Lehigh University
Lehigh Preserve

Theses and Dissertations

1985

Design of an interactive computer graphics simulator of VAL-II, the programming language of Unimation's PUMA robot /

Seied Abrishamchian Langrudi
Lehigh University

Follow this and additional works at: https://preserve.lehigh.edu/etd

 Part of the Mechanical Engineering Commons

Recommended Citation
Langrudi, Seied Abrishamchian, "Design of an interactive computer graphics simulator of VAL-II, the programming language of Unimation's PUMA robot /" (1985). *Theses and Dissertations*. 4594.
https://preserve.lehigh.edu/etd/4594

This Thesis is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

DESIGN OF AN INTERACTIVE COMPUTER GRAPHICS SIMULATOR OF VAL-II, THE PROGRAMMING LANGUAGE OF UNIMATION'S PUMA ROBOT

BY

SEIED ABRISHAMCHIAN LANGRUDI

A thesis

presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Mechanical Engineering

Lehigh University

1985

CERTIFICATE OF APPROVAL

This thesis is accepted and approved

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Mechanical Engineering

12/16/85
date

John D Ochs
Professor in charge

F. Ecdogan
Chairman of Department

ACKNOWLEDGEMENTS

I wish to give sincere thanks to Dr. John Ochs for his support and supervision throughout the course of this work.

I would also like to thank Dr. Tulga Ozsoy for his valuable suggestions and technical advise.

The continuous encouragement extended to me by my parents during my graduate syudies is especially appreciated.

Finally, I thank Greg Loney, Drew Landman and Jack Wentz who made my stay at Lehigh more enjoyable.

TABLE OF CONTENTS

CHAPTER 6

CHPATER 7

CHPATER 8

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Industrial Robots have gained wide acceptance in industry due to their flexibilty and reliability ass well as their ever increasing ease of teaching and programming. The key factor for their further acceptance in contemporary programmable automation is the improvements in the capabilities and efficiency of robot language. Language simulators such as VAL-II SIMULATOR for Unimation's PUMA are valuable tools for teaching, developing and testing robot control programs.

VAL-II SIMULATOR is a real-time, interactive, computer graphics package which simulates the off-line programming language, VAL-II. It employs a Lehigh developed graphics package to provide display of the robot behavior upon execution of key VAL-II commands. The simulator is menu-driven with an on-line help feature for all control levels. Information regarding robot configurations is available to the user at any time. Arithmetic and logical expressions are also available for decision making. Three levels of collision detection are available to the user and the simulator has a simple sensor interface. User-defined geometries may be interactively retrieved from other Computer-Aided Design (CAD) data-bases using the International Graphics Exchange Standard (IGES) and then

easily repositioned in the PUMA's work environment through a manipulation feature in the simulator. The simulator can be used to design and experiment with a variety of setups, investigate assembly tasks and develop various programming possibilities. Once a work-cell is created and the VAL-II program is developed and tested with the simulator, the program may be down-loaded to the PUMA controller for final testing and verification.

CHAPTER 1

1. INTRODUCTION

Due to great technological advancements in recent years, industrial robots have become an integral part of flexible automation with some thousands of robots at work in the United States. Robots now perform many of the tasks formerly done by humans. Robots are able to work longer, handle heavier payloads, do repetitive tasks, and operate under conditions that are considered hazardous to human health. During the past decade automation equipment has become both more complex and costly to put in place. The ability to simulate the selection, installation and operation of this equipment through the use of software offers great potential for time-saving and added the assurance of maximum efficiency in manufacturing operations.

The ability to use an off-line language to program a robot externaliy, without tying the robot from the production-line, or using a "spare" robot, indeed contributes to both safety and efficiency.

Due to their ease of programming, their ever increasing flexibility and reliability, robots have gained acceptance by industry. However, a key factor to their

further acceptance as a programmable device is the availability of off-line robot languages and this means to learn these languages quickly and to develope and test reliable application programs.

Since robots are computer-controlled, they are only as smart as the person programming them. Thus, the method of programming is a source of error in developing flexible manufacturing applications. Off-line programming may hold the answer to many programming difficulties. Off-line programming is the programming method by which a robot is programmed via a language. This language has certain vocabulary, grammer and symbols which can be checked by the use of a so called compiler or interpreter. Once the off-line program is checked by the compiler it can be loaded into an existing robot to produce the programmed motions. By using off-line programming, the time dedicated to program a robot is more effectively spent, because robot program logic is more likely to be correct, since it is implemented off-line, not under the pressure of a downed production line. Therefore, the programmer has the time to "walk through" the program, checking for errors in the logic. Also, the work area may be more suitable to the programming task, and an individual has ready access to resources.

Many robot control languages have been developed by robot builders and research laboratories to perform complex

assembly and machine loading tasks. Some of the more important ones are: AL (Stanford University), AML (IBM Corporation), HELP (General Electric Corporation), JARS (Jet Propulsion Laboratory), MCL (McDonnel Douglas Corporation), RAIL (Automatix Inc.), RPL (SRI International), and VAL (Unimation Inc.).

An important aspect in the development of robot programming languages is the use of Dynamic-Interactive computer-graphics. Computer graphics is one method which has proven effective for evaluating manufacturing systems, and determining the relative merits, efficiency and effectiveness of manufacturing systems design. Simulation packages can aid an engineer in testing and designing various work-cell layouts, and studying motion and dynamic characteristics of industrial robots. The use of these special simulation packages provides significant time-saving in the layout and modeling of robot work-cell components and confirming that the final installation will perform as intended. Graphics simulators can also be utilized as an instructional and training tool to give a better insight into robot kinematics and dynamics in a three dimensional environment.

Various robot simulation programs that employ Interactive Computer Graphics have been developed in universities, research institutions, industrial laboratories, and CAD/CAM houses [1-8]. Most of them are

based on Wire-frame modeling for object presentation. Examples include VAL SIMULATOR developed by Clifton and Ochs [3], GRASP by Derby [4], PLACE and ANIMATE by Kretch [5]. Some are based on solid modeling techniques. For example, EMULA developed by Meyer [6], was based on GDP [9], a solid modeler representing objects using polyhedral approximations, and the work by Soroka [7] was based on generalized cones for describing 3D objects. A survey on robot simulation can be found in reference [10].

1.1 PROBLEM STATEMENT

The general thrust for all these efforts is to develope an inexpensive, user friendly, interactive, computer-graphics simulator of robotic work-cells, robots, and their off-line control languages which would feature the following:

1) Create suitable design tools for robot programming.

2) Help the robot user in developing and evaluating program sequences.

3) Be an instructional instrument for learning the off-line programming language.

4) Help the robot user obtain a better insight into the mathematical description of robot kinematics.

5) Provide a linkage with other CAD data-bases to allow

part transfer and work-cell component repositioning.

6) Evaluate complete manufacturing work-cells.

7) Evaluate mechanical systems for ease of assembly.

1.2 PROJECT HISTORY

To develop a general purpose graphics simulator capable of featuring all of the above objectives is a monumental task and certainly beyond the scope of one master's thesis.

Initially, under the supervision of the Lehigh University CAD/CAM program an interactive computer-graphics simulator of the VAL language [11] was developed. VAL is an off-line programming language for Unimation's PUMA robots. VAL and PUMA were chosen for their fundamental representative attributes. VAL's Englishlike mnemonics and elementary structure made it a versatile, commercially available, off-line programming language and a strong candidate for a graphical simulator. The PUMA 600 (Figure 1.1) is an anthropomorphic robot with six rotational joints capable of joint-interpolated and straight-line motions. VAL SIMULATOR [12] uses a wire-frame or edge representation scheme to graphically display the kinematic behavior of PUMA robot upon execution of key VAL commands. Algorithms for joint-coordinated and straight-line motions, and collision detection were developed.

1.3 APPROACH TO PROBLEM

Although VAL SIMULATOR [12] offered some fundamental features, it lacked the flexibility and effectiveness a simulation package must retain. A major improvement was necessary to make the package as versatile and user-friendly as possible. Arithmetic and logical expressions needed to be simulated in order to adequately increase the level of programming control, which would then be enhanced by enabling the operator to call user-defined subroutines. The limitation on the number of workpieces to be used needed to be relaxed, since the latter had introduced major difficulties in that only simple tasks could have been simulated. The linkage with other CAD systems had to be added, which would in turn confine the user in generating various work-cell components. While using VAL SIMULATOR the simulator had to be exited to create or modify user-generated robot programs. This would result in the loss of the location data and setups.

In 1984 Unimation Introduced VAL-II [14], a new robot control system and programming language. In addition to its fundamental programming and control features of VAL, it also includes: network communication capability which enables a remote computer to totally supervise the operation of robot systems, computational and logical statements like those found in high-level computer

languages, a general method including sensory information, real-time path control and concurrent process control.

The VAL-II SIMULATOR is written in FORTRAN 77 and consists of a series of inter-related subroutines. This approach to the organization of the software allows for easy modifications and future expansions of any individual routines without involving major changes in the other parts of the software.

VAL-II SIMULATOR was developed in the Computer-Aided Design Laboratory of Lehigh University. It employs an internally developed graphics package GRAPH3D.LU [15], to graphically display the model and robot work-cell components. It currently runs on DEC VAX 11/780 minicomputer with VMS operating system and VS11 graphics terminals. The VS11 is a relatively inexpensive color raster display that features direct memory access capability and a dual buffered memory option to allow smooth motion of the model. It retails for approximately $15,000 as opposed to Vector Generators upon which commercially available simulators run, which sell for over $70,000.

VAL-II SIMULATOR is an Interactive Computer-Graphics Simulator of one robot and one language, written in a way that allows for future expansion in order to simulate other robots and languages. The simulator is a very effective instrument for teaching various functions of the language

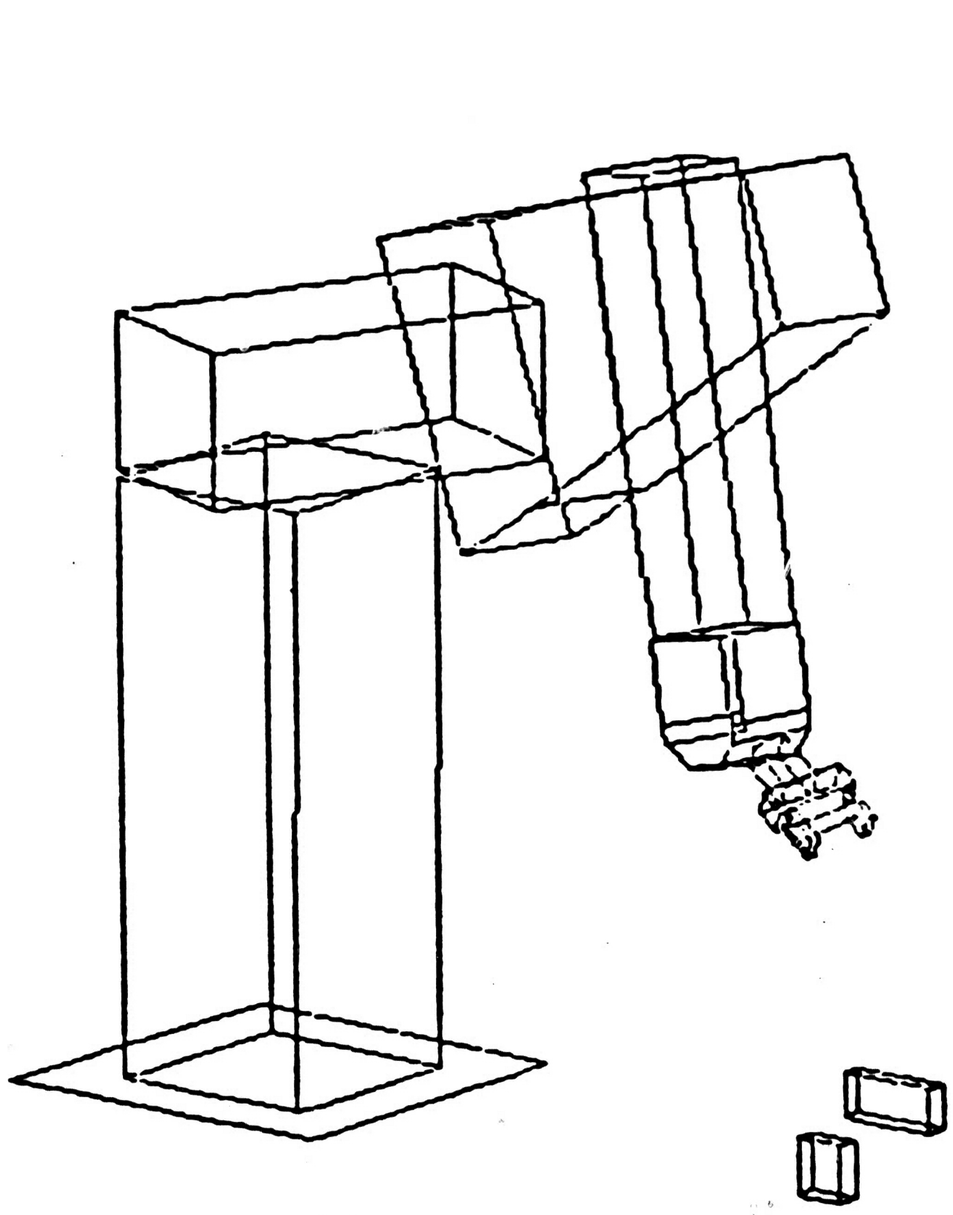


Figure 1-1. Unimation's PUMA 600 robot and simulator model in the same configuration.

to a robot user. It uses wire-frame presentation to display the kinematic behavior of a PUMA robot (Figure 1-1). All data is stored in polyhedron form with appropriate topology. The program is menu-driven and features on-line HELP at all control levels. Moreover, information regarding the Configuration Indicators is available at any time. With the simulator a user is able to design and evaluate a variety of work-cell setups, locations and program possibilities. If needed, user created geometries may be retrieved from other CAD data-bases using Initial Graphics Exchange Standard (IGES) conversions or through the interface with program POLYGON [19].Then, any individual part may interactively be repositioned inside the work environment, with a constant readout of the positional data of the part under question plus the part number. Arithmetic and logical expressions, like those found in high-level computer languages are available for decision making, performing repeating tasks, making branching or jumps, as well as some limited sensory interfacing. Also collisions between the robot and the workpieces are detected automatically. Once program and locations have been verified on the simulator they may be down-loaded to the PUMA controller for final testing and implementation.

1.4 ORGANIZATION OF THESIS

The thesis presented here, provides a thorough treatise of VAL-II SIMULATOR. It describes clearly the fundamental concepts and algorithms, the code for each simulated command, and how to use the simulator. For a more effective approach, figures, tables and examples are presented throughout the work. Chapter 2 includes model and workpiece representation, and a description of the interactive features. These features include information on hidden-surface removal, robot kinematics, transformation of user-defined objects, simulation speed, and collision detection. Chapter 3 provides a thorough description of the feature which graphically displays the functions of a "Teach Pendant" for the PUMA robot. VAL-II SIMULTAOR provides the linkage with other CAD systems. Once the parts are transfered to the VAL-II environment, they may interactively be repositioned within the robot work-cell. In the simulator this is done through the so called MANIPULATION MODE, which is fully described in Chapter 4. Chapter 5 provides a complete description of VAL-II SIMULATOR "EDIT MODE", which covers all the functions related to this mode. Chapter 6 considers each VAL-II command that is simulated. It describes how each command is decoded and then implemented. A brief description of key VAL-II commands is also presented. The Case Study is

presented in Chapter 7, followed the by Conclusion, Chapter 8, which includes a discussion of the limitations of the simulator and recommendations for further study. Programs use flow charts, and flow diagrams for key subroutines are provided. At the end and in the appendices a User's Manual with some examples appears.

CHAPTER 2

2. FEATURES OF THE SYSTEM

2.1 MODEL AND WORKPIECE REPRESENTATION

A language simulator which uses computer-graphic representation of the robot and its surroundings is valuable to the robot designer and end user in many applications. Thses may include: 1) Performance Evaluation, 2) Robotics Training, 3) Work-Cell Layout, 4) Assembly of Parts, etc.

Geometric representation of robot related data may be in the form of a two-dimensional (planar) figure, a three-dimensional edge representation (wire-frame), or a color shaded representation. The storage of the geometric data is key to the ease in which various representations can be generated. This is particularly true when real-time interactive dynamic graphics is required. At the same time collision detection and realistic representation require that the geometric data contain more than just edge information. (See [16] for a collection of papers on the subject and [17] for text.)

The apparent motion in a graphics simulator is achieved by transforming the present locations to desired ones, quickly erasing the screen, updating the display data, and

displaying the entities. The process of entity transformation is a time consuming one. Obviously, there is a direct relationship between the number of entities to be transformed and the time required to perform the entity transformation and the speed of the resulting display of motion.

Two dimensional stick figures are easy to draw, and quick to transform, but provide very limited visualization. Three dimensional colored solid models are desirable, because the display is clear and collisions may be detected visually. Then resulting motion is very slow however, since each solid model requires several seconds to draw, unless expensive, dedicated processors are used. For the simulator the various criteria applied to evaluate the several representations were:

* Amount of storage space needed
* Ease of transformation
* Cost of the equipment
* Display smoothness
* Level of collision detection

As a result, it was decided to utilize extruded wire-frame convex polyhedrons (Figure 2-1). By definition, a polyhedron is a volume completely enclosed by polygons [18]. The polygons are generated by simply connecting the corresponding vertices. In the simulator, the PUMA robot is represented by a collection of a series of polyhedrons,

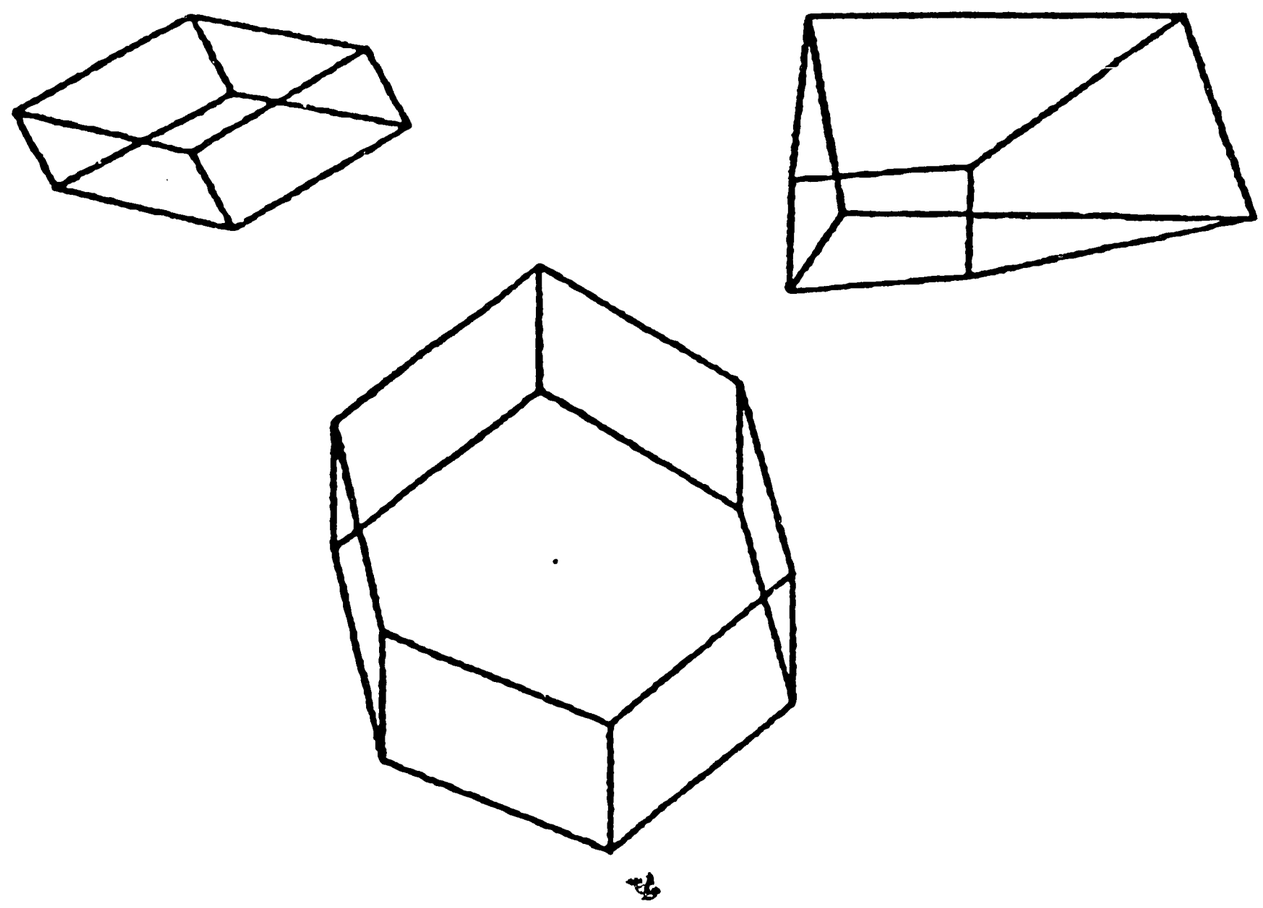


FIGURE 2-1. EXAMPLES OF WIRE-FRAME EXTRUDED CONVEX POLYHEDRONS

connected at specific locations to define revolute joints. The robot work-cell may also contain other sets of polyhedrons to represent the surrounding work environment.

The advantages of this approach may be summarized as follows:

1) Hidden-faces may be eliminated by simple and quick computations

2) Since surfaces are defined, interferences may easily be checked

3) Skewed and tapered polyhedrons may be drawn

4) Most objects may be approximated by circumscribed parallelepipeds, so only the coordinates of eight vertices need to be specified

Another element which plays a vital role in a graphics representation is the type of display terminal to be used. Expensive vector generator terminals with dedicated processors are available for animation, but are unnecessary for language simulators. Storage tubes are attractive due to their low cost, but are unsuitable for animation purposes, since even the simplest geometries require several seconds to draw. Raster scan terminals are the obvious compromise, since, they are relatively inexpensive and provide fast and clear visualization.

2.2 INTERACTIVE FEATURES

VAL-II SIMULATOR was developed to enable the user to position cell components, display the motion of various robot components, check robot reach limits, simulate moving objects, detect collisions, and other functions, all through an interactive, and user-friendly graphics package. The simulator is menu-driven, which means that a list of options are available at all control levels. The options are convenient and easy to use. The user is often required to do no more than press a single key to enter a command.

Upon initialization of a session, a user is prompted whether it is required to read an IGES file, or a UNIGRAPHICS-II (UGII) file, or an existing setup file. It is often the case that cell components are created in a different CAD system and the user would like to transfer the parts to VAL-II SIMULATOR environment for use. If this is the case, and a part file has already been converted to IGES format, the user may interactively transfer the part and position it inside the robot work-cell. Using the IGES output from any three-dimensional CAD system the part data is transfered to another program called POLYGON.LU which converts the three-dimensional wire-frame to a boundary representation polyhedron form [19]. This file can then be read into the VAL-II SIMUALTOR and parts be positioned interactively in the work-cell. A work file for a given

MAIN MENU

ENTER SUBMENUS BY PRESSING:

K_EYBOARD	:	TO KEYBOARD (MONITOR) MODE
M_ANIPULATION	:	TO MANIPULATE WORK PIECE LOCATIONS FROM THE KEYBOARD
P_ENDANT	:	TO PENDANT MODE
E_DITOR	:	TO EDIT MODE
V_AL-II	:	TO DISPLAY A LIST OF EXECUTABLE VAL-II COMMANDS

OR PRESS:

R_ETURN	:	TO RETURN TO PICTURE OF MODEL AND VICE VERSA
B_EGIN	:	TO START OVER
I_NITIALIZE	:	TO INITIALIZE WITH SAME SETUP FILE
S_TOP	:	TO EXIT PROGRAM AND CLEAR SCREEN
H_ARDCOPY	:	TO CREATE A PRINTOUT FILE
G_ENERATE	:	TO GENERATE POLYGON INPUT FILES

Figure 2-2. Main Menu

cell can then be manipulated and saved. The operator may also wish to read an existing setup file. In any case, the specific file will be opened and read, and user-defined objects displayed. The user is then directed to the MAIN MENU (Figure 2-2). At this point a user has several easy options from which to select. A user may select to enter the KEYBOARD Mode (also called Monitor Mode, Figure 2-3) MANIPULATION Mode, PENDANT Mode, EDIT Mode, or VAL-II Mode. While in the MAIN MENU, a user may reinitialize the work-cell setup, or begin with a new setup file. Also, one is able to get hardcopies of the cell layout. While in this mode, a user may display, for convenient reference, a list of VAL-II commands and switch between this list, the model, and the menu. In the Main Menu, files may also be created automatically to be used as input to POLYGON.LU [19] to generating color shaded pictures (Figure 2-4). Whenever an error is encountered an informative warning message will be displayed. Other interactive features include: Hidden-surface removal Collision Detection, and directory information, which will be discussed later in this chapter.

2.3 HIDDEN-SURFACE REMOVAL

In the raster scan terminals the screen is redrawn many times each second. So, as the number of entities to draw decreases, the smoothness of the display is enhanced.

KEYBOARD MENU

ENTER COMMANDS BY PRESSING:

L_IST - THEN TYPE LISTP PROG. NAME: VAL-II FILE LISTING
L_IST - THEN TYPE LISTF: VAL-II PROGRAMS LIST
L_IST - THEN TYPE LISTL: VAL-II LOCATION DATA FILES

P_OSITION : TO DISPLAY POSITION OF CLAMP
S_MOOTH : COLLISION CHECKS (ON/OFF)
F_ACES : BACK FACES (ON/OFF)
O_RIENT : CALLS ORIENT1 TO SCALE, TRANSLATE OR ROTATE MODEL
A_XES : COORDINATE FRAMES (ON/OFF)

R_ETURN : TO RETURN TO THE TOP OF A MENU
M_ODEL : TO DISPLAY THE MODEL
V_AL-II : BEFORE ENTERING A VAL COMMAND

Figure 2-3. Keyboard Menu

To reduce the number of lines being drawn and tocreate an illusion that the interior region of a displayed surface is opaque, it must be ensured that the sections of the object which would be hidden from an observer by the opaque surface are not displayed. As explained before, the simulator uses the flat faces, which introduces the property for the vertices of the bounding polygon that they all lie on one and the same plane. More strictly, in the simulator, it is stipulated that the objects be not only plane-faced, but also convex. In this case, a simple calculation of the normal of a surface is sufficient for determining whether this face is a "front face" and potentially visible, or a "back face" and thus invisible [20]. Those polygons whose normals point into the terminal's screen are not displayed. Needless to say, this algorithm only removes a face(s) which is hidden by the volume of the polyhedron it belongs to.

This process of eliminating the back surfaces makes two important contributions. With this algorithm all of the polygons are drawn separately, so that they may be displayed selectively. Sometimes, this causes an edge to be duplicated, unless the wire-frame polyhedrons are drawn efficiently. It is often the case that the number of lines that are not displayed is equal to that of those that are duplicated (Figure 2-5). Therefore, for either display, the number of actual lines drawn is the same. Hence, the

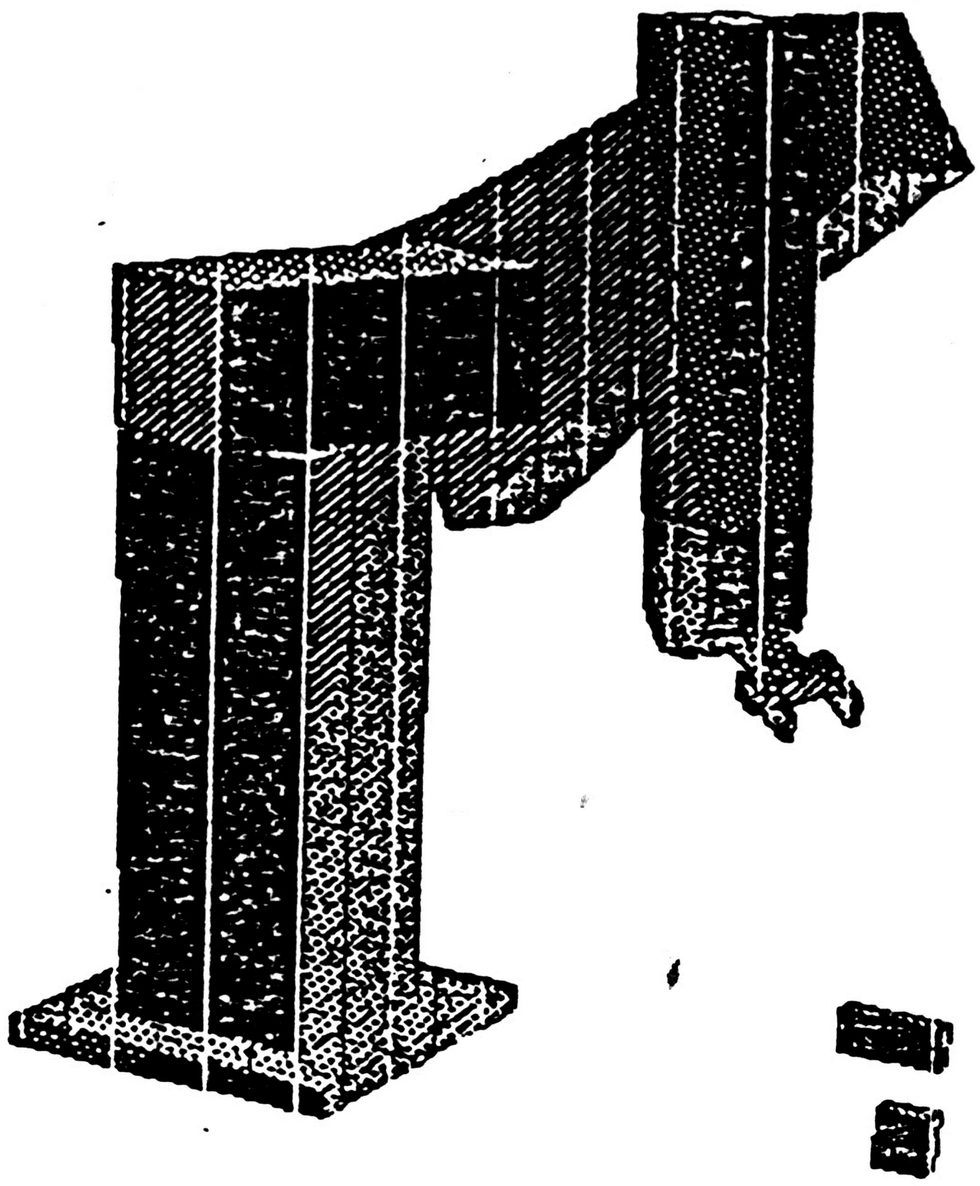
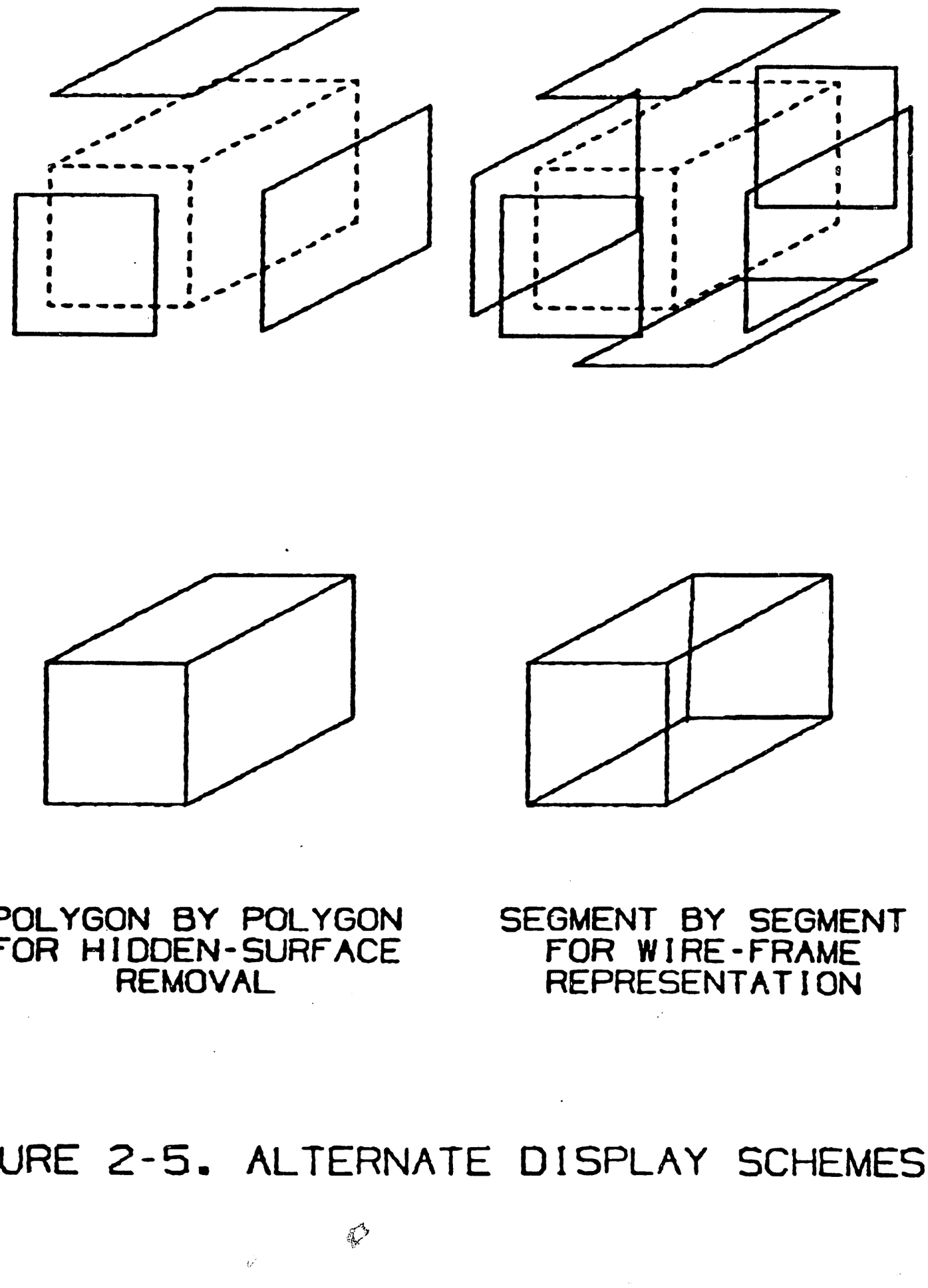


Figure 2-4. A Color Shaded Solid Model representation produced by interface to GEOMOD through POLYGON

statements determining which polygons to display have a significant effect upon the simulation display speed.

This so called back-face elimination algorithm requires the calculation of the coordinates of normals for all of the polygons. At the .pa beginning of each VAL-II session, the simulator uses the first three vertices (numbered clockwise) of any polygon to define lines in the plane of the polygon. The next step is to determine their vector cross products, which define the normal lines to these polygons. Then, in the subroutine PICTUR (Appendix A) and before the polygons are drawn, the coordinates of the normals are checked to see if they point into the terminal's screen or not. This is done by looking at the value of the Z component of the coordinate of a normal. If this value is less than zero, the normal points into the display screen, a flag is set, and as a result the polygon is not displayed. The faces and their normals define a coordinate system in each face of a polyhedron. This means that, whenever a polyhedron definition data is rotated, so are the coordinates of the respective normals. However, this is done only when the flags are set for hidden-surface elimination. Thus, the coordinates of normals are not rotated when the full wire-frame display option is used.



POLYGON BY POLYGON FOR HIDDEN-SURFACE REMOVAL

SEGMENT BY SEGMENT FOR WIRE-FRAME REPRESENTATION

FIGURE 2-5. ALTERNATE DISPLAY SCHEMES.

2.4 ROBOT KINEMATICS

As indicated above, the PUMA has been modeled by a series of extruded polyhedrons. The simulation of PUMA motion is based on the method of coordinate transformation [21] for describing robot kinematics and manipulating graphics objects. A coordinate frame is assigned to each robot link using the Denavit-Hartenberg convention [22]. Themethod used here has already been presented in [3], but for continuity will be given here. The column vector

$$\{u\} = \begin{Bmatrix} x \\ y \\ z \\ w \end{Bmatrix} \qquad (1)$$

represents a point in space. It can also be represented as

$$u = (x/w)\hat{i} + (y/w)\hat{j} + (z/w)\hat{k} \qquad (2)$$

where x, y, and z are the components in the $\hat{i}$, $\hat{j}$, and $\hat{k}$ directions and w is a scale factor. Given the point {u}, its transformation {v} is represented by the matrix product

$$\{v\} = [H] * \{u\} \qquad (3)$$

where [H] is a 4 by 4 homogeneous matrix representing any combination of rotation, translation, perspective or

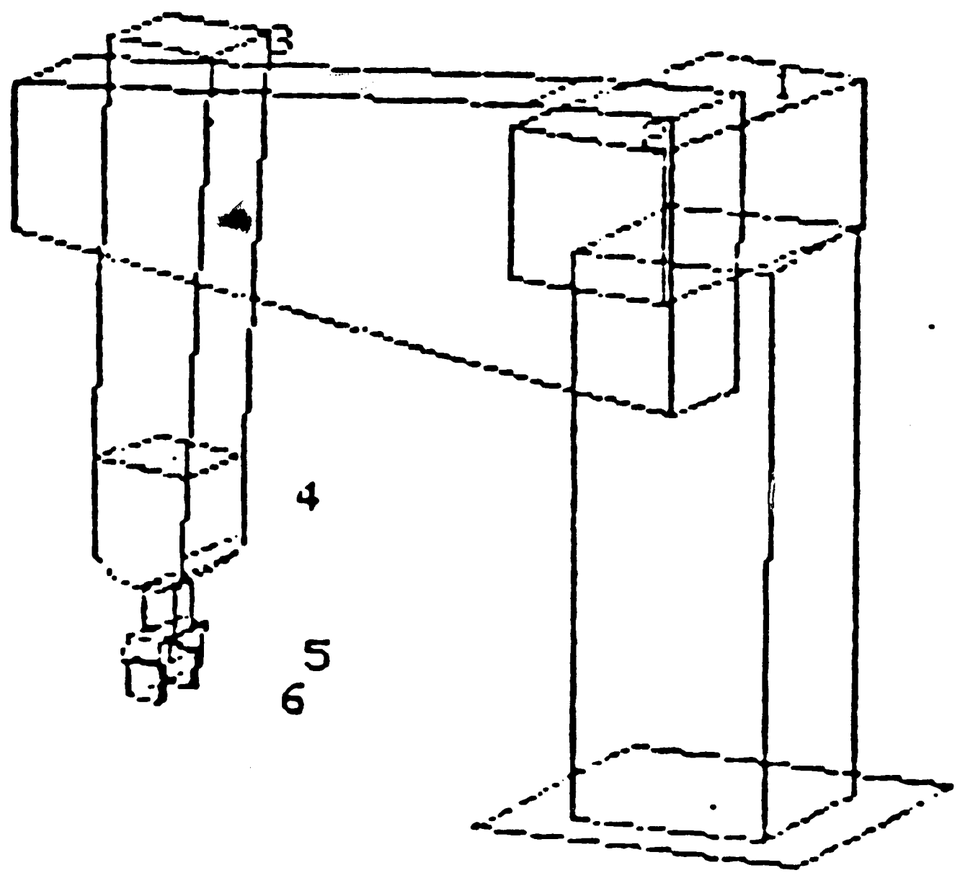


Figure 2-6. Definition of joints

scaling transformations. In the simulator 3 by 3 transformation matrices are used, since all of the PUMA joints are revolute and only rotations are required. As an example, let a=x/w, b=y/w, c=z/w be the coordinates of a point in space. Then, a rotation by an angle O, about the fixed Screen Coordinate System (SCS) Y-axis, gives

$$\begin{Bmatrix} a' \\ b' \\ c' \end{Bmatrix} = \begin{bmatrix} \text{Cos } \theta & 0 & \text{Sin } \theta \\ 0 & 1 & 0 \\ -\text{Sin } \theta & 0 & \text{Cos } \theta \end{bmatrix} * \begin{Bmatrix} a \\ b \\ c \end{Bmatrix} \qquad (4)$$

The rotated data (a', b', c') have been obtained by premultiplying the definition data by the homogeneous transformation matrix. Similar rotation matrices exist for rotations about X and Z axes. Referring to Figure 2-6, one sees that rotating the first joint (WAIST) requires premultiplying the coordinates of the first polyhedron by the above transformation matrix. However, the rotation of joint two (SHOULDER) requires a transformation with respect to the first joint and not with respect to the fixed SCS. Thus, the rotation about the X-axis, shifted to the rotation axis of the second polyhedron, follows the rotation about the Y-axis. Therefore, the transformation matrices must be multiplied together to obtain the concatenated transformation matrix for each joint. The shift to the second rotation axis or translation may be included in the concatenated matrix as in the Denavit-

Hrtenberg convention [22]. As pointed out, in the simulator instead of 4 by 4 transformation matrices, 3 by 3 homogeneous matrices have been used and the respective translations are included later. This has the advantage in that it makes the algorithm easy to follow and also results in quicker response, hence smoother motion, in the price of making the transformation subroutine less applicable to other robots.

To understand this algorithm in more detail, consider the first two links, namely the Waist and the Shoulder of the model and Table 2-1. The polyhedra definition coordinates are stored in an array DEF_DAT, the rotated data in an array ROT_DAT, and the display data in another array called DIS_DAT. Using the above algorithm, to obtain the rotated data, ROT_DAT, the coordinates of the polyhedrons in DEF_DAT must be premultiplied by the appropriate transformation matrix. Then, to determine the display data, DIS_DAT, the appropriate translations are added to the rotated data. Column 5 labels the points stored in the various arrays, DEF_DAT, ROT_DAT, DIS_DAT. Each polyhedron of the model is marked by a reference or a pivot point (REF1, REF2,...). Column 1 contains the appropriate transformation matrices which multiply the reference points and coordinates of the vertices of the polyhedron definition data, in the manner of equation (4). Forming the transformation matrices requires a great deal

TABLE 2-1. DISPLAY ALGORITHM LOGIC

1		2 definition		3 rotated		4 display		5
[I]	•	(a1,b1,c1)	=	(a1,b1,c1)	=REF1			REF1
[T1]	•	(a2,b2,c2)	=	(a'2,b'2,c'2)		(a'2,b'2,c'2)+REF1		
[T1]	•	(a3,b3,c3)	=	(a'3,b'3,c'3)	=ROTDAT1	(a'3,b'3,c'3)+REF1	=DISDAT1	POLY1
[T1]	•	(a4,b4,c4)	=	(a'4,b'4,c'4)		(a'4,b'4,c'4)+REF1		
		⋮		⋮		⋮		
[T1]	•	(an,bn,cn)	=	(a'n,b'n,c'n)	=REF2			REF2
[T2]	•	(an+1,bn+1,cn+1)	=	(a'n+1,b'n+1,c'n+1)		(a'n+1,b'n+1,c'n+1)+REF1+REF2		
[T2]	•	(an+2,bn+2,cn+2)	=	(a'n+2,b'n+2,c'n+2)	=ROTDAT2	(a'n+2,b'n+2,c'n+2)+REF1+REF2	=DISDAT2	POLY2
[T2]	•	(an+3,bn+3,cn+3)	=	(a'n+3,b'n+3,c'n+3)		(a'n+3,b'n+3,c'n+3)+REF1+REF2		
		⋮		⋮				
[T2]	•	(an+m,bn+m,cn+m)	=	(a'n+3,b'n+3,c'n+3)	=REF3			REF3

of sine and cosine calculations of various angles. So, to minimize the time required for these computations, the sines and cosines of angles are computed once and stored.

