



National Library  
of Canada

Bibliothèque nationale  
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada  
K1A 0N4

## NOTICE

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

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

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

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

## AVIS

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

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

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

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

UNIVERSITY OF ALBERTA

GRAPHICAL INTERFACE DESIGN FOR REAL TIME ROBOT/WORKCELL  
SIMULATION

BY



ARA SIMONIAN

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment  
of the requirement for the degree of Master of Science.

DEPARTMENT OF MECHANICAL ENGINEERING

Edmonton, Alberta

Spring 1992



National Library  
of Canada

Bibliothèque nationale  
du Canada

Canadian Theses Service    Service des thèses canadiennes

Ottawa, Canada  
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-73110-9

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: ARA SIMONIAN

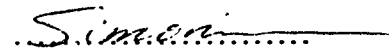
TITLE OF THESIS: GRAPHICAL INTERFACE DESIGN FOR REAL TIME  
ROBOT/WORKCELL SIMULATION

DEGREE: MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED: SPRING 1992

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



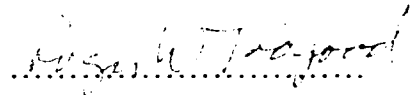
Ara Simonian  
3A 9006, 112 street  
Edmonton, Alberta

Date: 22, April, 1992

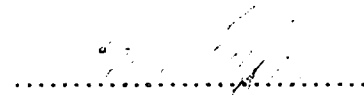
UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

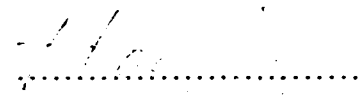
The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Graphical Interface Design for Real Time Robot/Workcell Simulation** submitted by Ara Simonian in partial fulfillment of the requirements for the degree of Master of Science.



Dr. R. W. Toogood



Dr. K. R. Fyfe



Dr. H. Zhang

Date: *21, April, 1992*

## Abstract

Computer graphics simulation has proven to be an effective tool in robot design, on-line control, off-line programming and development of advanced control systems. A few commercial programs are available for modelling industrial robots. These often use expensive graphics hardware and offer sophisticated, complete models of the manipulator (for example, solid modelling, surface rendering multiple light sources, etc.). Such systems are very expensive, often equivalent to the cost of a robot.

The present work consists of the development of a real time graphical simulation software for an *Excalibur* robotic arm. The study describes the different aspects of an interactive graphical interface for the robotic systems. The usefulness of the system is demonstrated for the cases of off-line programming and remote on-line manipulation. Two different virtual view cameras are evaluated and the results reveal the advantages of each. In particular, the gripper view camera has proven to be very a powerful viewing tool.

Furthermore, the study investigated the factors involved in the man-machine interface. The human factors engineering aspects of the interface were thoroughly considered to provide the user with a practical and user friendly interface. The study investigated the different input devices and also the command tools that are made available to the program.

The result of this work reveals the possibility of successful implementation of such an interface on the PCs. This will make robotics more accessible to small and medium sized companies and promote the use of robotics in industry. Also the affordable programming system on PCs promises an even easier development and testing tool for high level command languages for robotic systems.

## Table of content

### Chapter 1 - Introduction

1. Introduction . . . . .	1
2. Components of the System . . . . .	7
System Outline . . . . .	7
Manipulator . . . . .	8
Master Arm . . . . .	8
Controller . . . . .	9
Host computer . . . . .	11
Programming Language . . . . .	11

### Chapter 2 - Principles of human computer interaction

2.1 Introduction . . . . .	12
2.2 Basic principles . . . . .	12
2.3 Interface Analysis . . . . .	13
User analysis . . . . .	13
Task analysis . . . . .	14
2.4 Screen and display requirements . . . . .	14
Menu . . . . .	14
Color . . . . .	15
Displayed information . . . . .	15
Help . . . . .	15
Graphics . . . . .	16
Status Information . . . . .	16
Command language . . . . .	16
System response time . . . . .	16
Command and Input Devices . . . . .	17
Error messages and On-line assistance . . . . .	18
Design concepts for the program . . . . .	18

### Chapter 3 - Program Design

3.1 Introduction . . . . .	20
3.2 Interface considerations . . . . .	22
3.3 Graphical display . . . . .	22

3.4 Program Structure . . . . .	26
Communication with the arm . . . . .	28
World object data file . . . . .	29
View considerations . . . . .	29
View cameras . . . . .	31
"Move arm" mode . . . . .	34
Command tools . . . . .	35
Individual joint control . . . . .	36
Inverse kinematics . . . . .	37
Directly specifying the position and orientation . . . . .	38
Using 3-D cursor to position the end effector . . . . .	38
Relative moves in world coordinates . . . . .	39
Relative moves in gripper coordinates . . . . .	40
Problems with the inverse kinematic related tools . . . . .	41
Simulation . . . . .	42
Teach and playback . . . . .	42
Options . . . . .	43
Use of a mouse as an alternative input device . . . . .	43
Communication concerns . . . . .	44
 Chapter 4 - Program evaluation	
4.1 Introduction . . . . .	45
4.2 Performance of the Program . . . . .	45
Expectations . . . . .	45
On-line commanding . . . . .	46
Off-line programming . . . . .	46
EXCAL evaluation . . . . .	47
Interface analysis . . . . .	47
Animation and framing rate . . . . .	49
On-line commanding . . . . .	55
Off-line programming . . . . .	57
Compatibility with the controller . . . . .	57
4.3 Program testing . . . . .	58
The workcell . . . . .	61
Testing procedure . . . . .	61
User approved features . . . . .	64
User disliked features . . . . .	65
 Chapter 5 - Conclusion . . . . .	66
 References . . . . .	68



