 - Right

Figure 25. Descriptive Array Elements Used for Boundary Tracking

PNTTRK (20, 250, 2)

1 - 1st Obj.	1 - 1st Point	1 - Equals X
2 - 2nd Obj.	2 - 2nd Point	Coord.
3 - 3rd Obj.	3 - 3rd Point	2 - Equals Y
.	.	Coord.

20 - 20th Obj. 250 - 250th Point

SDSTAT (20, 50)

1 - 1st Parent	1 - Equals Len. of 1st Side
2 - 2nd Parent	2 - Equals Len. of 2nd Side
3 - 3rd Parent	3 - Equals Len. of 3rd Side
.	.

20 - 20th Parent 50 - Equals Len. of 50th Side

PNSTAT (20, 50)

1 - 1st Parent	1 - Dist. of Cent. from 1st Point
2 - 2nd Parent	2 - Dist. of Cent. from 2nd Point
3 - 3rd Parent	3 - Dist. of Cent. from 3rd Point
.	.

20 - 20th Parent 50 - Dist. of Cent. from 50th Point

PERIM (20)

1 - Equals Perimeter of 1st Parent Object
2 - Equals Perimeter of 2nd Parent Object
3 - Equals Perimeter of 3rd Parent Object
.

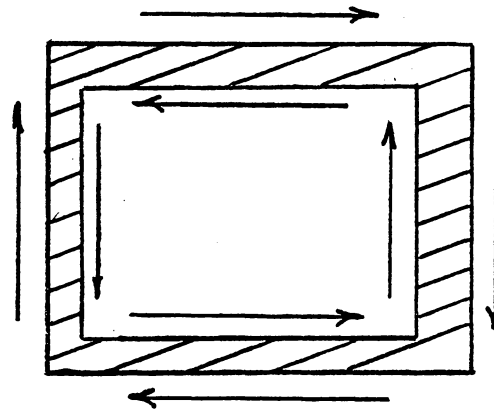
20 - Equals Perimeter of 20th Parent Object

Figure 26. Descriptive Array Elements Used for Boundary Tracking
(cont'd)

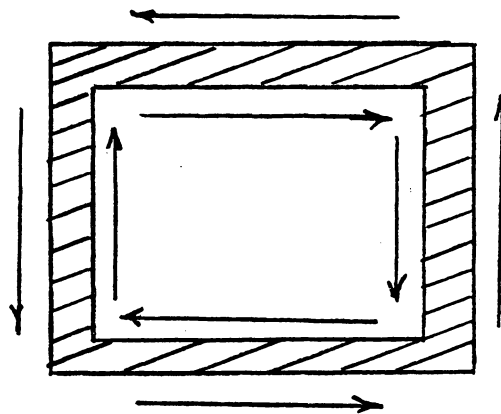
merges with a surrounding parent object, then that object can expect to be scanned in the opposite direction since it is the parent's normal scanning direction which dominates. In addition, all of the merging object's offspring can expect to be scanned in the opposite direction since their parent's scan direction has changed.

The development of an object's routing map, Figure 28, involves sequentially scanning its offspring in LBDTRK and RBDTRK, and determining at which lines these offspring were merged with. However, LBDTRK and RBDTRK are created during the boundary identification phase and the offspring listed in their rows are entered in the order in which they merged and not in the order in which they are to be scanned. To correct this, the offspring in LBDTRK and RBDTRK are rearranged so that they reflect the order in which they are to be scanned.

The manner in which the offspring are rearranged is the following. Each parent in LBDTRK is checked for object type and scan direction. If the parent is body whose scan direction is clockwise, then its offspring are arranged in reverse order. If the parent is a hole whose scan direction is counter clockwise, then its offspring are also arranged in reverse order. If the parent is a hole whose scan direction is clockwise, then its offspring are arranged in reverse order, and all hole offspring which merged at the same line are arranged in reverse group order.



Clockwise Boundary Scan



Counter Clockwise Boundary Scan

Figure 27. Clockwise and Counter Clockwise Scanning

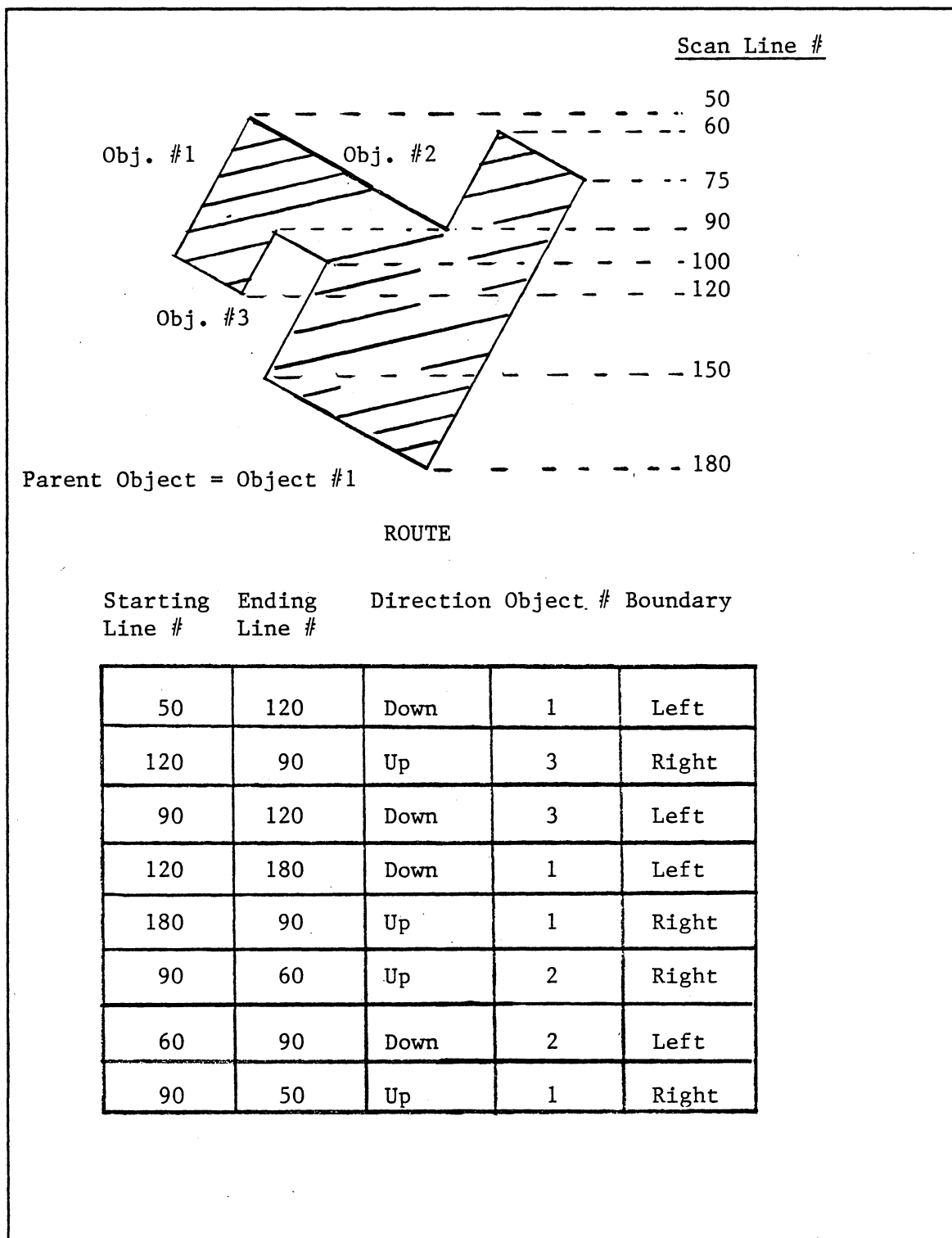


Figure 28. Boundary Routing Example

Subsequently, each parent in RBDTRK is checked for object type and scan direction. If the parent is a hole whose scan direction is counter clockwise, then its offspring are arranged in reverse order. If the parent is a body whose scan direction is clockwise, then its offspring are also arranged in reverse order. If the parent is a body whose scan direction is counter clockwise, then its offspring are arranged in reverse order, and all body offspring which merged at the same line are arranged in reverse group order.

The sequential route map of an object is stored in ROUTE, which contains an ordered list of scan directions. Each scan map contains the following information:

1. The line at which to begin the scan,
2. The line at which to end the scan,
3. The direction of the scan,
4. The object to be scanned, and
5. The boundary to be scanned.

The actual construction of ROUTE revolves around the following rules:

1. An object is not completely mapped until both its left and right boundaries have been accounted for, and
2. One boundary must be completely mapped before mapping of the other boundary can begin.

For example, a body scanned in the counter clockwise direction must have its left boundary completely mapped before its right boundary can be mapped.

To construct ROUTE for any object involves the use of a stack array called STACK. Stack is used to store the object number and line number of a parent boundary during the mapping of a merging object's boundary. When the offspring has been mapped, the information is popped of the stack, so that the parent boundary can be mapped from the point of entry. A stack array is an ideal data structure for this job, since there can be many levels of object merges while mapping a single boundary.

Scan maps are developed by starting at the beginning of a parent's boundary and performing the following operations:

1. Determining the object to be scanned,
2. Searching through either LBDTRK or RBDTRK (depending upon which boundary is to be scanned) for the first merging boundary,
3. Recording the route parameters into a scan map,
4. Pushing the object number and exit line number on to STACK,
5. Determining the merging object to be scanned,
6. Searching through either LBDTRK or RBDTRK (depending upon which boundary is to be scanned) for its first merging boundary,
7. If no merging boundary is found, then recording the information for the first boundary into a scan map, and doing (6) for the next boundary,

8. If a merging boundary is found then doing (3) for the boundary,
9. After mapping the second boundary of the merging object, popping the object number and exit line number of the parent boundary,
10. Following operation sequence, 1 through 9 until the first parent boundary is completely mapped, and
11. Following operation sequence, 1 through 9 until the second parent boundary is mapped.

The subroutines used to construct ROUTE and the other descriptive arrays are discussed in the following sub-sections.

4.2.1.1 LBDINT and RBDINT

The vision subroutines, LBDINT and RBDINT, are called by BNDDTH and HLETRM to enter a parent object and its offspring into LBDTRK and RBDTRK respectively.

Both routines search the first column of their respective arrays for the parent object listed in the variable, FLAG1. If the object is not found, then the routines enter the parent object into the array and then enter the offspring listed in the variable, FLAG2, into the same row at the appropriate column.

4.2.1.2 RTECHG

The vision subroutine, RTECHG, is called upon by BNDDTH and HLETRM to compare the scan direction of a merging offspring to its parent's, and to alter the scan direction of the merging offspring and its offspring if they do not match.

RTECHG first retrieves the scan direction of the parent object (listed in the variable, FLAG1) and the merging object (listed in the variable, FLAG2) and compares the two. If they differ, RTECHG then searches the first column of LBDTRK for the merging object. When it is found, RTECHG proceeds to change the scan direction of each offspring listed. The same procedure is subsequently performed on RBDTRK.

4.2.1.3 RERTE

The vision subroutine RERTE is called upon by PNTDEF (discussed in a later section) to rearrange the offspring in LBDTRK and RBDTRK so that they reflect the order in which they are to be scanned.

RERTE first scans through the first column of LBDTRK and checks the type and scan direction of the parent object. If the parent is a body and its scan direction is clockwise, then the vision subroutine, CSWTCH, is called to arrange the offspring in reverse order. If the parent is a hole and its scan direction is counter clockwise, then CSWTCH is also called. If the parent is a hole and its scan direction is clockwise, then CSWTCH is

called and the vision subroutine, PSWTCH, is called to arrange hole offspring with the same merge line number in reverse group order.

RERTE subsequently checks each of the parents in RBDTRK. If the parent is a hole and its scan direction is counter clockwise, then CSWTCH is called. If the parent is a body, and its scan direction is clockwise, then CSWTCH is called also. If the parent is a body and its scan direction is counter clockwise, then CSWTCH and PSWTCH are called.

4.2.1.4 CSWTCH

The vision subroutine, CSWTCH, is called upon to arrange the offspring in either LBDTRK or RBDTRK in reverse order. If FLAG1 is equal to 0, then CSWTCH operates on LBDTRK. If FLAG1 is equal to 1, then CSWTCH operates on RBDTRK.

CSWTCH first goes to the row position listed in the variable, PARPNT, in either LBDTRK or RBDTRK. It then copies the row's offspring (in reverse order) into the array, CHGROW. It subsequently copies the offspring back into the row.

4.2.1.5 PSWTCH

The vision subroutine, PSWTCH, is called upon to either arrange groups of holes (with the same merge line number) in reverse order in LBDTRK or groups of bodies (with the same merge line number) in reverse order

in RBDTRK. If FLAG1 is equal to 0, then PSWTCH operates on LBDTRK. If FLAG2 is equal to 1, then PSWTCH operates on RBDTRK.

PSWTCH first goes to the row listed in PARPNT in either LBDTRK or RBDTRK. It then scans the row for groups of offspring with the same merge line numbers. If a group is found, PSWTCH copies the offspring (in reverse order) into CHGROW, and then copies the offspring back into the group's original position. This procedure is conducted until all groups in the row have been processed.

4.2.1.6 RTEPLN

The vision subroutine, RTEPLN, is called upon to construct the map array, ROUTE, for the object listed in the variable, MARK1. RTEPLN utilizes the stack array, STACK, to implement the scan map construction procedure discussed previously. Upon exit, ROUTE contains the scan maps needed to completely navigate the boundary of the object. In addition, the variable, RTEPNT, marks the row number of the last scan map in ROUTE.

4.2.2 Corner Point Definition

The second phase of the Boundary Tracking method creates the following descriptive arrays:

1. PNTRK,
2. SDSTAT,
3. PNSTAT, and

4. PERIM.

PNTTRK describes the X and Y coordinates of every object corner point described in the binary pixel array. SDSTAT lists the length of every side for each parent object. PNSTAT lists the distance between an object's centroid and everyone of its corner points; and PERIM lists the perimeter of every parent object.

Using the array, ROUTE, as a map, this method scans the boundary of an object and checks for the existence of corner points. Corner points are determined by selecting three points along the object's boundary. These points form two boundary vectors on which a vector analysis is performed. If the vectors meet any of the corner point criteria, then the second of the three points is considered to be a corner point.

The corner point criteria, Figure 29, are the following:

1. The positive angle between the two boundary vectors is greater than 30 degrees,
2. The X component of each boundary vector has a different sign, and
3. The Y component of each boundary vector has a different sign.

After applying the corner point criteria to a set of points, a new set of points is obtained by incrementing one point along the boundary. Thus the second point becomes the first point. The third point becomes the

second point; and the succeeding point becomes the third point. The boundary point at the beginning of a scan is always considered to be the first corner point.

The number of points selected as corner points by the criteria is a function of the distance between each point in a set and the roughness of the boundary. The greater the distance between each point, the less the number of points accepted as corner points. The greater the roughness of the boundary, the greater the number of points accepted as corner points.

The distance between points is controlled by selecting the number of rows between each boundary point. In this research, four rows are used. Roughness indicates how serrated an object's boundary appears in a binary image. The roughness of an object's boundary is dependent upon the object's orientation. The more parallel the boundary is to an axis of the field of view, the rougher the boundary.

In any event, the corner point criteria ensure that all of the corner points of an object are recorded in PNTTRK. The next task is to scan through PNTTRK and to discard those points that are false corner points. To do this, two filtering techniques are employed back to back.

The first filtering technique applies the corner point criteria. Three consecutive points are selected from an object's row in PNTTRK. A vector analysis is then conducted on the two vectors formed by the points.

If the vectors do not meet any of the corner point criteria, then the second point is not considered a true corner point and it is discarded. The third point then becomes the second point,; and the succeeding point becomes the third point.

If the vectors pass the corner point criteria, the second point is considered to be a true corner point. It then becomes the first point; and the third and succeeding points become the second and third points respectively. The analysis is conducted until all points have been checked.

The first filtering technique effectively filters out false corner points along the sides of an object. However it sometimes fails to eliminate points that are near true corner points. To eliminate these points, a second filtering technique is employed.

This technique checks the distance between successive points in PNTTRK. If the distance is less than a threshold value (0.3 inches), then the second point is considered a false corner point and is discarded. The succeeding point then becomes the second point and the distance between it and the first point is checked.

If the distance exceeds the threshold value, then the second point is considered to be a true corner point. It then becomes the first point, and the distance between it and the succeeding point is checked. This analysis is conducted until all points are compared.

After object's corner points are entered in PNTTRK and filtered, the lengths of its sides, the distances between its centroid and its corner points, and its perimeter are computed and entered into SDSTAT, PNSTAT, and PERIM. The subroutines used to implement the second phase of the Boundary Tracking method are discussed in the following sub-sections.

4.2.2.1 PNTDEF

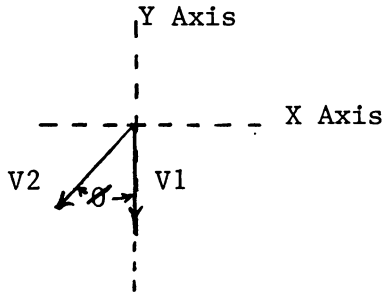
The vision subroutine, PNTDEF, scans the boundary of every object identified in the binary pixel array and updates PNTTRK, SDSTAT, PNSTAT, and PERIM.

PNTDEF first calls upon the vision subroutine, RERTE, to rearrange the offspring in LBDTRK and RBDTRK. It then sequentially scans through the array, BDOBJ, for parent objects. Each parent object is processed in the following manner.

PNTDEF calls upon ROUTE to develop the scanning map of the object. Using the parameters outlined in each scan map of ROUTE, it then scans the boundary of the object. Three points, each four rows apart, are selected from the boundary. Two boundary vectors are formed from these points and are subsequently subjected to the corner point criteria.

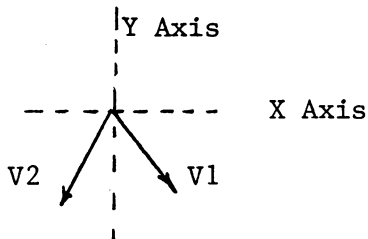
If the vectors pass anyone of the criteria, then the second point of the set is recorded into the object's row in PNTTRK. After the analysis, the set of points is incremented by one point.

Corner Point Criteria



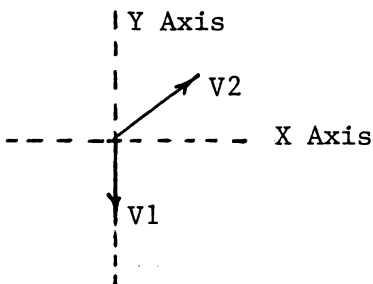
(1)

$$|\theta| \geq 30 \text{ degrees}$$



(2)

$$\begin{aligned} &V1_x > 0, \text{ and} \\ &V2_x < 0, \text{ or} \\ &V1_x < 0, \text{ and} \\ &V2_x > 0 \end{aligned}$$



(3)

$$\begin{aligned} &V1_y > 0, \text{ and} \\ &V2_y < 0, \text{ or} \\ &V1_y < 0, \text{ and} \\ &V2_y > 0 \end{aligned}$$

V1 = Bnd. Vector #1
V2 = Bnd. Vector #2
 θ = Positive Angle
Between V1 and
V2

$V1_x$ = X Component of V1
 $V1_y$ = Y Component of V1
 $V2_x$ = X Component of V2
 $V2_y$ = Y Component of V2

Figure 29. Boundary Vector Analysis

After PNTDEF has completely scanned the object, it then calls upon the vision subroutine, CLNUP2, to apply the first filtering technique to the points in PNTTRK. It then calls upon the vision subroutine, CLNUP1, to apply the second filtering technique.

Once the corner points in PNTTRK have been completely filtered, PNTDEF then calls upon the vision subroutine, RTESTT, to compute the lengths of the object's sides, the distances between its centroid and its corner points, and its perimeter, and to enter these values into SDSTAT, PNSTAT, and PERIM respectively.

After processing each parent object, PNTDEF returns execution to the calling routine.

4.2.2.2 CLNUP1 and CLNUP2

The vision subroutine, CLNUP2, applies the first filtering technique to the row in PNTTRK indicated by the variable, MARK1. The vision subroutine, CLNUP1, applies the second filtering technique to the row in PNTTRK indicated by MARK1.

4.2.2.3 RTESTT

The vision subroutine, RTESTT, updates the arrays, SDSTAT, PNSTAT, and PERIM for the row in PNTTRK indicated by MARK1. RTESTT first scans the row in PNTTRK and computes the distance between consecutive corner points and enters the values in their corresponding columns in SDSTAT. It si-

multaneously adds each value to the corresponding perimeter in PERIM. Upon completion of the scan both SDSTAT and PERIM are updated.

RTESTT then conducts a second scan of the row and computes the distance between each corner point and the object's centroid. Each value is entered into the corresponding column in PNSTAT. Upon completion of the scan, PNSTAT is updated.

4.3 CIRCUIT BOARD IDENTIFICATION AND ORIENTATION

This segment of software checks each parent object and determines whether the object is an "A" type or "B" type fixture. If a fixture is identified, the software then determines the location of three foundation corner points. Upon exit from this code, the arrays ATRK and BTRK, Figure 30, are updated.

ATRK lists the row numbers in PNTRK which represent "A" type fixtures and the column numbers which hold the location of each of the three foundation corner points. BTRK holds the same information for "B" type fixtures.

The descriptive parameters used to identify fixtures are area, number of sides, and perimeter. Each fixture type has its own descriptive data base. "A" type fixtures must have six sides and an area ranging from 23.5 inches² to 26.5 inches² and a perimeter ranging from 18.5 inches to 22.5 inches. "B" type fixtures must have six sides and an area ranging

ATRK (4,	4	)
1 - 1ST "A" Type Obj.	1 - Equals Row Position in	
.	PNTTRK	
.	2 - Equals Col. Position of	
	1st Corner Point in	
	PNTTRK	
4 - 4th "A" Type Obj.	3 - Equals Col. Position of	
	2nd Corner Point in	
	PNTTRK	
	4 - Equals Col. Position of	
	3rd Corner Point in	
	PNTTRK	
BTRK (4,	4	)
1 - 1ST "B" Type Obj.	1 - Equals Row Position in	
.	PNTTRK	
.	2 - Equals Col. Position of	
	1st Corner Point in	
	PNTTRK	
4 - 4th "B" Type Obj.	3 - Equals Col. Position of	
	2nd Corner Point in	
	PNTTRK	
	4 - Equals Col. Position of	
	3rd Corner Point in	
	PNTTRK	

Figure 30. Descriptive Arrays Used for Circuit Board Positioning

from 36.5 inches² to 41.5 inches² and a perimeter ranging from 24.0 inches to 29.5 inches

The procedure for identifying fixtures is to obtain each parent object's area, side number, and perimeter from AREA, PNTNUM, and PERIM; and to sequentially compare them to the descriptive data bases described above.

After identifying a fixture, the next task is to determine its location and orientation. The locations of an object's corner points are stored in PNTTRK. Thus the task is reduced to determining the correspondence between the corner points in PNTTRK and the corner points of the fixture.

If a fixture has a designated top surface and a corner point that is unique among the rest, then each of the fixture's corner points can be sequentially numbered, as shown in Figure 31. Thus if the unique corner point can be identified in PNTTRK, all other corner points can be identified by their relative column positions from that point.

The fixtures designed in this research both have unique corner points. Each unique corner point is designated point #1. What makes these corner points unique is that they are the closest to the centroids of their respective fixtures. Thus point #1 for each fixture is identified in PNTTRK by checking the centroidal distance of each corner point (extracted from PNSTAT) and tagging the one with the smallest value. The three foundation points, point #2, point #3, and point #4 are located by their relative column positions in PNTTRK.

The vision subroutines, ASEROR and BSEROR, are called to identify and orientate "A" type and "B" type fixtures respectively. Both routines work in an identical fashion.

Each subroutine sequentially extracts every parent object's area, side number, and perimeter from AREA, PNTNUM, and PERIM and compares it to their respective fixture's data base. If a match exists, then the row number of the object in PNTTRK is entered into the first column of either ATRK or BTRK.

Each subroutine then checks the centroidal distance of each corner point listed in PNTTRK. The column number of the point with the smallest distance is recorded in the variable, MARK3. Using the sequential relationship described earlier, the column position of each of the foundation corner points is then recorded in columns 2 through 4 of either ATRK or BTRK.

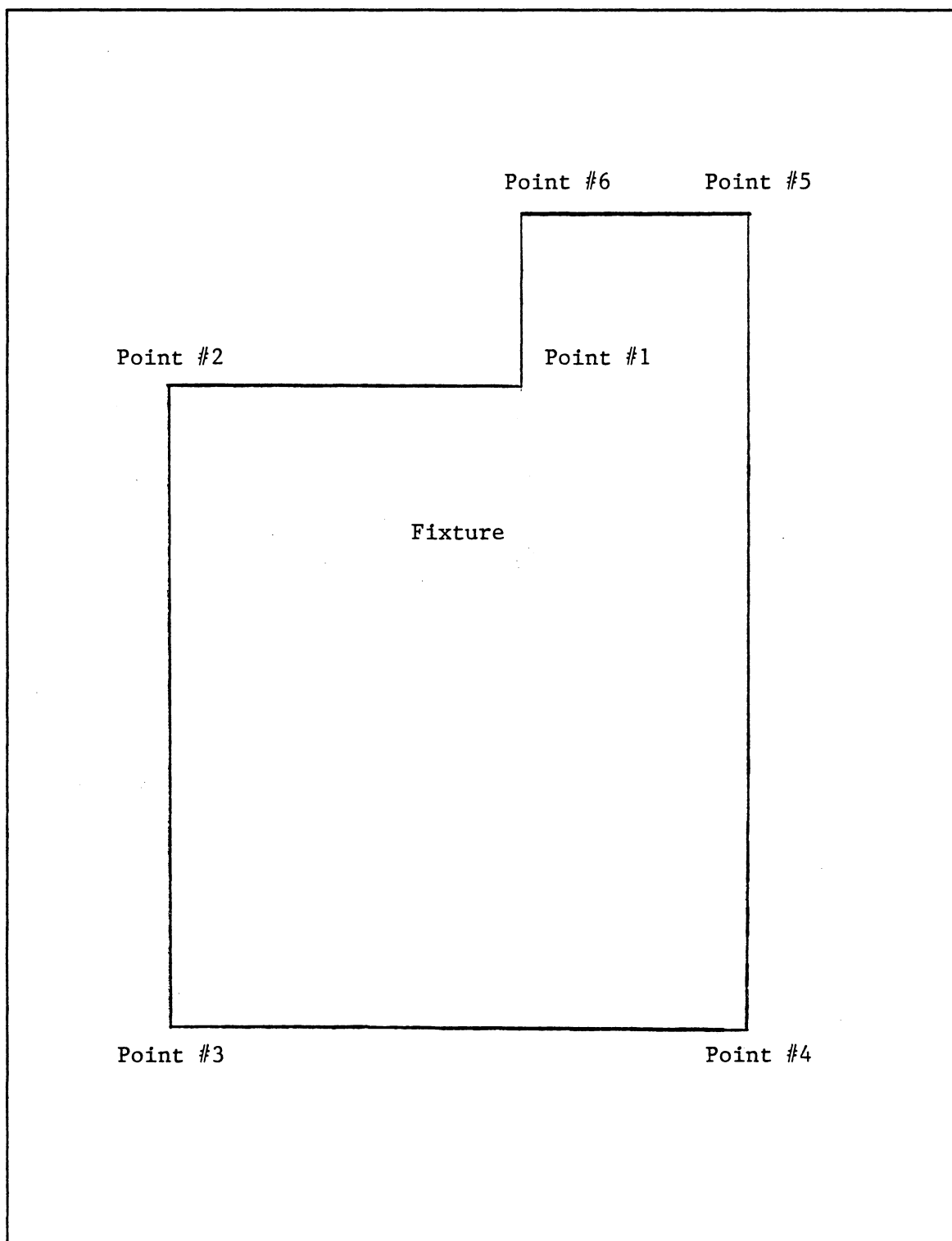


Figure 31. Corner Point Ordering

5.0 TACTILE SYSTEM SOFTWARE LIBRARY

The tactile system software library is comprised of the two following segments:

1. Tactile Probe Calibration, and
2. Tactile Search and Region Definition.

This chapter discusses the theory and software associated with each segment. All software can be referred to in Appendix C.

5.1 TACTILE PROBE CALIBRATION

The tactile probe has an inherent eccentricity with respect to the Z axis of the robot. This eccentricity is due to warping in the needle, eccentricity in the hole in the face plate, and misalignment between the mating surfaces of the face plate and the lower body interface.

To overcome the potential inaccuracy of the eccentric probe, error compensation equations are created to relate the eccentricity of the probe, the vector along which the probe is moving, the rotation of the tactile probe, and the true point of contact. The equations compute the error compensation vector to be added to the point of contact, obtained during a tactile search. The equations are based on the probe being oriented at 0 degrees.

The error compensation equations, Figure 32, are created by performing the following operations:

1. Moving the tactile probe along a vector normal to a known surface (with the probe rotated at 0 degrees) and recording the position of contact,
2. Moving the probe along the same vector (with the probe rotated at 180 degrees) and recording the position of contact,
3. Moving the probe along a vector normal to a surface that is normal to the original surface (with the probe rotated at 0 degrees) and recording the position of contact,
4. Moving the probe along the same vector (with the probe rotated at 180 degrees) and recording the position of contact, and
5. Using the data to compute the magnitude and direction of the probe's eccentricity.

In addition to the eccentricity of the tactile probe, the radius of the tactile probe at the point of contact is taken into account. A quality of the tactile search routine used in this research is the fact that a search is not terminated until the very tip of the tactile probe makes contact with the fixture. Thus, the radius of the probe at the point of contact is essentially constant, reducing any error due to tool point geometry.

The software developed to calibrate the tactile probe reside in both IBM AT and the IBM 7545 controller and have a hierarchical relationship. The

software in the IBM AT acts as the master system. This software computes the points along a vector, down loads the points to the slave program in controller memory, commands the slave program to execute, and accepts the point of contact from the slave program upon completion of its execution.

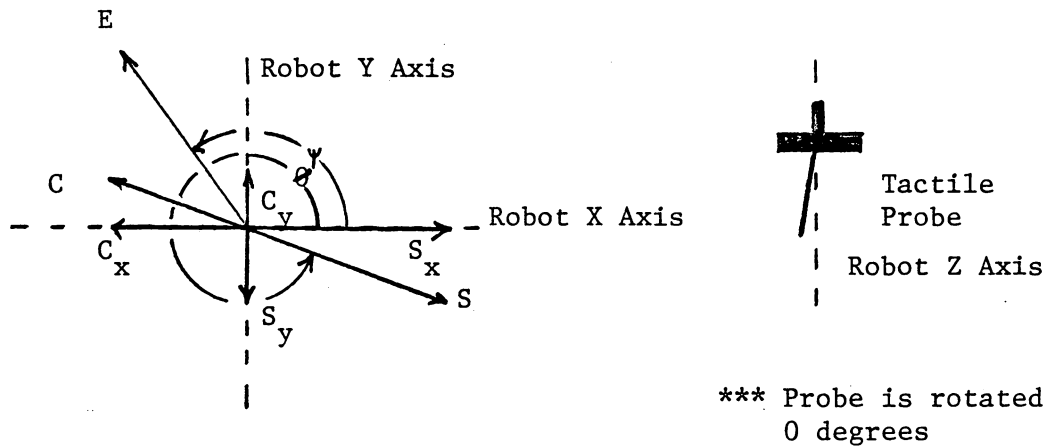
The software in the IBM 7545 controller acts as the slave program. This software accepts a group of points from the master program in the IBM AT and commands the robot to move to each point until contact is made with an object.

The tactile subroutines used to calibrate the tactile probe are discussed in the following sub-sections.

5.1.1 ERCALC

The tactile subroutine, ERCALC, is called upon to develop the error compensation parameters for the tactile probe. This routine was written in FORTRAN and resides in the IBM AT.

ERCALC first calls upon GRAPH1 to inform the user of the calibration operation. It then calls upon EXEC to command the controller to return the robot to the home position and to activate the slave program, TTOUCH. It subsequently calls upon the subroutine, TOUCH, to locate two points along the surface of the calibration block. To utilize TOUCH, ERCALC passes down the starting point of the search, the path vector, and the R and Z coordinates of the robot.



$$|C| = |E| \cos (\theta - \psi) \quad (8)$$

$$|C_x| = |C| \times |S_x| \quad (9)$$

$$|C_y| = |C| \times |S_y| \quad (10)$$

Where: C = Error Compensation Vector,
 E = Probe Eccentricity Vector,
 S = Unit Search Vector,
 C_x = X Component of C,
 C_y = Y Component of C,
 S_x = X Component of S,
 S_y = Y Component of S,
 θ = Angular Displacement of E from the Robot X Axis, and
 ψ = Angular Displacement of S from the Robot X Axis.

Figure 32. Error Compensation Equations

The two points obtained from the initial searches are used to define the vectors representing the calibration surfaces. These vectors in turn are used to compute the starting points and vector paths to be used in the search for the four calibration points.

Following the sequence of operations described previously, ERCALC obtains the four calibration data points and then calls upon the subroutine, ANGCLC, to aid in the computation of the error compensation parameters. Finally, ERCALC calls CNTRL to reset the robot controller and returns execution to the calling routine. Upon exit, the magnitude of the eccentricity is stored in MAG and the angular position is stored in PHI.

5.1.2 TOUCH

The tactile subroutine, TOUCH, conducts a two pass search for the side of an object and adds an error compensation vector to the point of contact. This routine was written in FORTRAN and resides in the IBM AT.

Upon entry into TOUCH, the X coordinate of the starting point, the Y coordinate of the starting point, the X component of the search vector, the Y component of the search vector, and the R and Z components of the robot are stored in the variables, XPNT1, YPNT1, X1, Y1, THETA, and Y3. In addition, FLAG1 contains the error compensation code. If FLAG1 equals 1, then the error compensation vector is computed and added to the point of contact. If FLAG1 does not equal 1, the error compensation vector is not computed.

TOUCH first computes a set of 50 points along the search vector and simultaneously calls WRITEP to download the points to the controller. The points are 0.05 inches apart.

TOUCH subsequently calls EXEC to command the robot to search for object's side. It then waits for the controller to reach a break point, which occurs when the robot has made contact with the object and has moved the probe to the starting point of the second search. When the controller reaches the break point, TOUCH calls READP to read in the starting point.

The second set of 50 points are then computed and downloaded to the controller. The points are 0.001 inches apart, and are along the original search vector. TOUCH then commands the robot to conduct its second search and waits for the controller to reach the end of the search program, which occurs when the robot has made contact with the object.

When the controller reaches the end of program execution, TOUCH reads in the point of contact. If FLAG1 is equal to 1, then TOUCH calls upon ANGSUB to aid in the computation of the error compensation vector. The vector is subsequently added to the contact point. If FLAG1 is not equal to 1, then no error compensation is added to the contact point. In addition, a vector whose magnitude is the radius of the tip of the probe (0.005 inches) and whose direction is the same as the search vector, is added to the point of contact. Upon exit, the X coordinate of the contact point is stored in XPNT1 and the Y coordinate is stored in YPNT1.

5.1.3 ANGCLC and ANGSUB

The tactile subroutines, ANGCLC and ANGSUB, are used in the computation of angles. The routines were written in FORTRAN and reside in the IBM AT.

ANGCLC computes the positive angular displacement of a vector relative to the positive X axis of the robot. Upon entry, the X component of the vector is stored in X1 and the Y component is stored in Y1. Upon exit, the angle (radians) is stored in THETA.

ANGSUB computes the difference between the angles stored in A and B. Upon exit, the result (radians) is stored in THETA.

5.1.4 TTOUCH

The tactile subroutine, TTOUCH, commands the robot to conduct a two pass search for an object. A contact point is recorded when the very tip of the tactile probe makes contact with the object on the second pass. This essentially eliminates any error due to tool point geometry.

This routine was written in AML/E and resides in the IBM 7545 controller. The initial starting point of the search, and the group of points along the search vector are stored in the constant, ST_PT2, and the variable group, VECTOR. The values stored in these variables are down loaded from the IBM AT before and during program execution.

TTOUCH initially moves the tactile probe to the starting point listed in ST_PT2. It then moves the tactile probe to each of the points described in VECTOR. Each move results in a probe displacement of 0.05 inches. In between each move, TTOUCH checks the status of input port #10. If the input port is turned on, then the probe has made contact with object. TTOUCH subsequently backs up and reads the position of the probe into the constant, TCH_PT, using the WHERE command.

After reading in the point, TTOUCH reaches a break point. At this point, the subroutine TOUCH reads TCH_PT from controller memory and down loads a new set of vector points. It subsequently commands the controller to resume execution of TTOUCH.

TTOUCH moves the tactile probe to each point defined in VECTOR, while simultaneously checking the status of input port #10. Each move results in a probe displacement of 0.001 inches.

When contact with the object is detected, the contact point is recorded. The Z coordinate of every succeeding point in VECTOR is then increased by 0.01 inches. This is accomplished by moving the probe to a point, using the DPMOVE command to elevate the probe, using the WHERE command to record the new point, and then moving the probe to the subsequent point. This method is very time consuming and causes excess wear on the fixture. However, since AML/E does not have the mathematical capability of updating the points, and because the transfer of the points to and from

the IBM AT (which has the capability to mathematically update the points) takes even longer, this method is used.

After updating the points, TTOUCH moves the probe to each of the new points and iterates the described procedure until no contact is made. At this moment, it records the last contact point into TCH_PT. This point represents the point of contact between the tip of the probe and the side of the object. TTOUCH then moves the probe above the fixture and reaches its end point.

5.2 TACTILE SEARCH AND REGION DEFINITION

This segment of code performs a two pass tactile search for two adjacent sides on each fixture identified by the vision system. The data obtained from these searches is used to compute the bounding points of the region defined by the fixture.

Upon exit from this code, the descriptive arrays, ABND and BBND are created. ABND, Figure 33, lists the X and Y coordinates of every regional corner point for each "A" type fixture. BBND lists the same information for each "B" type fixture.

Tactile searches are conducted by first utilizing the arrays, ATRK and BTRK, to extract the location of a fixture's foundation corner points from PNTTRK. The three foundation points form two boundary vectors which represent two adjacent sides on the fixture. A vector normal to each side is computed and a tactile search is conducted along these normal vectors

ABND (4,	4,	2	)
1 - 1st "A" Type Fixture	1 - Lower Left Corner Pnt.	1 - Equals X Coord.	
4 - 4th "A" Type Fixture	2 - Upper Left Corner Pnt.	2 - Equals Y Coord.	
	3 - Lower Right Corner Pnt.		
	4 - Upper Right Corner Pnt.		
BBND (4,	4,	2	)
1 - 1st "B" Type Fixture	1 - Lower Left Corner Pnt.	1 - Equals X Coord.	
4 - 4th "B" Type Fixture	2 - Upper Left Corner Pnt.	2 - Equals Y Coord.	
	3 - Lower Right Corner Pnt.		
	4 - Upper Right Corner Pnt.		

Figure 33. Descriptive Arrays Used for Tactile Search and Region Definition

to locate two points on each side of the object. Each contact point is 15 percent of the vector length away from each foundation point.

The accurate use of the error compensation equations requires that the tactile probe search along a vector that is truly normal to the side of a fixture. Since the position of each of the original foundation corner points is obtained through the vision system, their coordinates are only approximate. Since the coordinates are approximate, the path vector used in the initial search is approximate, which further means that coordinates of the contact points are inaccurate.

Since the same vector is used in the search for two points on a side, the coordinates of the points were modified by the same error compensation vector. Thus the vector defined by the two points accurately represents the vector defined by the side of the fixture.

To accurately locate each of the adjacent fixture sides, the contact points are used to define two new side vectors. In turn, these vectors are used a basis for a second tactile search. All tactile searches are conducted with the tactile probe oriented at 0 degrees.

Once the four contact points have been obtained, they are used to compute the corner points of the region defined by the fixture, Figure 34. The slope and Y intercept of the line passing through each set of side points is computed. The intersection of these lines defines the lower left corner point of the region. By extending the vectors defined by the side

points, from the lower left region point, the upper left, lower right, and upper right corner points are defined.

The purpose for defining a region about a fixture is to develop a coordinate system from which holes in the fixture can be referenced. The final objective of robot/sensory system is to utilize the defined regions to conduct probe insertion routines on all of the fixtures. Since the tactile probe is eccentric, it is not enough to accurately define the corner points of a fixture/region. Instead, the defined region must be shifted along a vector in order to account for the skewness of the probe.

When the robot uses regional coordinates, it defines the vector between the lower left and lower right point to be the positive X axis, Figure 35. To maintain the same rotational relationship between the robot coordinate system and the region, It rotates its R axis to an angle that is equal to the angular displacement of the region's positive X axis.

The region compensation vector is determined by computing the angular displacement between the eccentricity of the probe and the region's X axis, adding 180 degrees to the value, and using the error compensation equations to compute the vector. The compensation vector is then added to every regional corner point.

The subroutines used to conduct tactile searches and define regions reside in the IBM AT and the IBM 7545 controller. The software residing in the

controller, TTOUCH, has already been discussed. The following subsections will discuss the FORTRAN subroutines residing in the IBM AT.

5.2.1 TACTIL

The tactile subroutine, TACTIL, performs a tactile search and defines a region for every "A" type and "B" type fixture identified by the vision system.

TACTIL first calls the tactile subroutine, RECOR, to convert the coordinates of the foundation points listed in PNTTRK from vision reference to robot reference coordinates. It then calls upon EXEC to return the robot to the home position and to activate TTOUCH.

Subsequently, it calls GRAPH1 to inform the user of the pending tactile search for all "A" type fixtures. It then repeatedly calls upon the tactile subroutine, VECDEF, to conduct a tactile search for each "A" type fixture. The coordinates of the contact points are stored in ABND. After searching for "A" type fixtures, TACTIL repeats the above procedure for "B" type fixtures.

After the last tactile search, TACTIL calls upon EXEC to reset the controller. It then computes the region corner points for all "A" type fixtures using the methodology described previously. An error compensation vector is computed for each region and added to the region's corner points. This procedure is repeated for all "B" type fixtures. Upon re-

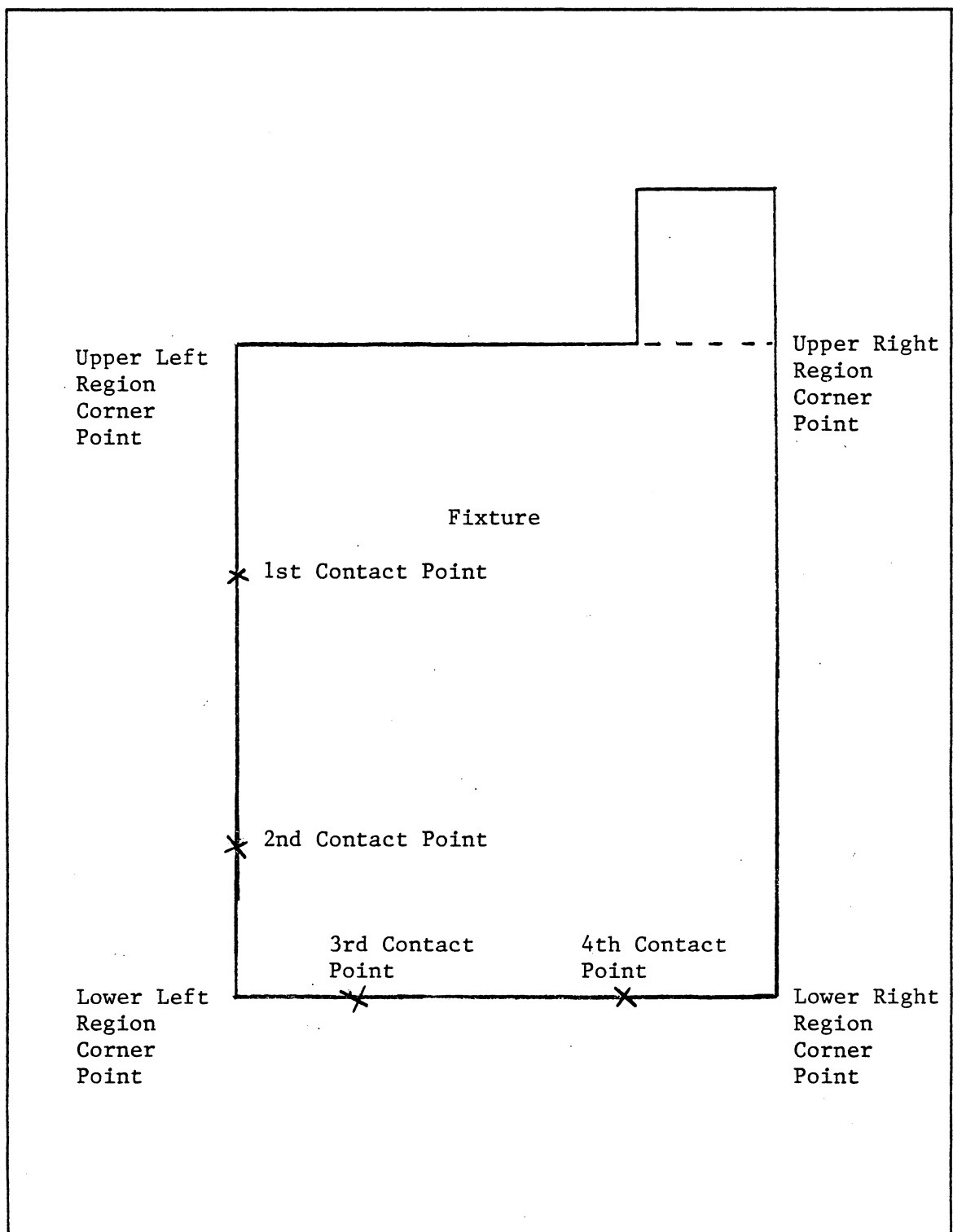


Figure 34. Fixture Defined Region

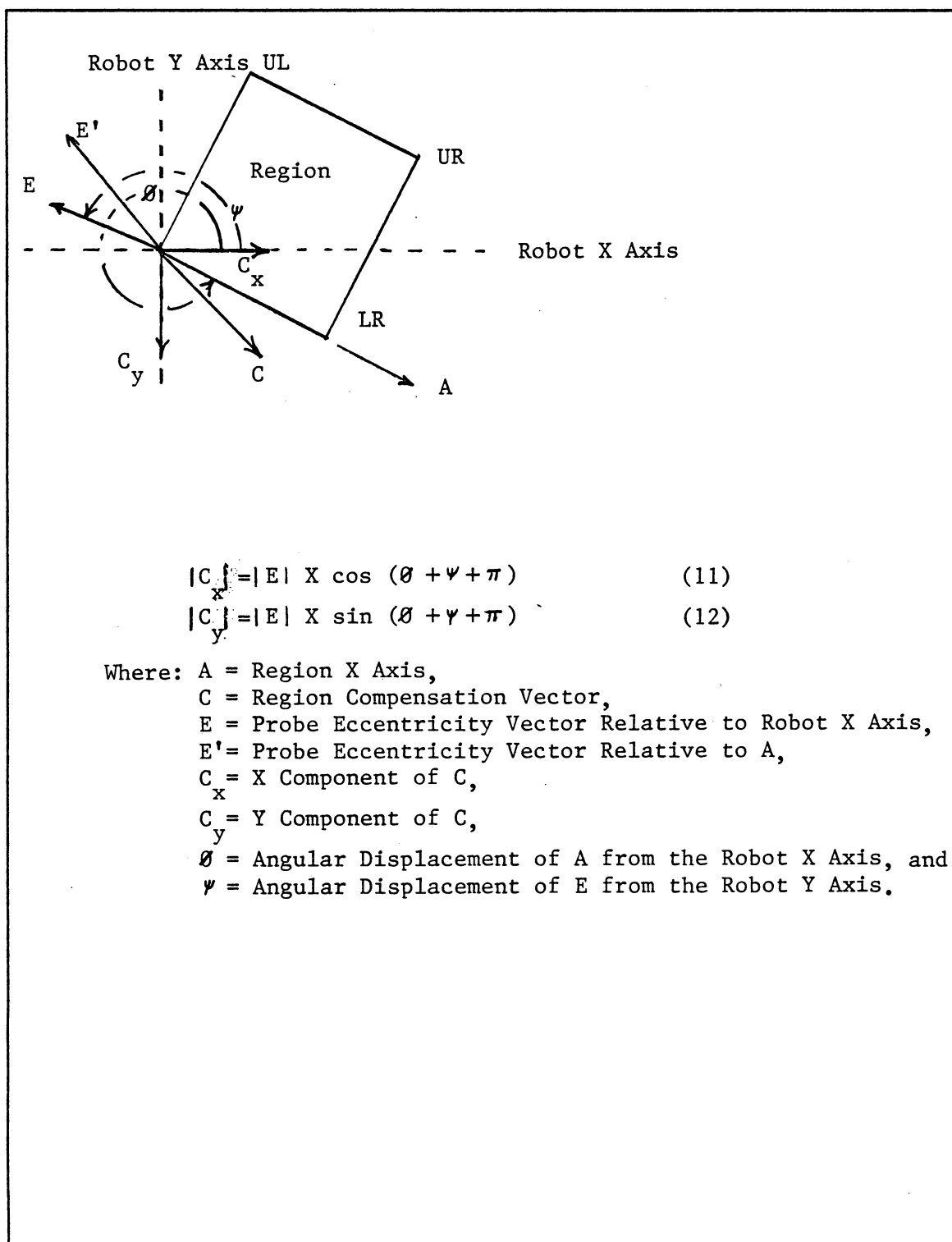


Figure 35. Region Modification

turn to the calling routine, all region corner points are stored in ABND and BBND.

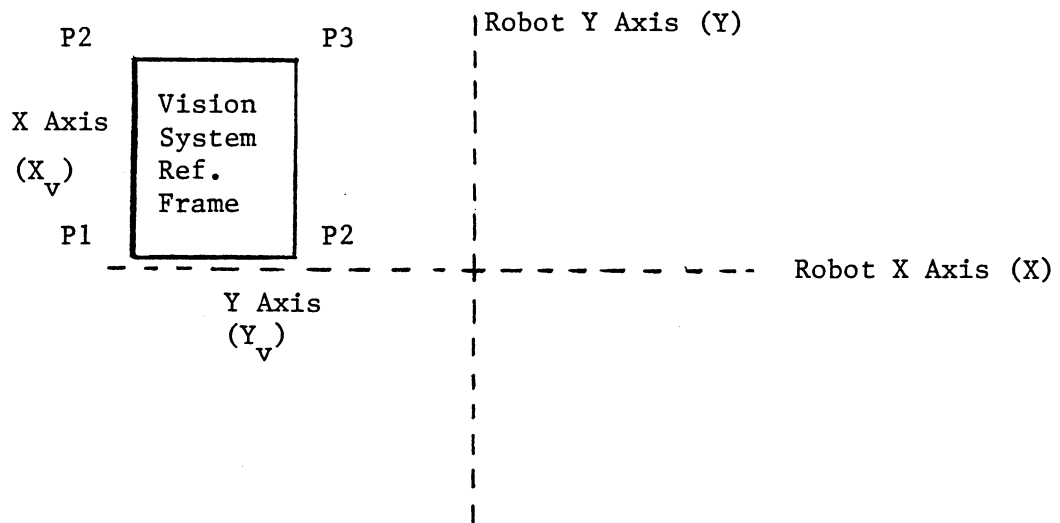
5.2.2 RECOR

The tactile subroutine, RECOR, converts the foundation corner points in PNTTRK from vision reference coordinates to robot reference coordinates, Figure 36. RECOR locates each foundation corner point in PNTTRK by scanning through ATRK and BTRK. It then uses the equations outlined in Figure 36 to convert the X and Y coordinates of the corner points. Upon exit from the routine, the converted corner points are stored in PNTTRK.

5.2.3 VECDEF

The tactile subroutine, VECDEF, conducts a tactile search for the adjacent sides of each fixture identified by the vision system. VECDEF scans through ATRK and BTRK to find the location of each fixture's foundation corner points in PNTTRK.

For each set of foundation corner points, PNTTRK computes the the side vector and normal vector for side 1 and calls upon TOUCH to locate contact points 1 and 2. VECDEF then uses contact points 1 and 2 to compute the new side vector and normal vector for the second search. After TOUCH returns the new contact points 1 and 2, VECDEF then performs the above procedure for side 2 of the object.



<u>Point</u>	<u>Vision Frame of Reference</u>	<u>Robot Frame of Reference</u>
P1	(0,0)	(-25.9, 0.4)
P2	(25.0, 0)	(-25.9, 25.0)
P3	(25.0, 18.5)	(7.4, 25.0)
P4	(0, 18.5)	(7.4, 0.4)

** All Values in inches

$$X = Y_v - 25.9 \quad (13)$$

$$Y = X_v + 0.4 \quad (14)$$

Where: X = X Coordinate in Robot Frame of Reference,
Y = Y Coordinate in Robot Frame of Reference,
 X_v = X Coordinate in Vision Frame of Reference, and
 Y_v = Y Coordinate in Vision Frame of Reference.

Figure 36. Vision and Robot Reference Systems

Upon return of program execution to TACTIL, the coordinates of each contact point are stored in ABND or BBND.

6.0 PROBE INSERTION SOFTWARE

This chapter describes the theory and subroutines associated with the probe insertion software library. All software can be referred to in Appendix D.

A region is a coordinate system, Figure 37, defined within the absolute coordinate system of the robot. A region in AML/E is defined by the following

FIXTURE: STATIC REGION (LLPT,ULPT,LRPT,URPT,LS,RS,BS,TS)

Where:

FIXTURE equals the defined region,

LLPT equals the lower left boundary corner point,

ULPT equals the upper left boundary corner point,

LRPT equals the lower right boundary corner point,

URPT equals the upper right boundary corner point,

LS equals the designated length of the left side,

RS equals the designated length of the right side,

BS equals the designated length of the bottom side, and

TS equals the designated length of the top side.

A region does not have to be rectangular in shape. Its left side can be a different length than its right side, and its top side can be a different length than its bottom side. The bottom side is considered to be the X axis of the region, and the left side is considered to be the Y axis.

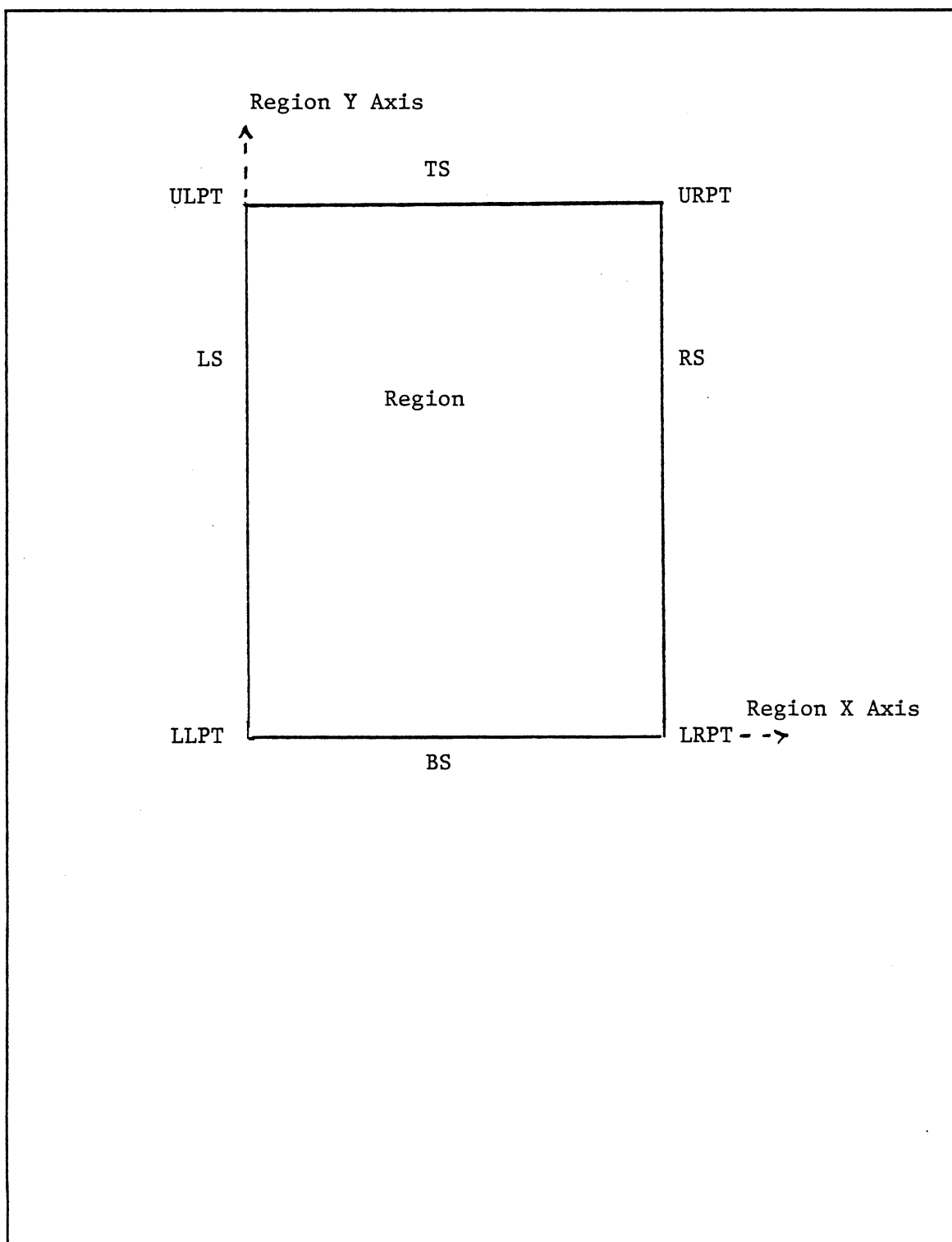


Figure 37. Region Sub-Coordinate System

The X coordinate within a region is obtained by interpolating along the bottom side and the top side and connecting a line between the two points. The Y coordinate is obtained by interpolating along the left side and right side and connecting a line between the two points. Where the lines intersect, defines the location of the point on the planar surface of the region. Interpolation involves dividing a coordinate by the appropriate side length.

The planar surface of the region does not have to normal to the Z axis of the robot. The planar surface is defined by the lower left, upper left, and lower right corner points. The Z coordinate within a region is defined normal to the surface, where a value of 0 corresponds to touching the surface.

The R coordinate is defined relative to the region's X axis. All R coordinates are considered to be positive displacements from the X axis.

The XMOVE statement is used to move the robot to a point defined within a region. This statement looks like the following

```
XMOVE (REGION, PNT)
```

Where:

REGION is the name of the region, and

PNT is the name of the regional point.

This software segment commands the robot to conduct a probe insertion routine on each fixture defined by the tactile system. The insertion

routine involves the robot sequentially moving to holes along a fixture's surface and inserting and withdrawing the tactile probe. The holes are defined relative to the region representing the fixture.

The probe insertion pattern for an "A" type object is presented in Figure 38, and the insertion pattern for a "B" type object is presented in Figure 39.

The subroutines in this library reside in both the IBM AT and the IBM 7545 controller and have a hierarchical relationship. The routines residing in the IBM AT were written in FORTRAN and act as the master program. They download region points to the slave routines in the controller and command the routines to conduct the probe insertion routines.