For example, if the rotated data for the second polyhedron (POLY2) is required, the concatenated transformation is obtained by multiplying the transformation matrices together, as follows:

$$[T2] = \begin{bmatrix} C1 & 0 & S1 \\ 0 & 1 & 0 \\ -S1 & 0 & C1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & C2 & -S2 \\ 0 & S2 & C2 \end{bmatrix} = \begin{bmatrix} C1 & S1S2 & S1C2 \\ 0 & C2 & -S2 \\ -S1 & C1S2 & C1C2 \end{bmatrix} \quad (5)$$

Note that, in obtaining the concatenated matrix, the order of multiplication is critical. Also, here for ease of notation, Sj = Sin (joint angly j), and Cj = Cos (joint anglej) (Table 2-2). Now, the rotated dataisobtained by premultiplying the definition data of the coordinates of the corresponding polyhedron by the above homogeneous transformation matrix

$$\begin{Bmatrix} a'_{n+i} \\ b'_{n+i} \\ c'_{n+i} \end{Bmatrix} = \begin{bmatrix} C1 & S1S2 & S1C2 \\ 0 & C2 & -S2 \\ -S1 & C1S2 & C1C2 \end{bmatrix} * \begin{Bmatrix} a_{n+i} \\ b_{n+i} \\ c_{n+i} \end{Bmatrix} \quad (6)$$

where (i=1,2,..,m) represents the indices of the coordinates of the second polyhedron (POLY2) and the

$$[T1] = \begin{bmatrix} C1 & 0 & S1 \\ 0 & 1 & 0 \\ -S1 & 0 & C1 \end{bmatrix}$$

$$[T2] = \begin{bmatrix} C1 & S1S2 & S1C2 \\ 0 & C2 & -S2 \\ -S1 & C1S2 & C1C2 \end{bmatrix}$$

$$[T3] = \begin{bmatrix} C1 & S1S23 & S1C23 \\ 0 & C23 & -S23 \\ -S1 & C1S23 & C1C23 \end{bmatrix}$$

$$[T4] = \begin{bmatrix} C1C4+S1S23S4 & -C1S4+S1S23C4 & S1C23 \\ C23S4 & C23C4 & -S23 \\ -S1C4+C1S23S4 & S1S4+C1S23C4 & C1C23 \end{bmatrix} = \begin{bmatrix} M11 & M12 & M13 \\ M21 & M22 & M23 \\ M31 & M32 & M33 \end{bmatrix}$$

$$[T5] = \begin{bmatrix} M11C6+M12C5S6+M13S5S6 & -M11S6+M12C5C6+M13S5C6 & -M12S5+M13C5 \\ M21C6+M22C5S6+M23S5S6 & -M21S6+M22C5C6+M23S5C6 & -M22S5+M23C5 \\ M31C6+M32C5S6+M33S5S6 & -M31S6+M32C5C6+M33S5C6 & -M32S5+M33C5 \end{bmatrix}$$

where $Sij = SIN(angle\ i + angle\ j)$
and $Cij = COS(angle\ i + angle\ j)$

ROTATION TERMS FOR BOTH JOINTS 5 & 6 ARE INCLUDED IN [T5] OR THE CLAMP TRANSFORMATION MATRIX. SINCE A ROTATION ABOUT EITHER ONE, ONLY TRANSFORMS THE CLAMP.

TABLE 2-2. ROTATION MATRICES USED FOR EACH JOINT

reference point for the third polyhedron, (POLY3). Now, in order to obtain the display data for the second polyhedron, DIS_DAT2, the rotated data is added to REF1 and REF2. For example, the display data of a point in the array DIS_DAT2 is

$$a''_{n+1} = a'_{n+1} + a_1 + a'_n \qquad (7)$$

$$b''_{n+1} = b'_{n+1} + b_1 + b'_n \qquad (8)$$

$$c''_{n+1} = c'_{n+1} + c_1 + c'_n \qquad (9)$$

Similarly, we can determine the subsequent polyhedrons display data. However, the process of obtaining the transformation matrices is a complicated and time-consuming one. The amount of computations must be minimized as much as possible to provide a fairly quick response and smooth motion. As explained earlier, PUMA has 6 revolute joints. If we consider a rotation about the waist of the model, the display data must be computed for the first polyhedron as well as for all of the subsequent ones. But, for example, a rotation about joint three does not affect joints 1 and 2, and so on. For this reason, dedicated routines have been used so that no display data is computed unless necessary. For instance, if the FOREARM is to be rotated, a subroutine named ELBOW (in Appendix A) is called. This routine calculates all the sines and cosines of corresponding

angles and forms the corresponding transformation matrix, and obtains the rotated data. It then calls another subroutine called FOREARM (in Appendix A) and process continues until all the data is computed for all joints. At this point a flag is checked to see whether the clamp is to be displayed in the "OPEN" or "CLOSE" position and a subroutine is called to calculate the display data. This routitne also checks to see if any user defined object is to be rotated. Lastly, the reference points are added to the respective rotated links and object data and the new model configuration is displayed.

2.4.1 JOINT-COORDINATED MOTION

The above rotation algorithm has been used to simulate joint-coordinated motion. In the simulator, the algorithm for this type of motion has been coded so that, initially, the largest joint angle difference is computed. It is then divided by a speed dependent increment to determine the number of configurations to be displayed between the current and final desired configurations.Then, based upon the number of increments, a step size is calculated for each joint's rotation angle difference. At this point all the joint angles are incremented simultaneously by the corresponding step size, and the necessary display data generated for each intermediate configuration until the

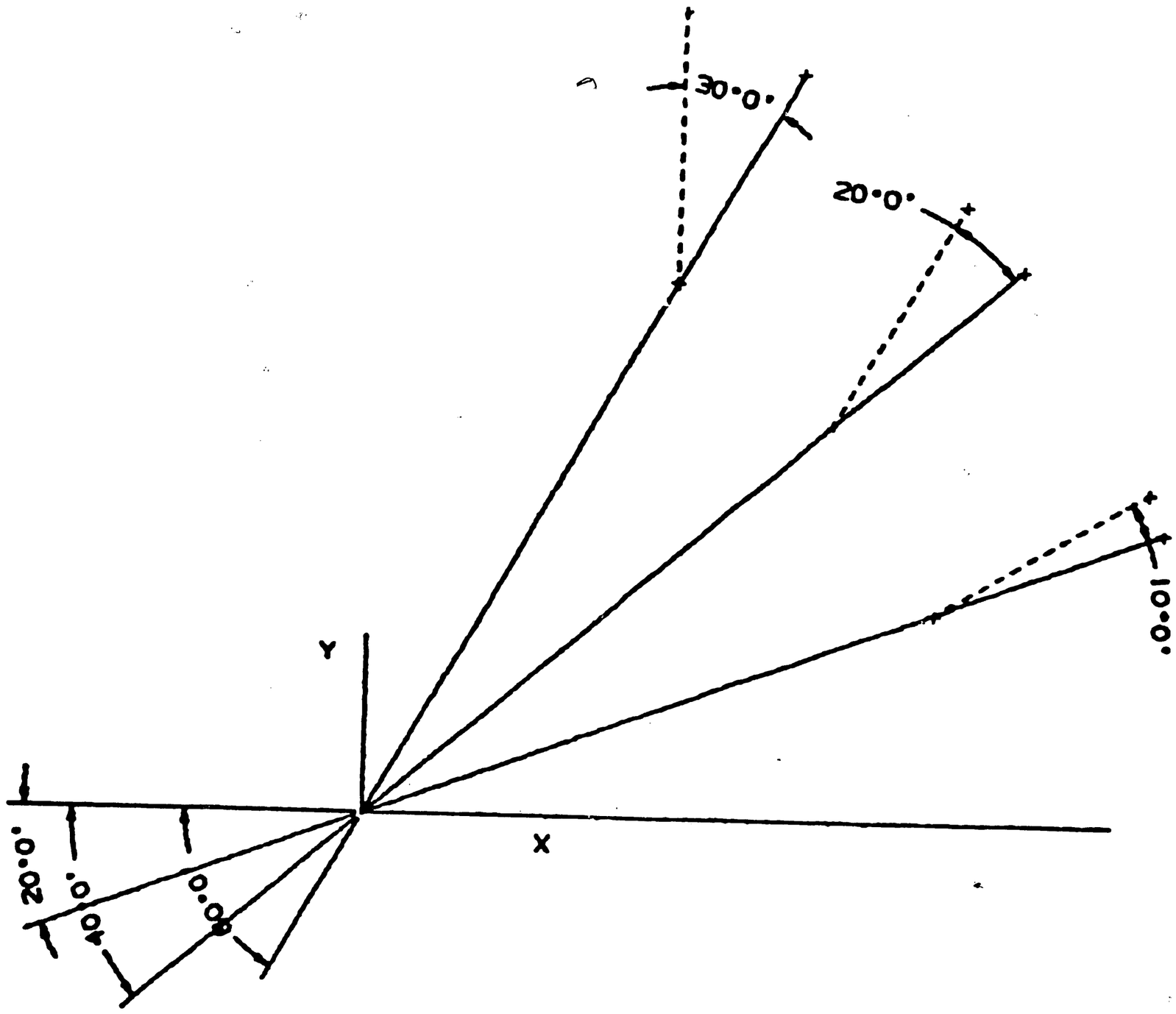


FIGURE 2-7. TWO DIMENSIONAL STICK-FIGURE JOINT-COORDINATED MOTION

final desired configuration is reached. With joint-coordinated motion, the clamp speed is not constant, since it follows a complicated three dimensional space-curve. To understand the method used here, refer to Figure 2-7. It shows how this method is applied to a two dimensional stick figure model. In the configuration drawn the first link is oriented 60 degrees from the horizontal and the second link is 30 degrees from the first one. In the final desired configuration, assume that they both are horizontal. Suppose that in this case, the speed dependent increment is 20 degrees. The largest joint angle difference is 60 degrees, hence, the number of steps is easily determined to be 3. The second joint angle difference, 30 degrees, is then divided by the number of steps to yield a step size of 10 degrees. This procedure may be repeated if there are more joints. Upon computation of all step sizes, the links are rotated simultaneously by the respective incremental amounts and the result, in this case, is displayed at two intermediate locations and the final configuration. If the calculated number of steps is not a whole number, the increment between the last intermediate and the final configuration for each joint is adjusted to a size proportional to the remainder.

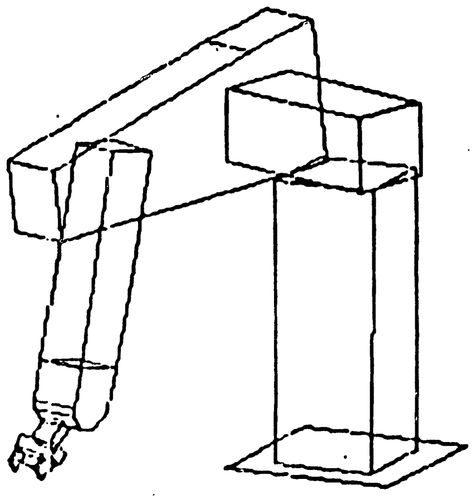
2.4.2 STRAIGHT-LINE MOTION

For controlling manipulator arms which exhibit anthropomorphic geometrical and mechanical characteristics (i.e. an arm with solely rotating joints and with redundant degrees of freedom) it is necessary to solve inverse kinematics equations which is a very difficult task [23].

In the simulator, straight-line motion uses the inverse kinematic solution by Lee [24], to invert between the PUMA's location data format and the joint angles.

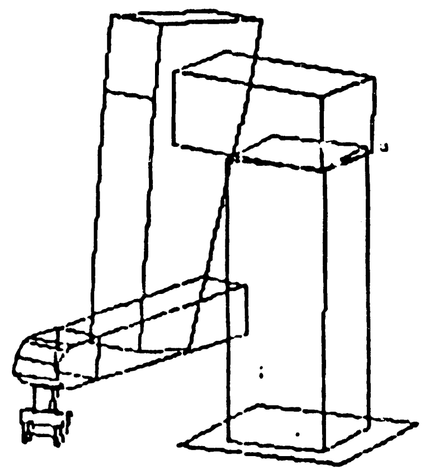
Lee's geometric approach, determines various arm configurations of a PUMA robot, based on the link coordinate systems and human arm geometry, with the help of three configuration indicators. These indicators enable one to find a solution from the possible four solutions for the first three joints, and a solution from the possible two solutions for the last three joints, for a six-link PUMA robot. This method can be extended to any robot arm with rotary joints. Orthogonal coordinate frames are defined at each joint with the Z-axis pointing in the direction of motion and the X-axis points away from and is normal to the previous Z-axis. The labeling of the coordinate systems begins from the supporting base to the end-effector of the PUMA arm (Figure 2-6). As indicated above, associated with the joint solution are three indicators, two with the solution of the first three joints (either a LEFT or RIGHT

OPENED	XW	YW	ZW	O	A	T
RIGHTY	338.53	534.32	-565.51	163.19	42.95	170.85
ABOVE	JT 1	JT 2	JT 3	JT 4	JT 5	JT 6
NOFLIP	-108.82	172.08	19.34	-1.52	35.64	-9.54



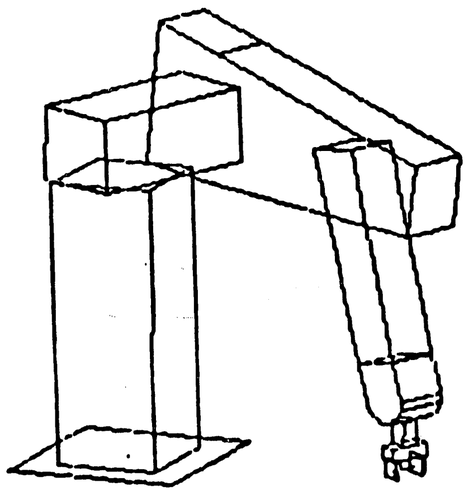
a.

OPENED	XW	YW	ZW	O	A	T
RIGHTY	249.46	494.67	-550.56	167.32	82.94	158.35
BELOW	JT 1	JT 2	JT 3	JT 4	JT 5	JT 6
NOFLIP	-108.82	102.63	168.66	-8.33	-84.24	-18.99



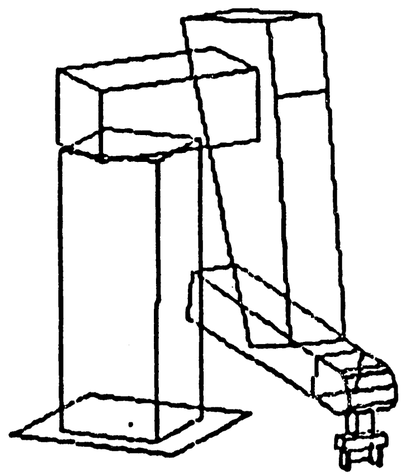
b.

OPENED	XW	YW	ZW	O	A	T
LEFTY	-537.29	61.79	-610.00	65.00	89.00	0.00
ABOVE	JT 1	JT 2	JT 3	JT 4	JT 5	JT 6
NOFLIP	156.36	8.18	159.13	-179.90	-13.68	1.27



c.

OPENED	XW	YW	ZW	O	A	T
LEFTY	-511.60	175.40	-610.00	52.64	89.00	0.00
BELOW	JT 1	JT 2	JT 3	JT 4	JT 5	JT 6
NOFLIP	144.00	76.14	20.87	-179.98	-84.00	1.36



d.

Figure 2-8. Various PUMA arm configurations

hand arm configuration, and either elbow ABOVE or BELOW the wrist configuration, Figure 2-8), and one with the last three joints (FLIP or NOFLIP). These configurations are prespecified by the user for finding the inverse solution. To determine the first joint angle, the projection of the position vector in the X_0-Y_0 plane from the WCS origin to the intersection point of the last three joint axes is found. The first joint angle is determined from the equations which result from equating the components of the projection of the position vector desired by the appropriate concatenated transformation matrix. The second configuration indicator specifies an elbow ABOVE or BELOW the wrist. For joint two, the projection of the same position vector is made onto the X_1-Y_1 plane. Joint two is determined with the equations which result from the geometry in this plane and the second configuration indicator. For joint three, we project the same position vector onto the X_2-Y_2 plane. The third joint angle is obtained from the resulting geometric equations and the first two configuration variables. Knowing the first three joint angles, we can find the solution of the last three joints. The solution of the last three joints of PUMA robot arm can be found by setting these joints to meet the following criteria:

1) Set joint 4 such that a rotation about joint 5 will align the axis of motion of joint 6 with the given approach

OPENED	XW	YW	ZW	O	A	T
LEFTY	990.60	158.75	0.00	90.00	0.00	0.00
ABOVE	JT 1	JT 2	JT 3	JT 4	JT 5	JT 6
NOFLIP	0.00	0.00	90.00	0.00	0.00	0.00

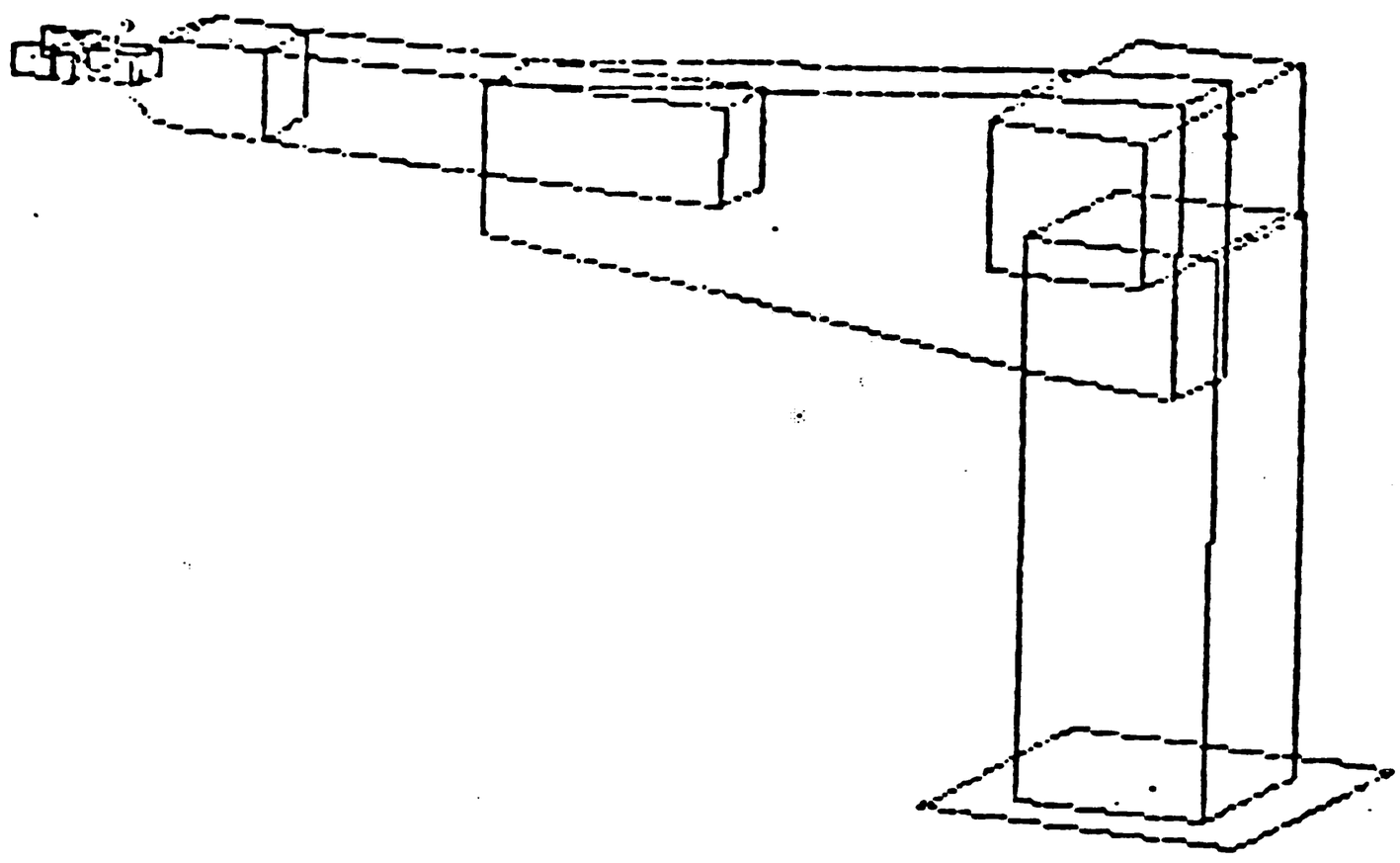


Figure 2-9 . Three zero angle convention according to Lee

vector

2) Set joint 5 to align the axis of motion of joint 6 with the approach vector

3) Set joint 6 to align the given orientation vector and normal vector

For a better understanding of the inverse algorithm, refer to Lee's paper [24], and subroutine INVERSE in Appendix A.

In the simulator, three zero angle position conventions are used. Lee's geometric solution takes advantage of the angle convention which defines the horizontal robot configuration as the zero position (Figure 2-9). The simulator uses the widely accepted statically balanced vertical configuration [21] which defines the zero position for each joint mid-way between the stop limits (Figure 2-10). The PUMA's convention is different and uses a combination of the other 2 (Figure 2-11). Presumably, this convention was chosen for stepper motor convenience. To conveniently calculate the inverse solution, to rotate model efficiently, and to display data in the format familiar to PUMA users, in the simulator, conversions are made between the three conventions mentioned above. The simulator determines the joint angles in Lee's convention for the current configuration by simply adding 90 degrees to the second joint in the simulator convention.

Straight-line motion is more involved, and requires more computations, and thus is slower than joint-

OPENED	XW	YW	ZW	O	A	T
RIGHTY	0.00	158.75	990.60	90.00	-90.00	0.00
ABOVE	JT 1	JT 2	JT 3	JT 4	JT 5	JT 6
NOFLIP	0.00	-90.00	90.00	0.00	0.00	0.00

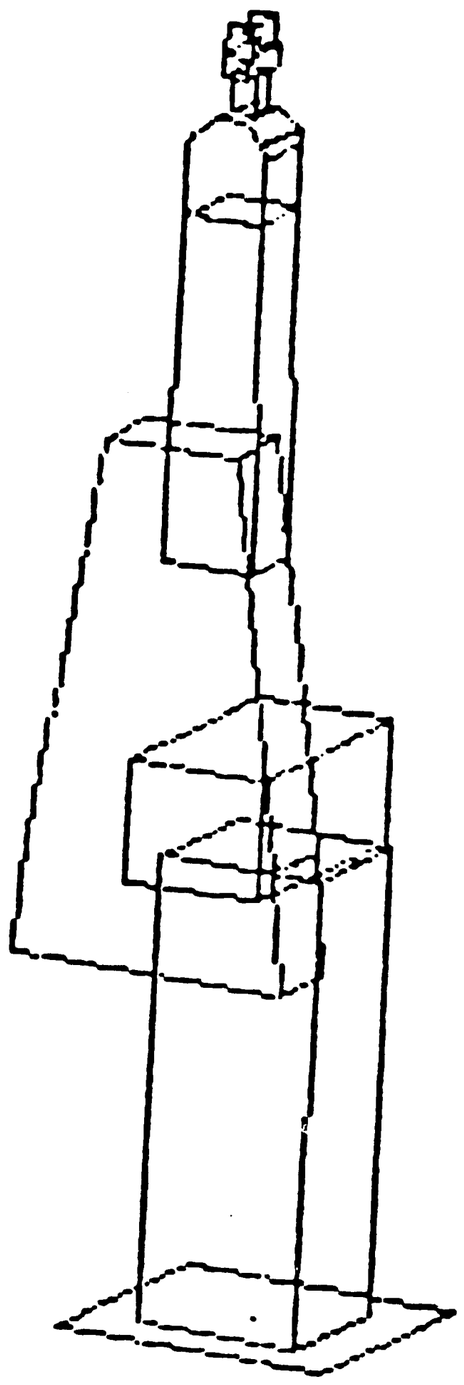


Figure 2-10. Three zero angle convention according to the Simulator

coordinated motion. It is used in WORLD and TOOL and VAL-II modes. For straight-line motion the first step is to determine the joint angles in Lee's convention as described above. The joint angles for the final desired configuration are determined by applying the inverse kinematic solution to the corresponding Lee's convention data. Once known, the joint angles are used to calculate orientation matrices and theoretical configuration indicators [24]. In the VAL-II mode, the user defined configuration indicators and the next predicted configuration values are compared. If they disagree, no move is made. The same procedure is followed while in PENDANT mode. The theoretical configuration indicators at the current and the next predicted configuration are compared. Again, if there is a disagreement between the corresponding values, PUMA controller will not allow any motion, and an informative warning message is displayed. Just like in the joint-coordinated motion, for the straight-line motion along an allowable path, differences between the beginning and the end positions and entities in the orientation matrices are calculated. At this point, the number of steps is determined by dividing the maximum position coordinate differences by a speed dependent increment. Once the number of steps is known, differences for each position coordinate and each orientation matrix are incremented from the current to the next desired configuration by the

OPENED	XW	YW	ZW	O	A	T
RIGHTY	429.06	158.75	558.80	90.00	-90.20	0.00
ABOVE	JT 1	JT 2	JT 3	JT 4	JT 5	JT 6
NOFLIP	0.00	0.00	-0.20	0.00	0.00	0.00

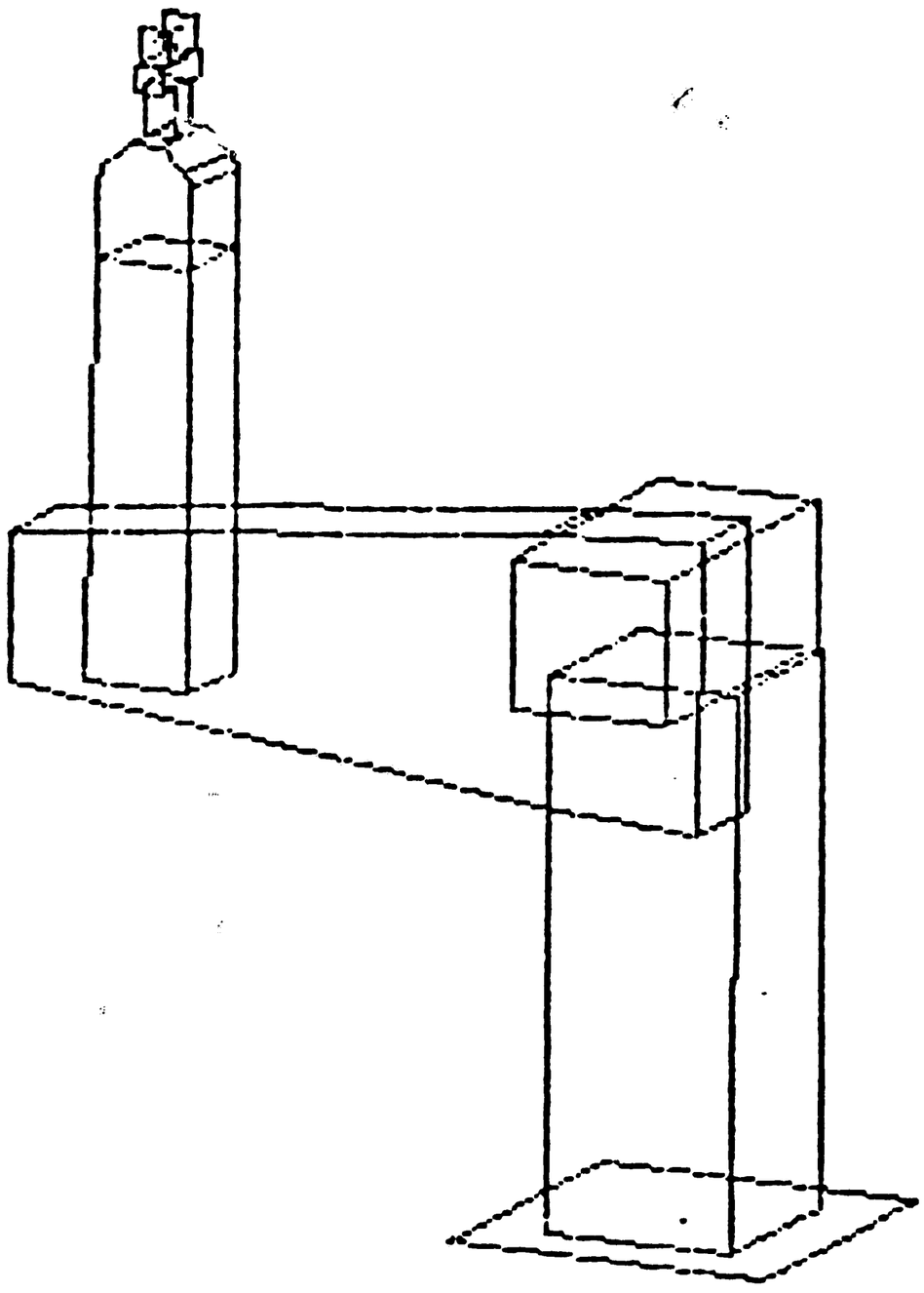


Figure 2-11. Three zero angle convention according to the PUMA

appropriate step sizes. Table 2-3 shows an example beginning, two intermediate, and a final desired orientation matrix for a straight-line PUMA robot motion. As can be seen the only elemnt that is changes is the translational elemnt and the rest remain the same.

2.5 TRANSFORMATION OF USER-DEFINED OBJECTS

The simulator can be used to move user-defined objects inside the robot work-cell. This is one of the important features of VAL-II SIMULATOR. This process of grasping an object, moving it, and then detaching from it is all simulated in a user-friendly fashion. Once the LED sensor is intersected by any of the workpieces, a flag is set and a wrist to object reference point (first corner) distance is calculated. As mentioned previously, the transformations are done with respect to the fixed SCS, which means that the object must be shifted to SCS. This is essentially done by subtracting the coordinates of the reference vertex for the object from all of the vertices display data. At this point the object definition data which is redefined with respect to SCS, must be redefined. This is done by premultiplying the object data by transpose of the clamp transformation matrix, $[T5]^T$. Now, to redefine the object with respect to the wrist, and make it a part of the clamp, the redefined object definition data must be

$$\begin{bmatrix} -0.62 & 0.15 & -0.77 & 12.84 \\ -0.07 & 0.97 & 0.24 & \underline{21.11} \\ 0.79 & 0.20 & -0.59 & 24.71 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$$

$$\begin{bmatrix} -0.62 & 0.15 & -0.77 & 12.84 \\ -0.07 & 0.97 & 0.24 & \underline{18.50} \\ 0.79 & 0.20 & -0.59 & 24.71 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$$

$$\begin{bmatrix} -0.62 & 0.15 & -0.77 & 12.84 \\ -0.07 & 0.97 & 0.24 & \underline{15.89} \\ 0.79 & 0.20 & -0.59 & 24.71 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$$

$$\begin{bmatrix} -0.62 & 0.15 & -0.77 & 12.84 \\ -0.07 & 0.97 & 0.24 & \underline{13.27} \\ 0.79 & 0.20 & -0.59 & 24.71 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$$

TABLE 2-3. EXAMPLE ORIENTATION MATRICES FOR STRAIGHT-LINE MOTION

premultiplied by the clamp matrix, [T5], and then shifted by the sum of all of the reference distance calculated in the first step. A reverse procedure is followed to detach an object from the clamp. When the flags are properly set to detach an object, the distance between the object and the SCS origin is calculated. The rotated object data is redefined, so that, when this new reference distance is added to the reference vertex of the object, it is displayed in the same place as it was when defined with respect to the wrist. To better understand this process, refer to subroutine LATCH .pa and DETACH in Appendix A.

2.6 SIMULATION SPEED

One major difference between the PUMA robot and its simulation is in that the dynamic effects such as acceleration, deceleration, or gravity and inertia effects have not yet been included in the simulator.

One of the advantages of robots is that they may be run at high speeds. Robots of course, follow a continuous path, as opposed to the simulator which displays the model at discrete positions along a segmented path. When this is done quickly, the repetitive images are blended to create the apparent motion. The speed variations in the simulator are based upon a relative speed and not a true one.

The PUMA controller allows the speed to be set in the

monitor mode, and then fine-tuned with the teach pendant or the VAL-II mode. But in the simulator the speed set in the pendant mode is independent of that in the VAL-II mode. In the simulator while in the pendant mode, the default speeds have been chosen sothat the resulting motion is slow to be fully observable. In joint-coordinated mode, the default speed corresponds to an adjustable joint rotation increment of 8 degrees for each key depression. Similarly, in the VAL-II mode, the default speeds for joint-coordinated and straight-line motion are adjusted with VAL-II commands. In the VAL-II mode, the default value for the largest joint angle rotation increment (full speed) is 12 degrees. This was chosen arbitrarily to produce segmented displays which run slowly enough to be easily observable, yet quickly to minimize turn-around time while debugging VAL-II programs.

2.7 COLLISION DETECTION ALGORITHM

As explained earlier in this chapter, the simulator uses extruded wire-frame convex polyhedrons to represent the robot model and the user-defined workpieces. One disadvantage of this geometric modeling scheme is that, when one object extends over and covers a part of another, it is very difficult to verify visually, whether one object is in front of, behind, or intersecting another. In order

to detect precisely, any unwanted collisions between the robot model and workpieces, between moving and stationary objects, and between the different parts of the robot itself the simulator uses both a coarse and a fine mathematical intersection check. These algorithms are valuable tools for robot users, since many undesired collisions may be prevented during VAL-II program debugging and location definition.

The coarse check algorithm, "grows" all possible obstacles using a speed dependent error margin to form a parallelepiped envelope around them. This method was suggested by Pieper [25]. It then checks to see if the midpoint of the clamp or any other moving polyhedron is inside any of these parallelepiped envelopes. In the case, when a midpoint lies inside one of these parallelepipeds, the fine check algorithm is used. Refer to Figure 2-12 to better understand this algorithm. At the beginning of a VAL-II session, the radii of circumscribed spheres, shown in yellow, for every polyhedron and clamp are calculated.

For the coarse check algorithm, the radius of moving polyhedron is added to the extreme coordinates of the vertices of each stationary polyhedron to form the yellow box shown. Simultaneously, a speed dependent margin is also added to form the lavender box. The coarse check compares the coordinates of the center point of a moving polyhedron

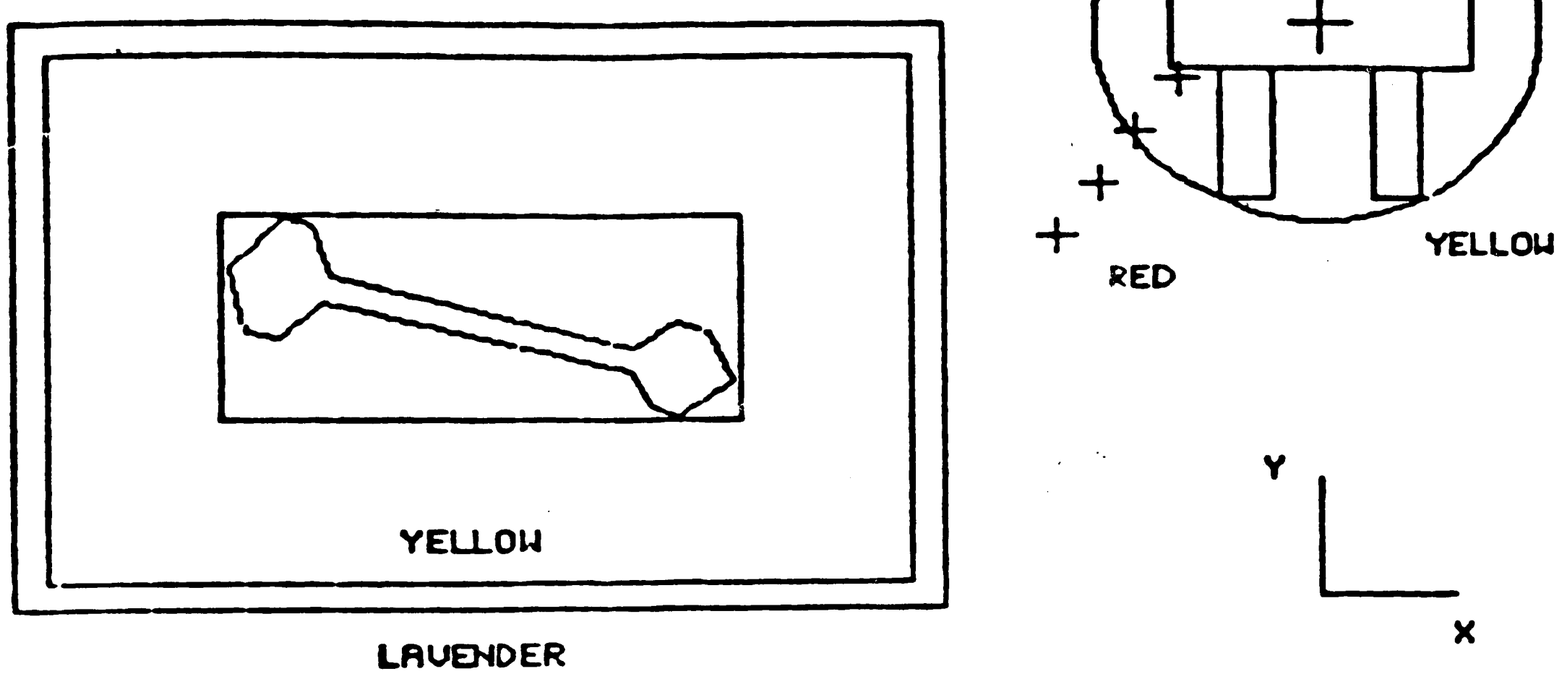


Figure 2-12. Coarse Collision Check "growing" a Polyhedron

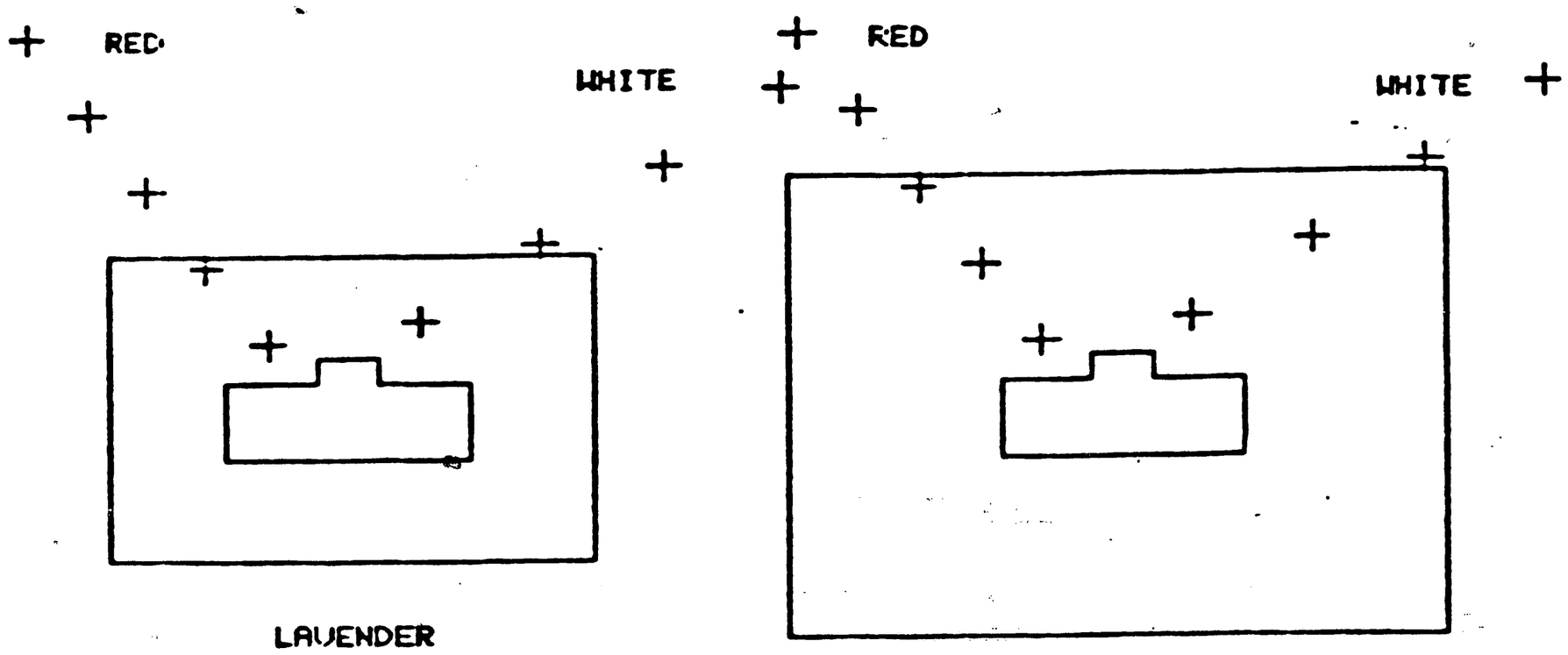


Figure 2-13. Coarse Collision Check Adding Speed Dependent Error Margin

and the maximum and minimum coordinates of the lavender box. A yellow warning message is displayed, when the coordinates of the moving center-point are inside the envelope, at which time the fine check algorithm is applied.

In order to provide precise results and quick display response, a speed dependent error margin is necessary. To understand the use of this error margin refer to Figure 2-13. In the figure the same object is shown surrounded by a small and a large lavender box. Also, a fast and a slow clamp path is shown for each, represented by a series of red and white cross hairs. If the box is small and the distance between the steps along the path represented by cross hairs large, there will not be sufficient warning time between the imminent and actual collision. On the other hand, if the box is large and the distance between the steps along the path small, the fine check algorithm is applied more often than necessary, thus slowing down the display speed of the simulator. The error margin has been selected arbitrarily to provide a good compromise.