## Appendix A - Kinematic Modelling

Introduction . . . . .	70
Conventions used by Denavit-Hartenberg model . . . . .	71
Notation . . . . .	71
"A" matrices . . . . .	72
Joint coordinate convention . . . . .	73
Specification of $T_6$ in terms of A matrices . . . . .	74
Kinematic Equations for <i>Excalibur</i> Manipulator . . . . .	76
Forward Solution . . . . .	78
Inverse Solution . . . . .	79

## Appendix B - Implementation of the program in the 'C' language

Introduction . . . . .	85
Frame coordinates . . . . .	85
Graphic representation of the objects . . . . .	86
Data-Base design . . . . .	86
Transformations . . . . .	87
Perspective projection . . . . .	89
Hidden surface removal . . . . .	90
Drawing a frame . . . . .	91
Grid . . . . .	92
Arm base . . . . .	92
Axis . . . . .	92
World objects . . . . .	92
Arm . . . . .	93
Transformation of the grasped object . . . . .	93
Object grasping procedure . . . . .	95
Object dropping procedure . . . . .	96
Animation . . . . .	99
View camera transformations . . . . .	99
Relative moves . . . . .	101
Moves relative to world coordinate . . . . .	101
Moves relative to gripper . . . . .	101
Menu . . . . .	102

## Appendix C - EXCAL program verification/testing data sheet . . . . . 103

## List of Tables

Table 1 - Summary of some of the robot simulation programs. . . . .	2
Table 2 - Time comparison for different polygon drawing methods . . . . .	50
Table 3 - CPU time usage for image generation on the microcomputers . . . . .	50
Table 4 - Display time of an image for different number of world objects . . . . .	53
Table 5 - Display time of an image for different number of world objects . . . . .	54
Table 6 - Denavit-Hartenberg parameters for <i>Excalibur</i> . . . . .	76

## List of Figures

Figure 1 - Excalibur manipulator system . . . . .	8
Figure 2 - Manipulator arm . . . . .	9
Figure 3 - Master arm . . . . .	10
Figure 4 - Excalibur's program structure . . . . .	26
Figure 5 - Main screen of the EXCAL . . . . .	27
Figure 6 - View of the workcell using the main camera . . . . .	31
Figure 7 - Virtual camera mounted on the gripper . . . . .	32
Figure 8 - Move arm environment . . . . .	34
Figure 9 - Inverse mode with 3D cursor . . . . .	39
Figure 10 - Gripper coordinate frame . . . . .	40
Figure 11 - Time usage on 486/33 . . . . .	51
Figure 12 - Time usage on 386/33 . . . . .	51
Figure 13 - Display time for the <i>EXCAL</i> program on 486/33C machine . . . . .	53
Figure 14 - Display time on 386/33C and 486/33C machines . . . . .	54
Figure 15 - Errors in position before calibration . . . . .	59
Figure 16 - Errors in joint positions after electrical calibration . . . . .	60
Figure 17 - Test user's workcell layout . . . . .	62
Figure 18 - Test user's generated task . . . . .	63
Figure 19 - Link parameters $\alpha$ , $\theta$ , d and a . . . . .	73
Figure 20 - Coordinate frames for the <i>Excalibur</i> . . . . .	76
Figure 21 - Frame coordinates . . . . .	85
Figure 22 - Perspective projection parameters . . . . .	89
Figure 23 - Hidden surface removal . . . . .	91
Figure 24 - Relative position of an object and the gripper . . . . .	95
Figure 25 - Dropped object . . . . .	96

# Chapter 1

## 1. Introduction

Simulation of a robot manipulator and its workcell provides a cost effective basis for research and development in robotics. These applications include manipulator design, task and work-cell design, human factors engineering, investigation of robot dynamics and control strategies. The use of an interactive graphical interface facilitates development of advanced robot programming languages, control systems and input devices for remote manipulators. From the early stages of the development of robotics, commanding of robot manipulators has been a demanding task. In this thesis "commanding" refers to the communication between the user and the robot that directs the system to perform the desired moves.

A few commercial programs are available for modelling industrial robots. These often use expensive graphics hardware and offer sophisticated visual models of the manipulator (for example, solid modelling, surface rendering, multiple light sources, etc.). Such systems are very expensive, often equivalent to the cost of a robot. This high price suggests the need for more affordable animation systems. This thesis describes the study and design of such a program for microcomputers. This program allows the user to simulate and control a robot manipulator in real-time.

The graphical simulation program can benefit two main areas in robotics, **off-line programming** and **on-line control**. Many simulation programs have been developed to deal with the off-line programming. Such products include McAuto's PLACE [1, 2], CATIA [3] and MnCELL [4] which use powerful and very expensive hardware. For moderately priced hardware such as Sun and Apollo and Silicon Graphics IRIS, the

ROPS(Robot Programming System) [3, 7] from Cincinnati Milacron, CimStation [5] of Silma Inc., GALOP/2D [7] of IBM and I-GRIP [5, 1] are available. For the PC microcomputers the PC-Grasp [7] which has evolved from Grasp (General Robot Arm Simulation Program) is available. These programs are mainly written to handle off-line programming. There are no reports on any work done for the real-time on-line control and commanding. Table 1 shows a more detailed list of the robot simulation programs commercially available.