The routines residing in the IBM 7545 controller were written AML/E and act as the slave routines. They accept region corner points from the master routine and then command the robot to insert the tactile probe into each hole defined within the region.

These subroutines are discussed in the following sub-sections.

6.1 AINSERT AND BINSERT (IBM AT)

The probe insertion subroutines, AINSERT and BINSERT, download a set of region corner points to the slave routines in the robot controller and command the routines to execute probe insertion routines on all "A" type

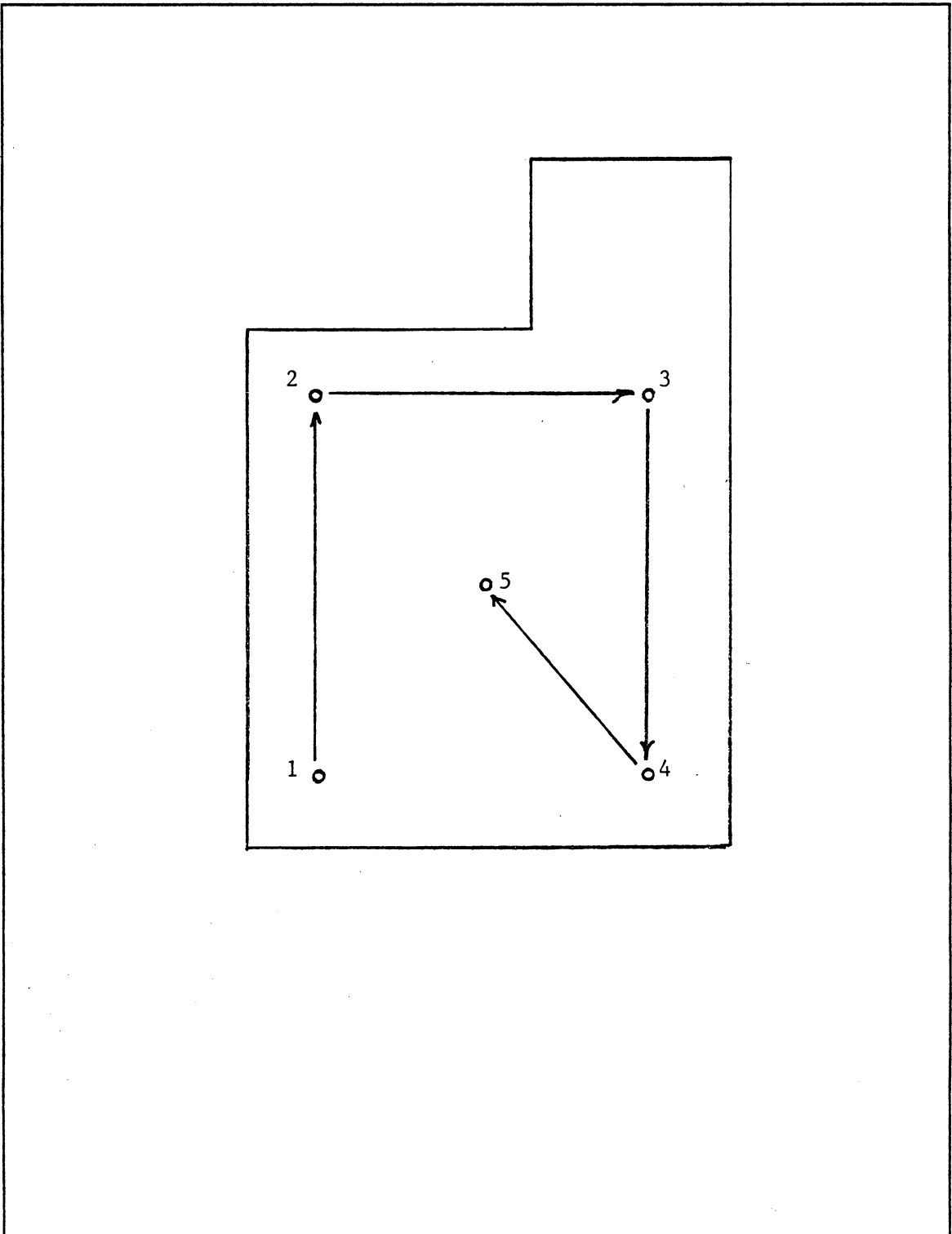


Figure 38. Probe Insertion Pattern for an "A" Type Fixture

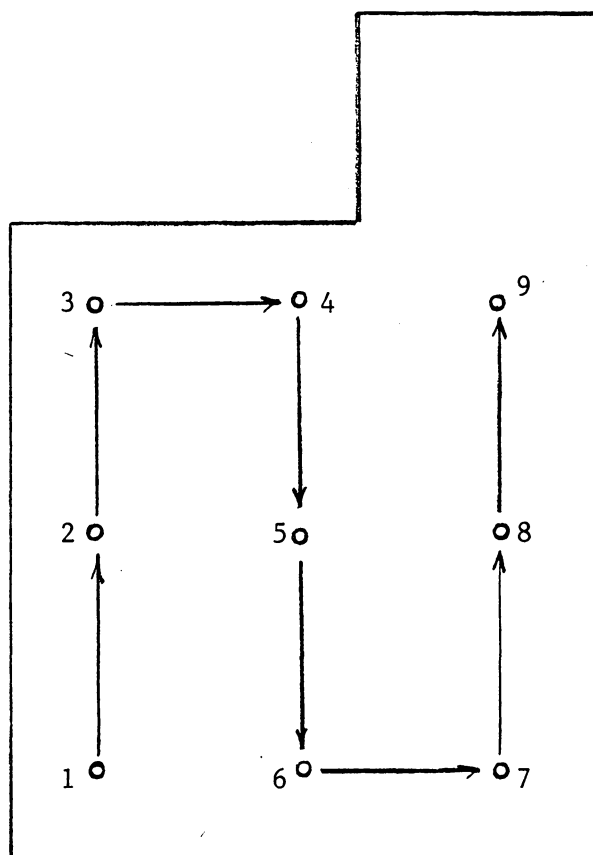


Figure 39. Probe Insertion Pattern for an "B" Type Fixture

and "B" type fixtures respectively. The subroutines work in identical fashion.

The subroutines first call upon EXEC to command the controller to return the robot to the home position and to activate the appropriate slave subroutine (AINsert or BINSert). They then down load a set of region points stored in ABND or BBND to the controller through the repeated calling of WRITEP.

They subsequently call upon EXEC to command the controller to begin execution of the insertion subroutines and to stop upon completion. After a probe insertion routine has been performed on every fixture described in ABND or BBND, the subroutines then call upon CNTRL to reset the controller.

6.2 AINSert AND BINSert (IBM 7545 CONTROLLER)

The probe insertion subroutines, AINSert and BINSert, accept a set of region boundary points from the master subroutines in the IBM AT and command the robot to insert the tactile probe into each hole defined within their respective regions. The subroutines work in identical fashion.

They first command the robot to move above the lower left corner point of their respective regions. They subsequently call the internal subroutine, INSERT, to move the tactile probe to every hole defined with their regions and to insert and withdraw the tactile probe. There is a

one second delay between each movement. AINSERT commands the robot to perform the insertion pattern described in Figure 38 while BINSERT commands the robot to perform the pattern described in Figure 39.

Upon completion of the insertion routines, the subroutines move the tactile probe above the lower left corner point of their regions. Subsequently an end statement is reached and the subroutines halt program execution.

7.0 SYSTEM ANALYSIS

A system analysis was performed on the robot/sensory system to determine its execution time, the repeatability of the tactile system, and the error encountered during probe insertion. This chapter discusses the methodology used in this analysis as well as the results and conclusions. All data can be referred to in Appendix I.

7.1 EXECUTION TIME

A test was conducted on the system to determine the time needed to calibrate the tactile probe, visually process an image of an "A" type and "B" type fixture on the workcell platform, and conduct a tactile search and region definition for each fixture.

The timing of each operation was based on the appearance of the operation messages appearing on the CRT screen. For example, the timing of the calibration process began when the message, "PROBE CALIBRATION IS BEING CONDUCTED", appeared on the CRT screen and ended when the message, "IMAGE DATA IS BEING PROCESSED," appeared on the screen.

The results of the analysis are the following. The system took 19.3 minutes to calibrate the tactile probe, 3.4 minutes to process the image of the fixtures, 8.3 minutes to search for and define a region about the "A" type fixture, and 8.5 minutes to search for and define a region about

the "B" type fixture. Thus it took 19.3 minutes for system initialization and 20.2 minutes to accurately locate two fixtures.

The execution speed of this system obviously makes it commercially infeasible. However, through two modifications, this system could very well be placed on the factory floor. The first modification would replace the software driven vision system with a hardware driven system. Hardware driven systems can process an image in a matter of seconds and are commercially available.

The second modification would replace the version 4.0 AML/E used in this research with version 4.1. Version 4.1 has the math capability and variable flexibility that version 4.0 lacks.

Using version 4.1 would dramatically decrease the execution time of the tactile and insertion software in two ways. First, all tactile and insertion hardware could reside in the IBM 7545 controller. This would eliminate the need for the constant up loading and down loading of points during probe calibration, tactile searches, and probe insertion routines. Each communication sequence takes approximately 20 seconds. Thus the time savings are obvious.

The second major time reduction results from version 4.1's math capability. Currently, the tactile subroutine, TTOUCH, has to perform an iterative process to update the Z coordinates of a group of vector points. This updating is done to insure that the final point of contact between

the tactile probe and a fixture occurs at the tip of the (refer to section 4.1.3).

This iterative process involves the physical movement of the probe to a group point, the elevation of the probe, the recording of the new probe position into the group point, and the subsequent move to the next group point. This is a very inefficient and time consuming process, but is necessary since version 4.0 does not have the capability of mathematically updating the group points. On the other hand, version 4.1 has the capability to mathematically update the group points and could thus eliminate this process.

7.2 TACTILE SYSTEM REPEATABILITY

The repeatability of the tactile probe was determined by moving the robot to a point above the surface of the calibration block, conducting a tactile search for the surface along a normal vector, and recording the point of contact. This operation was performed ten times.

The averages of the X and Y coordinates were computed, and the distance between the average point and each data point was then determined. The average and standard deviation of this new data was then computed. The standard deviation of the points is 0.003 inches. Thus the three sigma limit of the points as well as the repeatability of the system is considered to be +/- 0.009 inches.

The repeatability of the tactile system is remarkably good, considering that the reported positional repeatability of the robot is ± 0.002 inches.

7.3 PROBE INSERTION ERROR

Probe insertion error relates to the accuracy of locating one of the drill holes in a fixture. The probe insertion error for each of the corner drill holes (holes 1, 3, 7, and 9) in a "B" type fixture was measured with the system defining the fixture's sides with two points per side and with three points per side.

Data was initially collected by running the system in its normal mode (two points defining a fixture side). The fixture was oriented in four different positions, and the robot/sensory system was commanded to sense the location of the fixture and to perform an insertion routine for each position. Two runs were conducted for fixture positions 1 and 2, and one run was conducted for positions 3 and 4.

The rotation of the tactile probe and the vectors representing sides 1 and 2 were printed to the screen. In addition, tape was placed over each drill hole so that the probe pierced the tape upon insertion. The coordinates of the true hole center and insertion point were determined by placing the robot in the teach mode, rotating the tactile probe to the R coordinate used in the insertion routine, moving the probe over the desired point, and recording the coordinates.

The tactile subroutines, TACTIL and VECDEF, were then modified to define each side of the fixture using three contact points instead of two. VECDEF was modified by having it search for three points along the side of the fixture. TACTIL was modified by having it perform a linear regression on the contact points to define the line passing through them.

This new system was then run with the fixture oriented at positions 1 and 2. Two runs per position were conducted. The coordinate data was collected as described previously.

The error vectors, which are defined by the difference in location between the insertion points and the hole centers, were then computed for each set of data. In addition, the angle between the two computed side vectors for each run was also determined.

The average insertion point error for the two point system over all fixture positions is 0.0337 inches, while the average standard deviation is 0.0151 inches. The average and standard deviation of the insertion point error for the two point system over positions 1 and 2, are 0.0362 inches and 0.0161 inches respectively. The average angle between the side vectors is 89.67 degrees, while the standard deviation is 0.148 degrees.

The average and standard deviation of the insertion point error for the three point system over positions 1 and 2 are 0.0249 inches and 0.00812 inches respectively. The average angle between the side vectors is 89.95 degrees, while the standard deviation is 0.0709 degrees.

Observation of these results suggests that the three point system is superior to the two point system. This is logical, since a line defined by three points is statistically more accurate than one defined by two points.

The relationship between insertion point error and the distance between a drill hole and the origin of the defining region can be seen in Figure 40.. For the two point system, the error ranges from 0.00878 inches (closest point) to 0.0834 inches (farthest point). For the three point system, the error ranges from 0.00478 inches (closest point) to 0.05 inches (farthest point). It is obvious that the farther away a drill point is from the origin of the defining region, the greater the insertion point error.

The major reason behind this relationship appears to be the accuracy to which AML/E can locate regionally defined points. The reported accuracy for a palletizing operation ranges from 0.02 inches to 0.051 inches [6], depending upon the pallet size.

A pallet is similar to a region in that it is a coordinate system defined within the coordinate system of the robot. In addition, the coordinates of points defined within the pallet, like regions, are computed internally. However, a pallet is rectangular in shape, thus the internal computation of points is assumed to be less complex than that of a region.

Due to the greater complexity of a region versus a pallet, it is believed that the accuracy of defining a point within a region ranges beyond 0.051 inches and is linearly related to the distance from the region's origin.

To overcome the inaccuracy associated with the internal computation of region points, the coordinates of points defined within regions could be easily computed by an external subroutine, utilizing a data base and simple vector addition.

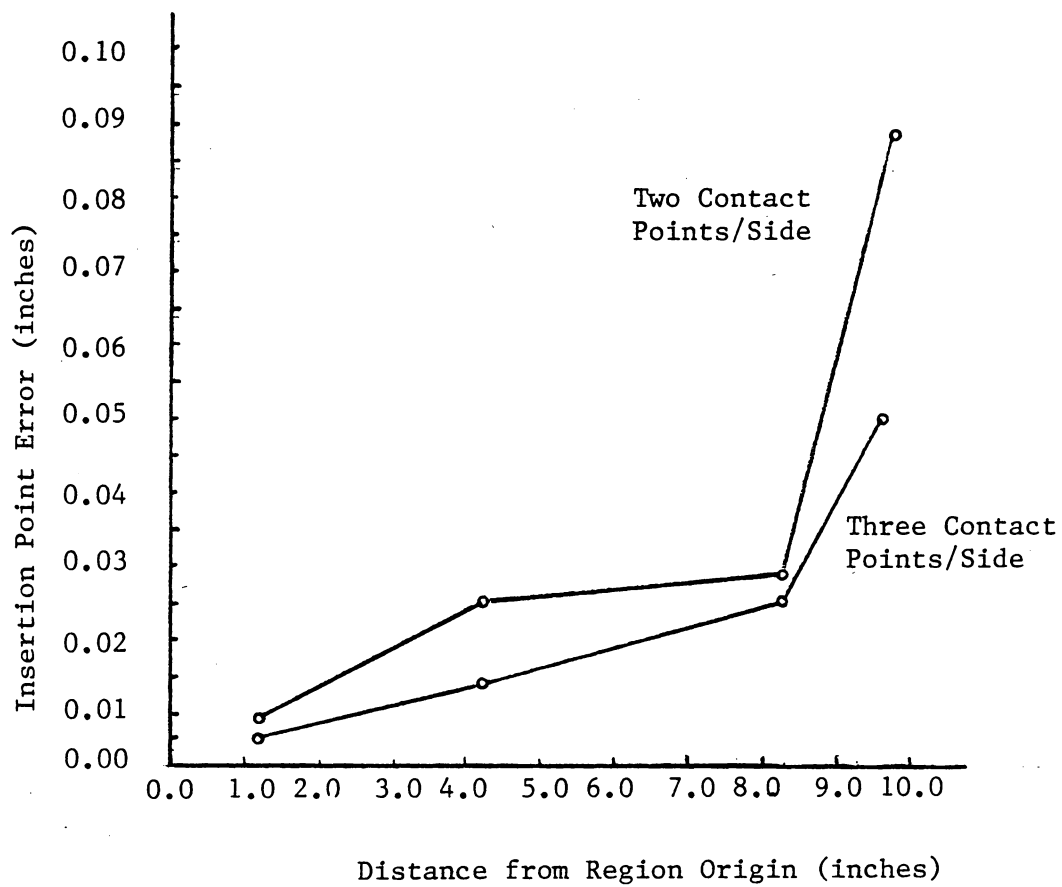


Figure 40. Insertion Point Error

8.0 CONCLUSIONS AND RECOMMENDATIONS

This research demonstrates that a robot workcell utilizing visual and tactile sensing can accurately locate an object within a large workcell area.

The vision system developed is capable of creating a data base on any two dimensional object. In addition, it has the ability to define the location and orientation of any straight edged object which has a positive angle of 90 degrees or greater between adjacent sides.

The processing time of the system was determined to be approximately 3.4 minutes. It is recommended that similar workcells in the future utilize commercially available, hardware driven vision systems in order to drastically reduce processing time, and to increase system capability.

The tactile system developed is self-calibrating and has a repeatability of 0.009 inches. The time needed for probe calibration is 19.3 minutes while the time needed to perform a tactile search for an object is 8.5 minutes. Communication between the host computer and the robot controller and the group point redefinition were determined to be the major bottle necks in the system. To eliminate these bottle necks, it is recommended that future tactile systems, which utilize AML/E, use version 4.1 rather than 4.0.

Tactile systems which define the sides of fixtures with three points were shown to be more accurate in terms of insertion point error than two point systems. In addition, insertion point error appears to increase with an increase in the distance of a point from the origin of its defining region. The major source of insertion point error was determined to be internal coordinate computation by the controller. To potentially eliminate this error, it is recommended that external subroutines be used to compute the coordinates of points defined within regions.

Future research in this area could vary from the simple implementation of the recommendations outlined in this chapter, to the integration of more sophisticated sensing systems for the assembly of 3 dimensional parts.

LIST OF REFERENCES

1. Agrawala, Ashok, and Kulkarnu, Ashok, "A Sequential Approach to the Extraction of Shape Features" Computer Graphics and Image Processing 6, 538-557 (1977)
2. Amat, J., Casals, A., and Llarío, V., "Location of Work-Pieces and Guidance of Industrial Robots with a Vision System," Proceedings of the 4th International Conference on Robot Vision and Sensory Controls, (Oct., 1984)
3. AML/Entry Version 3 User's Guide, IBM Manual 8577125
4. Didocha, R.J, Lyons, D.W., and Thompson, J.C., "Integration of Tactile Sensors and Machine Vision for Control of Robotic Manipulators," Proceedings of the 15th International Symposium on Industrial Robots/Robots 9 Conference, (June, 1985)
5. Hewit, J.R. and Jones, R.E., "Boiler Tube Platen Manufacture Using Robots Aided by Vision Sensing," Proceedings of the 14th International Symposium on Industrial Robots/Robots 8 Conference, (June, 1984)
6. IBM Manufacturing Systems Hardware Library, IBM Manual 8577126
7. Kelly, Robert B., Birk, John R., Martins, Henrique, and Tella, Richard, "A Robot System which Acquires Cylindrical Workpieces from Bins," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 1, No. 2, 204-213 (March/April, 1982)
8. Kohno, M., Horino, H., and Isobe, M., "Intelligent Assembly Robot," Hitachi Review, Vol. 30, No.4, 211-216 (1981)
9. Lam, Lisa S., Moore, Timothy J., and Snell, Sidnee, "Robotic Probing of Printed Circuit Boards Undergoing Test," Proceedings of the 15th International Symposium on Industrial Robots/Robots 9 Conference, (June 1985)
10. Morgan, Christopher L., Blue Book of Assembly Routines for the IBM PC & XT, 24-40, Plume & Waite (1984).
11. Mertens, Peter M., "An Automated Palletizing System for Industrial Robots," MS Thesis, Virginia Polytechnic Institute and State University (1985).

12. Nehr, G. and Martini, P., "The Coupling of a Workpiece Recognition System with an Industrial Robot," Robot Vision, IFS (Publications) Ltd., UK., 83-96, (1983)
13. Snader, Drew, A., "Toward The Development Of A Universal Programming/Documentation System For Programmable Controllers On A Host Microcomputer," MS Thesis, Virginia Polytechnic Institute and State University (1985).

**The vita has been removed from
the scanned document**

Volume II

THE INTEGRATION OF VISUAL AND TACTILE SENSING FOR
THE DEFINITION OF REGIONS WITHIN A ROBOT WORKCELL

by

Edward Christopher De Meter

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Industrial Engineering and Operations Research

June 11, 1986

Blacksburg, Virginia

APPENDIX A. MAIN SYSTEM SOFTWARE

PROGRAM MAIN

```
C *****
C *
C * PROGRAM: MAIN
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 5-5-86
C *
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *     1. DETERMINES THE ERROR COMPENSATION PARAMETERS FOR THE TACTILE
C *        PROBE,
C *
C *     2. GRABS AND CONVERTS A VISUAL IMAGE INTO A BINARY IMAGE,
C *
C *     3. DEFINES OBJECTS WITHIN THE BINARY IMAGE,
C *
C *     4. LOCATES THE CORNER POINTS OF EACH OBJECT WITHIN THE
C *        COORDINATE SYSTEM OF THE BINARY IMAGE,
C *
C *     5. DETERMINES WHETHER ANY OF THE OBJECTS ARE EITHER "A" TYPE
C *        OR "B" TYPE OBJECTS,
C *
C *     6. CONDUCTS A TACTILE PROBE SEARCH FOR ALL "A" TYPE OBJECTS AND
C *        "B" TYPE OBJECTS, WHILE SIMULTANEOUSLY COMPENSATING FOR THE
C *        ERROR DUE TO ECCENTRICITY IN THE TACTILE PROBE,
C *
C *     7. CONDUCTS A PROBE INSERTION ROUTINE FOR ALL "A" TYPE AND "B"
C *        TYPE OBJECTS, AND
C *
C *     8. ALLOWS THE USER TO EITHER RECONDUCT THE INSERTION ROUTINES,
C *        PROCESS A NEW BINARY IMAGE, OR END THE PROGRAM.
C *
C *
C * SUBROUTINES USED
C *
C * THIS PROGRAM CALLS UPON THE FOLLOWING SUBROUTINES:
C *
C *     1. ERCALC
C *     2. GRAPH1
C *     3. GRAPH2
C *     4. GRAPH3
C *     5. FRMGRB
C *     6. CNVRT
C *     7. DFBND
C *     8. FNSTAT
C *     9. PNTDEF
C *    10. ASEROR
C *    11. BSEROR
C *    12. TACTIL
C *    13. AINSERT
C *    14. BINSERT
C *    15. EXEC
C *
C *****
```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * DETERMINE THE ERROR COMPENSATION PARAMETERS FOR THE TACTILE PROBE.
C *
C *****

```

CALL ERCALC

```

C *****
C *
C * INITIALIZE PROGRAM VARIABLES.
C *
C *****

```

```

10 OBJCNT = 0
   PARCNT = 0
   SCNFLG = 0

```

```

BDCNT = 0
OBCNT = 0
DO 20 MARK1 = 1,40

YNGROW(MARK1) = 0
TYPE(MARK1) = 0
AREA(MARK1) = 0.0
XMOMNT(MARK1) = 0.0
YMOMNT(MARK1) = 0.0

20 CONTINUE

DO 30 MARK1 = 1,20

OFSCNT(MARK1) = 0
PNTNUM(MARK1) = 0
RMARK(MARK1) = 0
LMARK(MARK1) = 0
LTCNT(MARK1) = 0
RTCNT(MARK1) = 0

30 CONTINUE

DO 50 MARK1 = 1,20

DO 40 MARK2 = 1,10

PARENT(MARK1,MARK2) = 0

40 CONTINUE

50 CONTINUE

55 LINNUM = 1
CALL GRAPH1(LINNUM)

C *****
C *
C * GRAB A PICTURE FRAME, CONVERT IT TO A BINARY IMAGE, AND DEFINE AND
C * ASSIGN OBJECT BOUNDARIES.
C *
C *****

CALL FRMGRB
CALL CNVRT
CALL DFBND

C *****
C *
C * COMPUTE THE TOTAL AREA AND CENTROIDAL COORDINATES FOR EACH OBJECT,
C * LOCATE AND DEFINE THE CORNER POINTS FOR EACH OBJECT, AND DETERMINE
C * WHICH OBJECTS IF ANY ARE "A" TYPE OR "B" TYPE OBJECTS.
C *
C *****

CALL FNSTAT
CALL PNTDEF
CALL ASEROR
CALL BSEROR

CALL GRAPH2(BDCNT,ANUM,BNUM)

C *****
C *

```

```

C      * CONDUCT A TACTILE PROBE SEARCH ON ANY "A" TYPE OR "B" TYPE OBJECTS.
C      *
C      *****

      CALL TACTIL

C      *****
C      *
C      * IF THE NUMBER OF "A" TYPE OBJECTS EXCEEDS ZERO, THEN CONDUCT A PROBE
C      * INSERTION ROUTINE ON THOSE OBJECTS.
C      *
C      *****

      60 IF (ANUM .GT. 0) THEN

          LINNUM = 4
          CALL GRAPH1(LINNUM)
          CALL AINSERT

      ENDIF

C      *****
C      *
C      * IF THE NUMBER OF "B" TYPE OBJECTS EXCEEDS ZERO, THEN CONDUCT A PROBE
C      * INSERTION ROUTINE ON THOSE OBJECTS.
C      *
C      *****

      IF (BNUM .GT. 0) THEN

          LINNUM = 5
          CALL GRAPH1(LINNUM)
          CALL BINSERT

      ENDIF

C      *****
C      *
C      * DETERMINE INTERACTIVELY WHETHER THE USER DESIRES TO RECONDUCT THE
C      * INSERTION ROUTINES, PROCESS A NEW IMAGE, OR END THE PROGRAM.
C      *
C      *****

      CALL GRAPH3(KEY)

      IF (KEY .EQ. 1) THEN

          GO TO 60

      ENDIF

      IF (KEY .EQ. 2) THEN

          CALL EXEC(RTRN,ERROR)

          GO TO 10

      ENDIF

      CALL EXEC(RTRN,ERROR)

C      *****
C      *
C      * STOP PROGRAM EXECUTION
C      *
C      *****

```

200 STOP
END

BLOCK DATA

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROM,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

DATA HOLE,UNDFD,CCW,UP/4*0/BODY,CMLPTE,CW,DOWN/4*1/
DATA LEFT/1/RIGHT/2/AUTO/20/STRCYC/22/STPCYC/23/STPMEM/24/
DATA RECLL/12/RTRN/11/APP1/31/APP2/32/APP3/33/RESET/20/
DATA FLDWDH/24.623/ARATIO/1.3516/THRSH/60/
END

```

APPENDIX B. VISION SYSTEM SOFTWARE

SUBROUTINE FRMGRB

```
C *****
C *
C * SUBROUTINE FRMGRB
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE 1-10-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM INITIALIZES THE FRAME GRABBER VISION SYSTEM AND GRABS
C * A PICTURE FRAME.
C *
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C *     1. VINIT
C *     2. FREEZE
C *     3. RAMP
C *
C *****
```

```
INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRT(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```
REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA
```

```
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROM,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
```

```

COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CH,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * INITIALIZE THE VISION HARDWARE, GRAB A PICTURE FRAME , AND SET THE LOOK
C * UP TABLE.
C *
C *****

```

```

CALL VINIT(ERROR)
CALL FREEZE(ERROR)
CALL RAMP(ERROR)

```

```

C *****
C *
C * INITIALIZE THE VISION HARDWARE, GRAB A PICTURE FRAME, AND SET THE LOOK
C * UP TABLE A SECOND TIME.
C *
C *****

```

```

CALL VINIT(ERROR)
CALL FREEZE(ERROR)
CALL RAMP(ERROR)

```

```

C *****
C *
C * RETURN TO MAIN PROGRAM
C *
C *****

```

```

RETURN
END

```

```

SUBROUTINE CNVRT

```

```

C *****
C *
C * SUBROUTINE CNVRT
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-10-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CONVERTS A GRAY SCALE PICTURE IMAGE TO A BINARY IMAGE
C * USING A SIMPLE THRESHOLDING TECHNIQUE. ALL PIXEL ROWS LESS THAN 50
C * OR GREATER THAN 400 ARE GIVEN A VALUE OF 255. ALL PIXEL COLUMNS
C * LESS THAN 50 OR GREATER THAN 450 ARE GIVEN A VALUE OF 255.
C *
C *
C *

```



```

C      * SUBROUTINES USED
C      *
C      * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C      *
C      *      1. VRDROW
C      *      2. VWRROW
C      *
C      *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER  OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER  BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER  TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER  CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER  MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER  LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER  LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER  PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER  CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER  ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL      AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL      AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL      PNTTRK(20,250,2),PERIM(20)
REAL      SDSTAT(20,50),PNSTAT(20,50)
REAL      XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL      A,B,THETA,PHI,MAG,SIGMA

COMMON    RESET,STPMEM,RECLL
COMMON    VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON    BNDTRK,SCNROW,ERROR,LINE
COMMON    DTHROW,BUFROW,BTHROW
COMMON    OLDROW,NEWROW,BNDROW
COMMON    OBJROW,PARENT,BNDCNT,LFETRK
COMMON    BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON    TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON    CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON    PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON    MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON    OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON    LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON    PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON    CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON    ANUM,BNUM,ATRK,BTRK
COMMON    AREA,XMOMNT,YOMNT,FLDWDH,ARATIO
COMMON    AAREA,XXMNT,YYMNT,ABND,BBND
COMMON    PNTTRK,PERIM
COMMON    SDSTAT,PNSTAT
COMMON    XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON    A,B,THETA,PHI,MAG,SIGMA

DO 30 LINE = 0,450

C      *****
C      *
C      * READ THE PIXEL ROW IN FROM THE HARDWARE AND ASSIGN IT TO THE ARRAY
C      * SCNROW.

```

```

C      *
C      *****
10  CALL VRDROW (LINE,SCNROW,ERROR)

      DO 20 COL = 1,512

C      *****
C      *
C      * IF THE PIXEL ELEMENT (LINE,COL) IS GREATER THAN THE THRESHOLD VALUE,
C      * THEN ASSIGN THAT ELEMENT THE VALUE 255. IF THE ELEMENT IS LESS THAN
C      * THEN THE THRESHOLD VALUE, THEN ASSIGN THAT ELEMENT THE VALUE 0.
C      *
C      *****

      IF (SCNROW(COL) .GT. THRS) THEN

        SCNROW(COL) = 255

      ELSE

        SCNROW(COL) = 0

      ENDIF

C      *****
C      *
C      * IF THE LINE NUMBER IS LESS THAN 50 OR GREATER THAN 400 THEN ASSIGN THE
C      * PIXEL ELEMENT THE VALUE 255. THIS IS DONE TO ELIMINATE ANY SHADOWING
C      * EFFECT AROUND THE EDGES OF THE FIELD OF VIEW.
C      *
C      *****

      IF (LINE .LT. 50 .OR. LINE .GT. 400) THEN

        SCNROW(COL) = 255

      ENDIF

C      *****
C      *
C      * IF THE COL. NUMBER IS LESS THAN 50 OR GREATER THAN 450 THEN ASSIGN THE
C      * PIXEL ELEMENT THE VALUE 255. THIS IS DONE TO ELIMINATE ANY SHADOWING
C      * EFFECT AROUND THE EDGES OF THE FIELD OF VIEW.
C      *
C      *****

      IF (COL .LT. 50 .OR. COL .GT. 450) THEN

        SCNROW(COL) = 255

      ENDIF

20  CONTINUE

C      *****
C      *
C      * WRITE THE NEW VALUES FROM SCNROW BACK INTO THE PIXEL ARRAY
C      *
C      *****

      CALL VWRROW (LINE,SCNROW,ERROR)

```

30 CONTINUE

```
C *****
C *
C * RETURN TO THE MAIN PROGRAM
C *
C *****
```

RETURN
END

SUBROUTINE DFBND

```
C *****
C *
C * SUBROUTINE DFBND
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-20-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *     1. LOCATES AND ASSIGNS BOUNDARIES FOR EACH OBJECT (BODIES AND HOLES)
C *        IN THE BINARY IMAGE ARRAY
C *
C *     2. DETERMINES WHICH OBJECTS ARE PARENTS OF OTHER OBJECTS, AND
C *
C *     3. UPDATES STATISTICAL INFORMATION SUCH AS AREA AND MOMENTS OF
C *        OF INERTIA FOR EACH OBJECT.
C *
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C *     1. VRDROW
C *     2. BNDSCN
C *     3. BNDPTH
C *     4. BNDDTH
C *     5. STSCAN
C *     6. LCNSRT
C *
C *****
```

```
INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLPTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRT(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```
REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
```

```

REAL    PNTTRK( 20,250,2),PERIM( 20 )
REAL    SDSTAT( 20,50 ),PNSTAT( 20,50 )
REAL    XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL    A,B,THETA,PHI,MAG,SIGMA

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROM,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNROW,DHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * INITIALIZE BNDTRK WITH 514
C *
C *****

```

```

DO 40 MARK1 = 0,479
DO 35 MARK2 = 1,40

```

```

BNDTRK(1,MARK2,MARK1) = 514
BNDTRK(2,MARK2,MARK1) = 514

```

```

35 CONTINUE
40 CONTINUE

```

```

DO 100 LINE = 0,400

```

```

C *****
C *
C * INITIALIZE BTHROW AND DTHROW WITH 0.
C *
C *****

```

```

DO 45 MARK1 = 1,100

```

```

BTHROW(1,MARK1,1) = 0
BTHROW(1,MARK1,2) = 0
BTHROW(2,MARK1,1) = 0
BTHROW(2,MARK1,2) = 0
DTHROW(1,MARK1,1) = 0
DTHROW(1,MARK1,2) = 0
DTHROW(2,MARK1,1) = 0
DTHROW(2,MARK1,2) = 0

```

```

45 CONTINUE

```

```

C *****
C *
C * SCAN THE ROW AND DETERMINE BOUNDARIES
C *
C *****

CALL VRDROW(LINE,SCNROW,ERROR)
BND CNT(LINE) = 0

DO 50 COL = 1,512

IF (COL .EQ. 1 .AND. SCNROW(COL) .EQ. 0) THEN

    BND CNT(LINE) = BND CNT(LINE) + 1
    NEWROW (1,BND CNT(LINE),1) = COL

ELSE IF (COL .EQ. 512 .AND. SCNROW(COL - 1) .EQ. 0) THEN

    NEWROW (2,BND CNT(LINE),1) = COL

ELSE IF (COL .GT. 0 .AND. SCNROW(COL) .EQ. 0 .AND. SCNROW(COL-1) .
2 EQ. 255) THEN

    BND CNT(LINE) = BND CNT(LINE) + 1
    NEWROW (1,BND CNT(LINE),1) = COL

ELSE IF (COL .GT. 0 .AND. SCNROW(COL) .EQ. 255 .AND. SCNROW(COL-1)
2 .EQ. 0) THEN

    NEWROW (2,BND CNT(LINE),1) = COL

ENDIF

50 CONTINUE

C *****
C *
C * IF BND CNT EQUALS 0 AND SCNFLG EQUALS 0, THEN UPDATE BNDTRK.
C *
C *
C *****

IF (BND CNT(LINE) .EQ. 0 .AND. SCNFLG .EQ. 0) THEN

    GO TO 100

ENDIF

C *****
C *
C * IF THE TOP OF AN OBJECT HAS BEEN DISCOVERED, THEN SET INITIAL STATISTICS
C * FOR BNDTRK, TYPE, OBSTAT, AND NEWROW. SET OLDROW EQUAL TO NEWROW.
C * SET SCNFLG EQUAL TO 1.
C *
C *****

IF (BND CNT(LINE) .GT. 0 .AND. SCNFLG .EQ. 0) THEN

    DO 60 MARK1 = 1,BND CNT(LINE)

    OBJCNT = OBJCNT +1

```

```

    PARCNT = PARCNT +1
    MARK2 = MARK1+OBJCNT-1

    NEWROW(1,MARK1,2) = MARK2
    NEWROW(2,MARK1,2) = MARK2

    YNGROW(MARK2) = MARK2
    PARENT(PARCNT,1) = OBJCNT
    LBDTRK(PARCNT,1,1) = OBJCNT
    RBDTRK(PARCNT,1,1) = OBJCNT

    BNDTRK(1,MARK2,LINE) = NEWROW(1,MARK1,1)
    BNDTRK(2,MARK2,LINE) = NEWROW(2,MARK1,1)

    TYPE(MARK2) = BODY
    OBSTAT(MARK2) = UNDFD
    LFETRK(MARK2,1) = LINE
    BNDRTE(MARK2) = CCH

60    CONTINUE

    SCNFLG = 1
    GO TO 90

ENDIF

C    *****
C    *
C    * DETERMINE WHICH BOUNDARIES HAVE CONTINUED AND TRACK THEM.
C    *
C    *****

    CALL BNDSCN

C    *****
C    *
C    * CHECK TO SEE IF ANY BOUNDARIES HAVE BEEN CREATED.  IF SO, THEN
C    * DETERMINE WHICH BODIES OR HOLES HAVE BEEN CREATED.
C    *
C    *****

    BTHFLG = 0
    DO 70 MARK1 = 1,BNDCNT(LINE)

    IF (BTHROW(1,MARK1,1) .EQ. 0 .OR. BTHROW(2,MARK1,1) .EQ. 0) THEN

        BTHFLG = 1

    ENDIF

70 CONTINUE

    IF (BTHFLG .EQ. 0) THEN

        GO TO 75

    END IF

    CALL BNDBTH

C    *****
C    *
C    * CHECK TO SEE IF ANY BOUNDARIES HAVE DISAPEARED.  IF SO, THEN DETERMINE
C    * WHICH BODIES HAVE TERMINATED OR MERGED.

```

```

C      *
C      *****

75 DTHFLG = 0

    DO 80 MARK1 = 1,BNDCNT(LINE-1)

    IF (DTHROW(1,MARK1,1) .EQ. 0 .OR. DTHROW(2,MARK1,1) .EQ. 0) THEN

        DTHFLG = 1

    ENDIF

80 CONTINUE

    IF (DTHFLG .EQ. 0) THEN

        GO TO 90

    ENDIF

    CALL BNDDTH

C      *****
C      *
C      * IF BNDCNT (LINE) EQUALS 0 AND SCNFLG EQUALS 1, THEN SET SCNFLG EQUAL
C      * TO 0.
C      *
C      *****

    IF (BNDCNT(LINE) .EQ. 0 .AND. SCNFLG .EQ. 1) THEN

        SCNFLG = 0

    ENDIF

C      *****
C      *
C      * ASSIGN NEWROW TO OLDROW
C      *
C      *****

90  DO 95 MARK1 = 1,100

    OLDROW(1,MARK1,1) = NEWROW(1,MARK1,1)
    OLDROW(1,MARK1,2) = NEWROW(1,MARK1,2)
    OLDROW(2,MARK1,1) = NEWROW(2,MARK1,1)
    OLDROW(2,MARK1,2) = NEWROW(2,MARK1,2)

95  CONTINUE

C      *****
C      *
C      * UPDATE THE AREA AND CENTROID CALCULATIONS FOR EACH OBJECT
C      *
C      *****

    CALL STSCAN

100 CONTINUE

C      *****
C      *

```

```

C      * IF AN OBJECT IS UNDEFINED AT THE BOTTOM OF THE FIELD OF VIEW, THEN
C      * CHANGE ITS STATUS TO COMPLETE AND ASSIGN IT A DEATH ATTRIBUTE.
C      *
C      *****
LINE = 400

DO 110 MARK1 = 1,OBJCNT

IF (OBSTAT(MARK1) .EQ. UNDFD) THEN

    IF (TYPE(MARK1) .EQ. BODY) THEN

        OBSTAT(MARK1) = CMLPTE
        DHTYP(MARK1) = 1
        LFETRK(MARK1,2) = LINE
        MARK10 = MARK1
        CALL LCNSRT

    ELSE

        OBSTAT(MARK1) = CMLPTE
        DHTYP(MARK1) = 0
        LFETRK(MARK1,2) = LINE

    ENDIF

ENDIF

110 CONTINUE

C      *****
C      *
C      * RETURN TO HOME
C      *
C      *****

RETURN
END
SUBROUTINE BNDSCN

C      *****
C      *
C      * SUBROUTINE BNDSCN
C      *
C      * WRITTEN BY: EDWARD C. DE METER
C      * DATE: 1-12-86
C      *
C      * PROGRAM DESCRIPTION
C      *
C      * THIS PROGRAM DOES THE FOLLOWING:
C      *
C      *     1. SCANS THROUGH THE ARRAYS OLDROW AND NEWROW AND DETERMINES
C      *        WHICH OBJECT BOUNDARIES ARE CONTINUATIONS OF OLD OBJECT
C      *        BOUNDARIES,
C      *
C      *     2. UPDATES THE ARRAY, NEWROW,AND
C      *
C      *     3. UPDATES THE ARRAYS, BTHROW AND DTHROW.
C      *
C      * SUBROUTINES USED: NONE
C      *
C      *****

```



```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMPLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DTHYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMPLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * THE FOLLOWING OUTER DO LOOP SCANS THROUGH THE ARRAY CONTAINING THE
C * BOUNDARIES OF THE PREVIOUS ROW SCAN. THE FOLLOWING INNER DO LOOP
C * SCANS THROUGH THE ARRAY CONTAINING THE BOUNDARIES OF THE CURRENT ROW
C * SCAN.
C *
C *****
DO 20 MARK1 = 1,BNDCNT(LINE-1)
DO 10 MARK2 = 1,BNDCNT(LINE)

```

```

C *****
C *
C * IF THE POINTERS FOR BOTH ARRAYS EQUAL TO 1, THEN THE FOLLOWING LOGIC
C * DETERMINES WHICH LEFT OR RIGHT BOUNDARIES ARE CONTINUATIONS OF THE

```

```

C      * BOUNDARIES FROM THE PREVIOUS ROW SCAN. IF A CONTINUATION IS DETECTED,
C      * THEN THE APPROPRIATE DESCRIPTIVE ARRAYS SUCH AS BNDTRK, DTHROW AND
C      * BTHROW ARE UPDATED.
C      *
C      ****
      IF (MARK1 .EQ. 1 .AND. MARK2 .EQ. 1) THEN

        IF (OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
2         NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1)) THEN

          NEWROW(1,MARK2,2) = OLDROW(1,MARK1,2)
          I = NEWROW(1,MARK2,2)
          BNDTRK(1,I,LINE) = NEWROW(1,MARK2,1)
          BTHROW(1,MARK2,1) = 1
          BTHROW(1,MARK2,2) = MARK1
          DTHROW(1,MARK1,1) = 1
          DTHROW(1,MARK1,2) = MARK2

        ENDIF

        IF (MARK2 .LT. BNDCNT(LINE) .AND. MARK1 .LT. BNDCNT(LINE-1))
2         THEN

          IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2           OLDROW(2,MARK1,1) .LT. NEWROW(1,MARK2+1,1) .AND.
3           OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
4           NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

            NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
            I = NEWROW(2,MARK2,2)
            BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
            BTHROW(2,MARK2,1) = 1
            BTHROW(2,MARK2,2) = MARK1
            DTHROW(2,MARK1,1) = 1
            DTHROW(2,MARK1,2) = MARK2

          ENDIF

        ENDIF

        IF (MARK2 .LT. BNDCNT(LINE) .AND. MARK1 .EQ. BNDCNT(LINE-1))
2         THEN

          IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2           OLDROW(2,MARK1,1) .LT. NEWROW(1,MARK2+1,1) .AND.
3           OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN

            NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
            I = NEWROW(2,MARK2,2)
            BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
            BTHROW(2,MARK2,1) = 1
            BTHROW(2,MARK2,2) = MARK1
            DTHROW(2,MARK1,1) = 1
            DTHROW(2,MARK1,2) = MARK2

          ENDIF

        ENDIF

        IF (MARK2 .EQ. BNDCNT(LINE) .AND. MARK1 .LT. BNDCNT(LINE-1))
2         THEN

          IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2           OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
3           NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

```

```

NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
I = NEWROW(2,MARK2,2)
BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
BTHROW(2,MARK2,1) = 1
BTHROW(2,MARK2,2) = MARK1
DTHROW(2,MARK1,1) = 1
DTHROW(2,MARK1,2) = MARK2

ENDIF

ENDIF

IF (MARK2 .EQ. BNDCNT(LINE) .AND. MARK1 .EQ. BNDCNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2 OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN

        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2

    ENDIF

ENDIF

ENDIF

ENDIF

C *****
C *
C * IF THE PREVIOUS BOUNDARY POINTER IS EQUAL TO 1 AND THE CURRENT BOUNDARY
C * POINTER IS GREATER THAN 1, THEN THE FOLLOWING LOGIC DETERMINES WHICH
C * LEFT OR RIGHT BOUNDARIES ARE CONTINUATIONS OF BOUNDARIES FROM THE
C * PREVIOUS ROW SCAN. IF A CONTINUATION IS DETECTED, THEN THE APPROPRIATE
C * DESCRIPTIVE ARRAYS SUCH AS BNDTRK, DTHROW, AND BTHROW ARE UPDATED.
C *
C *****

IF (MARK1 .EQ. 1 .AND. MARK2 .GT. 1) THEN

    IF (NEWROW(2,MARK2-1,1) .LT. OLDROW(1,MARK1,1) .AND.
2 OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
3 NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1)) THEN

        NEWROW(1,MARK2,2) = OLDROW(1,MARK1,2)
        I = NEWROW(1,MARK2,2)
        BNDTRK(1,I,LINE) = NEWROW(1,MARK2,1)
        BTHROW(1,MARK2,1) = 1
        BTHROW(1,MARK2,2) = MARK1
        DTHROW(1,MARK1,1) = 1
        DTHROW(1,MARK1,2) = MARK2

    ENDIF

    IF (MARK2 .LT. BNDCNT(LINE) .AND. MARK1 .LT. BNDCNT(LINE-1))
2 THEN

        IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2 OLDROW(2,MARK1,1) .LT. NEWROW(1,MARK2+1,1) .AND.
3 OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
4 NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

```

```

NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
I = NEWROW(2,MARK2,2)
BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
BTHROW(2,MARK2,1) = 1
BTHROW(2,MARK2,2) = MARK1
DTHROW(2,MARK1,1) = 1
DTHROW(2,MARK1,2) = MARK2

ENDIF

ENDIF

IF (MARK2 .LT. BNDCNT(LINE) .AND. MARK1 .EQ. BNDCNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2 OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2+1,1) .AND.
3 OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN

        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2
    ENDIF

ENDIF

ENDIF

IF (MARK2 .EQ. BNDCNT(LINE) .AND. MARK1 .LT. BNDCNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2 OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
3 NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2

    ENDIF

ENDIF

ENDIF

IF (MARK2 .EQ. BNDCNT(LINE) .AND. MARK1 .EQ. BNDCNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2 OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN
        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2

    ENDIF

ENDIF

ENDIF

```

ENDIF

```
C *****
C *
C * IF THE PREVIOUS BOUNDARY POINTER IS GREATER THAN 1 AND THE CURRENT
C * BOUNDARY POINTER IS EQUAL TO 1, THEN THE FOLLOWING LOGIC DETERMINES
C * WHICH LEFT OR RIGHT BOUNDARIES ARE CONTINUATIONS OF BOUNDARIES FROM THE
C * PREVIOUS ROW SCAN. IF A CONTINUATION IS DETECTED, THEN THE APPROPRIATE
C * ARRAYS SUCH AS BNDTRK, DTHROW, AND BTHROW ARE UPDATED.
C *
C *****
```

```
IF (MARK1 .GT. 1 .AND. MARK2 .EQ. 1) THEN
  IF (OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
2    OLDROW(2,MARK1-1,1) .LT. NEWROW(1,MARK2,1) .AND.
3    NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1)) THEN

    NEWROW(1,MARK2,2) = OLDROW(1,MARK1,2)
    I = NEWROW(1,MARK2,2)
    BNDTRK(1,I,LINE) = NEWROW(1,MARK2,1)
    BTHROW(1,MARK2,1) = 1
    BTHROW(1,MARK2,2) = MARK1
    DTHROW(1,MARK1,1) = 1
    DTHROW(1,MARK1,2) = MARK2
```

ENDIF

```
IF (MARK2 .LT. BNDCNT(LINE) .AND. MARK1 .LT. BNDCNT(LINE-1))
2 THEN

  IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2    OLDROW(2,MARK1,1) .LT. NEWROW(1,MARK2+1,1) .AND.
3    OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
4    NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

    NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
    I = NEWROW(2,MARK2,2)
    BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
    BTHROW(2,MARK2,1) = 1
    BTHROW(2,MARK2,2) = MARK1
    DTHROW(2,MARK1,1) = 1
    DTHROW(2,MARK1,2) = MARK2
```

ENDIF

ENDIF

```
IF (MARK2 .LT. BNDCNT(LINE) .AND. MARK1 .EQ. BNDCNT(LINE-1))
2 THEN

  IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2    OLDROW(2,MARK1,1) .LT. NEWROW(1,MARK2+1,1) .AND.
3    OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN

    NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
    I = NEWROW(2,MARK2,2)
    BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
    BTHROW(2,MARK2,1) = 1
    BTHROW(2,MARK2,2) = MARK1
    DTHROW(2,MARK1,1) = 1
    DTHROW(2,MARK1,2) = MARK2
```

ENDIF

ENDIF

```

2 IF (MARK2 .EQ. BNDCNT(LINE) .AND. MARK1 .LT. BNDCNT(LINE-1))
  THEN

```

```

2 IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
  OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
3   NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

```

```

  NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
  I = NEWROW(2,MARK2,2)
  BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
  BTHROW(2,MARK2,1) = 1
  BTHROW(2,MARK2,2) = MARK1
  DTHROW(2,MARK1,1) = 1
  DTHROW(2,MARK1,2) = MARK2

```

```

  ENDIF

```

```

ENDIF

```

```

2 IF (MARK2 .EQ. BNDCNT(LINE) .AND. MARK1 .EQ. BNDCNT(LINE-1))
  THEN

```

```

2 IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
  OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN

```

```

  NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
  I = NEWROW(2,MARK2,2)
  BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
  BTHROW(2,MARK2,1) = 1
  BTHROW(2,MARK2,2) = MARK1
  DTHROW(2,MARK1,1) = 1
  DTHROW(2,MARK1,2) = MARK2

```

```

  ENDIF

```

```

ENDIF

```

```

ENDIF

```

```

C *****
C *
C * IF BOTH BOUNDARY POINTERS ARE GREATER THAN 1, THEN THE FOLLOWING LOGIC
C * DETERMINES WHICH LEFT OR RIGHT BOUNDARIES ARE CONTINUATIONS OF
C * BOUNDARIES FROM THE PREVIOUS ROW SCAN. IF A CONTINUATION IS DETECTED,
C * THEN THE APPROPRIATE DESCRIPTIVE ARRAYS SUCH AS BNDTRK, DTHROW, AND
C * BTHROW ARE UPDATED.
C *
C *****

```

```

IF (MARK1 .GT. 1 .AND. MARK2 .GT. 1) THEN

```

```

2 IF (NEWROW(2,MARK2-1,1) .LT. OLDROW(1,MARK1,1) .AND.
  OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
3   OLDROW(2,MARK1-1,1) .LT. NEWROW(1,MARK2,1) .AND.
4   NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1)) THEN

```

```

  NEWROW(1,MARK2,2) = OLDROW(1,MARK1,2)
  I = NEWROW(1,MARK2,2)
  BNDTRK(1,I,LINE) = NEWROW(1,MARK2,1)
  BTHROW(1,MARK2,1) = 1
  BTHROW(1,MARK2,2) = MARK1
  DTHROW(1,MARK1,1) = 1
  DTHROW(1,MARK1,2) = MARK2

```

```

  ENDIF

```

```

IF (MARK2 .LT. BND CNT(LINE) .AND. MARK1 .LT. BND CNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2     OLDROW(2,MARK1,1) .LT. NEWROW(1,MARK2+1,1) .AND.
3     OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
4     NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2

    ENDIF

ENDIF

IF (MARK2 .LT. BND CNT(LINE) .AND. MARK1 .EQ. BND CNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2     OLDROW(2,MARK1,1) .LT. NEWROW(1,MARK2+1,1) .AND.
3     OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN

        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2

    ENDIF

ENDIF

IF (MARK2 .EQ. BND CNT(LINE) .AND. MARK1 .LT. BND CNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2     OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1) .AND.
3     NEWROW(2,MARK2,1) .LT. OLDROW(1,MARK1+1,1)) THEN

        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)
        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2

    ENDIF

ENDIF

IF (MARK2 .EQ. BND CNT(LINE) .AND. MARK1 .EQ. BND CNT(LINE-1))
2 THEN

    IF (NEWROW(1,MARK2,1) .LT. OLDROW(2,MARK1,1) .AND.
2     OLDROW(1,MARK1,1) .LT. NEWROW(2,MARK2,1)) THEN

        NEWROW(2,MARK2,2) = OLDROW(2,MARK1,2)

```

```

        I = NEWROW(2,MARK2,2)
        BNDTRK(2,I,LINE) = NEWROW(2,MARK2,1)
        BTHROW(2,MARK2,1) = 1
        BTHROW(2,MARK2,2) = MARK1
        DTHROW(2,MARK1,1) = 1
        DTHROW(2,MARK1,2) = MARK2

    ENDIF

ENDIF

ENDIF

10 CONTINUE
20 CONTINUE

C *****
C *
C * RETURN TO MAIN PROGRAM
C *
C *****

RETURN
END
SUBROUTINE BNDPTH

C *****
C *
C * SUBROUTINE BNDPTH
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-13-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM SCANS THROUGH THE ARRAY, BTHROW, TO DETERMINE IF ANY
C * NEW OBJECT BOUNDARIES HAVE BEEN BORN. UPON DETERMINING THE BIRTH
C * OF AN OBJECT, THE PROGRAM UPDATES ANY DESCRIPTIVE ARRAYS PERTAINING
C * TO THAT OBJECT.
C *
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C * 1. PARFND
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OB_CNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDORTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT

```



```

INTEGER  ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL     AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL     AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL     PNTTRK(20,250,2),PERIM(20)
REAL     SDSTAT(20,50),PNSTAT(20,50)
REAL     XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL     A,B,THETA,PHI,MAG,SIGMA

```

```

C
COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

125 MARK1 = 1

```

C *****
C *
C * CHECK THE DESCRIPTIVE ARRAY ELEMENTS IN BTHROW TO DETERMINE IF A HOLE
C * BOUNDARY HAS BEEN BORN. IF IT HAS, THEN UPDATE DESCRIPTIVE ARRAYS SUCH
C * NEWROW, BNDTRK, OBSTAT, AND TYPE.
C *
C *****

```

```

130 IF (BTHROW(2,MARK1,1) .EQ. 1 .AND. BTHROW(1,MARK1,1) .EQ. 0 .AND.
2 MARK1 .NE. 1 .AND. BTHROW(2,MARK1-1,1) .EQ. 0) THEN

```

```

    OBJCNT = OBJCNT+1
    NEWROW(1,MARK1,2) = OBJCNT
    NEWROW(2,MARK1-1,2) = OBJCNT
    BNDTRK(1,OBJCNT,LINE) = NEWROW(1,MARK1,1)
    BNDTRK(2,OBJCNT,LINE) = NEWROW(2,MARK1-1,1)
    TYPE(OBJCNT) = HOLE
    OBSTAT(OBJCNT) = UNDFD
    LFETRK(OBJCNT,1) = LINE
    BNDRTE(OBJCNT) = CW
    I = NEWROW(2,MARK1,2)
    J = NEWROW(1,MARK1-1,2)
    K = NEWROW(2,MARK1,2)

```

```

C *****
C *
C * THE FOLLOWING LOGIC DETERMINES WHAT OBJECT WILL BE CONSIDERED THE HOLE'S

```

```

C      * PARENT.
C      *
C      *****

      IF (TYPE(I) .EQ. BODY) THEN

          PARNT = I

      ELSE IF (TYPE(J) .EQ. BODY) THEN

          PARNT = J

      ELSE

          PARNT = YNGROW(K)

      ENDIF

      YNGROW(OBJCNT) = PARNT
      OBJCT = OBJCNT

C      *****
C      *
C      * CALL SUBROUTINE PARFND TO UPDATE THE PARENTAL DESCRIPTIVE ARRAY, PARENT.
C      *
C      *****

      CALL PARFND

C      *****
C      *
C      * MASK OUT THE DESCRIPTIVE ARRAY ELEMENTS IN BTHROW, AND START THE
C      * SEARCH PROCEDURE OVER AGAIN
C      *
C      *****

      BTHROW(1,MARK1,1) = 1
      BTHROW(2,MARK1-1,1) = 1
      GO TO 125

      ENDIF

      MARK1 = MARK1+1

C      *****
C      *
C      * IF THE POINTER FOR BTHROW IS NOT AT THE LAST DESCRIPTIVE ARRAY ELEMENT,
C      * THEN CONTINUE THE SEARCH FOR HOLES. IF IT IS, THEN BEGIN THE SEARCH
C      * FOR BODY BOUNDARY BIRTHS.
C      *
C      *****

      IF (MARK1 .LE. BNDCNT(LINE)) THEN
          GO TO 130
      ENDIF

      DO 140 MARK1 = 1,BNDCNT(LINE)

C      *****
C      *
C      * CHECK THE DESCRIPTIVE ARRAY ELEMENTS IN BTHROW TO DETERMINE IF ANY BODY
C      * BOUNDARIES ARE BORN. IF SO, THEN UPDATE THE DESCRIPTIVE ARRAYS NEWROW,
C      * BNDTRK, OBSTAT, AND TYPE.

```

```

C      *
C      *****

IF (BTHROW(1,MARK1,1) .EQ. 0 .AND. BTHROW (2,MARK1,1) .EQ. 0)
2  THEN

    OBJCNT = OBJCNT+1
    NEWROW(2,MARK1,2) = OBJCNT
    NEWROW(1,MARK1,2) = OBJCNT
    BNDTRK(1,OBJCNT,LINE) = NEWROW(1,MARK1,1)
    BNDTRK(2,OBJCNT,LINE) = NEWROW(2,MARK1,1)
    TYPE(OBJCNT) = BODY
    OBSTAT(OBJCNT) = UNDFD
    LFETRK(OBJCNT,1) = LINE
    PARCNT = PARCNT +1
    PARENT(PARCNT,1) = OBJCNT
    LBDTRK(PARCNT,1,1) = OBJCNT
    RBDTRK(PARCNT,1,1) = OBJCNT
    YNGROW(OBJCNT) = OBJCNT
    BNDORTE(OBJCNT) = CCM