As mentioned before, when the midpoint of a moving polyhedron is inside a parallelepiped, the program applies the fine check algorithm. This algorithm is applied when we need to determine if a line segment of a polyhedron has intersected a plane of a polygon of another polyhedron or vice versa. The problem is solved by first determining if

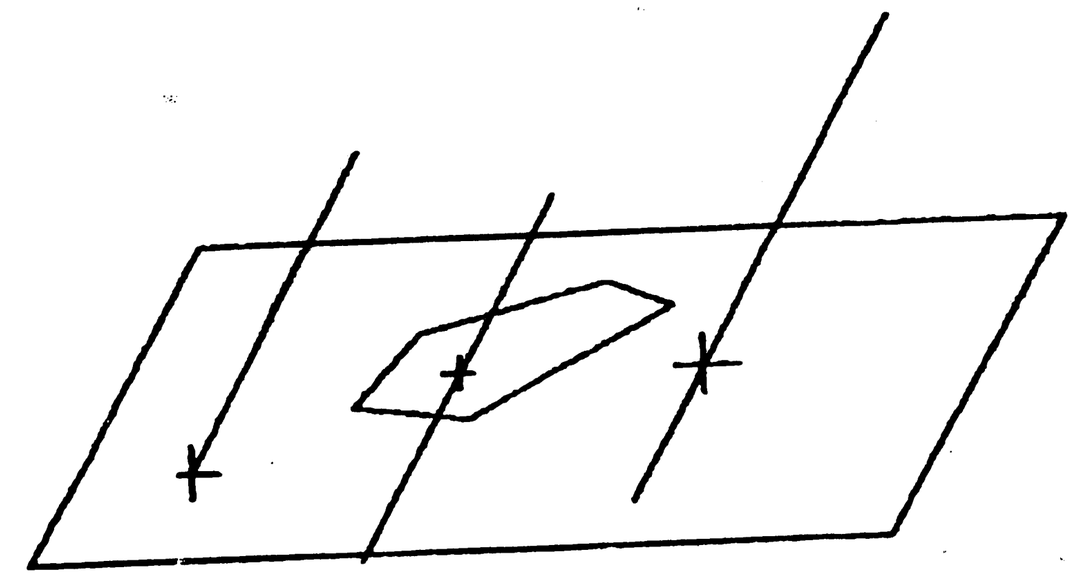


FIGURE 2-14A. FINE COLLISION CHECK-BREAKING POLYHEDRONS INTO POLYGONS AND LINE SEGEMNTS

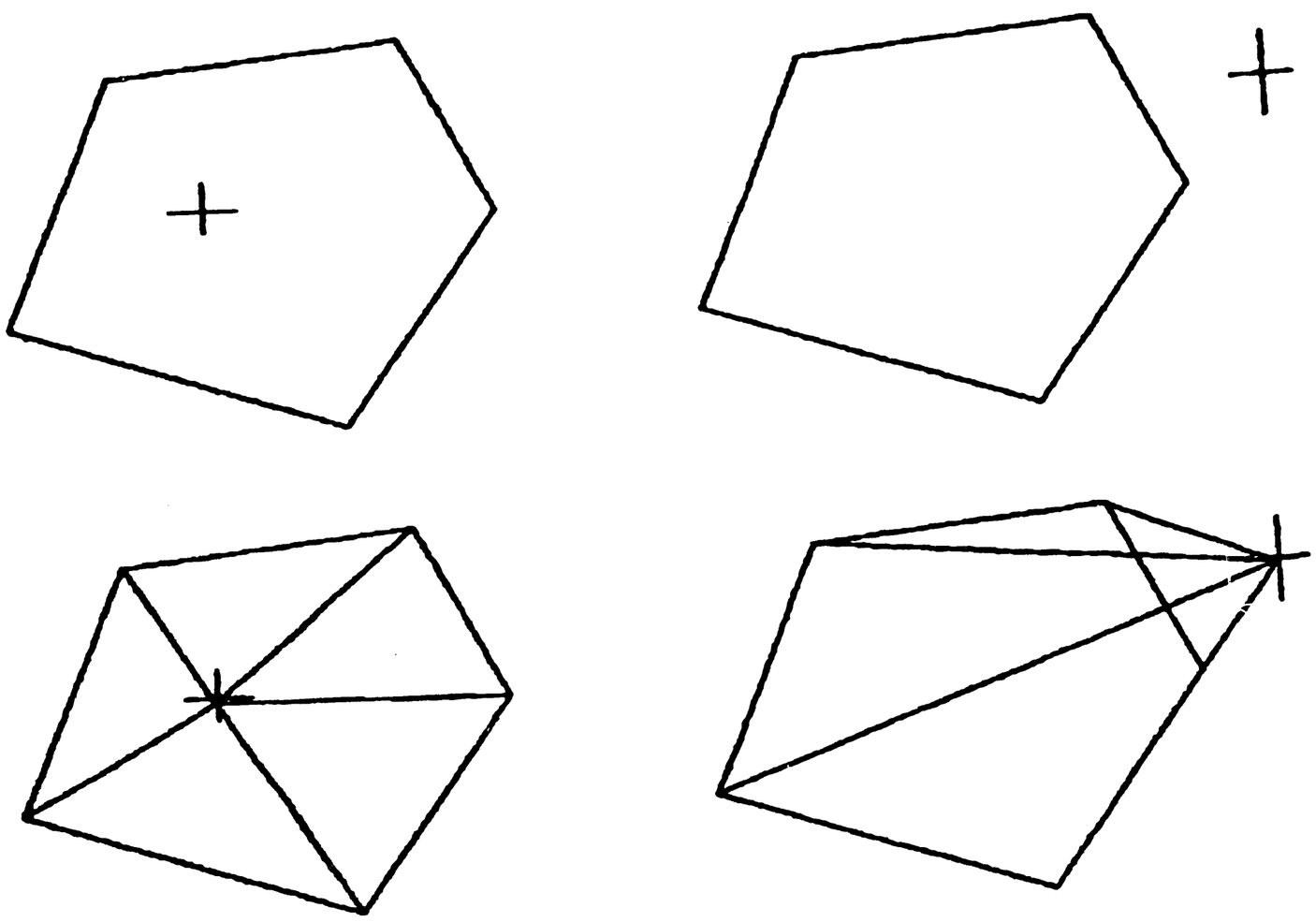


FIGURE 2-14B. FINE COLLISION CHECK-CHECKING THE INTERSECTION POINT BY SUMMING THE AREAS OF TRIANGLES

and then where the projection of a line segment intersects the plane of a polygon. There are three possibilities, since if there are line segments parallel to the plane of a polygon, there must also be lines in a direction which intersects the plane (Figure 2-14 A and B). The algorithm first checks to see if the intersection point of the line and the plane lies on the line segment. If it does, lines are drawn from the intersection point to adjacent vertices of the polygon, resulting in generation of some triangles. Then the areas of these triangles are summed. If the sum of the areas is larger than the area of the polygon, the point is outside the bounds of the polygon, otherwise a collision has taken place.

2.8 DIRECTORY INFORMATION

Information regarding the names of individual robot control programs created in the VAL-II environment, as well as information on location data files may be obtained while in this mode. Also, the content of any VAL-II programs may be displayed on the screen.

CHAPTER 3

3. PENDANT MODE

This chapter of the thesis is dedicated to discussion of the PENDANT mode (figure 3-1) of the VAL-II SIMULATOR. This portion of the simulation program was developed to graphically display the functions and usage of the "Teach Pendant" for a six-link PUMA Robot.

While in this mode, each of the 6 joints of the PUMA may be rotated independently of others, by pressing any one of the numbers 1 through 6 on the keyboard to move the corresponding joints. The joints have been numbered from the supporting base (link 0) to the robot end-effector (link 6), as in Figure 2-6. The rotation algorithm is the same as that explained in chapter 2. Also, two special cases of the straight-line motion, namely: World and Tool modes are simulated, by which the robot clamp may be moved along a straight-line parallel to any one of the axes of the World or Tool Coordinate Systems. This may be done by simply pressing 'W' (W_ORLD), or 'T' (T_OOL) followed by 'X', 'Y', or 'Z' to move the clamp along the respective axes of the corresponding coordinate systems.

The Speed in this mode is independent of that in VAL-II

PENDANT MODE

ENTER COMMANDS BY PRESSING:

1 - <JOINT 1> TO ROTATE ABOUT WAIST
2 - <JOINT 2> TO ROTATE ABOUT SHOULDER
3 - <JOINT 3> TO ROTATE ABOUT ELBOW
4 - <JOINT 4> TO ROTATE ABOUT FOREARM
5 - <JOINT 5> TO ROTATE CLAMP UP AND DOWN
6 - <JOINT 6> TO ROTATE CLAMP <CLOCKWISE/COUNTERCLOCKWISE>

O - <OPEN> TO OPEN THE CLAMP
C - <CLOSE> TO CLOSE THE CLAMP

W_ORLD : TO MOVE MODEL IN WORLD MODE
T_OOL : TO MOVE MODEL IN TOOL MODE

I_NCREASE : TO INCREASE THE ROTATIONAL INCREMENT
D_ECREASE : TO DECREASE THE ROTATIONAL INCREMENT
N_EGATIVE : TO CHANGE SIGN OF THE ROTATIONAL INCREMENT

R_ETURN : TO RETURN TO THE TOP OF A MENU
V_AL-II : BEFORE ENTERING A VAL COMMAND

Figure 3-1. PENDANT Menu

Mode. However, the rotational increment may interactively be changed by pressing 'I" (I_NCREASE), or 'D' (D_ECREASE) to increase or decrease the rotational increment, respectively. Also, the direction of joint rotations may be changed by simply pressing 'N' (N_EGATIVE), which switches from clockwise rotation to counterclockwise, or vice versa.

The clamp of the robot may be displayed in either "OPEN", or "CLOSED" position by pressing 'O' (O_PEN), or 'C' (C_LOSE), respectively.

To keep the user updated on positional data of the robot and also robot status, at all levels the position of the clamp, Euler orientation angles, and configuration indicators may be displayed.

3.1 JOINT ROTATIONS

The exact method to teach an industrial robot the points on its path is to move each link of the robot independently of others to reach the desired position and orientation and then storing it in the computer memory [26].

In the simulator each of the six joints may be rotated by pressing any one of the numbers 1-6 on the keyboard to rotate the corresponding links in the joint-coordinated motion.

No display data for the vertices of model polyhedrons

are calculated unless necessary. For example, if a rotation of joint 1 (WAIST) is required, the display data for all the subsequent joints needs to be updated. However, if the FOREARM is to be rotated, the display data for the polyhedrons of the Waist and Shoulder need not be recalculated, since they remain stationary during the coarse of this motion, but the display data for the rest of the joints (3,..,6) must be calculated. (Refer to chapter 2 for more detailed information.)

The default joint rotation increment is set arbitrarily to 8 degrees, and may interactively be increased or decresed by pressing 'I' (I_NCREASE), or 'D' (D_ECREASE), respectively. If a switch in the direction of joint rotation is required one may press 'N' (N_EGATIVE) to switch from clockwise to counterclockwise or vice versa.

3.2 WORLD MODE

World mode is a special case of straight-line motion. In this mode the clamp of the robot may be moved along straight-line parallel to any one of the axes of the World Coordinate System (WCS), while maintaining the same orientation. This may be done by pressing 'W' (W_ORLD) followed by 'X', 'Y', or 'Z' to move along the corresponding direction. For this motion, Lee's algorithm is applied to determine the final joint angles. Initially,

the orientation matrix is formed using the current joint angles. Also, user specified position along the TCS's Z-axis is referenced. The next desired position is determined by adding the increments along the appropriate axes of the WCS to the referenced current position.

In the PENDANT Mode, the simulator does not use the user-defined configuration indicators. Instead it temporarily redefines these configuration indicator values to the theoretical ones for the current position and oientation. Lee's solution is then applied to compute the final joint angles using the theoretical values obtained, thus far, as well as the final position and current orientation matrix. Once the joint angles are predicted, they are checked to see if they do not exceed the joint angles stop limits. If they are valid, the display data is calculated and the new model configuration displayed. Otherwise, the PUMA controller does not allow any motion and a warning message is displayed.

3.3 TOOL MODE

This is another special case of straight-line motion, by which the clamp may be moved along straight-line parallel to any one of the axes of the Tool Coordinate System, (TCS). This may be done by pressing 'T' (T_OOL) and the 'X', 'Y', or 'Z' to move in the corresponding

direction. As in the World Mode, the clamp maintains the same orientation during the motion.

To determine the final joint angles, current user specified position along the TCS's Z-axis and the orientation matrix is used. Initially, the position along the TCS's Z-axis is aligned with the Z-axis of the SCS. This is done by premultiplying the coordinates of the position data by the inverse of the current clamp matrix $[T5]^{-1}$. Once aligned with the SCS, the point is shifted by user controlled increment size along the appropriate axis. The incremented position is transformed back to the TCS by premultiplying it by [T5]. (One property of the transformation matrix used here is that its inverse is the same as its transpose.)

At this point, the simulator temporarily redefines the configuration indicators' values, and checks the validity of the solution obtained. If the joint angles do not exceed their prespecified stop limits, the new display data is calculated, and the arm is displayed in the new configuration. Otherwise, a warning message will appear on the screen and PUMA controller will not allow any motion.

In both, World and Tool Modes, before displaying the model, it is also checked to see if the predicted motion is inside the robot work volume. This is done by comparing the interactive joint angle rotation increment to the magnitude of joint angle 3 (ELBOW). If the angle is smaller than the

user defined rotational increment, links 2 and 3 are nearly aligned.

3.4 CLAMP POSITION AND ROBOT STATUS

In order to keep the operator updated on robot configuration (LEFTY, RIGHTY, ABOVE, ...) and also the clamp position and oientation, also its status (OPEN, or CLOSED), the proper information may be displayed on the screen, upon user's request.

At any time, the position of the clamp, Euler orientation angles, joint angles and configuration indicators may be displayed by pressing 'P' (P_OSITION) and then '1', '2', or '3' corresponding to position along the TCS's Z-axis at the origin, LED sensor and clamp tip, respectively.

CHAPTER 4

4. MANIPULATION MODE

The first step in the manufacturing engineering process is the design and layout of the work-cell. The manual process is often quite laborious and time-consuming. Perhaps, the most critical factor is the designer's real lack of knowledge of the three dimensional spatial relationship of the cell components as related to the required motion of the robot.

As mentioned before, VAL-II SIMULATOR is a language simulator and not a drafting package, meaning that, it does not duplicate the features of a CAD geometric modeler. For this reason, other CAD systems capable of 3D modeling must be used in order to create robot work-cell components. Once the cell-components are created they must be transfered to VAL-II environment. VAL-II SIMULATOR allows geometries to be transfered via an IGES software link or UNIGRAPHICS-II files (UGII files must first be interfaced with program POLYGON [19], to generate file.DAT, which is used by VAL-II SIMULATOR).

This portion of the simulation program was developed to enable the user to reposition cell components, as a simulation tool for designing and evaluating work-cell

layouts. The key direction of this section is the off-line simulation and checkout of the part positioning or flow before installation.

Upon completion of part transfering process, and after entering the MAIN MENU, one may wish to reposition any one of the cell-components inside the work environment. This may be done by pressing 'M' (M_ANIPULATION) to enter the MANIPULATION MODE. This mode enables the user to translate or rotate a user specified part. The operator may select to move the individual part by system defined increments, thru pressing 'K' (K_EYBOARD), or by user defined increments, by pressing 'I' (I_NCREMENTS). For convenience, the coordinate axes display may be turned ON or OFF. Also, during the coarse of part translation, a constant readout of the positional data of the moving workpiece with respect to both, the World and the Screen Coordinate Systems is displayed. Whenever, a part is repositioned,
its new display data is calculated and the part will be displayed in the new position. At the end of a MANIPULATION session, the user may save the new setup under the same or a new user-defined name. A flow chart describing various levels of this mode is shown in Appendix A.

4.1 COMPONENT REPOSITIONING VIA KEYBOARD

If the keyboard option is used, a menu will appear on

the screen (Figure 4-1). In order to avoid complexity and also for a neater display, and since often users memorize various options under this menu, the menu is not displayed at all times unless the operator requests for its display. While in this mode, any individual part may be translated along straight-line parallel to any one of three axes of SCS. This may be done by pressing 'B' (B_ACKWARD), 'F' (F_ORWARD), 'U' (U_PWARD), 'D' (D_OWNWARD), 'L' (L_EFT), or 'R' (R_IGHT) to move the part in positive or nrgative direction of the X, Y, or Z axes of the fixed SCS, respectively. A constant readout of the positional data, with respect to both, the World and Screen Coordinate Systems, plus the moving part number are displayed during the part repositioning process. The default increments are set to <5 mm> for each key depression. Upon each key depression the new display data of the part is calcualted and the part is displayed in the new position. At any time the user may decide to end repositioning one part and start with another.

Upon completion of a MANIPULATION session, the user is asked whether it is required to save the new setup or not. If affirmative, the new setup file may be saved under a new file-name, or the old one will be updated.

At this time, the control will be transfered to the top of the program, namely the MAIN MENU.

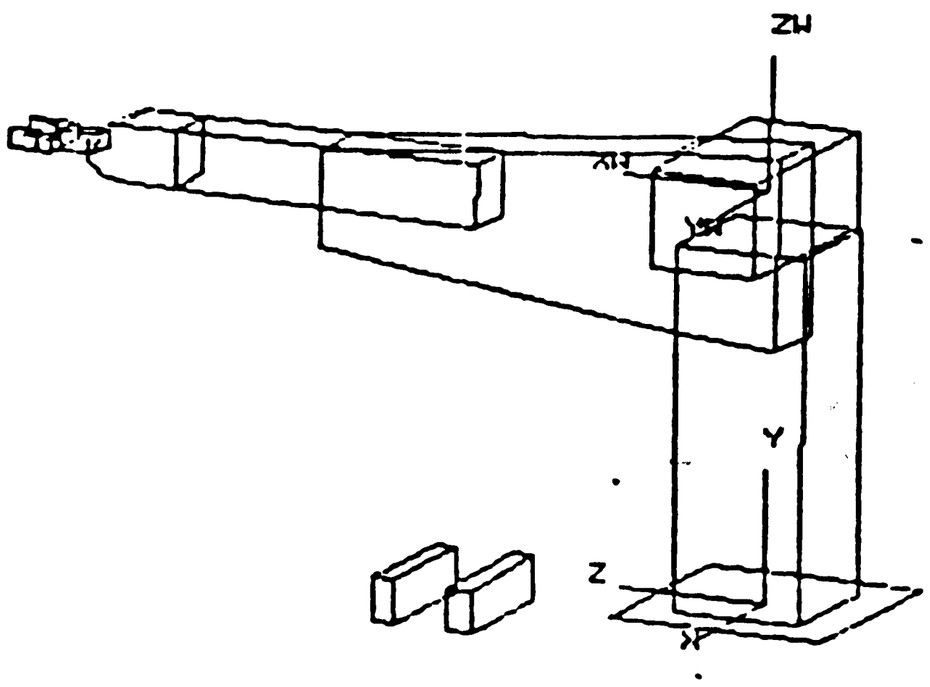
MANIPULATION MENU

ENTER COMMANDS BY PRESSING:

U_PWARD	:	MOVE THE PART UPWARD
D_OWNWARD	:	MOVE THE PART DOWNWARD
R_IGHT	:	MOVE THE PART TO RIGHT
L_EFT	:	MOVE THE PART TO LEFT
F_ORWARD	:	MOVE THE PART FORWARD
B_ACKWARD	:	MOVE THE PART BACKWARD
X_ROT	:	ROTATE THE PART ABOUT X-AXIS
Y_ROT	:	ROTATE THE PART ABOUT Y-AXIS
Z_ROT	:	ROTATE THE PART ABOUT Z-AXIS
A_XES	:	COORDINATE AXES (ON/OFF)
S_TOP	:	RETURN TO MAIN MENU

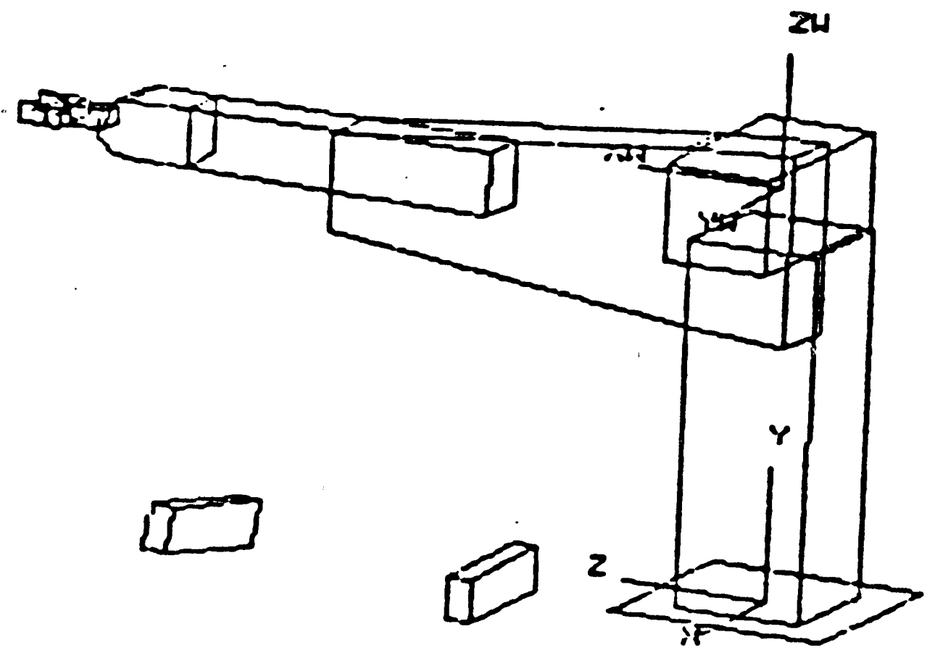
Figure 4-1. Manipulation Menu

WORLD COORDINATES FOR PART NO.	1	XW	YW	ZW
		210.00	500.00	104.40
SCREEN COORDINATES FOR PART NO.	1	X	Y	Z
		500.00	130.40	210.00



a.

WORLD COORDINATES FOR PART NO.	1	XW	YW	ZW
		510.00	500.00	104.40
SCREEN COORDINATES FOR PART NO.	1	X	Y	Z
		500.00	130.40	510.00



b.

Figure 4-2. Moving user-defined workpiece

4.2 COMPONENT REPOSITIONING VIA USER-DEFINED INCREMENTS

The user may also select to move the part by specified increments with respect to fixed SCS. Conveniently, the user may turn the coordinate axes display ON or OFF.

As before translations and rotations are allowed. The user may select to translate the specified part parallel to any one of the SCS axes. He is then asked if it is required to rotate the part. If so,the rotation incrermments are entered by the user and the new display data calculated, and the part will be displayed in the new position (Figure 4-2). A constant readout of the moving part positional data, along with the part number is displayed, at all times. At any point, the user may decide to start repositioning another part. At the end of the session the user is allowed to save the new setup under the same or a new user-defined name. The control is then transfered to the MAIN MENU.

CHAPTER 5

5. EDIT MODE

In the recent years manufacturing tasks have become complex and costly. As a result of this, robot programs specially those written for complex and sophisticated product assembly, now require more logic, functional checking, and programming constructs.

In a robot program which has been written using any one of the robot programming languages, the logic and conventional programming constructs form a significant portion of the code as opposed to "geometric move" commands. Even in a highly geometric application, such as automobile spot welding, the robot program resembles a traditional computer program [27]. In one existing automobile plant there are more than one hundred spot welding robots programmed in a high-level language. The programs consist of 500 to 1000 steps, of which the weld path is only 20 steps. Here, the geometric portion is less than 5% of the total program. Although, this is not typical of today's spot welding programs, it is an indication of the future direction of robot programming [28].

In order to make the program editing in VAL-II

SIMULATOR as easy and user-friendly as possible, a considerable amount of effort was devoted during the initial design of the EDIT Mode. The task was to make certain that, like other sections of the package, this mode was easy to use, but yet powerful enough to provide all of the desired functionality. The major design criterion was that a user, with some knowledge of VAL-II programming, could effectively use the EDITOR of VAL-II SIMULATOR (EVS), by only spending a few minutes of reviewing the Editor Menu Options (EMO).

The Editor of VAL-II SIMULATOR (EVS) is menu-driven, which means that a list of options is available at all times. The user may start out with an existing VAL-II program for further extension, or modification. It is also possible to create new programs. For convenience, each line within a program is referenced to by a unique step number. Steps are numbered consecutively, and numbers are automatically adjusted by the editor, whenever program lines are deleted or new ones inserted. These step numbers are likely to be changed as a program evolves, hence they would not be useful for identifying steps for program-controlled branching. (For this reason, program steps can contain step labels.) The Editor continuously checks the syntax of a program while it is being created or modified. That is, when the user completes a program-line by pressing the RETURN key (<CR>), the editor checks to make sure the

syntax of the line is acceptable. The line is rejected and an error message is output, if any error is found.

The Edit session may be invoked by pressing 'E' (E_DIT) and the corresponding menu (Figure 5-1) is displayed. The commands associated with the VAL-II editor are listed in this menu. These editing commands can be used for creating and modifying user programs. The menu consists of several options. One may press 'C' (C_REATE) to invoke the progrm creation mode. Any program line may be deleted, or a new lines inserted, by pressing 'D' (D_ELETE), or 'I' (I_NSERT), respectively. At the end of each session, one may press 'S' (S_AVE) to save a new or modified version of a program, or 'E' (E_XIT) to exit from the Edit Mode and return to the MAIN MENU. A flow diagram of the EDITOR structure is shown in Appendix A.

5.1 PROGRAM CREATION

The EVS allows the user to create VAL-II robot programs, without having to exit the VAL-II SIMULATOR environment, hence preservingthe programs, setup, and locations. A file may be created by typing EDIT followed by the program name, file.DAT. The EVS will first open the file DATFILER.DAT and checks to see whether the user-defined program name matches one of the entries in the

EDITOR COMMANDS

LEGAL EDITING COMMANDS

```
    EDIT     . :  INVOKES EDIT MODE

    C_REATE   :  INVOKES CREATION MODE
    I_NSERT   :  ALLOWS FOR LINE INSERTION
    D_ELETE   :  ALLOWS FOR LINE DELETION
    S_AVE     :  SAVES THE FILE
    E_XIT     :  ENDS THE EDIT SESSION

TYPE "EDIT" AND THEN PROGRAM NAME TO START EDIT SESSION
```

Figure 5-1. Editor Menu Options

list. If it does, it means that the file is an old one, hence it will be opened and the content of it displayed on the screen. However, if the file is a new one, a flag is set, the user is notified that the input file does not exist, and therefore, a new file under the user-defined name will be initialized. In this case, the new program name is added to the list, and the content of the file DATFILER.DAT is updated. The system continuously checks the syntax of the program-lines as they are entered. If a line is acceptable, it will be added to the file, and the user may proceed to the next step. Otherwise, the line is rejected and an error message is displayed.

5.2 PROGRAM MODIFICATION

For program debugging purposes, it is often necessary to delete or add VAL-II program lines. As mentioned previously, each line within a program has a unique step number referenced to it. In order to delete a program line, one must type 'D' (D_ELETE) followed by the step number. Program line insertion may be done in more or less the same method. To insert a line one must type 'I' (I_NSERT) followed by the corresponding step number. This may be repeated for as many time as necessary. Whenever, lines are inserted or deleted, the step numbers are automatically adjusted by the EVS.

Upon any program-line removal or insertion, the modified version of the VAL-II program will be displayed for user verification. During program modification, whenever, a wrong key is depressed an informative message is displayed, notifying the user of his input and the modified file will automatically be saved.

The Editor of VAL-II SIMULATOR, constantly checks the syntax of a program while it is being modified. A line is rejected and an error message is displayed if any error is encountered. Otherwise, the program line is accepted and is automatically added to the program, immediately adjusting the step numbers.

CHAPTER 6

6. VAL-II COMMANDS

As mentioned in the first chapter, VAL-II SIMULATOR utilizes some of the basic concepts and fundamental structure of its predecessor, VAL SIMULATOR [12]. Also, some of the simulated commands which were developed for the VAL SIMULATOR, have been made compatible with VAL-II SIMULATOR. However, the main emphasis of this thesis was to focus on more advanced features in terms of programming control and graphics display. Therefore, a great deal of consideration has been given to this type of VAL-II commands. For completeness, some of more important commands and those which have been made compatible with the VAL-II SIMULATOR are also discussed below. In the simulator, the VAL-II commands are grouped into eight categories: 1) Location, 2) Motion Control, 3) Hand Control, 4) Program Control, 5) Configuration Control, 6) Trajectory Control, 7) Assignment Instructions, and 8) Miscellaneous Commands. The following information is provided for each command:

1) The specific command syntax

2) An indication of when the command can be used

3) A description of the operation performed by the command

In addition to these, it is also explained how the FORTRAN

"DECODE" statement is used to read in the user input. VAL-II commands consist of a command name, usually followed by one or more command arguments. An optional comment can be included on a command line by preceding it with a semicolon. Thus, the general command syntax is

<COMMAND name> {<space><argument>} {<space>;<comment>

For clarity, all VAL-II SIMULATOR command names are shown in uppercase in this chapter. Command arguments which are to be supplied by the user are shown in lowercasse. Also, some shortened notations are used as well. Angle brackets, < >, are used to enclose a description of the actual argument to appear. Note that, these brackets are used for clarification and are never to be included as part of a command. In this chapter, optional arguments are enclosed in braces, { }. If there is a comma following such an argument, the comma must be retained if argument is omitted. For oue example, the BASE command has the form

BASE {<dx>}, {<dy>}, {<dz>}, {<z rotation>}

To specify a 300 millimeters translation in the Z-axis, the operation could be entered in any one of the following ways:

BASE 0,0,300,0 or BASE ,,300,

6.1 LOCATION COMMANDS

The three location commands, namely: HERE, WHERE, and BASE are monitor commands and specially important, at the same time very powerful, and provide significant flexibility in VAL-II programming. These instructions can be used to define or understand various robot configurations during a VAL-II program execution.

6.1.1 HERE <location variable>

HERE is a monitor command and as the mnemonic implies it defines a particular position and clamp orientation for future references. The simulator reads the HERE command followed by location variable and immediately converts the joint angles from current simulator joint angles data format to those of PUMA convention and Euler orientation angles defined with respect to the current WCS. It then associates them to the user-defined location variable and stores the converted joint angles in an array.

6.1.2 WHERE

Upon execution of this command, the current location and clamp orientation of the robot with respect to the WCS is displayed. The Cartesian World Coordinates are displayed

in millimeters and the joint angles in degrees. Note, depending upon the clamp path, rotation angle values displayed may be shifted by 360 degrees. That is a value shown as 170 degrees can also be interpretted as -190 degrees.

In the simulator, WHERE command is more consistant and provides more information than the PUMA controller. In the simulator the angles displayed for a particualr configuration are always the same regardless of the paths followed.As discussed in section 3.4, there are several position display options. Depending upon the one chosen by the user, the position display may be any one of three points along the TCS's Z-axis. The information on robot clamp status and configuration indicators as well as Euler and joint angles are displayed in addition to the above information.

6.1.3 BASE {<dx>}, {<dy>}, {<dz>}, {<z rotation>}

Upon program initialization, the origin of the World Coordinate System of the robot is assumed to be located at the intersection of the axes of joints 1 and 2, with the X, Y, and Z axes fixed in space. The BASE command offsets and rotates the reference frame as specified. It is used in VAL-II mode to perform specified tranformations with respect to the fixed WCS and user-defined work-cell.

Transformations are done with respect to SCS. For this reason, the definition and rotated object data (Table 2-1) is rotated and then translated back to its position with respect to the WCS's initial orientation. This is done first, because the BASE command prescribes the transformations to take place with respect to the initial WCS frame; then, to determine the value of the shift and rotation increments, DECODE statement of FORTRAN language is used. During the next step, the original object data is shifted by subtracting the increment and then rotating the object in the negative direction. In the same way, data for the WCS frame is read from a file and transformed. The robot model, however, remains centered on the screen at all times, and the shifts and rotation transformation are performed on the user-defined work-cell components. Note, before implementation of the BASE command, the simulator checks to see if any user-defined object is attached to the clamp. If this is the case, the same transformation for the robot and user-defined objects will be performed separately.

6.1.4 SET <location variable> = <location varibale>

This command sets the value of the location variable on the left equal to the location variable on the right of the equal sign. The simulator stores the configuration data for

the right hand name also with the first name.

6.1.5 SHIFT <pointA> BY {<dx>}, {<dy>}, {<dz>}

This command modifies the respective user-defined increments along the axes of the WCS. If the position does not violate the robot reach limits and does not exceed the allowable joint rotation limits, the modified configuration will be displayed. Otherwise, the robot controller will not allow any motion and an error message will be displayed on the screen. In this case where the configuration specified by "point A" does not exist, a simulator error message will be displayed.

6.2 MOTION COMMANDS

6.2.1 MOVE <location> / MOVES <location>

These commands cause the robot to move to the position and orientation specified by "location." In the simulator the location name is decoded and the corresponding PUMA data obtained and converted to simulator joint angle format. Intermediate configurations are determined using the joint-coordinated or straight-line motion discussed in Chapter 2. The simulator first compares the user-defined configuration indicators for the final desired location

with the theoretical ones. If the solution is not valid, no motion is allowed, and an error message is displayed.

6.2.2 ALIGN

This command causes the tool to be rotated so that its Z-axis is aligned parallel to the nearest axis of the WCS, hence, forming an orientation matrix that contains only zeros and ones. This instruction is primarily helpful for lining up the tool before a series of locations are taught. Again, the user-defined configuration indicators and the theoretical ones are first compared. The motion is then simulated only if the solution is valid and the path possible.

6.2.3 APPRO <location>,<mm> / APPROS <location>,<mm>

These commands move the tool to the position and orientation specified by "location," using joint-coordinated or straight-line motions, respectively, but offset along the tool Z-axis by the distance given in millimeters. A positive distance sets the tool "back" (negative tool-Z) from the specified location; a negative distance offsets the tool "forward" (positive tool-Z). The offset along the TCS's Z-axis is analogous to shifting a milling machine tool to a clearacne plane. These commands

are very important for insuring that while approaching the location no collision will take place.

6.2.4 DEPART <mm> / DEPARTS <mm>

These commands move the tool (by joint-coordinated or straight-line) the distance given in millimeters, along the current Z-axis of the tool. A positive distance moves the tool back; a negative distance moves the tool forward. These commands are useful specially when the hand grasps an object and departs from alocationin such a way that the object does not collide with the table or fixture from which it is taken. It is also used to avoid collision between the clamps and an object when departing with the clamps open after releasing an object at a given location.

6.2.5 DRIVE <joint>,<change>,<speed>

This command operates the single specified joint changing its joint variable by "change" amount (in degrees) at the "speed" percent of full speed. The joint number, "joint" can be 1,2,..,6. In the simulator, first the joint number is decoded and then the "change" amount followed by the "speed." The simulator defines a new location by adding the amount "change" to the joint variable. It then

uses the joint-coordinated motion and sets the simulation speed as described in Chapter 2.

6.2.6 READY

This command moves the robot into the statically balanced vertical configuration. At this point all the configuration indicators, namely IARM, IELBOW, and IWRIST have positive values. As explained before, the simulator takes advantage of the angle convention and defines zero joint angles in this position. The simulator then creates a location for the zero position and moves the arm to this location using joint-coordinated motion.

6.2.7 DELAY <time>

This command causes the robot motion to stop for the specified period of time.

6.3 HAND CONTROL

The robot tool frequently has the form of some kind of a grasping device, refered to as a hand. Here, it is assumed that the robot has electrically driven proportional hands [13].

6.3.1 OPEN {<hand opening>} / OPENI {<hand opening>}

In the simulator OPEN and OPENI behave in the same fashion, and the hand opening is changed to the "hand opening" percent of fully open position. To open the clamp in the specified amount, the definition data for the reference vertices of the polyhedrons representing the two clamp faces are translated by the user-defined increment to achieve the desired hand opening. Here, it is assumed that no object sticks to the clamp. The detach algorithm explained in section 2.5 is used and the flags are set so that the objects are redefined with respect to the fixed SCS.

6.3.2 CLOSE <hand opening> / CLOSEI <hand opening>

These commands operate in a way similar to OPEN and OPENI commands. Here, the definition data for the reference vertices of the clamp polyhedrons are translated by the user-defined increments, to achieve the desired closing position. As poited out previously, the fine check algorithm is used to see if it is possible to grasp a user-defined workpiece, by checking to see if the LED line segment has intersected any of the polygons belonging to the polyhedrons representing the hand. If it has, the flag is set and the attach algorithm described in section 2.5,

is applied. Confusion may arise when two small objects are placed near one another, or a collision warning was ignored by the user; in either case one may expect the robot to attach both objects to the clamp. However, the simulator only attaches the clamp to the lowest numbered object polyhedron (the object number is determined by its location in the SETUP data file). For efficiency, the simulator does not repeat these steps when the clamp is already closed.

6.3.3 GRASP <hand opening>,{<label>}

In VAL-II this instruction causes the hand to close immediately, and then checks to see if the final opening is less than specified amount.If it is, the program branches to the step specified by the program label. Otherwise, it will close the clamp 100% and continues with the very next step. In the simulator, however, a flag is checked to see if the LED sensor has intersected any of the user-defined polyhedrons. If it has, it closes the clamp by the specified amount, and branches to the step specified by the label. Otherwise, the program execution will continue with the next step. Thus, this instruction simply provides a one-step method of grasping a part and then branching to a different part of the program, something which normally requires two individual steps.

6.4 PROGRAM CONTROL

The following instructions control the sequence in which user program instructions are executed. Thus, they can be used to control the logical flow within user programs.

6.4.1 GOTO <label>

This command performs an unconditional branch to the program step identified by "label." The simulator, first decodes the label and then rewinds the active file searching for the specified step label. If it is found, the program execution will continue from that step. Otherwise, an error message is displayed, and the program execution will continue with the program step following the GOTO statement.

6.4.2 CALL <program>

This command temporarily suspends the execution of current program, and execution continues at the first step of the indicated user program, which is then considered a subroutine. In the simulator, this is done by simply opening the file "program".DAT while the active program is already opened. Execution automatically returns to the

current program when a RETURN instruction is executed in the subroutine.

6.4.3 RETURN

Terminates execution of the current subroutine and resumes execution of the last-suspended program at the step following the instruction which caused the subroutine to be invoked.

6.4.4 PAUSE <MESSAGE>

This command causes an executing VAL-II program to temporarily stop execution. After a PAUSE command is executed and the active program stopped, for implementation, a user may enter any VAL-II command. The simulator will decode the user input, and accepted commands are implemented until the PROCEED command is encountered, at which time control is returned to the VAL-II progrm.

6.4.5 STOP <MESSAGE> / HALT <MESSAGE>

These commands terminate the execution of the user program unless more loops are to be completed, in which case, the control will return to the first step of the

program. It can be said that the STOP instruction marks the end of a VAL-II program execution pass. In the PUMA controller, however, the HALT instruction terminates the program execution regardless of any number of loops remaining to be completed. In the simulator, this command is regarded in the same manner as the STOP command. These commands may be followed by informative messages which are printed for the user on the screen.

6.4.6 DO ... UNTIL

This structure provides a way to control the execution of a group of instructions based on a control expression. The syntax for the DO structure is as follows:

```
DO
        {<group of instructions>}
UNTIL <logical expression>
```

In the simulator, the group of instruction steps are executed, then the UNTIL statement is encountered, and the logical expression is evaluated. If the logical condition is not satisfied, the program is rewinded and the DO statement is found and the program execution continues at the first step following the DO statement. Otherwise, the very next step following the UNTIL statement is executed. Note that, the group of instructions within the DO

structure is always executed at least one time (Appendix A).

6.4.7 IF ... THEN ... ELSE ... END

This form of IF instruction provides a means for conditionally executing a group of instructions, or alternating groups. The complete syntax is:

```
IF <logical expression> THEN
        <first group of steps>
ELSE
        <second group of steps>
END
```

In the simulator, when IF command is encountered, the logical expression is examined. If it is satisfied, the first group of steps is executed, and the control will be transfered to the first step following the END step. Otherwise, the second group of instructions is executed, and the control will transfer to the first step following the END statement. The simulator first decodes the variable names followed by the mathematical condition used. It then opens the file CHECK.DAT to see if the first variable name exists in the list, and if it does it obtains its value. The same thing is done for the second variable name. It then compares the two values obtained thus far, and decides whether the logical expression holds or not; and based on

the result the corresponding group of instructions is executed. In any case, if either one of the variable names was not declared before an error message is displayed and the control is transfered to the very first line following the END step (Appendix A).

6.4.8 WHILE ... DO ... END

This structure provides another means for executing a group of instructions until a control condition is satisfied. The complete syntax for the WHILE structure is:

```
WHILE <logical condition> DO
        {<group of instructions>}
END
```

In processing the WHILE structure, first the logical expression is evaluated. If the logical condition is satisfied, the group of steps is executed and the control is transfered to the step containing the WHILE command to re-exam the logical condition. However, when the logical condition is not satisfied, the program execution will continue at the first instruction after the END step. In the simulator, the variable names and the mathematical condition are first decoded. Then, the file CHECK.DAT is opened to examine the validity of the variable names. If

the names appear in the list, their corresponding values are immediately determined. Then, the logical condition is examined and results obtained. Whenever, anyone of the varibale names does not appear in the file CHECK.DAT, an error message is displayed, and the WHILE structure is ignored (Appendix A).

6.4.9 EXECUTE <program>,<loops>

This command causes the specific program to be executed "loops" times. In the simulator, upon decoding the EXECUTE command the file "program".DAT is opened and the content of it executed step after step. Whenever, a STOP or HALT command is encountered, an execution loop is terminated. Just before opening the "program" data file, the interrupt flag is set, so as to enable the user to abort the program execution without leaving the VAL-II SIMULATOR environment. As the commands are processed, they are read, decoded and implemented,one by one, each command is scrolled on the display terminal, for evaluation and comparison purposes with the actual PUMA robot's behavior. If the number of loops is negative, the PUMA controller executes the VAL-II program indefinitely. However, in such a case, the simulator repeats the program 1000 times.

6.4.10 ABORT

This command terminates execution of the active robot program, immediately after completion of the step currently being simulated. The simulator does not accept this command, but has the same capability. In the simulator, a display terminal is used to simulate the behavior of the PUMA upon execution of VAL-II commands. Hence, to simulate the effects of an ABORT instruction, one may depress the "CTRL" and "C" keys, simultaneously. This will interrupt the active program execution and sets the corresponding flag which is checked before reading and decoding each VAL-II command from the file. Interrupting a program execution by the use of "CTRL" and "C" keys preserves the user-defined locations and setups.

6.5 CONFIGURATION CONTROL

For an anthropomorphic manipulator arm (i.e. an arm with solely rotating joints and redundant degree of freedom) like a PUMA 600, most points in its wotk space may be reached by specifying one of the eight possible spatial configurations [13]. Normally, the robot remains in the default configuration, those set upon program initialization, until the user requests for a different configuration, or when a READY instruction is executed.

The first two configuration indicators, RIGHTY and LEFTY, request a change in the robot configuration so that the first three joints of the robot resemble a human's right or left arm,respectively. The second two, ABOVE and BELOW, control the configuration so that the "elbow" of the robot is pointed up (ABOVE), or down (BELOW) (Figure 2-8). The last two, NOFLIP and FLIP, change the range of operation of joint 5 (wrist rotation) to positive (NOFLIP) or negative (FLIP) angles. This is the difference between the clamp pointing upward or downward.

In the simulator, these commands are being implemented by setting the sign of the configuration indicators IARM, IELBOW and IWRIST, for the three joints in question. Positive values for the configuration indicators are assigned for RIGHTY, ABOVE, NOFLIP; negative values correspond to LEFTY, BELOW, and FLIP. It must be kept in mind that, joint angles obtained by Lee's approach to the inverse kinematic solution [24] is effected by changing these configuration indicators.