Table 1 - Summary of some of the robot simulation programs.

Previous Works		
<b>PC-GRASP: Developed from GRASP, 1986 [7]</b>		
Hardware: IBM-PC and compatible	Application: Off-line programming and Simulation Cycle time evaluation	cost: not available
<b>Graphically Interactive Robot Simulation On PCs [8]</b>		
Hardware: IBM-PC and compatible Using accelerator card	Application: Simulation and off-line programming	Cost: not available
<b>AML/2 [5]</b>		
Hardware: IBM-PC and compatible	Application: Data not available	Cost: \$4,250
<b>CimStation [5]</b>		
Hardware: SGI-4D, SUN, APollo	Application: Basic CAD tools, Robot modelling, Path generation Kinematic and dynamic simulation	Cost: \$65,000
<b>I-GRIP [5, 1]</b>		
Hardware: Silicon Graphics 4D series	Application: Basic CAD tools, Robot modelling, Path generation Kinematic simulation	Cost: \$50,000

Previous Works		
<b>McAuto's PLACE [1, 2]</b>		
Hardware: VAX series, R-100 Graphics terminal	Application: Cell design, check reach Task Programming and Simulation Cycle time evaluation	Cost: not available
<b>CATIA [3]</b>		
Hardware: IBM 4300, IBM 5080 for graphics system, IBM 3279 for text display	Application: Robot/Workcell modelling Simulation and task programming Cycle time evaluation	Cost: not available
<b>MnCELL [4]</b>		
Hardware: VAX 11/780, EVANS & SUTHERLAND DS300, RAMTAK color terminal, TEKTRONIX vector display	Application: Robot modelling Simulation of several robots REAL-time animation Collision avoidance using interference detection	Cost: not available

In on-line commanding, the operator of the manipulator has to deal with many problems simultaneously. The nature of on-line commanding is therefore stressful and operations are subject to human error. In an ordinary command interface, all the actions taken by the operator are final and the real-time nature of the interface will affect the robot almost instantaneously. This means that the moves can be wrong or inefficient and potentially disastrous. The requirements on the system (which involves the manipulator, workcell and the objects in the cell) can be very demanding and expensive. The operator needs to be able to view the manipulator at will from different directions, and has to be able to manoeuvre the arm through obstacles with ease and confidence.

These problems are magnified for the remote manipulators, for which direct view is impossible and in many instances indirect view can be impractical and inefficient. Also, in this application, long delays exist between commanding the remote manipulator and the subsequent response from the manipulator.

These difficulties suggest the need of an easy and more dependable interface. A **graphical simulation** program can be viewed as a high level commanding interface. The interface can provide the operator with many advanced features to ease the command task. The obvious advantages of such an interface are its ability to view the robot manipulator from any view direction required. Moves can also be planned ahead of time and pre-viewed to enable the operator to visually check for collisions. Time delay, for example with space-based teleoperation, can significantly affect the operator's ability to carry out the teleoperation tasks. Simulation will allow the operator to plan the motion with a real-time simulation. The pre-planned motion can then be downloaded to the remote robot site.

This interface can also be used for *off-line programming*. The only additional requirements for the program is the ability to store the robot path, and replay the stored path at a later time.

In designing a real-time simulation program three main problem areas are identified. The first problem is the *dynamic control* of the robot. This problem is raised by the multi-link nature of the manipulator. Such a structure has highly non-linear dynamic responses, and in most cases an exact mathematical modelling of the system is impractical. The second problem area, *motion coordination and trajectory planning*, is also a consequence of the multi-body nature of the robot mechanism. The third problem is the *interaction of the robot with the world*. This area covers the actions taken by the operator in commanding and programming the robot manipulator. This includes the methods used to avoid collision, grasping consideration, force of the gripper and the sensory feedbacks from the manipulator.

This study is mainly concerned with the third problem, *interaction of the robot with the world*. However the other two problems can have major bearing on the overall performance of the program. A good simulation program has to be real-time, and for off-

line programming, it has to be able to simulate the dynamic behavior of the robot in close approximation. It is obvious that the simulation program needs to be real time for on-line commanding. In off-line programming the program has to reflect the changes in the moves due to changes in the inertia of the arm. The robot controllers are designed to achieve the requested moves regardless of the payload on the robot. However, it is possible that the arm will have increased overshoots for heavy loads. This can pose a safety problem if the simulation does not reflect the altered path. This problem is more significant for the older robots where the linkage system is not very rigid.

The uncertainty in accuracy can introduce large errors for accurate jobs. Most robot manufacturers quote the repeatability and not the accuracy of the robot. Repeatability refers to the ability of the arm to repeat a taught position within some small tolerance. The repeatability of the robot is mainly dependent on the accuracy of the joint encoders and the servo mechanism for the actuators. Therefore if a position is taught on-line, the manipulator can repeat the position within the accuracy of the encoders. This repeatability is usually adequate if the robot is programmed on-line.