ENDIF

140 CONTINUE

C      *****
C      *
C      * RETURN TO MAIN PROGRAM
C      *
C      *****

RETURN
END
SUBROUTINE BNDDTH

C      *****
C      *
C      * SUBROUTINE BNDDTH
C      *
C      * WRITTEN BY: EDWARD C. DE METER
C      * DATE: 1-18-86
C      *
C      *
C      * PROGRAM DESCRIPTION
C      *
C      * THIS PROGRAM SCANS THROUGH THE ARRAY, DTHROW, TO DETERMINE IF
C      * OBJECT BOUNDARIES HAVE EITHER TERMINATED OR MERGED WITH OTHER
C      * BOUNDARIES. IF A BOUNDARY DEATH IS DETECTED, THE PROGRAM
C      * UPDATES ANY DESCRIPTIVE ARRAYS PERTAINING TO THAT OBJECT.
C      *
C      *
C      * SUBROUTINES USED
C      *
C      * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C      *
C      * 1. PARFND
C      * 2. RBDINT
C      * 3. LBDINT
C      * 4. RTECHG
C      * 5. XXSCH
C      * 6. RX2SCH
C      * 7. LX2SCH
C      * 8. OBCHK
C      * 9. BBCHK
C      * 10. HLETRM
C      *

```

```

C      *
C      *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER  OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER  BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER  TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER  CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER  MARK6,MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG
INTEGER  LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER  LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER  PNTNUM(20),LMARK(20),RMARK(20),BNDRT(40),ROUTE(200,5)
INTEGER  CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER  ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL      AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL      AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL      PNTTRK(20,250,2),PERIM(20)
REAL      SDSTAT(20,50),PNSTAT(20,50)
REAL      XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL      A,B,THETA,PHI,MAG,SIGMA

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRT,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

C      *****
C      *
C      * COPY THE CONTENTS OF DTHROW INTO BUFROW AND CONSTRUCT BNDROW.
C      *
C      *****

DO 10 MARK1 = 1,BNDCNT(LINE-1)

BUFROW(1,MARK1,1) = DTHROW(1,MARK1,1)
BUFROW(1,MARK1,2) = DTHROW(1,MARK1,2)
BUFROW(2,MARK1,1) = DTHROW(2,MARK1,1)
BUFROW(2,MARK1,2) = DTHROW(2,MARK1,2)

I = OLDROW(1,MARK1,2)
BNDROW(I,1) = MARK1
I = OLDROW(2,MARK1,2)

```

```

      BNDROW(I,2) = MARK1
10  CONTINUE

20  MARK1 = 1

C *****
C *
C * CHECK THE DESCRIPTIVE ARRAY ELEMENTS OF DTHROW TO DETERMINE IF EITHER
C * A HOLE TERMINATION OR BODY MERGE HAS TAKEN PLACE
C *
C *****

30  IF (DTHROW(2,MARK1,1) .GT. 0 .AND. DTHROW(1,MARK1,1) .EQ. 0
      2   .AND. MARK1 .NE. 1 .AND. DTHROW(2,MARK1-1,1) .EQ. 0) THEN

C *****
C *
C * CHECK TO SEE IF THE TERMINATING BOUNDARY BELONGS TO A BODY
C *
C *****

      I = OLDROW(1,MARK1,2)

      IF (TYPE(I) .EQ. BODY) THEN

40      MARK2 = MARK1-1
50      J = OLDROW(2,MARK2,2)

C *****
C *
C * IF THE NEAREST CONTINUING BOUNDING OBJECT IS A HOLE WHICH ENCLOSES THE
C * BODY, THEN TERMINATE THE BODY AS WELL AS ALL TERMINABLE BODIES BETWEEN
C * THE CONTINUING HOLE BOUNDARY AND THE RIGHT SIDE OF THE BODY. IN
C * ADDITION, EXTEND THE BOUNDARY OF THE HOLE TO THE RIGHT SIDE OF THE BODY.
C *
C *****

60      IF ((TYPE(J) .EQ. HOLE .AND. BNDROW(J,1) .GT. BNDROW(I,2))
      2   .OR. (DTHROW(2,MARK2,1) .EQ. 2)) THEN

          DTHROW(2,MARK2,1) = 1
          GO TO 70

      END IF

      GO TO 90

70      MARK3 = MARK2+1

C *****
C *
C * THE FOLLOWING LOGIC CHECKS ALL THE OBJECTS BETWEEN THE TERMINATING
C * BODY AND THE FINAL TERMINATING POINT. ALL MERGING BODIES ARE
C * TERMINATED AND THE DESCRIPTIVE ARRAYS ARE UPDATED.
C *
C *****

```

```

80      K = OLDROW(1,MARK3,2)

      IF (DTHROW(1,MARK3,1) .EQ. 0 .AND. TYPE(K) .EQ. BODY) THEN

        LFETRK(K,2) = LINE
        OBSTAT(K) = CPLTE
        DTHTYP(K) = 0
        YNGROW(K) = J
        PARNT = J
        OBJECT = K
        CALL PARFND
        L = BNDROW(K,2)
        DTHROW(1,MARK3,1) = 1
        DTHROW(2,L,1) = 1
        MARK3 = L+1
        FLAG1 = J
        FLAG2 = K
        CALL RBDINT
        CALL RTECHG

        IF (MARK3 .GT. BNDROW(I,2)) THEN

          IF (BNDTRK(2,I,LINE) .EQ. 514) THEN

            M = BNDROW(I,2)
            OLDROW(2,M,2) = J

            IF (M+1 .LT. BNDCNT(LINE-1) .AND. DTHROW(1,M+1,1)
2              .EQ. 0) THEN

              DTHROW(2,M,1) = 2

              ENDIF

              GO TO 20

            ELSE

              M = BNDROW(I,2)
              N = DTHROW(2,M,2)
              BNDTRK(2,J,LINE) = BNDTRK(2,I,LINE)
              BNDTRK(2,I,LINE) = 514
              BNDROW(J,2) = M
              NEWROW(2,N,2) = J
              GO TO 20

            ENDIF

          ENDIF

        ELSE

          MARK3 = MARK3+1
          IF (MARK3 .LE. BNDROW(I,2)) THEN
            GO TO 80

          ELSE

            GO TO 20

          ENDIF

        ENDIF

      ENDIF

90      J = OLDROW(1,MARK2,2)

```

```

      K = BNDROW(J,2)
      JJ = OLDROW(2,MARK2,2)
      KK = BNDROW(JJ,1)

C *****
C *
C * IF THE NEAREST CONTINUING BOUNDARY IS A BODY, THEN TERMINATE THE BODY
C * AS WELL AS ALL MERGING BODIES BETWEEN THE RIGHT SIDE OF THE CONTINUING
C * BODY AND THE LEFT SIDE OF THE TERMINATING BODY. IN ADDITION, EXTEND
C * THE BOUNDARY OF THE CONTINUING BODY TO THE RIGHT SIDE OF THE
C * TERMINATING BODY.
C *
C *****

      IF (DTHROW(1,MARK2,1) .EQ. 1 .AND. DTHROW(2,K,1) .EQ. 0)
2      THEN

          MARK3 = K+1
          DTHROW(2,K,1) = 1
          GO TO 80

      ENDIF

C *****
C *
C * IF THE OBJECT WITH MERGING RIGHT BOUNDARY IS A BODY, WHOSE LEFT SIDE
C * IS MERGING WITH A HOLE, THEN TERMINATE THE BODY WITH THE MERGING LEFT
C * SIDE, AND EXTEND THE RIGHT SIDE OF THE CONTINUING BODY.
C *
C *****

      IF (TYPE(JJ) .EQ. BODY .AND. DTHROW(1,KK,1) .EQ. 0 .AND.
2      DTHROW(2,KK,1) .EQ. 0 .AND. KK-1 .GT. 1 .AND.
3      DTHROW(2,KK-1,1) .EQ. 1) THEN

          MARK3 = MARK2
          DTHROW(2,MARK2,1) = 1
          GO TO 80

      ENDIF

      MARK2 = MARK2-1
      GO TO 50

    ELSE

      DTHROW(1,MARK1,1) = 1
      MARK2 = MARK1-1

100      J = OLDROW(2,MARK2,2)
          KK = BNDROW(JJ,1)

C *****
C *
C * SINCE THE TERMINATING BOUNDARY DOES NOT BELONG TO A BODY IT MUST BELONG
C * TO A HOLE.
C *
C *****

C *****
C *
C * IF THE NEAREST TERMINATING BOUNDARY BELONGS TO THE SAME HOLE, THEN
C * SIMPLY TERMINATE THE HOLE.
C *
C *****

```

```

      IF (J .EQ. I) THEN
        LFETRK(I,2) = LINE
        OBSTAT(I) = CMLPTE
        DHTYP(I) = 1
        MARK3 = MARK2+1
        DTHROW(2,MARK2,1) = 1
        GO TO 110

      ENDIF

C *****
C *
C * IF THE NEAREST RIGHT BOUNDARY IS A CONTINUATION OF A BODY, THEN
C * TERMINATE THE BODY AND EXTEND THE LEFT SIDE OF THE HOLE TO THE LEFT
C * SIDE OF THE TERMINATING BODY.
C *
C *****

      IF (DTHROW(2,MARK2,1) .EQ. 2 .AND. J .NE. I) THEN

        K = BNDROW(J,1)
        BNDTRK(1,I,LINE) = BNDTRK(1,J,LINE)
        BNDTRK(1,J,LINE) = 514
        L = DTHROW(1,K,2)
        NEWROW(1,L,2) = I
        DTHROW(2,MARK2,1) = 1
        OBSTAT(J) = CMLPTE
        LFETRK(J,2) = LINE
        DHTYP(J) = 0
        YNGROW(J) = I
        PARNT = I
        OBJCT = J
        CALL PARFND
        FLAG1 = I
        FLAG2 = J
        CALL LBDINT
        CALL RTECHG
        BNDROW(I,1) = BNDROW(J,1)
        GO TO 20

      ENDIF

C *****
C *
C * IF THE NEAREST CONTINUING BOUNDARY BELONGS TO A BODY, THEN
C * TERMINATE THE BODY AS WELL AS ALL MERGING BODIES BETWEEN THE
C * BODY TO BE TERMINATED AND THE LEFT SIDE OF THE HOLE. IN ADDITION,
C * EXTEND THE LEFT SIDE OF THE HOLE TO THE LEFT SIDE OF THE BODY TO
C * BE TERMINATED.
C *
C *****

      MM = OLDROW(1,MARK2,2)
      K = BNDROW(MM,2)

      IF (DTHROW(1,MARK2,1) .EQ. 1 .AND. DTHROW(2,K,1) .EQ. 0)
2      THEN

        BNDTRK(1,I,LINE) = BNDTRK(1,MM,LINE)
        BNDTRK(1,MM,LINE) = 514
        L = DTHROW(1,MARK2,2)
        NEWROW(1,L,2) = I
        BNDROW(I,1) = MARK2
        DTHROW(2,K,1) = 1

```



```

        LFETRK(MM,2) = LINE
        OBSTAT(MM) = CMLPTE
        DHTYP(MM) = 0
        YNGROW(MM) = I

        PARNT = I
        OBJECT = MM
        CALL PARFND
        FLAG1 = I
        FLAG2 = MM
        CALL LBDINT
        CALL RTECHG
        MARK3 = MARK2+1
        GO TO 110

    ENDIF

    MARK2 = MARK2-1
    GO TO 100

110    IF (MARK3 .EQ. MARK1) THEN

        GO TO 20

    ELSE

C      *****
C      *
C      * THE FOLLOWING LOGIC CHECKS ALL THE OBJECTS BETWEEN THE TERMINATING
C      * BODY AND THE FINAL TERMINATING POINT. ALL MERGING BODIES ARE
C      * TERMINATED AND THE DESCRIPTIVE ARRAYS ARE UPDATED.
C      *
C      *****

120        K = OLDROW(1,MARK3,2)

        IF (DTHROW(1,MARK3,1) .EQ. 0 .AND. TYPE(K) .EQ. BODY)
2          THEN

            LFETRK(K,2) = LINE
            OBSTAT(K) = CMLPTE
            DHTYP(K) = 0
            YNGROW(K) = I

            PARNT = I
            OBJECT = K
            CALL PARFND
            L = BNDROW(K,2)
            DTHROW(1,MARK3,1) = 1
            DTHROW(2,L,1) = 1
            MARK3 = L+1
            FLAG1 = I
            FLAG2 = K
            CALL LBDINT
            CALL RTECHG
            GO TO 110

        ENDIF

        MARK3 = MARK3+1
        IF (MARK3 .LE. MARK1) THEN

            GO TO 120
        ENDIF

```

```

        ENDIF
    ENDIF
ENDIF

```

```

MARK1 = MARK1+1

```

```

IF (MARK1 .LE. BNDcnt(LINE-1)) THEN

```

```

    GO TO 30

```

```

ENDIF

```

```

C *****
C *
C * THE FOLLOWING SECTION OF CODE CHECKS THE DESCRIPTIVE ARRAY ELEMENTS OF
C * BUFRW TO DETERMINE IF EITHER A HOLE HAS MERGED OR A BODY HAS
C * TERMINATED.
C *
C *****

```

```

HLFLG = 0
OBCNT = 0

```

```

DO 130 MARK1 = 1,BNDcnt(LINE-1)

```

```

C *****
C *
C * THE FOLLOWING CODE COMPARES THE ARRAY ELEMENTS OF BUFRW TO DTHROW
C * TO DETERMINE WHICH ARRAY ELEMENTS HAVE BEEN MODIFIED BY THE PREVIOUS
C * SECTION OF CODE. IT THEN MODIFIES THE ARRAY ELEMENTS OF BUFRW
C * INORDER TO PREPARE THE ARRAY FOR THE SUBSEQUENT SEARCH FOR MERGING
C * HOLES OR TERMINATING BODIES.
C *
C *****

```

```

IF (DTHROW(1,MARK1,1) .GT. 0 .AND. DTHROW(2,MARK1,1) .GT. 0) THEN

```

```

2   IF (BUFRW(1,MARK1,1) .EQ. 1 .AND. BUFRW(2,MARK1,1) .EQ. 0)
    THEN

```

```

        BUFRW(2,MARK1,1) = 2

```

```

    ENDIF

```

```

2   IF (BUFRW(1,MARK1,1) .EQ. 0 .AND. BUFRW(2,MARK1,1) .EQ. 1)
    THEN

```

```

        BUFRW(1,MARK1,1) = 2

```

```

    ENDIF

```

```

2   IF (BUFRW(1,MARK1,1) .EQ. 0 .AND. BUFRW(2,MARK1,1) .EQ. 0)
    THEN

```

```

        BUFRW(1,MARK1,1) = 1
        BUFRW(2,MARK1,1) = 1

```

```

    ENDIF

```

```

ENDIF

```

```

I = OLDROW(1,MARK1,2)

```

```

C *****
C *
C * IF THE LEFT BOUNDARY REPRESENTED BY THE ARRAY ELEMENT IN OLDROW
C * BELONGS TO A BODY, THEN STORE THE BOUNDARY LOCATIONS OF THE BODY IN
C * OBJROW AND INCREMENT OBCNT.
C *
C *****

IF (TYPE(I) .EQ. BODY) THEN

    OBCNT = OBCNT+1
    OBJROW(OBCNT,1) = MARK1
    OBJROW(OBCNT,2) = BNDROW(I,2)

ENDIF

C *****
C *
C * IF AN ARRAY ELEMENT IN DTHROW EQUALS TO 0, THEN SET HLFLG EQUAL TO 1.
C *
C *****

IF (DTHROW(1,MARK1,1) .EQ. 0) THEN

    HLFLG = 1

ENDIF

130 CONTINUE

C *****
C *
C * IF HLFLG IS EQUAL TO 0 AFTER SCANNING THROUGH DTHROW, THEN THERE ARE
C * NO MERGING HOLES OR TERMINATING BODIES REPRESENTED IN BUFROW.  THUS
C * EXIT THE ROUTINE AND RETURN TO THE MAIN PROGRAM.
C *
C *****

IF (HLFLG .EQ. 0) THEN

    GO TO 250

ENDIF

C *****
C *
C * THIS OUTER DO LOOP RESULTS IN THE ANALYSIS OF EACH BODY IDENTIFIED BY
C * THE PREVIOUS CODE.
C *
C *****

DO 240 MARK1 = 1, OBCNT

C *****
C *
C * IF THE BODY IS A SIMPLE ONE WITH NO HOLES OR OFFSPRING BODIES, THEN
C * SIMPLY TERMINATE THE HOLE AND UPDATE THE DESCRIPTIVE ARRAYS.
C *
C *****

IF (OBJROW(MARK1,1) .EQ. OBJROW(MARK1,2)) THEN

    K = OBJROW(MARK1,1)

    IF (BUFROW(1,K,1) .EQ. 0) THEN

```

```

        L = OLDROW(1,K,2)
        LFETR(L,2) = LINE
        OBSTAT(L) = CMLPTE
        DHTYP(L) = 1

    ENDIF

    GO TO 240

ENDIF

BLKFLG = 0

C *****
C *
C * CHECK ALL OF THE DESCRIPTIVE ELEMENTS OF BUFROW THAT ARE WITHIN THE
C * BOUNDARIES OF THE OBJECT IN QUESTION. IF THE LEFT AND RIGHT ELEMENTS
C * ARE EQUAL TO ZERO, THEN SET BLKFLG EQUAL TO 1.
C *
C *****

DO 140 MARK2 = OBJROW(MARK1,1),OBJROW(MARK1,2)

IF (BUFROW(1,MARK2,1) .EQ. 0 .AND. BUFROW(2,MARK2,1) .EQ. 0) THEN

    BLKFLG = 1

ENDIF

140 CONTINUE

C *****
C *
C * IF BLKFLG IS EQUAL TO 0 AFTER SCANNING THROUGH THE BODY, THEN THAT BODY
C * CONTAINS NO MERGING HOLES AND IT DOES NOT TERMINATE.
C *
C *****

IF (BLKFLG .EQ. 0) THEN

    GO TO 240

ENDIF

MARK2 = OBJROW(MARK1,1)+1
MARK3 = MARK2-1

150 MARK5 = MARK2

C *****
C *
C * CONDUCT A SEARCH OF BUFROW STARTING FROM THE ARRAY ELEMENT SUCCEEDING
C * THE ARRAY ELEMENT REPRESENTING THE LEFT BOUNDARY OF THE OBJECT.
C * THE SEARCH IS FOR THE LEFT SIDE OF A MERGING HOLE.
C *
C *****

151 CALL XXSCH

C *****
C *
C * IF THE SEARCH DOES NOT FIND THE LEFT SIDE OF A MERGING HOLE, THEN CHECK
C * THE RIGHT BOUNDARY OF THE BODY FOR ANOTHER POSSIBLE SET OF MERGING
C * CONDITIONS.

```

```

C      *
C      *****

      IF (XXFLG .EQ. 0) THEN

          GO TO 220

      ENDIF

C      *****
C      *
C      * IF THE POINTER FROM XXSCH EQUALS EITHER THE LEFT OR RIGHT BOUNDARIES
C      * OF THE OBJECT, THEN CONDUCT A SEARCH FOR THE RIGHT SIDE OF A MERGING
C      * HOLE.
C      *
C      *****

      IF (MARK5 .EQ. OBJROW(MARK1,1) .OR. MARK5 .EQ. OBJROW(MARK1,2))
2      THEN

          GO TO 155

      ENDIF

C      *****
C      *
C      * CHECK TO SEE IF THE BOUNDARY LOCATED BY XXSCH DOES NOT BELONG TO
C      * ANOTHER BODY. IF IT DOES, THEN CONTINUE THE SEARCH FOR THE LEFT
C      * SIDE OF A MERGING HOLE.
C      *
C      *****

      CALL OBCHK

      IF (OBFLG .EQ. 1) THEN

          I = OLDROW(1,MARK5,2)
          MARK2 = BNDROW(I,2)+1

          IF (MARK2 .GT. OBJROW(MARK1,2)) THEN

              GO TO 220

          ENDIF

          GO TO 150

      ENDIF

155 MARK2 = MARK5
      I = OLDROW(1,MARK2,2)
      MARK5 = MARK5-1

C      *****
C      *
C      * CONDUCT A SEARCH FOR THE RIGHT SIDE OF A MERGING HOLE.
C      *
C      *****

160 CALL RX2SCH

C      *****
C      *
C      * IF NO RIGHT SIDE OF A MERGING HOLE IS FOUND, THEN TERMINATE THE HOLE
C      * AS WELL AS ANY OTHER MERGING HOLES BETWEEN THE LEFT SIDE OF THE BODY

```

```

C      * AND THE LEFT SIDE OF THE LOCATED MERGING HOLE.  IN ADDITION, EXTEND
C      * THE LEFT SIDE OF THE BODY TO THE LOCATION OF THE LEFT SIDE OF THE
C      * LOCATED MERGING HOLE.
C      *
C      *****

      IF (RX2FLG .EQ. 0) THEN

          GO TO 190

      ELSE

          IF (MARK5 .EQ. MARK3) THEN

              GO TO 180

          ENDIF

C      *****
C      *
C      * CHECK TO SEE IF THE SUSPECTED RIGHT SIDE OF A MERGING HOLE IS REALLY
C      * THE BOUNDARY OF ANOTHER BODY.  IF IT IS, THEN CONTINUE THE SEARCH.
C      *
C      *****

      CALL OBCHK

      IF (OBFLG .EQ. 1) THEN

          K = OLDROW(2,MARK5,2)
          MARK5 = BNDROW(K,1)-1

          IF (MARK5 .GE. MARK3) THEN

              GO TO 160

          ELSE

              GO TO 190

          ENDIF

      ENDIF

C      *****
C      *
C      * CHECK TO SEE IF THERE ARE ANY DESCRIPTIVE ARRAY ELEMENTS BETWEEN THE
C      * BOUNDARIES OF THE MERGING HOLES WHICH ARE EQUAL TO 0.
C      *
C      *****

180    CALL BBCHK

C      *****
C      *
C      * IF NO ARRAY ELEMENTS EQUAL TO 0 ARE FOUND, THEN CHECK THE RIGHT
C      * BOUNDARY OF THE OBJECT FOR A NEW SET OF MERGING CONDITIONS OR CONTINUE
C      * FOR THE SEARCH FOR A NEW LEFT BOUNDARY OF A MERGING HOLE.
C      *
C      *****

      IF (BBFLG .EQ. 0) THEN

          MARK5 = MARK2+1

          IF (MARK5 .GT. OBJROW(MARK1,2)) THEN

```

```

        GO TO 220

    ELSE

        GO TO 151

    ENDIF

ELSE

    GO TO 200

ENDIF

ENDIF

190 OBSTAT(I) = CMPLTE
    LFETRK(I,2) = LINE
    DTHTYP(I) = 0
    K = OBJROW(MARK1,1)
    L = OLDROW(1,K,2)
    BNDTRK(1,L,LINE) = BNDTRK(1,I,LINE)
    BNDTRK(1,I,LINE) = 514
    M = BUFROW(1,MARK2,2)
    NEWROW(1,M,2) = L
    MARK6 = K
    MARK7 = MARK2
    HLEFLG = 0
    FLAG1 = L
    FLAG3 = 0
    CALL HLETRM
    GO TO 210

200 OBSTAT(I) = CMPLTE
    LFETRK(I,2) = LINE
    DTHTYP(I) = 0
    K = OLDROW(2,MARK5,2)
    L = BNDROW(K,1)
    BNDTRK(1,K,LINE) = BNDTRK(1,I,LINE)
    BNDTRK(1,I,LINE) = 514
    M = BUFROW(1,MARK2,2)
    NEWROW(1,M,2) = K
    MARK6 = L+1
    MARK7 = MARK2
    PARNT = K
    HLEFLG = 1
    FLAG1 = K
    FLAG3 = 0
    CALL HLETRM

210 MARK5 = MARK2+1

    IF (MARK5 .GT. OBJROW(MARK1,2)) THEN

        GO TO 220

    ENDIF

C *****
C *
C * CONDUCT A LEFTHAND SEARCH FOR THE RIGHT SIDE OF THE NEXT MERGING HOLE.
C *
C *****

215 CALL LX2SCH

```

```

C *****
C *
C * IF NO RIGHT SIDES OF MERGING HOLES ARE FOUND, THEN CHECK THE RIGHT
C * BOUNDARY OF THE BODY FOR A NEW SET OF MERGING CONDITIONS.
C *
C *****

IF (LX2FLG .EQ. 0) THEN

    GO TO 220

ENDIF

C *****
C *
C * IF THE POINTER FROM LX2SCH POINTS TO EITHER OF THE BODY'S BOUNDARIES
C * THEN EITHER RESET THE POINTER OR CHECK THE RIGHT BOUNDARY OF THE
C * BODY FOR ANOTHER SET OF MERGING HOLE CONDITIONS.
C *
C *****

IF (MARK5 .EQ. OBJROW(MARK1,1) .OR. MARK5 .EQ. OBJROW(MARK1,2))
2 THEN

    GO TO 216

ENDIF

C *****
C *
C * CHECK TO SEE IF THE ARRAY ELEMENT FOUND IN LX2SCH IS NOT THE BOUNDARY
C * OF ANOTHER OBJECT.
C *
C *****

CALL OBCHK

C *****
C *
C * IF THE ARRAY ELEMENT IS THE BOUNDARY OF ANOTHER OBJECT, THEN EITHER
C * CONTINUE THE SEARCH OR CHECK THE RIGHT BOUNDARY OF THE BODY FOR A
C * NEW SET OF MERGING HOLE CONDITIONS.
C *
C *****

IF (OBFLG .EQ. 1) THEN

    K = OLDROW(1,MARK5,2)
    MARK5 = BNDROW(K,2)+1

    IF (MARK5 .GT. OBJROW(MARK1,2)) THEN

        GO TO 220

    ELSE

        GO TO 215

    ENDIF

ENDIF

216 MARK3 = MARK5
    MARK2 = MARK5+1

```



```
IF (MARK2 .GT. OBJROW(MARK1,2)) THEN
```

```
GO TO 220
```

```
ELSE
```

```
GO TO 150
```

```
ENDIF
```

```
C *****
C *
C * THE FOLLOWING CODE LOOKS AT THE RIGHT BOUNDARY OF THE OBJECT. IF THE
C * RIGHT BOUNDARY OF THE OBJECT DOES NOT CONTINUE, THEN THIS CODE WILL
C * EITHER CAUSE THE SIMPLE TERMINATION OF THE ENTIRE BODY, OR IT WILL
C * EXTEND THE RIGHT BOUNDARY OF THE BODY TO THE RIGHT SIDE OF THE
C * NEAREST CONTINUING MERGING HOLE. IF THE RIGHT BOUNDARY DOES CONTINUE,
C * THEN CONTROL IS RETURNED TO THE OUTER LOOP.
C *
C *****
```

```
220 K = OBJROW(MARK1,2)
```

```
L = OLDROW(2,K,2)
```

```
IF (BUFROM(2,K,1) .EQ. 0) THEN
```

```
MARK2 = K
```

```
MARK5 = MARK2
```

```
MARK3 = OBJROW(MARK1,1)
```

```
C *****
C *
C * CONDUCT A RIGHT HAND SEARCH FOR THE CONTINUING RIGHT SIDE OF A MERGING
C * HOLE.
C *
C *****
```

```
225 CALL RX2SCH
```

```
C *****
C *
C * IF NO CONTINUING RIGHT SIDE OF A MERGING HOLE IS FOUND, THEN SIMPLY
C * TERMINATE THE BODY AND UPDATE THE DESCRIPTIVE ARRAYS.
C *
C *****
```

```
IF (RX2FLG .EQ. 0) THEN
```

```
LFETRK(L,2) = LINE
```

```
OBSTAT(L) = CMLTE
```

```
DHTYP(L) = 1
```

```
MARK6 = MARK3
```

```
MARK7 = K
```

```
FLAG1 = L
```

```
FLAG3 = 0
```

```
HLEFLG = 0
```

```
CALL HLETRM
```

```
GO TO 240
```

```
ENDIF
```

```

C *****
C *
C * IF THE POINTER FROM RX2FLG IS EQUAL TO THE LEFT BOUNDARY OF THE
C * BODY, THEN CHECK TO SEE IF THERE ARE BLANK ARRAY ELEMENTS BETWEEN
C * THE BODY'S BOUNDARIES.
C *
C *****

      IF (MARK5 .EQ. MARK3) THEN

        GO TO 230

      ENDIF

      CALL OBCHK

C *****
C *
C * IF THE ARRAY ELEMENT FOUND IN RX2SCH POINTS TO THE BOUNDARY OF ANOTHER
C * OBJECT, THEN EITHER CONTINUE THE SEARCH OR TERMINATE THE BODY.
C *
C *****

      IF (OBFLG .EQ. 1) THEN

        L = OLDROW(2,MARK5,2)
        MARK5 = BNDROW(L,1)-1

        IF (MARK5 .GE. MARK3) THEN

          GO TO 225

        ELSE

          L = OLDROW(2,K,2)
          LFETRK(L,2) = LINE
          OBSTAT(L) = CMLPTE
          DTHTYP(L) = 1
          MARK6 = MARK3
          MARK7 = K
          FLAG1 = L
          FLAG3 = 0
          HLEFLG = 0
          CALL HLETRM
          GO TO 240

        ENDIF

      ENDIF

230    CALL BBCHK

C *****
C *
C * IF THERE ARE NO BLANK ARRAY ELEMENTS BETWEEN RIGHT BOUNDARY OF THE
C * BODY OR THE RIGHT SIDE OF THE MERGING HOLE, THEN RETURN CONTROL TO
C * THE OUTER LOOP.
C *
C *****

      IF (BBFLG .EQ. 0) THEN

        GO TO 240

      ENDIF

```

```

L = OLDROW(2,MARK5,2)
OBSTAT(L) = CMLTE
LFETRK(L,2) = LINE
DTHYP(L) = 0
M = OLDROW(2,K,2)
BNDTRK(2,M,LINE) = BNDTRK(2,L,LINE)
BNDTRK(2,L,LINE) = 514
N = BUFROW(2,MARK5,2)
NEWROW(2,N,2) = M
MARK6 = MARK5+1
MARK7=K
FLAG1 = M
FLAG3 = 1
HLEFLG = 0
CALL HLETRM

```

ENDIF

240 CONTINUE

```

C *****
C *
C * RETURN TO MAIN PROGRAM
C *
C *****

```

250 RETURN

END
SUBROUTINE PARFND

```

C *****
C *
C * SUBROUTINE PARFND
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-11-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM SEARCHES THROUGH THE ARRAY, PARENT INORDER TO SEE IF THE
C * OBJECT NUMBER STORED IN PARNT IS A HEADER. IF IT FINDS A MATCH, IT
C * ASSIGNS THE OBJECT NUMBER STORED IN OBJCT TO THAT ROW. IF IT DOES NOT
C * FIND A MATCH, IT CREATES A NEW ROW AND THEN ASSIGNS OBJCT TO THAT ROW.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT

```

```

INTEGER  ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL     AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL     AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL     PNTTRK(20,250,2),PERIM(20)
REAL     SDSTAT(20,50),PNSTAT(20,50)
REAL     XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL     A,B,THETA,PHI,MAG,SIGMA

COMMON   RESET,STPMEM,RECLL
COMMON   VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON   BNDTRK,SCNROW,ERROR,LINE
COMMON   DTHROW,BUFROW,BTHROW
COMMON   OLDROW,NEWROW,BNDROW
COMMON   OBJROW,PARENT,BNDCNT,LFETRK
COMMON   BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON   TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON   CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON   PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON   MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON   OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON   LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON   PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON   CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON   ANUM,BNUM,ATRK,BTRK
COMMON   AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON   AAREA,XXMNT,YYMNT,ABND,BBND
COMMON   PNTTRK,PERIM
COMMON   SDSTAT,PNSTAT
COMMON   XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON   A,B,THETA,PHI,MAG,SIGMA

```

```
PARSPT = 0
```

```

C *****
C *
C * SEARCH THROUGH THE ARRAY PARENT FOR THE OBJECT THAT HAS JUST GAINED
C * AN OFFSPRING.
C *
C *****

```

```
DO 10 PARPNT = 1,PARCNT
```

```

C *****
C *
C * IF THE OBJECT IS FOUND IN THE ARRAY PARENT, THEN ENTER THE OFFSPRING
C * INTO THE ARRAY PARENT.
C *
C *****

```

```
IF (PARENT(PARPNT,1) .EQ. PARNT) THEN
```

```

PARSPT = 1
OFSCNT(PARPNT) = OFSCNT(PARPNT)+1
PARENT(PARPNT,OFSCNT(PARPNT)+1) = OBJCT

```

```
ENDIF
```

```
10 CONTINUE
```

```

C *****
C *
C * IF THE OBJECT IS A NEW PARENT, THEN ENTER BOTH THE NEW PARENT AND THE
C * OFFSPRING INTO THE ARRAY, PARENT.
C *
C *****

```

```

      IF (PARSPT .EQ. 0) THEN

          PARCNT = PARCNT+1
          PARENT(PARCNT,1) = PARNT
          OFSCNT(PARCNT) = OFSCNT(PARCNT)+1
          PARENT(PARCNT,OFSCNT(PARCNT)+1) = OBJECT

      ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

      RETURN
      END

      SUBROUTINE XXSCH

C *****
C *
C * SUBROUTINE XXSCH
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-14-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CONDUCTS A SEARCH THROUGH THE ARRAY, BUFROW, FOR TWO
C * COLUMN ELEMENTS, THE FIRST HAVING A VALUE EQUAL TO ONE, THE SECOND
C * HAVING A VALUE GREATER TO OR EQUAL TO ONE. THE SEARCH IS CONDUCTED
C * FROM THE COLUMN NUMBER STORED IN MARK5 TO THE LAST COLUMN.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *
C *****

      INTEGER*2 RESET,STPMEM,RECLL
      INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
      INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
      INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
      INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
      INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETR(40,2)
      INTEGER BDOBJ(20),YNGROW(40),DTHYP(40),STACK(20,4),BNDST(40,2)
      INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
      INTEGER CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
      INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
      INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG
      INTEGER LX2FLG,OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
      INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
      INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNORTE(40),ROUTE(200,5)
      INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
      INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

      REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
      REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
      REAL PNTTRK(20,250,2),PERIM(20)
      REAL SDSTAT(20,50),PNSTAT(20,50)
      REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
      REAL A,B,THETA,PHI,MAG,SIGMA

      COMMON RESET,STPMEM,RECLL

```

```

COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRT,ROUTE
COMMON  CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```
XXFLG = 0
```

```

C *****
C *
C * IF THE LEFT ELEMENT IN BUFROW IS EQUAL TO 1 AND THE RIGHT ELEMENT IN
C * BUFROW IS GREATER THAN OR EQUAL TO 1, THEN SET XXFLG EQUAL TO 1 AND
C * RETURN.
C *
C *****

```

```
20 IF (BUFROW(1,MARK5,1) .EQ. 1 .AND. BUFROW(2,MARK5,1) .GE. 1) THEN
```

```

    XXFLG = 1
    GO TO 30

```

```
ENDIF
```

```
MARK5 = MARK5+1
```

```

C *****
C *
C * IF THE POINTER IS GREATER THAN THE BODY'S RIGHT BOUNDARY ARRAY ELEMENT,
C * THEN EXIT THE SUBROUTINE.
C *
C *****

```

```
IF (MARK5 .LE. OBJROW(OBCNT,2)) THEN
```

```
GO TO 20
```

```
ENDIF
```

```

C *****
C *
C * RETURN TO THE CALLING ROUTINE.
C *
C *****

```

```

30 RETURN
END

```

SUBROUTINE RX2SCH

```

C *****
C *
C * SUBROUTINE RX2SCH
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-14-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CONDUCTS A SEARCH THROUGH THE ARRAY, BUFROW, FOR TWO
C * COLUMN ELEMENTS, THE FIRST HAVING A VALUE GREATER TO OR EQUAL TO
C * ONE, THE SECOND HAVING A VALUE EQUAL TO ONE. THE SEARCH IS CONDUCTED
C * FROM THE COLUMN NUMBER STORED IN MARK5 BACK TOWARD THE FIRST COLUMN.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BND RTE(40),ROUTE(200,5)
INTEGER CCH,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BND RTE,ROUTE
COMMON CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND

```

```

COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```

RX2FLG = 0

```

```

C *****
C *
C * IF THE LEFT ARRAY ELEMENT OF BUFROW IS GREATER THAN OR EQUAL TO 1 AND
C * THE RIGHT ARRAY ELEMENT IS EQUAL TO 1, THEN SET RX2FLG EQUAL TO 1 AND
C * RETURN TO THE CALLING ROUTINE.
C *
C *****

```

```

40 IF (BUFROW(1,MARK5,1) .GE. 1 .AND. BUFROW(2,MARK5,1) .EQ. 1) THEN

```

```

    RX2FLG = 1
    GO TO 50

```

```

ENDIF

```

```

MARK5 = MARK5-1

```

```

C *****
C *
C * IF THE POINTER IS LESS THAN THE ARRAY ELEMENT REPRESENTING THE LEFT
C * BOUNDARY OF THE BODY, THEN RETURN TO THE CALLING ROUTINE.
C *
C *****

```

```

IF (MARK5 .GE. MARK3) THEN

```

```

    GO TO 40

```

```

ENDIF

```

```

C *****
C *
C * RETURN TO THE CALLING ROUTINE.
C *
C *****

```

```

50 RETURN
END

```

```

SUBROUTINE LX2SCH

```

```

C *****
C *
C * SUBROUTINE LX2SCH
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-14-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CONDUCTS A SEARCH THROUGH THE ARRAY, BUFROW, FOR TWO
C * COLUMN ELEMENTS, THE FIRST HAVING A VALUE EQUAL TO ONE, AND THE
C * SECOND HAVING A VALUE GREATER TO OR EQUAL TO ONE. THE SEARCH IS
C * CONDUCTED FROM THE COLUMN NUMBER STORED IN MARK5 TOWARD THE LAST
C * COLUMN.
C *

```



```

C      *
C      * SUBROUTINES USED: NONE
C      *
C      *
C      *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER  OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER  BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER  TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER  CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER  MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER  LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER  LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER  PNTNUM(20),LMARK(20),RMARK(20),BNDLTE(40),ROUTE(200,5)
INTEGER  CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER  ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL  AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL  AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL  PNTRK(20,250,2),PERIM(20)
REAL  SDSTAT(20,50),PNSTAT(20,50)
REAL  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL  A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON .RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDLTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

LX2FLG = 0

```

```

C      *****
C      *
C      * IF THE LEFT ARRAY ELEMENT OF BUFROW IS GREATER THAN OR EQUAL TO 1 AND
C      * THE RIGHT ARRAY ELEMENT IS EQUAL TO 1, THEN SET LX2FLG EQUAL TO 1, AND
C      * RETURN TO THE CALLING ROUTINE.
C      *****

```

```

60 IF (BUFROW(1,MARK5,1) .GE. 1 .AND. BUFROW(2,MARK5,1) .EQ. 1) THEN

```

```

        LX2FLG = 1
        GO TO 70

ENDIF

C *****
C *
C * IF BOTH ELEMENTS OF BUFROW ARE EQUAL TO 0, THEN SET LX2FLG EQUAL TO
C * 1, AND RETURN TO THE CALLING ROUTINE.
C *
C *****

IF (BUFROW(1,MARK5,1) .EQ. 0 .AND. BUFROW(2,MARK5,1) .EQ. 0) THEN

        LX2FLG = 1
        GO TO 70

ENDIF

MARK5 = MARK5+1

IF (MARK5 .LE. OBJROW(OBCNT,2)) THEN

        GO TO 60

ENDIF

C *****
C *
C * RETURN TO THE CALLING ROUTINE.
C *
C *****

70 RETURN
END

SUBROUTINE OBCHK

C *****
C *
C * SUBROUTINE OBCHK
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-14-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CHECKS TO SEE WHETHER THE COLUMN NUMBER STORED
C * IN MARK5 REPRESENTS ONE OF THE BOUNDARIES IN THE ARRAY BUFROW THAT
C * DEFINE A SEPARATE BODY.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER  OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER  BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER  TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER  CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT

```

```

INTEGER  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER  MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER  LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER  LBDTRK( 20,10,2),RBDTRK( 20,10,2),LTCNT( 20),RTCNT( 20)
INTEGER  PNTNUM( 20),LMARK( 20),RMARK( 20),BNDRTE( 40),ROUTE( 200,5)
INTEGER  CCW,CW,UP,DOWN,CHGROW( 10,2),LEFT,RIGHT,RTEPNT
INTEGER  ANUM,BNUM,ATRK( 4,4),BTRK( 4,4)

```

```

REAL     AREA( 40),XMOMNT( 40),YMOMNT( 40),FLDWDH,ARATIO
REAL     AAREA( 40),XXMNT( 40),YYMNT( 40),ABND( 4,4,2),BBND( 4,4,2)
REAL     PNTTRK( 20,250,2),PERIM( 20)
REAL     SDSTAT( 20,50),PNSTAT( 20,50)
REAL     XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL     A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```
OBFLG = 0
```

```
DO 80 MARK4 = 1,OBCNT
```

```

C *****
C *
C * IF THE ARRAY ELEMENT, MARK5 IS EQUAL TO ONE OF THE DEFINING BOUNDARY
C * ELEMENTS, THEN SET OBFLG EQUAL TO 1.
C *
C *****

```

```

IF (MARK5 .EQ. OBJROW(MARK4,1) .OR. MARK5 .EQ. OBJROW(MARK4,2))
2 THEN

```

```
OBFLG = 1
```

```
ENDIF
```

```
80 CONTINUE
```

```

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

```

RETURN
END

SUBROUTINE BBCHK

```

C *****
C *
C * SUBROUTINE BBCHK
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-14-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CONDUCTS A SEARCH THROUGH THE ARRAY, BUFROW, FOR TWO
C * COLUMN ELEMENTS WHICH HAVE VALUES EQUAL TO ZERO. THE SEARCH IS
C * CONDUCTED BETWEEN THE COLUMN NUMBERS STORED IN MARK5, AND MARK2.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL AREA(40),XMMNT(40),YMMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK

```

```

COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```
BBFLG = 0
```

```
DO 90 MARK4 = MARK5,MARK2
```

```

C *****
C *
C * IF THE LEFT AND RIGHT ELEMENTS IN BUFROW EQUAL TO 0, THEN SET BBFLG
C * EQUAL TO 1.
C *
C *****

```

```
IF (BUFROW(1,MARK4,1) .EQ. 0 .AND. BUFROW(2,MARK4,1) .EQ. 0) THEN
```

```
    BBFLG = 1
```

```
ENDIF
```

```
90 CONTINUE
```

```

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

```

```
RETURN
END
```

```
SUBROUTINE HLETRM
```

```

C *****
C *
C * SUBROUTINE HLETRM
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-14-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *     1. IF HLEFLG IS EQUAL TO ONE, IT CHECKS TO SEE IF THE OBJECT NUMBER
C *        STORED IN PARNT IS A HEADER IN THE ARRAY, PARENT. IF IT IS NOT,
C *        IT CREATES A NEW ROW IN PARENT.
C *
C *     2. IF HLEFLG IS EQUAL TO ZERO, IT TERMINATES ALL HOLES DESCRIBED BY
C *        THE ARRAY, BUFROW, BETWEEN THE COLUMN NUMBERS STORED IN MARK6
C *        AND MARK7. IN ADDITION, IT ASSIGNS THE HOLES AS OFFSPRING TO
C *        THE PARENT OBJECT IN THE ARRAY, PARENT. IN ADDITION, IT ALSO
C *        UPDATES SUCH DESCRIPTIVE ARRAYS SUCH AS LBDTRK AND RBDTRK.
C *
C *     3. IF HLEFLG IS EQUAL TO ONE, IT CONDUCTS THE OPERATIONS DESCRIBED
C *        IN 2. IN ADDITION, IT FINDS THE ORIGINAL PARENT OF EACH MERGING
C *        HOLE, AND ELIMINATES THE MERGING HOLE FROM THE PARENT'S GROUP
C *        OF OFFSPRING.
C *
C *

```

```

C      * SUBROUTINES USED
C      *
C      * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C      *
C      *      1. RBDINT
C      *      2. LBDINT
C      *      3. RTECHG
C      *      4. PARCHG
C      *
C      *
C      *****
C
C      INTEGER*2 RESET,STPMEM,RECLL
C      INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
C      INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
C      INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
C      INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
C      INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
C      INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
C      INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
C      INTEGER CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
C      INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
C      INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
C      INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
C      INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
C      INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
C      INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
C      INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
C
C      REAL AREA(40),XMOMNT(40),YOMMNT(40),FLDWDH,ARATIO
C      REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
C      REAL PNTTRK(20,250,2),PERIM(20)
C      REAL SDSTAT(20,50),PNSTAT(20,50)
C      REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
C      REAL A,B,THETA,PHI,MAG,SIGMA
C
C      COMMON RESET,STPMEM,RECLL
C      COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
C      COMMON BNDTRK,SCNROW,ERROR,LINE
C      COMMON DTHROW,BUFROW,BTHROW
C      COMMON OLDROW,NEWROW,BNDROW
C      COMMON OBJROW,PARENT,BNDCNT,LFETRK
C      COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
C      COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
C      COMMON CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
C      COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
C      COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
C      COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
C      COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
C      COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
C      COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
C      COMMON ANUM,BNUM,ATRK,BTRK
C      COMMON AREA,XMOMNT,YOMMNT,FLDWDH,ARATIO
C      COMMON AAREA,XXMNT,YYMNT,ABND,BBND
C      COMMON PNTTRK,PERIM
C      COMMON SDSTAT,PNSTAT
C      COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
C      COMMON A,B,THETA,PHI,MAG,SIGMA
C
C      *****
C      *
C      * IF HLEFLG IS EQUAL TO 1, THEN DETERMINE WHETHER THE PARENT HOLE IS
C      * ALREADY IN THE ARRAY, PARENT. IF IT ISN'T, THEN ASSIGN IT TO THAT
C      * ARRAY.

```

```

C      *
C      *****

      IF (HLEFLG .EQ. 1) THEN

          PARSPT = 0
          MARK4 = 1

100      IF (PARNT .EQ. PARENT(MARK4,1)) THEN

          PARSPT = 1
          MMM = MARK4

          ENDIF

          MARK4 = MARK4+1

          IF (MARK4 .LE. PARCNT) THEN

              GO TO 100

          ENDIF

          IF (PARSPT .EQ. 0) THEN

              PARCNT = PARCNT+1
              PARENT(PARCNT,1) = PARNT
              MMM = PARCNT

          ENDIF

      ENDIF

C      *****
C      *
C      * TERMINATE EVERY MERGING HOLE BETWEEN THE BOUNDARIES IN QUESTION.
C      * REASSIGN THE MERGING HOLES FROM THEIR ORIGINAL PARENT BODY TO THEIR NEW
C      * PARENT HOLE IF HLFLG IS EQUAL TO 1.
C      *
C      *****

      DO 110 MARK4 = MARK6, MARK7

      OBJECT = OLDROW(1, MARK4, 2)

      IF (TYPE(OBJECT) .EQ. HOLE) THEN

          OBSTAT(OBJECT) = CMLPTE
          LFETRK(OBJECT, 2) = LINE
          FLAG2 = OBJECT

          IF (FLAG3 .EQ. 1) THEN

              CALL RBDINT

          ELSE

              CALL LBDINT

          ENDIF

          CALL RTECHG

          IF (HLEFLG .EQ. 1) THEN

```

```

        OFSCNT(MMM) = OFSCNT(MMM)+1
        PARENT(MMM,OFSCNT(MMM)+1) = OBJECT

C *****
C *
C * CALL UPON PARCHG TO REASSIGN THE MERGING HOLE'S PARENTS.
C *
C *****

        CALL PARCHG

        ENDIF

    ENDIF

110 CONTINUE

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

    RETURN
    END

SUBROUTINE PARCHG

C *****
C *
C * SUBROUTINE PARCHG
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-16-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CONDUCTS A SEARCH THROUGH THE ARRAY, PARENT, FOR THE
C * PARENT OF THE OBJECT NUMBER STORED IN OBJECT. WHEN THE PARENT IS
C * FOUND, THE OFFSPRING IS ELIMINATED FROM THE OFFSPRING ROW IN THE
C * ARRAY, PARENT.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

    INTEGER*2 RESET,STPMEM,RECLL
    INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
    INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
    INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
    INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
    INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
    INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
    INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
    INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
    INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
    INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
    INTEGER LX2FLG,OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
    INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
    INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
    INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
    INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```



```

REAL    AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL    AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL    PNTTRK(20,250,2),PERIM(20)
REAL    SDSTAT(20,50),PNSTAT(20,50)
REAL    XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL    A,B,THETA,PHI,MAG,SIGMA

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DTHYTP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMPLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * FIND THE LOCATION OF THE HOLE'S PARENT IN THE ARRAY, PARENT.
C *
C *****

```

```

NN = YNGROW(OBJECT)
MARK8 = 1

```

```

120 IF (NN .EQ. PARENT(MARK8,1)) THEN

```

```

    GO TO 130

```

```

ENDIF

```

```

MARK8 = MARK8+1
GO TO 120

```

```

C *****
C *
C * FIND THE LOCATION OF THE OBJECT IN THE ARRAY, PARENT.
C *
C *****

```

```

130 DO 140 MARK9 = 1,OFSCNT(MARK8)

```

```

    IF (OBJECT .EQ. PARENT(MARK8,MARK9+1)) THEN

```

```

        MARK10 = MARK9

```

```

    ENDIF

```

```

140 CONTINUE

```

```

C *****
C *
C * SHIFT ALL OF THE PARENT'S OFFSPRING BACK ONE ARRAY ELEMENT, STARTING
C * FROM THE OBJECT'S POSITION.
C *
C *****

```

```
DO 150 MARK9 = MARK10,OFSCNT(MARK8)-1
```

```
PARENT(MARK8,MARK9+1) = PARENT(MARK8,MARK9+2)
```

```
150 CONTINUE
```

```
OFSCNT(MARK8) = OFSCNT(MARK8)-1
```

```

C *****
C *
C * RETURN TO THE CALLING ROUTINE.
C *
C *****

```

```
RETURN
END
```

SUBROUTINE STSCAN

```

C *****
C *
C * SUBROUTINE STSCAN
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-11-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *   1. COMPUTES THE INCREMENTAL AREA DEFINED BY ONE SCAN LINE AND
C *   TWO BOUNDARIES FOR EACH UNDEFINED OBJECT AND ADDS IT TO THE
C *   OBJECT'S TOTAL AREA,
C *
C *   2. COMPUTES THE INCREMENTAL X AREA MOMENT FOR EACH UNDEFINED
C *   OBJECT AND ADDS IT TO THE OBJECT'S TOTAL X AREA MOMENT, AND
C *
C *   3. COMPUTES THE INCREMENTAL Y AREA MOMENT FOR EACH UNDEFINED
C *   OBJECT AND ADDS IT TO THE OBJECT'S TOTAL Y AREA MOMENT.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *
C *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APPI,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT

```

```

INTEGER  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER  MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER  LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER  LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER  PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER  CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER  ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL     AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL     AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL     PNTTRK(20,250,2),PERIM(20)
REAL     SDSTAT(20,50),PNSTAT(20,50)
REAL     XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL     A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMPLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```
DO 160 MARK9 = 1,OBJCNT
```

```

C *****
C *
C * IF THE OBJECT IS UNDEFINED, THEN COMPUTE ITS INCREMENTAL AREA AND X
C * AND Y MOMENT OF INERTIA AND ADD THEM TO THE TOTAL VALUES.
C *
C *****

```

```
IF (OBSTAT(MARK9) .EQ. UNDFD) THEN
```

```

X = REAL(BNDTRK(1,MARK9,LINE))
Y = REAL(BNDTRK(2,MARK9,LINE))
R = REAL (LINE)

```

```
A = (((Y-X+1.0)*FLDWDH**2)/(ARATIO*512.0*480.0))
```

```
AREA(MARK9) = AREA(MARK9)+A
```

```

D = (((Y+X)*0.5)-1.0)*1.95313E-3 * FLDWDH
D = (D + (FLDWDH*9.76565E-4))*A
XMOMNT(MARK9) = XMOMNT(MARK9)+D

```

```

C = ((R*FLDWDH/(ARATIO*480.0))+(FLDWDH/(ARATIO*960.0)))* A
YMOMNT(MARK9) = YMOMNT(MARK9)+C

```

```
ENDIF
```

160 CONTINUE

```
C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****
```

RETURN
END

SUBROUTINE FNSTAT

```
C *****
C *
C * SUBROUTINE FNSTAT
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-11-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *   1. ADDS THE TOTAL AREA OF EACH OFFSPRING TO ITS PARENT,
C *
C *   2. ADDS THE TOTAL X AREA MOMENT OF EACH OFFSPRING TO ITS PARENT,
C *
C *   3. ADDS THE TOTAL Y AREA MOMENT OF EACH OFFSPRING TO ITS PARENT,
C *
C *   4. COMPUTES THE X AND Y CENTROID OF EACH OBJECT WITH RESPECT TO
C *       THE ORIGIN OF THE FIELD OF VIEW.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *
```

```
INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```
REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA
```

COMMON RESET,STPMEM,RECLL

```

COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * THE FOLLOWING CODE SCANS THROUGH THE ARRAY, PARENT, AND ASSIGNS THE
C * PARENT OF EACH OFFSPRING TO YNGROW.
C *
C *****

```

```
DO 166 MARK9 = 1,PARCNT
```

```
IF (OFSCNT(PARCNT) .GT. 0) THEN
```

```
DO 165 MARK8 = 1,OFSCNT(MARK9)
```

```
OBJECT =PARENT(MARK9,MARK8+1)
YNGROW(OBJECT) = PARENT(MARK9,1)
```

```
165 CONTINUE
```

```
ENDIF
```

```
166 CONTINUE
```

```
BDCNT = 0
```

```

C *****
C *
C * THE FOLLOWING CODE SCANS THROUGH YNGROW. IF THE OBJECT IS ITS OWN PARENT
C * THEN OBCNT IS INCREMENTED AND THE OBJECT IS ASSIGNED TO THE ARRAY,BDOBJ.
C *
C *****

```

```
DO 170 MARK9 = 1,OBJCNT
```

```
IF (YNGROW(MARK9) .EQ. MARK9) THEN
```

```
BDCNT = BDCNT+1
BDOBJ(BDCNT) = MARK9
```

```
ENDIF
```

```

C *****
C *
C * THE FOLLOWING CODE ASSIGNS THE EACH INDIVIDUAL OBJECT'S AREA, X AND Y
C * MOMENTS OF INERTIA TO AAREA, XXMNT, AND YYMNT RESPECTIVELY.