6.6 TRAJECTORY CONTROL

6.6.1 SPEED <percent>

The speed of the arm is set by this command. In joint-coordinated motion, this command sets the rotation time for

the largest joint angle difference, namely for a joint which moves farthest. In straight-line motion it sets the clamp speed. In the simulator the increment size used by joint-coordinated and straight-line motions is set by the SPEED command. If the requested speed is less than 10% or more than 100% of maximum speed, the speed is automatically set to 10% or 100% of maximum speed, respectively. Also, when the SPEED command is decoded, the speed dependent error margin is aslo calculated (the speed dependent error margin was discussed when we talked about coarse collision check algorithm).

6.7 ASSIGNMENT INSTRUCTION

Implicit assignment instructions are aslo simulated in the VAL-II SIMULATOR. For example, a variable "ROW" may be declared to have a value of 10, by typing "ROW=10". Variable names may be up to 9 characters long. In the simulator, whenever this situation is encountered, the file CHECK.DAT is opened and its contents are checked. If the variable name already exists, its corresponding value will be updated. Otherwise, the new name will be added to the list of file CHECK.DAT, and its value stored. Also, other forms of assignment statements such as: "ROW=ROW+X", or "ROW=ROW-Y" may be included in the VAL-II programs.

6.8 MISCELLANEOUS COMMANDS

6.8.1 ; {<character string>}

This provides a "comment" line in a VAL-II program. The comment lines are used only for the programmer's benefit, and are ignored when a program is executed.

6.8.2 TYPE {<character string>}

This command operates more or less in the same way as the ";" (remark) command; in that it provides a comment line in a VAL-II program, and the message which appears following this statement is ignored by the simulator, and is only for the programmer's benefit.

6.8.3 TYPE {<variable name>}=

Upon execution of this command, the information described by the output specification is displayed on the terminal. In the simulator, the information is limited to the values of various variable names declared during a VAL-II session. When this command is executed, the file CHECK.DAT is opened and the entries in the list read, one by one. If the variable name exists in the list, its last

declared value is retrieved, and its last declared value is displayed. Otherwise, an error message is displayed. Note that, if the " = " sign is omitted, the simulator will regard this command as a simple TYPE statement explained in section 6.8.2.

6.8.4 STATUS

Upon execution of this command, status information for the system, and the user robot control program being executed is displayed. In the simulator, this includes the speed of the robot set by the last SPEED command, the number of program execution loops completed and those still remaining.

6.8.5 HELP

This command provides on-line help for the VAL-II SIMULATOR users. Upon execution of this command the HELP MENU appears on the screen. The operator is then asked whether he needs more information on a specific topic. If the answer is "yes", the user will then be asked to input the topic for which help is needed. The user input is then decoded and a file called HELPER.DAT, which contains a list of all the help topics, is opened. The user input is checked against individual entries in this list. Whenever,

the user input matches one of the entries in the file, a pointer is set and the corresponding information is displayed on the screen. A flow chart is provided for better understanding of how this program operates in Appendix A. The following information is provided for each topic: 1) the specific command syntax; 2) a description of the operation performed by the command; 3) an indication of when the command can be used. Help is available for all VAL-II commands and some of the Monitor commands. Since, it is very difficult to memorize all the entries in the HELP MENU, the user is frequently asked whether it is necessary to display the HELP MENU or not.

CHAPTER 7

7. CASE STUDY

In order to explain how the simulator may be used, a simple part processing task is shown, in a step by step manner. It will be explained how the parts which have already been created using a UNIGRAPHICS-II system, were transfered to VAL-II SIMULATOR environment, through IGES interface. Also, preparation of location data and instructions for VAL-II SIMULATOR use are given.

7.1 PART TRANSFER

Part files may be transfered to the VAL-II SIMULATOR environment either via IGES interface, or through POLYGON [19] interface. It is assumed that the parts have already been created using a CAD system capable of 3D modeling. In order to use IGES interface, the part file data format must first be converted to IGES format via means provided by the individual CAD system. Once the file is written in IGES format and the file "program.IGS" is generated, one can start a VAL-II session. After the copy right information is displayed, the following menu is displayed:

I_GES FILE

U_GII FILE

E_XISTING FILE

INPUT "I", "U", OR "E":

Except in the Eitor and VAL-II modes, most of the commands only require single key inputs from the user. In this example, only IGES interfaceis used, hence, the input should be "I." This requests the initialization of the section of the package which interactively reads in the "program.IGS", and places the parts at the origin of the user specified coordinate axes. Then, the following question will be displayed on the screen:

ENTER IGES FILE NAME TO BE READ:

The name of the IGES file created is input, "program.IGS". The IGES file is then opened and read in, and converted to a format which may then be read and used by VAL-II SIMULATOR. This file will have a <.SET> extension. This file may have a different name from the IGES file. For this reason, the user is asked to input the desired setup file name:

ENTER SET-UP FILE NAME:

This file will contain all the geometric data, in a format which can be used by the VAL-II SIMULATOR. In order to successfully complete writing this file, the following information is required: a) the number of workpieces in the work-cell, b) the units of the workpieces, which could

either be in millimeters (MM), or in inches (IN), and c) the location of the of the reference coordinate system in which the data have been defined. This information is input by answering the following questions:

ENTER THE NUMBER OF WORKPIECES IN CELL:

ENTER UNITS OF WORKPIECES (MM/IN):

ENTER COORDINATE AXES SYSTEM (W/S):

If the data are input properly, the SIMULATOR will write the file and a message is displayed, informing the user that the file xxx.SET has successfully been written. At any point if a wrong answer is input, an error message will be displayed, and the user asked, whether or not he wishes to continue the part transfer. The part(s) will be placed at the origin of the coordinate axes system specified by the user. A setup file corresponding to our example is shown in Figure 7-1. The first line contains the number of user-defined objects in the work-cell, in the example 14. The object coordinates may be in millimeters or inches. The second line specifies the units of workpieces; either "MM", or "IN" is used. The third line, shows the coordinate system with respect to which the object data is defined. A "W" will designate the WCS, and a "S" will specify the SCS. Following the third line of each setup file, lines of data, one corresponding to each user-defined object appear to indicate each object's size; this is specified by giving the number of vertices belonging to one polyhedron. The

```
14
MM
W
1.8
          440.00    -50.00   -320.00
            0.00      0.00      0.00
            0.00      0.00     70.00
            0.00     30.00     70.00
            0.00     30.00      0.00
           50.00      0.00      0.00
           50.00      0.00     70.00
           50.00     30.00     70.00
           50.00     30.00      0.00
1.8
          440.00    105.83   -320.00
            0.00      0.00      0.00
            0.00      0.00     70.00
            0.00     30.00     70.00
            0.00     30.00      0.00
           50.00      0.00      0.00
           50.00      0.00     70.00
           50.00     30.00     70.00
           50.00     30.00      0.00
1.8
         -463.51    500.00   -320.00
            0.00      0.00      0.00
            0.00      0.00     40.00
            0.00    600.00     40.00
            0.00    600.00      0.00
          500.00      0.00      0.00
          500.00      0.00     40.00
          500.00    600.00     40.00
          500.00    600.00      0.00
1.8
         -463.51    500.00   -320.00
           40.00      0.00      0.00
           40.00      0.00   -250.00
           40.00     40.00   -250.00
           40.00     40.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     40.00   -250.00
            0.00     40.00      0.00
1.8
         -463.51   1060.00   -320.00
           40.00      0.00      0.00
           40.00      0.00   -250.00
           40.00     40.00   -250.00
           40.00     40.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     40.00   -250.00
            0.00     40.00      0.00
1.8
           -2.51   1060.00   -320.00
           40.00      0.00      0.00
           40.00      0.00   -250.00
           40.00     40.00   -250.00
           40.00     40.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     40.00   -250.00
            0.00     40.00      0.00
1.8
           -2.51    500.00   -320.00
           40.00      0.00      0.00
           40.00      0.00   -250.00
           40.00     40.00   -250.00
           40.00     40.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     40.00   -250.00
            0.00     40.00      0.00
1.8
          595.00   -600.00   -320.00
            0.00      0.00      0.00
            0.00      0.00    -20.00
            0.00    800.00    -20.00
            0.00    800.00      0.00
         -250.00      0.00      0.00
         -250.00      0.00    -20.00
         -250.00    800.00    -20.00
         -250.00    800.00      0.00
1.8
          575.00   -600.00   -320.00
           20.00      0.00      0.00
           20.00      0.00   -250.00
           20.00     20.00   -250.00
           20.00     20.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     20.00   -250.00
            0.00     20.00      0.00
1.8
          575.00    200.00   -320.00
           20.00      0.00      0.00
           20.00      0.00   -250.00
           20.00     20.00   -250.00
           20.00     20.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     20.00   -250.00
            0.00     20.00      0.00
1.8
          345.00    200.00   -320.00
           20.00      0.00      0.00
           20.00      0.00   -250.00
           20.00     20.00   -250.00
           20.00     20.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     20.00   -250.00
            0.00     20.00      0.00
1.8
          345.00   -600.00   -320.00
           20.00      0.00      0.00
           20.00      0.00   -250.00
           20.00     20.00   -250.00
           20.00     20.00      0.00
            0.00      0.00      0.00
            0.00      0.00   -250.00
            0.00     20.00   -250.00
            0.00     20.00      0.00
1.8
          495.00    400.00   -320.00
            0.00      0.00      0.00
            0.00      0.00   -150.00
          176.78    176.78   -150.00
          176.78    176.78      0.00
         -176.78    176.78      0.00
         -176.78    176.78   -150.00
            0.00    353.56   -150.00
            0.00    353.56      0.00
1.8
          495.00    400.00   -470.00
            0.00      0.00      0.00
            0.00      0.00   -100.00
          176.78    176.78   -100.00
          176.78    176.78      0.00
         -176.78    176.78      0.00
         -176.78    176.78   -100.00
            0.00    353.56   -100.00
            0.00    353.56      0.00
```

Figure 7-1. Example Program Setup data file

arrays have been adjusted so that up to 30 objects, excluding the robot model itself, may be used in each work-cell, with hidden-surfaces properly removed. To have the collisions checked for all the parts, the dimension of the INDEX array is set to 1100. In our example, each object has 8 vertices, as shown in Figure 7-1. This is shown by specifying the first and last vertex with a comma separating them (1,8). Due to the data format chosen, the first digit should be 1, and the last one an even integer larger than 6. As explained in previous chapters, associated with each object is a reference point (the first corner). The set of data following the number of vertices for the first polyhedron is the coordinates of the reference point, namely the first corner of the first object. Following this line, the coordinates of each vertex is given with respect to this reference point, given in (X,Y,Z) order separated by commas or spaces, corresponding to prespecified coordinate system. As mentioned in Section 2.1, the simulator uses polygons to form convex polyhedron envelopes which represent objects. The vertices representing the front and back faces must be given either in a clockwise, or counterclockwise order, when looking toward the origin along the line of the normal axis. When a counterclockwise convention is used, the magnitude of the coordinates along the normal axis, for the first polygon is larger than that of the second one, and vice versa, when a

clockwise ordering is used.

7.2 PART REPOSITIONING

The user is then directed to the MAIN MENU. Since, the objects are all at the origin, they must be moved to the desired locations in the work-cell to represent the designed layout. To move to MANIPULATION Mode, 'M' (M_ANIPULATION) is pressed on the keyboard. A menu is then displayed which corresponds to manipulation mode using the system defined increments. The user is first asked, the part number to be moved. Parts are numbered 1,2,...,n consecutively, according to their location in the setup file, where n is the total number of parts in the work-cell. After the part number is input, the following menu is displayed:

I_NCREMENTS

K_EYBOARD

R_ETURN

ENTER COMMAND#

A single key input is required. The operator may select to use the KEYBOARD option by pressing 'K' from the keyboard, and then by depressing 'U' (U_PWARD), 'D' (D_OWNWARD), etc. to move the individual part in the desired direction, until the final desired location is reached. If the locations are precisely known, one may prefer to move the individual part

by specifying the proper translational and rotational increments. If the latter is chosen, by pressing 'I', it is then asked whether the user would like to have the coordinate axes display ON or OFF. The user is then asked to input the corresponding translational and rotational increments. The translations and rotations are performed with respect to the SCS and a readout is available along with the part number for reference. Once done with repositioning one part, one may start to move another by answering 'NO' to question which asks whether it is desired to continue moving the current object. In our case the proper translational and rotational increments are input for all the parts so that the work-cell will be displayed as shown in Figure 7-2. The end of a Manipulation session is marked by answering 'NO' to the question which asks whether it is desired to move another part. We want to save the new setup, so we answer 'YES' to question

DO YOU WANT TO SAVE THE NEW SET-UP FILE (Y/N) ?:

and it is then asked, whether it is prefered to use the old setup file name or a new one is to be specified. The file name may contain up to 9 characters and must be followed by a ".SET". The control is then returned to the MAIN MENU.

7.3 LOCATION DATA

The functions of the "Teach Pendant" may be used to

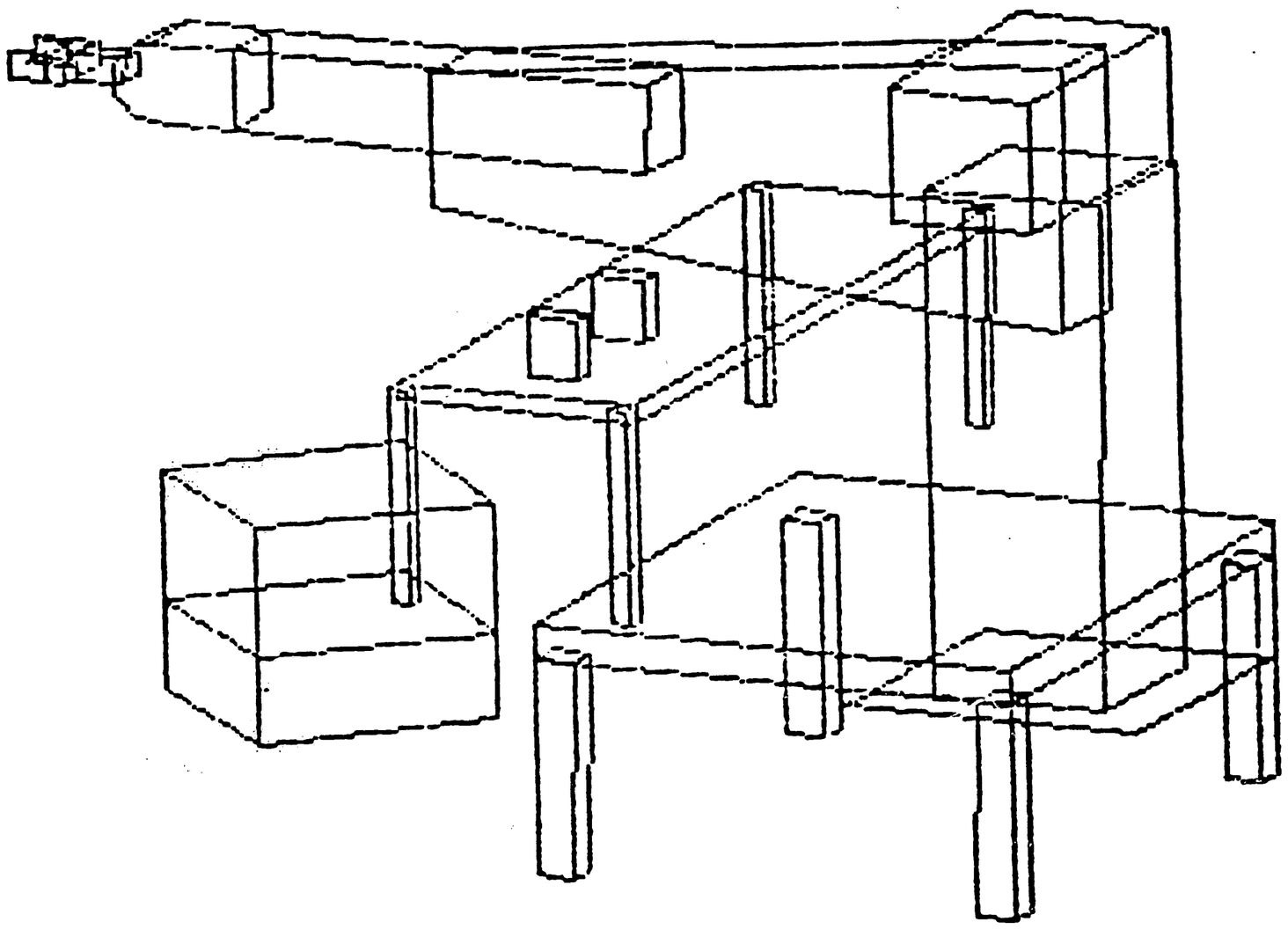


Figure 7-2. Simulator Output Image for Program Process

independently move each link of the PUMA robot by pressing any one of the numbers 1-6 on the keyboard for the corresponding link. This is the exact method, a real industrial robot is taught points in its space [26]. Once a desired location is reached, one may save that location by entering the VAL-II mode and typing " HERE <point name>". There is a quicker method to define and store locations, and that is to write them in a "program.LOC" file, by looking at the positions of the objects defined in the setup file. (However, to take advantage of this method, the user must leave the VAL-II SIMULATOR environment.) Any location data contains the positions and orientation angles. In the file, each location is identified bythe label, POINT, followed by a space and then an appropriate name of up to 9 characters long, a comma, and the position and orientation data separated by commas (Figure 7-3).

7.4 SIMULATOR USE

The simulator may be used in an interactive manner to perform the following tasks:

-to move the end-effector to different locations,

-to orient the end-effector,

-to open and close the gripper,

-to perform repeating tasks,

-to make conditional and unconditional branching or

```
POINT A, 465. , -45. , -245. , 89. , 91. , 0.
POINT B, 465. , 120. , -245. , 90. , 90. , 0.
POINT C, -180. , 600. , -204. 16, 90. , 90. , -90.
POINT D, -100. , 600. , -204. 16, 90. , 90. , -90.
POINT E, 425. , 590. , -220. , 90. , 90. , -45.
POINT F, 425. , 590. , -400. , 90. , 90. , -45.
```

Figure 7-3. Example Program Location data file

jumps.

The order in which the individual instructions are to be executed is critical to simulate a manufacturing application effectively and efficiently. Oftentimes, several iterations are necessary to find the most suitable locations, work-cell setups and the optimum VAL-II program.

A program session may be started with no previous setups or a previous session may be continued. In either case the Editor is used to develop the VAL-II program. To invoke the EDIT Mode, from the MAIN MENU, one must press 'E' (E_DIT). This will cause the Editor Menu to be displayed. Then to create a new program to perform the part processing task, we type EDIT followed by the program name, xxxxx.DAT. If this is a new session the following message will appear on the screen:

INPUT FILE xxxxx.DAT DOES NOT EXIST

If you are continuing development of an existing file the Editor will load your program. For a new session the file is initialized, and one may press 'C' for CONTINUE, and the VAL-II program is then typed in step by step. End of each line is marked by pressing the RETURN key on the keyboard, and the next step may be input after pressing 'C'. Once the VAL-II program editing is completed, the file is saved by pressing 'S' (S_AVE) and the Editor may be exited by typing 'E' (E_XIT), which will transfer the control back to the MAIN MENU. To test the program, we must enter the VAL-II

mode. This specific program is written to pick up parts, process them, and then place them on a table. The processing task is simulated by inserting the part in a part processor.

After the locations are taught to the robot, we enter the VAL-II mode. VAL-II commands are input to visualize the performance of the PUMA. An example program edited in the EDIT Mode appears in Figure 7-4. If any modification is required, we may go back to the EDIT Mode, and delete lines of instructions or add new ones. Figures 7-5 to 7-6 show the simulation of the example program. In these figures one part is picked up (Figure 7-5 a), moved straight up (Figure 7-5 b), moved along straight line for processing (Figure 7-5 c) and inserted in the part processor (Figure 7-5 d). Once processed, it is moved to a location which is offset from the final location (Figure 7-6 a), the part is released and the clamp departed from that location (Figure 7-6 b), and moved to grasp the second part (Figure 7-6 c). Once the program and locations have been verified on the simulator, they may be down-loaded to PUMA controller for final testing and implementation.

7.5 OBSERVATIONS

Since, the effects of changes in payloads due to clamp velocity and acceleration are not simulated, and also, the

```
; PROGRAM : PROCESS
; THIS PROGRAM SIMULATES A PART PROCESSING APPLICATION
READY
LEFTY
ABOVE
APPRO A,-50
TYPE PICK UP THE FIRST PART
MOVE A
CLOSEI 50
DEPARTS 100
MOVES E
MOVES F
TYPE PROCESS THE FIRST PART
DELAY 1
DEPARTS 200
APPROS C,-50
MOVES C
OPENI 100
DEPARTS 100
APPROS B,-50
MOVES B
TYPE PICK UP THE SECOND PART
CLOSEI 50
DEPARTS 100
MOVES E
TYPE PROCESS THE SECOND PART
DELAY 1
SHIFT E,0,0,-200
MOVES E
APPROS D,-50
MOVES D
OPENI 100
DEPARTS 100
READY
STOP
```

Figure 7-4. Example Program Command data file

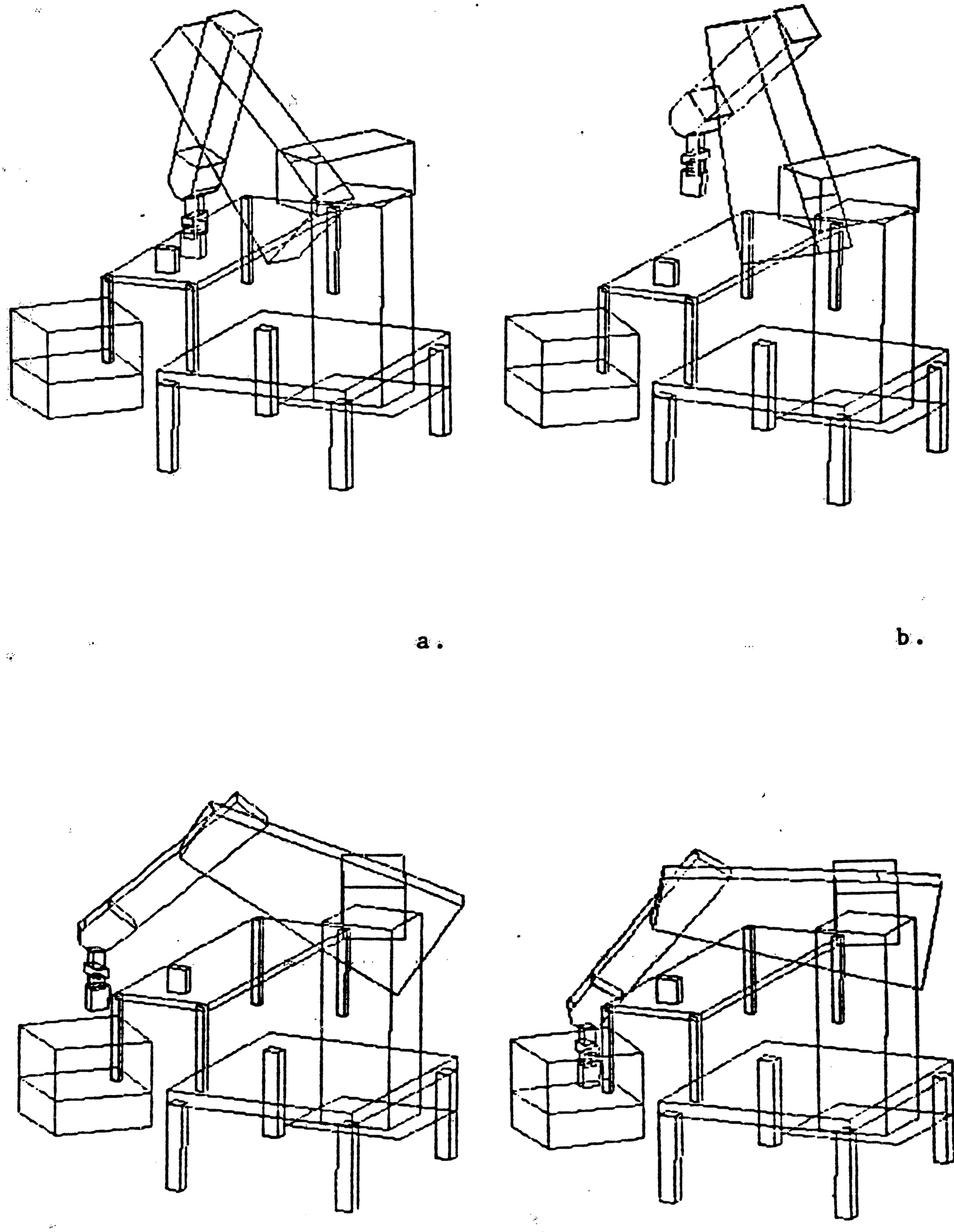
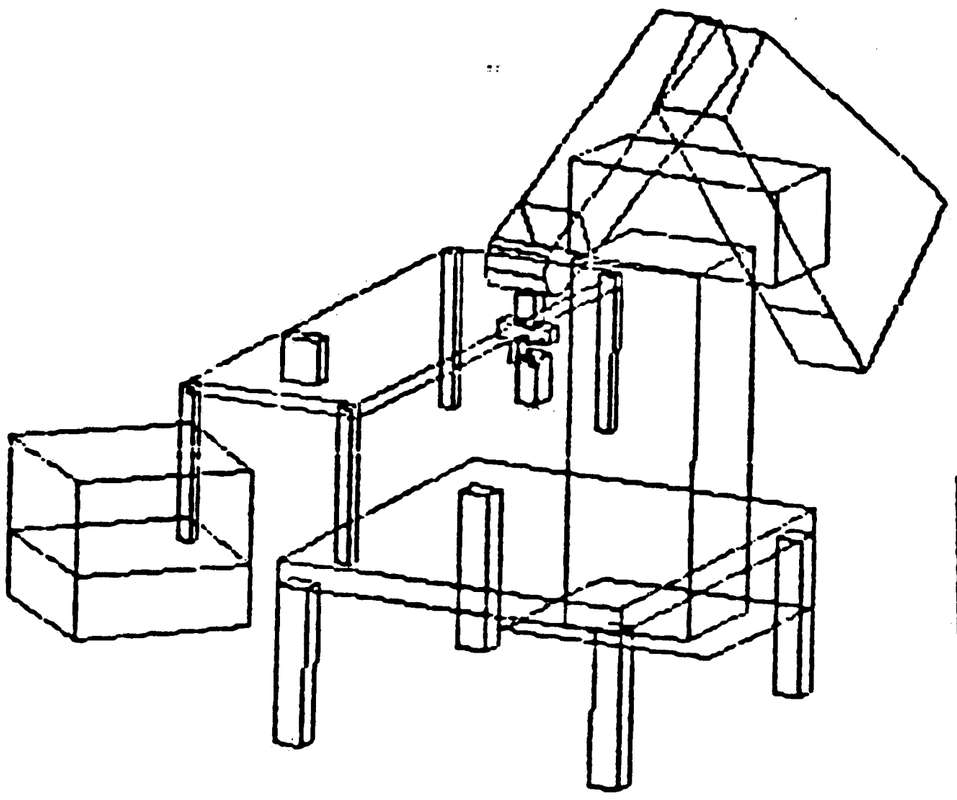
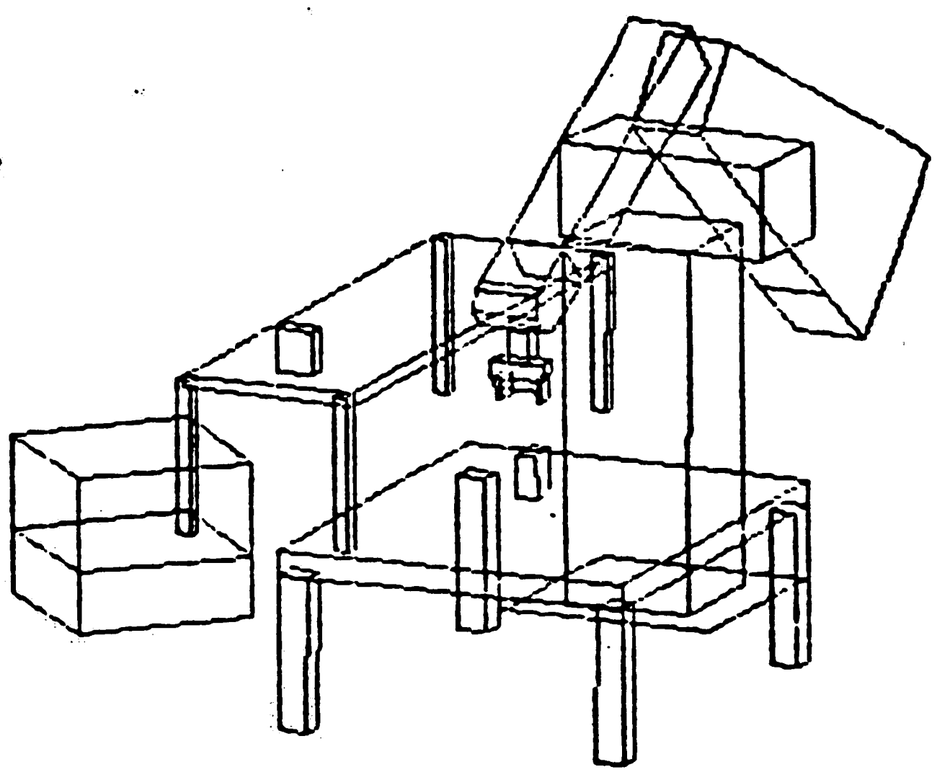


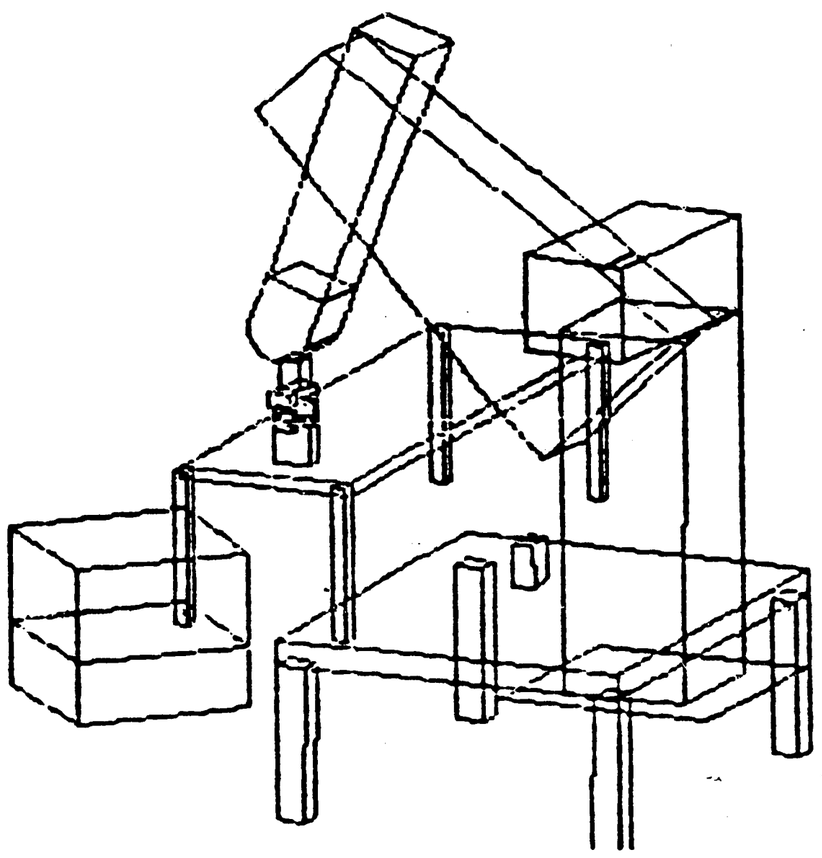
Figure 7-5. Simulator Output Images for Example



a.



b.



c.

Figure 7-6. Simulator Output Images for Example 110

location data in the simulator and those in the real work-cell environment are different, the successful simulation does not guarantee foolproof VAL-II program implementation. One serious limitation of the graphical simulation is that the third link, FOREARM, is shown symmetrical and does not show the assymetrical taper found on the actual PUMA. For this reason the clamp is not located precisely where predicted. This may cause problems in stacking parts.

As discussed before, the simulation speed does not directly correspond to actual speed. In the actual VAL-II Language setting the speed to 25% of the full speed prescribes a safe program execution and corresponds to approximately full simulation speed. However, if in the simulator, the robot is run at full speed, a part may appear to be placed in the precise specified location. But, in the real work environment the part may reach its final position too quickly and when it is placed in the specified location, it is actually "thrown". Since the simulator does not simulate the system dynamics, it is recommended that the simulation speed be set so that it corresponds to a 25% PUMA speed.

CHAPTER 8

8. SUMMARY

Due to great technological advancements in recent years, robot language simulators, like the VAL-II SIMULATOR are being considered A most important assist in the implementation of robot systems. These simulators are used for off-line programming of robots. In addition the simulators may be used as instructional instruments and help in evaluating program sequences, as well as designing various work-cell layouts. In manufacturing, they may be used for off-line programming, hence, minimizing the production-line down time.

VAL-II SIMULATOR is menu-driven with an on-line help feature for all control levels. VAL-II SIMULATOR utilizes the basic concepts and fundamental structure of VAL SIMULATOR and offers more advanced features in terms of programming control and finer graphics display. Information regarding configuration indicators is available to user at any time. Arithmetic and logical expressions are available for decision making, for performing repeating tasks, and for making unconditional or conditional branching or jumps. Collisions are detected automatically and a simple sensor

interface is provided. User-defined geometries may interactively be retrieved from other CAD data bases using the International Graphics Exchange Standard (IGES) and then easily positioned in the PUMA's work environment. With these features the simulator can be used to design and experiment with a variety of work-cell setups, investigating assembly tasks, and to develop various VAL-II programming possibilities. Once a work-cell is created and the VAL-II program is developed and tested with the simulator, the VAL-II program may be down-loaded directly to the PUMA controller for final testing and verification.

8.1 LIMITATIONS AND RECOMMENDED APPROACHES

The simulator does not currently change the configuration indicators if needed. If the values of the user-defined configuration indicators do not agree with the theoretical ones, the motion is halted. Oftentimes, the user does not know whether the path is possible or not. In such a case, an automatic change option would considerably enhance the simulator's performance.

In the Simulator's VAL-II Mode, each input string must first be decoded by the system to recognize the command for implementation. Therefore, there is a short stop at each instruction.

The collision check algorithm requires a great deal of

computations to precisely inform the user of unwanted events, hence, resulting in slower simulation display.

The simulator uses extruded wire-frame polyhedrons which necessitates defining envelopes around the objects rather than use the actual geometries.

It must also be kept in mind that VAL-II SIMULATOR is not a robot emulator. The effects of gravity or inertia or changing payloads, or dynamic effects such as accelerations and decelerations are not simulated.

Very limitedsensory interface is available.

8.2 FUTURE EFFORTS

VAL-II SIMULATOR has been written in FORTRAN 77, and consists of a series of inter-related subroutines, which allows for easy modification, or extension of individual routines without involving major changes in the rest of the program.

The FRAME and INVERSE commands may be added by introducing compound transformations.

For more programming control, IF-SIG commands may be included. For the IF-SIG command to be practical, along with the robot some hardware should also be simulated.

Routines may be developed to optimize clamp trajectories.

A rouitne may be developed which would enable the

computation of estimated cycle times.

The simulator in its present form only simulates one robot and one language. A library may be created so that other robots as well as their programming languages could be simulated.

REFERENCES

[1] Leu, M.C. and Mahajan, R., "Simulation of Robot Kinematics using Interactive Computer Graphics," Proceedings of ASEE 1983. Annual Conference, Rochester Institute of Technology, NY, June 1983, pp. 34-39.

[2] Leu, M.C. and Park, S.H.,"Use of Computer Graphics for Robotics Instruction," Proceedings of ASEE 1983. Frontiers in Education Conference, Worcester Polytechnique Institute, MA, October 1983.

[3] Clifton, M.B. and Ochs, J.B.,"An Interactive Computer Graphics Simulation of VAL, Programming Language of the Unimation PUMA robot," Proceedings from the 26th IEEE Computer Society International Conference, Arlington, VA, September 25-29, 1983, pp. 193-200.

[4] Derby, S.J.,"Kinematic Elast-Dynamic Analysis and Computer Graphics Simulation of General Purpose Robot Manipulators," Ph.D. Thesis, Rensselaer Polytechnique Institute, Troy, New York, August 1981.

[5] Kretch, S.J.,"Robotic Animation," Proceedings of the 2nd International Computer Engineering Conference, ASME Computer Engineering Division, August 15-19, 1982, pp. 87-92.

[6] Meyer, J.,"An Emulation System for Programmable Sensory Robots," IBM Journal of Research and Development, Vol. 25, No. 6, Nov. 1981, pp. 955-962.

[7] Soroka, B.I.,"Debugging Robot Programs With a Simulator," Proceedings SME Autofact West, CAD/CAM VIII, Anaheim, CA, November 1980.

[8] Tilove, R.B.,"Extending Solid Modeling Systems for Mechanism Design and Kinematic Simulation," GM Research Publication GMR-4246, Jan. 1983.

[9] Wesley, M.A., Lozano-Prez, T., Lieberman, L.I., Lavin, M.A., and Grossman, D.D.,"A Geometric Modeling Systems for Automated Mechanical Assembly," IBM Journal of Research and Development, Vol. 24, No. 1., Jan. 1980, pp. 64-74.

[10] Derby, S.J.,"Computer Graphics Robot Simulation Programs: A Comparison," Published in the bound volume "Robotics Research and Advanced Applications," ASME Winter

Annual Meeting, November 1982, pp. 203-212.

[11] "User's Guide to VAL", Unimation, Inc., Danbury, Conn., June 1980.

[12] Clifton, M.B.,"VAL Robot Simulator," M.S. Thesis, Lehigh University, Bethlehem, Penn., May 1984.

[13] "User's Guide to VAL-II", Unimation, Inc., Danbury, Conn., April 1983.

[14] "A Robot Programming System Incorporating Real Time and Supervisory Control: VAL-II"

[15] Ozsoy, T., Ochs, J.B.,"Lehigh University's VS11-3D Graphics Package," Proceeding from 1983 DECUS Meeting, St. Louis, MO, May 26, 1983.

[16] Barnhill, R.E., Riesenfeld, R.F.,"Computer Aided Geometric Design," Academic Press, New York, 1975.

[17] Faux, I.D., Pratt, M.J.,"Computational Geometry for Design and Manufacturing," John Wiley, New York, 1979.

[18] Foley, J.D., VAN DAM, A., FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS, Adison-Westly Publishing Company, Inc., 1982, pp. 505-537.

[19] Oszoy, T., Ochs, J.B.,"An Interactive Modelling Program for the Generation of Planar Polygons for Boundary Type Solid Representation of Wire-Frame Models," Proceedings from the Symposium on Computer Aided Geometry Modeling, NASA Langley, Hampton, VA, April 21, 1983.

[20] Giloi, W.K., INTERACTIVE COMPUTER GRAPHICS, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

[21] Paul, R.P.,"Robot Manipulators: Mathematics, Programming and Control", MIT Press, 1981.

[22] Denavit, J. and Hartenberg, R.S.,"A Kinematics Notation for Lower Pair Mechanisms Based on Matrices," Journal of Applied Mechanics, Vol. 22, ASME Transactions, Vol. 77, 1955, pp. 215-221.

[23] Gaglio, S., Morasso, P., Tagliasso, V., Zaccaria, R.,"Computation of Inverse Kinematics and Inverse Dynamics in Manipulato Arm Control," Proceedings from 11th International Symposium on Industrial Robots, Tokyo Japan, 7. 8. and 9. October 1981.

[24] Lee, C.G.S., Ziegler, M.,"A Geometric Approach in solving the Inverse Kinematics of PUMA Robots," Proceedings from the 13th International Symposium on Industrial Robots and Robots 7, Vol. 2, April 17-21, 1983, pp. 16.1-16.18.

[25] Pieper, D.L.,"The Kinematics of Manipulators Under Force Control," Ph.D. Thesis, Computer Science Department, Stanford University, October 1968.

[26] Mahajan, R. and Mogal, J.S.,"An Interactive Graphic Robotics Instructional Program IGRIP, A Study of Robot Motion and Workspace Constraints"

[27] Jacobs, M.P.,"Off-Line Robot Programming: A Current Practical Approach," Proceedings from Robots 8 Conference, Vol. 1, Applications for Today, Detroit, Michigan, June 4-7, 1984, pp. 4.1-4.11.

[28] Rossol, L.," Technological Barries in Robotics: A Perspective from Industry" Presented at the First International Symposium of Robotics Research, Bretton Woods, New Hampshire, August 28- September 2, 1983.