With off-line programming, however, it is necessary to know how accurate a commanded position can be achieved. The accuracy of the robot depends on many factors. First a calculation based on the kinematic model must be performed to find the required joint angles. This calculation and the kinematic model usually are based on the assumption that the robot linkages are perfectly rigid. However, this assumption is not true and the robotic arms are designed with substantial flexibility in their joints. Neglecting the effects of the flexibility of the robot can introduce unacceptable errors in the robot positioning. The following factors increase the effects of the flexibility [9]:

- increased payload
- longer links
- increased mass of the links
- increased operational speed of the robot



However, some robot manufacturers are moving toward producing more accurate robots with rigid linkages and accurate drive mechanisms. This will eliminate many of those concerns. By incorporating direct drive electrical motors in the joint, the gear and its related problems are eliminated. Control characteristics can be made programmable by putting the computer inside the motor. By changing **EPROMs**, and hence the software they contain, one can alter the velocity and torque profile of the drive mechanisms[10].

This study consists of the following components:

- ◆ aspects of human factors engineering
- ◆ development of a three dimensional graphical animation program
- ◆ examination of different user input devices as means of interfacing the robot
- ◆ real-time serial communication with the remote manipulator.
- ◆ development of a test procedure
- ◆ evaluation of the developed interface.

The computer program *EXCAL* is written for an "EXCALIBUR" robot. The support material includes a user's manual [11] for the software and the technical manuals [12] for the operation of the Excalibur arm.

The program was tested from two points of view. First the graphical interface was studied for its apparent visible aspects such as computer screen display features, view capabilities and the input devices. Then the commanding tools provided by the program were evaluated for their effectiveness and usefulness. This part of the study, known as "Human factors engineering", is where the programmers can have direct feedback from the potential users. The user's comments were then used as a guide for further improvement of the system.

## 2. Components of the System

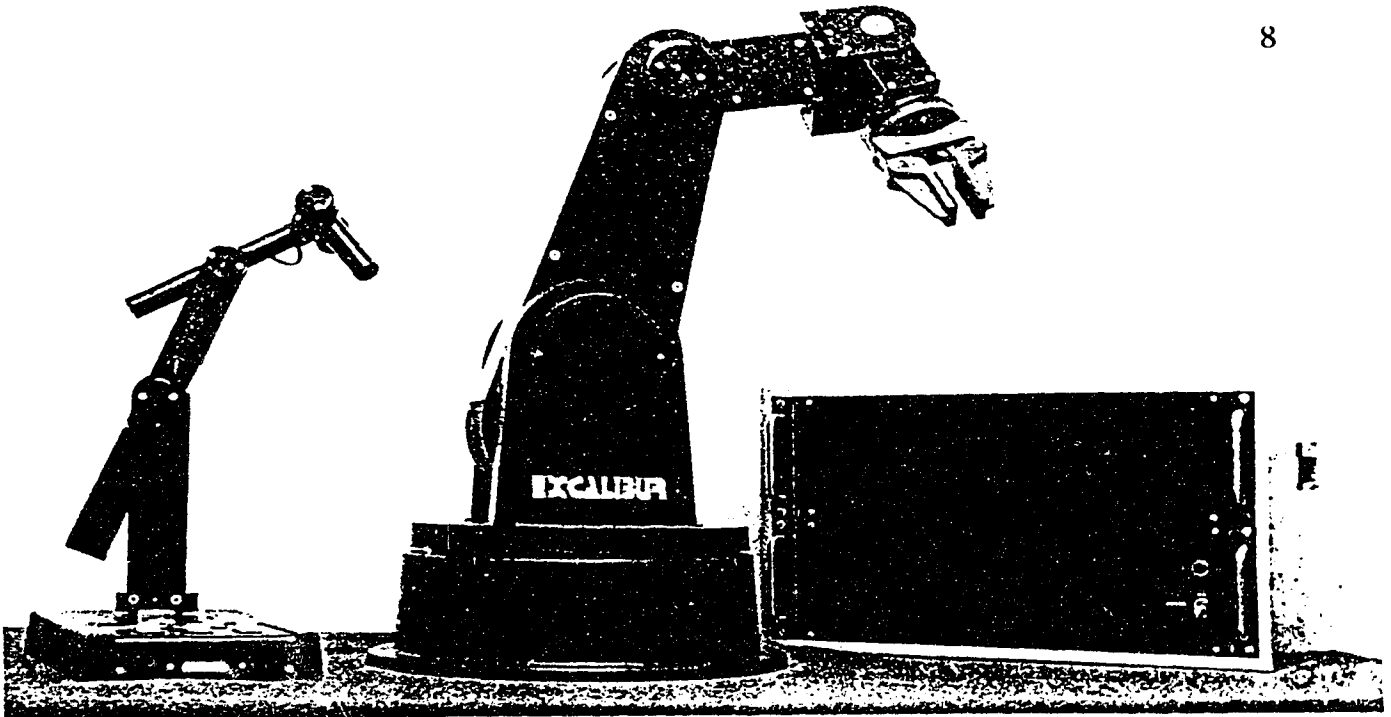
### System Outline

In this section the main components of the system are introduced. These are:

- ◆ the Excalibur manipulator arm (slave)
- ◆ the Excalibur controller
- ◆ the small scale master arm
- ◆ the Gateway 2000 486/33C host computer

Figure 1 shows the Excalibur system. The multi-jointed arm is the **manipulator**, the part of the robotic system that does the actual work. The smaller jointed arm is the **master arm**. This is a scaled model of the manipulator and can be used to control its motions in some types of operations.

The electronics of the robot are contained in the cabinet referred to as the **controller**. An outline of the function of each of these components follows.