```

```

C      *
C      *****

      AAREA(MARK9) = AREA(MARK9)
      XXMNT(MARK9) = XMOMNT(MARK9)
      YYMNT(MARK9) = YMOMNT(MARK9)
      OBSTAT(MARK9) = UNDFD

170 CONTINUE

C      *****
C      *
C      * THE FOLLOWING CODE DETERMINES WHETHER AN OBJECT IS A PARENT.  IF IT ISNT
C      * THEN THAT OBJECT'S TOTAL AREA, X AND Y MOMENTS OF INERTIA ARE ADDED TO
C      * THE PARENT'S AREA, X AND Y MOMENTS OF INERTIA.
C      *
C      *****

      DO 190 MARK9 = 1,OBJCNT

      PARSPT = 0

      DO 180 MARK8 = 1,PARCNT

      IF (MARK9 .EQ. PARENT(MARK8,1)) THEN

        PARSPT = 1

      ENDIF

180 CONTINUE

      IF (PARSPT .EQ. 0) THEN

        PARNT = YNGROW(MARK9)
        AREA(PARNT) = AREA(PARNT)+AREA(MARK9)
        XMOMNT(PARNT) = XMOMNT(PARNT)+XMOMNT(MARK9)
        YMOMNT(PARNT) = YMOMNT(PARNT)+YMOMNT(MARK9)
        OBSTAT(MARK9) = CMLTE

      ENDIF

190 CONTINUE

C      *****
C      *
C      * THE FOLLOWING CODE SCANS THROUGH THE ARRAY PARENT, FINDS THE PARENTS
C      * WITH OFFSPRING, AND ADDS THEIR OFFSPRING'S AREA, AND X AND Y MOMENTS
C      * OF INERTIA TO THEIR RESPECTIVE AREAS, AND X AND Y MOMENTS OF INERTIA.
C      *
C      *****

      DO 210 MARK9 = 1,PARCNT

      IF (OFSCNT(MARK9) .GT. 0) THEN

        DO 200 MARK10 = 1,OFSCNT(MARK9)

        OBJCT = PARENT(MARK9,MARK10+1)

        IF (OBSTAT(OBJCT) .EQ. UNDFD) THEN

          PARNT = YNGROW(OBJCT)
          AREA(PARNT) = AREA(PARNT)+AREA(OBJCT)
          XMOMNT(PARNT) = XMOMNT(PARNT)+XMOMNT(OBJCT)
          YMOMNT(PARNT) = YMOMNT(PARNT)+YMOMNT(OBJCT)

```

```

        ENDIF

200    CONTINUE

        ENDIF

210    CONTINUE

C      *****
C      *
C      * THE FOLLOWING CODE COMPUTES THE CENTROIDAL X AND Y DISTANCE OF EACH
C      * OBJECT AND OBJECT SUM FROM THE LEFT CORNER OF THE FIELD OF VIEW.
C      *
C      *****

DO 220 MARK9 = 1,OBJCNT

XOMNT(MARK9) = ABS(XOMNT(MARK9)/AREA(MARK9))
XXMNT(MARK9) = ABS(XXMNT(MARK9)/AAREA(MARK9))
YOMNT(MARK9) = ABS(YOMNT(MARK9)/AREA(MARK9))
YYMNT(MARK9) = ABS(YYMNT(MARK9)/AAREA(MARK9))

220    CONTINUE

C      *****
C      *
C      * RETURN TO THE CALLING ROUTINE.
C      *
C      *****

RETURN
END
SUBROUTINE LBDINT

C      *****
C      *
C      * SUBROUTINE LBDINT
C      *
C      * WRITTEN BY: EDWARD C. DE METER
C      * DATE: 1-16-86
C      *
C      * PROGRAM DESCRIPTION
C      *
C      * THIS PROGRAM SEARCHES THROUGH THE ARRAY, LBDTRK, FOR THE OBJECT HEADER
C      * STORED IN FLAG1. IF THE HEADER IS FOUND, THEN THE PROGRAM ENTERS THE
C      * MERGING OFFSPRING INTO THE HEADER'S ROW. IF NO MATCH IS FOUND, THEN
C      * THE PROGRAM CREATES A NEW ROW IN LBDTRK AND ENTERS THE OFFSPRING INTO
C      * THE NEW ROW.
C      *
C      * SUBROUTINES USED: NONE
C      *
C      *
C      *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER  OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER  BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER  TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER  CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER  MARK6,MARK7,MARK8,MARK9,MARK10,OB_CNT,BLKFLG,XXFLG,RX2FLG

```

```

INTEGER  LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER  LBDTRK( 20,10,2),RBDTRK( 20,10,2),LTCNT( 20),RTCNT( 20)
INTEGER  PNTNUM( 20),LMARK( 20),RMARK( 20),BNDRTE(40),ROUTE(200,5)
INTEGER  CCW,CW,UP,DOWN,CHGROW( 10,2),LEFT,RIGHT,RTEPNT
INTEGER  ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL     AREA(40),XMOMNT(40),YOMMNT(40),FLDWDH,ARATIO
REAL     AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL     PNTTRK( 20,250,2),PERIM( 20)
REAL     SDSTAT( 20,50),PNSTAT( 20,50)
REAL     XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL     A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNROW,DHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLAG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBSCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```
PARSPT = 0
```

```

C *****
C *
C * SEARCH THROUGH THE ARRAY,LBDTRK FOR THE OBJECT THAT HAS JUST MERGED
C * WITH AN OFFSPRING.
C *
C *****

```

```
DO 10 PARPNT = 1,PARCNT
```

```

C *****
C *
C * IF THE OBJECT IS FOUND IN THE ARRAY, LBDTRK, THEN ENTER THE OFFSPRING
C * INTO THE ARRAY, LBDTRK.
C *
C *****

```

```
IF (LBDTRK(PARPNT,1,1) .EQ. FLAG1) THEN
```

```

PARSPT = 1
LTCNT(PARPNT) = LTCNT(PARPNT)+1
LBDTRK(PARPNT,LTCNT(PARPNT)+1,1) = FLAG2
LBDTRK(PARPNT,LTCNT(PARPNT)+1,2) = LINE

```

```
ENDIF
```

```
10 CONTINUE
```



```

C *****
C *
C * IF THE OBJECT IS A NEW PARENT, THEN ENTER BOTH THE NEW PARENT AND THE
C * OFFSPRING INTO THE ARRAY, LBDTRK.
C *
C *****

      IF (PARSPT .EQ. 0) THEN

          LBDTRK(PARCNT,1,1) = FLAG1
          LTCNT(PARCNT) = LTCNT(PARCNT)+1
          LBDTRK(PARCNT,LTCNT(PARCNT)+1,1) = FLAG2
          LBDTRK(PARCNT,LTCNT(PARCNT)+1,2) = LINE

      ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

      RETURN
      END

      SUBROUTINE RBDINT

C *****
C *
C * SUBROUTINE RBDINT
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-16-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM SEARCHES THROUGH THE ARRAY, LBDTRK, FOR THE OBJECT HEADER
C * STORED IN FLAG1. IF THE HEADER IS FOUND, THEN THE PROGRAM ENTERS THE
C * MERGING OFFSPRING INTO THE HEADER'S ROW. IF NO MATCH IS FOUND, THEN
C * THE PROGRAM CREATES A NEW ROW IN LBDTRK AND ENTERS THE OFFSPRING INTO
C * THE NEW ROW.
C *
C * SUBROUTINES USED: NONE
C *
C *****

      INTEGER*2 RESET,STPMEM,RECLL
      INTEGER*2 VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
      INTEGER*2 BNDTRK(2,40,0:479),SCNR0W(512),ERROR,LINE
      INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
      INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
      INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
      INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
      INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
      INTEGER CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
      INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
      INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
      INTEGER LX2FLG,OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
      INTEGER LBDTRK(2,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
      INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
      INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
      INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL    AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL    AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL    PNTTRK(20,250,2),PERIM(20)
REAL    SDSTAT(20,50),PNSTAT(20,50)
REAL    XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL    A,B,THETA,PHI,MAG,SIGMA

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```
PARSPT = 0
```

```

C *****
C *
C * SEARCH THROUGH THE ARRAY,RBDTRK FOR THE OBJECT THAT HAS JUST MERGED
C * WITH AN OFFSPRING.
C *
C *****

```

```
DO 10 PARPNT = 1,PARCNT
```

```

C *****
C *
C * IF THE OBJECT IS FOUND IN THE ARRAY, LBDTRK, THEN ENTER THE OFFSPRING
C * INTO THE ARRAY, RBDTRK.
C *
C *****

```

```
IF (RBDTRK(PARPNT,1,1) .EQ. FLAG1) THEN
```

```

PARSPT = 1
RTCNT(PARPNT) = RTCNT(PARPNT)+1
RBDTRK(PARPNT,RTCNT(PARPNT)+1,1) = FLAG2
RBDTRK(PARPNT,RTCNT(PARPNT)+1,2) = LINE

```

```
ENDIF
```

```
10 CONTINUE
```

```

C *****
C *
C * IF THE OBJECT IS A NEW PARENT, THEN ENTER BOTH THE NEW PARENT AND THE
C * OFFSPRING INTO THE ARRAY, RBDTRK.
C *
C *****

```

```
IF (PARSPT .EQ. 0) THEN
```

```
  RBDTRK(PARCNT,1,1) = FLAG1
  RTCNT(PARCNT) = RTCNT(PARCNT)+1
  RBDTRK(PARCNT,RTCNT(PARCNT)+1,1) = FLAG2
  RBDTRK(PARCNT,RTCNT(PARCNT)+1,2) = LINE
```

```
ENDIF
```

```
C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****
```

```
RETURN
END
```

```
SUBROUTINE LCNSRT
```

```
C *****
C *
C * SUBROUTINE LCNSRT
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 1-16-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM SEARCHES THROUGH THE ARRAY, PARENT, FOR THE OBJECT HEADER
C * STORED IN MARK10. WHEN THE HEADER IS FOUND, THE PROGRAM THEN SEARCHES
C * THROUGH ITS OFFSPRING FOR ANY UNDEFINED HOLES. IF A HOLE IS FOUND,
C * IT IS MERGED INTO THE PARENT AND THE NECESSARY DESCRIPTIVE ARRAYS ARE
C * UPDATED.
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C *   1. LBDINT
C *   2. RTECHG
C *
C *****
```

```
INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```

REAL    AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL    AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL    PNTTRK(20,250,2),PERIM(20)
REAL    SDSTAT(20,50),PNSTAT(20,50)
REAL    XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL    A,B,THETA,PHI,MAG,SIGMA

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEUROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMPLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```
MARK5 = 1
```

```

C *****
C *
C * LOCATE THE BODY IN THE ARRAY, PARENT. IF ANY OF ITS OFFSPRING ARE
C * HOLES AND ARE UNDEFINED, THEN TERMINATE THE HOLES AND UPDATE LBDTRK
C * AND BNDRTE.
C *
C *****

```

```
10 IF (MARK10 .EQ. PARENT(MARK5,1))THEN
```

```
    DO 20 MARK4 = 1,OFSCNT(MARK5)
```

```
    M = PARENT(MARK5,MARK4+1)
```

```
    IF (TYPE(M) .EQ. HOLE .AND. OBSTAT(M) .EQ. UNDFD) THEN
```

```
        FLAG1 = MARK10
```

```
        FLAG2 = M
```

```
        CALL LBDINT
```

```
        CALL RTECHG
```

```
    ENDIF
```

```
20 CONTINUE
```

```
    GO TO 30
```

```
ENDIF
```

```
MARK5 = MARK5+1
```

```
IF (MARK5 .LE. PARCNT) THEN
```

```

        GO TO 10

ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

30 RETURN
END

SUBROUTINE RTECHG

C *****
C *
C * SUBROUTINE RTECHG
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 2-17-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM COMPARES THE ROUTING OF A MERGING OFFSPRING WITH ITS
C * PARENT. IF THEY ARE DIFFERENT, THEN THE PROGRAM GOES THROUGH LBDTRK
C * AND RBDTRK, FINDS THE OFFSPRING, AND CHANGES THE ROUTING OF ALL OF THE
C * OFFSPRING'S OFFSPRING.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER MARK6,MARK7,MARK8,MARK9,MARK10,OB_CNT,BLKFLG,XXFLG,RX2FLG
INTEGER LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST

```

```

COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBSCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

MM = BNDRTE(FLAG2)
BNDRTE(FLAG2) = BNDRTE(FLAG1)

```

```

C *****
C *
C * IF THE BOUNDARY ROUTE OF THE OFFSPRING IS NOT THE SAME AS ITS PARENT
C * THEN SEARCH THROUGH LBDTRK FOR THE OFFSPRING AND CHANGE THE ROUTING
C * OF ALL OF ITS OFFSPRING.
C *
C *****

```

```

IF (MM .NE. BNDRTE(FLAG2)) THEN

  DO 20 PARPNT = 1,PARCNT

    IF (LBDTRK(PARPNT,1,1) .EQ. FLAG2) THEN

      DO 10 MMM = 1,LTCNT(PARPNT)

        NN = LBDTRK(PARPNT,LTCNT(PARPNT)+1,1)

        IF (BNDRTE(NN) .EQ. CW) THEN

          BNDRTE(NN) = CCW

        ELSE

          BNDRTE(NN) = CW

        ENDIF

      10 CONTINUE

    ENDIF

  20 CONTINUE

```

```

C *****
C *
C * IF THE BOUNDARY ROUTE OF THE OFFSPRING IS NOT THE SAME AS ITS PARENT
C * THEN SEARCH THROUGH RBDTRK FOR THE OFFSPRING AND CHANGE THE ROUTING
C * OF ALL OF ITS OFFSPRING.
C *
C *****

```

```

DO 40 PARPNT = 1,PARCNT

  IF (RBDTRK(PARPNT,1,1) .EQ. FLAG2) THEN

```

```

DO 30 MMM = 1, RTCNT(PARPNT)

NN = RBDTRK(PARPNT, RTCNT(PARPNT)+1, 1)

IF (BND RTE(NN) .EQ. CW) THEN

    BND RTE(NN) = CCW

ELSE

    BND RTE(NN) = CW

ENDIF

30    CONTINUE

ENDIF

40    CONTINUE

ENDIF

C    *****
C    *
C    * RETURN TO CALLING ROUTINE
C    *
C    *****

RETURN
END

SUBROUTINE PNTDEF

C    *****
C    *
C    * SUBROUTINE PNTDEF
C    *
C    * WRITTEN BY: EDWARD C. DE METER
C    * DATE: 2-19-86
C    *
C    * PROGRAM DESCRIPTION
C    *
C    * THIS PROGRAM DOES THE FOLLOWING:
C    *
C    *     1. DETERMINES THE ROUTING FOR AN OBJECT'S BOUNDARY,
C    *
C    *     2. SCANS THE OBJECT'S BOUNDARY, USING THREE BOUNDARY POINTS
C    *        FOR THE CREATION OF TWO BOUNDARY VECTORS,
C    *
C    *     3. ANALYZES THE RELATIVE DIRECTION OF BOTH BOUNDARY VECTORS AND
C    *        DETERMINES WHETHER A CORNER POINT HAS BEEN LOCATED,
C    *
C    *     4. FILTERS OUT THE POINTS THAT WERE MISTAKENLY RECORDED AS
C    *        CORNER POINTS, AND
C    *
C    *     5. COMPUTES THE DISTANCE OF EACH CORNER POINT FROM THE OBJECT'S
C    *        CENTROID AND THE LENGTH OF EACH OF THE OBJECT'S SIDES.
C    *
C    *
C    * SUBROUTINES USED
C    *
C    * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C    *
C    *     1. RERTE

```

```

C      *      2. RTEPLN
C      *      3. CLNUP1
C      *      4. CLNUP2
C      *      5. RTESTT
C      *
C      *
C      *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2  BNDTRK( 2,40,0:479),SCNROW( 512),ERROR,LINE
INTEGER*2  DTHROW( 2,100,2),BUFROW( 2,100,2),BTHROW( 2,100,2)
INTEGER*2  OLDROW( 2,100,2),NEWROW( 2,100,2),BNDROW(40,2)
INTEGER    OBJROW(40,2),PARENT( 20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER    BDOBJ( 20),YNGROW(40),DHTYP(40),STACK( 20,4),BNDST(40,2)
INTEGER    TYPE(40),OBSTAT(40),OFSCNT( 20),HOLE,BODY,UNDFD,BDCNT
INTEGER    CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER    PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5
INTEGER    MARK6,MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG
INTEGER    LX2FLG,OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER    LBDTRK( 20,10,2),RBDTRK( 20,10,2),LTCNT( 20),RTCNT( 20)
INTEGER    PNTNUM( 20),LMARK( 20),RMARK( 20),BNDRTE(40),ROUTE( 200,5)
INTEGER    CCW,CW,UP,DOWN,CHGROW( 10,2),LEFT,RIGHT,RTEPNT
INTEGER    ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL       AREA(40),XMOMNT(40),YOMMNT(40),FLDWDH,ARATIO
REAL       AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL       PNTTRK( 20,250,2),PERIM( 20)
REAL       SDSTAT( 20,50),PNSTAT( 20,50)
REAL       XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL       A,B,THETA,PHI,MAG,SIGMA

COMMON     RESET,STPMEM,RECLL
COMMON     VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON     BNDTRK,SCNROW,ERROR,LINE
COMMON     DTHROW,BUFROW,BTHROW
COMMON     OLDROW,NEWROW,BNDROW
COMMON     OBJROW,PARENT,BNDCNT,LFETRK
COMMON     BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON     TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON     CMLPTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON     PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON     MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON     OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON     LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON     PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON     CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON     ANUM,BNUM,ATRK,BTRK
COMMON     AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON     AAREA,XXMNT,YYMNT,ABND,BBND
COMMON     PNTTRK,PERIM
COMMON     SDSTAT,PNSTAT
COMMON     XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON     A,B,THETA,PHI,MAG,SIGMA

C      *****
C      *
C      * REORDER THE RELATIVE POSITION OF THE ARRAY ELEMENTS IN LBDTRK AND
C      * RBDTRK SO THAT THEY WILL REFLECT THE PROPER SEQUENCE FOR TRACKING
C      * AN OBJECT'S BOUNDARY.
C      *
C      *****
C      CALL RTEPTE

```



```

C *****
C *
C * INCLUDE ALL HOLES THAT TERMINATED INTO THE ARRAY BDOBJ.
C *
C *****

OBCNT = BDCNT

DO 10 MARK1 = 1,OBJCNT

IF (TYPE(MARK1) .EQ. HOLE .AND. DTHTYP(MARK1) .EQ. 1) THEN

    OBCNT = OBCNT+1
    BDOBJ(OBCNT) = MARK1

ENDIF

10 CONTINUE

C *****
C *
C * DETERMINE THE CORNER POINTS FOR ALL OBJECTS LISTED IN BDOBJ.
C *
C *****

DO 60 MARK1 = 1,OBCNT

C *****
C *
C * IF THE OBJECT HAS TOO SMALL OF AN ABSOLUTE AREA, THEN IGNORE IT.
C *
C *****

MMM = BDOBJ(MARK1)

IF (AREA(MMM) .LT. 4.0 .AND. AREA(MMM) .GT. -4.0) THEN

    GO TO 60

ENDIF

C *****
C *
C * MAP OUT THE ROUTE FOR THE BOUNDARY TRACKING FOR THE OBJECT IN QUESTION.
C * SET THE SYSTEM PARAMETERS: M1-M6.
C *
C *****

CALL RTEPLN

M1 = ROUTE(1,1)
M2 = ROUTE(1,2)
M3 = ROUTE(1,3)
M5 = ROUTE(1,4)
M6 = ROUTE(1,5)

C *****
C *
C * THE FIRST CORNER POINT IS THE FIRST PIXEL DETECTED FOR THE OBJECT.
C *
C *****

XPNT1 = REAL(BNDTRK(M2,M1,M3))*FLDWDH/512.0
YPNT1 = REAL(M3)*FLDWDH/(480.*ARATIO)

```

```

PNTTRK(MARK1,1,1) = XPNT1
PNTTRK(MARK1,1,2) = YPNT1
PNTNUM(MARK1) = 1
FLAG1 = 0
FLAG2 = 0
MARK2 = 1

IF (M6 .EQ. DOWN) THEN

    M4 = M3+8

ELSE

    M4 = M3-8

ENDIF

C *****
C *
C * IF THE SCAN IS GOING DOWN THE SIDE OF AN OBJECT, THEN ESTABLISH THE
C * THREE POINTS THAT DEFINE THE TWO SIDE VECTORS.
C *
C *****

20 IF (M6 .EQ. DOWN) THEN

30    XPNT1 = REAL(BNDTRK(M2,M1,M4-8))*FLDWDH/512.0
        YPNT1 = REAL(M4-8)*FLDWDH/(480.0*ARATIO)
        XPNT2 = REAL(BNDTRK(M2,M1,M4-4))*FLDWDH/512.0
        YPNT2 = REAL(M4-4)*FLDWDH/(480.0*ARATIO)
        XPNT3 = REAL(BNDTRK(M2,M1,M4))*FLDWDH/512.0
        YPNT3 = REAL(M4)*FLDWDH/(480.0*ARATIO)

35    X1 = XPNT1-XPNT2
        Y1 = YPNT2-YPNT1
        X2 = XPNT2-XPNT3
        Y2 = YPNT3-YPNT2

C *****
C *
C * IF THE TWO VECTORS HAVE X COMPONENTS WITH DIFFERENT SIGNS, THEN ENTER
C * THE SECOND POINT INTO PNTTRK.
C *
C *****

        IF (X1 .LT. 0 .AND. X2 .GT. 0 .OR. X1 .GT. 0 .AND. X2 .LT. 0)
2            THEN

                PNTNUM(MARK1) = PNTNUM(MARK1)+1
                PNTTRK(MARK1,PNTNUM(MARK1),1) = XPNT2
                PNTTRK(MARK1,PNTNUM(MARK1),2) = YPNT2
                WRITE(1,37) M4,PNTNUM(MARK1)
37            FORMAT(2X,'M4,PNTNUM,1',2X,I3,2X,I3)
                M4 = M4
                GO TO 36

        ENDIF

        X1 = ABS(X1)
        X2 = ABS(X2)
        A = SQRT(X1**2 + Y1**2)
        B = SQRT(X2**2 + Y2**2)
        THETA = ACOS(((X1*Y2)+(Y1*X2))/(A*B))
        THETA = ABS(THETA)

```

```

C *****
C *
C * IF THE ANGLE BETWEEN THE TWO VECTORS IS GREATER THAN 0.52 RADS, THEN
C * ENTER THE SECOND POINT INTO THE ARRAY, PNTTRK.
C *
C *****

      IF (THETA .GT. 0.52359) THEN

          PNTNUM(MARK1) = PNTNUM(MARK1)+1
          PNTTRK(MARK1,PNTNUM(MARK1),1) = XPNT2
          PNTTRK(MARK1,PNTNUM(MARK1),2) = YPNT2
          M4 = M4

      ENDIF

36      M4 = M4+4

C *****
C *
C * IF THE SCAN IS AT THE END OF AN OBJECT'S BOUNDARY, THEN CHANGE ROUTE
C * AND DETERMINE IF A CORNER POINT HAS BEEN ESTABLISHED, ELSE DETERMINE
C * THE NEXT THREE POINTS IN THE SCAN PROCESS.
C *
C *****

      IF (M4 .LE. M5-1) THEN

          GO TO 30

      ELSE

C *****
C *
C * IF THE COMPLETE SCANNING AROUND THE OBJECT HAS BEEN COMPLETED, THEN
C * DETERMINE WHETHER THE LAST POINT IS PART OF THE FIRST CORNER POINT OR
C * A SEPARATE POINT BY ITSELF.
C *
C *****

          MARK2 = MARK2+1

          IF (MARK2 .GT. RTEPNT) THEN

              XPNT2 = PNTTRK(MARK1,1,1)
              YPNT2 = PNTTRK(MARK1,1,2)
              A = SQRT((XPNT2-XPNT3)**2 + (YPNT2-YPNT3)**2)

              IF (A .GT. 1.5) THEN

                  PNTNUM(MARK1) = PNTNUM(MARK1)+1
                  PNTTRK(MARK1,PNTNUM(MARK1),1) = XPNT1
                  PNTTRK(MARK1,PNTNUM(MARK1),2) = YPNT1

              ENDIF

              GO TO 50

          ELSE

C *****
C *
C * IF THE DIRECTION OF THE SCAN GOES FROM DOWN TO UP, THEN ENTER THE THIRD
C * POINT INTO THE ARRAY, PNTTRK, AND PREPARE THE NEXT THREE POINTS FOR
C * SUBSEQUENT SCAN THROUGH THE NEW ROUTE.

```

```

C      *
C      *****

      IF (ROUTE(MARK2,5) .EQ. UP) THEN

          PNTNUM(MARK1) = PNTNUM(MARK1)+1
          PNTTRK(MARK1,PNTNUM(MARK1),1) = XPNT3
          PNTTRK(MARK1,PNTNUM(MARK1),2) = YPNT3

          FLAG1 = 0
          M1 = ROUTE(MARK2,1)
          M2 = ROUTE(MARK2,2)
          M3 = ROUTE(MARK2,3)
          M5 = ROUTE(MARK2,4)
          M6 = ROUTE(MARK2,5)

          M4 = M3-9

          GO TO 20

      ELSE

C      *****
C      *
C      * PLAN THE NEXT THREE POINTS FOR THE SUBSEQUENT SCAN THROUGH THE NEW
C      * ROUTE.
C      *
C      *****

          XPNT1 = XPNT2
          YPNT1 = YPNT2
          M1 = ROUTE(MARK2,1)
          M2 = ROUTE(MARK2,2)
          M3 = ROUTE(MARK2,3)
          M5 = ROUTE(MARK2,4)
          M6 = ROUTE(MARK2,5)
          M4 = M3+4
          XPNT2 = REAL(BNDTRK(M2,M1,M3))*FLDWDH/512.0
          YPNT2 = REAL(M3)*FLDWDH/(ARATIO*480.0)
          XPNT3 = REAL(BNDTRK(M2,M1,M3+4))*FLDWDH/512.0
          YPNT3 = REAL(M3+4)*FLDWDH/(ARATIO*480.0)

          GO TO 35

      ENDIF

      ENDIF

      ENDIF

      ENDIF

C      *****
C      *
C      * IF THE SCAN IS GOING UP THE SIDE OF AN OBJECT, THEN ESTABLISH THE
C      * THREE POINTS THAT DEFINE THE TWO SIDE VECTORS.
C      *
C      *****

      IF (M6 .EQ. UP) THEN

40      XPNT3 = REAL(BNDTRK(M2,M1,M4))*FLDWDH/512.0
          YPNT3 = REAL(M4)*FLDWDH/(ARATIO*480.0)
          XPNT2 = REAL(BNDTRK(M2,M1,M4+4))*FLDWDH/512.0
          YPNT2 = REAL(M4+3)*FLDWDH/(ARATIO*480.0)
          XPNT1 = REAL(BNDTRK(M2,M1,M4+8))*FLDWDH/512.0

```

```

        YPNT1 = REAL(M4+6)*FLDWDH/(ARATIO*480.0)

45      X1 = XPNT1-XPNT2
        Y1 = YPNT2-YPNT1
        X2 = XPNT2-XPNT3
        Y2 = YPNT3-YPNT2

C      *****
C      *
C      * IF THE TWO VECTORS HAVE X COMPONETS WITH DIFFERENT SIGNS, THEN ENTER
C      * THE SECOND POINT INTO PNTRK.
C      *
C      *****

2      IF (X1 .LT. 0 .AND. X2 .GT. 0 .OR. X1 .GT. 0 .AND. X2 .LT. 0)
        THEN

            PNTNUM(MARK1) = PNTNUM(MARK1)+1
            PNTRK(MARK1,PNTNUM(MARK1),1) = XPNT2
            PNTRK(MARK1,PNTNUM(MARK1),2) = YPNT2
            M4 = M4
            GO TO 46

        ENDIF

        X1 = ABS(X1)
        X2 = ABS(X2)
        X1 = -X1
        X2 = -X2

        A = SQRT(X1**2 + Y1**2)
        B = SQRT(X2**2 + Y2**2)
        THETA = ACOS(((X1*Y2)+(Y1*X2))/(A*B))
        THETA = ABS(THETA)

C      *****
C      *
C      * IF THE ANGLE BETWEEN THE TWO VECTORS IS GREATER THAN 0.52 RADS, THEN
C      * ENTER THE SECOND POINT INTO THE ARRAY, PNTRK.
C      *
C      *****

        IF (THETA .GT. 0.52359) THEN

            PNTNUM(MARK1) = PNTNUM(MARK1)+1
            PNTRK(MARK1,PNTNUM(MARK1),1) = XPNT2
            PNTRK(MARK1,PNTNUM(MARK1),2) = YPNT2
            M4 =M4

        ENDIF

46      M4 = M4-4

C      *****
C      *
C      * IF THE SCAN IS AT THE END OF AN OBJECT'S BOUNDARY, THEN CHANGE ROUTE
C      * AND DETERMINE IF A CORNER POINT HAS BEEN ESTABLISHED, ELSE DETERMINE
C      * THE NEXT THREE POINTS IN THE SCAN PROCESS.
C      *
C      *****

        IF (M4 .GE. M5) THEN

            GO TO 40

        ELSE

```

```

MARK2 = MARK2+1

C *****
C *
C * IF THE COMPLETE SCANNING AROUND THE OBJECT HAS BEEN COMPLETED, THEN
C * DETERMINE WHETHER THE LAST POINT IS PART OF THE FIRST CORNER POINT OR
C * A SEPARATE POINT BY ITSELF.
C *
C *****

IF (MARK2 .GT. RTEPNT) THEN

    XPNT2 = PNTTRK(MARK1,1,1)
    YPNT2 = PNTTRK(MARK1,1,2)
    A = SQRT((XPNT2-XPNT3)**2 + (YPNT2-YPNT3)**2)

    IF (A .GT. 1.50) THEN

        PNTNUM(MARK1) = PNTNUM(MARK1)+1
        PNTTRK(MARK1,PNTNUM(MARK1),1) = XPNT1
        PNTTRK(MARK1,PNTNUM(MARK1),2) = YPNT1

    ENDIF

    GO TO 50

C *****
C *
C * IF THE DIRECTION OF THE SCAN GOES FROM UP TO DOWN, THEN ENTER THE THIRD
C * POINT INTO THE ARRAY, PNTTRK, AND PREPARE THE NEXT THREE POINTS FOR
C * SUBSEQUENT SCAN THROUGH THE NEW ROUTE.
C *
C *****

ELSE

    IF (ROUTE(MARK2,5) .EQ. DOWN) THEN

        PNTNUM(MARK1) = PNTNUM(MARK1)+1
        PNTTRK(MARK1,PNTNUM(MARK1),1) = XPNT3
        PNTTRK(MARK1,PNTNUM(MARK1),2) = YPNT3

        M1 = ROUTE(MARK2,1)
        M2 = ROUTE(MARK2,2)
        M3 = ROUTE(MARK2,3)
        M5 = ROUTE(MARK2,4)
        M6 = ROUTE(MARK2,5)

        M4 = M3+9

        GO TO 20

C *****
C *
C * PLAN THE NEXT THREE POINTS FOR THE SUBSEQUENT SCAN THROUGH THE NEW
C * ROUTE.
C *
C *****

ELSE

    XPNT1 = XPNT2
    YPNT1 = YPNT2
    M1 = ROUTE(MARK2,1)
    M2 = ROUTE(MARK2,2)

```

```

M3 = ROUTE(MARK2,3)
M5 = ROUTE(MARK2,4)
M6 = ROUTE(MARK2,5)
M4 = M3-4
XPNT2 = REAL(BNDTRK(M2,M1,M3-1))*FLDWDH/512.0
YPNT2 = REAL(M3-1)*FLDWDH/(ARATIO*480.0)
XPNT3 = REAL(BNDTRK(M2,M1,M3-4))*FLDWDH/512.0
YPNT3 = REAL(M3-4)*FLDWDH/(ARATIO*480.0)

```

```

GO TO 45

```

```

ENDIF

```

```

ENDIF

```

```

ENDIF

```

```

ENDIF

```

```

C *****
C *
C * FILTER OUT THE NON CORNER POINTS THAT WERE ENTERED INTO PNTRK AND
C * COMPUTE THE LENGTH OF EACH SIDE OF THE OBJECT AS WELL AS THE DISTANCE
C * OF EACH CORNER POINT FROM THE CENTROID OF THE OBJECT.
C *
C *****

```

```

50 CALL CLNUP2
   CALL CLNUP1
   CALL RTESTT

```

```

60 CONTINUE

```

```

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

```

```

RETURN
END
SUBROUTINE RTEPLN

```

```

C *****
C *
C * SUBROUTINE RTEPLN
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 2-17-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM CREATES A ROUTE PLAN FOR CIRCUMNAVIGATING THE BOUNDARY
C * OF AN OBJECT. THE PROGRAM DOES THIS BY DOING THE FOLLOWING:
C *
C * 1. SCANNING THROUGH AN OBJECT'S ROW IN LBDTRK AND RBDTRK TO
C * DETERMINE WHICH OBJECTS HAVE BEEN MERGED WITH,
C *
C * 2. TRACKING WHERE OBJECT BOUNDARIES BEGIN AND END AND IN WHICH
C * DIRECTION THEY ARE TO BE SCANNED, AND
C *
C * 3. UTILIZING A STACK ARRAY, TO AID IN THE CREATION OF A ROUTING
C * ARRAY, ROUTE.
C *
C *****

```

```

C      *
C      * EACH ROW OF THE ARRAY, ROUTE, CONTAINS THE FOLLOWING INFORMATION:
C      *
C      *      1. WHICH LINE NUMBER TO START THE TREK ALONG THE BOUNDARY,
C      *
C      *      2. WHICH LINE NUMBER TO STOP THE TREK ALONG THE BOUNDARY,
C      *
C      *      3. WHICH DIRECTION (UP OR DOWN) TO GO ALONG THE BOUNDARY,
C      *
C      *      4. WHICH OBJECT BOUNDARY, AND
C      *
C      *      5. WHICH BOUNDARY (LEFT OR RIGHT).
C      *
C      *
C      * SUBROUTINES USED: NONE
C      *
C      *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLAG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCH,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL      AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL      AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL      PNTTRK(20,250,2),PERIM(20)
REAL      SDSTAT(20,50),PNSTAT(20,50)
REAL      XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL      A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROM,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLAG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```



```

C *****
C *
C * SET SUBROUTINE PARAMETERS: M4-M10.
C *
C *****

M9 = BDOBJ(MARK1)
M10 = M9
M8 = 0
M7 = 0
M5 = LFETRK(M9,1)
M4 = DOWN

C *****
C *
C * IF THE PARENT BOUNDARY BELONGS TO A BODY, THEN PROCESS THIS ROUTE.
C *
C *****

IF (TYPE(M10) .EQ. BODY) THEN

C *****
C *
C * IF THE MERGING BOUNDARY BELONGS TO A BODY, THEN DETERMINE WHETHER
C * IT BELONGS TO THE PARENT OBJECT OR TO AN OFFSPRING.
C *
C *****

10 IF (TYPE(M9) .EQ. BODY) THEN

    IF (M9 .NE. M10) THEN

        GO TO 35

    ENDIF

C *****
C *
C * IF THE BOUNDARY BELONGS TO THE PARENT OBJECT, AND THE LEFT SIDE
C * PROCESSING HAS NOT BEEN COMPLETED, THEN DETERMINE WHETHER THE
C * OBJECT IS LISTED IN LBDTRK.
C *
C *****

15 IF (BNDST(M9,1) .NE. CPLTE)

    MARK5 = 0

    DO 20 MARK3 = 1,PARCNT

        IF (M9 .EQ. LBDTRK(MARK3,1,1)) THEN

            MARK5 = MARK3

        ENDIF

20 CONTINUE

    IF (MARK5 .EQ. 0) THEN

        GO TO 30

    ENDIF

```

```

C *****
C *
C * IF THE OBJECT IS LISTED IN LBDTRK, THEN DETERMINE THE NEXT MERGING
C * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST
C * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C * AS COMPLETE, AND UPDATE ROUTE.
C *
C *****

      LMARK(MARK5) = LMARK(MARK5)+1

      IF (LMARK(MARK5) .LE. LTCNT(MARK5)) THEN

          M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = LEFT
          ROUTE(M8,3) = M5
          M6 = LBDTRK(MARK5,LMARK(MARK5)+1,2)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = DOWN
          M7 = M7+1
          STACK(M7,1) = M9
          STACK(M7,2) = M6
          M9 = LBDTRK(MARK5,LMARK(MARK5)+1,1)
          M5 = M6
          GO TO 10

      ENDIF

30      M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = LEFT
          ROUTE(M8,3) = M5
          M6 = LFETRK(M9,2)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = DOWN
          BNDST(M9,1) = CMLTE
          M5 = M6

C *****
C *
C * IF THE COMPLETED BOUNDARY DOES NOT BELONG TO THE PARENT OBJECT, THEN
C * POP THE ARRAY, STACK, FOR THE NEXT BOUNDARY.
C *
C *****

      IF (M9 .NE. M10) THEN

          M9 = STACK(M7,1)
          M5 = STACK(M7,2)
          M7 = M7-1
          GO TO 10

      ELSE

          GO TO 10

      ENDIF

      ENDIF

C *****
C *
C * IF THE BOUNDARY BELONGS TO THE PARENT OBJECT, AND THE RIGHT SIDE
C * PROCESSING HAS NOT BEEN COMPLETED, THEN DETERMINE WHETHER THE

```

```

C      * OBJECT IS LISTED IN RBDTRK.
C      *
C      *****

35      IF (BNDST(M9,2) .NE. CPLTE) THEN

          MARK5 = 0

          DO 40 MARK3 = 1,PARCNT

              IF (M9 .EQ. RBDTRK(MARK3,1,1)) THEN

                  MARK5 = MARK3

              ENDIF

40      CONTINUE

          IF (MARK5 .EQ. 0) THEN

              GO TO 50

          ENDIF

C      *****
C      *
C      * IF THE OBJECT IS LISTED IN RBDTRK, THEN DETERMINE THE NEXT MERGING
C      * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C      * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST
C      * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C      * AS COMPLETE, AND UPDATE ROUTE.
C      *
C      *****

          RMARK(MARK5) = RMARK(MARK5)+1

          IF (RMARK(MARK5) .LE. RTCNT(MARK5)) THEN

              M8 = M8+1
              ROUTE(M8,1) = M9
              ROUTE(M8,2) = RIGHT
              ROUTE(M8,3) = M5
              M6 = RBDTRK(MARK5,RMARK(MARK5)+1,2)
              ROUTE(M8,4) = M6
              ROUTE(M8,5) = UP
              M7 = M7+1
              STACK(M7,1) = M9
              STACK(M7,2) = M6
              M9 = RBDTRK(MARK5,RMARK(MARK5)+1,1)
              M5 = M6
              GO TO 10

          ENDIF

50      M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = RIGHT
          ROUTE(M8,3) = M5
          M6 = LFETRK(M9,1)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = UP
          BNDST(M9,2) = CPLTE

C      *****
C      *
C      * IF THE COMPLETED BOUNDARY BELONGS TO THE PARENT OBJECT, THEN SET RTEPNT

```

```

C      * EQUAL TO THE ARRAY COUNTER, M8. IF NOT, THEN RESET M5 AND START FROM
C      * THE TOP OF THE LOGIC.
C      *
C      *****
      IF (M9 .EQ. M10) THEN
          RTEPNT = M8
          GO TO 190
      ELSE
          M5 = M6
          GO TO 10
      ENDIF
      ENDIF
      GO TO 15
      ENDIF

C      *****
C      *
C      * IF THE MERGING BOUNDARY BELONGS TO A HOLE, THEN DETERMINE WHETHER THE
C      * BOUNDARY HAS BEEN COMPLETED.
C      *
C      *****

      IF (TYPE(M9) .EQ. HOLE) THEN

C      *****
C      *
C      * IF THE RIGHT SIDE OF THE HOLE HAS NOT BEEN COMPLETED, THEN DETERMINE
C      * WHETHER THE OBJECT IS LISTED IN RBDTRK.
C      *
C      *****

      IF (BNDST(M9,2) .NE. CMLPTE) THEN
          MARK5 = 0
          DO 60 MARK3 = 1,PARCNT
              IF (M9 .EQ. RBDTRK(MARK3,1,1)) THEN
                  MARK5 = MARK3
              ENDIF
          60  CONTINUE
              IF (MARK5 .EQ. 0) THEN
                  GO TO 70
              ENDIF

C      *****
C      *
C      * IF THE OBJECT IS LISTED IN RBDTRK, THEN DETERMINE THE NEXT MERGING
C      * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C      * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST

```

```

C      * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C      * AS COMPLETE, AND UPDATE ROUTE.
C      *
C      *****

      RMARK(MARK5) = RMARK(MARK5)+1

      IF (RMARK(MARK5) .LE. RTCNT(MARK5)) THEN

          M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = RIGHT
          ROUTE(M8,3) = M5
          M6 = RBDTRK(MARK5,RMARK(MARK5)+1,2)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = UP
          M7 = M7+1
          STACK(M7,1) = M9
          STACK(M7,2) = M6
          M9 = RBDTRK(MARK5,RMARK(MARK5)+1,1)
          M5 = M6
          GO TO 10

      ENDIF

70      M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = RIGHT
          ROUTE(M8,3) = M5
          M6 = LFETRK(M9,1)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = UP
          BNDST(M9,2) = CMLPTE
          M5 = M6
          GO TO 10

      ENDIF

C      *****
C      *
C      * IF THE LEFT SIDE PROCESSING IS NOT COMPLETE, THEN DETERMINE WHETHER THE
C      * HOLE IS LISTED IN LBDTRK.
C      *
C      *****

      IF (BNDST(M9,1) .NE. CMLPTE) THEN

          MARK5 = 0

          DO 80 MARK3 = 1,PARCNT

              IF (M9 .EQ. LBDTRK(MARK3,1,1)) THEN

                  MARK5 = MARK3

              ENDIF

80      CONTINUE

          IF (MARK5 .EQ. 0) THEN

              GO TO 90

          ENDIF

```

```

C *****
C *
C * IF THE OBJECT IS LISTED IN LBDTRK, THEN DETERMINE THE NEXT MERGING
C * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST
C * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C * AS COMPLETE, AND UPDATE ROUTE.
C *
C *****

      LMARK(MARK5) = LMARK(MARK5)+1

      IF (LMARK(MARK5) .LE. LTCNT(MARK5)) THEN

          M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = LEFT
          ROUTE(M8,3) = M5
          M6 = LBDTRK(MARK5,LMARK(MARK5)+1,2)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = DOWN
          M7 = M7+1
          STACK(M7,1) = M9
          STACK(M7,2) = M6
          M9 = LBDTRK(MARK5,LMARK(MARK5)+1,1)
          M5 = M6
          GO TO 10

      ENDIF

90      M8 = M8+1
      ROUTE(M8,1) = M9
      ROUTE(M8,2) = LEFT
      ROUTE(M8,3) = M5
      M6 = LFETRK(M9,2)
      ROUTE(M8,4) = M6
      ROUTE(M8,5) = DOWN
      BNDST(M9,1) = CMPLTE
      M5 = M6
      M9 = STACK(M7,1)
      M5 = STACK(M7,2)
      M7 = M7-1
      GO TO 10

      ENDIF

      ENDIF

      ENDIF

C *****
C *
C * IF THE PARENT BOUNDARY BELONGS TO A HOLE, THEN PROCESS THIS ROUTE.
C *
C *****

      IF (TYPE(M10) .EQ. HOLE) THEN

C *****
C *
C * IF THE MERGING BOUNDARY BELONGS TO A BODY, THEN DETERMINE WHICH SIDE
C * IF ANY HAS BEEN COMPLETED.
C *
C *****

100      IF (TYPE(M9) .EQ. BODY) THEN

```

```

C *****
C *
C * IF THE LEFT SIDE PROCESSING IS NOT COMPLETE, THEN DETERMINE WHETHER THE
C * BODY IS LISTED IN LBDTRK.
C *
C *****

      IF (BNDST(M9,1) .NE. CPLTE) THEN

          MARK5 = 0

          DO 110 MARK3 = 1,PARCNT

              IF (M9 .EQ. LBDTRK(MARK3,1,1)) THEN

                  MARK5 = MARK3

              ENDIF

110      CONTINUE

          IF (MARK5 .EQ. 0) THEN

              GO TO 120

          ENDIF

C *****
C *
C * IF THE OBJECT IS LISTED IN LBDTRK, THEN DETERMINE THE NEXT MERGING
C * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST
C * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C * AS COMPLETE, AND UPDATE ROUTE.
C *
C *****

      LMARK(MARK5) = LMARK(MARK5)+1

      IF (LMARK(MARK5) .LE. LTCNT(MARK5)) THEN

          M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = LEFT
          ROUTE(M8,3) = M5
          M6 = LBDTRK(MARK5,LMARK(MARK5)+1,2)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = UP
          M7 = M7+1
          STACK(M7,1) = M9
          STACK(M7,2) = M6
          M9 = LBDTRK(MARK5,LMARK(MARK5)+1,1)
          M5 = M6
          GO TO 100

      ENDIF

120      M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = LEFT
          ROUTE(M8,3) = M5
          M6 = LFETRK(M9,1)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = UP
          BNDST(M9,1) = CPLTE

```

```

        M5 = M6
        GO TO 100

    ENDIF

C *****
C *
C * IF THE RIGHT SIDE OF THE BODY HAS NOT BEEN COMPLETED, THEN DETERMINE
C * WHETHER THE OBJECT IS LISTED IN RBDTRK.
C *
C *****

        IF (BNDST(M9,2) .NE. CMLPTE) THEN

            MARK5 = 0

            DO 130 MARK3 = 1,PARCNT

                IF (M9 .EQ. RBDTRK(MARK3,1,1)) THEN

                    MARK5 = MARK3

                ENDIF

130        CONTINUE

            IF (MARK5 .EQ. 0) THEN

                GO TO 140

            ENDIF

C *****
C *
C * IF THE OBJECT IS LISTED IN RBDTRK, THEN DETERMINE THE NEXT MERGING
C * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST
C * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C * AS COMPLETE, AND UPDATE ROUTE.
C *
C *****

            RMARK(MARK5) = RMARK(MARK5)+1

            IF (RMARK(MARK5) .LE. RTCNT(MARK5)) THEN

                M8 = M8+1
                ROUTE(M8,1) = M9
                ROUTE(M8,2) = RIGHT
                ROUTE(M8,3) = M5
                M6 = RBDTRK(MARK5,RMARK(MARK5)+1,2)
                ROUTE(M8,4) = M6
                ROUTE(M8,5) = DOWN
                M7 = M7+1
                STACK(M7,1) = M9
                STACK(M7,2) = M6
                M9 = RBDTRK(MARK5,RMARK(MARK5)+1,1)
                M5 = M6
                GO TO 100

            ENDIF

140        M8 = M8+1
            ROUTE(M8,1) = M9
            ROUTE(M8,2) = RIGHT
            ROUTE(M8,3) = M5

```



```

        M6 = LFETRK(M9,2)
        ROUTE(M8,4) = M6
        ROUTE(M8,5) = DOWN
        BNDST(M9,2) = CMLPTE
        M5 = M6
        M9 = STACK(M7,1)
        M5 = STACK(M7,2)
        M7 = M7-1
        GO TO 100

    ENDIF

ENDIF

C *****
C *
C * IF THE MERGING BOUNDARY BELONGS TO A HOLE, THEN DETERMINE WHETHER
C * IT BELONGS TO THE PARENT OBJECT OR TO AN OFFSPRING.
C *
C *****

    IF (TYPE(M9) .EQ. HOLE) THEN

        IF (M9 .NE. M10) THEN

            GO TO 165

        ENDIF

C *****
C *
C * IF THE BOUNDARY BELONGS TO THE PARENT OBJECT, AND THE RIGHT SIDE
C * PROCESSING HAS NOT BEEN COMPLETED, THEN DETERMINE WHETHER THE
C * OBJECT IS LISTED IN RBDTRK.
C *
C *****

145    IF (BNDST(M9,2) .NE. CMLPTE) THEN

        MARK5 = 0

        DO 150 MARK3 = 1,PARCNT

            IF (M9 .EQ. RBDTRK(MARK3,1,1)) THEN

                MARK5 = MARK3

            ENDIF

150    CONTINUE

        IF (MARK5 .EQ. 0) THEN

            GO TO 160

        ENDIF

C *****
C *
C * IF THE OBJECT IS LISTED IN RBDTRK, THEN DETERMINE THE NEXT MERGING
C * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST
C * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C * AS COMPLETE, AND UPDATE ROUTE.

```

```

C      *
C      *****

      RMARK(MARK5) = RMARK(MARK5)+1

      IF (RMARK(MARK5) .LE. RTCNT(MARK5)) THEN

          M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = RIGHT
          ROUTE(M8,3) = M5
          M6 = RBDTRK(MARK5,RMARK(MARK5)+1,2)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = DOWN
          M7 = M7+1
          STACK(M7,1) = M9
          STACK(M7,2) = M6
          M9 = RBDTRK(MARK5,RMARK(MARK5)+1,1)
          M5 = M6
          GO TO 100

      ENDIF

160      M8 = M8+1
          ROUTE(M8,1) = M9
          ROUTE(M8,2) = RIGHT
          ROUTE(M8,3) = M5
          M6 = LFETRK(M9,2)
          ROUTE(M8,4) = M6
          ROUTE(M8,5) = DOWN
          BNDST(M9,2) = CMLPTE
          M5 = M6

C      *****
C      *
C      * IF THE COMPLETED BOUNDARY DOES NOT BELONG TO THE PARENT OBJECT, THEN
C      * POP THE ARRAY, STACK, FOR THE NEXT BOUNDARY.
C      *
C      *****

      IF (M9 .NE. M10) THEN

          M9 = STACK(M7,1)
          M5 = STACK(M7,2)
          M7 = M7-1
          GO TO 100

      ELSE

          GO TO 100

      ENDIF

      ENDIF

C      *****
C      *
C      * IF THE BOUNDARY BELONGS TO THE PARENT OBJECT, AND THE LEFT SIDE
C      * PROCESSING HAS NOT BEEN COMPLETED, THEN DETERMINE WHETHER THE
C      * OBJECT IS LISTED IN LBDTRK.
C      *
C      *****

165      IF (BNDST(M9,1) .NE. CMLPTE) THEN

```

```

MARK5 = 0

DO 170 MARK3 = 1,PARCNT

IF (M9 .EQ. LBDTRK(MARK3,1,1)) THEN

    MARK5 = MARK3

ENDIF

170 CONTINUE

IF (MARK5 .EQ. 0) THEN

    GO TO 180

ENDIF

C *****
C *
C * IF THE OBJECT IS LISTED IN LBDTRK, THEN DETERMINE THE NEXT MERGING
C * BODY. IF THE LAST MERGING BODY HAS NOT BEEN PROCESSED, THEN UPDATE
C * ROUTE, AND PLACE THE PARENT OBJECT IN THE ARRAY,STACK. IF THE LAST
C * MERGING BODY HAS BEEN PROCESSED, THEN CLASSIFY THE OBJECT'S BOUNDARY
C * AS COMPLETE, AND UPDATE ROUTE.
C *
C *****

LMARK(MARK5) = LMARK(MARK5)+1

IF (LMARK(MARK5) .LE. LTCNT(MARK5)) THEN

    M8 = M8+1
    ROUTE(M8,1) = M9
    ROUTE(M8,2) = LEFT
    ROUTE(M8,3) = M5
    M6 = LBDTRK(MARK5,LMARK(MARK5)+1,2)
    ROUTE(M8,4) = M6
    ROUTE(M8,5) = UP
    M7 = M7+1
    STACK(M7,1) = M9
    STACK(M7,2) = M6
    M9 = LBDTRK(MARK5,LMARK(MARK5)+1,1)
    M5 = M6
    GO TO 100

ENDIF

180 M8 = M8+1
    ROUTE(M8,1) = M9
    ROUTE(M8,2) = LEFT
    ROUTE(M8,3) = M5
    M6 = LFETRK(M9,1)
    ROUTE(M8,4) = M6
    ROUTE(M8,5) = UP
    BNDST(M9,1) = CMLTE

    IF (M9 .EQ. M10) THEN

        RTEPNT = M8
        GO TO 190

    ELSE

```

```

        M5 = M6
        GO TO 100

    ENDIF

ENDIF

    GO TO 145

ENDIF

ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

190 RETURN
END

SUBROUTINE RERTE

C *****
C *
C * SUBROUTINE RERTE
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 2-19-86
C *
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM PREPARES THE ARRAYS LBDTRK AND RBDTRK TO BE USED BY
C * RTEPLN BY ARRANGING THE ELEMENTS IN EACH ROW IN THE PROPER ORDER.
C * THE MANNER OF REARRANGEMENT IS BASED ON WHETHER THE MERGING PARENT
C * IS EITHER A BODY OR A HOLE AND WHETHER THE PARENT IS TO BE SCANNED
C * IN THE CLOCKWISE OR COUNTER CLOCKWISE DIRECTION.
C *
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C *     1. CSWTC
C *     2. PSWTC
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OB_CNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BND RTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT

```

```
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```
REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA
```

```
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA
```

```
C *****
C *
C * CHECK THE NUMBER OF LEFT HAND MERGES FOR EACH OBJECT. IF GREATER THAN
C * ZERO, THEN REARRANGE THE ROW OF OFFSPRING OF THAT OBJECT IN LBDTRK.
C *
C *****
```

```
DO 10 PARPNT = 1,PARCNT
```

```
    IF (LTCNT(PARPNT) .LE. 1) THEN
```

```
        GO TO 10
```

```
    ENDIF
```

```
    MM = LBDTRK(PARPNT,1,1)
```

```
C *****
C *
C * IF THE OBJECT IS A BODY, AND ITS ROUTE SCHEME IS CW, THEN INTERCHANGE
C * THE POSITION OF EVERYONE OF ITS OFFSPRING IN LBDTRK.
C *
C *****
```

```
    IF (TYPE(MM) .EQ. BODY) THEN
```

```
        IF (BNDRTE(MM) .EQ. CW) THEN
```

```
            FLAG1 = 0
            CALL CSWCH
```

```
        ENDIF
```

```

ENDIF

C *****
C *
C * IF THE OBJECT IS A HOLE, AND ITS ROUTE SCHEME IS CCW, THEN INTERCHANGE
C * THE POSITION OF EVERYONE OF ITS BODY OFFSPRING IN LBDTRK. IF ITS ROUTE
C * SCHEME IS CW, THEN INTERCHANGE EVERYONE OF ITS OFFSPRING, AND THEN
C * FOLLOW UP BY INTERCHANGING EVERY BODY OFFSPRING WHICH HAVE THE SAME
C * MOMENT OF DEATH.
C *
C *****

IF (TYPE(MM) .EQ. HOLE) THEN

    IF (BND RTE(MM) .EQ. CCW) THEN

        FLAG1 = 0
        CALL PSWTC H

    ELSE

        FLAG1 = 0
        CALL CSWTC H
        CALL PSWTC H

    ENDIF

ENDIF

10 CONTINUE

C *****
C *
C * CHECK THE NUMBER OF RIGHT HAND MERGES FOR EACH OBJECT. IF GREATER THAN
C * ZERO, THEN REARRANGE THE ROW OF OFFSPRING OF THAT OBJECT IN RBDTRK.
C *
C *****

20 DO 30 PARPNT = 1,PARCNT

    IF (RTCNT(PARPNT) .LE. 1) THEN

        GO TO 30

    ENDIF

    MM = RBDTRK(PARPNT,1,1)

C *****
C *
C * IF THE OBJECT IS A BODY, AND ITS ROUTE SCHEME IS CW, THEN INTERCHANGE
C * THE POSITION OF EVERYONE OF ITS HOLE OFFSPRING IN RBDTRK. IF ITS ROUTE
C * SCHEME IS CCW, THEN INTERCHANGE EVERYONE OF ITS OFFSPRING, AND THEN
C * FOLLOW UP BY INTERCHANGING EVERY HOLE OFFSPRING WHICH HAVE THE SAME
C * MOMENT OF DEATH.
C *
C *****

IF (TYPE(MM) .EQ. BODY) THEN

    IF (BND RTE(MM) .EQ. CW) THEN

        FLAG1 = 1
        CALL PSWTC H,

```

```

        ELSE

            FLAG1 = 1
            CALL CSWTCH
            CALL PSWTCH

        ENDIF

    ENDIF

C *****
C *
C * IF THE OBJECT IS A HOLE, AND ITS ROUTE SCHEME IS CCW, THEN INTERCHANGE
C * THE POSITION OF EVERYONE OF ITS OFFSPRING IN RBDTRK.
C *
C *****

        IF (TYPE(MM) .EQ. HOLE) THEN

            IF (BND RTE(MM) .EQ. CCW) THEN

                FLAG1 = 1
                CALL CSWTCH

            ENDIF

        ENDIF

30 CONTINUE

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

40 RETURN
END

SUBROUTINE CSWTCH

C *****
C *
C * SUBROUTINE CSWTCH
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE 2-20-86
C *
C * PROGRAM DESCRIPTION
C *
C * IF FLAG1 IS EQUAL TO 0, THEN THIS PROGRAM WILL INTERCHANGE EACH
C * OBJECT IN THE DESIRED ROW IN LBDTRK. IF FLAG1 IS NOT EQUAL TO 0
C * THEN THIS PROGRAM WILL INTERCHANGE EACH OBJECT IN THE DESIRED ROW
C * IN RBDTRK.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE

```

```

INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETR(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLAG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROM,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETR
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLAG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * IF FLAG1 IS EQUAL TO ZERO, THEN INTERCHANGE THE ROW ELEMENTS IN LBDTRK
C *
C *****

```

```

IF (FLAG1 .EQ. 0) THEN

```

```

    DO 10 NN = 1,LTCNT(PARPNT)

```

```

        CHGROW(NN+1,1) = LBDTRK(PARPNT,LTCNT(PARPNT)-NN+2,1)
        CHGROW(NN+1,2) = LBDTRK(PARPNT,LTCNT(PARPNT)-NN+2,2)

```

```

10    CONTINUE

```

```

    DO 20 NN = 1,LTCNT(PARPNT)

```

```

        LBDTRK(PARPNT,NN+1,1) = CHGROW(NN+1,1)
        LBDTRK(PARPNT,NN+1,2) = CHGROW(NN+1,2)

```

```

20    CONTINUE

```



```

ELSE

C *****
C *
C * IF FLAG1 IS NOT EQUAL TO ZERO, THEN INTERCHANGE THE ROW ELEMENTS IN
C * RBDTRK.
C *
C *****

      DO 30 NN = 1,RTCNT(PARPNT)

      CHGROW(NN+1,1) = RBDTRK(PARPNT,RTCNT(PARPNT)-NN+2,1)
      CHGROW(NN+1,2) = RBDTRK(PARPNT,RTCNT(PARPNT)-NN+2,2)

30    CONTINUE

      DO 40 NN = 1,RTCNT(PARPNT)

      RBDTRK(PARPNT,NN+1,1) = CHGROW(NN+1,1)
      RBDTRK(PARPNT,NN+1,2) = CHGROW(NN+1,2)

40    CONTINUE

ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE.
C *
C *****

      RETURN
      END

SUBROUTINE PSWTCH

C *****
C *
C * SUBROUTINE PSWTCH
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE 2-20-86
C *
C * PROGRAM DESCRIPTION
C *
C * IF FLAG1 IS EQUAL TO 0, THEN THIS PROGRAM WILL INTERCHANGE EACH
C * HOLE IN THE DESIRED ROW OF LBDTRK WHICH MERGED AT THE SAME LINE
C * NUMBER. IF FLAG1 IS NOT EQUAL TO 0, THEN THIS PROGRAM WILL
C * INTERCHANGE EACH BODY IN THE DESIRED ROW OF RBDTRK WHICH MERGED
C * AT THE SAME LINE NUMBER.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

      INTEGER*2 RESET,STPMEM,RECLL
      INTEGER*2 VARPNT,STPCYC,AUTO,RTRN,APP1,APP2,APP3
      INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
      INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
      INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
      INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
      INTEGER BDOBJ(20),YNGROW(40),DTHYTP(40),STACK(20,4),BNDST(40,2)

```

```

INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLPTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YOMMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLPTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * IF FLAG1 IS EQUAL TO 0, THEN SCAN THROUGH THE ROW IN LBDTRK PERTAINING
C * TO PARPNT. FIND ALL HOLES WHICH MERGED ON THE SAME SCAN LINE AND
C * INTERCHANGE THEIR POSITIONS IN THE ROW.
C *
C *****

```

```

IF (FLAG1 .EQ. 0) THEN

```

```

    M = 2

```

```

10    NM = LBDTRK(PARPNT,M,1)

```

```

    IF (TYPE(NM) .EQ. HOLE .AND. M .LT. (LTCNT(PARPNT)+1)) THEN

```

```

        LINE = LBDTRK(PARPNT,M,2)

```

```

        N = M+1

```

```

        MN = LBDTRK(PARPNT,M,1)

```

```

20    IF (LBDTRK(PARPNT,N,2) .EQ. LINE .AND. TYPE(MN) .EQ. HOLE)
2        THEN

```

```

        IF (N .LT. LTCNT(PARPNT)+1) THEN
            N = N+1
            GO TO 20

        ENDIF

        N = N+1

    ENDIF

    N = N-1

    DO 30 MMM = M,N

        CHGROW(MMM,1) = LBDTRK(PARPNT,N-MMM+M,1)
        CHGROW(MMM,2) = LBDTRK(PARPNT,N-MMM+M,2)

30      CONTINUE

        DO 40 MMM = M,N

            LBDTRK(PARPNT,MMM,1) = CHGROW(MMM,1)
            LBDTRK(PARPNT,MMM,2) = CHGROW(MMM,2)

40      CONTINUE

        M = N+1

        GO TO 45

    ENDIF

    M = M+1

45    IF (M .LE. LTCNT(PARPNT)+1) THEN

        GO TO 10

    ENDIF

    GO TO 90

ELSE

C      *****
C      *
C      * IF FLAG1 IS NOT EQUAL TO 0, THEN SCAN THROUGH THE ROW IN RBDTRK
C      * PERTAINING TO PARPNT. FIND ALL BODIES WHICH MERGED ON THE SAME SCAN
C      * LINE, AND INTERCHANGE THEIR POSITIONS IN THE ROW.
C      *
C      *****

        M = 2

50      NM = RBDTRK(PARPNT,M,1)

        IF (TYPE(NM) .EQ. BODY .AND. M .LT. (RTCNT(PARPNT)+1)) THEN

            LINE = RBDTRK(PARPNT,M,2)
            N = M+1
            MN = RBDTRK(PARPNT,M,1)

60      IF (RBDTRK(PARPNT,N,2) .EQ. LINE .AND. TYPE(MN) .EQ. BODY)
2          THEN

```

```

        IF ( N .LT. RTCNT(PARPNT)+1) THEN
            N = N+1
            GO TO 60

        ENDIF

        N = N+1

    ENDIF

    N = N-1

    DO 70 MMM = M,N

        CHGROW(MMM,1) = RBDTRK(PARPNT,N-MMM+M,1)
        CHGROW(MMM,2) = RBDTRK(PARPNT,N-MMM+M,2)

70      CONTINUE

        DO 80 MMM = M,N

            RBDTRK(PARPNT,MMM,1) = CHGROW(MMM,1)
            RBDTRK(PARPNT,MMM,2) = CHGROW(MMM,2)

80      CONTINUE

        M = N+1

        GO TO 85

    ENDIF

    M = M+1

85      IF ( M .LE. RTCNT(PARPNT)+1) THEN

        GO TO 50

    ENDIF

ENDIF

C      *****
C      *
C      * RETURN TO CALLING ROUTINE.
C      *
C      *****

90 RETURN
END
SUBROUTINE CLNUP1

C      *****
C      *
C      * SUBROUTINE CLNUP1
C      *
C      * WRITTEN BY: EDWARD C. DE METER
C      * DATE: 3-1-86
C      *
C      * PROGRAM DESCRIPTION
C      *
C      * THIS PROGRAM DOES THE FOLLOWING:
C      *

```