APPENDIX A. Flow Charts

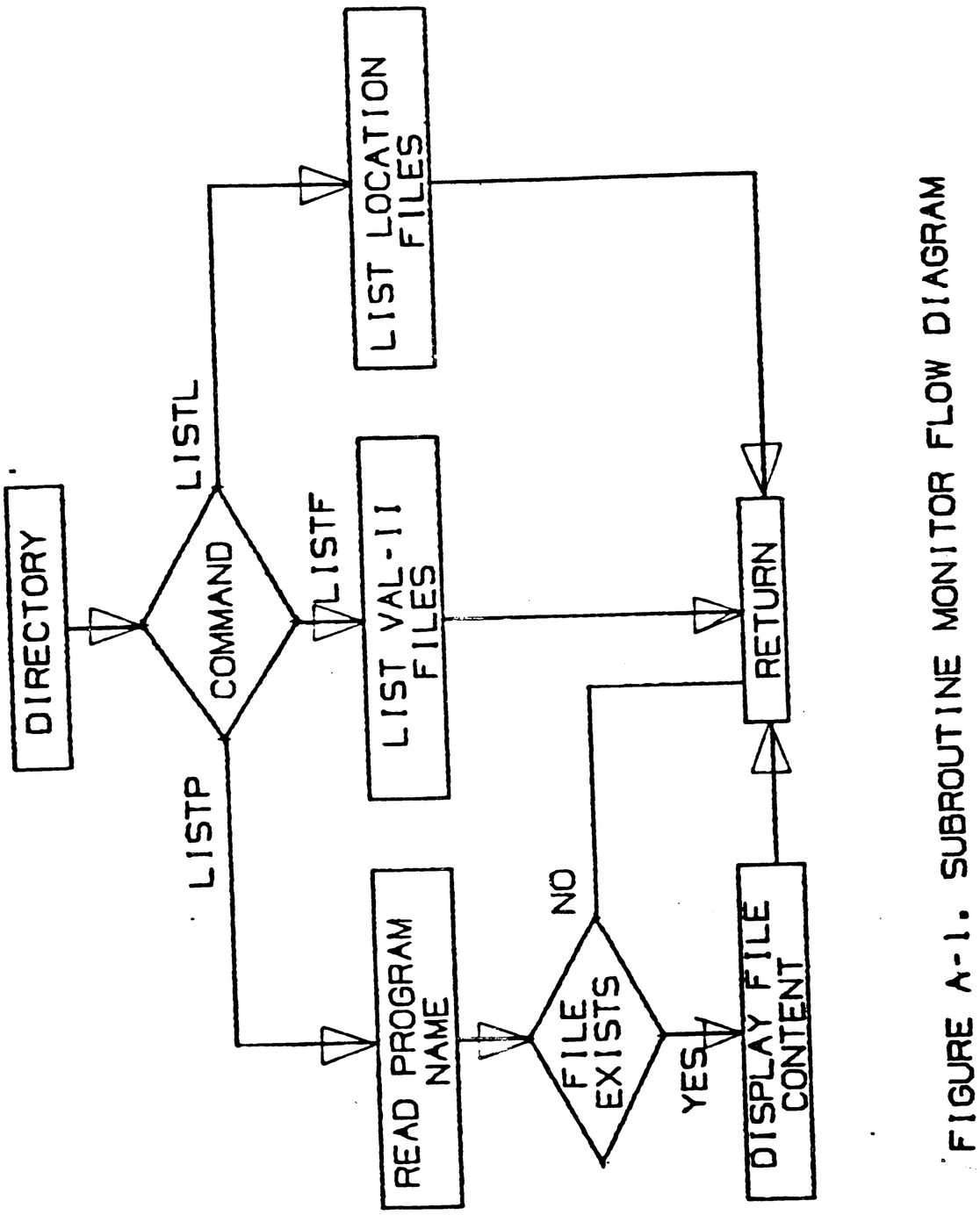


FIGURE A-1. SUBROUTINE MONITOR FLOW DIAGRAM

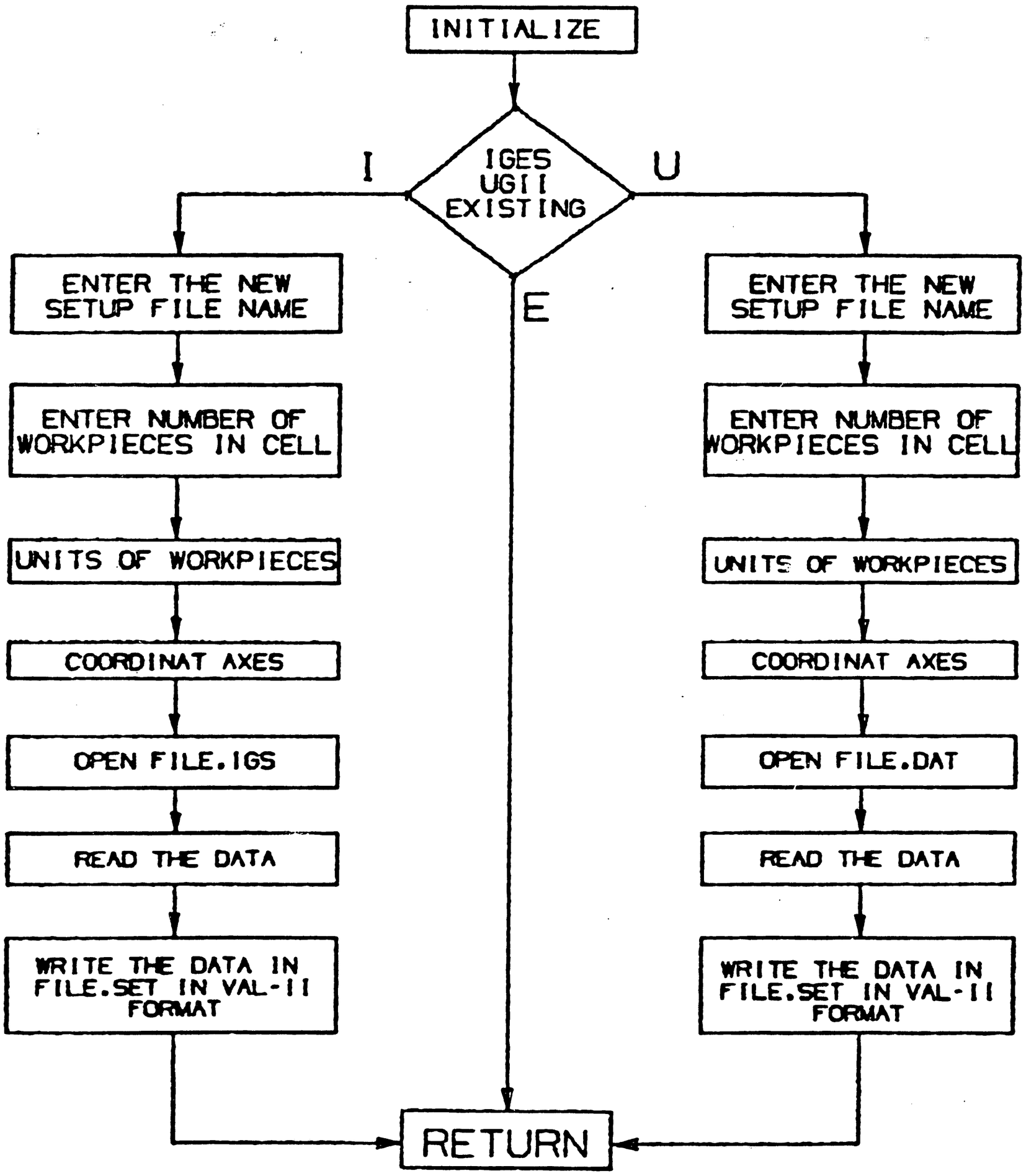


FIGURE A-2. SUBROUTINES IGESCONV & UGIICONV FLOW DIAGRAM

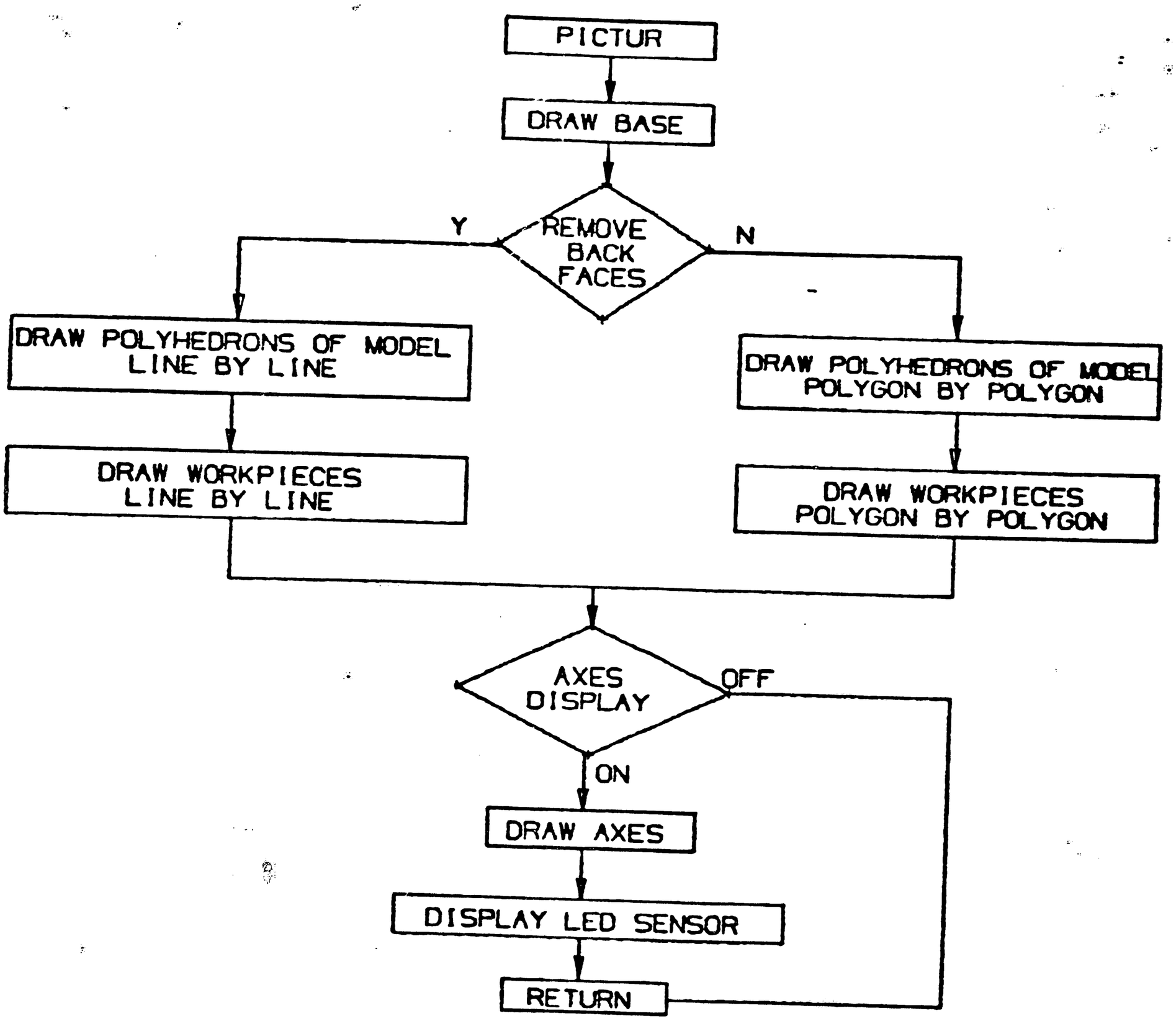


FIGURE A-3. SUBROUTINE PICTUR FLOW DIAGRAM

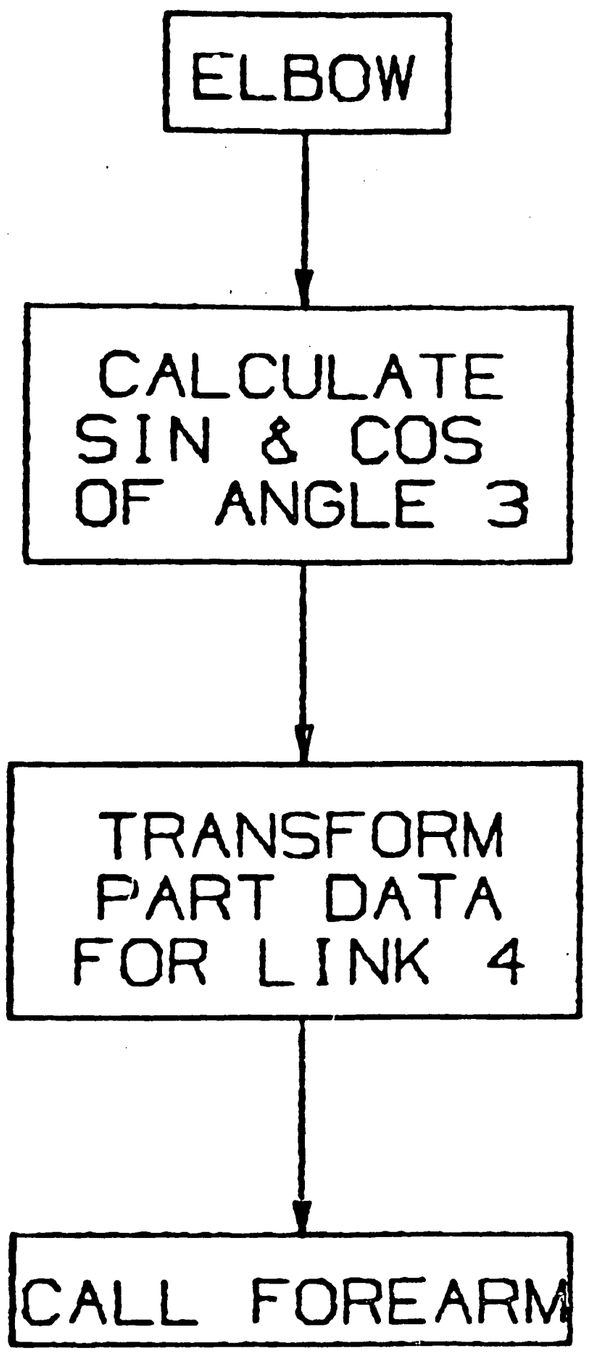


FIGURE A-4. SUBROUTINE ELBOW FLOW DIAGRAM.

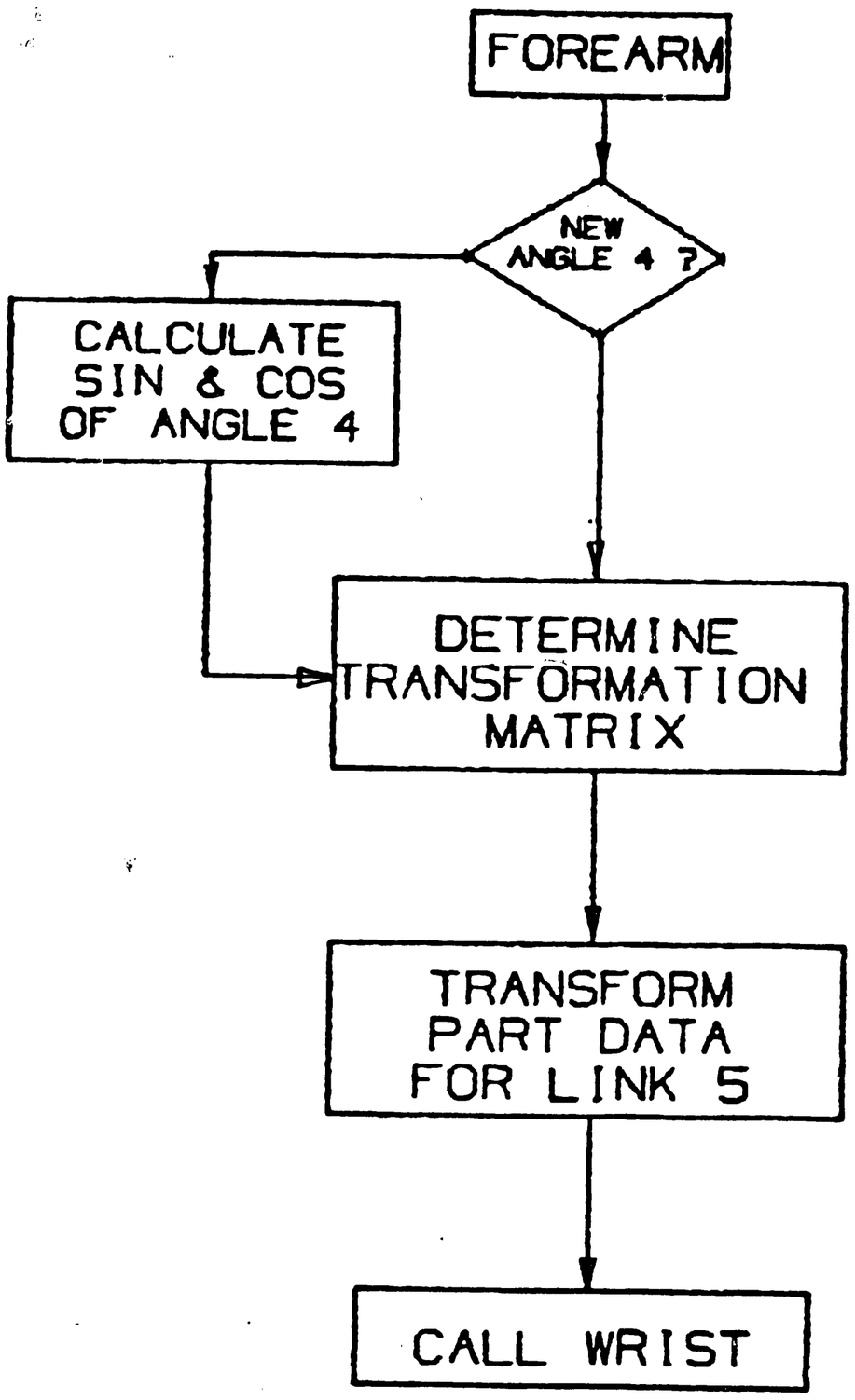


FIGURE A-5. SUBROUTINE FOREARM FLOW DIAGRAM

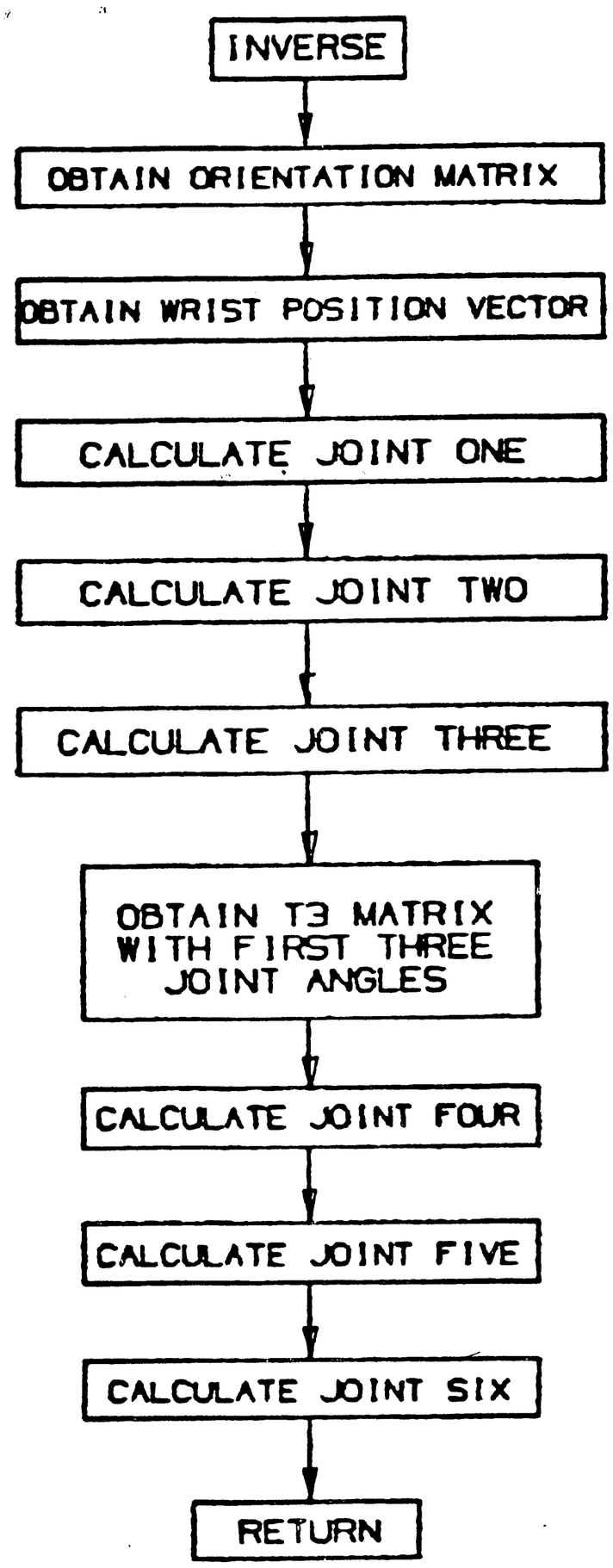


FIGURE A-6. SUBROUTINE INVERSE FLOW DIAGRAM

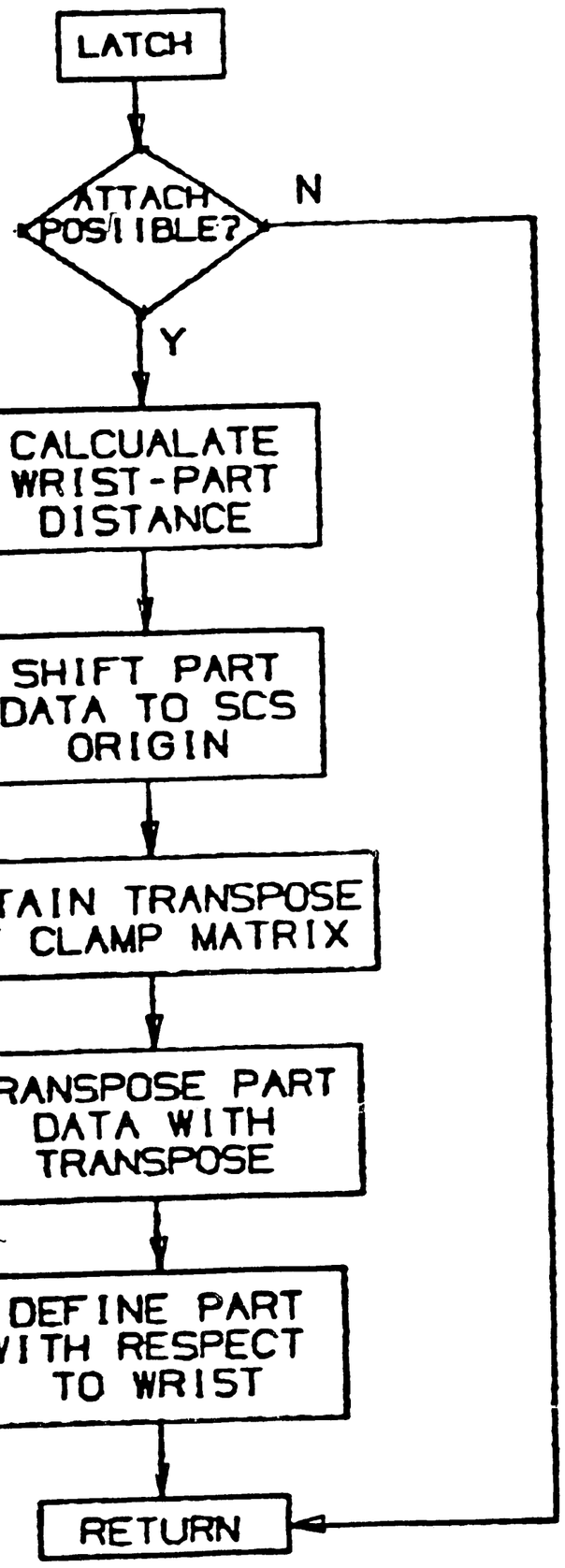


FIGURE A-7. SUBROUTINE LATCH FLOW DIAGRAM

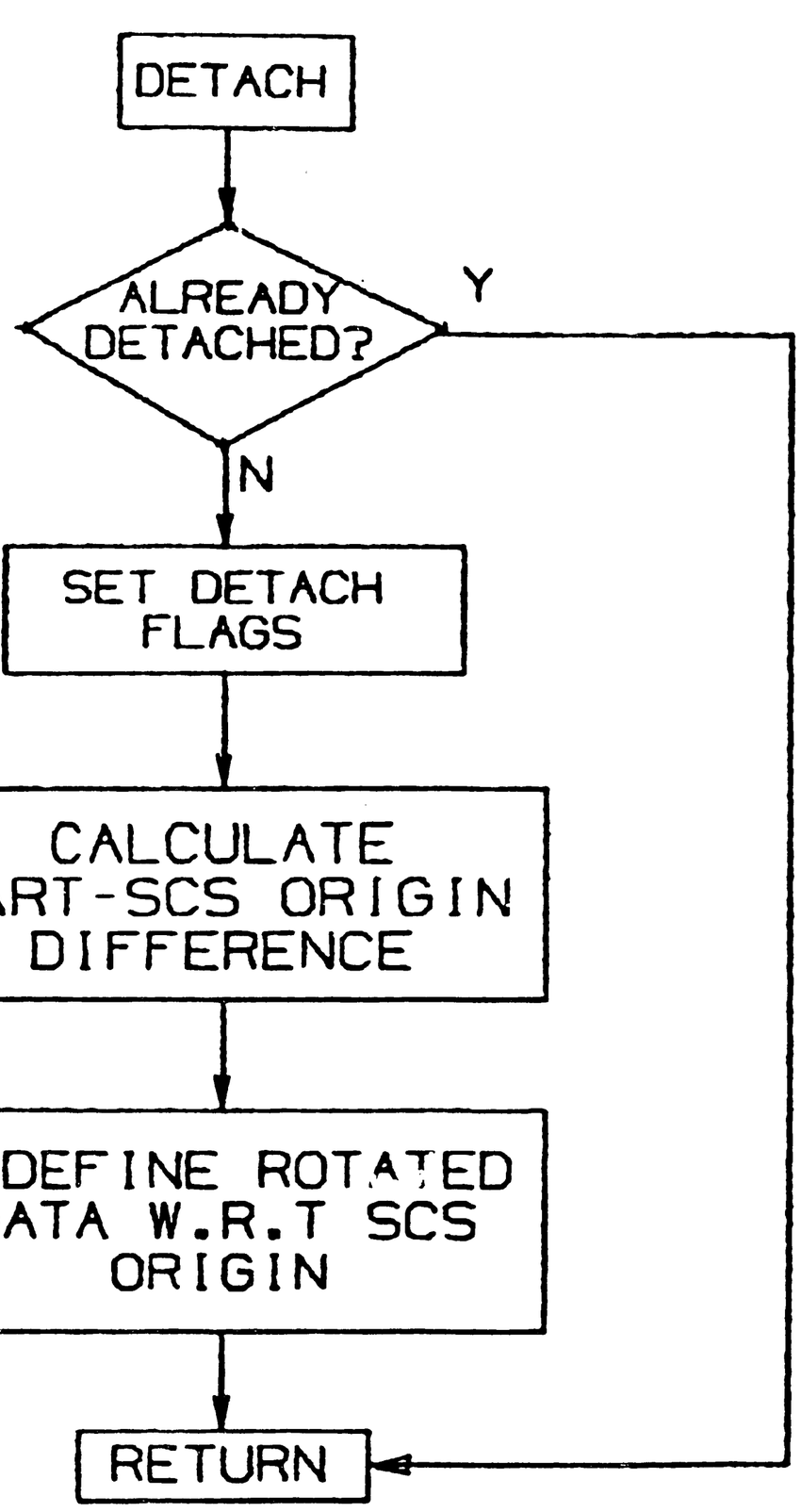


FIGURE A-8. SUBROUTINE DETACH FLOW DIAGRAM

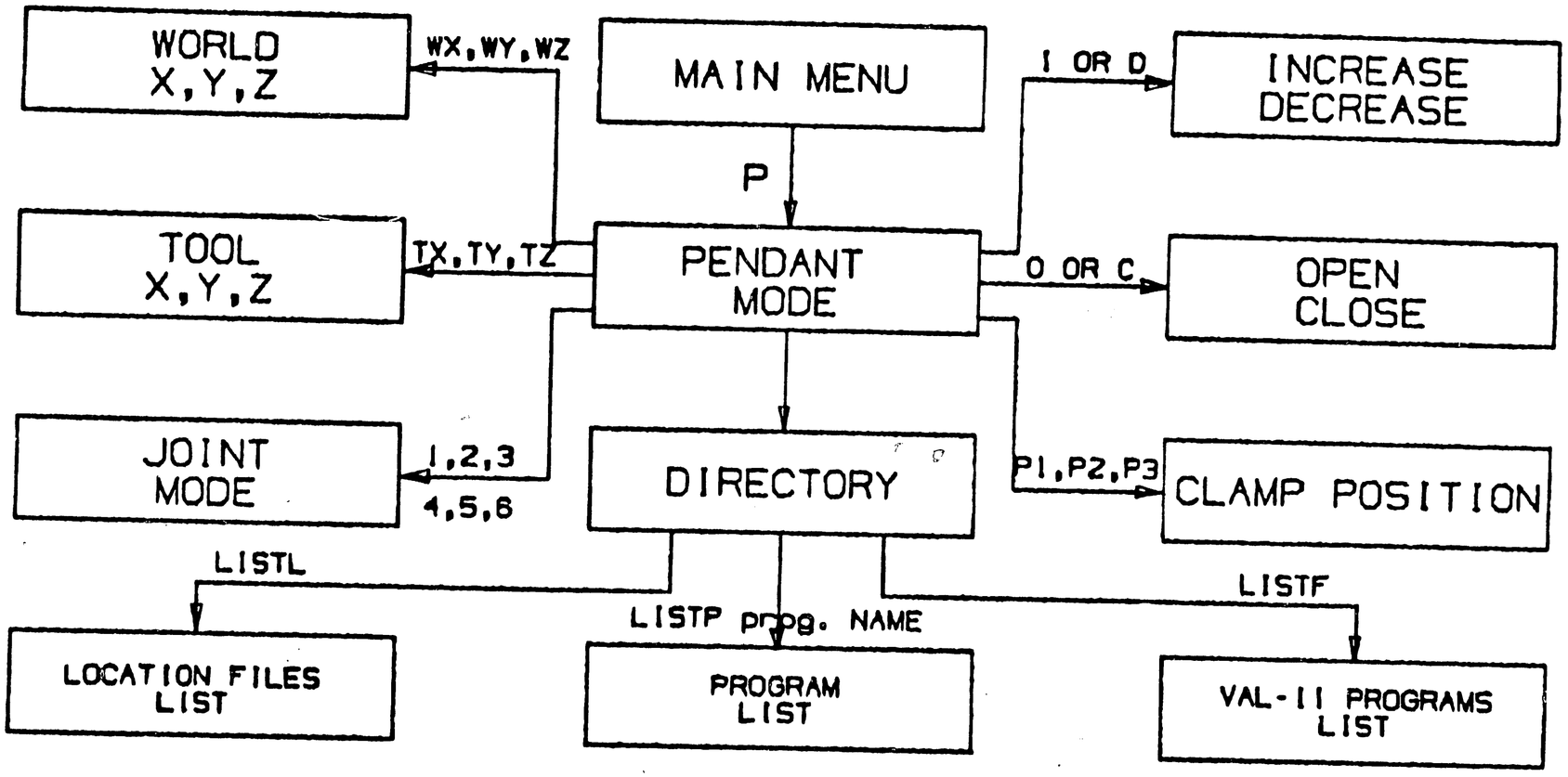


FIGURE A-9. PENDANT MODE FLOW DIAGRAM

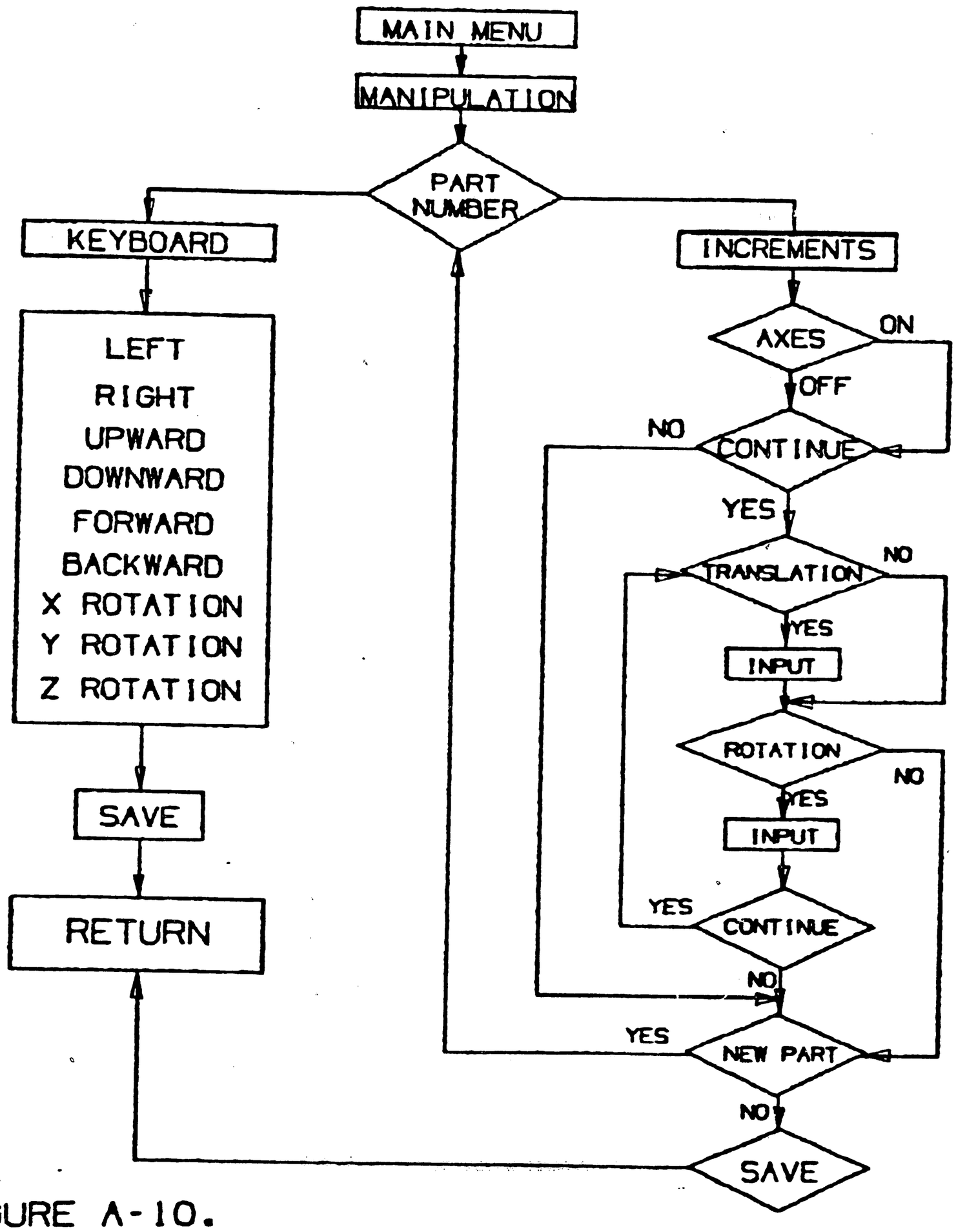


FIGURE A-10.