**Figure 1** - Excalibur manipulator system

### **Manipulator**

The manipulator arm shown in Figure 2 is the working end of the Excalibur. The manipulator (also known as manipulator arm or "slave arm") is a six-link mechanical arm with a hand or tool, known as the **end effector**. The Excalibur's six degrees of freedom, allow the end effector to reach any location in the robot's work envelope with any orientation. The Excalibur is equipped with a parallel-jaw gripper. The manipulator has a reach of 76.2 cm and can carry up to 3.5 Kg payload at a maximum linear speed of 0.4 meters per second.

### **Master Arm**

In essence, the master arm is a small-scale, unpowered model of the manipulator. It is equipped with potentiometers to sense the joint angles. Motions generated by the user at the master are recreated by the manipulator (slave). The controller attempts to drive the manipulator's joint angles to agree with those of the master arm. This is known as

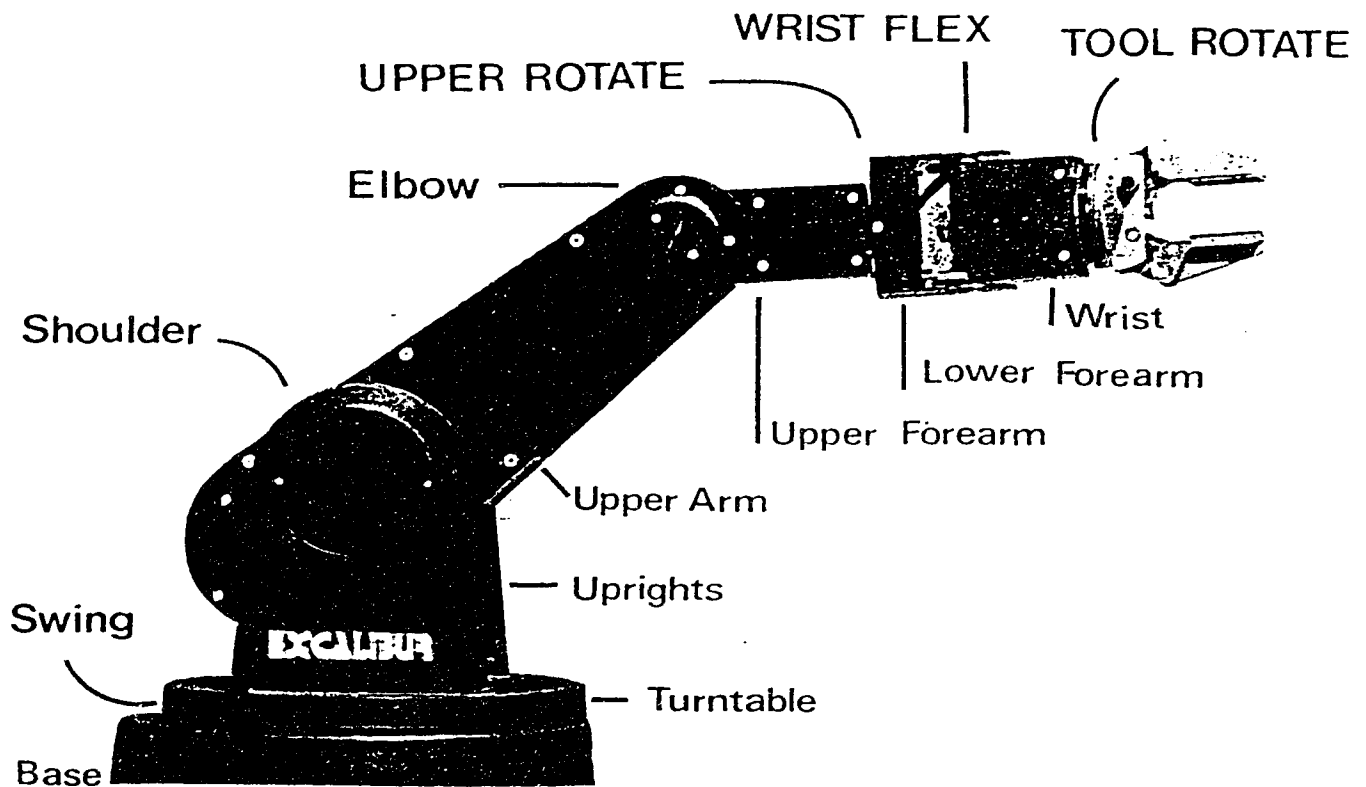


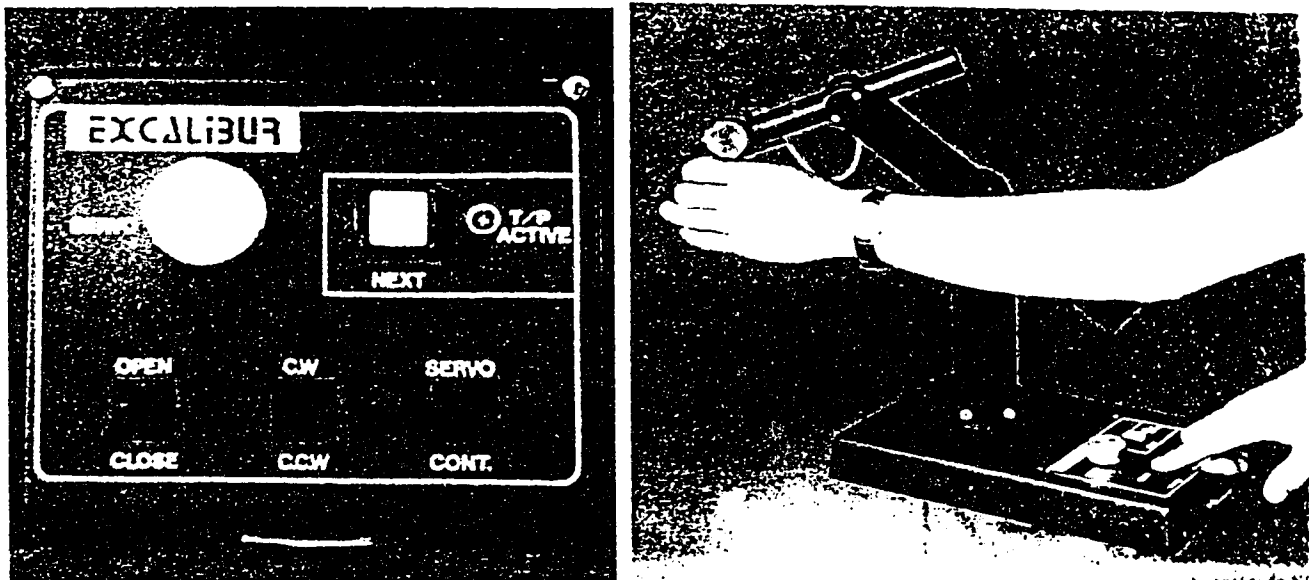
Figure 2 - Manipulator arm

**spatially correspondent system.** A number of switches and controls on the base of the master arm allow the user to control various functions of the Excalibur. These controls include wrist roll, gripper open/close, storing of current position in the controller memory, and specifying point to point or continuous path motion (Figure 3).

### Controller

Virtually all the electronics, a complete microcomputer system, and the power supply are enclosed in this unit. The front of the controller is shown in Figure 1. This unit processes commands from the host computer or master arm.

A series of analog servo controls provide continuous voltage commands which represent **error signals** between the actual and desired manipulator joint angles. These signals are in turn fed to the motor controllers which amplify and condition the signals used to actually drive the manipulator joints to the desired positions.



**Figure 3 - Master arm**

In addition, the controller can communicate with an external computer through a serial interface. The controller is a microcomputer, consisting of a Motorola 6809 microprocessor with 1 MHz clock, RAM and EPROM memory and various input/output peripheral channels.

Cables from the manipulator, the master arm and the host computer carry all the required signals to and from the controller. These signals include feedback values for joint angles (master and manipulator), switch closures and messages to and from the host computer. The controller can independently set the operating mode for the robot. Four operating modes are available:

- Host mode: In this mode the host computer can communicate with the controller.
- Manual mode: In this mode the master arm is used to control the slaved arm.
- Teach mode: A series of robot movements generated by the master arm are stored in the memory of the controller and can be played back at later stages.
- Playback mode: The taught path can be played back.

All these modes can be accessed through the host computer. It is also possible to disconnect the analog servo controls from the motor controls. In this case the host computer can run alternative servo control programs.