```

C      *      1. CYCLES THROUGH EACH POINT DESCRIBED IN A DESIRED ROW OF PNTRK,
C      *      AND
C      *
C      *      2. ELIMINATES ALL POINTS IN PNTRK THAT ARE WITHIN A 0.3 INCHES OF
C      *      OF THE DESIGNATED CORNER POINT.
C      *
C      *      SUBROUTINES USED: NONE
C      *
C      *
C      *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWRROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BND CNT(0:479),LFETR K(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTH TYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BND RTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL      AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL      AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL      PNTRK(20,250,2),PERIM(20)
REAL      SDSTAT(20,50),PNSTAT(20,50)
REAL      XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL      A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWRROW,BNDROW
COMMON OBJROW,PARENT,BND CNT,LFETR K
COMMON BDOBJ,YNGROW,DTH TYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BND RTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

M1 = 1
M2 = 2

```

```

C      *****
C      *
C      * COMPUTE THE DISTANCE BETWEEN THE POINTS IN PNTRK WHOSE RELATIVE

```

```

C      * POSITIONS WITHIN THE ARRAY ARE STORED IN M1 AND M2.
C      *
C      *****

10 XXX = PNTTRK(MARK1,M1,1)
   YYY = PNTTRK(MARK1,M1,2)
   WWW = PNTTRK(MARK1,M2,1)
   ZZZ = PNTTRK(MARK1,M2,2)

   AA = XXX-WWW
   BB = YYY-ZZZ
   CC = SQRT(AA**2 + BB**2)

C      *****
C      *
C      * IF THE DISTANCE BETWEEN THE TWO POINTS IS GREATER THAN 0.3, THEN
C      * INCREMENT M2 AND COMPUTE THE DISTANCE BETWEEN THE POINT INDICATED BY
C      * M1 AND THE NEW POINT.
C      *
C      *****

IF (CC .GT. 0.3) THEN

   M2 = M2+1

   IF (M2 .LE. PNTNUM(MARK1)) THEN

      GO TO 10

   ELSE

      M1 = M1+1

      IF (M1 .LT. PNTNUM(MARK1)) THEN

         M2 = M1+1
         GO TO 10

      ELSE

         GO TO 30

      ENDIF

   ENDIF

ELSE

C      *****
C      *
C      * IF THE DISTANCE BETWEEN THE TWO POINTS IS LESS THAN 0.3, THEN ELIMINATE
C      * THE POINT INDICATED BY M2 BY REPOSITIONING EACH POINT IN THE ARRAY
C      * PNTTRK BACK ONE POSITION, STARTING WITH POSITION M2+1.
C      *
C      *****

   IF (M2 .EQ. PNTNUM(MARK1)) THEN

      PNTNUM(MARK1) = PNTNUM(MARK1)-1
      M2 = M1+1

      IF (M1 .EQ. PNTNUM(MARK1)) THEN

         GO TO 30

      ENDIF

```

```

        GO TO 10

ENDIF

DO 20 MMM = M2,PNTNUM(MARK1)-1

    PNTTRK(MARK1,MMM,1) = PNTTRK(MARK1,MMM+1,1)
    PNTTRK(MARK1,MMM,2) = PNTTRK(MARK1,MMM+1,2)

20    CONTINUE

    PNTNUM(MARK1) = PNTNUM(MARK1)-1

    GO TO 10

ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

30 RETURN
END

SUBROUTINE CLNUP2

C *****
C *
C * SUBROUTINE CLNUP2
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-1-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *     1. CYCLES THROUGH EACH POINT DESCRIBED IN A DESIRED ROW OF PNTTRK,
C *
C *     2. FORMS TWO BOUNDARY VECTORS FROM THREE DESIGNATED CORNER POINTS,
C *         DESIGNATED POINT, DESIGNATED POINT+1,AND DESIGNATED POINT+2,
C *
C *     3. ANALYZES THE RELATIVE POSITION OF THE TWO BOUNDARY VECTORS TO
C *         DETERMINE WHETHER THE SECOND OF THE THREE POINTS IS TRULY A
C *         CORNER POINT, AND
C *
C *     4. ELIMINATES ALL FALSE CORNER POINTS FROM THE DESIRED ROW OF
C *         PNTTRK.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)

```

```

INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

M1 = 1
M2 = 2
M3 = 3
M4 = DOWN

```

```

C *****
C *
C * ASSIGN THE THREE CORNER POINTS IN PNTTRK WHICH WILL BE USED TO ESTABLISH
C * THE TWO POSITION VECTORS.
C *
C *****

```

```

10 XPNT1 = PNTTRK(MARK1,M1,1)
   YPNT1 = PNTTRK(MARK1,M1,2)
   XPNT2 = PNTTRK(MARK1,M2,1)
   YPNT2 = PNTTRK(MARK1,M2,2)
   XPNT3 = PNTTRK(MARK1,M3,1)
   YPNT3 = PNTTRK(MARK1,M3,2)

```

```

C *****
C *
C * IF THE SCAN DIRECTION IS DOWNWARD, THEN PROCEED WITH THIS LOGIC.
C *
C *****

```



```
IF (M4 .EQ. DOWN) THEN
```

```
C *****
C *
C * IF THE Y COMPONENT OF THE THIRD POINT IS LESS THAN THE Y COMPONENT OF
C * THE SECOND POINT, THEN UPDATE M1.
C *
C *****
```

```
IF (YPNT3 .LT. YPNT2) THEN
```

```
    M1 = M2
    M4 = UP
    GO TO 40
```

```
ENDIF
```

```
    X1 = XPNT1-XPNT2
    X2 = XPNT2-XPNT3
```

```
C *****
C *
C * IF THE VECTORS X1 AND X2 HAVE X COMPONENTS WITH DIFFERENT SIGNS, THEN
C * UPDATE M1.
C *
C *****
```

```
2 IF (X1 .GT. 0 .AND. X2 .LT. 0 .OR. X1 .LT. 0 .AND. X2 .GT. 0)
   THEN
```

```
    M1 = M2
    GO TO 40
```

```
ENDIF
```

```
    X1 = ABS(X1)
    X2 = ABS(X2)
    Y1 = YPNT2-YPNT1
    Y2 = YPNT3-YPNT2
    A = SQRT(X1**2 +Y1**2)
    B = SQRT(X2**2 +Y2**2)
    THETA = ACOS((X1*Y2 +X2*Y1)/(A*B))
```

```
C *****
C *
C * IF THE ANGLE BETWEEN THE VECTORS X1 AND X2 IS GREATER THAN 1.48 RADIANS
C * THEN UPDATE M1. OTHERWISE, ELIMINATE THE POINT POINTED TO IN PNTTRK BY
C * M2.
C *
C *****
```

```
IF ( THETA .GT. 1.48353) THEN
```

```
    M1 = M2
    GO TO 40
```

```
ELSE
```

```
    GO TO 20
```

```
ENDIF
```

```
ENDIF
```

```

C *****
C *
C * IF THE SCAN DIRECTION IS UPWARD, THEN PROCEED WITH THIS LOGIC.
C *
C *****

IF (M4 .EQ. UP) THEN

C *****
C *
C * IF THE Y COMPONENT OF THE SECOND POINT IS LESS THAN THE Y COMPONENT OF
C * THIRD POINT, THEN UPDATE M1.
C *
C *****

    IF (YPNT3 .GT. YPNT2) THEN

        M1 = M2
        M4 = DOWN
        GO TO 40

    ENDIF

    X1 = XPNT1-XPNT2
    X2 = XPNT2-XPNT3

C *****
C *
C * IF THE VECTORS X1 AND X2 HAVE X COMPONENTS WITH DIFFERENT SIGNS, THEN
C * UPDATE M1.
C *
C *****

    IF (X1 .GT. 0 .AND. X2 .LT. 0 .OR. X1 .LT. 0 .AND. X2 .GT. 0)
2 THEN

        M1 = M2
        GO TO 40

    ENDIF

    X1 = ABS(X1)
    X2 = ABS(X2)
    X1 = -X1
    X2 = -X2
    Y1 = YPNT2-YPNT1
    Y2 = YPNT3-YPNT2
    A = SQRT(X1**2 +Y1**2)
    B = SQRT(X2**2 +Y2**2)
    THETA = ACOS((X1*Y2 +X2*Y1)/(A*B))

C *****
C *
C * IF THE ANGLE BETWEEN THE VECTORS X1 AND X2 IS GREATER THAN 1.48 RADIANS
C * THEN UPDATE M1. OTHERWISE, ELIMINATE THE POINT POINTED TO IN PNTRK BY
C * M2.
C *
C *****

    IF ( THETA .GT. 1.48353) THEN

        M1 = M2
        GO TO 40

```

```

ELSE
    GO TO 20
ENDIF
ENDIF

C *****
C *
C * ELIMINATE THE POINT IN PNTTRK POINTED TO BY M2 BY REPOSITIONING EACH
C * SUBSEQUENT POINT ONE POSITION BACKWARD.
C *
C *****

20 DO 30 MMM = M2,PNTNUM(MARK1)-1

    PNTTRK(MARK1,MMM,1) = PNTTRK(MARK1,MMM+1,1)
    PNTTRK(MARK1,MMM,2) = PNTTRK(MARK1,MMM+1,2)

30 CONTINUE

    PNTNUM(MARK1) = PNTNUM(MARK1)-1

40 M2 = M1+1
    M3 = M1+2

    IF (M3 .LE. PNTNUM(MARK1)) THEN

        GO TO 10

    ENDIF

C *****
C *
C * USING THE IDENTICAL CRITERIA ABOVE, DETERMINE WHETHER THE LAST POINT
C * IN PNTTRK SHOULD BE ELIMINATED BY COMPARING IT TO THE FIRST POINT IN
C * PNTTRK AND THE SECOND TO THE LAST POINT IN PNTTRK.
C *
C *****

M1 = PNTNUM(MARK1)-1
M2 = PNTNUM(MARK1)

XPNT1 = PNTTRK(MARK1,M1,1)
YPNT1 = PNTTRK(MARK1,M1,2)
XPNT2 = PNTTRK(MARK1,M2,1)
YPNT2 = PNTTRK(MARK1,M2,2)
XPNT3 = PNTTRK(MARK1,1,1)
YPNT3 = PNTTRK(MARK1,1,2)

X1 = XPNT1-XPNT2
X2 = XPNT2-XPNT3

IF (YPNT2 .GT. YPNT1 .AND. YPNT2 .GT. YPNT3) THEN

    GO TO 50

ENDIF

IF (X1 .GT. 0 .AND. X2 .LT. 0 .OR. X1 .LT. 0 .AND. X2 .GT. 0)
2THEN

    GO TO 50

```

ENDIF

X1 = ABS(X1)
X2 = ABS(X2)
X1 = -X1
X2 = -X2
Y1 = YPNT2-YPNT1
Y2 = YPNT3-YPNT2
A = SQRT(X1**2 +Y1**2)
B = SQRT(X2**2 +Y2**2)
THETA = ACOS((X1*Y2 +X2*Y1)/(A*B))

IF (THETA .GT. 1.48353) THEN

GO TO 50

ELSE

PNTNUM(MARK1) = PNTNUM(MARK1)-1

ENDIF

```
C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****
```

50 RETURN
END

SUBROUTINE RTESTT

```
C *****
C *
C * SUBROUTINE RTESTT
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-2-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C * 1. CYCLES THROUGH EACH POINT DESCRIBED IN A DESIRED ROW OF PNTTRK,
C *
C * 2. COMPUTES THE DISTANCE BETWEEN THE CORNER POINT AND THE CENTROID
C *    OF THE OBJECT, AND
C *
C * 3. COMPUTES THE LENGTH OF EACH SIDE OF THE OBJECT.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****
```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)

```

INTEGER OBJROW(40,2),PARENT(20,10),BND CNT(0:479),LFETR K(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTH TYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BND RTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YOMMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEUROW,BNDROW
COMMON OBJROW,PARENT,BND CNT,LFETR K
COMMON BDOBJ,YNGROW,DTH TYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BND RTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * COMPUTE THE DISTANCE BETWEEN CONSECUTIVE POINTS IN PNTTRK.
C *
C *****

```

```

PERIM(MARK1) = 0

DO 10 M2 = 2,PNTNUM(MARK1)

M1 = M2-1
XXX = PNTTRK(MARK1,M1,1)
YYY = PNTTRK(MARK1,M1,2)
WWW = PNTTRK(MARK1,M2,1)
ZZZ = PNTTRK(MARK1,M2,2)

AA = XXX-WWW
BB = YYY-ZZZ
CC = SQRT(AA**2 + BB**2)

SDSTAT(MARK1,M1) = CC
PERIM(MARK1) = PERIM(MARK1) +CC

```

10 CONTINUE

```
M2 = PNTNUM(MARK1)
XXX = PNTTRK(MARK1,1,1)
YYY = PNTTRK(MARK1,1,2)
WWW = PNTTRK(MARK1,M2,1)
ZZZ = PNTTRK(MARK1,M2,2)
```

```
AA = XXX-WWW
BB = YYY-ZZZ
CC = SQRT(AA**2 + BB**2)
```

```
SDSTAT(MARK1,M2) = CC
PERIM(MARK1) = PERIM(MARK1) +CC
```

```
MMM = BDOBJ(MARK1)
```

```
C *****
C *
C * COMPUTE THE DISTANCE BETWEEN EACH POINT IN PNTTRK AND THE CENTROID
C * OF THE OBJECT.
C *
C *****
```

```
DO 20 M1 = 1,PNTNUM(MARK1)
```

```
XXX = PNTTRK(MARK1,M1,1)
YYY = PNTTRK(MARK1,M1,2)
WWW = XMOMNT(MMM)
ZZZ = YMOMNT(MMM)
```

```
AA = XXX-WWW
BB = YYY-ZZZ
CC = SQRT(AA**2 + BB**2)
```

```
PNSTAT(MARK1,M1) = CC
```

20 CONTINUE

```
C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****
```

```
RETURN
END
SUBROUTINE ASEROR
```

```
C *****
C *
C * SUBROUTINE ASEROR
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-4-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C * 1. CHECKS THE AREA, NUMBER OF SIDES, AND PERIMETER OF EACH
C * PARENT OBJECT TO DETERMINE IF THAT OBJECT IS AN "A" TYPE
C * OBJECT,
C *
C * 2. IDENTIFIES THE KEY CORNER POINT, WHICH IS USED TO ESTABLISH
```

```

C      *      THE ORIENTATION OF THE "A" TYPE OBJECT, AND
C      *
C      *      3. STORES THE ELEMENT NUMBERS OF THE THREE KEY BOUNDARY POINTS
C      *      OF THE IN THE ARRAY, ATRK.
C      *
C      *
C      * SUBROUTINES USED: NONE
C      *
C      *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETR(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBSCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCH,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL      AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL      AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL      PNTTRK(20,250,2),PERIM(20)
REAL      SDSTAT(20,50),PNSTAT(20,50)
REAL      XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL      A,B,THETA,PHI,MAG,SIGMA

C
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROM,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETR
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBSCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

ANUM = 0

DO 20 MARK1 = 1,BDCNT

MMM = BDOBJ(MARK1)

C      *****
C      *

```

```

C      * IF THE OBJECT'S AREA IS WITHIN THE BOUNDS OF AN "A" TYPE OBJECT, THEN
C      * CHECK ITS NUMBER OF CORNER POINTS.
C      *
C      *****

      IF (AREA(MMM) .GE. 25.0 .AND. AREA(MMM) .LE. 28.5) THEN

          GO TO 30

      ELSE

          GO TO 20

      ENDIF

C      *****
C      *
C      * IF THE OBJECT'S NUMBER OF CORNER POINTS EQUALS 6, THEN CHECK ITS
C      * PERIMETER.
C      *
C      *****

30 IF (PNTNUM(MARK1) .EQ. 6) THEN

      GO TO 40

  ELSE

      GO TO 20

  ENDIF

C      *****
C      *
C      * IF THE OBJECT'S PERIMETER IS WITHIN THE BOUNDS OF AN "A" TYPE OBJECT,
C      * THEN INCREMENT ANUM AND DETERMINE THE LOCATION OF THE KEY POINT.
C      *
C      *****

40 IF (PERIM(MARK1) .GE. 19.5 .AND. PERIM(MARK1) .LE. 23.5) THEN

      ANUM = ANUM+1

      ATRK(ANUM,1) = MARK1

      MARK3 = 1

      DO 50 MARK2 = 1,6

          IF (PNSTAT(MARK1,MARK3) .GT. PNSTAT(MARK1,MARK2) ) THEN

              MARK3 = MARK2

          ENDIF

50      CONTINUE

      IF ( MARK3 .LE. 3) THEN

          ATRK(ANUM,2) = MARK3+1
          ATRK(ANUM,3) = MARK3+2
          ATRK(ANUM,4) = MARK3+3

      ENDIF

      IF (MARK3 .EQ. 4) THEN

```



```

      ATRK(ANUM,2) = 5
      ATRK(ANUM,3) = 6
      ATRK(ANUM,4) = 1

```

```

ENDIF

```

```

IF (MARK3 .EQ. 5) THEN

```

```

      ATRK(ANUM,2) = 6
      ATRK(ANUM,3) = 1
      ATRK(ANUM,4) = 2

```

```

ENDIF

```

```

IF (MARK3 .EQ. 6) THEN

```

```

      ATRK(ANUM,2) = 1
      ATRK(ANUM,3) = 2
      ATRK(ANUM,4) = 3

```

```

ENDIF

```

```

ENDIF

```

```

20 CONTINUE

```

```

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

```

```

RETURN
END

```

```

SUBROUTINE BSEROR

```

```

C *****
C *
C * SUBROUTINE BSEROR
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-4-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *   1. CHECKS THE AREA, NUMBER OF SIDES, AND PERIMETER OF EACH
C *     PARENT OBJECT TO DETERMINE IF THAT OBJECT IS AN "B" TYPE
C *     OBJECT,
C *
C *   2. IDENTIFIES THE KEY CORNER POINT, WHICH IS USED TO ESTABLISH
C *     THE ORIENTATION OF THE "B" TYPE OBJECT, AND
C *
C *   3. STORES THE ELEMENT NUMBERS OF THE THREE KEY BOUNDARY POINTS
C *     OF THE IN THE ARRAY, BTRK.
C *
C *
C * SUBROUTINES USED: NONE
C *
C *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROM,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

BNUM = 0

DO 20 MARK1 = 1,BDCNT

MMM = BDOBJ(MARK1)

```

C *****
C *
C * IF THE OBJECT'S AREA IS WITHIN THE BOUNDS OF AN "B" TYPE OBJECT, THEN
C * CHECK ITS NUMBER OF CORNER POINTS.
C *
C *****

```

IF (AREA(MMM) .GE. 37.0 .AND. AREA(MMM) .LE. 43.5)THEN

GO TO 30

```

ELSE
    GO TO 20
ENDIF

C *****
C *
C * IF THE OBJECT'S NUMBER OF CORNER POINTS EQUALS 6, THEN CHECK ITS
C * PERIMETER.
C *
C *****

30 IF (PNTNUM(MARK1) .EQ. 6) THEN
    GO TO 40
ELSE
    GO TO 20
ENDIF

C *****
C *
C * IF THE OBJECT'S PERIMETER IS WITHIN THE BOUNDS OF AN "B" TYPE OBJECT,
C * THEN INCREMENT ANUM AND DETERMINE THE LOCATION OF THE KEY POINT.
C *
C *****

40 IF (PERIM(MARK1) .GE. 25.0 .AND. PERIM(MARK1) .LE. 30.5) THEN
    BNUM = BNUM+1
    BTRK(BNUM,1) = MARK1
    MARK3 = 1
    DO 50 MARK2 = 1,6
    IF (PNSTAT(MARK1,MARK3) .GT. PNSTAT(MARK1,MARK2) ) THEN
        MARK3 = MARK2
    ENDIF
50 CONTINUE
    IF ( MARK3 .LE. 3) THEN
        BTRK(BNUM,2) = MARK3+1
        BTRK(BNUM,3) = MARK3+2
        BTRK(BNUM,4) = MARK3+3
    ENDIF
    IF (MARK3 .EQ. 4) THEN
        BTRK(BNUM,2) = 5
        BTRK(BNUM,3) = 6
        BTRK(BNUM,4) = 1
    ENDIF

```

```
IF (MARK3 .EQ. 5) THEN
```

```
  BTRK(BNUM,2) = 6  
  BTRK(BNUM,3) = 1  
  BTRK(BNUM,4) = 2
```

```
ENDIF
```

```
IF (MARK3 .EQ. 6) THEN
```

```
  BTRK(BNUM,2) = 1  
  BTRK(BNUM,3) = 2  
  BTRK(BNUM,4) = 3
```

```
ENDIF
```

```
ENDIF
```

```
20 CONTINUE
```

```
C *****  
C *  
C * RETURN TO CALLING ROUTINE  
C *  
C *****
```

```
RETURN  
END
```

APPENDIX C. TACTILE SYSTEM SOFTWARE

SUBROUTINE TACTIL

```
C *****
C *
C * SUBROUTINE TACTIL
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-10-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *   1. CONVERTS THE CORNER POINTS OF ALL "A" TYPE AND "B" TYPE OBJECTS
C *     FROM VISION COORDINATES TO 7545 ROBOT WORCKCELL COORDINATES,
C *
C *   2. CALLS UPON THE 7545 ROBOT TO PERFORM A TACTILE SEARCH OF
C *     THE DESIRED OBJECT TO LOCATE TWO POINTS FOR EACH OF THE
C *     DEFINING SIDES,
C *
C *   3. COMPUTES THE LOCATION OF THE CORNER POINTS WHICH DEFINE THE
C *     REGION PERTAINING TO THE DESIRED OBJECT, AND
C *
C *   4. RECOMPUTES THE LOCATION OF THE REGIONAL CORNER POINTS INORDER
C *     TO ACCOUNT FOR THE ECCENTRICITY OF THE TACTILE PROBE.
C *
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C *   1. RECOR
C *   2. EXEC
C *   3. GRAPH1
C *   4. VECDEF
C *   5. CNTRL
C *   6. ANGCLC
C *
C *****

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRT(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
```

```

REAL    SDSTAT( 20,50 ),PNSTAT( 20,50 )
REAL    XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL    A,B,THETA,PHI,MAG,SIGMA

C
COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROW,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

C *****
C *
C * CONVERT THE COORDINATES IN PNTTRK INTO ROBOT WORKCELL COORDINATES.
C *
C *****

CALL RECOR

C *****
C *
C * RETURN THE ROBOT TO HOME, PLACE THE CONTROL IN THE AUTO MODE, AND
C * POINT TO APPLICATION 2 (TTOUCH).
C *
C *****

CALL EXEC(RTRN,ERROR)
CALL EXEC(AUTO,ERROR)
CALL EXEC(APP2,ERROR)

C *****
C *
C * IF THERE EXISTS ONE OR MORE "A" TYPE OBJECTS, THEN CALL VECDEF TO
C * DETERMINE THE ACTUAL POINTS ALONG THE OBJECT'S SIDES.
C *
C *****

IF (ANUM .GT. 0) THEN

    LINNUM = 2
    CALL GRAPH1(LINNUM)

    DO 10 MARK1 = 1,ANUM

    MARK5 = 1
    CALL VECDEF

10    CONTINUE

```

```

ENDIF

C *****
C *
C * IF THERE EXISTS ONE OR MORE "B" TYPE OBJECTS, THEN CALL VECDEF TO
C * DETERMINE THE ACTUAL POINTS ALONG THE OBJECT'S SIDES.
C *
C *****

IF (BNUM .GT. 0) THEN

    LINNUM = 3
    CALL GRAPH1(LINNUM)

    DO 20 MARK1 = 1,BNUM

        MARK5 = 2
        CALL VECDEF

20    CONTINUE

ENDIF

C *****
C *
C * RESET THE CONTROLLER
C *
C *****

CALL CNTRL(RESET,ERROR)

C *****
C *
C * IF THE NUMBER OF "A" TYPE OBJECTS EXCEEDS ZERO, THEN COMPUTE THE REGION
C * BOUNDARIES FOR EACH OBJECT, TAKING INTO ACCOUNT THE ECCENTRICITY OF THE
C * TACTILE PROBE
C *
C *****

IF (ANUM .GT. 0) THEN

    DO 30 MARK1 = 1,ANUM

        XPNT1 = ABND(MARK1,1,1)
        YPNT1 = ABND(MARK1,1,2)
        XPNT2 = ABND(MARK1,2,1)
        YPNT2 = ABND(MARK1,2,2)
        XPNT3 = ABND(MARK1,3,1)
        YPNT3 = ABND(MARK1,3,2)
        XPNT4 = ABND(MARK1,4,1)
        YPNT4 = ABND(MARK1,4,2)

        S1 = (YPNT1-YPNT2)/(XPNT1-XPNT2)
        S2 = (YPNT4-YPNT3)/(XPNT4-XPNT3)
        B1 = YPNT1-(S1*XPNT1)
        B2 = YPNT3-(S2*XPNT3)

        X = (B2-B1)/(S1-S2)
        Y = (S1*X)+B1

        X2 = XPNT1-X
        Y2 = YPNT1-Y
        R2 = SQRT(X2**2 + Y2**2)
        X2 = X2/R2
        Y2 = Y2/R2

```

```

X1 = XPNT4-X
Y1 = YPNT4-Y
R1 = SQRT(X1**2 + Y1**2)
X1 = X1/R1
Y1 = Y1/R1

CALL ANGCLC

THETA = THETA + PHI
THETA = THETA + 3.141592
X3 = MAG*COS(THETA)
Y3 = MAG*SIN(THETA)

ABND(MARK1,2,1) = X + (6.0000*X2) + X3
ABND(MARK1,2,2) = Y + (6.0000*Y2) + Y3
ABND(MARK1,1,1) = X + X3
ABND(MARK1,1,2) = Y + Y3
ABND(MARK1,3,1) = X + (4.5000*X1) + X3
ABND(MARK1,3,2) = Y + (4.5000*Y1) + Y3
ABND(MARK1,4,1) = ABND(MARK1,3,1) + (6.0000*X2) + X3
ABND(MARK1,4,2) = ABND(MARK1,3,2) + (6.0000*Y2) + Y3

```

30 CONTINUE

ENDIF

```

C *****
C *
C * IF THE NUMBER OF "B" TYPE OBJECTS EXCEEDS ZERO, THEN COMPUTE THE REGION
C * BOUNDARIES FOR EACH OBJECT, TAKING INTO ACCOUNT THE ECCENTRICITY OF THE
C * TACTILE PROBE
C *
C *****

```

IF (BNUM .GT. 0) THEN

DO 40 MARK1 = 1,BNUM

```

XPNT1 = BBND(MARK1,1,1)
YPNT1 = BBND(MARK1,1,2)
XPNT2 = BBND(MARK1,2,1)
YPNT2 = BBND(MARK1,2,2)
XPNT3 = BBND(MARK1,3,1)
YPNT3 = BBND(MARK1,3,2)
XPNT4 = BBND(MARK1,4,1)
YPNT4 = BBND(MARK1,4,2)

```

```

S1 = (YPNT1-YPNT2)/(XPNT1-XPNT2)
S2 = (YPNT4-YPNT3)/(XPNT4-XPNT3)
B1 = YPNT1-(S1*XPNT1)
B2 = YPNT3-(S2*XPNT3)

```

```

X = (B2-B1)/(S1-S2)
Y = (S1*X)+B1

```

```

YPNT1 = B1 + S1*XPNT1
YPNT4 = B2 + S2*XPNT4

```

```

X2 = XPNT1-X
Y2 = YPNT1-Y
R2 = SQRT(X2**2 + Y2**2)
X2 = X2/R2
Y2 = Y2/R2

```



```

X1 = XPNT4-X
Y1 = YPNT4-Y
R1 = SQRT(X1**2 + Y1**2)
X1 = X1/R1
Y1 = Y1/R1
CALL ANGCLC

```

```

THETA = THETA + PHI
THETA = THETA + 3.141592
X3 = MAG*COS(THETA)
Y3 = MAG*SIN(THETA)

```

```

BBND(MARK1,2,1) = X + (9.0000*X2) + X3
BBND(MARK1,2,2) = Y + (9.0000*Y2) + Y3
BBND(MARK1,1,1) = X + X3
BBND(MARK1,1,2) = Y + Y3
BBND(MARK1,3,1) = X + (4.5000*X1) + X3
BBND(MARK1,3,2) = Y + (4.5000*Y1) + Y3
BBND(MARK1,4,1) = BBND(MARK1,3,1) + (9.0000*X2) + X3
BBND(MARK1,4,2) = BBND(MARK1,3,2) + (9.0000*Y2) + Y3

```

40 CONTINUE

ENDIF

```

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

```

50 RETURN
END

SUBROUTINE VECDEF

```

C *****
C *
C * SUBROUTINE VECDEF
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3.-18-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C * 1. COMPUTES TWO VECTORS WHICH ARE NORMAL TO A VECTOR DEFINED BY
C *    TWO CORNER POINTS OBTAINED BY THE VISION SYSTEM,
C *
C * 2. COMMANDS THE 7545 ROBOT TO CONDUCT A PROBE SEARCH ALONG THE
C *    TWO VECTORS,
C *
C * 3. USES THE TWO POINTS OBTAINED BY THE INITIAL TACTILE SEARCH OF
C *    THE BOUNDING SIDE TO COMPUTE A NEW BOUNDING VECTOR,
C *
C * 4. COMPUTES TWO VECTORS WHICH ARE NORMAL TO THE COMPUTED BOUNDING
C *    VECTOR,
C *
C * 5. COMMANDS THE 7545 ROBOT TO CONDUCT A PROBE SEARCH ALONG THE
C *    TWO NEW VECTORS,
C *
C * 6. RECORDS TWO POINTS OBTAINED BY THE TACTILE SEARCH OF THE

```

```

C      *      BOUNDING SIDE IN EITHER ADEF OR BDEF, DEPENDING UPON THE
C      *      TYPE OF THE OBJECT, AND
C      *
C      *      7. PERFORMS OPERATIONS 1-6 ON THE OTHER BOUNDING SIDE OF THE
C      *      OBJECT.
C      *
C      *      SUBROUTINES USED
C      *
C      *      THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C      *
C      *      1. TOUCH
C      *
C      *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCH,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL      AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL      AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL      PNTTRK(20,250,2),PERIM(20)
REAL      SDSTAT(20,50),PNSTAT(20,50)
REAL      XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL      A,B,THETA,PHI,MAG,SIGMA

```

```

C
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMPLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

C      *****
C      *

```

```

C      * IF MARK5 EQUALS 1, THEN SEARCH FOR AN "A" TYPE OBJECT.  OTHERWISE,
C      * SEARCH FOR A "B" TYPE OBJECT.
C      *
C      *****

IF (MARK5 .EQ. 1) THEN

    MM = ATRK(MARK1,1)

    NN = ATRK(MARK1,2)
    XXPNT1 = PNTTRK(MM,NN,1)
    YYPNT1 = PNTTRK(MM,NN,2)
    NN = ATRK(MARK1,3)
    XXPNT2 = PNTTRK(MM,NN,1)
    YYPNT2 = PNTTRK(MM,NN,2)
    NN = ATRK(MARK1,4)
    XXPNT3 = PNTTRK(MM,NN,1)
    YYPNT3 = PNTTRK(MM,NN,2)

ELSE

    MM = BTRK(MARK1,1)

    NN = BTRK(MARK1,2)
    XXPNT1 = PNTTRK(MM,NN,1)
    YYPNT1 = PNTTRK(MM,NN,2)

    NN = BTRK(MARK1,3)
    XXPNT2 = PNTTRK(MM,NN,1)
    YYPNT2 = PNTTRK(MM,NN,2)

    NN = BTRK(MARK1,4)
    XXPNT3 = PNTTRK(MM,NN,1)
    YYPNT3 = PNTTRK(MM,NN,2)

ENDIF

C      *****
C      *
C      * SET UP THE STARTING POINTS AND SEARCH VECTORS NEEDED TO SEARCH FOR
C      * TWO POINTS ON THE FIRST SIDE OF THE OBJECT.
C      *
C      *****

XX1 = XXPNT2-XXPNT1
YY1 = YYPNT2-YYPNT1
RR = SQRT(XX1**2 + YY1**2)
XX1 = XX1/RR
YY1 = YY1/RR

XPNT2 = XXPNT1 + (0.15*RR*XX1)
YPNT2 = YYPNT1 + (0.15*RR*YY1)

IF (YYPNT3 .LT. YYPNT2) THEN

    Y1 = 0.50

ELSE

    Y1 = -0.50

ENDIF

X1 = -Y1*YY1/XX1

```

```

XPNT1 = XPNT2 + X1
YPNT1 = YPNT2 + Y1
SXPNT1 = XPNT1
SYPNT1 = YPNT1
RRR = SQRT(X1**2 + Y1**2)
X1 = (-X1/RRR)/20.0
Y1 = (-Y1/RRR)/20.0
Y3 = -8.516
THETA = 0.0
FLAG1 = 1

C *****
C *
C * SEARCH FOR THE 1ST POINT ON SIDE 1
C *
C *****

CALL TOUCH

TXPNT1 = XPNT1
TYPNT1 = YPNT1

XPNT2 = XXPNT1 + (0.85*RR*XX1)
YPNT2 = YYPNT1 + (0.85*RR*YY1)

IF (YYPNT3 .LT. YYPNT2) THEN

    Y1 = 0.50

ELSE

    Y1 = -0.50

ENDIF

X1 = -Y1*YY1/XX1

XPNT1 = XPNT2 + X1
YPNT1 = YPNT2 + Y1
SXPNT2 = XPNT1
SYPNT2 = YPNT1
RRR = SQRT(X1**2 + Y1**2)
X1 = (-X1/RRR)/20.0
Y1 = (-Y1/RRR)/20.0
Y3 = -8.516
THETA = 0.0
FLAG1 = 1

C *****
C *
C * SEARCH FOR THE 2ND POINT ON SIDE 1
C *
C *****

CALL TOUCH

TXPNT2 = XPNT1
TYPNT2 = YPNT1

C *****
C *
C * RECOMPUTE THE VECTOR THAT DEFINES THE SIDE OF THE OBJECT, USING THE
C * POINTS FOUND DURING THE INITIAL TACTILE PROBE SEARCH. FOLLOW UP BY
C * PERFORMING ANOTHER SEARCH USING THE NEW PARAMETERS. THIS MAXIMIZES THE

```

```

C      * ACCURACY OF THE SEARCH.
C      *
C      *****

      XX1 = TXPNT2 - TXPNT1
      YY1 = TYPNT2 - TYPNT1
      RR  = SQRT(XX1**2 + YY1**2)
      XX1 = XX1/RR
      YY1 = YY1/RR

      IF (YYPNT3 .LT. YYPNT2) THEN

          Y1 = -0.50

      ELSE

          Y1 = 0.50

      ENDIF

      X1 = -Y1*YY1/XX1
      RRR = SQRT(X1**2 + Y1**2)
      X1 = (X1/RRR)/20.0
      Y1 = (Y1/RRR)/20.0
      X2 = X1
      Y2 = Y1

      XPNT1 = SXPNT1
      YPNT1 = SYPNT1
      Y3 = -8.516
      THETA = 0.0
      FLAG1 = 1

C      *****
C      *
C      * SEARCH FOR THE 1ST POINT ON SIDE 1
C      *
C      *****

      CALL TOUCH

      IF (MARK5 .EQ. 1) THEN

          ABND(MARK1,1,1) = XPNT1
          ABND(MARK1,1,2) = YPNT1

      ELSE

          BBND(MARK1,1,1) = XPNT1
          BBND(MARK1,1,2) = YPNT1

      ENDIF

      X1 = X2
      Y1 = Y2
      XPNT1 = SXPNT2
      YPNT1 = SYPNT2
      Y3 = -8.516
      THETA = 0.0
      FLAG1 = 1

C      *****
C      *
C      * SEARCH FOR THE 2ND POINT ON SIDE 1
C      *
C      *****

```

```

CALL TOUCH

IF (MARK5 .EQ. 1) THEN

    ABND(MARK1,2,1) = XPNT1
    ABND(MARK1,2,2) = YPNT1

ELSE

    BBND(MARK1,2,1) = XPNT1
    BBND(MARK1,2,2) = YPNT1

ENDIF

C *****
C *
C * SET UP THE STARTING POINTS AND SEARCH VECTORS NEEDED TO SEARCH FOR
C * TWO POINTS ON THE SECOND SIDE OF THE OBJECT.
C *
C *****

XX1 = XXPNT3-XXPNT2
YY1 = YYPNT3-YYPNT2
RR = SQRT(XX1**2 + YY1**2)
XX1 = XX1/RR
YY1 = YY1/RR

XPNT2 = XXPNT2 + (0.15*RR*XX1)
YPNT2 = YYPNT2 + (0.15*RR*YY1)

IF (XXPNT1 .LT. XXPNT2) THEN

    X1 = 0.50

ELSE

    X1 = -0.50

ENDIF

Y1 = -X1*XX1/YY1
XPNT1 = XPNT2 + X1
YPNT1 = YPNT2 + Y1
SXPNT1 = XPNT1
SYPNT1 = YPNT1
RRR = SQRT(X1**2 + Y1**2)
X1 = (-X1/RRR)/20.0
Y1 = (-Y1/RRR)/20.0
Y3 = -8.516
THETA = 0.0
FLAG1 = 1
C *****
C *
C * SEARCH FOR THE 1ST POINT ON SIDE 2
C *
C *****

CALL TOUCH

TXPNT1 = XPNT1
TYPNT1 = YPNT1

```

```

XPNT2 = XXPNT2 + (0.85*RR*XX1)
YPNT2 = YYPNT2 + (0.85*RR*YY1)

```

```

IF (XXPNT1 .LT. XXPNT2) THEN

```

```

    X1 = 0.50

```

```

ELSE

```

```

    X1 = -0.50

```

```

ENDIF

```

```

Y1 = -X1*XX1/YY1

```

```

XPNT1 = XPNT2 + X1
YPNT1 = YPNT2 + Y1
SXPNT2 = XPNT1
SYPNT2 = YPNT1
RRR = SQRT(X1**2 + Y1**2)
X1 = (-X1/RRR)/20.0
Y1 = (-Y1/RRR)/20.0
Y3 = -8.516
THETA = 0.0
FLAG1 = 1

```

```

C *****
C *
C * SEARCH FOR THE 2ND POINT ON SIDE 2
C *
C *****

```

```

CALL TOUCH

```

```

TXPNT2 = XPNT1
TYPNT2 = YPNT1

```

```

C *****
C *
C * RECOMPUTE THE VECTOR THAT DEFINES THE SIDE OF THE OBJECT, USING THE
C * POINTS FOUND DURING THE INITIAL TACTILE PROBE SEARCH. FOLLOW UP BY
C * PERFORMING ANOTHER SEARCH USING THE NEW PARAMETERS. THIS MAXIMIZES THE
C * ACCURACY OF THE SEARCH.
C *
C *****

```

```

XX1 = TXPNT2 - TXPNT1
YY1 = TYPNT2 - TYPNT1
RR = SQRT(XX1**2 + YY1**2)
XX1 = XX1/RR
YY1 = YY1/RR

```

```

IF (XXPNT1 .LT. XXPNT2) THEN

```

```

    X1 = -0.50

```

```

ELSE

```

```

    X1 = 0.50

```

```

ENDIF

```

```

Y1 = -X1*XX1/YY1
RRR = SQRT(X1**2 + Y1**2)
X1 = (X1/RRR)/20.0
Y1 = (Y1/RRR)/20.0
X2 = X1
Y2 = Y1

XPNT1 = SXPNT1
YPNT1 = SYPNT1
Y3 = -8.516
THETA = 0.0
FLAG1 = 1

C *****
C *
C * SEARCH FOR THE 1ST POINT ON SIDE 2
C *
C *****

CALL TOUCH

IF (MARK5 .EQ. 1) THEN

    ABND(MARK1,3,1) = XPNT1
    ABND(MARK1,3,2) = YPNT1

ELSE

    BBND(MARK1,3,1) = XPNT1
    BBND(MARK1,3,2) = YPNT1

ENDIF

X1 = X2
Y1 = Y2
XPNT1 = SXPNT2
YPNT1 = SYPNT2
Y3 = -8.516
THETA = 0.0
FLAG1 = 1

C *****
C *
C * SEARCH FOR THE 2ND POINT ON SIDE 2
C *
C *****

CALL TOUCH

IF (MARK5 .EQ. 1) THEN

    ABND(MARK1,4,1) = XPNT1
    ABND(MARK1,4,2) = YPNT1

ELSE

    BBND(MARK1,4,1) = XPNT1
    BBND(MARK1,4,2) = YPNT1

ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE

```



```

C      *
C      *****

```

```

100 RETURN
END

```

SUBROUTINE RECOR

```

C      *****
C      *
C      * SUBROUTINE RECOR
C      *
C      * WRITTEN BY: EDWARD C. DE METER
C      * DATE: 3-11-86
C      *
C      * PROGRAM DESCRIPTION
C      *
C      * THIS PROGRAM CONVERTS THE CORNER POINTS OF ALL "A" TYPE AND "B" TYPE
C      * OBJECTS FROM VISION SYSTEM COORDINATES TO 7545 ROBOT WORKCELL
C      * COORDINATES.
C      *
C      *
C      * SUBROUTINES USED: NONE
C      *
C      *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNORTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATTRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

```

C
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT

```

```

COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CH,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * IF THE NUMBER OF "A" TYPE OBJECTS EXCEEDS ZERO, THEN CONVERT THE
C * COORDINATES OF THE OBJECTS FROM VISUAL COORDINATES TO ROBOT WORKCELL
C * COORDIANTES.
C *
C *****

```

```
IF (ANUM .GT. 0) THEN
```

```
DO 20 MARK1 = 1,ANUM
```

```
MM = ATRK(ANUM,1)
```

```
DO 10 MARK2 = 1,6
```

```
RX = PNTTRK(MM,MARK2,1)
```

```
RY = PNTTRK(MM,MARK2,2)
```

```
RX = RX + 0.346
```

```
RY = RY - 25.852
```

```
PNTTRK(MM,MARK2,1) = RY
```

```
PNTTRK(MM,MARK2,2) = RX
```

```
10 CONTINUE
```

```
20 CONTINUE
```

```
ENDIF
```

```

C *****
C *
C * IF THE NUMBER OF "A" TYPE OBJECTS EXCEEDS ZERO, THEN CONVERT THE
C * COORDINATES OF THE OBJECTS FROM VISUAL COORDINATES TO ROBOT WORKCELL
C * COORDIANTES.
C *
C *****

```

```
IF (BNUM .GT. 0) THEN
```

```
DO 40 MARK1 = 1,BNUM
```

```
MM = BTRK(BNUM,1)
```

```
DO 30 MARK2 = 1,6
```

```
RX = PNTTRK(MM,MARK2,1)
```

```
RY = PNTTRK(MM,MARK2,2)
```

```
RX = RX + 0.346
```

```
RY = RY - 25.852
```

```
PNTTRK(MM,MARK2,1) = RY
```

```
PNTTRK(MM,MARK2,2) = RX
```

```
30 CONTINUE
```

40 CONTINUE

ENDIF

```
C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****
```

RETURN
END

SUBROUTINE ERCALC

```
C *****
C *
C * SUBROUTINE ERCALC
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-18-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM IS USED TO DEVELOP THE ERROR COMPENSATION PARAMETERS FOR
C * THE TACTILE PROBE BY DOING THE FOLLOWING:
C *
C * 1. COMMANDING THE 7545 ROBOT TO CONDUCT A PROBE SEARCH ALONG
C * TWO PRESCRIBED VECTORS INORDER TO MAKE CONTACT WITH THE
C * "Y" CALIBRATION SURFACE,
C *
C * 2. USING THE TWO POINTS OBTAINED IN THE INTIAL SEARCH TO COMPUTE
C * THE VECTOR REPRESENTING THE "Y" SURFACE OF THE CALIBRATION
C * OBJECT,
C *
C * 3. COMPUTING A NORMAL VECTOR TO THE "Y" VECTOR,
C *
C * 4. COMMANDING THE 7545 ROBOT TO CONDUCT A PROBE SEARCH FOR THE
C * "Y" SURFACE WITH THE PROBE ORIENTATED AT 0 DEGREES, AND THEN
C * AT 180 DEGREES,
C *
C * 5. COMMANDING THE 7545 ROBOT TO CONDUCT A PROBE SEARCH FOR THE
C * "X" SURFACE ALONG THE "Y" VECTOR WITH THE PROBE ORIENTATED
C * AT 0 DEGREES, AND THEN AT 180 DEGREES,AND
C *
C * 6. USEING THE POINTS OBTAINED FROM THE PROBE SEARCH TO COMPUTE
C * THE ERROR COMPENSATION PARAMETERS, MAG AND PHI.
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C * 1. GRAPH1
C * 2. EXEC
C * 3. TOUCH
C * 4. CNTRL
C * 5. ANGCLC
C *
C *****
```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE

```

INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEUROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMPLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEUROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMPLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

LINNUM = 6
CALL GRAPH1(LINNUM)

```

```

C *****
C *
C * RETURN THE ROBOT TO HOME, PLACE THE CONTROL IN THE AUTO MODE, AND
C * POINT TO APPLICATION 2 (TTOUCH).
C *
C *****

```

```

CALL EXEC(RTRN,ERROR)
CALL EXEC(AUTO,ERROR)
CALL EXEC(APP2,ERROR)

```

```

FLAG1 = 0

```

```

XPNT1 = -8.344
YPNT1 = 17.717
THETA = 0.0
X1 = 0.05
Y1 = 0.00

```

```

Y3 = -8.547

C *****
C *
C * SEARCH FOR THE 1ST POINT ON THE 1ST SIDE OF THE CALIBRATION OBJECT.
C *
C *****

CALL TOUCH

XPT1 = XPNT1
YPT1 = YPNT1

XPNT1 = -8.344
YPNT1 = 14.386
X1 = 0.05
Y1 = 0.00
THETA = 0.0
Y3 = -8.547

C *****
C *
C * SEARCH FOR THE 2ND POINT ON THE 1ST SIDE OF THE CALIBRATION OBJECT.
C *
C *****

CALL TOUCH

XPT2 = XPNT1
YPT2 = YPNT1

C *****
C *
C * COMPUTE THE STARTING POINT AND VECTORS NEEDED TO CONDUCT THE CALIBRATION
C * OF THE TACTILE PROBE.
C *
C *****

XX1 = XPT2-XPT1
YY1 = YPT2-YPT1
RR = SQRT(XX1**2 +YY1**2)
XX1 = XX1/RR
YY1 = YY1/RR

XPNT2 = XPT1+ (0.5*RR*XX1)
YPNT2 = YPT1+ (0.5*RR*YY1)

X1 = -0.3
Y1 = -X1*XX1/YY1
XPNT2 = XPNT2 + X1
YPNT2 = YPNT2 + Y1
RRR = SQRT(X1**2 + Y1**2)
X2 = (-X1/RRR)/20.0
Y2 = (-Y1/RRR)/20.0

XPNT1 = XPNT2
YPNT1 = YPNT2
X1 = X2
Y1 = Y2
THETA = 0.0
Y3 = -8.547

C *****
C *

```

```

C  * WITH THE PROBE ORIENTATED AT 0 DEGREE, SEARCH FOR THE SIDE OF THE 1ST
C  * SIDE OF THE CALIBRATION OBJECT.
C  *
C  *****

CALL TOUCH

XXX1 = XPNT1
YYY1 = YPNT1
DX = ABS(XXX1 - XPNT2)
DY = ABS(YYY1 - YPNT2)
R1 = SQRT(DX**2 + DY**2)

XPNT1 = XPNT2
YPNT1 = YPNT2
X1 = X2
Y1 = Y2
THETA = 180.0
Y3 = -8.547

C  *****
C  *
C  * WITH THE PROBE ORIENTATED AT 180 DEGREE, SEARCH FOR THE SIDE OF THE
C  * 1ST SIDE OF THE CALIBRATION OBJECT.
C  *
C  *****

CALL TOUCH

XXX2 = XPNT1
YYY2 = YPNT1
DX = ABS(XXX2 - XPNT2)
DY = ABS(YYY2 - YPNT2)
R2 = SQRT(DX**2 + DY**2)

XPNT1 = -7.303
YPNT1 = 18.225
X1 = XX1/20.0
Y1 = YY1/20.0
THETA = 0.0
Y3 = -8.547

C  *****
C  *
C  * WITH THE PROBE ORIENTATED AT 0 DEGREE, SEARCH FOR THE SIDE OF THE 2ND
C  * SIDE OF THE CALIBRATION OBJECT.
C  *
C  *****

CALL TOUCH

XXX3 = XPNT1
YYY3 = YPNT1
DX = ABS(XXX3 + 7.303)
DY = ABS(YYY3 - 18.225)
R3 = SQRT(DX**2 + DY**2)

XPNT1 = -7.303
YPNT1 = 18.225
X1 = XX1/20.0
Y1 = YY1/20.0
THETA = 180.0
Y3 = -8.547

```

```

C *****
C *
C * WITH THE PROBE ORIENTATED AT 180 DEGREE, SEARCH FOR THE SIDE OF THE
C * 2ND SIDE OF THE CALIBRATION OBJECT.
C *
C *****

CALL TOUCH

XXX4 = XPNT1
YYY4 = YPNT1
DX = ABS(XXX4 + 7.303)
DY = ABS(YYY4 - 18.225)
R4 = SQRT(DX**2 + DY**2)

C *****
C *
C * RESET THE CONTROLLER AND RETURN THE ROBOT TO THE HOME POSITION.
C *
C *****

CALL CNTRL(RESET,ERROR)
CALL EXEC(RTRN,ERROR)

C *****
C *
C * COMPUTE THE MAGNITUDE AND ORIENTATION OF THE ECCENTRICITY OF THE TACTILE
C * PROBE.
C *
C *****

Y1 = Y2
X1 = X2
CALL ANGCLC

SIGMA = THETA

Y1 = R2-R1
X1 = R4-R3

CALL ANGCLC

PHI = THETA + 4.71239 + SIGMA
MAG = ABS(SIN(THETA))
MAG = (R2-R1)/(2.0*MAG)
MAG = ABS(MAG)

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

RETURN
END

SUBROUTINE ANGCLC

C *****
C *
C * SUBROUTINE ANGCLC
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-16-86

```

```

C      *
C      * PROGRAM DESCRIPTION
C      *
C      * THIS PROGRAM COMPUTES THE POSTIVE ANGULAR DISPLACEMENT (RELATIVE TO THE
C      * ROBOT'S POSITIVE X AXIS) OF THE VECTOR DEFINED BY X1 AND Y1. THE RESULT
C      * IS IN RADIANs.
C      *
C      * SUBROUTINES USED: NONE
C      *
C      *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDcnt(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DTHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCH,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL      AREA(40),XMOMNT(40),YOMMNT(40),FLDWDH,ARATIO
REAL      AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL      PNTTRK(20,250,2),PERIM(20)
REAL      SDSTAT(20,50),PNSTAT(20,50)
REAL      XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL      A,B,THETA,PHI,MAG,SIGMA

```

```

C
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDcnt,LFETRK
COMMON BDOBJ,YNGROW,DTHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YOMMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

```

IF (X1 .GE. 0 .AND. Y1 .GE. 0) THEN

```

```

    THETA = ATAN(Y1/X1)
    GO TO 10

```

```

ENDIF

```



```
IF (X1 .LT. 0 .AND. Y1 .GE. 0) THEN
```

```
    THETA = ATAN(Y1/X1)
    THETA = 3.1416+THETA
    GO TO 10
```

```
ENDIF
```

```
IF (X1 .LT. 0 .AND. Y1 .LT. 0) THEN
```

```
    THETA = ATAN(Y1/X1)
    THETA = 3.1416+THETA
    GO TO 10
```

```
ENDIF
```

```
IF (X1 .GE. 0 .AND. Y1 .LT. 0) THEN
```

```
    THETA = ATAN(Y1/X1)
    THETA = 6.2832+THETA
```

```
ENDIF
```

```
C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****
```

```
10 RETURN
END
```

```
SUBROUTINE ANGSUB
```

```
C *****
C *
C * SUBROUTINE ANGSUB
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-17-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM COMPUTES THE ANGULAR DIFFERENCE BETWEEN THE TWO ANGLES
C * A AND B. THE RESULT IS IN RADIANS.
C *
C * SUBROUTINES USED: NONE
C *
C *****
```

```
INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETR(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBSCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
```

```
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```
REAL AREA(40),XMOMNT(40),YOMMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA
```

C

```
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROM,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCH,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA
```

```
10 IF (B .GE. A) THEN
```

```
    THETA = B-A
    GO TO 40
```

```
ENDIF
```

```
20 IF (A .GT. 6.283185 ) THEN
```

```
    A = A - 6.283185
    GO TO 20
```

```
ENDIF
```

```
30 IF (B .GT. 6.283185 ) THEN
```

```
    B = B - 6.283185
    GO TO 30
```

```
ENDIF
```

```
THETA = 6.283185-A+B
```

```
C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****
```

```
40 RETURN
```

END

SUBROUTINE TOUCH

```
C *****
C *
C * SUBROUTINE TOUCH
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-12-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS COMMANDS THE 7545 ROBOT TO CONDUCT A TACTILE SEARCH STARTING
C * FROM A SPECIFIED POINT ALONG A SPECIFIED VECTOR BY DOING THE
C * FOLLOWING:
C *
C * 1. DOWN LOADS A STARTING POINT TO THE AML PROGRAM, TTOUCH,
C *    LOCATED IN APPLICATION 2 IN THE 7545 CONTROLLER,
C *
C * 2. DOWN LOADS POINTS ALONG A SPECIFIED VECTOR TO TTOUCH
C *    (DISTANCE BETWEEN POINTS IS 0.05 INCHES),
C *
C * 3. COMMANDS THE ROBOT TO CONDUCT THE SEARCH AND WAITS FOR
C *    CONTACT TO BE MADE,
C *
C * 4. READS IN THE CONTACT POINT (NEW STARTING POINT), AND
C *    COMPUTES A NEW SET OF POINTS ALONG THE SAME VECTOR
C *    (DISTANCE BETWEEN POINTS IS 0.001 INCHES),
C *
C * 5. READS IN THE CONTACT POINT, AND
C *
C * 6. COMPUTES AND ADDS ERROR COMPENSATION VECTOR TO THE CONTACT POINT
C *    IF FLAG1 IS EQUAL TO 1, OTHERWISE IT RETURNS THE CONTACT POINT
C *    AS IS TO THE CALLING PROGRAM.
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C * 1. EXEC
C * 2. WRITEP
C * 3. READP
C * 4. ANGCLC
C * 5. ANGSUB
C *
C *****
```

```
INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROM(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARNT(20,10),BNDCNT(0:479),LFETR(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBcnt,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,8BFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
```

```
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```
REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA
```

C

```
COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDCNT,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA
```

```
Z = 0.0
R = 0.0
```

```
C *****
C *
C * SEND THE COORDINATES OF THE INITIAL STARTING POINTS OF THE PROBE SEARCH
C * TO THE CONTROLLER.
C *
C *****
```

```
VARPNT = 38
CALL WRITEP(VARPNT,XPNT1,YPNT1,Z,THETA,ERROR)
VARPNT = 42
CALL WRITEP(VARPNT,XPNT1,YPNT1,Y3,THETA,ERROR)
```

```
C *****
C *
C * SEND THE POINTS, ALONG THE VECTOR IN WHICH THE PROBE WILL TRAVEL,
C * TO THE CONTROLLER. THE STEP INCREMENTS ARE 0.05 INCHES.
C *
C *****
```

```
VARPNT = 246
```

```
DO 10 MARK7 = 1,50
```

```
XPNT1 = XPNT1 + X1
YPNT1 = YPNT1 + Y1
VARPNT = VARPNT + 4
```

```
CALL WRITEP(VARPNT,XPNT1,YPNT1,Y3,THETA,ERROR)
```

10 CONTINUE

```
C *****
C *
C * START THE TACTILE PROBE SEARCH AND WAIT FOR THE 1ST POINT TO BE TOUCHED;
C * THEN READ IN THE CONTACT POINT.
C *
C *****

CALL EXEC(STRCYC,ERROR)

CALL EXEC(STPMEM,ERROR)

VARPNT = 46

CALL READP(VARPNT,XPNT1,YPNT1,Y3,THETA,ERROR)

X1 = X1/50.0
Y1 = Y1/50.0

C *****
C *
C * RECOMPUTE THE POINTS ALONG THE VECTOR, AND SEND THEM TO THE CONTROLLER.
C * THIS TIME USE STEP INCREMENTS OF 0.001 INCHES.
C *
C *****

VARPNT = 246

DO 20 MARK7 = 1,50

XPNT1 = XPNT1 + X1
YPNT1 = YPNT1 + Y1
VARPNT = VARPNT + 4

CALL WRITEP(VARPNT,XPNT1,YPNT1,Y3,THETA,ERROR)
```

20 CONTINUE

```
C *****
C *
C * CONTINUE THE TACTILE PROBE SEARCH AND WAIT FOR THE PROBE TO MAKE CONTACT
C * WITH THE OBJECT; WHEN IT DOES, READ IN THE CONTACT POINT.
C *
C *****

CALL EXEC(RECLL,ERROR)
CALL EXEC(STRCYC,ERROR)
CALL EXEC(STPCYC,ERROR)

VARPNT = 46

CALL READP(VARPNT,XPNT1,YPNT1,Y3,THETA,ERROR)

C *****
C *
C * TAKE INTO ACCOUNT THE RADIUS OF THE PROBE IN THE LOCATION OF THE
C * CONTACT POINT.
C *
C *****

XPNT1 = XPNT1 + 5.00*X1
YPNT1 = YPNT1 + 5.00*Y1
```

```

C *****
C *
C * IF FLAG1 IS EQUAL TO 1, THEN COMPUTE THE ERROR COMPENSATON FACTOR
C * AND ADD IT TO THE LOCATION OF THE CONTACT POINT.
C *
C *****

IF (FLAG1 .EQ. 1) THEN

CALL ANGCLC
A = PHI
B = THETA

CALL ANGSUB

ERR = MAG*COS(THETA)
ERR = ERR*1000.0
XPNT1 = XPNT1 + ERR*X1
YPNT1 = YPNT1 + ERR*Y1

ENDIF

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

50 RETURN
END

-----
--*
--* PROGRAM: TTOUCH
--*
--* WRITTEN BY: EDWARD C. DE METER
--* DATE: 4-3-86
--*
--* PROGRAM DESCRIPTION
--*
--* THIS PROGRAM DOES THE FOLLOWING:
--*
--* 1. MOVES THE TACTILE PROBE TO A STARTING POSITION,
--*
--* 2. ITERATIVELY MOVES THE TACTILE PROBES THROUGH A GROUP OF
--* POINTS THAT LIE ALONG A VECTOR,
--*
--* 3. CHECKS TO SEE IF CONTACT WITH THE OBJECT HAS BEEN MADE BY
--* TESTING INPUT PORT #10,
--*
--* 4. IF CONTACT IS MADE, MOVES THE TACTILE PROBE BACK TO THE PREVIOUS
--* GROUP POINT, RECORDS THE POINT INTO TCH_PT, AND ENTERS A
--* BREAKPOINT,
--*
--* 5. ITERATIVELY MOVES THE TACTILE PROBE THROUGH A NEW GROUP OF POINTS
--* THAT LIE ALONG THE SAME VECTOR,
--*
--* 6. CHECKS TO SEE IF CONTACT WITH THE OBJECT HAS BEEN MADE BY
--* TESTING INPUT PORT #10,
--*
--* 7. IF CONTACT IS MADE, MODIFIES THE GROUP OF VECTOR POINTS BY
--* INCREASING THE Z VALUE,
--*
--* 8. ITERATIVELY MOVES THE TACTILE PROBE THROUGH THE NEW GROUP OF POINTS,
--*
--* 9. CHECKS TO SEE IF THE PROBE SWEEPS THROUGH THE GROUP OF POINTS BEFORE