SUBROUTINES MOVERINC & MOVERKEY FLOW DIAGRAMS

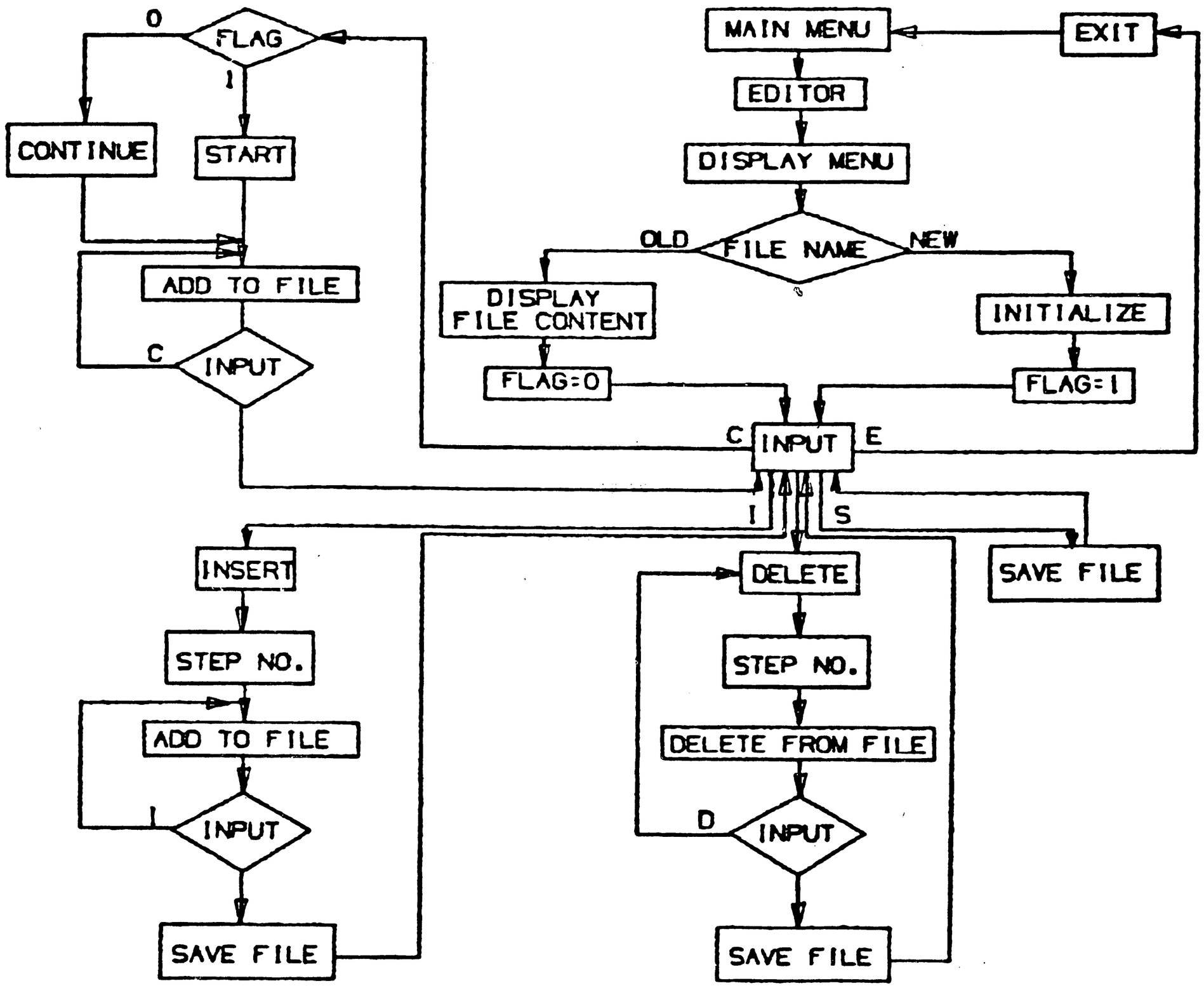


FIGURE A-11. SUBROUTINE EDITOR FLOW DIAGRAM

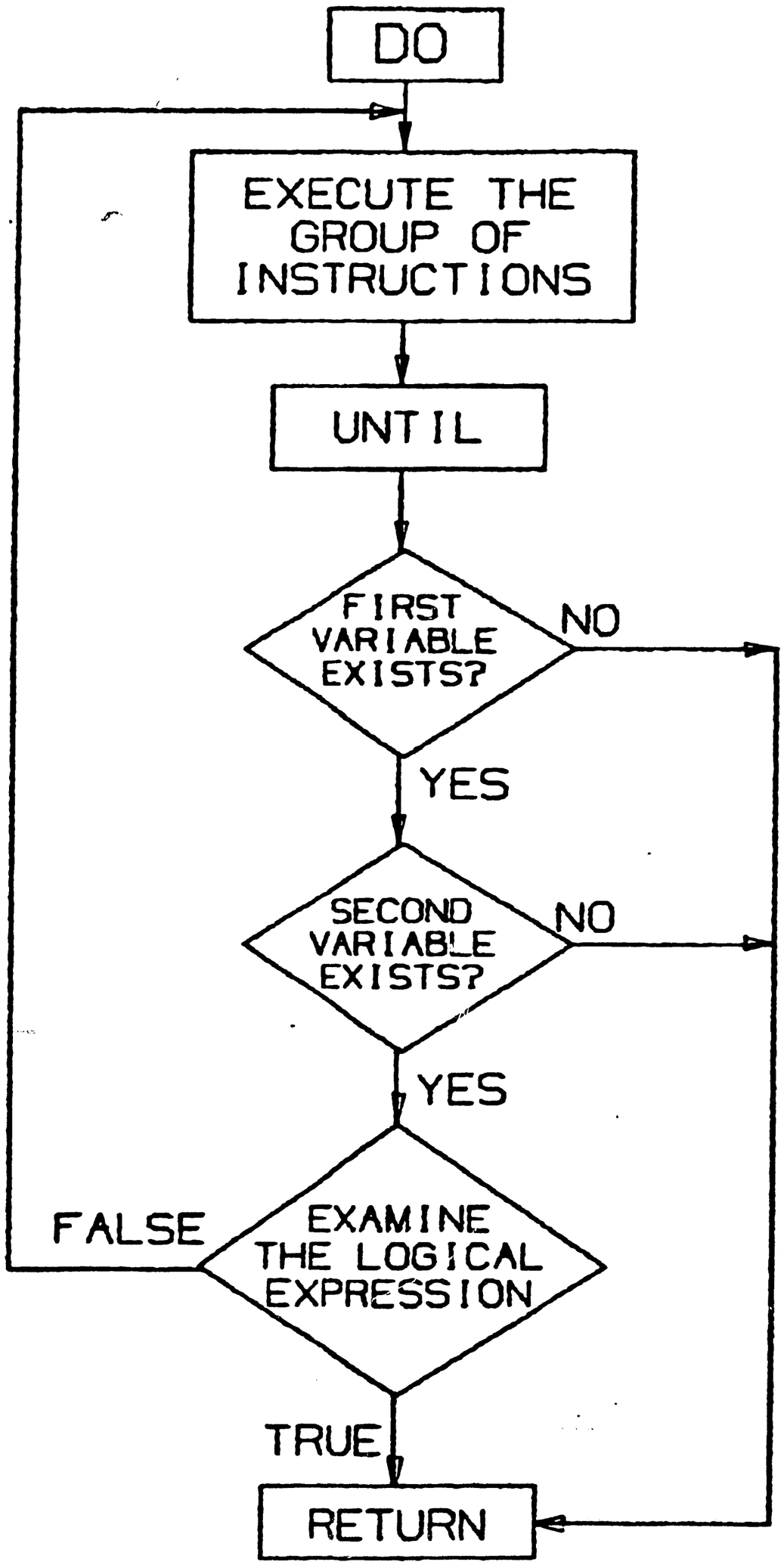


FIGURE A-12. SUBROUTINE UNTIL FLOW DIAGRAM

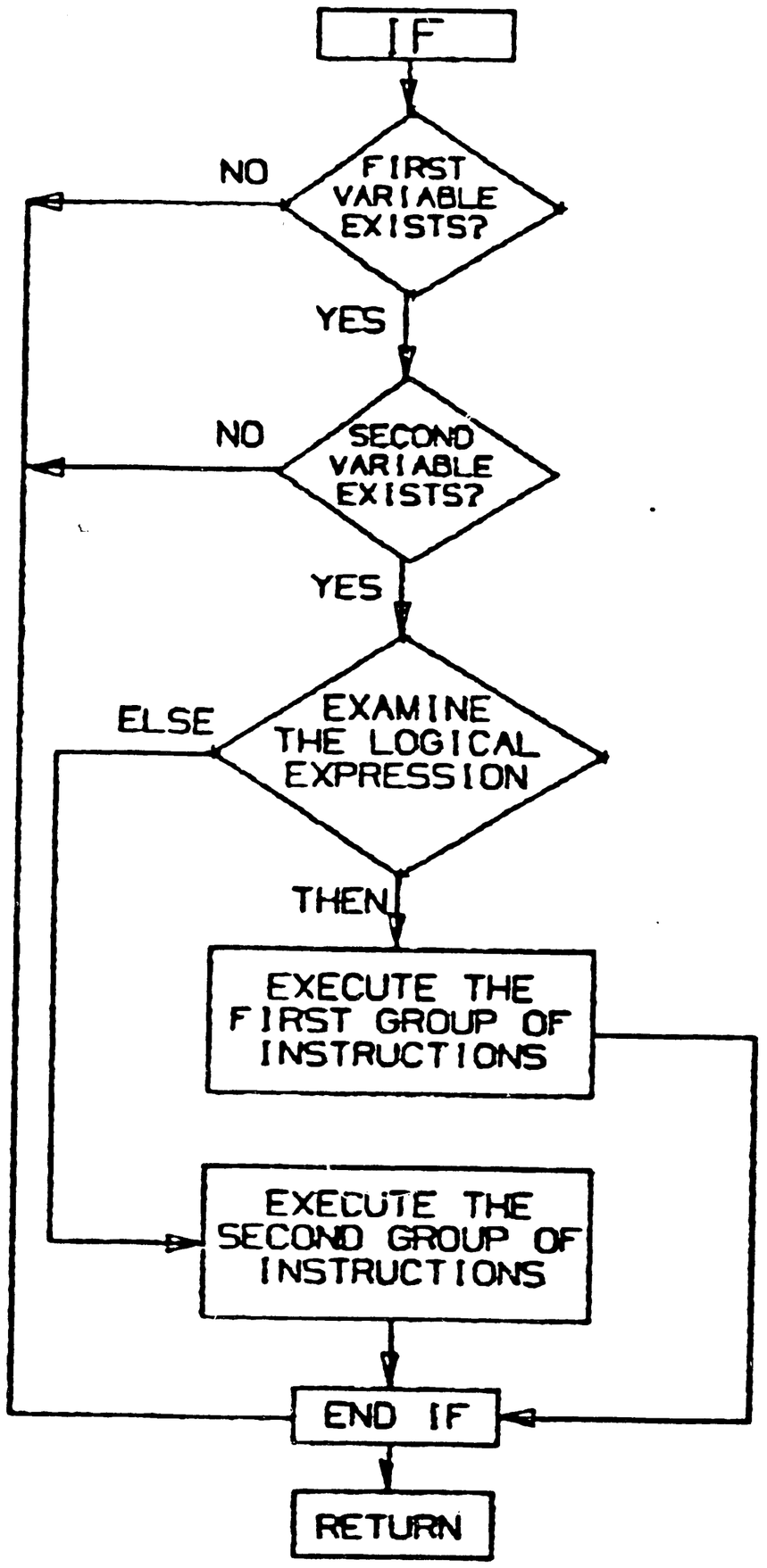


FIGURE A-13. SUBROUTINE LOGIC FLOW DIAGRAM

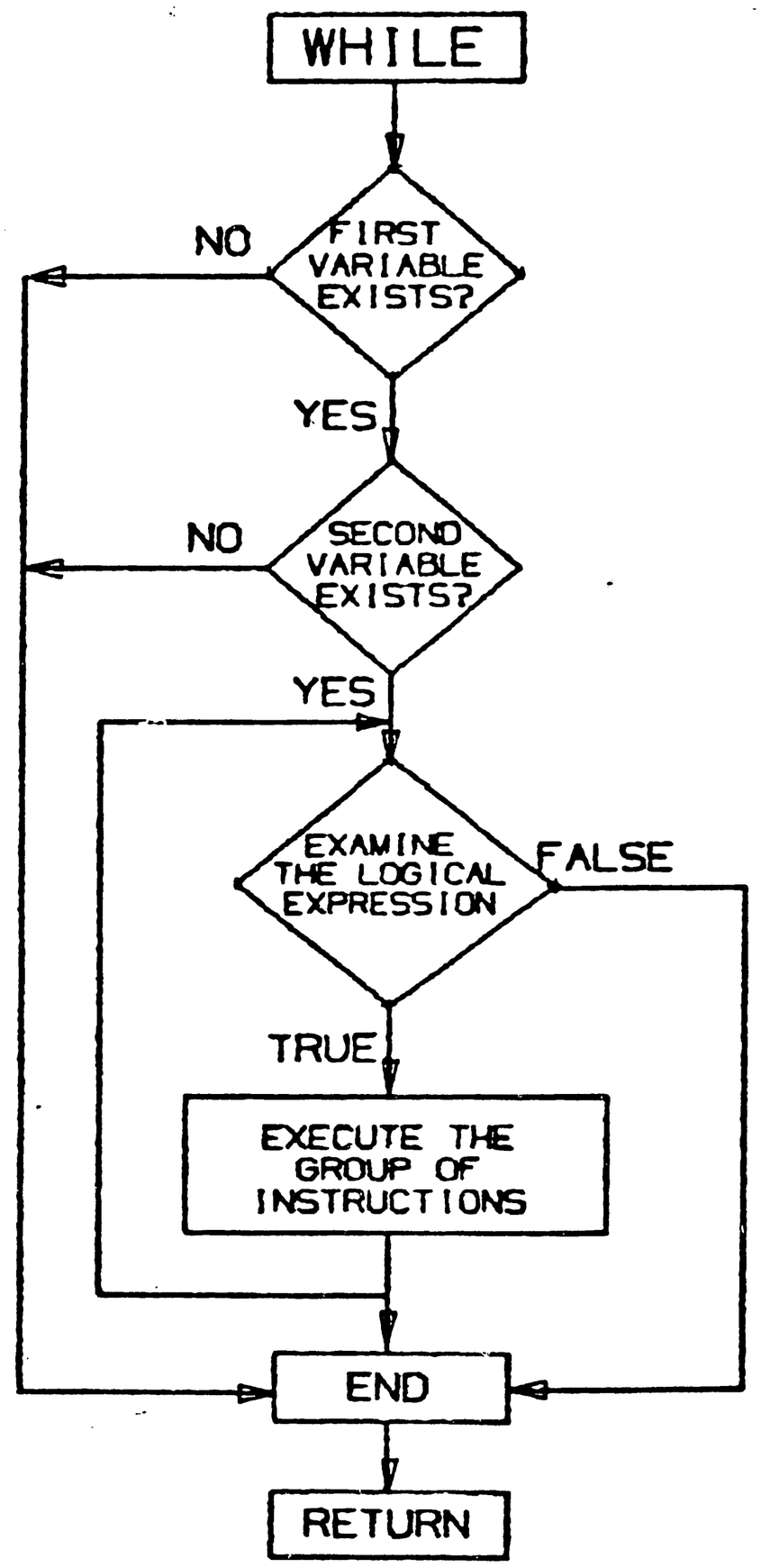


FIGURE A-14. SUBROUTINE WHILE FLOW DIAGRAM

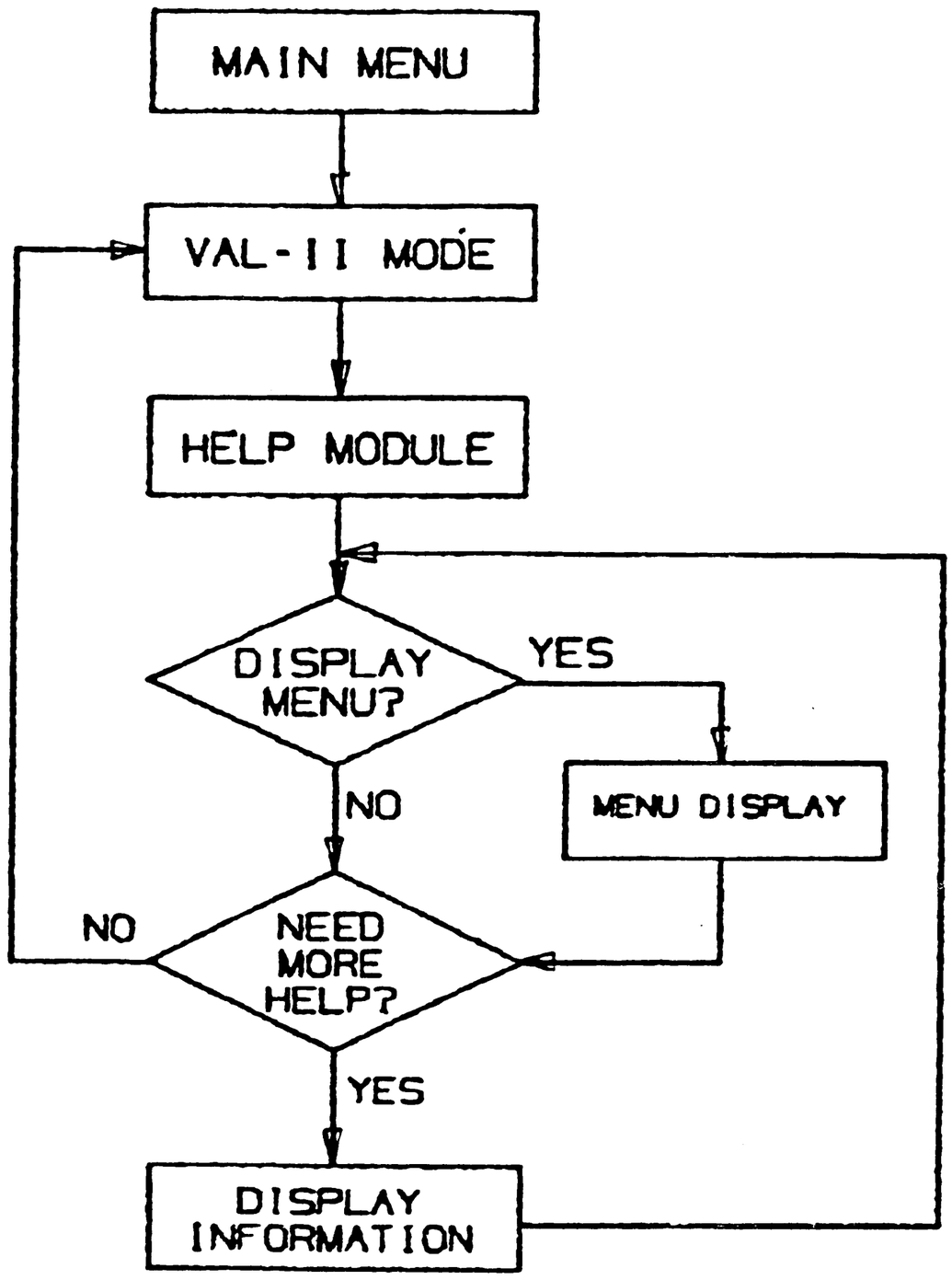


FIGURE A-15. SUBROUTINE HELP FLOW DIAGRAM

APPENDIX B.

```
COMMON /ANGLE/ ANG, THETA(6), CE(6), S(6), AMAX(6,2), LAST(6), SPEED(6)
COMMON /ARMS/ LPH, SF, CF, ARM(300,3), A(300,3), B(300,3), MVERT(30,3)
COMMON /CLMP/ ICHEK, ILATCH, IPOS, NPART, IATTACH(30)
COMMON /CONFIG/ IARM, IELBOW, IWRIST, KK
COMMON /GEOM/ A2, D2, D4, D6, H, PI
COMMON /GROWTH/ RGROW(30), XH, YH, ZH
COMMON /LOOP/ ILOOP, IREMAIN, HELPER(100)
COMMON /NORMS/ IHIDE, NORM(30,2), HID(150,3), HIDE(150,3)
COMMON /PER/ IPER, SPDERR, SPEEDM
COMMON /POS/ LPOINT, IEND, CHAR(100), DATA(100,6), COMM(70), LISTER(3)
COMMON /POS/ STAT(100), NUMB
COMMON /POS/ DATFILE(100), POS11, POSITION
COMMON /POSIT/ IND, BDXR, BDYR, BDZR, RZ, CZ, SZ, WCS(7,3)
COMMON /POSTN/ XTEMP, YTEMP, ZTEMP
COMMON /ROTM/ R4(3,3), R5(3,3), R5T(3,3)
COMMON /OUT/ ITIME
COMMON /SINCOS/ S1, S2, S3, S4, S5, S6, S23, C1, C2, C3, C4, C5, C6, C23
COMMON /FLAG/ IFLAG(100,3)
COMMON /VERIND/ NPI(30), NLI(30), IPINDEX(800,3), LINDEX(1100,2)
COMMON /SCPRMS/ SDXLNG, SDYLNG, CFAC, SRESX, SRESY
```

Table B-1. List of Common Blocks

BLOCK	VARIABLE	FUNCTION
ANGLE	ANG	rotational increments in Pendant Mode (radians)
	THETA (6)	joint angles in Simulator convention (radians)
	CS(6)	cosine of joint angles in Simulator convention
	S(6)	sine of joint angles in Simulator convention
	AMAX(6,2)	stop angles in PUMA's convention (degrees)
	LAST(6)	previous joint angles in simulator convention (radians) to avoid redundant calculations
	SPEED(6)	rotational increment for joint during joint-coordinated motion (radians)
ARMS	LPH	total number of polyhedrons
	SF	scale factor (pixel/inch)
	CF	conversion factor (mm/inch)
	ARMS(300,3)	definition geometric data (in)
	A(300,3)	rotated geometric data (pixels)
	B(300,3)	display geometric data (pixels)
	MVERT(30,3)	polyhedron index, its beginning and end vertex
CLMP	ICHEK	clamp flag: ope=0, close=1
	ILATCH	LED flag: grasp not possible=0 grasp possible =1
	IPOS	position flag: clamp base=1 LED =2 clamp tip =3

TABLE B-2. List of commoned blocks

BLOCK	variable	function
	IATTACH(30)	attachment flag: unattached=0 attached =1
CONFIG	IARM	arm configuration indicator RIGHTY=1, LEFTY=-1
	IELBOW	elbow configuration indicator ABOVE=1, BELOW=-1
	IWRIST	wrist configuration indicator NOFLIP=1, FLIP=-1
GEOM	A2	joint 2 to joint 3 offset (in)
	D2	joint 1 to joint 2 offset (in)
	D4	joint 3 to joint 4 offset (in)
	D6	joint 4 to clamp tip offset (in)
	H	WCS origin offset from SCS origin (in)
	PI	3.1415927
GROWTH	RGROW(30)	radii of enclosing sphere for each polyhedron (pixels)
	XH,YH,ZH	hand coordinates (pixels)
LOOP	ILOOP	number of program loops
	IREMAIN	number of program loops remaining to be completed
	HELPER (100)	help topics
NORMS	IHIDE	hidden-surface removal flag: back surfaces not removed=0 back surfaces removed =1
PER	IPER	percentage of maximum simulation speed
	SPEDRR	half of speed dependent error matgin

TABLE B-3. List of commoned variables

BLOCK	VARIABLE	FUNCTION
	SPEEDM	max. rotational increment in VAL-II mode (radians)
POS	LPOINT	location name pointer
		total no. of simulated commands
	CHAR(100)	location names
	DATA(100,6)	location data (mm,deg)
	COMM(70)	VAL-II Simulated commands
POS	STAT(100)	variable names declared by user
POS	DATFILE(20)	names of existing VAL-II files
	posll	user declared variable name
	position	user declared variable name
		used for assignment instructions
POSIT	IND	index of positions along TCS's Z-axis
	BDXR,	coordinates of displacement
	BDYR,	increment due to BASE command
	BDZR	
	RZ	rotational increment due to
		BASE command (degrees)
	CZ	cosine of RZ
	SZ	sine of RZ
	WCS(7,3)	coordinates of vertices which
		form WCS frame
POSTN	XTEMP,	temporary clamp position
	YTEMP,	
	ZTEMP	

TABLE B-4. List of commoned variables

BLOCK	VARIABLE	FUNCTION
ROTM	R4(3,3) R5(3,3) R5t(3,3)	rotation matrix for joint 4 rotation matrix for clamp transpose of clamp rotation matrix
OUT	ITIME	POLYGON output flags: first output=0 subsequent output =1
SINCOS	S1,S2,S3,S4, S5,S6,S23, C1,C2,C3,C4, C5,C6,C23	sines & cosines respectively of the joint angles in simulator convention
FLAG	IFLAG(100,3)	flags : IFLAG(1,1)=0 position display OFF (1,1)=1 position display ON (2,1)=0 motion continues (2,1)=1 joint angle exceeded (2,2)=N no. of joint angle exceeded (30,3)=0 NUNIT=5 (30,3)=1 NUNIT=4 (42,3)=0 logical condition satisfied in WHILE (42,3)=1 logical condition not satisfied (43,3)=0 group of instructions should be executed in WHILE (43,3)=1 group of instructions should not be executed in WHILE (46,3)=0 no VAL-II program is being executed (46,3)=1 VAL-II program execution continues

TABLE B-5. List of commoned variables

BLOCK	VARIABLE	FUNCTION
		IFLAG(48,3)=0 file being edited is a new one (48,3)=1 file already exists (61,3)=0 input is a VAL-II command (61,3)=1 input is a assignment instruction (65,3)=1 in editor, the syntax is checked (VAL-II command) (65,3)=0 in editor syntax is not checked (assignment instruction) (99,1)=0 motion continues (99,1)=1 impossible path (100,1)=0 collision check OFF (100,3)=0 Axes display ON (100,3)=0 Axes display ON (100,3)=1 Axes display OFF otherwise IFLAG(1,1)=0 no collision (1,1)=1 collision (1,2)=N near polyhedron N (1,3)=M near polyhedron M
VERIND	NPI(30)	indices of polygons for all polyhedrons
	NLI(30)	indices of line segments for all polyhedrons
	IPINDEX(800,3)	indices of polyhedrons, polygons and vertices
	LINDEX(400,3)	indices of vertices for line segment
SCPRMS	SDXLNG, SDYLNG, CFAC, SRESX, SRESY	used y VSI13D graphics package to set the window

TABLE B-6. List of commoned variables

APPENDIX C.

PROGRAM USER'S MANUAL

PROGRAM TITLE:	VAL-II* SIMULATOR
DATE:	NOVEMBER 24, 1985
AUTHOR:	S.A. LANGRUDI
PROGRAM LANGUAGE:	FORTRAN IV
GRAPHICS PACKAGE:	VS113D
MACHINE WRITTEN FOR:	DEC VAX 11/780
OPERATING SYSTEM:	VMS
DISPLAY TERMINAL USED:	VS11

TABLE OF CONTENTS

C COPYRIGHT Lehigh University, 1985.

* VAL-II is a trademark of Unimation, Inc.

1. INTRODUCTION

The VAL-II SIMULATOR is an interactive computer-graphics simulator of VAL-II, the programming language of Unimation's PUMA robot. The simulation program serves the following purposes:

-moves the end-effector to different locations

-orients the end-effector

-opens and closes the gripper

-performs repeating tasks

-makes conditional and unconditional branching or jumps

It employs an internally developed graphics package called GRAPH3D.LU to display the PUMA behavior upon execution of key VAL-II commands. The robot model and user-defined workpieces are presented using wire-frame or edge representation graphics scheme. Planar polygons are used to form extruded convex polyherdons. In order to create a more realistic representation of the model and its work environment, also to avoid complexity due to number of lines to be drawn, hidden-surfaces are removed. Also, algorithms for joint-coordinated and straight-line motions are developed. Information regarding configuration indicators and clamp status is available at any time. Arithmetic and logical expressions, like those found in

high-level computer languages are available for decision making and programming control. Collisions between moving and stationary objects, between the robot clamp and stationary objects, and between the robot links and themselves are automatically detected. User-defined objects may interactively be retrieved from other CAD data bases using the International Graphics Exchange Standard (IGES) and also using POLYGON [19] interface, and then easily positioned in the PUMA's work-environment. With these features, users may design and evaluate various work-cell layouts, investigate assembly tasks, and test VAL-II programming possibilities. VAL-II SIMULATOR is menu-driven which means that a list of options is available at all control levels. Once a work-cell is created and the VAL-II program is developed and tested with the simulator, the VAL-II program may be down-loaded directly to PUMA controller for final testing and verification.

The PUMA is a spherical robot with 6 revolute joints and six axes. For an anthropomorphic robot such as PUMA 600, 3 configuration indicators must be specified to gurantee one unique solution out of a possible four for the first three links, and one valid solution out of two for the last three joints. The first two configuration indicators are associated with the first three joints. It must first be specified whether the robot arm is to the right of the base (RIGHTY robot) or to the left of the base

(LEFTY robot); also, whether the elbow of the robot is above the wrist (ABOVE), or below it (BELOW). The last configuration indicator determines whether the clamp is to point upward or downward (FLIP or NOFLIP). Oftentimes, a user may want to move the robot through an impossible path which causes one or more of robot links to exceed their prespecified allowable joint angle rotation limits. In most cases, this may be taken care of by changing one or more of the configuration indicators, or introducing some intermediate locations. This however, requires a change in the VAL-II program which can easily be done using the EVS (Editor of VAL-II SIMULATOR).

Using the simulator can significantly reduce the amount of time spent on robot programming and debugging. It also does not require tying up a robot from the production-line or using a "spare" robot. It can also contribute to both the safety of the operator and the industrial robot and time-saving.

2. ORGANIZATION

The simulator is menu-driven, which means that a list of options is available at all control levels. Upon initialization of a VAL-II session and after the copy right information is displayed, the user is asked to select a setup file. The user may decide to retrieve part files from

other CAD data bases and reposition them in the work environment. This can be done using an IGES or POLYGON [19] interface. In either case proper menus are displayed and the user is required to input his commands using only single key inputs from the keyboard. Simulation's flow diagram is presented in Figure C-1. Once a setup file is selected and the file read in and displayed, the user is directed to the MAIN MENU (Figure C-2). From this menu, one may move to MANIPULATION Mode, EDIT Mode, KEYBOARD Mode, PENDANT Mode, and VAL-II Mode. This menu is also used to automatically create files that may be used for hardcopy printouts and for POLYGON [19] generated solid models. While in this menu, it is possible to initialize the work-cell or begin with an entirely new setup file or retrieve a different sets of parts from other CAD systems. Whenever, any error is encountered informative messages are displayed. To stop and exit the VAL-II environment, one may press 'S'. However, for protecting the new users, the simulator does not exit and asks the operator whether he is sure or not. From the MAIN MENU one may move to various control levels and back. If it is required to reposition any of the cell-components to design and test different layouts, the user may wish to move to the MANIPULATION Mode. The MANIPUMATION MENU is immediately displayed as shown in Figure C-5. While in this mode, any individual part in the work-cell may be moved by either the system

defined, or user-defined increments, along any one of the axes of the fixed Screen Coordinate System (SCS) to achieve the final desired positions. A constant readout of the individual part positional data with respect to both the SCS and WCS as well as the part number is displayed on the screen. This level of the simulation program is menu-driven and like other parts very easy to use. Any time, an error is encountered, an informative message is displayed. Whenever, the user decides to end repositioning one part, the simulator asks him whether he would like to save the new setup, and if so, whether he would like to save the new setup file under a new name or the old file should be updated. At this level, for convenience, the coordinate axes display may be turned ON or OFF. Once done with repositioning the part(s), the user may return to the top of the menu. A flow diagram of the manipulation mode is shown in figure C-6.

The simulator offers many special features. These interactive features are all presented in the KEYBOARD Mode. To enter the Keyboard Mode from the MAIN MENU, one must press 'K'. Following this input the KEYBOARD MENU will be displayed as shown in figure C-3. VAL-II program files and location files may be listed on the screen, while in this mode. A user may obtain a list of all the VAL-II robot control programs by typing LISTF. Also, contents of VAL-II program(s) may be displayed on the screen for reference and

verification. A list of all location data files used by VAL-II programs may be displayed on the screen by typing LISTL. In this mode, the position of the clamp may be displayed by pressing 'P' followed by '1', '2', or '3' corresponding to position along the TCS's Z-axis at the origin, LED sensor, and clamp tip, respectively. One drawback of using wire-frame or edge representation scheme is that, it is very difficult to visually detect whether an object is in front of, behind, or intersecting another one. For this reason, algorithms for collision detections are used. In order to create a more realistic work environment, and to avoid complexity, the hidden-surfaces are effectively removed upon user's request. Like any other user friendly graphics package, the simulator allows the user to translate, rotate and scale the entire work-view. For convenience, the coordiante axes display may be turned ON or OFF. At the end of a KEYBOARD session, the control may be transfered to the top of the menu, MAIN MENU, by pressing 'R' (R_ETURN). A flow diagram which describes this mode is presented in Figure C-4.

One section of the simulation program has been dedicated to graphically display the functions and usage of a "Teach Pendant". This mode may be invoked by pressing 'P' form the MAIN MENU. The PENDANT MENU, as shown in Figure C-10 will then be dsiplayed on the screen. This mode allows the user to move the PUMA joints independently of each

other. This is the exact method an industrial robot is taught points along various paths. The user is also enabled to move the clamp along straight-line parallel to any one of the axes of the WCS or TCS, by pressing 'W' or 'T' followed by 'X', 'Y', or 'Z', to move in the corresponding directions. The clamp may also be displayed in fully 'OPEN' or 'CLOSE' positions. Unlike the real PUMA, the speed set in Pendant mode and VAL-II mode are independent. However, it is possible to increase or decrease the joint angle rotation increment by pressing 'I' or 'D', respectively. It is also possible to change the direction of the joint rotations by pressing 'N'. This operates as a switch and changes the rotational sense from clockwise to counterclockwise and vice versa. A flow diagram corresponding to this mode is displayed in Figure C-11.

Another important feature of the simulator is its Editing capabilities. The EVS is a so called "Line Editor" meaning that to make any changes on any given line, the entire line must be retyped. In many situations, some of the most impossible paths are not known, until a robot program is run once. One way to get around this problem is to change one or more of the configuration indicators, or introduce some intermediate locations. In either case, modification of VAL-II program is necessary; something which can easily be done using the EVS. One may invoke the EDIT Mode by pressing 'E', while in the MAIN MENU. The

legal Editor commands are immediately displayed on the screen (Figure C-8). For convenience, this menuis displayed on the screen at all times while in this mode. At this point to start an Edit session, the user types in "EDIT" followed by the program name, for exapmle, xxx.DAT. The program name may be up to 9 characters long. The EVS will search for the input file name in its directory. If the name is matched with one of the entries in the list, the file is an existing one, and the contents of it will be dislayed on the screen. Otherwise, the user is signalled by displaying the following message:

INPUT FILE " xxx.DAT " DOES NOT EXIST

and the new name will be added to the entries in the list, and the file will be initialized. If the file is a new one, the user then presses 'C' to continue. In the EVS every line is referenced by a unique step number. Hence, upon depression of the key 'C', <S. 1> will appear on the screen. The user may then type in a line of VAL-II command(s) as desired. As a program evolves, many of the lines may be deleted, or new ones inserted. As a result of this, the step numbers are automatically adjusted. Therefore, the step numbers should not be used as line-addresses. Instead, the first four spaces in any line are reserved for step labels. Step labels are optional, and if the user does not wish to specify a step label for a given line, he must type in four spaces before typing in the VAL-

II command. Conveniently, the user may set the 'TAB' to <5>, and if a step label is to be omitted, one must depress the 'TAB' key before typing the VAL-II command. In the case when a robot control program has already been created and the user wishes to modify it, one may delete or insert lines by pressing 'D', or 'I' followed by corresponding step number, respectively. To protect new users, every time such modifications take place, the updated version of the program will be displayed on the screen. During program modification, if a wrong key is depressed, the user is immediately informed of that, and for protecting the modified version, the program is saved at once. A flow diagram is shown in Figure C-11, which clearly describes the functions and usage of this mode. To exit the EDIT Mode, one may press 'E' which willtransfer the control to the top of the program, namely the MAIN MENU.

The most important feature of the Simulation program is the VAL-II Mode. VAL-II commands are Englishlike and easy to memorize. For this reason the VAL-II Menu (Figure C-13) display is optional. In this mode, users may enter a VAL-II command and observe the PUMA's kinematic behavior upon execution of that command. Like the EVS, in the VAL-II mode, the first 4 spaces are reserved for step labels, and if the step label is to be omitted, the user may set the 'TAB' to <5>, and depress the 'TAB' key when necessary, or

simply type in 4 spaces and start the VAL-II command from the fifth column. End of each line is marked by pressing the RETURN key from the keyboard. The simulator will read the command string, decode it, set a pointer, and display the corresponding PUMA behavior. Following the execution of each VAL-II command the user must press 'C' to continue with the next line. In order to teach locations to robot one must press 'D', at which point a menu will be displayed which will ask the user whether he wishes to read the locations from a data file, or they are to be input interactively. If the latter is chosen, the user may input the desired positions and orientation angles as desired, and the robot will be displayed in those configuration. One may store various locations in a data file and ask the simulator to read the position and clamp orientation data from that file. In this case, the user is required to input the name of the data file in which the locations are stored. The simulator will open the file and start reading the data. Every time a line of data is read, the robot will be displayed in the corresponding position and clamp orientation, and the user is asked to either accept the location and continue with the next line of data, to reject the location but continue with the rest of the file, or to accept the location and quit reading the remaining lines of data in that file. The user is required to specify his choice by single key inputs. To wipe the 'TT' characters

out one may press 'W' before typing the next VAL-II command. Depression of 'R', transfers the control to the top of the menu. While in the VAL-II Mode, if any error is encountered, the user is immediately notified of that by displaying the appropriate message. User programs may be executed and user-defined subroutines can be called to accomodate various tasks and to increase programming control. Conditional and unconditional branching or jumps may be performed with limited sensory interface. To abort a VAL-II program without leaving the Simulator environment, and hence preserving the locations and setup, one may press the keys "CTRL" and "C", simultaneously, which acts as an "ABORT" command in VAL-II language. At all control levels, on-line HELP is available for all the VAL-II and some MONITOR commands. For each topic, the prper syntax, and a description of the function performed are given.

3. SCOPE

The apparent motion in the simulator is achieved by transforming the present locations to desired ones, quickly erasing the screen, update the data and display the entities in their new position. When this is done quickly, one's eye blends the images to create the apparent motion. Depending upon the distance between the current and the desired configuration, the intermediate locations are

determined. The model is then displayed in the initial, intermediate and final configurations. In the simulator, the speed variations are based on the relative speed and not the true one. The speed set in the Pendant Mode, and the one set in the VAL-II mode are independent of each other. The default speed, in the simulator, is set so as to provide smooth display. In the VAL-II mode, the default value for maximum joint rotation increment is set to 12 degrees (full speed), which may be adjusted by VAL-II commands. In the Pendant Mode however, the default value for joint rotation is set to 8 degrees which may be increased or decreased.

To create an illusion that the interior region of a displayed surface is opaque, and also for better visualization, the hidden-surfaces may be removed. This is done by calculating the coordinates of the normals of the planar polygons, to determine whether the face is a "front face" and potentially visible, or a "back face", and thus invisible. It must be noted that, the algorithm used here, only removes a face(s) which is hidden by the volume of the polyhedron it belongs to.

One drawback of wire-frame representation is that, when an object extends over, and covers a part of another object, it is very difficult to visually detect whether one object is in front of, behind, or intersecting another one. For this reason, routines have been developed which use

both the coarse and fine mathematical intersection check algorithms which notify users of potential collisions. Possible collisions between the moving and stationary objects, the robot links and themselves, and the robot links and stationary objects are checked, but impossible ones are not.

The simulator also models the interaction between the robot model and user-defined cell components. Latching onto, moving and detaching from objects is simulated.

The on-line help feature is one of the most important features of the VAL-II SIMULATOR and is available at all control levels. This feature makes the simulator a reliable and effective instructional instrument for training the future robot user. Help is available for all VAL-II commands and some Monitor commands. To increase the flexibility and programming control, the simulator takes advantage of a two pass compiler to simulate conditional and unconditional branching or jumps. In the first pass, the simulator makes a record of the step label, variable names, and examines the logical condition. In the second pass, it finds the specified step and transfers the control to that step. Also, limited sensory interface is available.

Editing features allow users to create and modify robot control programs. The Edit Mode provides a more suitable work area for the programmer, so that the individual could use his resources more efficiently.

User-defined geometries may be retrieved from other CAD systems using International Graphics Exchange Standard (IGES) interface or through POLYGON [19] interface; they may then be repositioned in the PUMA's work environment interactively.

It must be kept in mind that the VAL-II SIMULATOR is not a robot emulator. It does not simulate the effects of gravity nor inertia. Dynamic characteristics such as accelerations and decelerations are not included to minimize the amount of computations, thus providing smoother motion. One must also note that, the simulator picks up objects when the LED line-segment is intersected and the flag is set. The simulator does not know whether the object is between the two faces of the clamp nor if it is to heavy. So,the simulator picks up objects, when in reality the robot misses the object, or the object slips out, when they might be picked up.

Nearly, all the Location, Configuration, Motion Commands, as well as hand and programming control commands are simulated by VAL-II SIMULATOR. The simulator does not simulate any hardware nor can it handle precision points for compound transformations.

Other PUMA models may be simulated by modifying the present geometry data file. In order to simulate the kinematic behaviors of other robots which do not have six revolute joints in an identical configuration, requires

different transformation matrices. Once, the dedicated subroutine which forms a particular transformation matrix for a joint has been changed, the present method is used to obtain the display data for the subsequent links.

4. DATA PREPARATION

The robot model is presented by a series of extruded convex polyhedrons. The user-defined workpieces are shown by other polyhedrons. The simulator uses a wire-frame or edge representation scheme. To form a polyhedron, corresponding vertices of n-sided polygons are connected to form planar polygons which enclose a volume. The cell components may be defined by creating the parts in a different CAD system and then interactively retrieved and positioned in PUMA's work-environment, or by creating separate setup geometry file. A setup file is shown in Figure C-20. The first line contains the number of user-defined workpieces in the cell, 5. The object coordinates may be given in millimeters or inches. Either "MM" or "IN" may be used. The third line, shows the coordinate system with respect to which the object data is defined. A "W" will designate the WCS, and a "S" will specify the SCS. Following the third line of each setup file, lines of data, one corresponding to each user-defined object appear to indicate each object's size; this is

specified by giving the number of vertices belonging to one polyhedron. The object size is shown by specifying the first and last vertex with a comma separating them (1,8). Due to the data format chosen here, the first digit should be 1, and the lastone an even integer larger than 6. Associated with each object is a reference point (the first corner). Next line in the setup file, following the number of the vertices for the first polyhedron is the coordinates of the reference point, namely the first corner. Following this line, the coordinates of each vertex is given with respect to the reference point, shown in alphabetic order (X,Y,Z) separated by commas or spaces. The order in which the vertices are defined is critical. The vertices representing the front and back faces must be given either in a clockwise or counterclockwise order, when looking toward the origin along the line of the normal axis. This is very important for effectively removing the hidden-surfaces. When a counterclockwise convention is used, the magnitude of the coordinates along the normal axis for the first polygon is larger than that of the second one, and vice versa, when a clockwise convention is used.

The locations may be defined interactively, by typing the position and clamp orientation angles while in the VAL-II Mode. There is a quicker method to define locations, and that is to write them in a data file. The position and orientation angles may easily be determined by looking at

the position of the specific objects defined in a setup file. A location data file is shown in Figure C-19. In this file, each location is signalled by the label, POINT, followed by a space, and then an appropriate name of up to 9 characters long, a comma, followed by the position and clamp orientation angles, separated by commas. Once the setup and location files are created, one may enter the VAL-II SIMULATOR environment and observe the kinematic behavior of the PUMA upon execution of various VAL-II commands.

5. SIMULATOR USE

The simulator may interactively be used to serve the following purposes:

-to move the end-effector to different positions

-to orient the end-effector

-to open and close the gripper

-to perform repeating tasks

-to make conditional and unconditional branching or jumps

The order in which the VAL-II instructions are to be executed is critical to simulate a manufacturing task. Oftentimes, several iterations are necessary to determine the most efficient sequence of commands. The EVS may be used to interactively create and modify VAL-II programs.

Locations may be created either using the Pendant or VAL-II Modes.

Once programs and locations have been verified on the simulator, they may be down-loaded to PUMA-controller for final testing and implementation.

6. EXAMPLE SESSION

The following examples show, step by step, the features of the simulator. Except for the EDIT Mode and the VAL-II mode, user inputs are single key inputs. The proper syntax of each command is checked when it is read in. At any time, if any error is encountered, corresponding error messages are displayed. If a VAL-II command does not match one of the entries in the COMMAND.DAT file, it can not be simulated and an error message is displayed. The EVS continuously checks the syntax of the commands that are input. If the command is not one of those which may be simulated, the line is rejected and an error message will be displayed. In the steps of the table below, "RETURN" denotes a carriage return key, which marks the end of a line. When more than one input or output is given, they are separated by commas. In the format which is used here, the first column describes the user's input or the program's output; the second column gives a brief description of the inputs or outputs on the corresponding line, and the third

column references the figure which corresponds to the display produced by the terminal. The first example, PROCESS, uses some of the VAL-II commands that can be simulated. The second example, CONV1, clearly shows the functions and usage of arithmetic and logical expressions used in the VAL-II SIMULATOR. The program listings, location files, and setup files associated to these two examples are given in the figures which appear at the end of the chapter.

INPUT/OUTPUT	EXPLANATION	FIGURE
"R ROBSIM", RETURN	RUN VAL-II SIMULATOR	
E "DEMO3.SET" RETURN	read from existing setup file	C1
RETURN	Main Menu display	C-2
K	Keyboard Menu display	C-3
F	hidden-surfaces not removed	
F	hidden-surfaces removed	
A	Axes display ON	
A	Axes display OFF	
R	return to Main Menu	
M	Manipulation Menu display	C-5
1, RETURN	move part no. 1	
1	coordinate axes ON	C-7 a
Y	continue	
Y, 0,0,300	translate part by 0,0,300	
Y,0,0,45	rotate part about y by 45 deg.	C-7 b
N	stop moving first part	
Y,2,RETURN	move part no. 2	
Y,0,0,-200	translate by 0,0,-200	
N	no rotation	C-7 c
N	end moving parts	
Y, "TEST.DAT"	save file under the new name	
RETURN	back to main menu	
E	Edit Mode	C-8
"EDIT TEST.DAT"	edit program TEST.DAT	
C	continue	
" MOVE A"		
" MOVE B"		
" CLOSEI 50"		
" STOP"		

TABLE C-1. STEP BY STEP EXAMPLE SESSION

INPUT/OUTPUT	EXPLANATION	FIGURE
D,3, RETURN	delete step 3	
I,3,RETURN	insert a new step instead	
" MOVE E"	new step 3	
S,E	save file & exit from editor	
P	Pendant Mode	C-10
M	display model	
R	back to main menu	
K	Keyboard Mode	
S,R	collision check ON, return	
P	back to pendant mode	
M	modl display	
1,1,1,1,1,1	rotate joint 1	
3,3	rotate joint 3	
1,1,1,1	rotate joint 1	
3,3,3	rotate joint 3	
1,1,1	increase the rotational increment	
3	rotate joint 3	C-12 a
3	rotate joint 3	C-12 b
2	rotate joint 2	C-12 c
N	change direction of rotation	
3	rotate joint 3	C-12 d
R,K,S,R	turn collision check off	
B;E. PROCESS.SET	begin with a new setup	C-14
V	VAL-II menu display	C-13
V	VAL-II Mode	
"LE"	lefty robot	
"AB"	elbow above	
D,D	read locations from a data file	
"PROCESS.LOC"	location file name	C-15
RETURN		
A	accept and continue	
A,A,A,A	accept and continue	
E,W	accept, end reading the file wipe out TT characters	

TABLE C-2. STEP BY STEP EXAMPLE SESSION

INPUT/OUTPUT	EXPLANATION	FIGURE
"EXEC PROCESS,1"	execute program process, once	C-16
"READY"		
"LE"	lefty	
"AB"		
"APPRO A,-50"	approach A	
"MOVE A"	move to locaion A	C-17 a
"CLOSEI 50"	close the clamp	
"DEPARTS 100"	depart by 100	C-17 b
"MOVES E"	move to E along straight line	C-17 c
"MOVES F"	move to F	C-17 d
"DEPARTS 200"	depart after part processed	
"APPROS C,-50"	approach C	C-18 a
"MOVES C"	move to C	
"OPENI 100"	open the clamp	
"DEPARTS 100"	depart	C-18 b
"APPROS B,=%0"	approach B	
"MOVES B"	move to B	
"CLOSEI 50"	grasp the part	C-18 c
"DEPARTS 100"		
"MOVES E"	move to E	
"SHIFT E,0,0,-100"	shift the location	
"MOVES E"	insert the part in processor	
"DELAY 1"	wait till part processed	
"DEPARTS 200"	depart	
"APPROS D,-50"	move to D	
"MOVES D"		
"OPENI 100"	open the clamp	
"READY"		
"STOP	stop	
R	return to main menu	
S	ens the session	
Y	confirm	

TABLE C-3. STEP BY STEP EXAMPLE SESSION

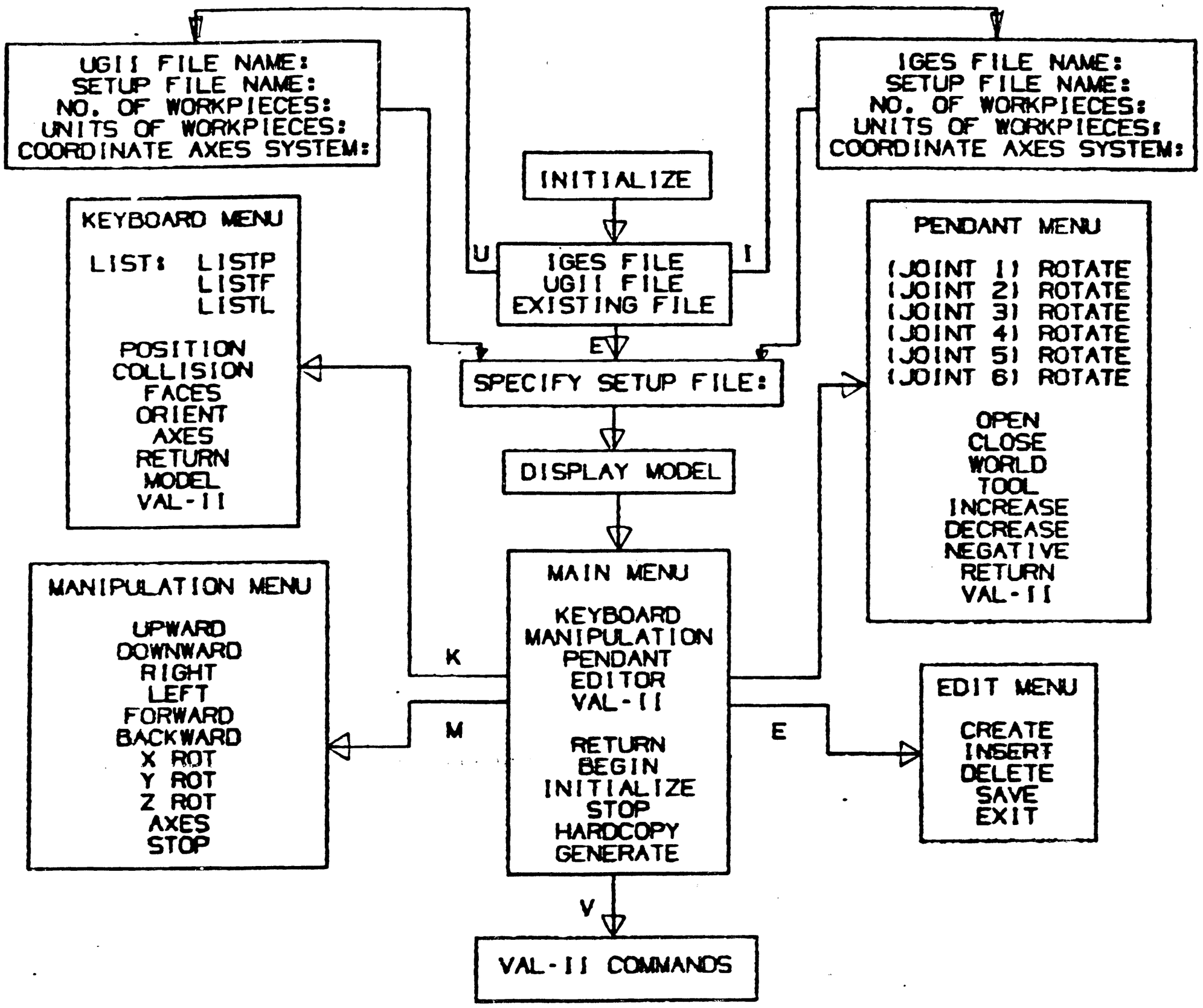


Figure C1. Simulator flow diagram

MAIN MENU

ENTER SUBMENUS BY PRESSING:

K_EYBOARD	:	TO KEYBOARD (MONITOR) MODE
M_ANIPULATION	:	TO MANIPULATE WORK PIECE LOCATIONS FROM THE KEYBOARD
P_ENDANT	:	TO PENDANT MODE
E_DITOR	:	TO EDIT MODE
V_AL-II	:	TO DISPLAY A LIST OF EXECUTABLE VAL-II COMMANDS

OR PRESS:

R_ETURN	:	TO RETURN TO PICTURE OF MODEL AND VICE VERSA
B_EGIN	:	TO START OVER
I_NITIALIZE	:	TO INITIALIZE WITH SAME SETUP FILE
S_TOP	:	TO EXIT PROGRAM AND CLEAR SCREEN
H_ARDCOPY	:	TO CREATE A PRINTOUT FILE
G_ENERATE	:	TO GENERATE POLYGON INPUT FILES

Figure C-2 Simulator Output Image for Example

KEYBOARD MENU

ENTER COMMANDS BY PRESSING:

L_IST - THEN TYPE LISTP PROG. NAME: VAL-II FILE LISTING
L_IST - THEN TYPE LISTF: VAL-II PROGRAMS LIST
L_IST - THEN TYPE LISTL: VAL-II LOCATION DATA FILES

P_OSITION : TO DISPLAY POSITION OF CLAMP
S_MOOTH : COLLISION CHECKS (ON/OFF)
F_ACES : BACK FACES (ON/OFF)
O_RIENT : CALLS ORIENT1 TO SCALE, TRANSLATE OR ROTATE MODEL
A_XES : COORDINATE FRAMES (ON/OFF)

R_ETURN : TO RETURN TO THE TOP OF A MENU
M_ODEL : TO DISPLAY THE MODEL
V_AL-II : BEFORE ENTERING A VAL COMMAND

Figure C-3. Simulator Output Image for Example

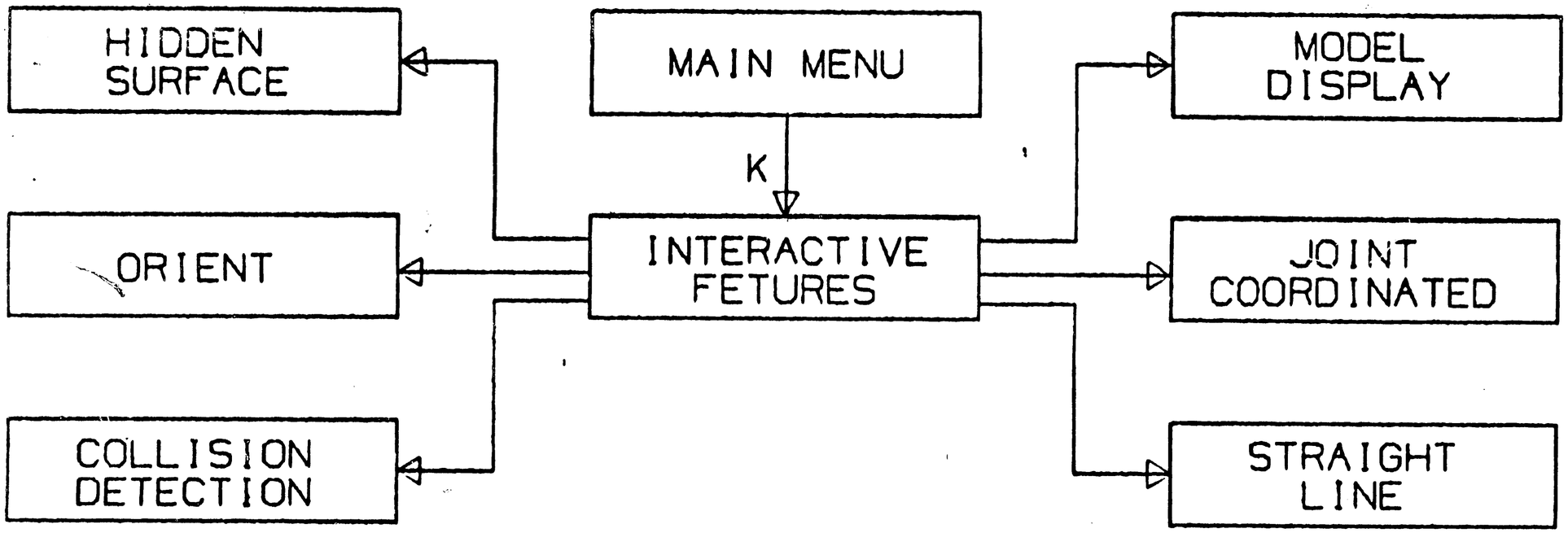


Figure C-4. Keyboard Mode Flow Diagram

MANIPULATION MENU

ENTER COMMANDS BY PRESSING:

U_PWARD	:	MOVE THE PART UPWARD
D_OWNWARD	:	MOVE THE PART DOWNWARD
R_IGHT	:	MOVE THE PART TO RIGHT
L_EFT	:	MOVE THE PART TO LEFT
F_ORWARD	:	MOVE THE PART FORWARD
B_ACKWARD	:	MOVE THE PART BACKWARD

X_ROT	:	ROTATE THE PART ABOUT X-AXIS
Y_ROT	:	ROTATE THE PART ABOUT Y-AXIS
Z_ROT	:	ROTATE THE PART ABOUT Z-AXIS
A_XES	:	COORDINATE AXES (ON/OFF)
S_TOP	:	RETURN TO MAIN MENU

Figure C-5. Simulator Output Image for Example

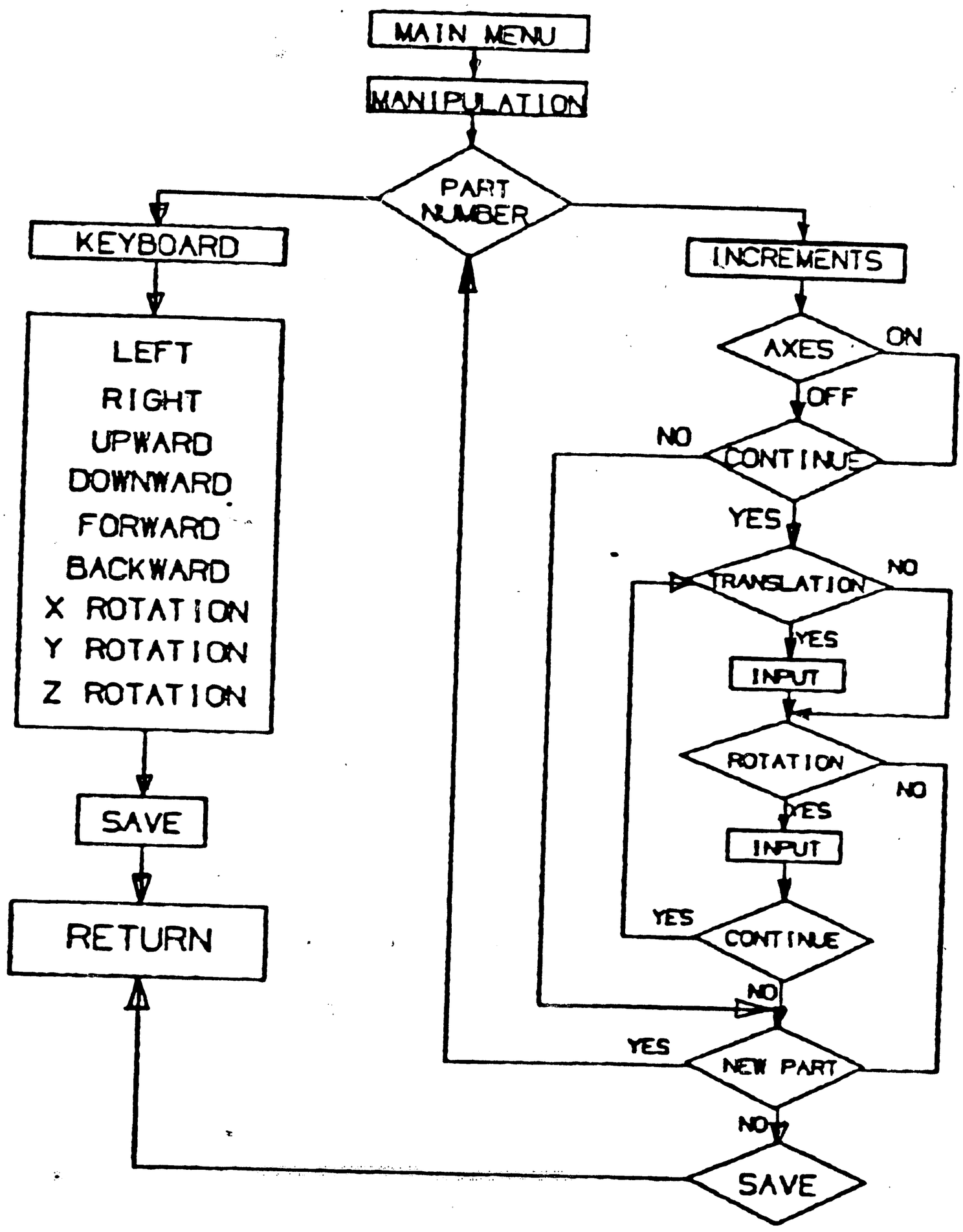
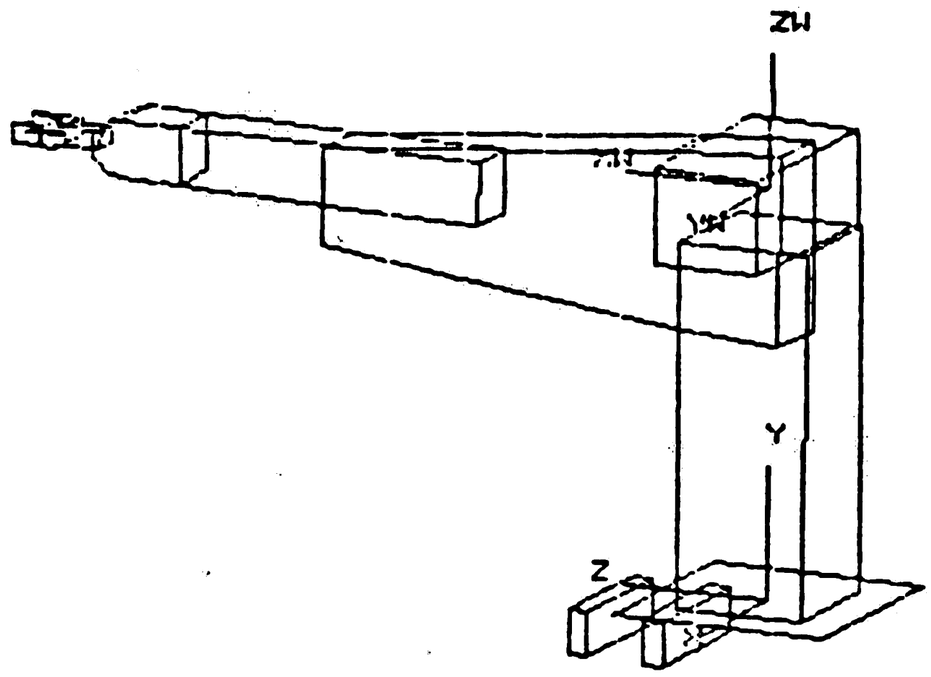
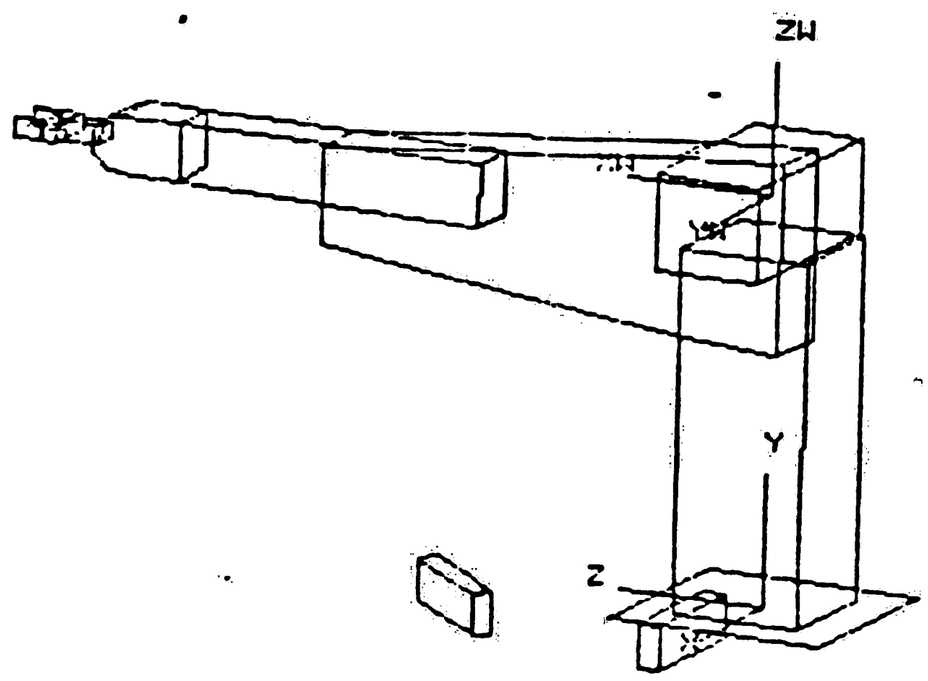


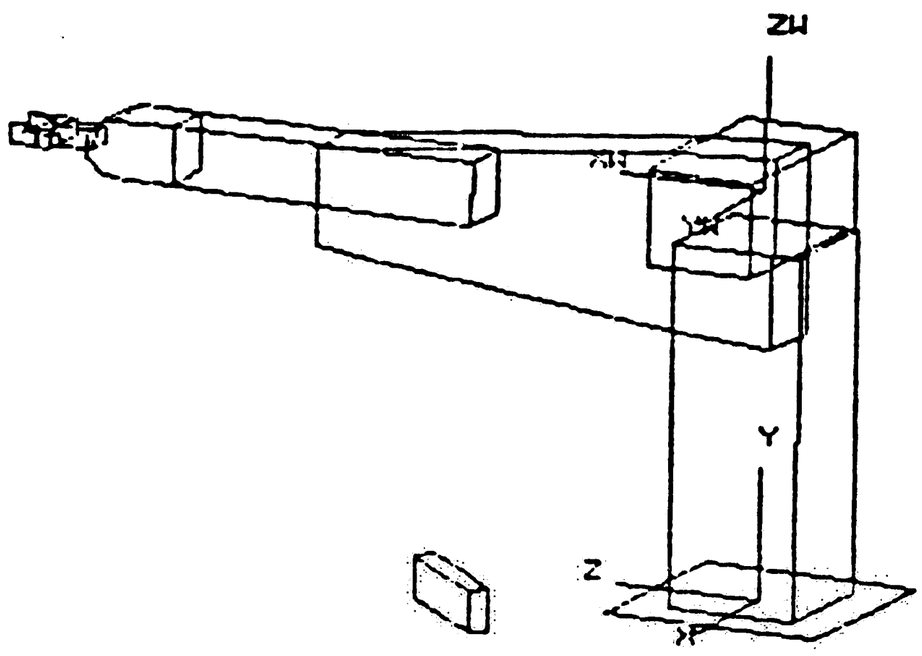
Figure C-6. Manipulation Mode Flow Diagram



a.



b.



c.

Figure C-7. Simulator Output Image for Example

```
                    EDITOR COMMANDS

LEGAL EDITING COMMANDS

     EDIT      . :  INVOKES EDIT MODE

     C_REATE   :  INVOKES CREATION MODE
     I_NSERT   :  ALLOWS FOR LINE INSERTION
     D_ELETE   :  ALLOWS FOR LINE DELETION
     S_AVE     :  SAVES THE FILE
     E_XIT     :  ENDS THE EDIT SESSION

TYPE "EDIT" AND THEN PROGRAM NAME TO START EDIT SESSION
```

Figure C-8. Simulator Output Image for Example

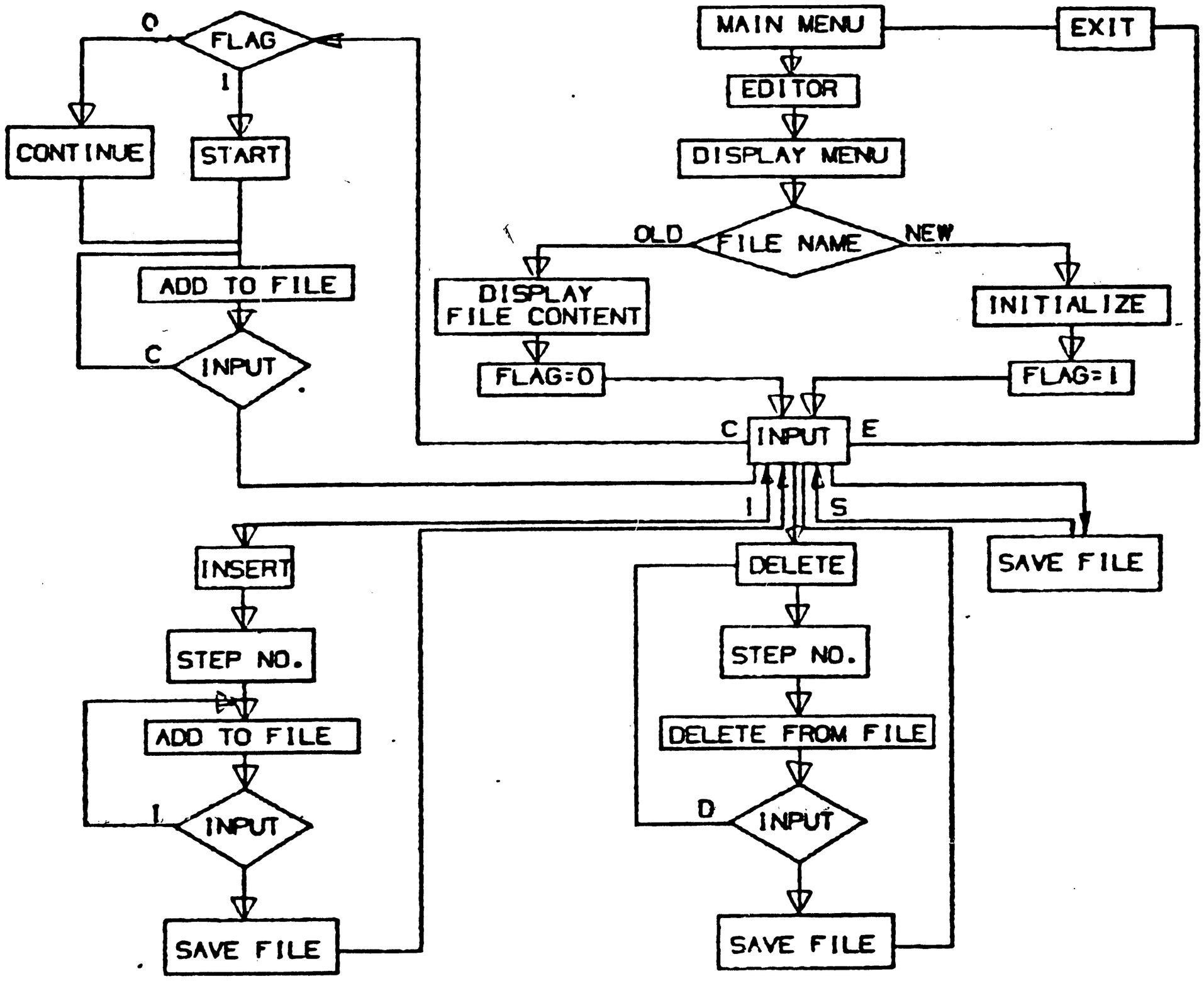


Figure C-9. Edit Mode Flow Diagram

PENDANT MODE

ENTER COMMANDS BY PRESSING:

1 - <JOINT 1> TO ROTATE ABOUT WRIST
2 - <JOINT 2> TO ROTATE ABOUT SHOULDER
3 - <JOINT 3> TO ROTATE ABOUT ELBOW
4 - <JOINT 4> TO ROTATE ABOUT FOREARM
5 - <JOINT 5> TO ROTATE CLAMP UP AND DOWN
6 - <JOINT 6> TO ROTATE CLAMP <CLOCKWISE/COUNTERCLOCKWISE>

O - <OPEN> TO OPEN THE CLAMP
C - <CLOSE> TO CLOSE THE CLAMP

W_ORLD : TO MOVE MODEL IN WORLD MODE
T_OOL : TO MOVE MODEL IN TOOL MODE

I_NCREASE : TO INCREASE THE ROTATIONAL INCREMENT
D_ECREASE : TO DECREASE THE ROTATIONAL INCREMENT
N_EGATIVE : TO CHANGE SIGN OF THE ROTATIONAL INCREMENT

R_ETURN : TO RETURN TO THE TOP OF A MENU
V_AL-II : BEFORE ENTERING A VAL COMMAND

Figure C-10. Simulator Output Image for Example

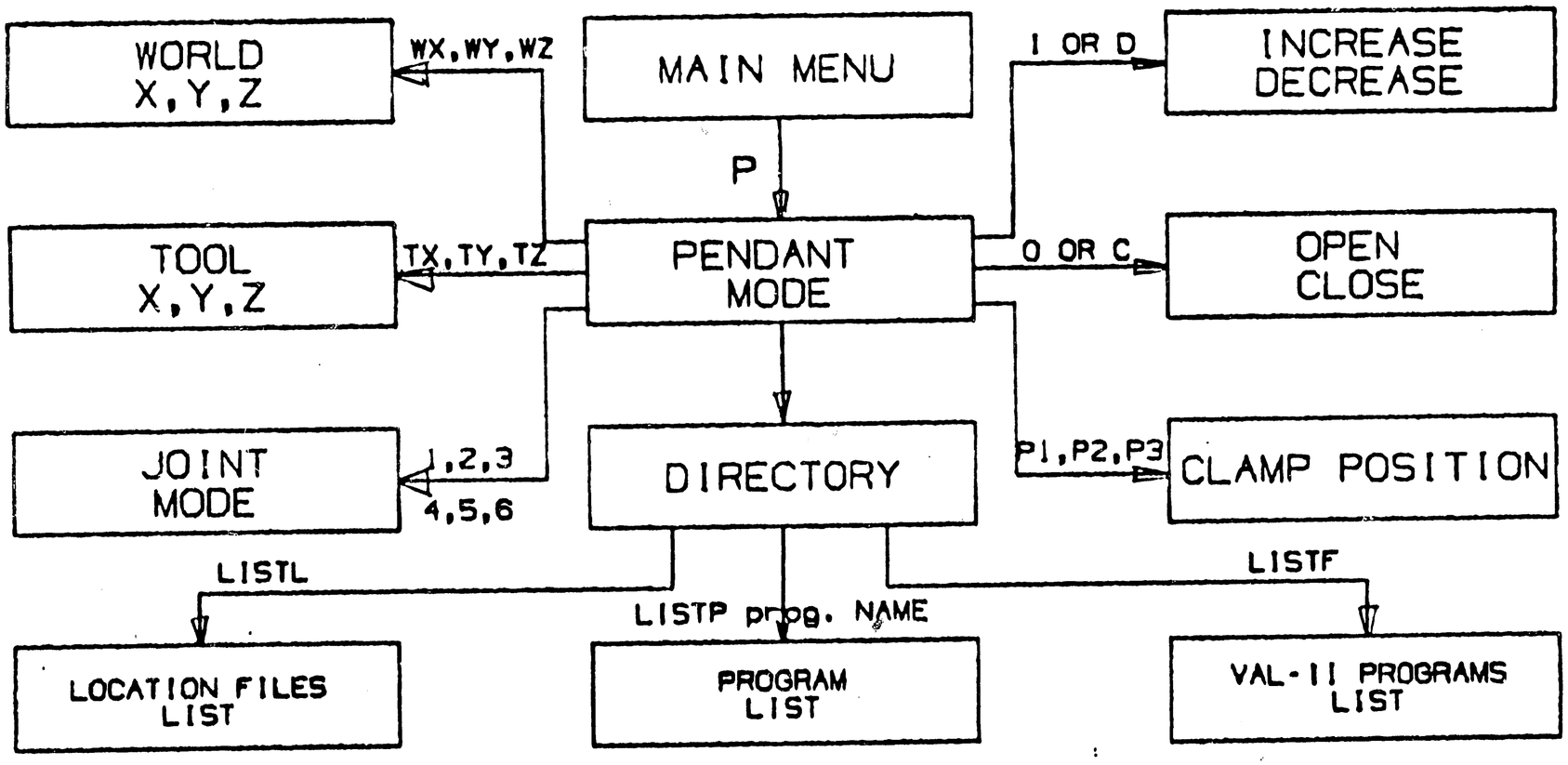
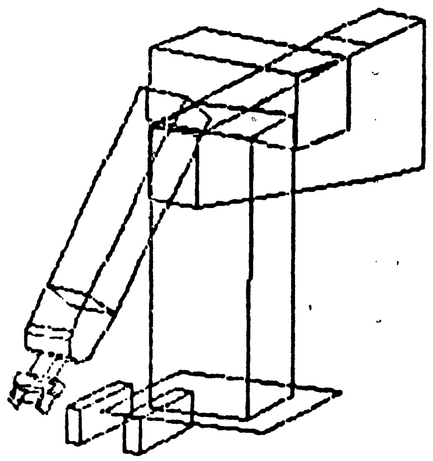


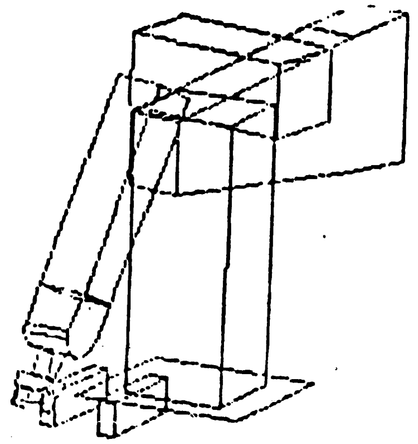
Figure C-11. Pendant Mode Flow Diagram

CLAMP IS NEAR PART 10



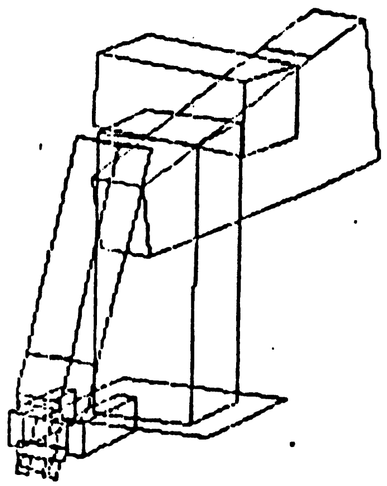
a.

CLAMP IS NEAR PART 10
10 COLLIDES WITH 7
10 COLLIDES WITH 6



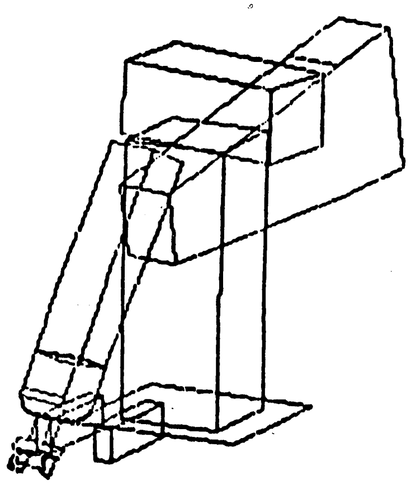
b.

CLAMP IS NEAR PART 10
10 COLLIDES WITH 6
PART 10 IS NEAR PART 5
10 COLLIDES WITH 5



c.

CLAMP IS NEAR PART 10
10 COLLIDES WITH 6
10 COLLIDES WITH 7
PART 10 IS NEAR PART 5
10 COLLIDES WITH 5



d.

Figure C-12. Simulator Output Image for Example 174

VAL-II MENU

ENTER COMMANDS BY TYPING IN:

```
READY          - VERTICAL CONFIGURATION
HERE P         - DEFINE CONFIGURATION P
MOVE P         - JOINT COORDINATE TO P
MOVES P        - STRAIGHT-LINE TO P
OPENI(X)       - OPEN CLAMP BY X%
CLOSEI(X)      - CLOSE CLAMP BY X%
SPEED S        - PERCENT OF FULL SPEED
WHERE          - DISPLAY POSITIONAL INFORMATION
APPRO P,D      - JOINT COORDINATE APPROACH
APPROS P,D     - STRAIGHT LINE APPROACH
STOP           - END VAL-II PROGRAM
HALT           - LIKE STOP (SEE CTRL_C)
DEPART D       - JOINT COORDINATE
DEPARTS D      - STRAIGHT LINE DEPART
DELAY N        - DELAY PROGRAM N SECONDS
PAUSE          - RETURN CONTROL TO USER
RIGHTY         - RIGHT HANDED ROBOT
LEFTY          - LEFT HANDED ROBOT
ABOVE          - ELBOW ABOVE WRIST
BELOW          - ELBOW BELOW WRIST
FLIP           - NEGATIVE JOINT 5 ANGLES
NOFLIP         - POSITIVE JOINT 5 ANGLES
```

VAL-II MENU CONTINUED

ENTER COMMANDS BY TYPING IN:

```
;                - BEFORE COMMENTS
TYPE (K=         - WRITES THE VALUE OF X
STATUS           - WRITES ROBOT STATUS (SPEED, PROG.  LOOP NO.
SET P1=P2        - EQUATE P1 TO P2
DRAW (X,Y,Z      - STRAIGHT LINE INCREMENT
DRIVE J,A,S      - MOVE JT. J,  A DEG. AT SP. S
BASE (X,Y,Z,R    - SHIFT AND ROTATE ROBOT
SHIFT P,X,Y,Z    - SHIFT POSITION P
EXECUTE NAME,N-  EXECUTE NAME,  N TIMES
CTRL_C           - TO ABORT A VAL-II PROGRAM
GRASP X,NUM      - CLOSE CLAMP BY X%. ELSE GO TO STATEMENT NO. (NUM)
HELP             - WRITES DESCRIPTION OF VAL-II COMMANDS
ALIGN            - ALIGNS THE TOOL Z_AXIS WITH NEAREST WORLD COORDS. Z_AXIS
VAR 1 = VAR 2    - SETS VAR. ON THE LEFT EQUAL TO ONE ON THE RIGHT OF EQUAL SIGN

HELP             - DISPLAYS HELP MENU

THEN PRESS:

C_ONTINUE        : TO CONTINUE TYPING IN VAL COMMANDS
W_IPE            : TO WIPE THE VT100 CHARACTERS FROM THE SCREEN
R_ETURN          : TO RETURN TO THE TOP OF A MENU
M_ODEL           : TO DISPLAY THE MODEL
D_EFINE          : TO DEFINE ORIENTATIONS
```

Figure C-13. Simulator Output Image for Example

```
14
MM
W
1.8
        440.00      -50.00     -320.00
          0.00        0.00        0.00
          0.00        0.00       70.00
          0.00       30.00       70.00
          0.00       30.00        0.00
         50.00        0.00        0.00
         50.00        0.00       70.00
         50.00       30.00       70.00
         50.00       30.00        0.00
1.8
        440.00      105.83     -320.00
          0.00        0.00        0.00
          0.00        0.00       70.00
          0.00       30.00       70.00
          0.00       30.00        0.00
         50.00        0.00        0.00
         50.00        0.00       70.00
         50.00       30.00       70.00
         50.00       30.00        0.00
1.8
       -463.51      500.00     -320.00
          0.00        0.00        0.00
          0.00        0.00       40.00
          0.00      600.00       40.00
          0.00      600.00        0.00
        500.00        0.00        0.00
        500.00        0.00       40.00
        500.00      600.00       40.00
        500.00      600.00        0.00
1.8
       -463.51      500.00     -320.00
         40.00        0.00        0.00
         40.00        0.00     -250.00
         40.00       40.00     -250.00
         40.00       40.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       40.00     -250.00
          0.00       40.00        0.00
1.8
       -463.51     1060.00     -320.00
         40.00        0.00        0.00
         40.00        0.00     -250.00
         40.00       40.00     -250.00
         40.00       40.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       40.00     -250.00
          0.00       40.00        0.00
1.8
         -2.51     1060.00     -320.00
         40.00        0.00        0.00
         40.00        0.00     -250.00
         40.00       40.00     -250.00
         40.00       40.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       40.00     -250.00
          0.00       40.00        0.00
1.8
         -2.51      500.00     -320.00
         40.00        0.00        0.00
         40.00        0.00     -250.00
         40.00       40.00     -250.00
         40.00       40.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       40.00     -250.00
          0.00       40.00        0.00
1.8
        595.00     -600.00     -320.00
          0.00        0.00        0.00
          0.00        0.00      -20.00
          0.00      800.00      -20.00
          0.00      800.00        0.00
       -250.00        0.00        0.00
       -250.00        0.00      -20.00
       -250.00      800.00      -20.00
       -250.00      800.00        0.00
1.8
        575.00     -600.00     -320.00
         20.00        0.00        0.00
         20.00        0.00     -250.00
         20.00       20.00     -250.00
         20.00       20.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       20.00     -250.00
          0.00       20.00        0.00
1.8
        575.00      200.00     -320.00
         20.00        0.00        0.00
         20.00        0.00     -250.00
         20.00       20.00     -250.00
         20.00       20.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       20.00     -250.00
          0.00       20.00        0.00
1.8
        345.00      200.00     -320.00
         20.00        0.00        0.00
         20.00        0.00     -250.00
         20.00       20.00     -250.00
         20.00       20.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       20.00     -250.00
          0.00       20.00        0.00
1.8
        345.00     -600.00     -320.00
         20.00        0.00        0.00
         20.00        0.00     -250.00
         20.00       20.00     -250.00
         20.00       20.00        0.00
          0.00        0.00        0.00
          0.00        0.00     -250.00
          0.00       20.00     -250.00
          0.00       20.00        0.00
1.8
        495.00      400.00     -320.00
          0.00        0.00        0.00
          0.00        0.00     -150.00
        176.78      176.78     -150.00
        176.78      176.78        0.00
       -176.78      176.78        0.00
       -176.78      176.78     -150.00
          0.00      353.56     -150.00
          0.00      353.56        0.00
1.8
        495.00      400.00     -470.00
          0.00        0.00        0.00
          0.00        0.00     -100.00
        176.78      176.78     -100.00
        176.78      176.78        0.00
       -176.78      176.78        0.00
       -176.78      176.78     -100.00
          0.00      353.56     -100.00
          0.00      353.56        0.00
```

Figure C-14. Program PROCESS

Setup data file

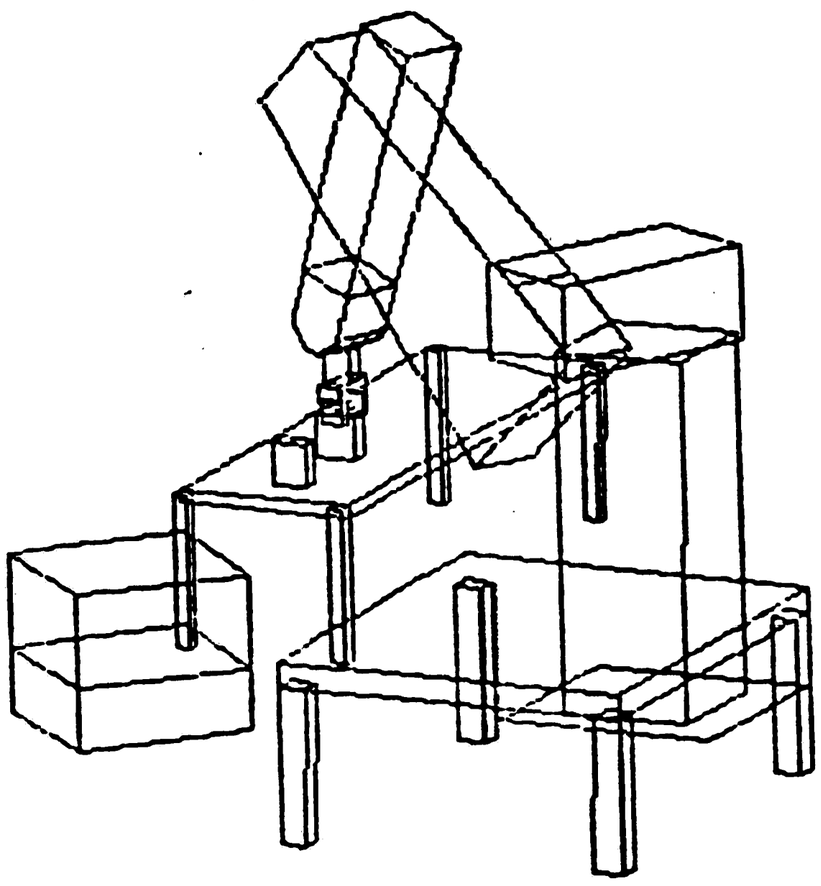
```
POINT A, 465. , -45. , -245. , 89. , 91. , 0.
POINT B, 465. , 120. , -245. , 90. , 90. , 0.
POINT C, -180. , 600. , -204. 16, 90. , 90. , -90.
POINT D, -100. , 600. , -204. 16, 90. , 90. , -90.
POINT E, 425. , 590. , -220. , 90. , 90. , -45.
POINT F, 425. , 590. , -400. , 90. , 90. , -45.
```

Figure C-15. Program Process location data file

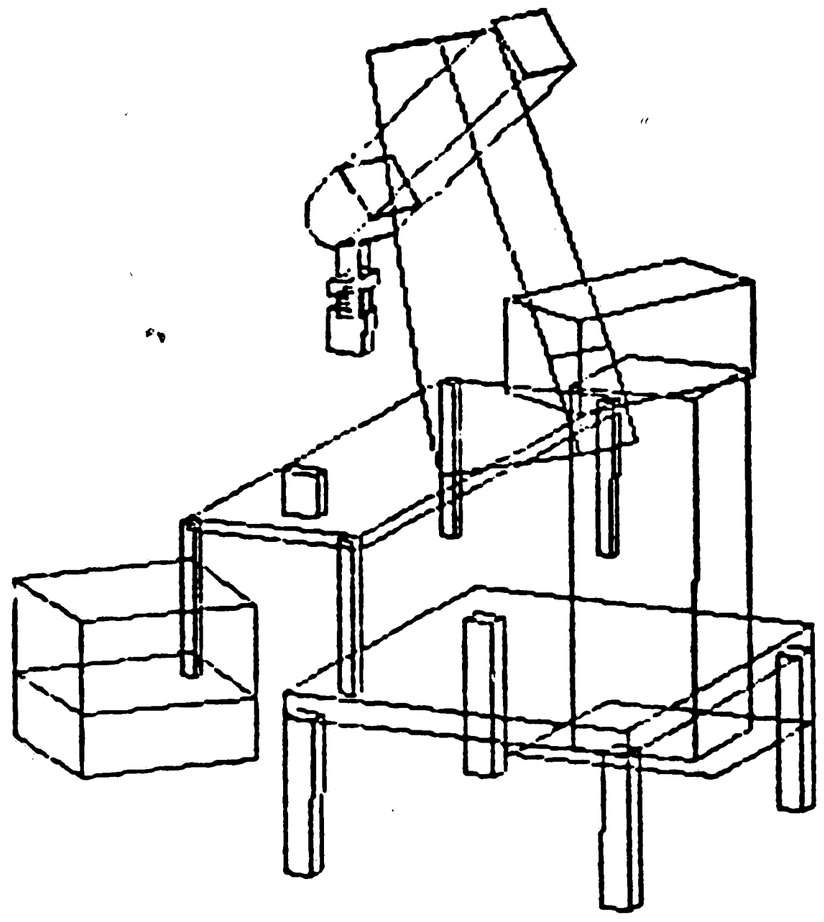
```
; PROGRAM : PROCESS
; THIS PROGRAM SIMULATES A PART PROCESSING APPLICATIOI
READY
LEFTY
ABOVE
APPRO A, -50
TYPE PICK UP THE FIRST PART
MOVE A
CLOSEI 50
DEPARTS 100
MOVES E
MOVES F
TYPE PROCESS THE FIRST PART
DELAY 1
DEPARTS 200
APPROS C, -50
MOVES C
OPENI 100
DEPARTS 100
APPROS B, -50
MOVES B
TYPE PICK UP THE SECOND PART
CLOSEI 50
DEPARTS 100
MOVES E
TYPE PROCESS THE SECOND PART
DELAY 1
SHIFT E, 0, 0, -200
MOVES E
APPROS D, -50
MOVES D
OPENI 100
DEPARTS 100
READY
STOP
```

Figure C-16. Program Process

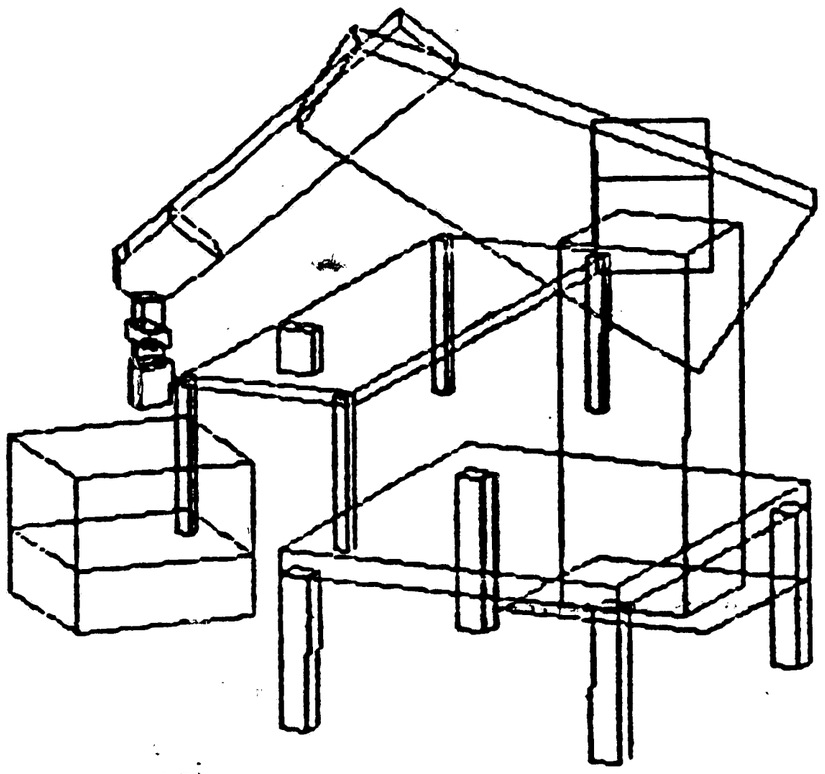
IT IS POSSIBLE TO GRASP THIS PART



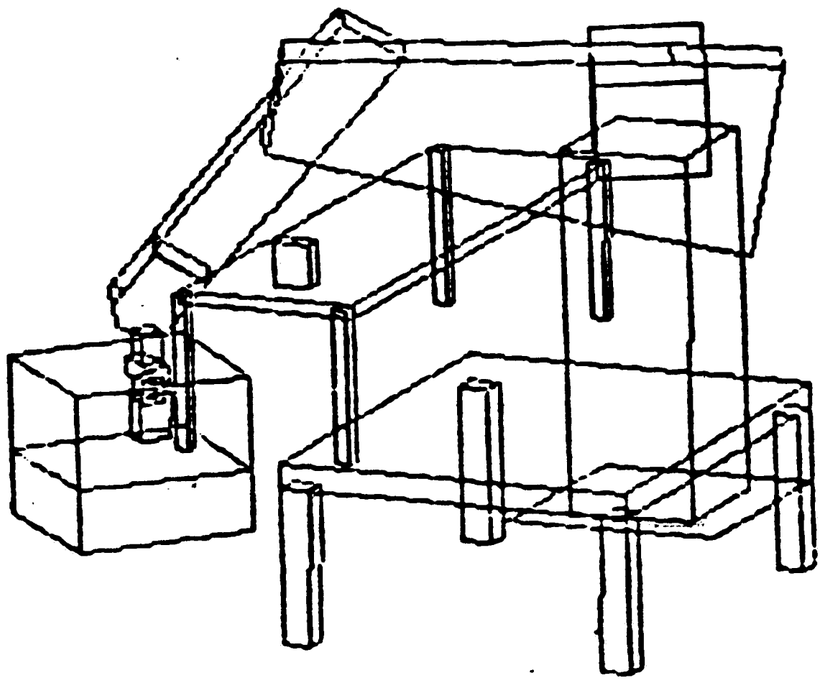
a.



b.