### **Host computer**

The heart of the graphical simulation interface is the host computer. The DOS-based computer communicates with the controller through a serial communication interface **RS-232c** in ASCII format. The computer used is a GATEWAY 2000, 486/33C computer.

The computer is equipped with 640 KByte standard RAM memory and eight MByte extended memory. The VGA/EGA graphics driver is available with one MByte of graphics memory.

### **Programming Language**

The simulation program is written in the C language using Microsoft's Quick C version 2.5.

Serial communication is done through the IBM based **8250 UART** integrated circuit, which is the standard board for serial ports. The software used to drive this board is the "**C ASYNCH MANAGER**" [13], by Blaise Computing INC. This package provides an interrupt driven communication interface and its functions are callable from the **QUICK C** programming environment.

## **Chapter 2**

### **Principles of human computer interaction**

#### **2.1 Introduction**

This chapter discusses the basis of human factors engineering relevant to the interface design. The principles and guidelines are based on the previous experience of the industry [14, 15]. Humans have excellent visualization and reasoning capabilities and can adapt to changes and unexpected conditions effectively. On the other hand machines are much more effective at routine, high volume and repetitive tasks. An effective interface can utilize both human and computer resources with a balanced and logical division of the task.

#### **2.2 Basic principles**

Design and performance of an interface can be significantly improved using simple guidelines. Based on human psychology, the following guidelines can be used to decrease the learning time and make the interface natural.

1. Using familiar patterns improves the learning process and significantly lowers the human memory requirements. The interface can use the previously developed ideas and benefit from the familiarity of the user with the concept. However, this principle should be used cautiously since overdoing this may limit the degree of innovation in new systems.

2. The steps required to perform a task should be organized. The predictability and presence of logical order can improve the efficiency of the whole operation.
3. The job frame should be devised to relieve the user from repetitive tasks. The computer can handle many repetitive tasks independently but the interface must maintain full user control at all times. Considering the inherent limitations on human's short-term memory, only relevant information should be presented through the interface. A screen full of unrelated information will only overload the user's memory. Excessive error messages and numerical data can confuse the users and affect their reasoning ability.
4. One important factor in the design of an interface is simplicity. The task should be achieved with the minimum number of steps and input from the user.

### **2.3 Interface Analysis**

Information on the nature of the job and the type of users involved can be used to establish the basis for the design of the interface. Using this knowledge, the job can be systematically divided between the human and the computer.