```

```

--*      CONTACT WITH THE OBJECT IS MADE, AND
--*
--*      10. IF NO CONTACT IS MADE, THEN THE LAST POINT OF CONTACT IS RECORDED
--*      INTO TCH_PT.
--*
--*****

```

```

UP:      NEW -5.00;                      --DECLARATION FIELD
DOWN:    NEW -8.547;
ST_PT1:  NEW PT(0,0,0,0);
ST_PT2:  NEW PT(0,0,0,0);
TCH_PT:  NEW PT(0,0,0,0);

```

```

PT1:     NEW PT(0,0,0,0);
PT2:     NEW PT(0,0,0,0);
PT3:     NEW PT(0,0,0,0);
PT4:     NEW PT(0,0,0,0);
PT5:     NEW PT(0,0,0,0);
PT6:     NEW PT(0,0,0,0);
PT7:     NEW PT(0,0,0,0);
PT8:     NEW PT(0,0,0,0);
PT9:     NEW PT(0,0,0,0);
PT10:    NEW PT(0,0,0,0);
PT11:    NEW PT(0,0,0,0);
PT12:    NEW PT(0,0,0,0);
PT13:    NEW PT(0,0,0,0);
PT14:    NEW PT(0,0,0,0);
PT15:    NEW PT(0,0,0,0);
PT16:    NEW PT(0,0,0,0);
PT17:    NEW PT(0,0,0,0);
PT18:    NEW PT(0,0,0,0);
PT19:    NEW PT(0,0,0,0);
PT20:    NEW PT(0,0,0,0);
PT21:    NEW PT(0,0,0,0);
PT22:    NEW PT(0,0,0,0);
PT23:    NEW PT(0,0,0,0);
PT24:    NEW PT(0,0,0,0);
PT25:    NEW PT(0,0,0,0);
PT26:    NEW PT(0,0,0,0);
PT27:    NEW PT(0,0,0,0);
PT28:    NEW PT(0,0,0,0);
PT29:    NEW PT(0,0,0,0);
PT30:    NEW PT(0,0,0,0);
PT31:    NEW PT(0,0,0,0);
PT32:    NEW PT(0,0,0,0);
PT33:    NEW PT(0,0,0,0);
PT34:    NEW PT(0,0,0,0);
PT35:    NEW PT(0,0,0,0);
PT36:    NEW PT(0,0,0,0);
PT37:    NEW PT(0,0,0,0);
PT38:    NEW PT(0,0,0,0);
PT39:    NEW PT(0,0,0,0);
PT40:    NEW PT(0,0,0,0);
PT41:    NEW PT(0,0,0,0);
PT42:    NEW PT(0,0,0,0);
PT43:    NEW PT(0,0,0,0);
PT44:    NEW PT(0,0,0,0);
PT45:    NEW PT(0,0,0,0);
PT46:    NEW PT(0,0,0,0);
PT47:    NEW PT(0,0,0,0);
PT48:    NEW PT(0,0,0,0);
PT49:    NEW PT(0,0,0,0);
PT50:    NEW PT(0,0,0,0);
VECTOR:  STATIC GROUP(PT1,PT2,PT3,PT4,PT5,PT6,PT7,PT8,PT9,PT10,PT11,PT12
,PT13,PT14,PT15,PT16,PT17,PT18,PT19,PT20,PT21,PT22

```

,PT23,PT24,PT25,PT26,PT27,PT28,PT29,PT30,PT31,PT32
 ,PT33,PT34,PT35,PT36,PT37,PT38,PT39,PT40,PT41,PT42
 ,PT43,PT44,PT45,PT46,PT47,PT48,PT49,PT50);

POINT1: STATIC COUNTER;
 POINT2: STATIC COUNTER;

MAIN: SUBR;

PMOVE(ST_PT1); --MOVE TO STARTING POINT
 PMOVE(ST_PT2);
 LINEAR(1);
 SETC(POINT1,1);

FIRST_PASS: PMOVE(VECTOR(POINT1)); --MOVE ALONG VECTOR
 DELAY(0.5);
 TESTI(10,1,BACK_UP1); --IF CONTACT IS MADE GO TO
 INCR(POINT1); --BACK_UP1
 COMPC(POINT1 > 50,EXIT);
 BRANCH (FIRST_PASS); --IF NOT, GO TO NEXT GROUP POINT

BACK_UP1: DECR(POINT1);
 PMOVE(VECTOR(POINT1)); --MOVE BACK TO LAST GROUP POINT
 WHERE(TCH_PT); --RECORD VALUE INTO TCH_PT
 BREAKPOINT; --WAIT FOR NEW STARTING POINT
 LINEAR(1); --AND VECTOR POINTS
 SETC(POINT1,1);

SECOND_PASS: PMOVE(VECTOR(POINT1)); --MOVE ALONG VECTOR
 TESTI(10,1,LIFT); --IF CONTACT IS MADE GO TO
 INCR(POINT1); --LIFT
 COMPC(POINT1>50,RECORD);
 BRANCH(SECOND_PASS); --IF NOT, RECORD LAST CONTACT
 --POINT

LIFT: SETC(POINT2,POINT1); --RECORD LAST CONTACT POINT
 CONTINUE: PMOVE(VECTOR(POINT1)); --MOVE TO NEXT POINT
 DPMOVE(<0,0,0.010,0>); --MOVE UP 0.010 INCHES
 WHERE(VECTOR(POINT1)); --RECORD LOCATION INTO GROUP POINT
 INCR(POINT1); --INCREMENT GROUP COUNTER
 COMPC(POINT1>50,RESET); --IF AT THE LAST POINT,BRANCH
 DPMOVE(<0,0,-0.010,0>); --TO RESET
 BRANCH(CONTINUE); --IF NOT, BRANCH TO CONTINUE

RESET: SETC(POINT1,POINT2);
 BRANCH(SECOND_PASS);

RECORD: PMOVE(VECTOR(POINT2)); --MOVE TO LAST CONTACT POINT
 WHERE(TCH_PT); --RECORD LOCATION INTO TCH_PT

EXIT: ZMOVE(UP);
 END;

APPENDIX D. PROBE INSERTION SOFTWARE

SUBROUTINE AINSERT

```
C *****
C *
C * SUBROUTINE AINSERT
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-24-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM COMMANDS THE 7545 ROBOT TO PERFORM AN PROBE INSERTION
C * ROUTINE ON EACH "A" TYPE OBJECT BY DOING THE FOLLOWING:
C *
C *     1. DOWN LOADING THE REGION BOUNDARY POINTS TO THE AML PROGRAM,
C *         AINSERT, WHICH IS LOCATED IN APPLICATION1 IN THE ROBOT
C *         CONTROLLER,
C *
C *     2. COMMANDING THE ROBOT TO BEGIN PROGRAM EXECUTION, AND
C *
C *     3. WAITING FOR PROGRAM EXECUTION TO END, BEFORE DOWN LOADING
C *         A NEW SET OF POINTS.
C *
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C *     1. EXEC
C *     2. WRITEP
C *     3. CNTRL
C *
C *****
```

```
INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STPCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDCNT(0:479),LFETR(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBSCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)
```

```
REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA
```

C

```

COMMON  RESET,STPMEM,RECLL
COMMON  VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON  BNDTRK,SCNROW,ERROR,LINE
COMMON  DTHROW,BUFROM,BTHROW
COMMON  OLDROW,NEWROW,BNDROW
COMMON  OBJROW,PARENT,BNDCNT,LFETRK
COMMON  BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON  TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON  CMLTE,COL,THRS,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON  PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON  MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON  OBJCT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON  LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON  PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON  CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON  ANUM,BNUM,ATRK,BTRK
COMMON  AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON  AAREA,XXMNT,YYMNT,ABND,BBND
COMMON  PNTTRK,PERIM
COMMON  SDSTAT,PNSTAT
COMMON  XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON  A,B,THETA,PHI,MAG,SIGMA

```

```

C *****
C *
C * RETURN THE ROBOT TO HOME, PLACE THE CONTROLLER IN THE AUTO MODE, AND
C * POINT TO APPLICATION 1 (AINSET).
C *
C *****

```

```

CALL EXEC(RTRN,ERROR)
CALL EXEC(AUTO,ERROR)
CALL EXEC(APP1,ERROR)

```

```

Z = 0
R = 0

```

```

C *****
C *
C * FOR EACH "A" TYPE OBJECT, CONDUCT A PROBE INSERTION ROUTINE.
C *
C *****

```

```

DO 20 MARK1 = 1,ANUM

```

```

C *****
C *
C * SEND THE BOUNDARY POINTS TO THE CONTROLLER.
C *
C *****

```

```

VARPNT = 34
MMM = 90

```

```

DO 10 MARK2 = 1,4

```

```

MMM = MMM+4
VARPNT = VARPNT+4
CALL WRITEP(VARPNT,ABND(MARK1,MARK2,1),ABND(MARK1,MARK2,2),Z,R,
2      ERROR)
CALL WRITEP(MMM,ABND(MARK1,MARK2,1),ABND(MARK1,MARK2,2),Z,R,
2      ERROR)

```

```

10 CONTINUE

```

```

C *****
C *
C * START THE INSERTION PROCEDURE AND WAIT FOR ITS COMPLETION.
C *
C *****

CALL EXEC(STRCYC,ERROR)
CALL EXEC(STPCYC,ERROR)

20 CONTINUE

C *****
C *
C * RESET THE CONTROLLER.
C *
C *****

CALL CNTRL(RESET,ERROR)

C *****
C *
C * RETURN TO THE CALLING ROUTINE.
C *
C *****

RETURN
END

SUBROUTINE BINSERT

C *****
C *
C * SUBROUTINE BINSERT
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 3-24-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM COMMANDS THE 7545 ROBOT TO PERFORM AN PROBE INSERTION
C * ROUTINE ON EACH "B" TYPE OBJECT BY DOING THE FOLLOWING:
C *
C * 1. DOWN LOADING THE REGION BOUNDARY POINTS TO THE AML PROGRAM,
C * AINSERT, WHICH IS LOCATED IN APPLICATION1 IN THE ROBOT
C * CONTROLLER,
C *
C * 2. COMMANDING THE ROBOT TO BEGIN PROGRAM EXECUTION, AND
C *
C * 3. WAITING FOR PROGRAM EXECUTION TO END, BEFORE DOWN LOADING
C * A NEW SET OF POINTS.
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C * 1. EXEC
C * 2. WRITEP
C * 3. CNTRL
C *
C *****

```

```

INTEGER*2 RESET,STPMEM,RECLL
INTEGER*2 VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
INTEGER*2 BNDTRK(2,40,0:479),SCNROW(512),ERROR,LINE
INTEGER*2 DTHROW(2,100,2),BUFROW(2,100,2),BTHROW(2,100,2)
INTEGER*2 OLDROW(2,100,2),NEWROW(2,100,2),BNDROW(40,2)
INTEGER OBJROW(40,2),PARENT(20,10),BNDcnt(0:479),LFETRK(40,2)
INTEGER BDOBJ(20),YNGROW(40),DHTYP(40),STACK(20,4),BNDST(40,2)
INTEGER TYPE(40),OBSTAT(40),OFSCNT(20),HOLE,BODY,UNDFD,BDCNT
INTEGER CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
INTEGER PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
INTEGER MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
INTEGER OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
INTEGER LBDTRK(20,10,2),RBDTRK(20,10,2),LTCNT(20),RTCNT(20)
INTEGER PNTNUM(20),LMARK(20),RMARK(20),BNDRTE(40),ROUTE(200,5)
INTEGER CCW,CW,UP,DOWN,CHGROW(10,2),LEFT,RIGHT,RTEPNT
INTEGER ANUM,BNUM,ATRK(4,4),BTRK(4,4)

```

```

REAL AREA(40),XMOMNT(40),YMOMNT(40),FLDWDH,ARATIO
REAL AAREA(40),XXMNT(40),YYMNT(40),ABND(4,4,2),BBND(4,4,2)
REAL PNTTRK(20,250,2),PERIM(20)
REAL SDSTAT(20,50),PNSTAT(20,50)
REAL XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
REAL A,B,THETA,PHI,MAG,SIGMA

```

C

```

COMMON RESET,STPMEM,RECLL
COMMON VARPNT,STRCYC,STPCYC,AUTO,RTRN,APP1,APP2,APP3
COMMON BNDTRK,SCNROW,ERROR,LINE
COMMON DTHROW,BUFROW,BTHROW
COMMON OLDROW,NEWROW,BNDROW
COMMON OBJROW,PARENT,BNDcnt,LFETRK
COMMON BDOBJ,YNGROW,DHTYP,STACK,BNDST
COMMON TYPE,OBSTAT,OFSCNT,HOLE,BODY,UNDFD,BDCNT
COMMON CMLTE,COL,THRSH,OBJCNT,PARCNT,MARK1,MARK2,SCNFLG,PARSPT
COMMON PARPNT,PARCHK,PARNT,DTHFLG,BTHFLG,MARK3,MARK4,MARK5,MARK6
COMMON MARK7,MARK8,MARK9,MARK10,OBCNT,BLKFLG,XXFLG,RX2FLG,LX2FLG
COMMON OBJECT,OBFLG,BBFLG,HLEFLG,HLFLG,FLAG1,FLAG2,FLAG3
COMMON LBDTRK,RBDTRK,LTCNT,RTCNT
COMMON PNTNUM,LMARK,RMARK,BNDRTE,ROUTE
COMMON CCW,CW,UP,DOWN,CHGROW,LEFT,RIGHT,RTEPNT
COMMON ANUM,BNUM,ATRK,BTRK
COMMON AREA,XMOMNT,YMOMNT,FLDWDH,ARATIO
COMMON AAREA,XXMNT,YYMNT,ABND,BBND
COMMON PNTTRK,PERIM
COMMON SDSTAT,PNSTAT
COMMON XPNT1,XPNT2,XPNT3,YPNT1,YPNT2,YPNT3,X1,X2,X3,Y1,Y2,Y3
COMMON A,B,THETA,PHI,MAG,SIGMA

```

C

```

*****

```

C

```

*

```

C

```

* RETURN THE ROBOT TO HOME, PLACE THE CONTROLLER IN THE AUTO MODE, AND

```

C

```

* POINT TO APPLICATION 3 (BINSET).

```

C

```

*

```

C

```

*****

```

```

CALL EXEC(RTRN,ERROR)

```

```

CALL EXEC(AUTO,ERROR)

```

```

CALL EXEC(APP3,ERROR)

```

```

Z = 0

```

```

R = 0

```

C

```

*****

```

C

```

*

```

C

```

* FOR EACH "B" TYPE OBJECT, CONDUCT A PROBE INSERTION ROUTINE.

```

```

C      *
C      *****

DO 20 MARK1 = 1,BNUM

C      *****
C      *
C      * SEND THE BOUNDARY POINTS TO THE CONTROLLER.
C      *
C      *****

VARPNT = 34
MMM = 122

DO 10 MARK2 = 1,4

MMM = MMM+4
VARPNT = VARPNT+4
CALL WRITEP(VARPNT,BBND(MARK1,MARK2,1),BBND(MARK1,MARK2,2),Z,R,
2      ERROR)
CALL WRITEP(MMM,BBND(MARK1,MARK2,1),BBND(MARK1,MARK2,2),Z,R,
2      ERROR)
10 CONTINUE

C      *****
C      *
C      * START THE INSERTION PROCEDURE AND WAIT FOR ITS COMPLETION.
C      *
C      *****

CALL EXEC(STRCYC,ERROR)
CALL EXEC(STPCYC,ERROR)

20 CONTINUE

C      *****
C      *
C      * RESET THE CONTROLLER.
C      *
C      *****

CALL CNTRL(RESET,ERROR)

C      *****
C      *
C      * RETURN TO THE CALLING ROUTINE.
C      *
C      *****

RETURN
END
--*****
--*
--* PROGRAM: AINSERT
--*
--* WRITTEN BY: EDWARD C. DE METER
--* DATE: 4-4-86
--*
--* PROGRAM DESCRIPTION
--*
--* THIS PROGRAM RECEIVES THE POINTS DEFINING THE REGION PC_BOARD AND
--* COMMANDS THE ROBOT TO PERFORM A PROBE INSERTION ROUTINE WITHIN THAT
--* REGION. THE REGION REPRESENTS A TYPE "A" OBJECT.

```

```

--*
--*****

LLPT: NEW PT(0,0,0,0);
ULPT: NEW PT(0,0,0,0);
LRPT: NEW PT(0,0,0,0);
URPT: NEW PT(0,0,0,0);
PT1: NEW PT(1.000,1.000,-7.0,0);
PT2: NEW PT(1.000,1.000,-8.551,0);
PT3: NEW PT(1.000,5.000,-7.0,0);
PT4: NEW PT(1.000,5.000,-8.551,0);
PT5: NEW PT(3.500,5.000,-7.0,0);
PT6: NEW PT(3.500,5.000,-8.551,0);
PT7: NEW PT(3.500,1.000,-7.0,0);
PT8: NEW PT(3.500,1.000,-8.551,0);
PT9: NEW PT(2.250,3.000,-7.0,0);
PT10: NEW PT(2.250,3.000,-8.551,0);
PNTS: NEW <PT1,PT2,PT1,PT3,PT4,PT3,PT5,PT6,PT5,PT7,PT8,PT7,PT9,PT10,PT9>
      ;
PC_BOARD: STATIC REGION(LLPT,ULPT,LRPT,URPT,6.000,6.000,4.500,4.500);
UP: NEW -7.0;
DOWN: NEW -8.551;

MAIN: SUBR;

      INSERT: SUBR(PNT);

            XMOVE(PC_BOARD,PNT);
            DELAY(1.0);
            END;

      PMOVE(LLPT);
      ITERATE ('INSERT',PNTS);
      PMOVE(LLPT);

      END;

--*****--*
--*
--* PROGRAM: BINSERT
--*
--* WRITTEN BY: EDWARD C. DE METER
--* DATE: 4-4-86
--*
--* PROGRAM DESCRIPTION
--*
--* THIS PROGRAM RECEIVES THE POINTS DEFINING THE REGION PC_BOARD AND
--* COMMANDS THE ROBOT TO PERFORM A PROBE INSERTION ROUTINE WITHIN THAT
--* REGION. THE REGION REPRESENTS A TYPE "B" OBJECT.
--*
--*
--*****

      ULPT: NEW PT(0,0,0,0);
      LLPT: NEW PT(0,0,0,0);
      LRPT: NEW PT(0,0,0,0);
      URPT: NEW PT(0,0,0,0);
      PT1: NEW PT(0.500,1.000,-7.0,0);
      PT2: NEW PT(0.500,1.000,-8.551,0);
      PT3: NEW PT(0.500,4.500,-7.0,0);
      PT4: NEW PT(0.500,4.500,-8.551,0);
      PT5: NEW PT(0.500,8.000,-7.0,0);
      PT6: NEW PT(0.500,8.000,-8.551,0);
      PT7: NEW PT(2.250,8.000,-7.0,0);

```

```

PT8: NEW PT(2.250,8.000,-8.551,0);
PT9: NEW PT(2.250,4.500,-7.0,0);
PT10: NEW PT(2.250,4.500,-8.551,0);
PT11: NEW PT(2.250,1.000,-7.0,0);
PT12: NEW PT(2.250,1.000,-8.551,0);
PT13: NEW PT(4.000,1.000,-7.0,0);
PT14: NEW PT(4.000,1.000,-8.551,0);
PT15: NEW PT(4.000,4.500,-7.0,0);
PT16: NEW PT(4.000,4.500,-8.551,0);
PT17: NEW PT(4.000,8.000,-7.0,0);
PT18: NEW PT(4.000,8.000,-8.551,0);
PNTS: NEW <PT1,PT2,PT1,PT3,PT4,PT3,PT5,PT6,PT5,PT7,PT8,PT7,PT9,PT10,PT9,
          PT11,PT12,PT11,PT13,PT14,PT13,PT15,PT16,PT15,PT17,PT18,PT17>;
PC_BOARD: STATIC REGION(LLPT,ULPT,LRPT,URPT,9.000,9.000,4.500,4.500);
UP: NEW -7.0;
DOWN: NEW -8.551;

MAIN: SUBR;

      INSERT: SUBR(PNT);                                --INSERTION SUBROUTINE

              XMOVE(PC_BOARD,PNT);                      --XMOVE TO INSERTION HOLE
              DELAY(1.0);                                --DELAY 1.0 SECONDS
              END;

      PMOVE(LLPT);                                       --MOVE TO LEFTHAND
      ITERATE ('INSERT',PNTS);                          --CORNER OF THE REGION
      PMOVE(LLPT);                                       --INSERT PROBE INTO HOLES

      END;

```

APPENDIX E. USER INFORMATION SOFTWARE

```
SUBROUTINE GRAPH1(LINNUM)

C *****
C *
C * SUBROUTINE GRAPH1
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 4-1-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C * 1. BLANKS THE CRT SCREEN, AND
C *
C * 2. PRINTS A MESSAGE TO THE CRT SCREEN CORRESPONDING TO THE VALUE
C *    OF LINNUM.
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C * 1. CLS
C * 2. LOCATE
C * 3. WRCHAT
C *
C *****

INTEGER LINNUM
INTEGER*2 ATTRIB,PAGE,ROW,COLUMN,COUNT
CHARACTER LINE1*29,LINE2*51,LINE3*51,LINE4*58,LINE5*58,CHAR*1
CHARACTER LINE6*36
DATA ATTRIB/15/PAGE/0/ROW/13/COUNT/1/

LINE1 = 'IMAGE DATA IS BEING PROCESSED'
LINE2 = 'PROBE SEARCH FOR TYPE "A" OBJECT IS BEING CONDUCTED'
LINE3 = 'PROBE SEARCH FOR TYPE "B" OBJECT IS BEING CONDUCTED'
LINE4 = 'INSERTION OPERATION FOR TYPE "A" OBJECT IS BEING CONDUCTE
2D'
LINE5 = 'INSERTION OPERATION FOR TYPE "B" OBJECT IS BEING CONDUCTE
2D'
LINE6 = 'PROBE CALIBRATION IS BEING CONDUCTED'
CALL CLS(ATTRIB)

C *****
C *
C * IF LINNUM EQUALS 1, WRITE MESSAGE #1.
C *
C *****

IF (LINNUM .EQ. 1) THEN

COLUMN = 25
CALL LOCATE (PAGE,ROW,COLUMN)

DO 10 M = 1,29

CHAR = LINE1(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)
```



```

10      CONTINUE

      ENDIF

C      *****
C      *
C      * IF LINNUM EQUALS 2, WRITE MESSAGE #2.
C      *
C      *****

      IF (LINNUM .EQ. 2) THEN

        COLUMN = 15
        CALL LOCATE (PAGE,ROW,COLUMN)

        DO 20 M = 1,51

          CHAR = LINE2(M:M)
          CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

20      CONTINUE

      ENDIF

C      *****
C      *
C      * IF LINNUM EQUALS 3, WRITE MESSAGE #3.
C      *
C      *****

      IF (LINNUM .EQ. 3) THEN

        COLUMN = 15
        CALL LOCATE (PAGE,ROW,COLUMN)

        DO 30 M = 1,51

          CHAR = LINE3(M:M)
          CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

30      CONTINUE

      ENDIF

C      *****
C      *
C      * IF LINNUM EQUALS 4, WRITE MESSAGE #4.
C      *
C      *****

      IF (LINNUM .EQ. 4) THEN

        COLUMN = 11
        CALL LOCATE (PAGE,ROW,COLUMN)

        DO 40 M = 1,58

          CHAR = LINE4(M:M)
          CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

40      CONTINUE

      ENDIF

C      *****
C      *

```

```

C      * IF LINNUM EQUALS 5, WRITE MESSAGE #5.
C      *
C      *****

      IF (LINNUM .EQ. 5) THEN

COLUMN = 11
CALL LOCATE (PAGE,ROW,COLUMN)

DO 50 M = 1,58

CHAR = LINE5(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

50      CONTINUE

      ENDIF

C      *****
C      *
C      * IF LINNUM EQUALS 6, WRITE MESSAGE #6.
C      *
C      *****

      IF (LINNUM .EQ. 6) THEN

COLUMN = 22
CALL LOCATE (PAGE,ROW,COLUMN)

DO 60 M = 1,36

CHAR = LINE6(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

60      CONTINUE

      ENDIF

C      *****
C      *
C      * RETURN TO CALLING ROUTINE
C      *
C      *****

      RETURN
      END

      SUBROUTINE GRAPH2(BDCNT,ANUM,BNUM)

C      *****
C      *
C      * SUBROUTINE GRAPH2
C      *
C      * WRITTEN BY: EDWARD C. DE METER
C      * DATE: 4-1-86
C      *
C      * PROGRAM DESCRIPTION
C      *
C      * THIS PROGRAM DOES THE FOLLOWING:
C      *
C      *      1. BLANKS THE CRT SCREEN,
C      *
C      *      2. PRINTS THE TOTAL NUMBER OF OBJECTS DETECTED BY THE VISION
C      *          SYSTEM TO THE SCREEN,
C      *

```

```

C      *      3. PRINTS THE TOTAL NUMBER OF "A" TYPE OBJECTS DETECTED TO
C      *      THE SCREEN,
C      *
C      *      4. PRINTS THE TOTAL NUMBER OF "B" TYPE OBJECTS DETECTED TO
C      *      THE SCREEN, AND
C      *
C      *      5. WAITS FOR THE USER TO ENTER A KEY BEFORE RETURNING TO THE
C      *      CALLING ROUTINE.
C      *
C      * SUBROUTINES USED
C      *
C      * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C      *
C      *      1. CLS
C      *      2. LOCATE
C      *      3. WRCHAT
C      *      4. INKEY
C      *
C      *****

```

```

INTEGER BDCNT,ANUM,BNUM,KEY,SCNKEY
INTEGER*2 ATTRIB,PAGE,ROW,COLUMN,COUNT
CHARACTER LINE1*26,LINE2*35,LINE3*35,LINE4*35,CHAR*1,CHARR(6)*1
DATA ATTRIB/15/PAGE/0/ROW/11/COUNT/1/COLUMN/22/

```

```

CHARR(1) = '0'
CHARR(2) = '1'
CHARR(3) = '2'
CHARR(4) = '3'
CHARR(5) = '4'
CHARR(6) = '5'
LINE1 = 'TOTAL NUMBER OF OBJECTS = '
LINE2 = 'TOTAL NUMBER OF TYPE "A" OBJECTS = '
LINE3 = 'TOTAL NUMBER OF TYPE "B" OBJECTS = '
LINE4 = 'ENTER ANY KEY TO BEGIN PROBE SEARCH'

```

```

CALL CLS(ATTRIB)

```

```

C      *****
C      *
C      * WRITE THE TOTAL NUMBER OF OBJECT DETECTED TO THE CRT SCREEN.
C      *
C      *****

```

```

CALL LOCATE(PAGE,ROW,COLUMN)

```

```

DO 10 M = 1,26

```

```

CHAR = LINE1(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

```

```

10 CONTINUE

```

```

DO 20 M = 1,6

```

```

IF (BDCNT .EQ. M-1) THEN

```

```

CALL WRCHAT(PAGE,CHARR(M),ATTRIB,COUNT)

```

```

ENDIF

```

```

20 CONTINUE

```

```

C *****
C *
C * WRITE THE TOTAL NUMBER OF "A" TYPE OBJECTS TO THE SCREEN.
C *
C *****

    ROW = 13

    CALL LOCATE(PAGE,ROW,COLUMN)

    DO 30 M = 1,35

    CHAR = LINE2(M:M)
    CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

30 CONTINUE

    DO 40 M = 1,6

    IF (ANUM .EQ. M-1) THEN

CALL WRCHAT(PAGE,CHARR(M),ATTRIB,COUNT)

    ENDIF

40 CONTINUE

C *****
C *
C * WRITE THE TOTAL NUMBER OF "B" TYPE OBJECTS TO THE SCREEN.
C *
C *****

    ROW = 15

    CALL LOCATE(PAGE,ROW,COLUMN)

    DO 50 M = 1,35

    CHAR = LINE3(M:M)
    CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

50 CONTINUE

    DO 60 M = 1,6

    IF (BNUM .EQ. M-1) THEN

CALL WRCHAT(PAGE,CHARR(M),ATTRIB,COUNT)

    ENDIF

60 CONTINUE

C *****
C *
C * WRITE ENTER INSTRUCTION TO USER AND WAIT FOR KEY TO BE ENTERED
C *
C *****

    ROW = 18

```

CALL LOCATE(PAGE,ROW,COLUMN)

DO 70 M = 1,35

CHAR = LINE4(M:M)

CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

70 CONTINUE

CALL INKEY(KEY,SCNKEY)

```
C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****
```

RETURN
END

SUBROUTINE GRAPH3(KEY)

```
C *****
C *
C * SUBROUTINE GRAPH3
C *
C * WRITTEN BY: EDWARD C. DE METER
C * DATE: 4-1-86
C *
C * PROGRAM DESCRIPTION
C *
C * THIS PROGRAM DOES THE FOLLOWING:
C *
C *     1. BLANKS THE CRT SCREEN,
C *
C *     2. PRINTS A MESSAGE TO THE CRT SCREEN INSTRUCTING THE USER TO
C *         ENTER EITHER A 1,2, OR 3, AND
C *
C *     3. WAITS FOR THE USER TO ENTER THE PROPER KEY.
C *
C * SUBROUTINES USED
C *
C * THE FOLLOWING SUBROUTINES ARE USED IN THIS PROGRAM:
C *
C *     1. CLS
C *     2. LOCATE
C *     3. WRCHAT
C *     4. INKEY
C *
C *****
```

INTEGER KEY,SCNKEY

INTEGER*2 ATTRIB,PAGE,ROW,COLUMN,COUNT

CHARACTER LINE1*6,LINE2*36,LINE3*35,LINE4*17,CHAR*1

DATA ATTRIB/15/PAGE/0/ROW/10/COUNT/1/COLUMN/14/

LINE1 = 'ENTER:'

LINE2 = '1. TO REINITIATE INSERTION PROCEDURE'

LINE3 = '2. TO DETECT AND LOCATE NEW OBJECTS'

LINE4 = '3. TO END PROGRAM'

10 COLUMN = 14

ROW = 10

```

CALL CLS(ATTRIB)

C *****
C *
C * WRITE ENTER INSTRUCTION TO USER
C *
C *****

CALL LOCATE(PAGE,ROW,COLUMN)

DO 20 M = 1,6

CHAR = LINE1(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

20 CONTINUE

COLUMN = 22
ROW = 12

CALL LOCATE(PAGE,ROW,COLUMN)

DO 30 M = 1,36

CHAR = LINE2(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

30 CONTINUE

ROW = 14

CALL LOCATE(PAGE,ROW,COLUMN)

DO 40 M = 1,35

CHAR = LINE3(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

40 CONTINUE

ROW = 16

CALL LOCATE(PAGE,ROW,COLUMN)

DO 50 M = 1,17

CHAR = LINE4(M:M)
CALL WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

50 CONTINUE

C *****
C *
C * WAIT FOR KEY TO BE ENTERED
C *
C *****

CALL INKEY(KEY,SCNKEY)

KEY = KEY - 48

```

```

C *****
C *
C * CHECK TO SEE IF THE PROPER KEY HAS BEEN ENTERED
C *
C *****

```

```

IF (KEY .GT. 3 .OR. KEY .LT. 1) THEN

```

```

GO TO 10

```

```

ENDIF

```

```

C *****
C *
C * RETURN TO CALLING ROUTINE
C *
C *****

```

```

RETURN
END

```

APPENDIX F. COMMUNICATION SYSTEM SOFTWARE

```
;  
; This library of assembly language routines can be interfaced with  
; routines written in IBM PROFESSIONAL FORTRAN so that a communication  
; interface can be established between an IBM PC and the IBM 7545 robot  
; controller.  
  
; The routines in this library are written in Intel 8086 assembly language  
; and they access an Intel 8087 coprocessor. This package is assembled  
; using the MICROSOFT MACRO ASSEMBLER.  
  
; The following routines are accessible from a FORTRAN program:  
  
; EXEC - Transmits X commands to the 7545 controller  
;  
; CNTRL - Transmits C commands to the 7545 controller  
;  
; WRITEP- Transmits a point to the 7545 controller  
;  
; WRITER- Transmits a real variable to the 7545 controller  
;  
; READP - Reads a point from the 7545 controller  
;  
; READR - Reads a real variable from the 7545 controller  
;  
; RSTATUS- Reads the reject or machine status from the 7545  
; controller as well as reads the current instruction  
; address  
;  
; RDIDO - Reads the status of anyone of the 16 DI or DO points  
; from the controller  
;  
; PUTWAIT- Waits for a PUT command to be executed by the 7545  
; controller
```

PAGE +
SUBTTL PARAMETER LIST CONTROL BLOCK AND DATA SEGMENT

```
-----  
;  
; PARAMETER LIST CONTROL BLOCK  
;  
-----  
  
PRAM STRUC  
PAR1 DD ? ;FIRST PARAMETER  
PAR2 DD ? ;SECOND PARAMETER  
PAR3 DD ? ;THIRD PARAMETER  
PAR4 DD ? ;FOURTH PARAMETER  
PAR5 DD ? ;FIFTH PARAMETER  
PAR6 DD ? ;SIXTH PARAMETER  
PRAM ENDS  
  
-----  
;  
; DATA SEGMENT FOR ALL ROUTINES  
;  
-----
```


DATAS	SEGMENT	'DATA'	
		DB	'COMMUNICATION'
SP_SAVE		DW	0 ;LOCATION TO SAVE STACK POINTER
		DD	DEC16OUT
		DD	0
PBUFFD		DD	4 DUP(?) ;DOUBLE WORD BUFFER FOR A POINT
PBUFFW		EQU	WORD PTR PBUFFD ;WORD BUFFER FOR A POINT
		DW	8 DUP(?)
TBUFF		DB	16 DUP(?) ;LOCATION OF GENERAL/INPUT BUFFER
TBUFFW		EQU	WORD PTR TBUFF ;LOCATION OF INPUT BUFFER (WORD)
		DW	8 DUP(?)
REAL86		DD	? ;LOCATION TO SAVE REAL # IN 8086
CBUFF		DB	? ;LOCATION OF TRANSMISSION BUFFER
CHARBUFF		DB	48,49,50,51
		DB	52,53,54,55
		DB	56,57,65,66
		DB	67,68,69,70
DBUFF		DB	16 DUP(?) ;LOCATION D-RECORD BUFFER
DDBUFF		EQU	WORD PTR DBUFF
		DW	8 DUP(?)
			;SINGLE PRECISION FORMAT
RBUFF		DW	2 DUP(?) ;LOCATION TO CONSTRUCT REAL # IN
RBUFF1		EQU	BYTE PTR RBUFF ;LSB OF RBUFF
RBUFF2		EQU	RBUFF1+1
RBUFF3		EQU	RBUFF1+2
RBUFF4		EQU	RBUFF1+3 ;MSB OF RBUFF
			;Z80 SINGLE PRECISION FORMAT
REALW1		EQU	WORD PTR REAL86 ;LSW OF REAL86
REALW2		EQU	REALW1+2 ;MSW OF REAL86
REAL1		EQU	BYTE PTR REAL86 ;LSB OF REAL86
REAL2		EQU	REAL1+1
REAL3		EQU	REAL1+2
REAL4		EQU	REAL1+3 ;MSB OF REAL86
EXPINT1		DW	? ;LOCATION OF EXPONENT COUNTER 1
EXPINT2		DW	? ;LOCATION OF EXPONENT COUNTER 2
RGRBGE		DD	? ;LOCATION OF REAL GARBAGE DUMP
IGRBGE		DW	? ;LOCATION OF INTEGER GARBAGE DUMP
BITSET		DW	? ;BITSET INTEGER (WORD)
BIT_TAB		DB	01H,02H,04H ;BIT SET LOOK UP TABLE
		DB	08H,010H,020H
		DB	040H,080H
BIT_TABW		DW	0001H,0002H ;BIT SET LOOK UP TABLE (WORD)
		DW	0004H,0008H
		DW	0010H,0020H
		DW	0040H,0080H
		DW	0100H,0200H
		DW	0400H,0800H
		DW	1000H,2000H
		DW	4000H,8000H
BITBUFF		DB	? ;BITSET BUFFER (BYTE)
ERROR		DW	? ;LOCATION OF ERROR STORAGE
BYTECNT		DW	? ;LOCATION OF BYTE COUNT STORAGE
CHKSUMW		DW	? ;LOCATION OF CHECK SUM WORD
CHKSUM		DB	? ;LOCATION OF CHECK SUM BYTE
VARNUM		DW	? ;LOCATION OF STARTING VAR #
DSET		DB	? ;LOCATION OF EXPECTED RECORD CHECK
ISTAT		DW	? ;LOCATION OF INPUT STATUS WORD
OSTAT		DW	? ;LOCATION OF OUTPUT STATUS WORD
STATWRD		DW	? ;LOCATION OF 8087 STATUS STORAGE
STATBYTE		EQU	BYTE PTR STATWRD+1 ;LOCATION OF MSB OF STATWRD
DATAS	ENDS		

PAGE +
 SUBTTL BEGINNING OF CODE SEGMENT

```

}-----
}
}   CODE SEGEMENT FOR ALL ROUTINES
}
}-----

```

```

CMMNCTN  SEGMENT  'CODE'
          ASSUME   CS:CMMNCTN,DS:DATAS
          DW       SEG DATAS

```

PAGE +
 SUBTTL EXEC

```

}   EXEC performs a remote function related to the operator control
}   panel. It does this by sending the appropriate X commands to the
}   7545 controller.

```

```

}   The calling sequence for EXEC is:

```

```

}       CALL EXEC (CODE,ERROR)

```

```

}   where,

```

```

}       CODE (INTEGER*2) can be one of the following:

```

- } 11 - Return Home
- } 12 - Recall Memory
- } 13 - Reset Error
- } 20 - Auto
- } 22 - Start Cycle
- } 23 - Stop Cycle
- } 24 - Stop and Mem
- } 25 - Step
- } 31 - Application 1
- } 32 - Application 2
- } 33 - Application 3
- } 34 - Application 4
- } 35 - Application 5

```

}   ERROR (INTEGER*2) contains the appropriate error return code:

```

- } 0 - Communication was successfully completed
- } 1 - An EOT was encountered, communication was not successful
- } 2 - A NAK was encountered, communication was not successful

```

}   ROUTINES CALLED: COMINIT,COMOUT,XNFIN,DEC16OUT,IDEND,ACKIN

```

EXEC	PUBLIC	PROC FAR EXEC	
	PUSH	AX	;SAVE REGISTERS
	PUSH	BX	
	PUSH	CX	
	PUSH	DX	
	PUSH	SI	
	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	MOV	SP_SAVE,SP	;SAVE STACK POINTER
	CALL	COMINIT	;INITIALIZE SERIAL PORT
	MOV	AL,88	;MOVE X (ASCII CODE) INTO AL
	CALL	COMOUT	;TRANSMIT X
	CALL	XNFIN	;WAIT FOR XOFF AND THEN XON
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	QDERROR	;IF SO, THEN JUMP TO QDERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	QEERROR	;IF SO, THEN JUMP TO QEERROR
	LDS	SI,ES:PAR1[BX]	;LOAD ADDRESS OF EXEC CODE
	MOV	DX,[SI]	;LOAD EXEC CODE INTO DX
	CALL	DEC16OUT	;TRANSMIT EXEC CODE
	CALL	IDEND	;TRANSMIT CR,LF
WAIT_BRANCH:	MOV	AX,[SI]	;LOAD EXEC CODE INTO AX
	CMP	AX,11	;DETERMINE WHETHER AN ACK OR
	JZ	XNFWAIT	;XOFF,XON IS TO BE RECEIVED
	CMP	AX,24	
	JZ	XNFWAIT	
	CMP	AX,23	
	JZ	XNFWAIT	
ACKWAIT:	CALL	ACKIN	;WAIT FOR ACK FROM CONTROLLER
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	QDERROR	;IF SO, THEN JUMP TO QDERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	QEERROR	;IF SO, THEN JUMP TO QEERROR
	JMP	XEXIT	;JUMP TO XEXIT
XNFWAIT:	CALL	XNFIN	;WAIT FOR XOFF AND XON FROM
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	QDERROR	;IF SO, THEN JUMP TO QDERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	QEERROR	;IF SO, THEN JUMP TO QEERROR
	JMP	XEXIT	;JUMP TO XEXIT
QDERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	XEXIT	;JUMP TO XEXIT
QEERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
			;CONTROLLER
XEXIT:	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX

```

LDS    SI,ES:PAR2[BX]    ;LOAD ADDRESS OF ERROR PARAMETER
MOV     [SI],AX           ;LOAD ERROR MESSAGE INTO ERROR
                                ;PARAMETER
POP     SI               ;RESTORE REGISTERS
POP     DX
POP     CX
POP     BX
POP     AX

RET                                ;RETURN

```

```

EXEC      ENDP

```

```

PAGE +
SUBTTL CNTRL

```

```

;    CNTRL controls execution of an application program by sending the
;    appropriate C commands to the 7545 controller.

;    The calling sequence for CNTRL is:

;        CALL CNTRL (CODE, ERROR)

;    where,

;        CODE (INTEGER*2) could be one of the following:

;            1 - Suspend program execution
;            2 - Restart program execution
;            4 - Execute until next terminator encountered
;            20 - Reset Controller

;        ERROR (INTEGER*2) is the same as described in EXEC.

;    ROUTINES CALLED: COMINIT,COUT,XNFIN,DEC16OUT,IDEND,CHAROUT,ACKIN

```

```

CNTRL

```

```

        PROC FAR
PUBLIC  CNTRL

        PUSH    AX           ;SAVE REGISTERS
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI

        MOV     AX,DATAS     ;POINT TO DATA SEGMENT
        MOV     DS,AX
        MOV     SP_SAVE,SP   ;SAVE STACK POINTER

        CALL    COMINIT      ;INITIALIZE THE SERIAL PORT

        CALL    COUT         ;TRANSMIT C TO THE CONTROLLER
        CALL    XNFIN        ;WAIT FOR XOFF AND XON

        MOV     AX,ERROR     ;MOVE CONTENTS OF ERROR INTO AX
        CMP     AX,04H       ;IS ERROR = EOT
        JZ      RRERROR      ;IF SO, THEN JUMP TO RRERROR
        CMP     AX,015H      ;IS ERROR = NAK
        JZ      SSERROR      ;IF SO, THEN JUMP TO SSERROR

```

	LDS	SI,ES:PAR1[BX]	;LOAD ADDRESS OF CODE
	MOV	AX,[SI]	;LOAD CODE INTO DX
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	CMP	AX,20	;DOES CODE = 20
	JNZ	CODE_OUT	;IF NOT , JUMP TO CODE_OUT
	MOV	DX,AX	;MOVE CODE INTO DX
	CALL	DEC16OUT	;TRANSMIT CODE TO CONTROLLER
	CALL	IDEND	;TRANSMIT CR,LF TO CONTROLLER
	JMP	ACK_INN	;JUMP TO ACK_INN
CODE_OUT:	MOV	CBUFF,AL	;LOAD CODE INTO CBUFF
	CALL	CHAROUT	;TRANSMIT CODE TO CONTROLLER
	CALL	IDEND	;TRANSMIT CR,LF TO CONTROLLER
	CMP	AX,04H	;IS CODE = 4
	JZ	TERMINATE	;IF SO, JUMP TO TERMINATE
ACK_INN:	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	RRERROR	;IF SO, THEN JUMP TO RRERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	SSERROR	;IF SO, THEN JUMP TO SSERROR
	JMP	FEXIT	;JUMP TO FEXIT
TERMINATE:	CALL	XNFIN	;WAIT FOR XOFF AND XON FROM
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	RRERROR	;IF SO, THEN JUMP TO RRERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	SSERROR	;IF SO, THEN JUMP TO SSERROR
	JMP	FEXIT	;JUMP TO FEXIT
RRERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	FEXIT	;JUMP TO FEXIT
SSERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
			;CONTROLLER
FEXIT:	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	LDS	SI,ES:PAR2[BX]	;LOAD ADDRESS OF ERROR PARAMETER
	MOV	[SI],AX	;LOAD ERROR MESSAGE INTO ERROR
	POP	SI	;RESTORE REGISTERS
	POP	DX	
	POP	CX	
	POP	BX	
	POP	AX	
			;PARAMETER
	RET		;RETURN
CNTRL	ENDP		

PAGE +
SUBTTL WRITEP

; WRITEP writes the X, Y, Z, R values of a point into four successive
; controller memory locations by sending the appropriate C command to
; the controller, followed by the necessary D records.

; The calling sequence for WRITEP is:

; CALL WRITEP (VARSTRT, X, Y, Z, R, ERROR)

; where:

; VARSTRT (INTEGER*2) is the starting variable number,

; X (REAL) is the X coordinate of the point,

; Y (REAL) is the Y coordinate of the point,

; Z (REAL) is the Z coordinate of the point,

; R (REAL) is the R coordinate of the point, and

; ERROR (INTEGER*2) is the same as described in EXEC.

; ROUTINES CALLED: COMINIT, COUT, XNFIN, DEC16OUT, IDEND, ACKIN,
; DRECOUT, CNVRT86Z80, EGOUT

WRITEP	PROC	FAR	
	PUBLIC	WRITEP	
	PUSH	AX	;SAVE REGISTERS
	PUSH	BX	
	PUSH	CX	
	PUSH	DX	
	PUSH	SI	
	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	CALL	COMINIT	;INITIALIZE THE SERIAL PORT
	LDS	SI,ES:PAR1[BX]	;LOAD ADDRESS OF VAR # IN DS
	MOV	AX,[SI]	;MOVE VARNUM INTO AX
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	MOV	VARNUM,AX	;MOV VAR # INTO VARNUM
	LDS	SI,ES:PAR2[BX]	;LOAD ADDRESS OF X COORDINATE
	MOV	AX,[SI]	;LOAD LSW INTO AX
	MOV	DX,[SI+2]	;LOAD MSW INTO DX
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	MOV	DI,14	;LOAD OFFSET
	MOV	PBUFFW[DI],DX	;LOAD MSW OF X CORD. INTO BUFFER
	MOV	PBUFFW[DI-2],AX	;LOAD LSW OF X CORD. INTO BUFFER
	LDS	SI,ES:PAR3[BX]	;LOAD ADDRESS OF Y COORDINATE
	MOV	AX,[SI]	;LOAD LSW INTO AX
	MOV	DX,[SI+2]	;LOAD MSW INTO DX

MOV	CX,DATAS	;POINT TO DATA SEGMENT
MOV	DS,CX	
MOV	DI,10	;LOAD OFFSET
MOV	PBUFFW[DI],DX	;LOAD MSW OF Y CORD. INTO BUFFER
MOV	PBUFFW[DI-2],AX	;LOAD LSW OF Y CORD. INTO BUFFER
LDS	SI,ES:PAR4[BX]	;LOAD ADDRESS OF Z COORDINATE
MOV	AX,[SI]	;LOAD LSW INTO AX
MOV	DX,[SI+2]	;LOAD MSW INTO DX
MOV	CX,DATAS	;POINT TO DATA SEGMENT
MOV	DS,CX	
MOV	DI,6	;LOAD OFFSET
MOV	PBUFFW[DI],DX	;LOAD MSW OF Z CORD. INTO BUFFER
MOV	PBUFFW[DI-2],AX	;LOAD LSW OF Z CORD. INTO BUFFER
LDS	SI,ES:PAR5[BX]	;LOAD ADDRESS OF R COORDINATE
MOV	AX,[SI]	;LOAD LSW INTO AX
MOV	DX,[SI+2]	;LOAD MSW INTO DX
MOV	CX,DATAS	;POINT TO DATA SEGMENT
MOV	DS,CX	
MOV	DI,2	;LOAD OFFSET
MOV	PBUFFW[DI],DX	;LOAD MSW OF R CORD. INTO BUFFER
MOV	PBUFFW[DI-2],AX	;LOAD LSW OF R CORD. INTO BUFFER
CALL	COUT	;TRANSMIT C TO THE CONTROLLER
CALL	XNFIN	;WAIT FOR XON AND XOFF
MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
CMP	AX,04H	;IS ERROR = EOT
JZ	DERROR	;IF SO, THEN JUMP TO DERROR
CMP	AX,015H	;IS ERROR = NAK
JZ	EERROR	;IF SO, THEN JUMP TO EERROR
MOV	DX,80	;MOVE 80 INTO DX
CALL	DEC16OUT	;TRANSMIT 80 TO THE CONTROLLER
CALL	IDEND	;TRANSMIT CR, LF TO CONTROLLER
CALL	ACKIN	;WAIT FOR ACK
MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
CMP	AX,04H	;IS ERROR = EOT
JZ	DERROR	;IF SO, THEN JUMP TO DERROR
CMP	AX,015H	;IS ERROR = NAK
JZ	EERROR	;IF SO, THEN JUMP TO EERROR
MOV	AX,4	;MOVE 4 INTO AX
MOV	BYTECNT,AX	;LOAD BYTECNT WITH 4
MOV	SI,2	;MOVE 2 INTO OFFSET
MOV	AX,VARNUM	;MOVE CONTENTS OF VARNUM INTO AX
MOV	DBUFF[SI],AX	;MOVE VAR # INTO DBUFF
MOV	AX,4	;MOVE 4 INTO AX
MOV	DBUFF,AX	;MOVE # OF VAR. INTO DBUFF
CALL	DRECOUT	;SEND D RECORD
CALL	ACKIN	;WAIT FOR ACK
MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
CMP	AX,04H	;IS ERROR = EOT
JZ	DERROR	;IF SO, THEN JUMP TO DERROR

	CMP	AX,015H	;IS ERROR = NAK
	JZ	EERROR	;IF SO, THEN JUMP TO EERROR
	JMP	POINT_START	;JUMP TO POINT START
DERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	YEXIT	;JUMP TO YEXIT
EERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
	JMP	YEXIT	;JUMP TO YEXIT
POINT_START:	MOV	SI,2	;LOAD REAL86 OFFSET WITH 2
	MOV	DI,16	;LOAD BUFFER OFFSET WITH 16
POINT_LOOP:	DEC	DI	;DECREMENT BUFFER OFFSET
	DEC	DI	;DECREMENT BUFFER OFFSET
	MOV	AX,PBUFFW[DI]	;LOAD MSW INTO AX
	MOV	REALN2,AX	;LOAD MSW INTO REAL86
	MOV	AX,PBUFFW[DI-2]	;LOAD LSW INTO AX
	MOV	REALW1,AX	;LOAD LSW INTO REAL86
	CALL	CNVRT86Z80	;CONVERT REAL # TO Z80 FORMAT
	MOV	AX,RBUFF[SI]	;MOVE MSW OF REAL # INTO AX
	MOV	DBUFF[SI],AX	;MOVE MSW OF REAL # INTO DBUFF
	MOV	AX,RBUFF	;MOVE LSW OF REAL # INTO AX
	MOV	DBUFF,AX	;MOVE LSW OF REAL # INTO DBUFF
	MOV	AX,4	;MOVE 4 INTO AX
	MOV	BYTECNT,AX	;SET BYTECNT = 4
	CALL	DRECOUT	;SEND D RECORD
	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	FERROR	;IF SO, THEN JUMP TO FERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	GERROR	;IF SO ,THEN JUMP TO GERROR
	CMP	DI,2	;IS OFFSET = 2
	JZ	POINT_OUT	;IF SO, JUMP TO POINT_OUT
	DEC	DI	;DECREMENT BUFFER OFFSET
	DEC	DI	;DECREMENT BUFFER OFFSET
	JMP	POINT_LOOP	;JUMP TO POINT_LOOP
POINT_OUT:	CALL	EGOUT	;TRANSMIT EG TO CONTROLLER
	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	FERROR	;IF SO, THEN JUMP TO FERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	GERROR	;IF SO ,THEN JUMP TO GERROR
	JMP	YEXIT	;JUMP TO YEXIT
FERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	YEXIT	;JUMP TO YEXIT
GERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
YEXIT:	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	LDS	SI,ES:PAR6[BX]	;LOAD ADDRESS OF PAR3 INTO DS


```

MOV      [SI],AX          ;LOAD ERROR MESSAGE INTO PAR3

POP      SI              ;RESTORE REGISTERS
POP      DX
POP      CX
POP      BX
POP      AX

RET                      ;RETURN

WRITEP      ENDP

```

PAGE +
SUBTTL WRITER

```

;    WRITER writes a real value into the controller memory by sending the
;    appropriate C command, followed by the necessary D records.

;    The calling sequence for WRITER is:

;    CALL WRITER (VARSTRT, VAL, ERROR)

;    where:

;    VARSTRT (INTEGER*2) is the starting variable number,
;    VAL (REAL) is the value to be sent to the controller, and
;    ERROR is the same as described in EXEC.

;    ROUTINES CALLED: COMINIT, COUT, XNFIN, DEC16OUT, IDEND, ACKIN,
;    DRECOUT, CNVRT86Z80, EGOUT

```

```

WRITER      PROC      FAR
PUBLIC      WRITER

PUSH        AX          ;SAVE REGISTERS
PUSH        BX
PUSH        CX
PUSH        DX
PUSH        SI

MOV         AX,DATAS     ;POINT TO DATA SEGMENT
MOV         DS,AX

CALL        COMINIT     ;INITIALIZE THE SERIAL PORT

LDS         SI,ES:PAR1[BX] ;LOAD ADDRESS OF VAR # IN DS
MOV         AX,[SI]      ;MOVE VARNUM INTO AX
MOV         CX,DATAS     ;POINT TO DATA SEGMENT
MOV         DS,CX
MOV         VARNUM,AX    ;MOV VAR # INTO VARNUM

LDS         SI,ES:PAR2[BX] ;LOAD ADDRESS OF REAL # IN DS
MOV         AX,[SI]      ;LOAD LSW OF REAL # IN AX
MOV         DX,[SI+2]    ;LOAD MSW OF REAL # IN DX

MOV         CX,DATAS     ;POINT TO DATA SEGMENT
MOV         DS,CX

```

	MOV	REALW1,AX	;MOVE LSW OF REAL # INTO REAL86
	MOV	REALW2,DX	;MOVE MSW OF REAL # INTO REAL86
	CALL	COUT	;TRANSMIT C TO THE CONTROLLER
	CALL	XNFIN	;WAIT FOR XON AND XOFF
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	HERROR	;IF SO, THEN JUMP TO HERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	IERROR	;IF SO, THEN JUMP TO IERROR
	MOV	DX,80	;MOVE 80 INTO DX
	CALL	DEC16OUT	;TRANSMIT 80 TO THE CONTROLLER
	CALL	IDEND	;TRANSMIT CR, LF TO CONTROLLER
	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	HERROR	;IF SO, THEN JUMP TO HERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	IERROR	;IF SO, THEN JUMP TO IERROR
	MOV	AX,4	;MOVE 4 INTO AX
	MOV	BYTECNT,AX	;LOAD BYTECNT WITH 4
	MOV	SI,2	;MOVE 2 INTO OFFSET
	MOV	AX,VARNUM	;MOVE CONTENTS OF VARNUM INTO AX
	MOV	DBUFF[SI],AX	;MOVE VAR # INTO DBUFF
	MOV	AX,1	;MOVE 1 INTO AX
	MOV	DBUFF,AX	;MOVE # OF VAR. INTO DBUFF
	CALL	DRECOUT	;SEND D RECORD
	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	HERROR	;IF SO, THEN JUMP TO HERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	IERROR	;IF SO, THEN JUMP TO IERROR
	JMP	REAL_OUT	;JUMP TO REAL_OUT
HERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	ZEXIT	;JUMP TO ZEXIT
IERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
	JMP	ZEXIT	;JUMP TO ZEXIT
REAL_OUT:.	CALL	CNVRT86Z80	;CONVERT REAL # TO Z80 FORMAT
	MOV	SI,2	;MOVE OFFSET INTO SI
	MOV	AX,RBUFF[SI]	;MOVE MSW OF REAL # INTO AX
	MOV	DBUFF[SI],AX	;MOVE MSW OF REAL # INTO DBUFF
	MOV	AX,RBUFF	;MOVE LSW OF REAL # INTO AX
	MOV	DBUFF,AX	;MOVE LSW OF REAL # INTO DBUFF
	CALL	DRECOUT	;SEND D RECORD
	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	BERROR	;IF SO, THEN JUMP TO BERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	CERROR	;IF SO ,THEN JUMP TO CERROR

	CALL	EGOUT	;TRANSMIT EG TO CONTROLLER
	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	BERROR	;IF SO, THEN JUMP TO BERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	CERROR	;IF SO, THEN JUMP TO CERROR
	JMP	ZEXIT	;JUMP TO ZEXIT
BERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	ZEXIT	;JUMP TO ZEXIT
CERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
ZEXIT:	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	LDS	SI,ES:PAR3[BX]	;LOAD ADDRESS OF PAR3 INTO DS
	MOV	[SI],AX	;LOAD ERROR MESSAGE INTO PAR3
	POP	SI	;RESTORE REGISTERS
	POP	DX	
	POP	CX	
	POP	BX	
	POP	AX	
	RET		;RETURN
WRITER	ENDP		

PAGE +
SUBTTL READP

; READP reads the X, Y, Z, R values of a point from four consecutive
; memory locations in the controller by sending the appropriate R command,
; followed by the transmission and reception of the necessary D records.

; The calling sequence for READP is :

; CALL READP (VARSTRT, X, Y, Z, R, ERROR)

; Where:

; VARSTRT (INTEGER*2) is the starting variable number,

; X (REAL) is the X coordinate of the point,

; Y (REAL) is the Y coordinate of the point,

; Z (REAL) is the Z coordinate of the point,

; R (REAL) is the R coordinate of the point, and

; ERROR (INTEGER*2) is the same as described in EXEC.