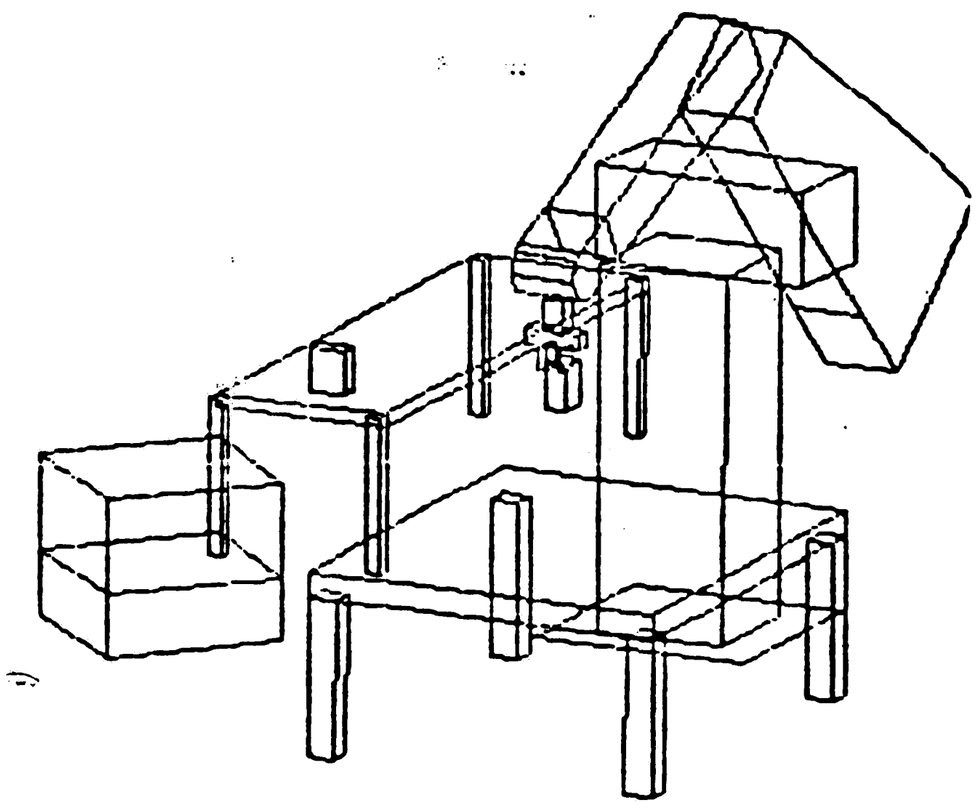


c.

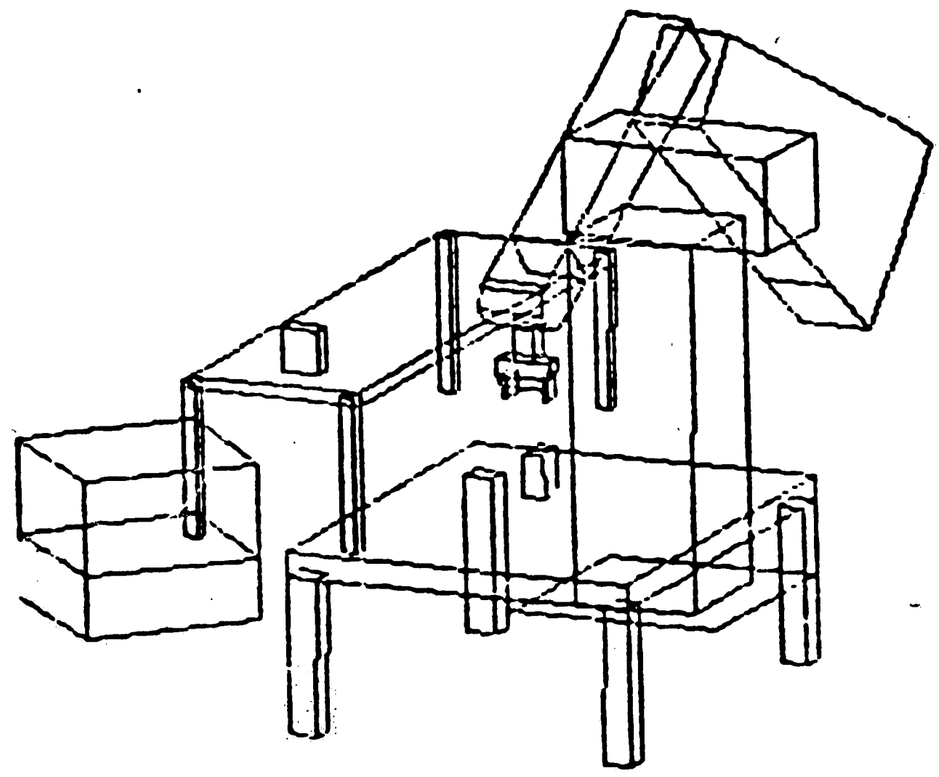


d.

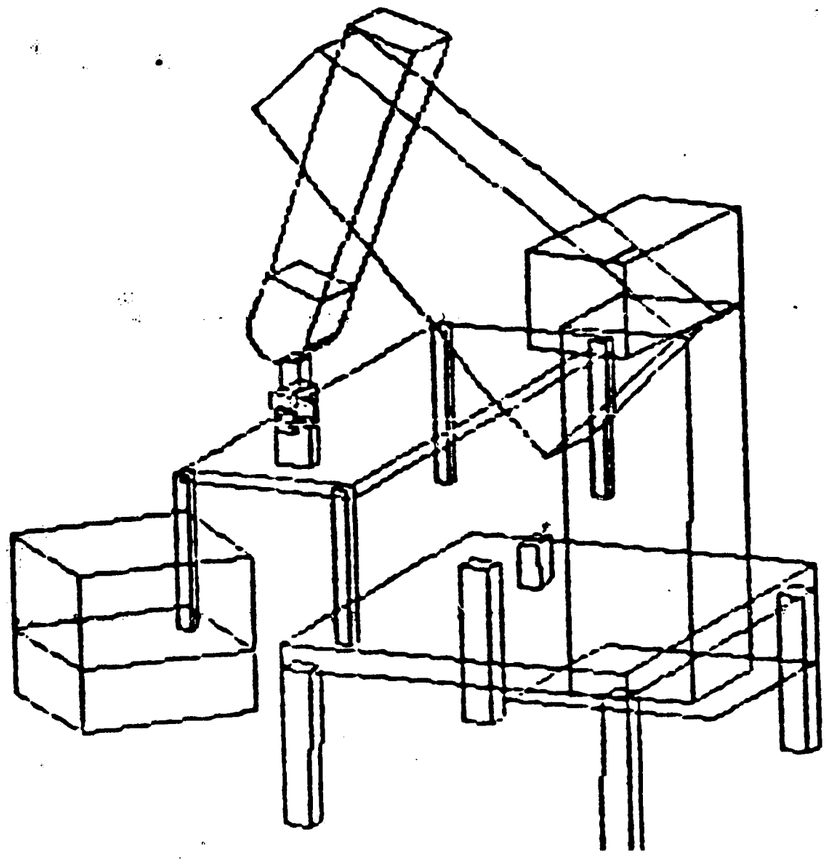
Figure C-17. Simulator Output Image for Example



a.



b.



c.

Figure C-18. Simulator Output Image for Example

```
      ; PROGRAM CONV1
      ; This program simulates a moving conveyor belt.
      READY
      LEFTY
      COL=1
      ROW=3
100   COL=COL+1
      TYPE COL=
      OPENI 100
      NOFLIP
      APPRO A,50
      MOVE A
      MOVE B
      CLOSEI 50
      DEPARTS 100
      MOVE F
      OPENI 100
      DEPARTS 100
      IF COL.EQ.2,THEN
      GOTO 300
      ELSE
      APPRO C,-50
300   ENDIF
      TYPE ROW=
      WHILE COL.LT.ROW.DO
      COL=COL+1
      TYPE COL=
      MOVE C
      CLOSEI 40
      DEPARTS 100
      MOVES L
      OPENI 100
      DEPARTS 150
      END
      DEPARTS 100
      CALL CONV2
      DO
      MOVE L
      CLOSEI 60
      DEPARTS 100
      MOVE C
      OPENI 100
      DRIVE 2,-30,100
      UNTIL COL.EQ.ROW.
      TYPE ROW=
      TYPE COL=
      RIGHTY
      MOVE M
      CLOSEI 50
      DEPARTS 150
      MOVE H
      OPENI 100
      DEPARTS 100
      MOVE A
      RETURN
      STOP
```

Figure C-19. Example Program Command file

```
; PROGRAM CONV2
OPENI 100
MOVE A
APPRO F,-50
MOVE F
CLOSEI 40
DEPARTS 150
MOVES B
OPENI 100
DEPART 100
MOVE A
RETURN
STOP
```

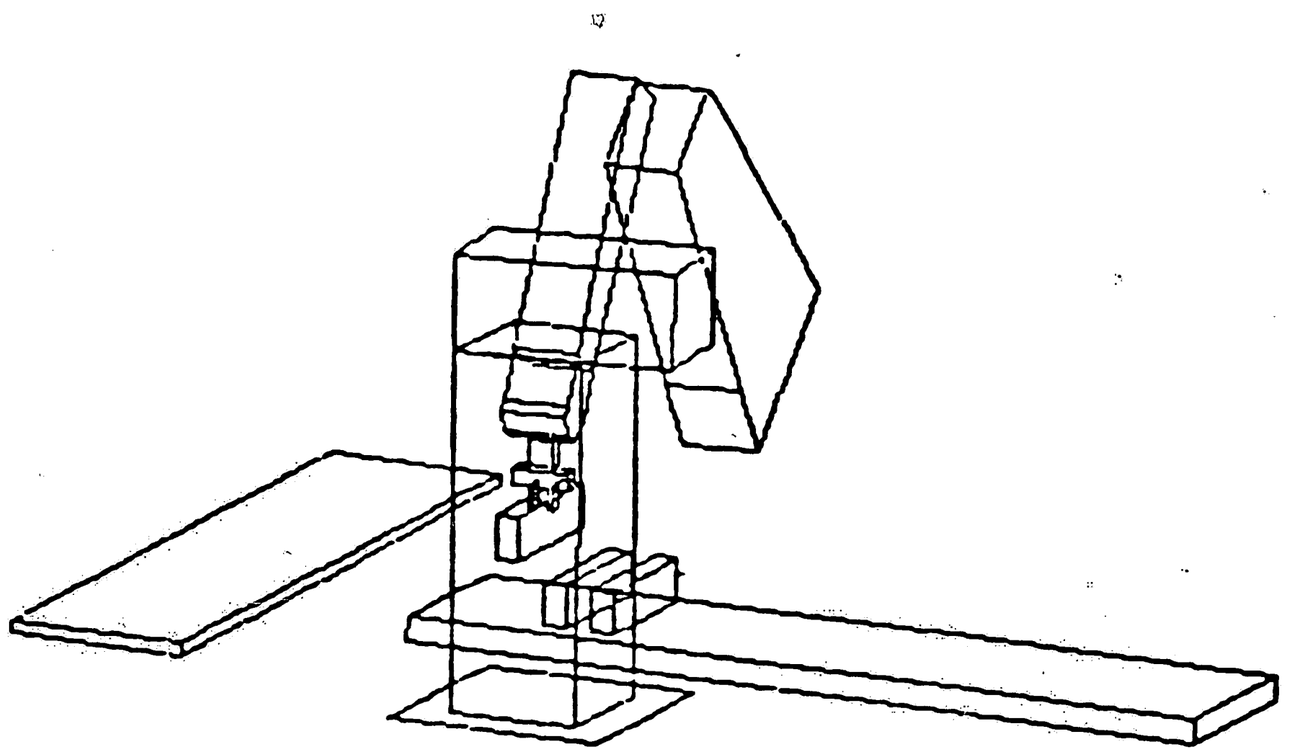
Figure C-20. Example Program Command file (subroutine)

```
POINT A,-215.78,534.31,-105.29,-174.,56.,0.
POINT B,-218.37,475.88,-262.64,-173.,73.,0.
POINT C,-294.,475.,-262.,-174.,60.,0.
POINT D,495.58,274.82,-430.,11.75,74.44,3.5
POINT F,523.58,274.82,-430.,11.75,74.44,3.5
POINT G,479.30,272.54,-363.59,96.24,57.09,-7.45
POINT H,396.,308.,-421.,146.,90.,148.
POINT L,464.60,263.35,-431.12,5.98,62.57,.27
POINT M,-369.,450.,-257.,-156.,90.,27.52
```

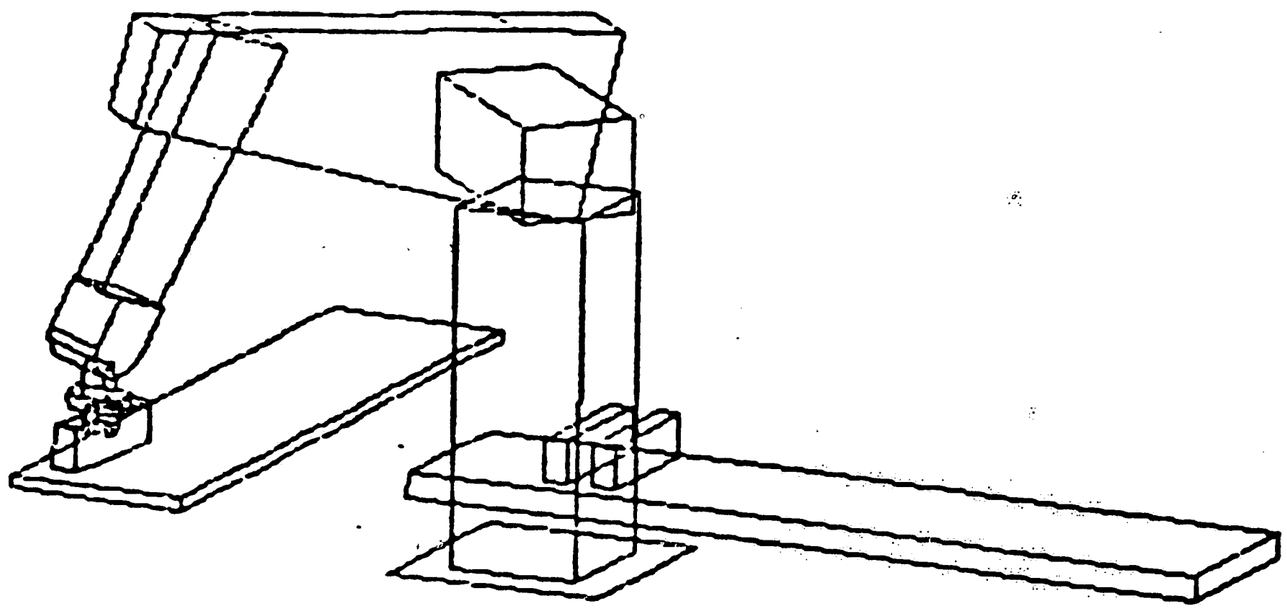
Figure C-21. Example Program Location data file

```
5
MM
W
1.8
                -241.00         400.00        -330.00
                   0.00           0.00           0.00
                   0.00           0.00          70.00
                   0.00         200.00          70.00
                   0.00         200.00           0.00
                  30.00           0.00           0.00
                  30.00           0.00          70.00
                  30.00         200.00          70.00
                  30.00         200.00           0.00
1.8
                -311.00         400.00        -330.00
                   0.00           0.00           0.00
                   0.00           0.00          70.00
                   0.00         200.00          70.00
                   0.00         200.00           0.00
                  30.00           0.00           0.00
                  30.00           0.00          70.00
                  30.00         200.00          70.00
                  30.00         200.00           0.00
1.8
                -381.00         400.00        -330.00
                   0.00           0.00           0.00
                   0.00           0.00          70.00
                   0.00         200.00          70.00
                   0.00         200.00           0.00
                  30.00           0.00           0.00
                  30.00           0.00          70.00
                  30.00         200.00          70.00
                  30.00         200.00           0.00
1.8
                -100.00         390.00        -330.00
                   0.00           0.00           0.00
                   0.00           0.00         -40.00
                   0.00         250.00         -40.00
                   0.00         250.00           0.00
               -1150.00           0.00           0.00
               -1150.00           0.00         -40.00
               -1150.00         250.00         -40.00
               -1150.00         250.00           0.00
1.8
                 595.00        -600.00        -500.00
                   0.00           0.00           0.00
                   0.00           0.00         -20.00
                   0.00        1000.00         -20.00
                   0.00        1000.00           0.00
                -250.00           0.00           0.00
                -250.00           0.00         -20.00
                -250.00        1000.00         -20.00
                -250.00        1000.00           0.00
```

Figure C-22. Example Program Setup data file

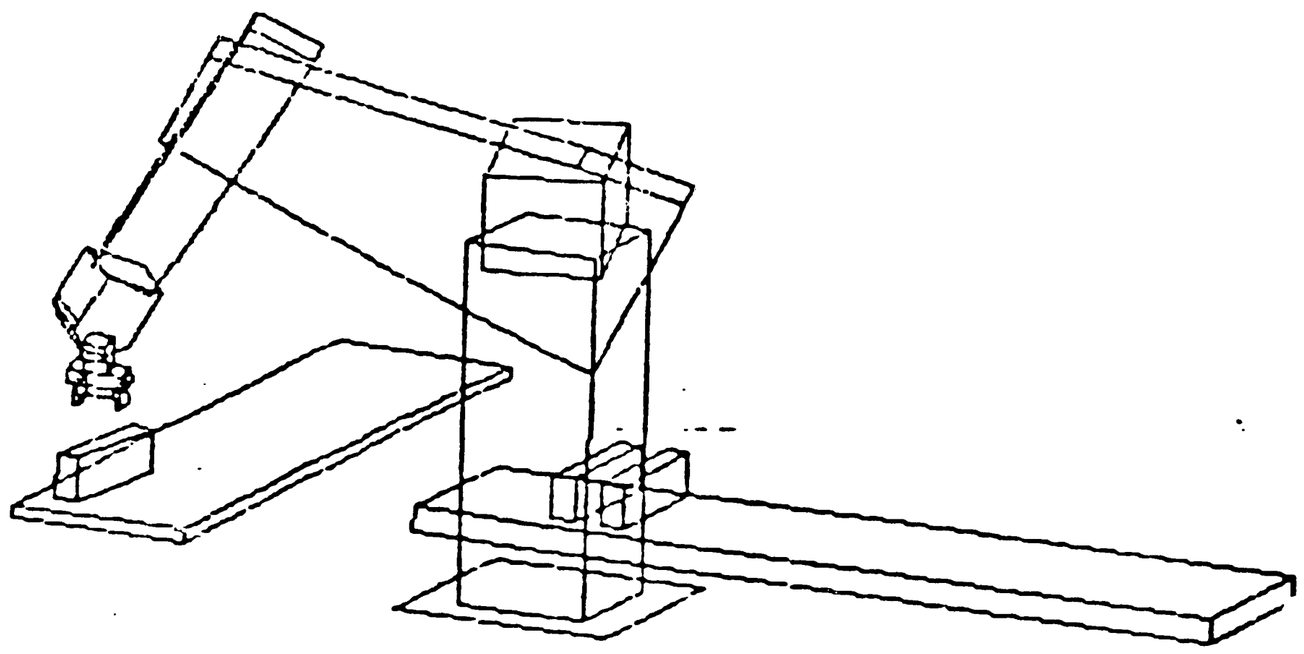


a.

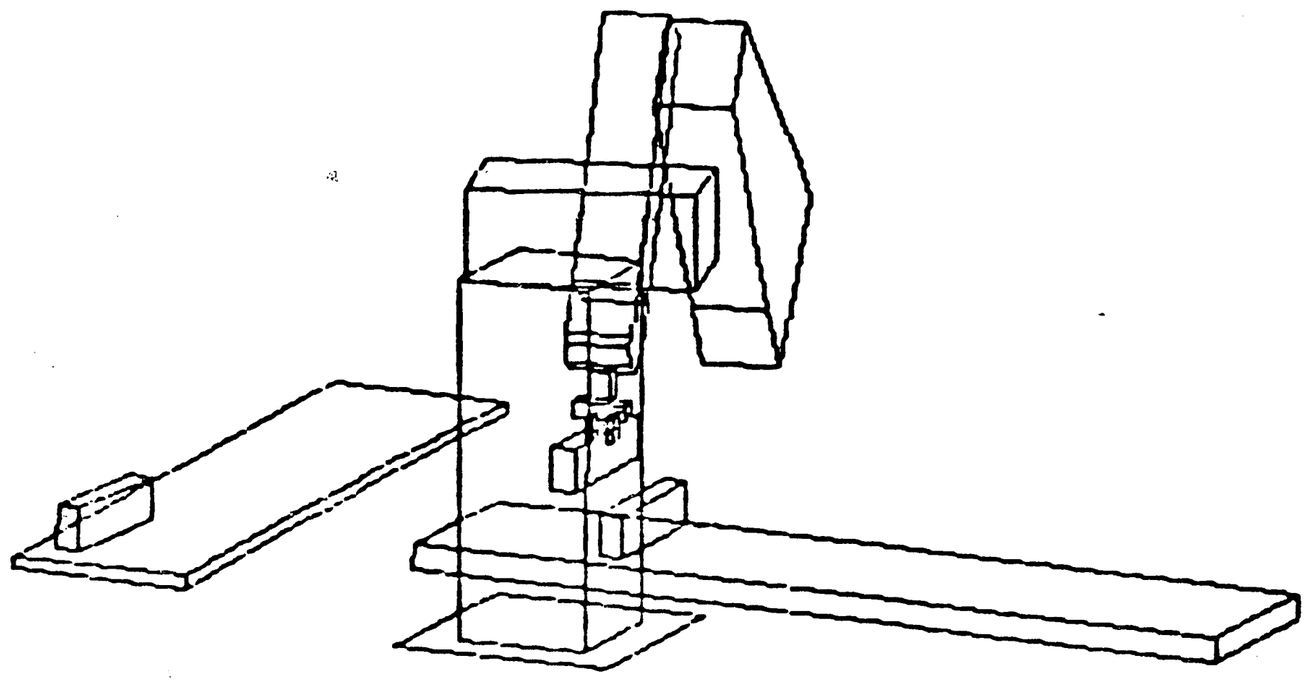


b.

Figure C-23. Simulator Output Image for Example

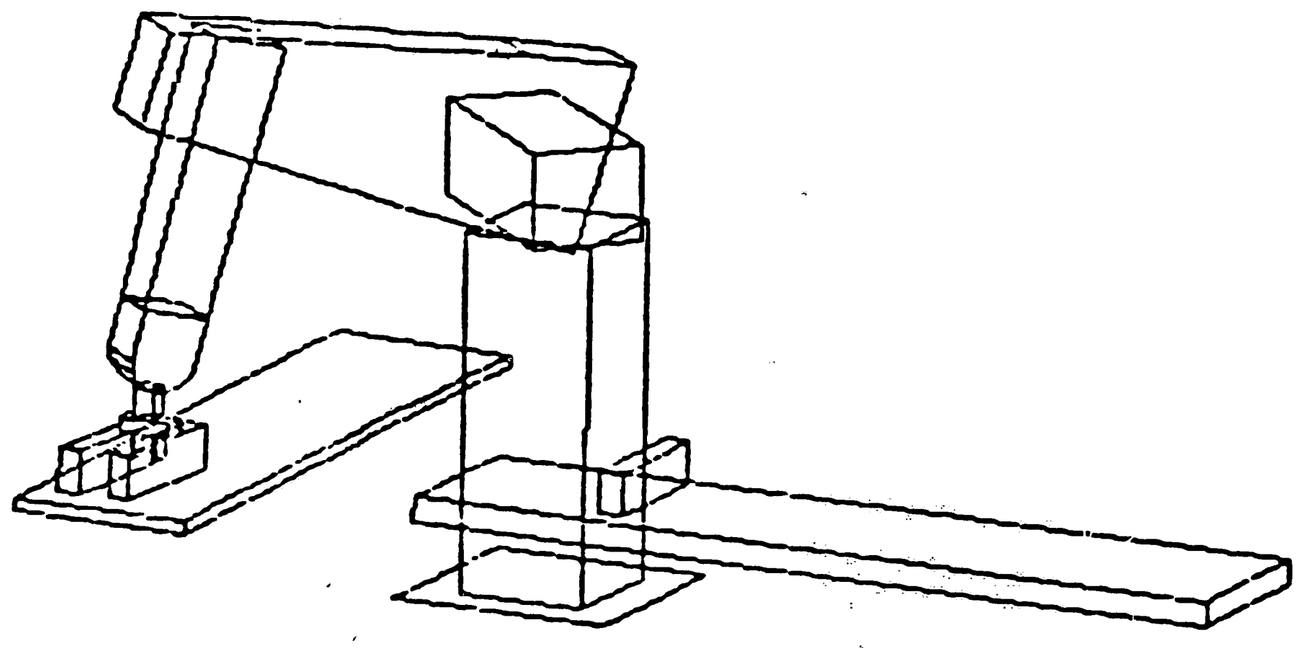


a.

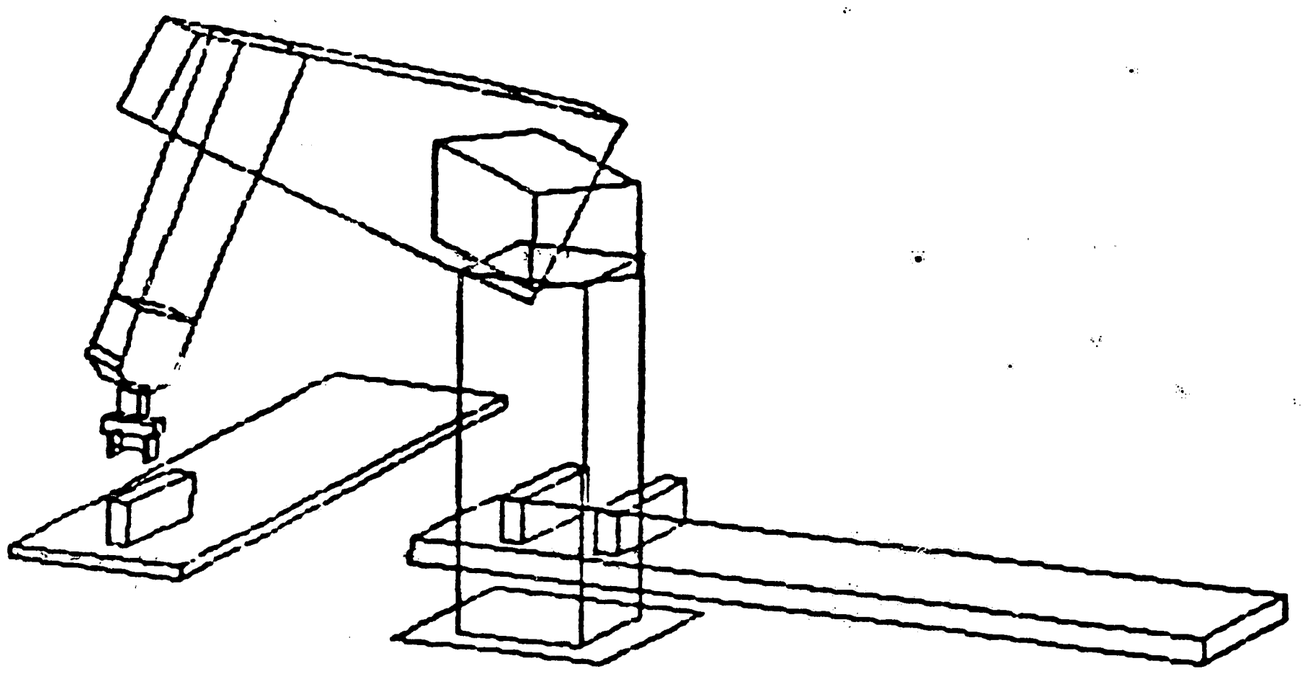


b.

Figure C-24. Simulator Output Image for Example

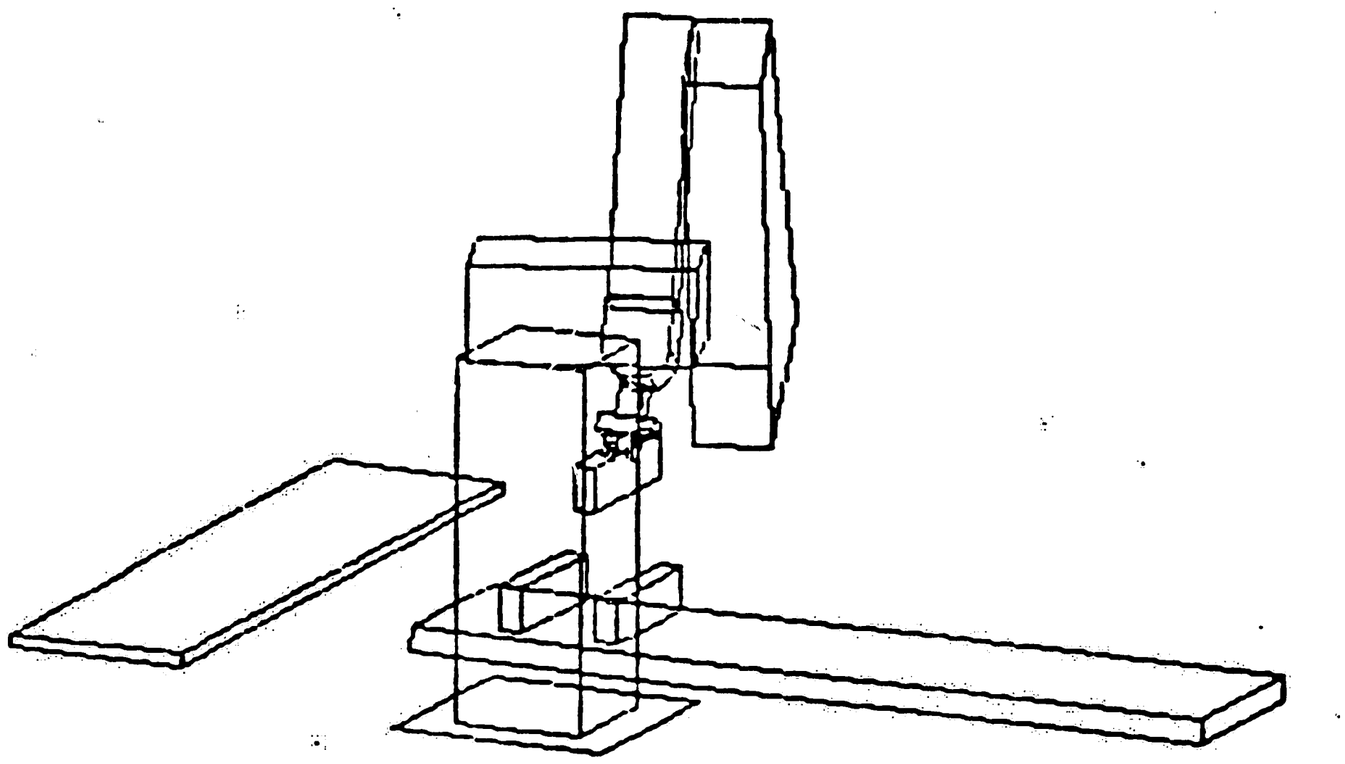


a.

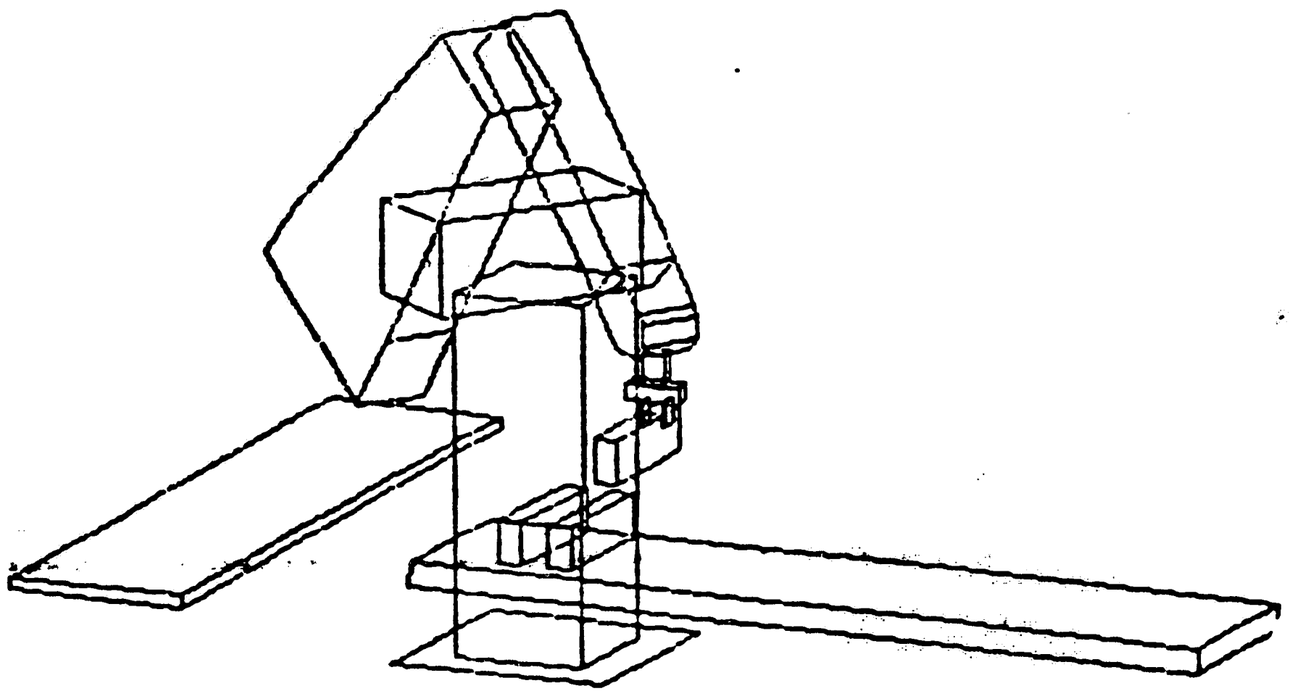


b.

Figure C-25. Simulator Output Image for Example

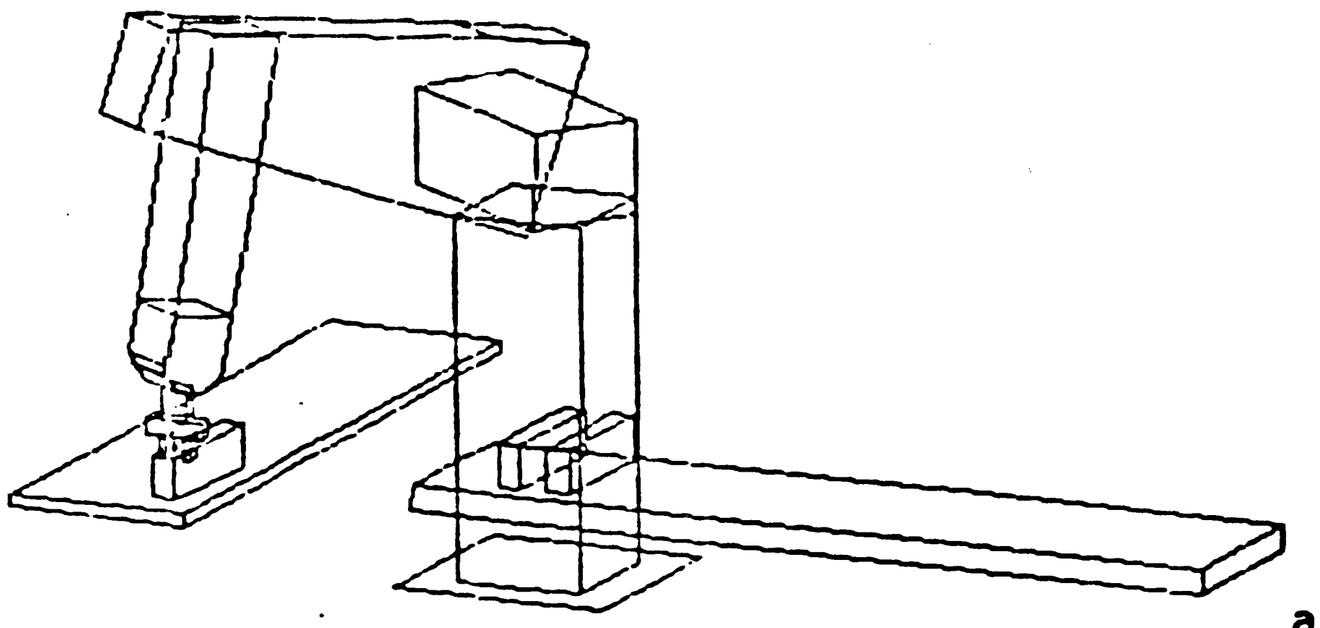


a.

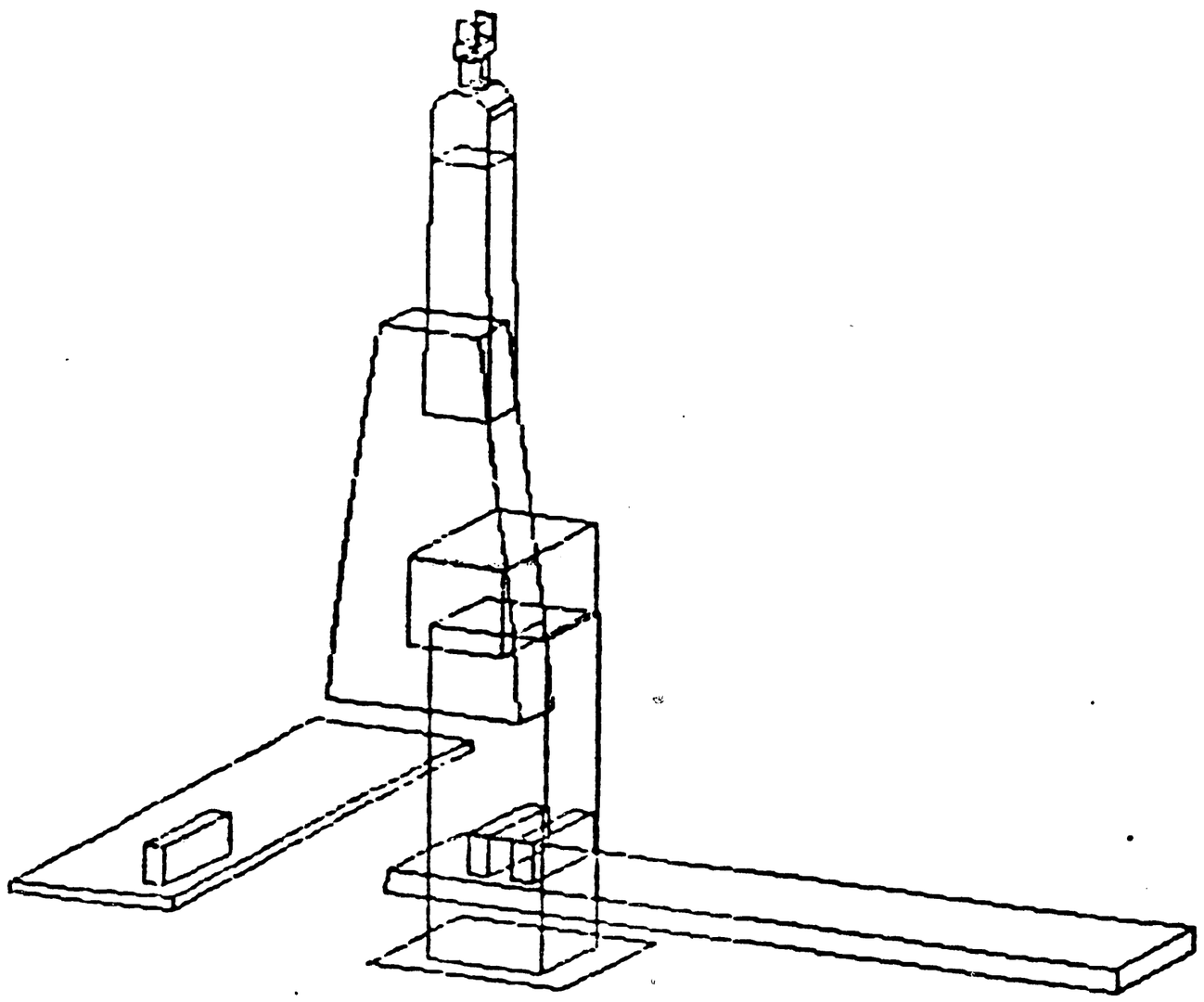


b.

Figure C-26. Simulator Output Image for Example



a.



b.

Figure C-27. Simulator Output Image for Example

VITA

Seied Abrishamchian Langrudi is a research assistant in the Computer-Aided Design Laboratory at Lehigh University. His current interests include Automation, Dynamic Systems Control and Finite Element Methods.

He received a B.S. degree in Mechanical Engineering Technology from Point Park College in 1983.

He instructed several short courses in robot work-cell simulation and computer-aided design. He was also a teaching assistant for a graduate level course in Product Design and Analysis.

Upon completion of his M.S. in Mechanical Engineering he will become a member of the technical staff at Graham Engineering Corporations.