#### **User analysis**

The type of users and their ability has a major influence on the design of the interface. The interface should match itself to the ability of the most common group of the users. This fact determines the sophistication of the interface and its support system.

The users are evaluated for their knowledge of the system and the skill level they are expected to achieve. The skill is affected by the frequency of use, familiarity with computers and knowledge of the task to be performed.



## **Task analysis**

To divide the job effectively between the human and the computer a thorough analysis of the nature of the job is a necessity. Tasks can be measured in terms of their complexity, attention to details, monitoring requirements and consequences of their failure.

Simple and repetitive tasks can be handled by computers with minimum supervision. The more complex jobs can be reasonably divided between human and computer. The interface should effectively use the human reasoning capabilities without overloading the human side of the interface.

## **2.4 Screen and display requirements**

### **Menu**

Menus provide a useful way to make computers more user friendly and accessible by reducing the demand on the user's memory. Ideally, menus should be designed to accommodate different levels of users. However, overly simplified and menu dependent systems can be frustrating for experienced users who prefer short cut keys. Response to menus should be reasonably fast. Graphical symbols can be used instead of words for ease of recognition.

Important factors in menu design are:

- Menu display style
- Activating/Selecting a menu
- Menu setup and hierarchy

Menus should be easily accessible. Too many sub-menus can make the process of finding and selecting a menu item frustrating. The menu display should be in harmony with the rest of the screen.

## **Color**

Color can be effectively used for many purposes if implemented carefully and conservatively. Color can help the user locate classes of information by highlighting data, categorizing information, establishing relationship between separated fields and clarification of the overall view. The effective use of color requires consideration of several factors [15]:

- How the selected color can improve the readability of the display.
- The effect of the color coding on the performance of the user.
- The impact of color codes on the capabilities of the computer and compatibility with systems not supporting the colors used.

## **Displayed information**

There should be adequate information to let the user identify the current work environment. Also to reduce the memory requirements on the user, relevant help can be displayed to guide the user. The status of the program and peripheral equipment should be clearly displayed.

## **Help**

Help text should be readily available to the user at any time. The program should be designed to provide the user with the help relevant to the task he/she performs at the time help is requested (ie: context sensitive). The help text should be clear and concise.

## **Graphics**

The use of graphical and pictorial representation has become an integral part of the user-friendly interfaces. The command language and menus use graphical symbols extensively. Graphical simulation has opened the doors to a virtual world where the testing and development of almost anything is made possible. The effectiveness of graphics for simulation can be measured in terms of visibility, clarity, depth information and relations between distinct objects.

## **Status Information**

The program must always provide an indication of the system status. After a request, users need acknowledgment. The program should let the user know its status especially if the process time for the request is long.

## **Command language**

The command language, traditionally as typed-in words, lets the user make a request. The modern approach often presents a list of available commands on the display. Commands can be selected using a pointer, or by entering the command's short cut key from the keyboard.

## **System response time**

"System response time" [15] and "user response time" [15] (think time) are two important factors in the design of an interface. These factors depend on the complexity of the program logic and on the nature of the task and responsibility involved in the decisions made. A fair balance between these two factors is vital for the efficiency of the overall interface.

System response time depends on the structure of the program and also the computer hardware used. The long delays can adversely affect the performance of the user. A fast system response time is always desirable, however, an acceptable system response time can depend on the "level of closure" [15]. Usually the delays after the user has completed a major unit of work are quite acceptable. Whereas slow response in minor steps in a major unit of work can cause the user to forget the next planned step.

The user response time also known as think time, is the time required by the user to respond to the updated information. There is a close relationship between the user response time and the system response time. A well established balance between these two factors is vital to the performance of the program. The system response time affects the user response time in the following ways:

- Long delays can disrupt the flow of thought and the short term memory for the next planned step.
- When long delays are expected the users can spend more time on organizing and checking their moves and thus avoiding errors.
- Conversely, fast system response time allows the user to respond quickly, not afraid of making errors, because the errors can be recovered quickly.

### **Command and Input Devices**

Traditionally, human-computer interfaces have relied heavily on the keyboard as the control device by which the user communicates with the computer. In commanding the robotic arm a more natural command device is preferable. The master arm (with master/slave relationship) can be used as a command/input device. A mouse or joystick is an alternative input device that can be considered to command a move for the robotic arm. These two devices can be used to point to icons representing different commands. Basically these devices are suitable to control two and three degrees of freedom, where in robotics input devices require to control six degrees of freedom. This makes them

unnatural for direct manipulation of the position of the arm. Voice command entry can be used to enter simple discrete commands keeping the user's hands free. Voice recognition can be slow and frequently misunderstood by the computer.

In robot control, the user is concerned with 3-D object manipulation. A hand gesture can be effective for specifying position and orientation. The DataGlove [16] is a new device (1987) that can be used for hand tracking. The DataGlove is a device that the user wears. Analog sensors measure the bending of fingers and the position and orientation of the hand in 3-D space is tracked by a Polhemus sensor.

### **Error messages and On-line assistance**

Error handling is a critical feature and requires careful consideration. The main elements of error handling are error correction, error message dialogue, and on-line guidance to help users understand the system and thus avoid errors. Adequate error handling unfortunately, requires a considerable amount of programming code which must be written in anticipation of a broad range of errors which can occur on either side of the interface.