; ROUTINES CALLED: COMINIT,ROUT,XNFIN,DEC16OUT,IDEND,ACKIN,
; DRECOUT,DRECIN,CNVRTZ8086

READP	PROC	FAR
	PUBLIC	READP

	PUSH	AX	;SAVE REGISTERS
	PUSH	BX	
	PUSH	CX	
	PUSH	DX	
	PUSH	SI	
	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	MOV	SP_SAVE,SP	;SAVE STACK POINTER
	CALL	COMINIT	;INITIALIZE SERIAL PORT
	MOV	AH,0	;MOVE 0 INTO AH
	MOV	DSET,AH	;CLEAR DSET
	LDS	SI,ES:PAR1[BX]	;LOAD ADDRESS OF VAR # IN DS
	MOV	AX,[SI]	;MOVE VARNUM INTO AX
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	MOV	VARNUM,AX	;MOV VAR # INTO VARNUM
	CALL	ROUT	;TRANSMIT R TO THE CONTROLLER
	CALL	XNFIN	;WAIT FOR XON AND XOFF
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	LERROR	;IF SO, THEN JUMP TO LERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	MERROR	;IF SO, THEN JUMP TO MERROR
	MOV	DX,80	;MOVE 80 INTO DX
	CALL	DEC16OUT	;TRANSMIT 80 TO THE CONTROLLER
	CALL	IDEND	;TRANSMIT CR, LF TO CONTROLLER
	CALL	ACKIN	;WAIT FOR ACK
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	LERROR	;IF SO, THEN JUMP TO LERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	MERROR	;IF SO, THEN JUMP TO MERROR
	MOV	AX,4	;MOVE 4 INTO AX
	MOV	BYTECNT,AX	;LOAD BYTECNT WITH 4
	MOV	SI,2	;MOVE 2 INTO OFFSET
	MOV	AX,VARNUM	;MOVE CONTENTS OF VARNUM INTO AX
	MOV	DDBUFF[SI],AX	;MOVE VAR # INTO DBUFF
	MOV	AX,4	;MOVE 1 INTO AX
	MOV	DDBUFF,AX	;MOVE # OF VAR. INTO DBUFF
	CALL	DRECOUT	;SEND D RECORD
	JMP	D_REC_IN	;JUMP TO D_REC_IN
LERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	WEXIT	;JUMP TO WEXIT
MERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
	JMP	WEXIT	;JUMP TO WEXIT
D_REC_IN:	CALL	DRECIN	;READ IN D RECORD
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,015H	;IS ERROR = NAK
	JZ	MERROR	;IF SO, THEN JUMP TO MERROR

POINT_OUT_LOOP:	MOV	DI,16	;LOAD DBUFF OFFSET WITH 16
	DEC	DI	;DECREMENT OFFSET
	DEC	DI	;DECREMENT OFFSET
	MOV	SI,2	;LOAD 2 INTO RBUFF OFFSET
	MOV	AX,DBUFF[DI]	;MOVE MSW INTO AX
	MOV	RBUFF[SI],AX	;MOVE MSW INTO RBUFF
	MOV	AX,DBUFF[DI-2]	;MOVE LSW INTO AX
	MOV	RBUFF,AX	;MOVE LSW INTO RBUFF
	CALL	CNVRTZ8086	;CONVERT REAL NUMBER INTO 8086 ;SINGLE PRECISION FORMAT
	MOV	AX,REALW1	;MOVE LSW INTO AX
	MOV	DX,REALW2	;MOVE MSW INTO DX
	DATA_OUT:	MOV	PBUFFW[DI],DX
MOV		PBUFFW[DI-2],AX	;MOVE LSW INTO PBUFFW
CMP		DI,2	;DOES DI POINT TO LAST REAL #
JZ		DATA_OUT	;IF SO, JUMP TO DATA OUT
DEC		DI	;DECREMENT OFFSET
DEC		DI	;DECREMENT OFFSET
JMP		POINT_OUT_LOOP	;CONVERT ANOTHER REAL #
MOV		DI,14	;LOAD OFFSET WITH 14
MOV		DX,PBUFFW[DI]	;LOAD MSW OF REAL # INTO DX
MOV		AX,PBUFFW[DI-2]	;LOAD LSW OF REAL # INTO AX
LDS		SI,ES:PAR2[BX]	;LOAD ADDRESS OF PAR2 INTO DS
MOV		[SI+2],DX	;LOAD MSW INTO PAR2
MOV	[SI],AX	;LOAD LSW INTO PAR2	
MOV	CX,DATAS	;POINT TO DATA SEGMENT	
MOV	DS,CX		
MOV	DI,10	;LOAD OFFSET WITH 10	
MOV	DX,PBUFFW[DI]	;LOAD MSW OF REAL # INTO DX	
MOV	AX,PBUFFW[DI-2]	;LOAD LSW OF REAL # INTO AX	
LDS	SI,ES:PAR3[BX]	;LOAD ADDRESS OF PAR3 INTO DS	
MOV	[SI+2],DX	;LOAD MSW INTO PAR3	
MOV	[SI],AX	;LOAD LSW INTO PAR3	
MOV	CX,DATAS	;POINT TO DATA SEGMENT	
MOV	DS,CX		
MOV	DI,6	;LOAD OFFSET WITH 6	
MOV	DX,PBUFFW[DI]	;LOAD MSW OF REAL # INTO DX	
MOV	AX,PBUFFW[DI-2]	;LOAD LSW OF REAL # INTO AX	
LDS	SI,ES:PAR4[BX]	;LOAD ADDRESS OF PAR4 INTO DS	
MOV	[SI+2],DX	;LOAD MSW INTO PAR4	
MOV	[SI],AX	;LOAD LSW INTO PAR4	
MOV	CX,DATAS	;POINT TO DATA SEGMENT	
MOV	DS,CX		
MOV	DI,2	;LOAD OFFSET WITH 2	
MOV	DX,PBUFFW[DI]	;LOAD MSW OF REAL # INTO DX	
MOV	AX,PBUFFW[DI-2]	;LOAD LSW OF REAL # INTO AX	

```

        LDS     SI,ES:PAR5[BX]      ;LOAD ADDRESS OF PAR5 INTO DS
        MOV     [SI+2],DX           ;LOAD MSW INTO PAR5
        MOV     [SI],AX             ;LOAD LSW INTO PAR5

        MOV     CX,DATAS            ;POINT TO DATA SEGMENT
        MOV     DS,CX

        JMP     WEXIT               ;JUMP TO REXIT

WEXIT:   MOV     AX,ERROR            ;MOVE CONTENTS OF ERROR INTO AX
        LDS     SI,ES:PAR6[BX]      ;LOAD ADDRESS OF PAR3 INTO DS
        MOV     [SI],AX            ;LOAD ERROR MESSAGE INTO PAR3

        POP     SI                  ;RESTORE REGISTERS
        POP     DX
        POP     CX
        POP     BX
        POP     AX

        RET                         ;RETURN

READP    ENDP

```

PAGE +
SUBTTL READR

```

;   READR reads a real value from the controller by sending the appropriate
;   R command, followed by the transmission and reception of the necessary
;   D records.

;   The calling sequence for READR is:

;       CALL READR (VARSTRT, VAL, ERROR)

;   Where:

;       VARSTRT (INTEGER*2) is the starting variable number,
;       VAL (REAL) is the value to be sent to the controller, and
;       ERROR is the same as described in EXEC.

;   ROUTINES CALLED: COMINIT,ROUT,XNFIN,DEC16OUT,IDEND,ACKIN,
;                   DRECOUT,DRECIN,CNVRTZ8086

```

```

READR    PROC    FAR
        PUBLIC  READR

        PUSH    AX                  ;SAVE REGISTERS
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI

        MOV     AX,DATAS            ;POINT TO DATA SEGMENT
        MOV     DS,AX
        MOV     SP_SAVE,SP         ;SAVE STACK POINTER

        CALL    COMINIT            ;INITIALIZE SERIAL PORT

```

MOV	AH,0	;MOVE 0 INTO AH
MOV	DSET,AH	;CLEAR DSET
LDS	SI,ES:PAR1[BX]	;LOAD ADDRESS OF VAR # IN DS
MOV	AX,[SI]	;MOVE VARNUM INTO AX
MOV	CX,DATAS	;POINT TO DATA SEGMENT
MOV	DS,CX	
MOV	VARNUM,AX	;MOV VAR # INTO VARNUM
CALL	ROUT	;TRANSMIT R TO THE CONTROLLER
CALL	XNFIN	;WAIT FOR XON AND XOFF
MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
CMP	AX,04H	;IS ERROR = EOT
JZ	JERROR	;IF SO, THEN JUMP TO HERROR
CMP	AX,015H	;IS ERROR = NAK
JZ	KERROR	;IF SO, THEN JUMP TO IERROR
MOV	DX,80	;MOVE 80 INTO DX
CALL	DEC16OUT	;TRANSMIT 80 TO THE CONTROLLER
CALL	IDEND	;TRANSMIT CR, LF TO CONTROLLER
CALL	ACKIN	;WAIT FOR ACK
MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
CMP	AX,04H	;IS ERROR = EOT
JZ	JERROR	;IF SO, THEN JUMP TO JERROR
CMP	AX,015H	;IS ERROR = NAK
JZ	KERROR	;IF SO, THEN JUMP TO KERROR
MOV	AX,4	;MOVE 4 INTO AX
MOV	BYTECNT,AX	;LOAD BYTECNT WITH 4
MOV	SI,2	;MOVE 2 INTO OFFSET
MOV	AX,VARNUM	;MOVE CONTENTS OF VARNUM INTO AX
MOV	DDBUFF[SI],AX	;MOVE VAR # INTO DBUFF
MOV	AX,1	;MOVE 1 INTO AX
MOV	DDBUFF,AX	;MOVE # OF VAR. INTO DBUFF
CALL	DRECOUT	;SEND D RECORD
CALL	DRECIN	;READ IN D RECORD
MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
CMP	AX,015H	;IS ERROR = NAK
JZ	KERROR	;IF SO, THEN JUMP TO KERROR
MOV	SI,2	;LOAD 2 INTO RBUFF OFFSET
MOV	AX,DDBUFF[SI]	;MOVE MSW INTO AX
MOV	RBUFF[SI],AX	;MOVE MSW INTO RBUFF
MOV	AX,DDBUFF	;MOVE LSW INTO AX
MOV	RBUFF,AX	;MOVE LSW INTO RBUFF
CALL	CNVRTZ8086	;CONVERT REAL NUMBER INTO 8086 ;SINGLE PRECISION FORMAT
MOV	AX,REALW1	;MOVE LSW INTO AX
MOV	DX,REALW2	;MOVE MSW INTO DX
LDS	SI,ES:PAR2[BX]	;LOAD ADDRESS OF PAR2 INTO DS
MOV	[SI],AX	;MOVE LSW INTO PAR2
MOV	[SI+2],DX	;MOVE MSW INTO PAR2
MOV	AX,DATAS	;POINT TO DATA SEGMENT
MOV	DS,AX	

```

                JMP      REXIT          ;JUMP TO REXIT

JERROR:         MOV      AX,1          ;LOAD 1 INTO AX
                MOV      ERROR,AX      ;SET ERROR = 1
                JMP      REXIT         ;JUMP TO REXIT

KERROR:         MOV      AX,2          ;LOAD 2 INTO AX
                MOV      ERROR,AX      ;SET ERROR = 2

REXIT:          MOV      AX,ERROR      ;MOVE CONTENTS OF ERROR INTO AX
                LDS      SI,ES:PAR3[BX] ;LOAD ADDRESS OF PAR3 INTO DS
                MOV      [SI],AX       ;LOAD ERROR MESSAGE INTO PAR3

                POP      SI            ;RESTORE REGISTERS
                POP      DX
                POP      CX
                POP      BX
                POP      AX

                RET                    ;RETURN

READR           ENDP

```

PAGE +
SUBTTL RSTATUS

```

;   RSTATUS reads the machine status, reject status, or current instruction
;   address from the controller by sending the appropriate R command followed
;   by the reception of the necessary D record.

```

```

;   The calling sequence for RSTATUS is:

```

```

;       CALL RSTATUS (CODE,VAL1,VAL2,ERROR)

```

```

;   Where,

```

```

;       CODE (INTEGER*2) could be one of the following:

```

```

;           1 - Read Machine Status
;           2 - Read Reject Status
;           8 - Read Current Instruction Address

```

```

;       VAL1 (INTEGER*2) is a return value containing the following if
;       CODE is 01:

```

```

;           128 - Servo Failure
;           64 - Power Failure
;           32 - Over Run
;           16 - Over Time
;           8 - Transmission Error
;           6 - AML/E Error
;           4 - Data Error

```


} If VAL1 contains 6, then VAL2 (INTEGER*2) will contain the additional
information:

- } 10 - Part Number Too Small For Pallet
- } 11 - Part Number Too Large For Pallet
- } 20 - Invalid Index For A Group
- } 30 - Communications Not Established

} If VAL1 contains 4, then VAL2 (INTEGER*2) will contain the additional
information:

- } 0 - No Data Error Present
- } 1 - Bus Error
- } 2 - Memory Test Error
- } 17 - Arithmetic Error
- } 18 - Programming Error
- } 19 - Invalid Op Code
- } 20 - Invalid Data
- } 21 - Invalid Port Number
- } 23 - Stack Error
- } 24 - Address Error
- } 64 - Point Out Of Workspace

} VAL1 (INTEGER*2) will contain the following if CODE is 2:

- } 16 - Record Format Error
- } 21 - Invalid Port Number
- } 32 - Undefined Record
- } 48 - Unacceptable Condition (Improper application startup sequence)
- } 49 - Unacceptable Condition (C record, but no application)
- } 64 - Point Out Of Workspace
- } 80 - Insufficient Memory
- } 81 - Invalid Robot Type
- } 82 - Double Select Error
- } 83 - Invalid Application Number
- } 84 - C Record Format Error
- } 96 - Invalid Identifier Sent Before N Record
- } 112 - XOFF Time Out (30 Seconds)

```

;      116 - Invalid Data
;
;      117 - Too Much Data
;
;      118 - Too Little Data
;
;      128 - Manipulator Power Off
;
;      160 - XOFF Time-Out State
;
;      161 - Can Not Accept Command (XON State)
;
;      162 - Communication Operation Not Allowed (Xto or Mxo mode)
;
;      170 - Controller Has Gone Off Line

;      ERROR (INTEGER*2) is the same as described in EXEC

;      ROUTINES CALLED: COMINIT,ROUT,XNFIN,CHAROUT,IDEND,DRECIN

```

RSTATUS	PROC PUBLIC	FAR RSTATUS	
	PUSH	AX	;SAVE REGISTERS
	PUSH	BX	
	PUSH	CX	
	PUSH	DX	
	PUSH	SI	
	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	MOV	SP_SAVE,SP	;SAVE STACK POINTER
	CALL	COMINIT	;INITIALIZE SERIAL PORT
	MOV	AH,0	;MOVE 0 INTO AH
	MOV	DSET,AH	;CLEAR DSET
	CALL	ROUT	;TRANSMIT R TO THE CONTROLLER
	CALL	XNFIN	;WAIT FOR XOFF AND THEN XON
	LDS	SI,ES:PAR1[BX]	;LOAD ADDRESS OF CODE
	MOV	DX,[SI]	;LOAD CODE INTO DX
	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	MOV	CBUFF,DL	;MOVE CODE INTO CBUFF
	CALL	CHAROUT	;TRANSMIT CODE TO CONTROLLER
	CALL	IDEND	;TRANSMIT CR,LF
	CMP	DX,1	;DOES CODE = 1
	JZ	MACHNE_STAT	;IF SO, JUMP TO MACHNE_STAT
	CMP	DX,2	;DOES CODE = 2
	JZ	REJECT_STAT	;IF SO, JUMP TO REJECT_STAT
INSTR_ADDR:	CALL	DRECIN	;READ IN D RECORD
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	PPERROR	;IF SO, THEN JUMP TO PPERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	QQERROR	;IF SO, THEN JUMP TO QQERROR

	MOV	DX,DBUFF	;MOVE VALUE FROM BUFFER INTO DX
	LDS	SI,ES:PAR2[BX]	;LOAD PAR3 ADDRESS INTO DS:ES
	MOV	[SI],DX	;LOAD INSTR ADDRESS INTO PAR3
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	JMP	TEXTIT	;JUMP TO TEXTIT
PPERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	TEXTIT	;JUMP TO TEXTIT
QQERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
	JMP	TEXTIT	;JUMP TO TEXTIT
REJECT_STAT:	CALL	DRECIN	;READ IN D RECORD
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	PERROR	;IF SO, THEN JUMP TO PERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	QERROR	;IF SO, THEN JUMP TO QERROR
	MOV	DX,0	;CLEAR DX
	MOV	DL,DBUFF	;MOVE LOW BYTE INTO DL
	LDS	SI,ES:PAR2[BX]	;LOAD PAR3 ADDRESS INTO DS:ES
	MOV	[SI],DX	;LOAD INSTR ADDRESS INTO PAR3
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	JMP	TEXTIT	;JUMP TO TEXTIT
MACHNE_STAT:	MOV	AH,255	;MOVE CHECK VALUE INTO AX
	MOV	SI,1	;LOAD BUFFER OFFSET WITH 1
	MOV	DBUFF[SI],AH	;MOVE CHECK VALUE INTO DBUFF
	CALL	DRECIN	;READ IN D RECORD
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	CMP	AX,04H	;IS ERROR = EOT
	JZ	PERROR	;IF SO, THEN JUMP TO PERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	QERROR	;IF SO, THEN JUMP TO QERROR
	MOV	SI,1	;LOAD BUFFER OFFSET WITH 1
	MOV	AH,DBUFF[SI]	;MOVE HIGH BYTE INTO AH
	CMP	AH,255	;DOES HIGH BYTE = 255
	JNZ	HIGH_BYTE_IN	;IF NOT, JUMP TO HIGH_BYTE_IN
	MOV	DX,0	;CLEAR DX
	MOV	DL,DBUFF	;MOVE LOW BYTE INTO DL
	LDS	SI,ES:PAR2[BX]	;LOAD PAR3 ADDRESS INTO DS:ES
	MOV	[SI],DX	;LOAD INSTR ADDRESS INTO PAR3
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	JMP	TEXTIT	;JUMP TO TEXTIT

```

HIGH_BYTE_IN:    MOV     CX,0           ;CLEAR CX
                  MOV     CL,AH         ;MOVE HIGH BYTE INTO CL

                  MOV     DX,0         ;CLEAR DX
                  MOV     DL,DBUFF      ;MOVE LOW BYTE INTO DL

                  LDS     SI,ES:PAR2[BX] ;LOAD PAR3 ADDRESS INTO DS:ES
                  MOV     [SI],CX       ;LOAD HIGH CODE INTO PAR3

                  LDS     SI,ES:PAR3[BX] ;LOAD PAR4 ADDRESS INTO DS:ES
                  MOV     [SI],DX       ;LOAD LOW CODE INTO PAR4

                  MOV     CX,DATAS      ;POINT TO DATA SEGMENT
                  MOV     DS,CX

                  JMP     TEXTIT         ;JUMP TO TEXTIT

PERROR:          MOV     AX,1           ;LOAD 1 INTO AX
                  MOV     ERROR,AX      ;SET ERROR = 1
                  JMP     TEXTIT        ;JUMP TO TEXTIT

QERROR:          MOV     AX,2           ;LOAD 2 INTO AX
                  MOV     ERROR,AX      ;SET ERROR = 2

TEXTIT:          MOV     AX,ERROR       ;MOVE CONTENTS OF ERROR INTO AX
                  LDS     SI,ES:PAR4[BX] ;LOAD ADDRESS OF PAR2 INTO DS
                  MOV     [SI],AX       ;LOAD ERROR MESSAGE INTO PAR2

                  POP     SI            ;RESTORE REGISTERS
                  POP     DX
                  POP     CX
                  POP     BX
                  POP     AX

                  RET                  ;RETURN

RSTATUS          ENDP

PAGE +
SUBTTL RDIDO

```

```

; RDIDO reads the status of one of the 16 DI or DO ports from the controller
; by sending the appropriate R command, followed by the reception of the
; necessary D record.

```

```

; The calling sequence for RDIDO is:

```

```

; CALL RDIDO (PORTNUM, STATUS, ERROR)

```

```

; Where:

```

```

; PORTNUM (INTEGER*2) is the desired port number:

```

```

; 1 - 16 for input ports 1 - 16
; 17 - 32 for output ports 1 - 16

```

```

; STATUS (INTEGER*2) is the status of the port:

```

```

; 1 - ON
; 0 - OFF

```

```

; ERROR ( INTEGER*2) is the same as described in EXEC

```

```

; ROUTINES CALLED: COMINIT,ROUT,XNFIN,DEC16OUT,IDEND,DRECIN

```

RDIDO	PROC PUBLIC	FAR RDIDO	
	PUSH PUSH PUSH PUSH PUSH	AX BX CX DX SI	;SAVE REGISTERS
	MOV MOV MOV	AX,DATAS DS,AX SP_SAVE,SP	;POINT TO DATA SEGMENT ;SAVE STACK POINTER
	CALL	COMINIT	;INITIALIZE SERIAL PORT
	MOV MOV	AH,1 DSET,AH	;MOVE 1 INTO AH ;SET DSET = 1
	CALL CALL MOV CALL CALL	ROUT XNFIN DX,10 DEC16OUT IDEND	;TRANSMIT R TO THE CONTROLLER ;WAIT FOR XOFF AND THEN XON ;LOAD 10 INTO DX ;TRANSMIT CODE ;TRANSMIT CR,LF
	CALL	DRECIN	;READ IN D RECORD
	MOV CMP JZ CMP JZ	AX,ERROR AX,04H TERROR AX,015H UERROR	;MOVE CONTENTS OF ERROR INTO AX ;IS ERROR = EOT ;IF SO, THEN JUMP TO TERROR ;IS ERROR = NAK ;IF SO, THEN JUMP TO UERROR
	MOV MOV	DX,DBUFF OSTAT,DX	;LOAD 0 STATUS WORD INTO DX ;LOAD VALUE INTO OSTAT
	MOV MOV MOV	SI,2 DX,DBUFF[SI] ISTAT,DX	;LOAD 2 INTO BUFFER OFFSET ;LOAD 1 STATUS WORD INTO DX ;LOAD VALUE INTO OSTAT
	JMP	EXTRA_IN	;JUMP TO EXTRA_IN
TERROR:	MOV MOV JMP	AX,1 ERROR,AX UEXIT	;LOAD 1 INTO AX ;SET ERROR = 1 ;JUMP TO UEXIT
UERROR:	MOV MOV JMP	AX,2 ERROR,AX UEXIT	;LOAD 2 INTO AX ;SET ERROR = 2 ;JUMP TO UEXIT
EXTRA_IN:	CALL	DRECIN	;READ IN EXTRANEIOUS D RECORD
	MOV CMP JZ CMP JZ	AX,ERROR AX,04H TERROR AX,015H UERROR	;MOVE CONTENTS OF ERROR INTO AX ;IS ERROR = EOT ;IF SO, THEN JUMP TO TERROR ;IS ERROR = NAK ;IF SO, THEN JUMP TO UERROR
	MOV MOV CALL	AH,0 DSET,AH DRECIN	;LOAD 0 INTO AH ;CLEAR DSET
	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX

	CMP	AX,04H	;IS ERROR = EOT
	JZ	RERROR	;IF SO, THEN JUMP TO RERROR
	CMP	AX,015H	;IS ERROR = NAK
	JZ	SERROR	;IF SO, THEN JUMP TO SERROR
	LDS	SI,ES:PAR1[BX]	;LOAD ADDRESS OF PAR1 INTO DS:ES
	MOV	AX,[SI]	;LOAD PORT # INTO AX
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	CMP	AX,17	;IS PORT # BELOW 17
	JB	INPUT_PORT	;IF SO, JUMP TO INPUT_PORT
OUTPUT_PORT:	SUB	AX,17	;SUBTRACT 17 FROM AX
	MOV	SI,15	;LOAD SI WITH 15
	SUB	SI,AX	;COMPUTE BIT_TABW OFFSET
	MOV	AX,SI	;MOVE OFFSET INTO AX
	MOV	DL,2	;MOVE MULTIPLICAND INTO DL
	MUL	DL	;MULTIPLY OFFSET BY 2
	MOV	SI,AX	;MOVE OFFSET BACK INTO SI
	MOV	AX,OSTAT	;MOVE OUTPUT STATUS WORD INTO AX
	AND	AX,BIT_TABW[SI]	;CHECK OUTPUT PORT
	JZ	OFF_STATUS	;IF 0, JUMP TO OFF_STATUS
	JMP	ON_STATUS	;OTHERWISE, JUMP TO ON_STATUS
INPUT_PORT:	SUB	AX,1	;SUBTRACT 1 FROM AX
	MOV	SI,15	;LOAD SI WITH 15
	SUB	SI,AX	;COMPUTE BIT_TABW OFFSET
	MOV	AX,SI	;MOVE OFFSET INTO AX
	MOV	DL,2	;MOVE MULTIPLICAND INTO DL
	MUL	DL	;MULTIPLY OFFSET BY 2
	MOV	SI,AX	;MOVE OFFSET BACK INTO SI
	MOV	AX,ISTAT	;MOVE INPUT STATUS WORD INTO AX
	AND	AX,BIT_TABW[SI]	;CHECK OUTPUT PORT
	JZ	OFF_STATUS	;IF 0, JUMP TO OFF_STATUS
ON_STATUS:	LDS	SI,ES:PAR2[BX]	;LOAD ADDRESS OF PAR2 INTO DS:ES
	MOV	AX,1	;MOVE 1 INTO AX
	MOV	[SI],AX	;MOVE 1 INTO STATUS PARAMETER
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	JMP	UEXIT	;JUMP TO UEXIT
OFF_STATUS:	LDS	SI,ES:PAR2[BX]	;LOAD ADDRESS OF PAR2
	MOV	AX,0	;MOVE 0 INTO AX
	MOV	[SI],AX	;MOVE 0 INTO STATUS PARAMETER
	MOV	CX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,CX	
	JMP	UEXIT	;JUMP TO UEXIT
RERROR:	MOV	AX,1	;LOAD 1 INTO AX
	MOV	ERROR,AX	;SET ERROR = 1
	JMP	UEXIT	;JUMP TO UEXIT
SERROR:	MOV	AX,2	;LOAD 2 INTO AX
	MOV	ERROR,AX	;SET ERROR = 2
UEXIT:	MOV	AX,ERROR	;MOVE CONTENTS OF ERROR INTO AX
	LDS	SI,ES:PAR3[BX]	;LOAD ADDRESS OF PAR3 INTO DS

```

MOV     [SI],AX           ;LOAD ERROR MESSAGE INTO PAR3
POP     SI                ;RESTORE REGISTERS
POP     DX
POP     CX
POP     BX
POP     AX

RET                     ;RETURN

RDIDO    ENDP

```

PAGE +
SUBTTL PUTWAIT

```

;   PUTWAIT waits for the execution of a PUT command by the 7545 controller
;   by sending the appropriate responses for the incoming D records.

;   The calling sequence for PUTWAIT is:

;       CALL PUTWAIT (ERROR)

;   Where:

;       ERROR (INTEGER*2) is the same as described in EXEC.

;       ROUTINES CALLED: COMINIT,XNOUT,DRECIN

```

```

PUTWAIT    PROC    FAR
PUBLIC     PUTWAIT

PUSH     AX               ;SAVE REGISTERS
PUSH     BX
PUSH     CX
PUSH     DX
PUSH     SI

MOV     AX,DATAS          ;POINT TO DATA SEGMENT
MOV     DS,AX
MOV     SP_SAVE,SP        ;SAVE STACK POINTER

CALL     COMINIT          ;INITIALIZE THE SERIAL PORT

MOV     AH,1              ;MOVE 1 INTO AH
MOV     DSET,AH           ;SET DSET = 1

CALL     XNOUT             ;SEND XON TO THE CONTROLLER
CALL     DRECIN           ;WAIT FOR FIRST D RECORD

MOV     AX,ERROR          ;MOVE CONTENTS OF ERROR INTO AX
CMP     AX,04H            ;IS ERROR = EOT
JZ      SRERROR           ;IF SO, THEN JUMP TO SRERROR
CMP     AX,015H           ;IS ERROR = NAK
JZ      RSERROR           ;IF SO, THEN JUMP TO RSERROR

MOV     AH,0              ;MOVE 0 INTO AH
MOV     DSET,AH           ;SET DSET = 0
CALL     DRECIN           ;WAIT FOR SECOND D RECORD

MOV     AX,ERROR          ;MOVE CONTENTS OF ERROR INTO AX
CMP     AX,04H            ;IS ERROR = EOT

```

```

                JZ      SRERROR      ;IF SO, THEN JUMP TO SRERROR
                CMP     AX,015H      ;IS ERROR = NAK
                JZ      RSERROR      ;IF SO, THEN JUMP TO RSERROR
                JMP     NEXIT         ;JUMP TO NEXIT

SRERROR:        MOV     AX,1         ;LOAD 1 INTO AX
                MOV     ERROR,AX     ;SET ERROR = 1
                JMP     NEXIT         ;JUMP TO NEXIT

RSERROR:        MOV     AX,2         ;LOAD 2 INTO AX
                MOV     ERROR,AX     ;SET ERROR = 2

NEXIT:          MOV     AX,ERROR     ;MOVE CONTENTS OF ERROR INTO AX

                LDS     SI,ES:PAR1[BX] ;LOAD ADDRESS OF PAR3 INTO DS
                MOV     [SI],AX      ;LOAD ERROR MESSAGE INTO PAR1

                POP     SI           ;RESTORE REGISTERS
                POP     DX
                POP     CX
                POP     BX
                POP     AX

                RET                 ;RETURN

PUTWAIT        ENDP

```

PAGE +
SUBTTL DRECOU

```

;   DRECOU constructs a D record by assembling the D identifier, byte count,
;   data, and check sum. It then sends the information package through the
;   serial port to the 7545 controller, followed by CR and LF.
;
;   Input: BYTECNT contains the number of bytes of data in the record.
;          DBUFF contains the data to be transmitted.
;
;   Output: A D record is sent out the serial port to the 7545 controller.
;
;   Registers Altered: None.
;
;   Routines Called: DOUT, IDEND, CHECKSUM, and CHAROUT.

```

```

DRECOU          PROC
PUBLIC          DRECOU

                PUSH     AX          ;SAVE REGISTERS
                PUSH     BX
                PUSH     CX
                PUSH     DX
                PUSH     SI

                MOV     AX,DATAS     ;POINT TO DATA SEGMENT
                MOV     DS,AX

                CALL     CHECKSUM     ;COMPUTE CHECKSUM OF DATA
                CALL     DOUT         ;TRANSMIT D TO THE CONTROLLER

```



```

                                MOV     DX,BYTECNT      ;MOVE BYTE COUNT INTO DX
                                MOV     CBUF,DL          ;MOVE LOWER BYTE INTO CBUF
                                CALL    CHAROUT          ;TRANSMIT BYTECOUNT TO CONTROLLER

DATA_OUT_LOOP:                MOV     SI,BYTECNT      ;MOVE BYTE COUNT INTO SI
                                DEC     SI              ;DECREMENT SI
                                MOV     DH,DBUFF[SI]    ;MOVE BYTE TO DH
                                MOV     CBUF,DH         ;MOVE BYTE TO CBUF
                                CALL    CHAROUT          ;TRANSMIT BYTE TO CONTROLLER

                                CMP     SI,0            ;IS SI = 0
                                JZ      CHECKSUM_OUT     ;IF SO, JUMP TO CHECKSUM_OUT
                                JMP     DATA_OUT_LOOP   ;JUMP TO DATA_OUT_LOOP

CHECKSUM_OUT:                 MOV     DH,CHKSUM        ;MOVE CHECK SUM INTO DH
                                MOV     CBUF,DH         ;MOVE BYTE TO CBUF
                                CALL    CHAROUT          ;TRANSMIT CHECK SUM TO CONTROLLER
                                CALL    IDEND           ;TRANSMIT CR,LF TO CONTROLLER

                                POP     SI              ;RESTORE REGISTERS
                                POP     DX
                                POP     CX
                                POP     BX
                                POP     AX
                                RET                     ;RETURN

DRECONT                       ENDP

```

PAGE +
SUBTTL DRECIN

```

; DRECIN receives a D record from the 7545 controller, determines whether
; the record is correct by computing the checksum of the record and
; comparing it to the transmitted value, transmits either an ACK or NAK
; to the controller depending upon the record's status, waits for an
; EG from the controller, and finally sends the controller an ACK to
; finish the communication sequence.
;
; If DSET equals 1, the routine will not wait for an EG from the controller.
; Instead it will simply send either an ACK or NAK before exiting. If
; DSET equals 0, then the routine will wait for an EG from the controller
; as described in above.
;
; Input: DSET contains either a 1 (Do not wait for EG) or a 0 (Wait for EG).
;
; Output:BYTECNT contains the number of bytes in the data field.
;         DBUFF contains the transmitted data.
;
; Registers Altered: None.
;
; Routines Called: COMINCK, CHARIN, CHECKSUM, COMOUT,and ACKOUT.

```

```

DRECIN                        PROC

                                PUSH    AX             ;SAVE REGISTERS
                                PUSH    CX
                                PUSH    SI
                                PUSH    DS

```

	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	MOV	AX,0	;MOVE 0 INTO AX
	MOV	ERROR,AX	;SET ERROR = 0
D_IN:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	D_IN	;BUFFER EMPTY, CHECK AGAIN
	CMP	AL,68	;IS CHARACTER = D
	JNZ	D_IN	;IF NOT, CHECK AGAIN
B_CNT_HIGH:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	B_CNT_HIGH	;IF EMPTY, CHECK AGAIN
	MOV	DH,AL	;MOVE HIGH BYTE INTO DH
B_CNT_LOW:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	B_CNT_LOW	;IF EMPTY, CHECK AGAIN
	MOV	AH,DH	;MOVE UPPER 4 BITS INTO AH
	CALL	CHARIN	;CONVERT ASCII VALUES INTO
			;USABLE FORM
	MOV	CX,0	;CLEAR CX
	MOV	CL,CBUFF	;MOVE ACTUAL BYTE INTO CL
	MOV	BYTECNT,CX	;MOVE VALUE INTO BYTECNT
	MOV	SI,CX	;MOVE VALUE INTO OFFSET
	DEC	SI	;DECREMENT OFFSET
DATA_IN_HIGH:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	DATA_IN_HIGH	;IF EMPTY, CHECK AGAIN
	MOV	DH,AL	;MOVE HIGH BYTE INTO DH
DATA_IN_LOW:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	DATA_IN_LOW	;IF EMPTY, CHECK AGAIN
	MOV	AH,DH	;MOVE UPPER 4 BITS INTO DH
	CALL	CHARIN	;CONVERT ASCII VALUES INTO
			;USABLE FORM
	MOV	AL,CBUFF	;MOVE VALUE INTO AL
	MOV	DBUFF[SI],AL	;MOVE VALUE INTO DBUFF
	CMP	CX,1	;IS CX = 1
	JZ	C_SUM_HIGH	;IF SO, JUMP TO C_SUM_HIGH
	DEC	CX	;DECREMENT COUNTER
	DEC	SI	;DECREMENT DBUFF OFFSET
	JMP	DATA_IN_HIGH	;RECEIVE NEXT BYTE OF DATA
C_SUM_HIGH:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	C_SUM_HIGH	;IF EMPTY, CHECK AGAIN
	MOV	DH,AL	;MOVE HIGH BYTE INTO DH
C_SUM_LOW:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	C_SUM_LOW	;IF EMPTY, CHECK AGAIN
	MOV	AH,DH	;MOVE UPPER 4 BITS INTO AH
	CALL	CHARIN	;CONVERT ASCII VALUES INTO
CR_IN:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	CR_IN	;IF EMPTY CHECK AGAIN
	CMP	AL,13	;DOES CHARACTER = CR
	JNZ	CR_IN	;IF NOT, CHECK AGAIN
LF_IN:	CALL	COMINCK	;CHECK COMMUNICATION LINE
	JZ	LF_IN	;IF EMPTY, CHECK AGAIN
	CMP	AL,10	;DOES CHARACTER = LF
	JNZ	LF_IN	;IF NOT, CHECK AGAIN
CHECK_SUM:	CALL	CHECKSUM	;COMPUTE CHECK SUM

```

                MOV     AH,CBUFF           ;MOVE TRANSMITTED CHECK SUM
                MOV     AL,CHKSUM           ;INTO AH
                CMP     AH,AL               ;MOVE COMPUTED CHECK SUM INTO AL
                JZ      ACK_OUT             ;ARE THEY EQUAL
                                           ;IF SO, TRANSMIT ACK

NAK_OUT:        MOV     AL,015H           ;LOAD NAK (ASCII) INTO AL
                CALL    COMOUT              ;TRANSMIT NAK TO CONTROLLER
                MOV     AX,015H           ;LOAD 2 INTO AX
                MOV     ERROR,AX           ;SET ERROR = 2
                JMP     QEXIT              ;JUMP TO QEXIT

ACK_OUT:        CALL    ACKOUT             ;TRANSMIT ACK TO CONTROLLER

                MOV     AH,DSET            ;MOVE CONTENTS OF DSET INTO AH
                CMP     AH,1               ;DOES AH = 1
                JZ      QEXIT              ;IF SO, JUMP TO QEXIT

E_IN:          CALL    COMINCK             ;CHECK COMMUNICATION LINE
                JZ      E_IN               ;IF EMPTY, CHECK AGAIN
                CMP     AL,69              ;DOES CHARACTER = E
                JNZ     E_IN               ;IF NOT, CHECK AGAIN

G_IN:          CALL    COMINCK             ;CHECK COMMUNICATION LINE
                JZ      G_IN               ;IF EMPTY, CHECK AGAIN
                CMP     AL,71              ;DOES CHARACTER = G
                JNZ     G_IN               ;IF NOT, CHECK AGAIN

CR_INN:        CALL    COMINCK             ;CHECK COMMUNICATION LINE
                JZ      CR_INN             ;IF EMPTY, CHECK AGAIN
                CMP     AL,13              ;DOES CHARACTER = CR
                JNZ     CR_INN             ;IF NOT, CHECK AGAIN

LF_INN:        CALL    COMINCK             ;CHECK COMMUNICATION LINE
                JZ      LF_INN             ;IF EMPTY, CHECK AGAIN
                CMP     AL,10              ;DOES CHARACTER = LF
                JNZ     LF_INN             ;IF NOT, CHECK AGAIN
                CALL    ACKOUT             ;TRANSMIT ACK TO CONTROLLER

QEXIT:         POP     DS                  ;RESTORE REGISTERS
                POP     SI
                POP     CX
                POP     AX
                RET                          ;RETURN

DRECIN        ENDP

```

PAGE +
SUBTTL CNVRT86Z80

```

;   CNVRT86Z80 converts a value stored in 8086 single precision floating
;   point format to Z80 single precision floating point format. This
;   routine utilizes the 8087 coprocessor.
;
;   Input: Value to be converted is stored in REAL86.
;
;   Output: Converted value is stored in RBUFF.
;
;   Registers Altered: None.
;
;   Routines Called: SETBIT.

```

```

CNVRT86Z80          PROC
                    PUSH    AX              ;SAVE REGISTERS

```

	PUSH	BX	
	PUSH	CX	
	PUSH	DX	
	PUSH	DS	
	PUSH	SI	
	PUSH	DI	
	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	MOV	AX,0	;MOV 0 INTO AX
	MOV	RBUF4,AX	;CLEAR LSM OF RBUF4
	MOV	DI,2	;MOV OFFSET INTO DI
	MOV	RBUF4[DI],AX	;CLEAR MSW OF RBUF4
EXP_SEARCH:	MOV	AH,REAL4	;LOAD MSB OF REAL86 INTO AH
	AND	AH,080H	;SAVE SIGN BIT
	MOV	RBUF4,AH	;SET SIGN BIT IN MSB OF RBUF4
	MOV	AH,REAL4	;LOAD MSB OF REAL86 INTO AH
	AND	AH,07FH	;CLEAR SIGN BIT
	MOV	REAL4,AH	;MOVE MSB BACK INTO REAL86
EXPFIND:	FINIT		;INITIALIZE THE 8087 PROCESSOR
INIT1:	FLD	REAL86	;LOAD REAL # ON TO STACK
ZERO_CHECK:	FLDZ		;LOAD ZERO ON TO STACK
	FCOM	ST(1)	;COMPARE REAL # TO ZERO
	FSTSW	STATHRD	;MOVE 8087 STATUS WORD INTO STOR
	FWAIT		;WAIT UNTIL STORE IS COMPLETE
	MOV	AH,STATBYTE	;LOAD STATUS INTO AH
	SAHF		;MOVE FLAGS [C0=CF,C3=ZF]
	JNZ	EXP_GO	;IF NOT EQUAL,JUMP TO EXP_GO
	MOV	AH,1	;SET AH = 1
	MOV	RBUF4,AH	;SET RBUF4 = 1
	MOV	AH,0	;SET AH = 0
	MOV	RBUF3,AH	;SET RBUF3 = 0
	MOV	RBUF2,AH	;SET RBUF2 = 0
	MOV	RBUF1,AH	;SET RBUF1 = 0
	FSTP	RGRBGE	;CLEAR TOP OF STACK
	JMP	EXIT	;JUMP TO EXIT
EXP_GO:	FSTP	RGRBGE	;CLEAR TOP OF STACK
	MOV	DX,17	;LOAD MAX EXP VALUE INTO DX
	MOV	CX,41	;LOAD COUNTER VALUE INTO CX
START1:	DEC	DX	;DECREMENT EXP VALUE
	FLD1		;LOAD 1.0 ON TO STACK
	MOV	EXPINT1,DX	;MOV EXP VALUE INTO EXPINT
	FILD	EXPINT1	;LOAD EXP VALUE ON TO STACK
	FXCH	ST(1)	;EXCHANGE ST(1) AND ST(0)
	FSCALE		;FIND 2**EXPINT
	FCOM	ST(2)	;COMPARE REAL # WITH COMPUTED #
	FSTSW	STATHRD	;MOVE 8087 STATUS WORD INTO STOR
	FWAIT		;WAIT UNTIL STORE IS COMPLETE
	MOV	AH,STATBYTE	;LOAD STATUS INTO AH
	SAHF		;MOVE FLAGS [C0=CF,C3=ZF]
	JE	EQUAL1	;IF REAL # = COMP #, THEN JUMP ;TO EQUAL1

	JB	BELOW1	;IF REAL # > COMP #, THEN JUMP ;TO BELOW1
FINISH1:	FSTP FISTP LOOP	RGRBGE IGRBGE START1	;POP REAL GARBAGE ;POP INTEGER GARBAGE ;LOOP TO START1
EQUAL1:	MOV AND ADD ADD MOV MOV MOV MOV MOV FSTP FISTP FSTP JMP	AH,DL AH,07FH AH,1 RBUFF4,AH AH,128 RBUFF3,AH AH,0 RBUFF2,AH RBUFF1,AH RGRBGE IGRBGE RGRBGE EXIT	;MOV EXP VALUE INTO AH ;CLEAR MSB OF EXPONENT ;ADD 1 TO AH ;SET EXPONENT OF RBUFF ;SET AH = 128 ;SET RBUFF1 = 255 ;SET AH = 0 ;SET RBUFF2 = 0 ;SET RBUFF3 = 0 ;CLEAR 8087 STACK ;JUMP TO EXIT
BELOW1:	INC MOV MOV AND ADD	DX EXPINT1,DX AL,DL AL,07FH RBUFF4,AL	;INCREMENT THE EXP VALUE ;MOV EXP VALUE INTO EXPINT1 ;MOV EXP VALUE INTO AX ;CLEAR MSB OF EXPONENT ;SET EXPONENT OF RBUFF
MANFIND:	DEC MOV MOV FILD FLD1 FSCALE FMUL FXCH FISTP FMUL FXCH FXCH MOV MOV CALL	DX CX,-24 EXPINT2,CX EXPINT2 ST(0),ST(4) ST(1) IGRBGE ST(0),ST(3) ST(2) ST(1) AX,1 BITSET,AX SETBIT	;DECREMENT THE EXPONENT IN DX ;MOVE EXP INTO CX ;MOVE EXP INTO EXPINT2 ;LOAD EXP ON TO STACK ;LOAD 1.0 ON TO STACK ;COMPUTE 2**EXPINT2 ;COMPUTE LOWER DIFF. BND. ;EXCHANGE ST(0) AND ST(1) ;CLEAR TOP OF STACK ;COMPUTE LOWER DIFF. BND. ;EXCHANGE ST(0) AND ST(2) ;EXCHANGE ST(0) AND ST(1) ;MOVE 1 INTO AX ;MOVE 1 INTO BITSET ;SET THE CORRESPONDING BIT
START2:	FSUBR FCOM FSTSW FWAIT MOV SAHF JA	ST(0),ST(3) ST(2) STATWRD AH,STATBYTE CONT1	;SUBTRACT ST(0) FROM ST(3) ;COMPARE DIFF TO LOWER DIFF BND ;MOVE 8087 STATUS WORD INTO STOR ;WAIT UNTIL STORE IS COMPLETE ;LOAD STATUS INTO AH ;MOVE FLAGS [C0=CF,C3=ZF] ;IF DIFF>BND, THEN JUMP TO CONT1
FINISH2:	FSTP FISTP FSTP FSTP JMP	RGRBGE IGRBGE RGRBGE RGRBGE EXIT	;CLEAR 8087 STACK
CONT1: CONT2:	FXCH FSTP FISTP MOV FILD FLD1 FSCALE	ST(3) RGRBGE IGRBGE EXPINT2,DX EXPINT2 	;EXCHANGE ST(0) AND ST(3) ;CLEAR TOP OF 8087 STACK ;LOAD NEW EXP INTO EXPINT2 ;LOAD NEW EXP ON TO STACK ;LOAD 1.0 ON TO STACK ;COMPUTE 2**EXPINT2

	FCOM	ST(3)	;COMPARE DIFF WITH NEW VALUE
	FSTSW	STATWRD	;LOAD 8087 STATUS WORD INTO STOR
	FWAIT		;WAIT UNTIL STORE IS COMPLETE
	MOV	AH,STATBYTE	;LOAD STATUS INTO AH
	SAHF		;MOVE FLAGS [C0=CF,C3=ZF]
	JA	CONT3	;IF VALUE>DIF,THEN JUMP TO CONT3
	MOV	AX,EXPINT1	;MOVE REAL # EXP INTO AX
	SUB	AX,DX	;SUBTRACT NEW EXP FROM REAL# EXP
	MOV	BITSET,AX	;MOVE DIFFERENCE INTO BITSET
	CALL	SETBIT	;SET THE APPROPRIATE BIT
	JMP	START2	
CONT3:	DEC	DX	;DECREMENT EXP VALUE IN DX
	JMP	CONT2	;JUMP TO CONT2
EXIT:	POP	DI	;RESTORE REGISTERS
	POP	SI	
	POP	DS	
	POP	DX	
	POP	CX	
	POP	BX	
	POP	AX	
	RET		;RETURN
CNVRT86Z80	ENDP		

PAGE +
SUBTTL CNVRTZ8086

```

; CNVRTZ8086 converts a real number stored in Z80 single precision floating
; point format into a value stored in 8086 single precision floating point
; format. This routine utilizes the 8087 coprocessor.

; Input: Value to be converted is stored in RBUFF.

; Output: Converted value is stored in REAL86.

; Registers Altered: None.

; Routines Called: EXPADD

```

CNVRTZ8086		PROC	
	PUSH	AX	;SAVE REGISTERS
	PUSH	BX	
	PUSH	CX	
	PUSH	DX	
	PUSH	SI	
	PUSH	DI	
	PUSH	DS	
	MOV	AX,DATAS	;POINT TO DATA SEGMENT
	MOV	DS,AX	
	FINIT		;INITIALIZE THE 8087 PROCESSOR
	FLDZ		;LOAD 0.0 ON TO STACK
	MOV	AH,RBUFF3	;MOVE MSB OF MANTISSA INTO AH
	MOV	BITBUFF,AH	;THEN MOVE IT INTO BITBUFF

	MOV	DX,-1	;MOVE BEGINNING EXP INTO DX
	CALL	EXPADD	;BEGIN CONSTRUCTION OF REAL #
	MOV	AH,RBUFF2	;MOVE NEXT 8 BITS OF MANTISSA
	MOV	BITBUFF,AH	;INTO BITBUFF
	MOV	DX,-9	;MOVE BEGINNING EXP INTO DX
	CALL	EXPADD	;CONT. CONSTRUCTION OF REAL #
	MOV	AH,RBUFF1	;MOVE FINAL 8 BITS OF MANTISSA
	MOV	BITBUFF,AH	;INTO BITBUFF
	MOV	DX,-17	;MOVE BEGINNING EXP INTO DX
	CALL	EXPADD	;FINISH CONSTRUCTION OF MANTISSA
	MOV	AH,RBUFF4	;MOVE EXP FIELD INTO AH
	TEST	AH,040H	;TEST THE EXP SIGN BIT
	JZ	POSSET	;IF 0, THEN JUMP TO POSSET
NEGSET:	TEST	AH,080H	;TEST MSB OF EXP FIELD
	JNZ	NEGDH	;IF SET, THEN JUMP TO NEGDH
	ADD	AH,080H	;SET THE MSB OF EXP FIELD
NEGDH:	MOV	DH,0FFH	;SET ALL OF THE BITS IN DH
	MOV	DL,AH	;MOVE EXP FIELD INTO DL
	JMP	EXPMULT	;JUMP TO EXPMULT
POSSET:	TEST	AH,080H	;TEST MSB OF EXP FIELD
	JZ	POSDH	;IF 0, THEN JUMP TO POSDH
	AND	AH,07FH	;CLEAR MSB OF EXP FIELD
POSDH:	MOV	DH,0	;CLEAR ALL BITS IN DH
	MOV	DL,AH	;MOVE EXP FIELD INTO DL
EXPMULT:	MOV	EXPINT1,DX	;MOVE EXP FIELD INTO EXPINT1
	FILD	EXPINT1	;LOAD EXP ON TO STACK
	FLD1		;LOAD 1.0 ON TO STACK
	FSCALE		;COMPUTE 2**EXPINT1
	FMUL	ST(0),ST(2)	;COMPUTE FINAL PRODUCT
	FSTP	REAL86	;STORE REAL # INTO REAL86
	FISTP	IGRBGE	;CLEAR 8087 STACK
	FSTP	RGRBGE	
	MOV	AH,RBUFF4	;MOVE EXP FIELD INTO AH
	TEST	AH,080H	;CHECK SIGN OF MANTISSA
	JZ	CEXIT	;IF 0, THEN JUMP TO CEXIT
	MOV	AX,REALW2	;MOVE MSW OF REAL # INTO AX
	ADD	AX,08000H	;SET SIGN OF MANTISSA
	MOV	REALW2,AX	;MOVE MSW BACK INTO REAL86
CEXIT:	POP	DS	;RESTORE REGISTERS
	POP	DI	
	POP	SI	
	POP	DX	
	POP	CX	
	POP	BX	
	POP	AX	
	RET		;RETURN
CNVRTZ8086	ENDP		

PAGE +
SUBTTL CHECKSUM

; CHECKSUM computes the modulo 256 of the data stored in DBUFF. This
; routine utilizes the 8087 coprocessor.

```

;
;   Input: BYTECNT contains the number of bytes to be checked.
;         DBUFF contains the data to be checked.
;
;   Output: CHKSUM contains the modulo 256 of the data stored in DBUFF.
;
;   Registers Altered: None.
;
;   Routines Called: None.

```

CHECKSUM	PROC	
	PUSH AX	;SAVE REGISTERS
	PUSH BX	
	PUSH CX	
	PUSH DX	
	PUSH SI	
	MOV AX,DATAS	;POINT TO DATA SEGMENT
	MOV DS,AX	
	MOV AX,0	;CLEAR AX
	MOV SI,BYTECNT	;MOVE BYTE COUNT INTO SI
	DEC SI	;DECREMENT OFFSET
	FINIT	;INITIALIZE THE 8087 PROCESSOR
	MOV AL,DBUFF[SI]	;MOVE THE FIRST BYTE INTO AL
	MOV CHKSUMW,AX	;MOVE THE VALUE INTO CHKSUMW
	FILD CHKSUMW	;LOAD VALUE ON TO STACK
	CMP SI,0	;IS SI EQUAL TO 1
	JZ MODULO	;IF YES, JUMP TO MODULO
MLOOP:	DEC SI	;DECREMENT THE BYT COUNT
	MOV AL,DBUFF[SI]	;MOVE NEXT BYTE INTO AL
	MOV CHKSUMW,AX	;MOVE THE VALUE INTO CHKSUMW
	FIADD CHKSUMW	;ADD VALUE TO TOP OF STACK
	CMP SI,0	;IS SI EQUAL TO 0
	JZ MODULO	;IF YES, JUMP TO MODULO
	JMP MLOOP	;JUMP TO MLOOP
MODULO:	MOV AX,256	;LOAD 256 INTO AX
	MOV CHKSUMW,AX	;LOAD THE VALUE INTO CHKSUMW
	FILD CHKSUMW	;LOAD THE VALUE ON TO THE STACK
	FXCH ST(1)	;EXCHANGE ST(0) AND ST(1)
	FPREM	;COMPUTE SUM MODULO 256
	FISTP CHKSUMW	;LOAD THE VALUE INTO CHKSUMW
	FISTP IGRBGE	;CLEAR THE STACK
	MOV AX,CHKSUMW	;LOAD CONTENTS OF CHKSUMW INTO AX
	MOV CHKSUM,AL	;LOAD CHECK SUM INTO CHKSUM
	POP SI	;RESTORE REGISTERS
	POP DX	
	POP CX	
	POP BX	
	POP AX	
	RET	;RETURN
CHECKSUM	ENDP	

PAGE +
SUBTTL DEC16OUT

```

;   DEC16OUT converts the value stored in DX into its appropriate
;   ASCII characters, and sends the characters through the serial
;   communications port to the 7545 robot.

;   Input: DX contains the value to be sent to the controller.

;   Output: A string of ASCII characters is sent to the controller and the
;           characters are stored in TBUF.

;   Registers Altered: None.

;   Routines Called: COMOUT

```

```

DEC16OUT                                PROC
PUBLIC DEC16OUT

    PUSH    DS                        ;SAVE REGISTERS
    PUSH    DI
    PUSH    DX
    PUSH    CX
    PUSH    AX

    MOV     AX,DATAS                  ;POINT TO DATA SEGMENT
    MOV     DS,AX

    MOV     CX,0                      ;INITIALIZE THE COUNTER
    LEA     DI,TBUF                   ;POINT TO THE BUFFER

DEC16OUT1:
    PUSH    CX                        ;SAVE CX
    MOV     AX,DX                      ;NUMERATOR
    MOV     DX,0                      ;CLEAR UPPER HALF
    MOV     CX,10                     ;DIVISOR OF 10
    DIV     CX                        ;DIVIDE
    XCHG    AX,DX                     ;GET QUOTIENT

    ADD     AL,30H                    ;ADD 30H
    MOV     [DI],AL                   ;PUT IN BUFFER
    INC     DI                         ;NEXT BYTE

    POP     CX                        ;RESTORE CX FOR COUNT
    INC     CX                        ;COUNT THE DIGIT
    CMP     DX,0                      ;DONE?
    JNZ     DEC16OUT1

DEC16OUT2:
    DEC     DI                        ;BACK UP THROUGH THE BUFFER
    MOV     AL,[DI]                   ;GET BYTE FROM BUFFER
    CALL    COMOUT

    LOOP    DEC16OUT2

    POP     AX                        ;RESTORE REGISTERS
    POP     CX
    POP     DX
    POP     DI
    POP     DS
    RET                                ;RETURN
DEC16OUT                                ENDP

```

DEC16OUT

ENDP

PAGE +
SUBTTL CHAROUT

```

; CHAROUT converts the byte of data stored in CBUF into its appropriate
; ASCII characters, and sends the characters to the controller through
; the serial communications port.

; Input: CBUF contains the byte of data to be sent.

; Output: Two ASCII characters are sent to the controller.

; Registers Altered: None.

; Routines Called: COMOUT

```

CHAROUT

PROC

```

PUSH AX      ;SAVE REGISTERS
PUSH CX
PUSH SI

```

```

MOV AX,DATA$ ;POINT TO DATA SEGMENT
MOV DS,AX

```

```

MOV AH,CBUF  ;LOAD BYTE INTO AH
ROR AH,1     ;ROTATE RIGHT 4 TIMES
ROR AH,1
ROR AH,1
ROR AH,1

```

```

AND AH,0FH   ;CLEAR UPPER 4 BITS
MOV CL,0     ;LOAD COMPARATOR WITH 0
MOV SI,0     ;LOAD OFFSET WITH 0

```

```

CHECKHIGH:   CMP AH,CL      ;COMPARE AH TO CL
              JZ TRANSHIGH  ;IF EQUAL, JUMP TO TRANSHIGH
              INC CL        ;INCREMENT COMPARATOR
              INC SI        ;INCREMENT OFFSET
              JMP CHECKHIGH ;JUMP TO CHECKHIGH

```

```

TRANSHIGH:   MOV AL,CHARBUF[SI];LOAD ASCII CHAR INTO AL
              CALL COMOUT    ;TRANSMIT CHARACTER TO CONTROLLER

```

```

MOV AH,CBUF  ;LOAD BYTE INTO AH
AND AH,0FH   ;CLEAR UPPER 4 BITS

```

```

MOV CL,0     ;LOAD 0 INTO THE COMPARATOR
MOV SI,0     ;LOAD 0 INTO THE OFFSET

```

```

CHECKLOW:    CMP AH,CL      ;COMPARE AH TO CL
              JZ TRANSLOW   ;IF EQUAL, JUMP TO TRANSLOW
              INC CL        ;INCREMENT COMPARATOR
              INC SI        ;INCREMENT OFFSET
              JMP CHECKLOW  ;JUMP TO CHECKLOW

```

```

TRANSLOW:    MOV AL,CHARBUF[SI];LOAD ASCII CHAR INTO AL

```

```

CALL    COMOUT        ;TRANSMIT CHARACTER TO CONTROLLER

POP     SI            ;RESTORE REGISTERS
POP     CX
POP     AX
RET                                ;RETURN

CHAROUT        ENDP

```

PAGE +
SUBTTL CHARIN

```

;    CHARIN converts the ASCII characters stored in AX into their corresponding
;    byte value, and stores this value in CBUF.

;    Input: AX contains the ASCII characters to be converted to a real value.

;    Output: CBUF contains the real byte of data.

;    Registers Altered: None.

;    Routines Called: None.

```

```

CHARIN        PROC

PUSH     AX            ;SAVE REGISTERS
PUSH     CX
PUSH     DX
PUSH     DS
PUSH     SI

MOV     CX,DATAS        ;POINT TO DATA SEGMENT
MOV     DS,CX

MOV     CL,0            ;CLEAR NIBBLE SET
MOV     SI,0            ;CLEAR CHARBUFF OFFSET

LOW_LOOP:    CMP     AL,CHARBUFF[SI];IS AL = ASCII EQUIVALENT OF CL
JZ      SET_LOW        ;IF SO, SET LOWER FOUR BITS
INC     CL              ;INCREMENT NIBBLE SET
INC     SI              ;INCREMENT OFFSET
JMP     LOW_LOOP        ;JUMP TO LOW_LOOP

SET_LOW:     MOV     DL,CL        ;MOVE NIBBLE SET INTO DL
MOV     CL,0            ;CLEAR NIBBLE SET
MOV     SI,0            ;CLEAR OFFSET

HIGH_LOOP:   CMP     AH,CHARBUFF[SI];IS AH = ASCII EQUIVALENT OF CL
JZ      HIGH_SET        ;IF SO, SET UPPER FOUR BITS
INC     CL              ;INCREMENT NIBBLE SET
INC     SI              ;INCREMTN OFFSET
JMP     HIGH_LOOP        ;JUMP TO HIGH_LOOP

HIGH_SET:    MOV     DH,CL        ;MOVE NIBBLE SET INTO DH
ROL     DH,1            ;ROTATE DH 4 TIMES
ROL     DH,1
ROL     DH,1
ROL     DH,1

AND     DH,0F0H        ;CLEAR LOWER 4 BITS

```

```

ADD     DL,DH           ;SET UPPER BITS IN DL
MOV     CBUF,DL         ;MOVE VALUE INTO CBUF
POP     SI              ;RESTORE REGISTERS
POP     DS
POP     DX
POP     CX
POP     AX
RET                               ;RETURN

```

```
CHARIN      ENDP
```

```
PAGE +
SUBTTL SETBIT
```

```

;   SETBIT sets a bit (defined by BITSET) in RBUF to the value 1.
;   Input: BITSET contains the number of the bit to be set.
;   Output: The appropriate bit in RBUF is set to the value 1.
;   Registers Altered: None.
;   Routines Called: None.

```

```

SETBIT      PROC

PUSH     AX              ;SAVE REGISTERS
PUSH     BX
PUSH     CX
PUSH     DX
PUSH     DI
PUSH     SI
PUSH     DS

MOV     AX,DATAS         ;SET POINTER TO DATA SEGMENT
MOV     DS,AX

MOV     AX,BITSET        ;MOVE VALUE IN BITSET TO AX
CMP     AX,8              ;COMPARE BITSET TO 8
JBE     BUF3             ;IF BITSET<=8,THEN JUMP TO BUF3
CMP     AX,16             ;COMPARE BITSET TO 16
JBE     BUF2             ;IF BITSET<=16,THEN JUMP TO BUF2
CMP     AX,24             ;COMPARE BITSET TO 24
JBE     BUF1             ;IF BITSET<=24,THEN JUMP TO BUF1

BUF3:      MOV     AX,8      ;MOVE 8 INTO AX
SUB     AX,BITSET          ;SUBTRACT BITSET FROM AX
MOV     SI,AX              ;MOVE OFFSET INTO SI
MOV     AH,BIT_TAB[SI]    ;FIND THE BIT TO BE SET
ADD     RBUF3,AH           ;SET THE BIT IN RBUF3
JMP     BEXIT              ;JUMP TO BEXIT

BUF2:      MOV     AX,16      ;MOVE 16 INTO AX
SUB     AX,BITSET          ;SUBTRACT BITSET FROM AX
MOV     SI,AX              ;MOVE OFFSET INTO SI
MOV     AH,BIT_TAB[SI]    ;FIND THE BIT TO BE SET
ADD     RBUF2,AH           ;SET THE BIT IN RBUF2

```

```

                JMP      BEXIT          ;JUMP TO BEXIT

BUFF1:          MOV      AX,24          ;MOVE 24 INTO AX
                SUB      AX,BITSET      ;SUBTRACT BITSET FROM AX
                MOV      SI,AX          ;MOVE OFFSET INTO SI
                MOV      AH,BIT_TAB[SI] ;FIND THE BIT TO BE SET
                ADD      RBUF1,AH        ;SET THE BIT IN RBUF1
                JMP      BEXIT          ;JUMP TO BEXIT

BEXIT:          POP      DS             ;RESTORE REGISTERS
                POP      SI
                POP      DI
                POP      DX
                POP      CX
                POP      BX
                POP      AX
                RET
SETBIT          ENDP

```

PAGE +
SUBTTL EXPADD

```

;   EXPADD receives a byte of mantissa stored in Z80 single precision
;   floating point format and converts the mantissa into a real number.
;   The real number is stored on ST (0) of the 8087 coprocessor, which
;   is utilized in this routine.

;   Input: BITBUFF contains the mantissa value to be converted.

;   Output: ST (0) of the 80287 coprocessor contains the converted number.

;   Registers Altered. None.

;   Routines Called: None.

```

```

EXPADD          PROC

                PUSH     AX             ;SAVE REGISTERS
                PUSH     SI

                MOV      AX,DATAS       ;POINT TO DATA SEGMENT
                MOV      DS,AX
                MOV      SI,7           ;MOVE 7 INTO OFFSET REGISTER

BSTART:         MOV      AH,BITBUFF     ;MOVE MANTISSA BITS INTO AH
                AND      AH,BIT_TAB[SI] ;TEST THE APPROPRIATE BIT
                JZ       BLOOP          ;IF CLEAR, JUMP TO BLOOP

                MOV      EXPINT1,DX      ;MOVE EXP INTO EXPINT1
                FILD     EXPINT1         ;LOAD EXP ON TO STACK
                FLD1     ;LOAD 1.0 ON TO STACK
                FSCALE    ;COMPUTE 2**EXPINT1

                FADD     ST(0),ST(2)     ;ADD RESULT TO SUM
                FXCH     ST(2)          ;EXCHANGE ST(0) WITH ST(2)
                FSTP     RGRBGE        ;CLEAR TOP OF 8087 STACK
                FISTP    IGRBGE

BLOOP:          CMP      SI,0           ;IS SI = 0

```

```

                JZ      DEXIT      ;IF YES, JUMP TO BEXIT
                DEC     SI          ;DECREMENT OFFSET
                DEC     DX          ;DECREMTNT EXP
                JMP     BSTART      ;JUMP TO BSTART

DEXIT:          POP     SI          ;RESTORE REGISTERS
                POP     AX
                RET               ;RETURN

EXPADD          ENDP

```

PAGE +
SUBTTL ROUT

```

;    ROUT transmits an R (ASCII) to the controller through the serial
;    communication port.

;    Input: None

;    Output: R (ASCII) is transmitted to the controller

;    Registers Altered: None

;    Routines Called: COMOUT

```

```

ROUT

                PUBLIC  PROC
                ROUT

                PUSH    AX          ;SAVE REGISTERS
                PUSH    DS

                MOV     AX,DATAS    ;POINT TO DATA SEGMENT
                MOV     DS,AX

                MOV     AL,82       ;MOVE R (ASCII) INTO AL
                CALL    COMOUT      ;TRANSMIT R TO CONTROLLER

                POP     DS          ;RESTORE REGISTERS
                POP     AX
                RET               ;RETURN

ROUT           ENDP

```

PAGE +
SUBTTL COUT

```

;    COUT transmits an C (ASCII) to the controller through the serial
;    communication port.

;    Input: None

;    Output: C (ASCII) is transmitted to the controller

;    Registers Altered: None

;    Routines Called: COMOUT

```

```

COUT                                PROC
                                PUBLIC COUT

                                PUSH AX      ;SAVE REGISTERS
                                PUSH DS

                                MOV AX,DATAS ;POINT TO DATA SEGMENT
                                MOV DS,AX

                                MOV AL,67    ;MOVE C (ASCII) INTO AL
                                CALL COMOUT   ;TRANSMIT C TO THE CONTROLLER
                                POP DS        ;RESTORE REGISTERS
                                POP AX
                                RET          ;RETURN

COUT                                ENDP

```

PAGE +
SUBTTL DOUT

```

; DOUT transmits an D (ASCII) to the controller through the serial
; communication port.

; Input: None

; Output: D (ASCII) is transmitted to the controller

; Registers Altered: None

; Routines Called: COMOUT

```

```

DOUT                                PROC
                                PUBLIC DOUT

                                PUSH AX      ;SAVE REIGSTERS
                                PUSH DS

                                MOV AX,DATAS ;POINT TO DATA SEGMENT
                                MOV DS,AX

                                MOV AL,68    ;MOVE D (ASCII) INTO AL
                                CALL COMOUT   ;TRANSMIT D TO THE CONTROLLER

                                POP DS        ;RESTORE REGISTER
                                POP AX
                                RET          ;RETURN

DOUT                                ENDP

```

PAGE +
SUBTTL IDEND

```

; IDEND transmits a CR (ASCII) and LF (ASCII) to the controller through
; serial communication port.

; Input: None

; Output: CR (ASCII) and LF (ASCII) ARE transmitted to the controller.

```

```

;   Registers Altered: None
;   Routines Called: COMOUT

```

```

IDEND                                PROC
                                     IDEND

                                     PUBLIC
                                     AX
                                     DS
                                     ;SAVE REGISTERS

MOV     AX,DATAS                     ;POINT TO DATA SEGMENT
MOV     DS,AX

MOV     AL,13                        ;LOAD CR (ASCII) TO AL
CALL    COMOUT                       ;TRANSMIT CR TO CONTROLLER

MOV     AL,10                        ;LOAD LF (ASCII) TO AL
CALL    COMOUT                       ;TRANSMIT LF TO CONTROLLER

POP     DS                           ;RESTORE REGISTERS
POP     AX
RET                                           ;RET

IDEND                                ENDP

```

PAGE +
SUBTTL XNOUT

```

;   XNOUT transmits an XON (ASCII) to the controller through the serial
;   communication port.

;   Input: None

;   Output: XON (ASCII) is transmitted to the controller

;   Registers Altered: None

;   Routines Called: COMOUT

```

```

XNOUT                                PROC
                                     XNOUT

                                     PUBLIC
                                     AX
                                     DS
                                     ;STORE REGISTERS

MOV     AX,DATAS                     ;POINT TO DATA SEGMENT
MOV     DS,AX

MOV     AL,17                        ;LOAD XON (ASCII) IN TO AL
CALL    COMOUT                       ;TRANSMIT XON TO CONTROLLER

POP     DS                           ;RESTORE REGISTERS
POP     AX
RET                                           ;RETURN

XNOUT                                ENDP

```


PAGE +
SUBTTL ACKOUT

```

;   ACKOUT transmits an ACK (ASCII) to the controller through the serial
;   communication port.

;   Input: None

;   Output: ACK (ASCII) is transmitted to the controller

;   Registers Altered: None

;   Routines Called: COMOUT

```

```

ACKOUT                                PROC
                                     PUBLIC  ACKOUT

                                     PUSH    AX           ;STORE REGISTERS
                                     PUSH    DS

                                     MOV     AX,DATAS      ;POINT TO DATA SEGMENT
                                     MOV     DS,AX

                                     MOV     AL,06H        ;LOAD ACK (ASCII) INTO AL
                                     CALL    COMOUT         ;TRANSMIT ACK TO CONTROLLER

                                     POP     DS           ;RESTORE REGISTERS
                                     POP     AX
                                     RET                     ;RETURN

ACKOUT                                ENDP

```

PAGE +
SUBTTL EGOUT

```

;   EGOUT transmits an E (ASCII) and a G (ASCII) to the controller through
;   serial communication port, followed by CR (ASCII) AND LF (ASCII).

;   Input: None

;   Output: E (ASCII), G (ASCII), CR (ASCII), and LF (ASCII) are sent to the
;           controller through the serial output port.

;   Registers Altered: None

;   Routines Called: COMOUT and IDEND

```

```

EGOUT                                PROC
                                     PUBLIC  EGOUT

                                     PUSH    AX           ;STORE REGISTERS
                                     PUSH    DS

                                     MOV     AX,DATAS      ;POINT TO DATA SEGMENT
                                     MOV     DS,AX

                                     MOV     AL,69         ;LOAD E (ASCII) INTO AL

```

```

CALL    COMOUT        ;TRANSMIT E TO CONTROLLER

MOV     AL,71         ;LOAD G (ASCII) INTO AL
CALL    COMOUT        ;TRANSMIT G TO CONTROLLER

CALL    IDEND         ;TRANSMIT CR,LF TO CONTROLLER

POP     DS            ;RESTORE REGISTERS
POP     AX
RET     ;RETURN

```

```
EGOUT      ENDP
```

```
PAGE +
SUBTTL XNFIN
```

```

;   XNFIN waits for the controller to send an XOFF (ASCII) and XON (ASCII)
;   to the host computer through the serial communication port. If the
;   controller sends either a EOT (ASCII) or NAK (ASCII), an error signal
;   will be stored in ERROR. Otherwise, ERROR will equal to zero upon exit.
;
;   Input: None.
;
;   Output: ERROR contains error code.
;
;   Registers Altered. None
;
;   Routines Called: COMINCK

```

```

XNFIN      PUBLIC      PROC
              XNFIN

              PUSH      AX          ;SAVE REGISTERS
              PUSH      BX
              PUSH      CX
              PUSH      DX

              MOV       AX,DATAS    ;POINT TO DATA SEGMENT
              MOV       DS,AX

              MOV       AX,0        ;MOVE 0 INTO AX
              MOV       ERROR,AX    ;SET ERROR = 0

XFIN:      CALL    COMINCK          ;CHECK COMMUNICATION LINE
              JZ     XFIN           ;BUFFER EMPTY, CHECK AGAIN
              CMP    AL,04H         ;IS CHARACTER = EOT
              JZ     XERROR         ;IF SO, JUMP TO XERROR
              CMP    AL,015H        ;IS CHARACTER = NAK
              JZ     XXERROR        ;IF SO ,JUMP TO XXERROR

              CMP    AL,13H         ;IS CHARACTER = XOFF
              JZ     XNIN           ;IF SO, WAIT FOR XON
              JMP    XFIN          ;IF NOT, CHECK AGAIN

XNIN:      CALL    COMINCK          ;CHECK COMMUNICATION LINE
              JZ     XNIN           ;BUFFER EMPTY, CHECK AGAIN
              CMP    AL,11H         ;IF CHARACTER = XON
              JZ     EEXIT          ;IF SO, EXIT
              JMP    XNIN          ;IF NOT, CHECK AGAIN

```

```

XERROR:      MOV     AX,04H           ;MOVE 04H INTO AX
              MOV     ERROR,AX        ;SET ERROR = 04H
              JMP     EEXIT           ;JUMP TO EEXIT

XXERROR:     MOV     AX,015H          ;MOVE 015H INTO AX
              MOV     ERROR,AX        ;SET ERROR = 015H

EEXIT:       POP     DX               ;RESTORE REGISTERS
              POP     CX
              POP     BX
              POP     AX
              RET                     ;RETURN

XNFIN        ENDP

```

PAGE +
SUBTTL ACKIN

```

;   ACKIN waits for the controller to send ACK (ASCII) to the host computer
;   through the serial communication port. If the controller sends either
;   a EOT (ASCII) or NAK (ASCII), An error signal will be stored in ERROR.
;   Otherwise, ERROR will equal zero upon exit.

;   Input: None.

;   Output: Error contains appropriate error code.

;   Registers Altered: None.

;   Routines Called: COMINCK

```

```

ACKIN                PUBLIC  PROC
                     ACKIN

                     PUSH    AX           ;SAVE REGISTERS
                     PUSH    BX
                     PUSH    CX
                     PUSH    DX

                     MOV     AX,DATAS     ;POINT TO DATA SEGMENT
                     MOV     DS,AX

                     MOV     AX,0         ;MOVE 0 INTO AX
                     MOV     ERROR,AX     ;SET ERROR = 0

ACKN:                CALL    COMINCK      ;CHECK COMMUNICATION LINE
                     JZ      ACKN         ;BUFFER EMPTY, CHECK AGAIN
                     CMP     AL,04H       ;IS CHARACTER = EOT
                     JZ      AERROR      ;IF SO, JUMP TO AERROR
                     CMP     AL,015H     ;IS CHARACTER = NAK
                     JZ      AAERROR     ;IF SO, JUMP TO AAERROR

                     CMP     AL,06H       ;IS CHARACTER = ACK
                     JZ      AEXIT       ;IF SO, EXIT
                     JMP     ACKN        ;IF NOT, CHECK AGAIN

AERROR:             MOV     AX,04H        ;MOVE 04H INTO AX
                     MOV     ERROR,AX     ;SET ERROR = 04H
                     JMP     AEXIT       ;JUMP TO AEXIT

```

```

AAERROR:      MOV      AX,015H      ;MOVE 015H INTO AX
               MOV      ERROR,AX    ;SET ERROR = 015H

AEXIT:        POP      DX           ;RESTORE REGISTERS
               POP      CX
               POP      BX
               POP      AX
               RET                  ;RETURN

ACKIN         ENDP

```

PAGE +
SUBTTL COMINIT

```

;   COMINIT initializes COM0 (serial communication port) with the appropriate
;   configuration for communication with 7545 robot.

;   Input: None.

;   Output: COM0 is initialized with the following configuration:

;           BAUD RATE = 4800
;           PARITY    = EVEN
;           DATA BITS = 7
;           STOP BITS = 2

;   Registers Altered: None.
;
;   Routines Called: BIOS interrupt 14h

```

```

COMINIT      PROC

               PUSH     AX           ;STORE REGISTERS
               PUSH     DX
               PUSH     DS

               MOV      AX,DATAS     ;POINT TO DATA SEGMENT
               MOV      DS,AX

               MOV      AL,0DEH
               MOV      DX,0

               MOV      AH,0         ;INITIALIZE
               INT      14H         ;RS232 BIOS CALL

               POP      DS          ;RESTORE REGISTERS
               POP      DX
               POP      AX
               RET                  ;RETURN

COMINIT      ENDP

```

PAGE +
SUBTTL COMINCK

```

;   COMINCK checks for data in the serial communication port input buffer.

;   Input: None.

```

```

;   Output: If the input buffer is full, then the 8086 flags are set to
;           non-zero, and the input data is stored in AL. If the buffer
;           is empty, then the flags are set for zero status.

```

```

;   Registers Altered: AX

```

```

;   Routines Called: None.

```

```

COMINCK                                PROC
PUBLIC COMINCK

    PUSH    DS                        ;SAVE REGISTERS
    PUSH    DX
    PUSH    SI

    MOV     SI,0                      ;LOOK UP ADDRESS OF COM LINE
    ADD     SI,SI                     ;DOUBLE TO INDEX INTO TABLE
    MOV     DX,40H                    ;SEGMENT OF TABLE
    MOV     DS,DX                     ;SET DATA SEGMENT TO THIS TABLE
    MOV     DX,[SI]                  ;NEW GET IT
    ADD     DX,5                       ;LINE STATUS
    IN      AL,DX                     ;GET IT
    TEST    AL,1                      ;RECEIVE BUFFER FULL?
    JZ      COMINCKEXIT

    MOV     DX,[SI]                   ;DATA REGISTER
    IN      AL,DX                     ;GET IT

COMINCKEXIT:
    POP     SI                        ;RESTORE REGISTERS
    POP     DX
    POP     DS
    RET

COMINCK                                ENDP

```

```

PAGE +
SUBTTL COMOUT

```

```

;   COMOUT transmits the ASCII character stored in AL to the controller
;   through the serial communication port.

;   Input: AL contains the ASCII character to be transmitted.

;   Output: The ASCII character is transmitted through the serial communication
;           port to the controller.

;   Registers Altered: None.

;   Routines Called: BIOS interrupt 14h

```

```

COMOUT                                PROC
PUBLIC COMOUT

    PUSH    AX                        ;SAVE REGISTERS
    PUSH    DX

    MOV     DX,0                      ;SET COMMUNICATION LINE

```

	MOV	AH,1	;SEND IT OUT
	INT	14H	;RS232 BIOS CALL
	POP	DX	;RESTORE REGISTERS
	POP	AX	
	RET		;RETURN
COMOUT	ENDP		
CMMNCTN	ENDS		
	END		

APPENDIX G. VISION HARDWARE INTERFACE SOFTWARE

```

}-----
}
}  SYMBOLS DEFINITION SECTION
}
}-----

PFG_REG EQU      0FF00H      ;CONTROL REGISTER BASE ADDRESS
PFG_SEG EQU      08000H      ;MEMORY SEGMENT BASE ADDRESS

CONL          EQU      0+PFG_REG      ;CONTROL REG ADDR LOW
CONH          EQU      1+PFG_REG      ;CONTROL REG ADDR HIGH
LUTA          EQU      2+PFG_REG      ;LUT ADDRESS REGISTER
LUTD          EQU      3+PFG_REG      ;LUT DATA REGISTER
MASK          EQU      4+PFG_REG      ;MASK REGISTER ADDRESS
FMBS          EQU      5+PFG_REG      ;FRAME MEMORY BLOCK SELECT

}-----
}
}  PARAMETER LIST CONTROL BLOCK
}
}-----

FRAME  STRUC
PAR1      DD      ?      ;FIRST PARAMETER
PAR2      DD      ?      ;SECOND PARAMETER
PAR3      DD      ?      ;THIRD PARAMETER
PAR4      DD      ?      ;FORTH PARAMETER

FRAME  ENDS

}-----
}
}  DATA SEGMENT FOR ALL ROUTINES
}
}-----

DSBLK  SEGMENT 'DATA'
      DB      'PCVISION'
SP_SAVE DW      0      ;LOCATION TO SAVE STACK POINTER
      DD      VINIT      ;LOCATION IF VINIT SUBROUTINE
      DD      0
BX_SAVE DW      0
DX_SAVE DW      0
ES_SAVE DW      0
TH      DW      0      ;THRESHOLD VALUE
M_OFFSET DW      0      ;MEMORY OFFSET FOR FRAME REF
DSBLK  ENDS

}-----
}
}  CODE SEGMENT FOR ALL SUBROUTINES
}
}-----

PCV_PACKAGE      SEGMENT 'CODE'
      ASSUME CS:PCV_PACKAGE,DS:DSBLK
      DW      SEG DSBLK

VINIT  PROC      FAR
      PUBLIC  VINIT

```

```

        MOV     AX,DSBLK                ;SAVE STACK POINTER
        MOV     DS,AX
        MOV     SP_SAVE,SP

        MOV     AL,9H
        MOV     DX,CONL
        OUT     DX,AL
        MOV     AL,0
        MOV     DX,CONH
        OUT     DX,AL

TST_LP: IN     MOV     DX,CONL
              AL,DX
              AND     AL,30H
              JNZ     TST_LP

              LDS     SI,ES:PAR1[BX]    ;SET ERROR RETURN TO 1
              MOV     AX,1
              MOV     [SI],AX

              RET                       ;RETURN TO FORTRAN

VINIT      ENDP

GRAB       PROC     FAR                ;CALL GRAB ENTRY POINT
        PUBLIC  GRAB

        MOV     AX,DSBLK
        MOV     DS,AX
        MOV     SP_SAVE,SP

        MOV     AL,39H                ;SET ACQMOD BITS TO 11B
        MOV     DX,CONL                ;GET I/O ADDR OF REG
        OUT     DX,AL                  ;AND WRITE TO PCV FG

        LDS     SI,ES:PAR1[BX]        ;SET ERROR RETURN TO 1
        MOV     AX,1
        MOV     [SI],AX

        RET                       ;RETURN TO FORTRAN

GRAB       ENDP

FREEZE     PROC     FAR                ;CALL FREEZE ENTRY PT
        PUBLIC  FREEZE

        MOV     AX,DSBLK
        MOV     DS,AX
        MOV     SP_SAVE,SP

        MOV     AL,29H                ;SET ACQMOD BITS TO 10B
        MOV     DX,CONL                ;GET I/O ADDR OF REG
        OUT     DX,AL                  ;AND WRITE TO PCV FG

        LDS     SI,ES:PAR1[BX]        ;SET ERROR RETURN TO 1
        MOV     AX,1
        MOV     [SI],AX

        RET                       ;RETURN TO FORTRAN

FREEZE     ENDP

RAMP       PROC     FAR                ;CALL RAMP ENTRY POINT
        PUBLIC  RAMP

```



```

        MOV     AX,DSBLK           ;SAVE STACK POINTER
        MOV     DS,AX
        MOV     SP_SAVE,SP

        MOV     AX,0               ;CLEAR THRESHOLD VALUE
        MOV     TH,AX

        MOV     CX,256             ;SETUP COUNTER FOR RAMP
        MOV     AL,0

RMP_LP:
        MOV     DX,LUTA           ;GET LUT ADDR PORT #
        OUT     DX,AL             ;AND WRITE VALUE
        MOV     DX,LUTD           ;GET LUT DATA PORT #
        OUT     DX,AL             ;AND WRITE SAME VALUE
        INC     AL                ;INC VALUE FOR RAMP
        LOOP    RMP_LP

        LDS     SI,ES:PAR1[BX]     ;SET ERROR RETURN TO 1
        MOV     AX,1
        MOV     [SI],AX

        RET                       ;FORTRAN RETURN

RAMP
        ENDP

INVRMP  PROC    FAR                ;CALL INVRMP ENTRY PT
        PUBLIC  INVRMP

        MOV     AX,DSBLK           ;SAVE STACK POINTER
        MOV     DS,AX
        MOV     SP_SAVE,SP

        MOV     AX,0               ;CLEAR THRESHOLD VALUE
        MOV     TH,AX

        MOV     CX,256             ;SETUP COUNTER FOR RAMP

INVRMP_LP:
        MOV     DX,LUTA           ;GET LUT ADDR PORT #
        OUT     DX,AL             ;AND WRITE VALUE
        MOV     DX,LUTD           ;GET LUT DATA PORT #
        XCHG    AX,CX
        DEC     AX
        OUT     DX,AL             ;AND WRITE DATA VALUE
        INC     AX
        XCHG    AX,CX
        INC     AL                ;INC VALUE FOR RAMP
        LOOP    INVRMP_LP

        LDS     SI,ES:PAR1[BX]     ;SET ERROR RETURN TO 1
        MOV     AX,1
        MOV     [SI],AX

        RET                       ;FORTRAN RETURN

INVRMP  ENDP

THOLD   PROC    FAR                ;CALL THOLD ENTRY PT
        PUBLIC  THOLD

        MOV     AX,DSBLK           ;SAVE SP,BX AND ES
        MOV     DS,AX
        MOV     SP_SAVE,SP
        MOV     BX_SAVE,BX
        MOV     ES_SAVE,ES

```

```

        LDS     SI,ES:PAR1[BX]          ;GET THRESHOLD VALUE
        MOV     AX,[SI]
        MOV     BX,DSBLK
        MOV     DS,BX
        MOV     TH,AX

        MOV     CX,256                  ;INITIALIZE COUNTER
        MOV     BL,0                    ;SETUP INDEX IN BL

TH_LP_1:
        MOV     DX,LUTA                  ;GET PORT # OF LUT ADDR
        MOV     AL,BL
        OUT     DX,AL                    ;AND WRITE TABLE ADDR
        MOV     DX,LUTD                  ;GET PORT # OF LUT DATA
        MOV     AL,255
        OUT     DX,AL                    ;AND WRITE TABLE DATA
        INC     BL                        ;INCREMENT INDEX IN BL
        LOOP    TH_LP_1                  ;NEXT TABLE ENTRY

        MOV     CX,TH
        MOV     BL,0                    ;NOW SET COUNTER TO TH

TH_LP_2:
        MOV     DX,LUTA                  ;GET PORT # OF LUT ADDR
        MOV     AL,BL
        OUT     DX,AL                    ;AND WRITE TABLE ADDR
        MOV     DX,LUTD                  ;GET PORT # OF LUT DATA
        MOV     AL,0
        OUT     DX,AL                    ;AND WRITE TABLE DATA
        INC     BL                        ;INCREMENT INDEX IN BL
        LOOP    TH_LP_2                  ;NEXT TABLE ENTRY

        MOV     BX,BX_SAVE               ;RESTORE BX
        LDS     SI,ES:PAR2[BX]           ;SET ERROR RETURN TO 1
        MOV     AX,1
        MOV     [SI],AX

        RET                               ;FORTRAN RETURN

THOLD    ENDP

VRDROW   PROC    FAR PUBLIC VRDROW      ;CALL VRDROW ENTRY PT

        MOV     AX,DSBLK                 ;SET UP DATA SEGMENT
        MOV     DS,AX
        MOV     SP_SAVE,SP              ;AND SAVE STACK POINTER

        LDS     SI,ES:PAR1[BX]           ;GET ROW NUMBER (0-479)
        MOV     AX,[SI]

        CMP     AX,255
        JA      VRD_BH                   ;JUMP TO BOTTOM HALF

VRD_TH:
        XCHG    DX,AX                    ;SAVE ROW # IN DX
        MOV     AX,DSBLK                 ;RESTORE DS TO DSBLK
        MOV     DS,AX
        XCHG    AX,DX                    ;RESTORE ROW #

        MOV     CL,8                     ;MULTIPLY BY 256
        SAL     AX,CL
        MOV     M_OFFSET,AX              ;SAVE MEMORY OFFSET VAL

VRD_TH_LEFT:

```

	MOV	AL,0	;SET FRM MEM BLK SEL
	MOV	DX,FMBS	
	OUT	DX,AL	
	MOV	DI,M_OFFSET	;SET UP MEMORY INDEX
	MOV	CX,256	;LOAD CNT FOR LFT HALF
	LDS	SI,ES:PAR2[BX]	;GET ADDR OF ARRAY
	MOV	DX,DS	;SAVE FORT DS IN DX REG
VRD_TH_LP_1:	MOV	AX,PFG_SEG	;SET DS TO FRAME MEM
	MOV	DS,AX	
	MOV	AX,0	;CLEAR AH REG
	MOV	AL,[DI]	;GET PIXEL FROM FRM MEM
	MOV	DS,DX	;RESTORE FORT DS VALUE
	MOV	[SI],AX	;PASS TO ARRAY
	INC	DI	;INCREMENT MEM INDEX
	INC	SI	;INCREMENT ARRAY INDEX
	INC	SI	; TWICE
	LOOP	VRD_TH_LP_1	;DO IT AGAIN
VRD_TH_RIGHT:	MOV	AX,DSBLK	;RESTORE DS TO DSBLK
	MOV	DS,AX	
	MOV	DX_SAVE,DX	;SAVE FORT DS VALUE
	MOV	AL,1	;SET FRM MEM BLK SEL
	MOV	DX,FMBS	
	OUT	DX,AL	
	MOV	DX,DX_SAVE	;RESTORE FORT DS TO DX
	MOV	DI,M_OFFSET	;SET UP MEMORY INDEX
	MOV	CX,256	;LOAD CNT FOR LFT HALF
VRD_TH_LP_2:	MOV	AX,PFG_SEG	;SET DS TO FRAME MEM
	MOV	DS,AX	
	MOV	AX,0	;CLEAR AH REG
	MOV	AL,[DI]	;GET PIXEL FROM FRM MEM
	MOV	DS,DX	;RESTORE FORT DS VALUE
	MOV	[SI],AX	;PASS TO ARRAY
	INC	DI	;INCREMENT MEM INDEX
	INC	SI	;INCREMENT ARRAY INDEX
	INC	SI	; TWICE
	LOOP	VRD_TH_LP_2	
	JMP	VRD_RETURN	
VRD_BH:	SUB	AX,256	;SUB 256 FROM ROW #
	XCHG	DX,AX	;SAVE ROW # IN DX
	MOV	AX,DSBLK	;RESTORE DS TO DSBLK
	MOV	DS,AX	
	XCHG	AX,DX	;RESTORE ROW #
	MOV	CL,8	;MULTIPLY BY 256
	SAL	AX,CL	
	MOV	M_OFFSET,AX	;SAVE MEMORY OFFSET VAL
VRD_BH_LEFT:	MOV	AL,2	;SET FRM MEM BLK SEL
	MOV	DX,FMBS	
	OUT	DX,AL	

	MOV	DI,M_OFFSET	;SET UP MEMORY INDEX
	MOV	CX,256	;LOAD CNT FOR LFT HALF
	LDS	SI,ES:PAR2[BX]	;GET ADDR OF ARRAY
	MOV	DX,DS	;SAVE FORT DS IN DX REG
VRD_BH_LP_1:	MOV	AX,PFG_SEG	;SET DS TO FRAME MEM
	MOV	DS,AX	
	MOV	AX,0	;CLEAR AH REG
	MOV	AL,[DI]	;GET PIXEL FROM FRM MEM
	MOV	DS,DX	;RESTORE FORT DS VALUE
	MOV	[SI],AX	;PASS TO ARRAY
	INC	DI	;INCREMENT MEM INDEX
	INC	SI	;INCREMENT ARRAY INDEX
	INC	SI	; TWICE
	LOOP	VRD_BH_LP_1	;DO IT AGAIN
VRD_BH_RIGHT:	MOV	AX,DSBLK	;RESTORE DS TO DSBLK
	MOV	DS,AX	
	MOV	DX_SAVE,DX	;SAVE FORT DS VALUE
	MOV	AL,3	;SET FRM MEM BLK SEL
	MOV	DX,FMBS	
	OUT	DX,AL	
	MOV	DX,DX_SAVE	;RESTORE FORT DS TO DX
	MOV	DI,M_OFFSET	;SET UP MEMORY INDEX
	MOV	CX,256	;LOAD CNT FOR LFT HALF
VRD_BH_LP_2:	MOV	AX,PFG_SEG	;SET DS TO FRAME MEM
	MOV	DS,AX	
	MOV	AX,0	;CLEAR AH REG
	MOV	AL,[DI]	;GET PIXEL FROM FRM MEM
	MOV	DS,DX	;RESTORE FORT DS VALUE
	MOV	[SI],AX	;PASS TO ARRAY
	INC	DI	;INCREMENT MEM INDEX
	INC	SI	;INCREMENT ARRAY INDEX
	INC	SI	; TWICE
	LOOP	VRD_TH_LP_2	
VRD_RETURN:	LDS	SI,ES:PAR3[BX]	;SET ERROR RETURN TO 1
	MOV	AX,1	
	MOV	[SI],AX	
	RET		
VRDROW	ENDP		
VWRROW	PROC	FAR	;CALL VWRROW ENTRY PT
		PUBLIC	VWRROW
	MOV	AX,DSBLK	;SET UP DATA SEGMENT
	MOV	DS,AX	
	MOV	SP_SAVE,SP	;AND SAVE STACK POINTER
	LDS	SI,ES:PAR1[BX]	;GET ROW NUMBER (0-479)
	MOV	AX,[SI]	
	CMP	AX,255	
	JA	VWR_BH	;JUMP TO BOTTOM HALF

```

VWR_TH:
    XCHG    DX,AX                ;SAVE ROW # IN DX
    MOV     AX,DSBLK            ;RESTORE DS TO DSBLK
    MOV     DS,AX
    XCHG    AX,DX                ;RESTORE ROW #

    MOV     CL,8                 ;MULTIPLY BY 256
    SAL     AX,CL
    MOV     M_OFFSET,AX         ;SAVE MEMORY OFFSET VAL

VWR_TH_LEFT:
    MOV     AL,0                 ;SET FRM MEM BLK SEL
    MOV     DX,FMBS
    OUT     DX,AL

    MOV     DI,M_OFFSET          ;SET UP MEMORY INDEX
    MOV     CX,256               ;LOAD CNT FOR LFT HALF

    LDS     SI,ES:PAR2[BX]       ;GET ADDR OF ARRAY
    MOV     DX,DS                ;SAVE FORT DS IN DX REG

VWR_TH_LP_1:
    MOV     DS,DX                ;RESTORE FORT DS VALUE
    MOV     AX,[SI]              ;GET VAL FROM ARRAY
    MOV     BP,AX
    MOV     AX,PFG_SEG           ;SET DS TO FRAME MEM
    MOV     DS,AX
    MOV     AX,BP
    MOV     [DI],AL              ;PUT PIX INTO FRAME MEM

    INC     DI                   ;INCREMENT MEM INDEX
    INC     SI                   ;INCREMENT ARRAY INDEX
    INC     SI                   ; TWICE
    LOOP    VWR_TH_LP_1          ;DO IT AGAIN

VWR_TH_RIGHT:
    MOV     AX,DSBLK            ;RESTORE DS TO DSBLK
    MOV     DS,AX
    MOV     DX_SAVE,DX           ;SAVE FORT DS VALUE
    MOV     AL,1                 ;SET FRM MEM BLK SEL
    MOV     DX,FMBS
    OUT     DX,AL
    MOV     DX,DX_SAVE           ;RESTORE FORT DS TO DX

    MOV     DI,M_OFFSET          ;SET UP MEMORY INDEX
    MOV     CX,256               ;LOAD CNT FOR LFT HALF

VWR_TH_LP_2:
    MOV     DS,DX                ;RESTORE FORT DS VALUE
    MOV     AX,[SI]              ;GET VALUE FROM ARRAY
    MOV     BP,AX
    MOV     AX,PFG_SEG           ;SET DS TO FRAME MEM
    MOV     DS,AX
    MOV     AX,BP
    MOV     [DI],AL              ;PUT PX INTO FRAME MEM

    INC     DI                   ;INCREMENT MEM INDEX
    INC     SI                   ;INCREMENT ARRAY INDEX
    INC     SI                   ; TWICE
    LOOP    VWR_TH_LP_2          ;DO IT AGAIN

    JMP     VWR_RETURN

VWR_BH:
    SUB     AX,256                ;SUB 256 FROM ROW #
    XCHG    DX,AX                ;SAVE ROW # IN DX

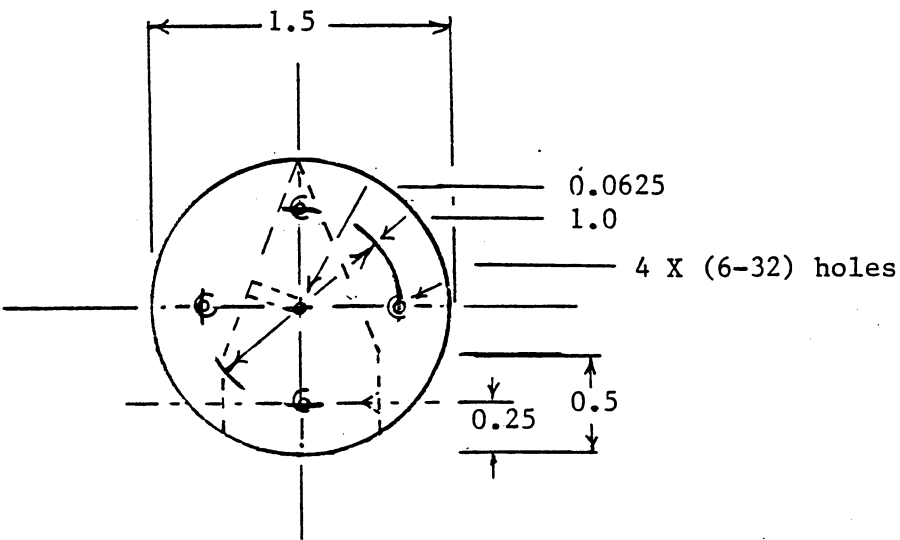
```

	MOV	AX,DSBLK	;RESTORE DS TO DSBLK
	MOV	DS,AX	
	XCHG	AX,DX	;RESTORE ROW #
	MOV	CL,8	;MULTIPLY BY 256
	SAL	AX,CL	
	MOV	M_OFFSET,AX	;SAVE MEMORY OFFSET VAL
VWR_BH_LEFT:			
	MOV	AL,2	;SET FRM MEM BLK SEL
	MOV	DX,FMBS	
	OUT	DX,AL	
	MOV	DI,M_OFFSET	;SET UP MEMORY INDEX
	MOV	CX,256	;LOAD CNT FOR LFT HALF
	LDS	SI,ES:PAR2[BX]	;GET ADDR OF ARRAY
	MOV	DX,DS	;SAVE FORT DS IN DX REG
VWR_BH_LP_1:			
	MOV	DS,DX	;RESTORE FORT DS VALUE
	MOV	AX,[SI]	;GET VAL FROM ARRAY
	MOV	BP,AX	
	MOV	AX,PFG_SEG	;SET DS TO FRAME MEM
	MOV	DS,AX	
	MOV	AX,BP	
	MOV	[DI],AL	;PUT PIX INTO FRAME MEM
	INC	DI	;INCREMENT MEM INDEX
	INC	SI	;INCREMENT ARRAY INDEX
	INC	SI	; TWICE
	LOOP	VWR_BH_LP_1	;DO IT AGAIN
VWR_BH_RIGHT:			
	MOV	AX,DSBLK	;RESTORE DS TO DSBLK
	MOV	DS,AX	
	MOV	DX_SAVE,DX	;SAVE FORT DS VALUE
	MOV	AL,3	;SET FRM MEM BLK SEL
	MOV	DX,FMBS	
	OUT	DX,AL	
	MOV	DX,DX_SAVE	;RESTORE FORT DS TO DX
	MOV	DI,M_OFFSET	;SET UP MEMORY INDEX
	MOV	CX,256	;LOAD CNT FOR LFT HALF
VWR_BH_LP_2:			
	MOV	DS,DX	;RESTORE FORT DS VALUE
	MOV	AX,[SI]	;GET VAL FROM ARRAY
	MOV	BP,AX	
	MOV	AX,PFG_SEG	;SET DS TO FRAME MEM
	MOV	DS,AX	
	MOV	AX,BP	
	MOV	[DI],AL	;PUT PIX INTO FRAME MEM
	INC	DI	;INCREMENT MEM INDEX
	INC	SI	;INCREMENT ARRAY INDEX
	INC	SI	; TWICE
	LOOP	VWR_TH_LP_2	
VWR_RETURN:			
	LDS	SI,ES:PAR3[BX]	;SET ERROR RETURN TO 1
	MOV	AX,1	
	MOV	[SI],AX	
	RET		

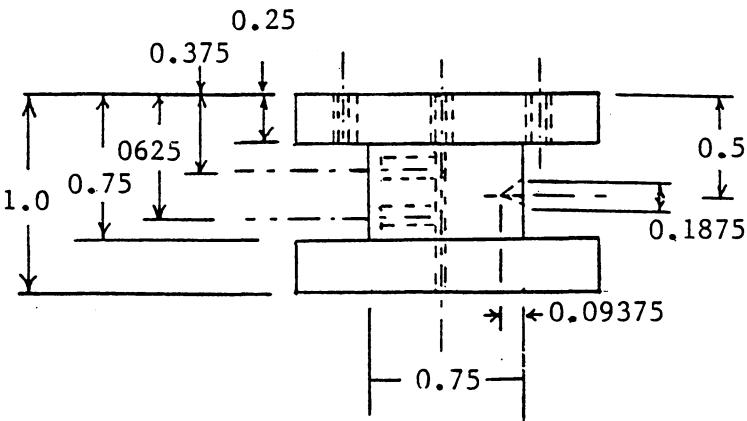
```
VHRRON  ENDP
PCV_PACKAGE  ENDS
END
```

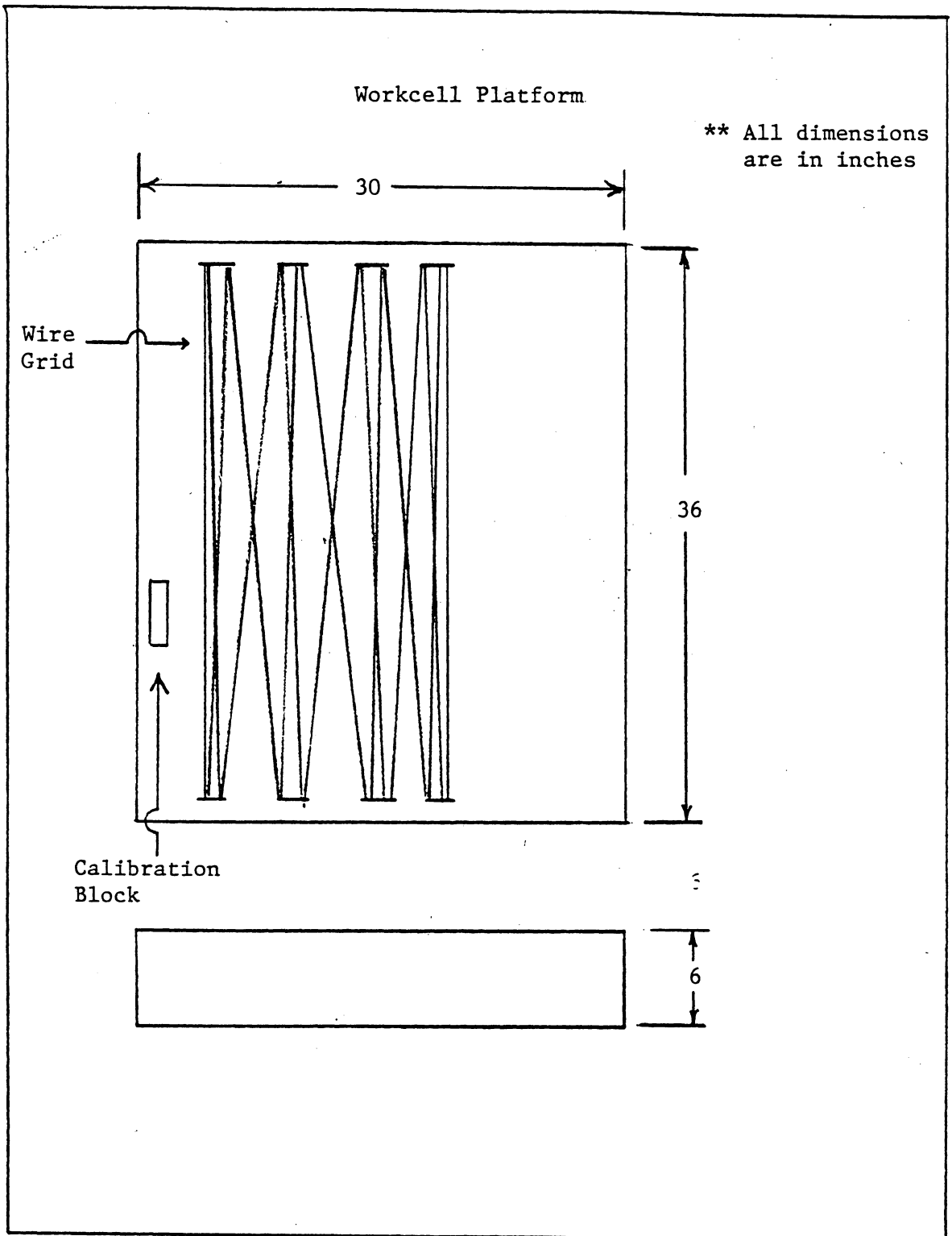
APPENDIX H. TACTILE SYSTEM HARDWARE

Face Plate	Edward C. DeMeter	Quantity: 1
Date: 4/14/86	Mat: Aluminum	Scale: 1:1



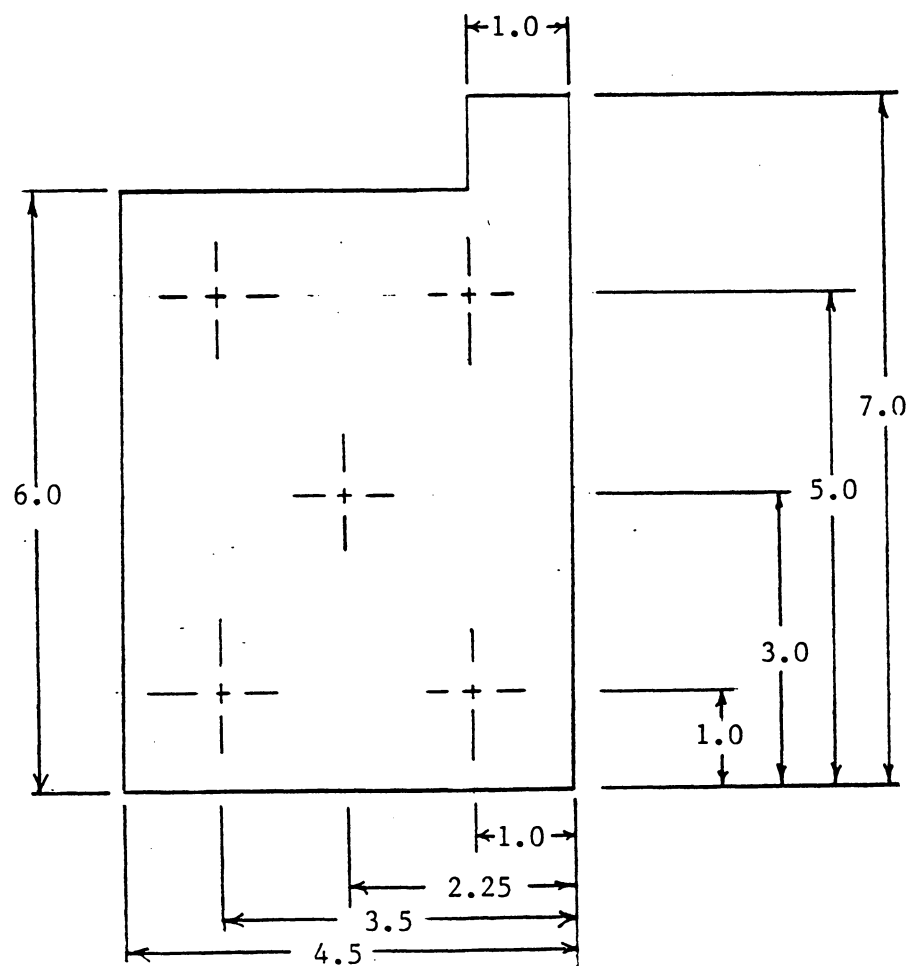
** All dimensions are in inches

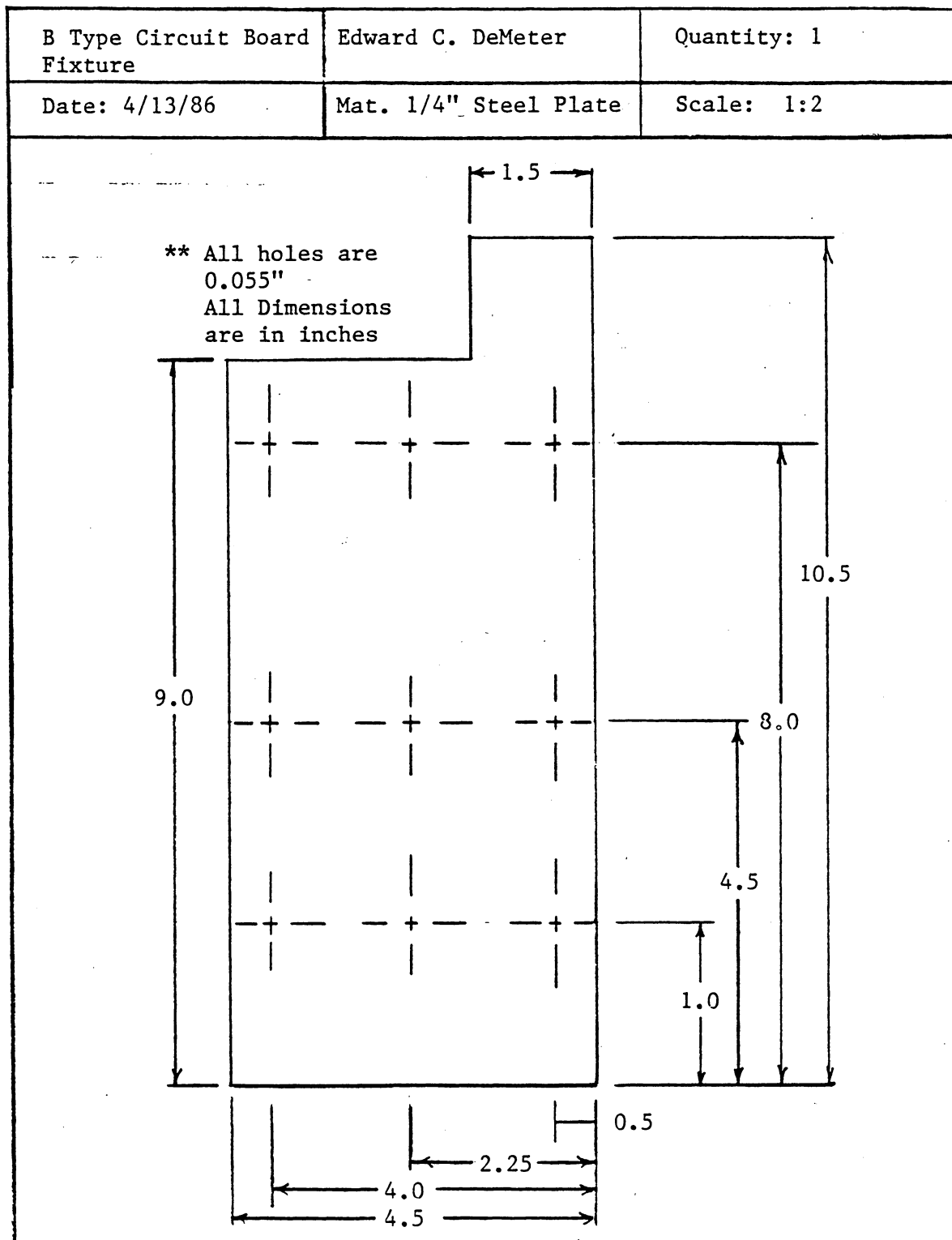




A Type Circuit Board Fixture	Edward C. DeMeter	Quantity: 1
Date: 4/13/86	Mat. 1/4" Steel Plate	Scale: 1:2

** All holes are 0.055"
All dimensions are in inches





APPENDIX I. SYSTEM DATA

Tactile System Repeatability Data

<u>Run</u>	<u>X Coord. (inches)</u>	<u>Y Coord. (inches)</u>
1	-8.448	17.717
2	-8.452	17.717
3	-8.450	17.717
4	-8.448	17.717
5	-8.452	17.717
6	-8.444	17.717
7	-8.446	17.717
8	-8.454	17.717
9	-8.450	17.717
10	-8.448	17.717

Average Coordinate = (-8.449, 17.717)
Standard Deviation = 0.003 inches
of Distance from
Average Coordinate

Insertion Point Error Data

<u>Point</u>	<u>Points/Side</u>	<u>Pos. #</u>	<u>Coord. Type</u>	<u>Angle (Deg.)</u>	<u>X Coord. (inches)</u>	<u>Y Coord. (inches)</u>
1	2	1	True	88.47	-14.419	4.215
1	2	1	Ins.	88.47	-14.419	4.212
3	2	1	True	88.47	-20.811	7.116
3	2	1	Ins.	88.47	-20.778	7.114
7	2	1	True	88.47	-12.970	7.396
7	2	1	Ins.	88.47	-12.967	7.388
9	2	1	True	88.47	-19.356	10.317
9	2	1	Ins.	88.47	-19.262	10.293
1	2	1	True	89.68	-14.413	4.209
1	2	1	Ins.	89.68	-14.413	4.203
3	2	1	True	89.68	-20.811	7.128
3	2	1	Ins.	89.68	-20.778	7.110
7	2	1	True	89.68	-12.963	7.398
7	2	1	Ins.	89.68	-12.969	7.388
9	2	1	True	89.68	-19.358	10.309
9	2	1	Ins.	89.68	-19.293	10.272
1	2	2	True	89.70	-12.201	8.031
1	2	2	Ins.	89.70	-12.195	8.020
3	2	2	True	89.70	-18.585	5.843
3	2	2	Ins.	89.70	-18.569	5.837
7	2	2	True	89.70	-13.195	11.535
7	2	2	Ins.	89.70	-13.183	11.490
9	2	2	True	89.70	-19.783	9.146
9	2	2	Ins.	89.70	-19.730	9.163
1	2	2	True	89.83	-12.201	8.031
1	2	2	Ins.	89.83	-12.201	8.017
3	2	2	True	89.83	-18.585	5.843
3	2	2	Ins.	89.83	-18.567	5.835

Insertion Point Error Data (cont'd)

<u>Point</u>	<u>Points/Side</u>	<u>Pos. #</u>	<u>Coord. Type</u>	<u>Angle (Deg.)</u>	<u>X Coord. (inches)</u>	<u>Y Coord. (inches)</u>
7	2	2	True	89.83	-13.195	11.535
7	2	2	Ins.	89.83	-13.195	11.499
9	2	2	True	89.83	-19.783	9.146
9	2	2	Ins.	89.83	-19.724	9.161
1	2	3	True	90.19	-12.996	12.697
1	2	3	Ins.	90.19	-13.004	12.687
3	2	3	True	90.19	-16.183	6.421
3	2	3	Ins.	90.19	-16.169	6.455
7	2	3	True	90.19	-16.116	14.281
7	2	3	Ins.	90.19	-16.112	14.280
9	2	3	True	90.19	-19.297	8.026
9	2	3	Ins.	90.19	-19.254	8.089
1	2	4	True	90.29	-12.283	11.297
1	2	4	Ins.	90.29	-12.297	11.289
3	2	4	True	90.29	-16.929	6.022
3	2	4	Ins.	90.29	-16.927	6.026
7	2	4	True	90.29	-14.913	13.620
7	2	4	Ins.	90.29	-14.909	13.613
9	2	4	True	90.29	-19.559	8.364
9	2	4	Ins.	90.29	-19.510	8.415
1	3	2	True	90.05	-11.998	8.254
1	3	2	Ins.	90.05	-11.998	8.256
3	3	2	True	90.05	-18.581	5.848
3	3	2	Ins.	90.05	-18.567	5.841
7	3	2	True	90.05	-13.207	11.551
7	3	2	Ins.	90.05	-13.191	11.543
9	3	2	True	90.05	-19.774	9.142
9	3	2	Ins.	90.05	-19.732	9.140

Insertion Point Error Data (cont'd)

<u>Point</u>	<u>Points/Side</u>	<u>Pos. #</u>	<u>Coord. Type</u>	<u>Angle (Deg.)</u>	<u>X Coord. (inches)</u>	<u>Y Coord. (inches)</u>
1	3	2	True	89.96	-11.998	8.254
1	3	2	Ins.	89.96	-11.996	8.256
3	3	2	True	89.96	-18.581	5.848
3	3	2	Ins.	89.96	-18.565	5.843
7	3	2	True	89.96	-13.207	11.551
7	3	2	Ins.	89.96	-13.193	11.541
9	3	2	True	89.96	-19.774	9.142
9	3	2	Ins.	89.96	-19.735	9.143
1	3	1	True	89.91	-14.414	4.210
1	3	1	Ins.	89.91	-14.413	4.210
3	3	1	True	89.91	-20.814	7.132
3	3	1	Ins.	89.91	-20.796	7.105
7	3	1	True	89.91	-12.966	7.401
7	3	1	Ins.	89.91	-12.972	7.391
9	3	1	True	89.91	-19.361	10.312
9	3	1	Ins.	89.91	-19.311	10.278
1	3	1	True	89.89	-14.413	4.210
1	3	1	Ins.	89.89	-14.410	4.204
3	3	1	True	89.89	-20.810	7.128
3	3	1	Ins.	89.89	-20.792	7.101
7	3	1	True	89.89	-12.965	7.400
7	3	1	True	89.89	-12.970	7.392
9	3	1	True	89.89	-19.359	10.310
9	3	1	Ins.	89.89	-19.311	10.276

Insertion Error Vectors

<u>Points/Side</u>	<u>Pos. #</u>	<u>Point 1</u>		<u>Point 3</u>		<u>Point 7</u>		<u>Point 9</u>	
		<u>Mag</u>	<u>Theta</u>	<u>Mag</u>	<u>Theta</u>	<u>Mag</u>	<u>Theta</u>	<u>Mag</u>	<u>Theta</u>
2	1	0.0026	270	0.0331	357	0.00854	357	0.0971	346
2	1	0.0060	270	0.0376	331	0.0117	239	0.1200	337
2	2	0.0125	299	0.0171	339	0.0466	285	0.0556	180
2	2	0.0140	270	0.0199	336	0.036	270	0.0608	14
2	3	0.0128	231	0.0368	68	0.00412	346	0.0763	56
2	4	0.0161	210	0.00447	64	0.00806	300	0.0707	46
3	2	0.00447	27	0.0157	333	0.0179	333	0.042	357
3	2	0.00283	45	0.0167	343	0.0172	324	0.039	2
3	1	0.00510	281	0.0329	332	0.0167	239	0.0605	326
3	1	0.00671	297	0.0329	332	0.00940	238	0.0588	325

* Magnitude values are in inches
 Theta values are in degrees

Statistics of Error Vectors

<u>Point</u>	<u>Points/Side</u>	<u>Positions</u>	<u>Average Mag.</u>	<u>STD Mag.</u>
1	2	1,2,3,4	0.00878	0.00539
1	2	1,2	0.01070	0.00520
1	3	1,2	0.00478	0.00161
3	2	1,2,3,4	0.02690	0.00997
3	2	1,2	0.02480	0.01320
3	3	1,2	0.02460	0.01540
7	2	1,2,3,4	0.02570	0.01856
7	2	1,2	0.01920	0.01760
7	3	1,2	0.01530	0.00390
9	2	1,2,3,4	0.0588	0.08340
9	2	1,2	0.0800	0.02430
9	3	1,2	0.0500	0.01150

* All values are in inches