### **Design concepts for the program**

Based on the subjects discussed in this chapter, a preliminary design idea for the program is deduced. The interface is for use of robot operators and in general for people with knowledge of computers. Therefore this interface is designed for skilled users. This means that the program should have efficient and short-cut paths to accomplish tasks. The user is assumed to be required to use the program as part of a routine job.

The program has a fair level of sophistication and will require a reasonable learning time. The nature of robot operation does not require frequent innovation in the operation

of the routine tasks and commanding of the arm, therefore the program is not designed to create an advanced command language through the software.

To accomplish the needs of the user, the program uses structured menus alongside the short-cut keys. Context sensitive on-line help is available. The program is structured with the needs of the user in mind and each environment accomplishes a specific task. This makes the purpose of the environment clear, however, the flexibility of the program is compromised.

This program would be considered rather complex. Nonetheless, a large portion of the overall job is given to the user and the user is expected to reason well and have a sound judgment. This is inevitable in the light of the current state of the Artificial Intelligence, AI. In order for the computer to have a larger portion of the job, an efficient AI system is necessary. For example, the user's work would be substantially simpler if the computer could automatically detect collision and lay a new path to avoid collision. Also the role of AI is obvious in the design of high level commands. For example a command to pick an object can be simplified to the level where the user issues the command with the object number as its single argument and the move is carried out by the interface. Proper execution of this command would require the arm to choose an appropriate path and orientation. The grasp orientation can be limited by the approach constraints or may depend on the next move. In either case large amount of decision making is required. The trend in industry is toward reducing the human side of the job handling, so the human operator can concentrate on more important issues. Also ideally the computer should handle the errors, and take care of the task failures. More complex errors will require the operator to intervene.

The next chapter will discuss the design of *EXCAL* in the context of human factors engineering presented here. This will be followed by a discussion of the success of the program design.

## Chapter 3

### Program Design

#### 3.1 Introduction

In this chapter, the basic layout and design of the *EXCAL* program is reviewed. The ideas behind the design and inclusion of the different parts of the program are discussed. The additional parts that could be included, but are not currently implemented are also addressed.

The chapter covers the view cameras, mainly the second camera mounted on the gripper, and the command tools included in the program. The benefits of each tool and its shortcomings are addressed and recommendations for further improvements are made.

The objective of this work was the development of a graphical interface for robots in a microcomputer environment. These computers are generally more affordable, but they have their inherent limitations. The main PCs under consideration are IBM compatible DOS based machines. These computers have no special hardware (eg: graphics accelerator) to handle graphics. However, the latest generation of this series is very fast and efficient at floating point operations.

Since the software is to be **real-time**, the goals are set keeping in mind that these machines have limited graphical capabilities. That is, if more sophisticated graphical display is planned, then the operational and animation speed would probably degenerate. Therefore, complexity of the graphics is limited by the overall operational speed of the combined hardware and software system.

Another factor that should be kept in mind is that this study focuses on the specific area of the graphical representation of the arm and its workcell. Robotics is a rapidly growing and changing field. There are still enormous problems left to be dealt with in almost any part of robotics. Thus the program ignored many parts that can be important in the overall performance of a robot programming system. As an example of such a case, this software does not have a collision protection routine<sup>\*</sup> even though such a routine is vital to safe operation of the robot, and can be of great assistance to the user. Collision protection (i.e. obstacle avoidance) methods are numerous and the subject of extensive study. However, current algorithms are neither fast enough nor general enough to handle complicated real world situation. For example, a simple 2-D obstacle avoidance algorithm was presented in a graduate thesis by Eugene Cao [17]. This reference also contains a discussion of other obstacle avoidance algorithms. Without implementing a completely automatic obstacle avoidance routine, it can be argued that the graphical representation presented here can assist the operator in identifying a potential collision.

<sup>\*</sup> A simple routine was implemented to prevent the gripper from striking either the bench top or its own base.



### **3.2 Interface considerations**

This section has close ties to **human factors engineering**. The software design should be such that the user is not overloaded by information, or is facing a cluttered display or has to deal with clumsy input devices in order to interact with the robot.

There are two major parts to be considered. The first consideration is how the screen display should be designed. This screen is the immediate interface panel between the user and the machine. Some concerns would be: menu system, amount of information displayed to the user at any time, amount of information available to user as help, color and command tools.

The second consideration involves the methods available to the user to interact with the robot. To command the robot and communicate with the machine, the user has to use some input devices. The standard input device is the keyboard. Therefore, the keyboard layout and availability of shortcut keys is very important. The program can use the master arm as an alternative input device, so that the best control devices can be evaluated, and if possible other input means can be explored.

### **3.3 Graphical display**

The most significant element of a graphical simulation interface is the display of the manipulator and the amount of information that this display provides. This display is the immediate feedback from the robot and workcell.

Often it is not possible to view the robot directly. Even if the robot is in direct view, the operator has only a limited point of view. To overcome this, a video camera can assist the operator, but a video camera is also bound by its movements and limited view direction. Therefore a powerful synthetic viewing capability is a necessity to command a robot manipulator.

The display should have powerful features to produce adequate visual feedback for the user. The display should clarify the position and orientation of the robot manipulator. It is favourable to have a detailed simulation of the arm, that is a graphical display with solid surfaces, good lighting and proper shading. However considering the inherent limitations of the host computer, it was decided to limit the arm display to a wire frame representation.

The data-base of the program stores the coordinates of each rectangular prism, used for constructing the arm and the objects in the workcell, in two dimensional arrays. A second 2-D array holds the connectivity values between surfaces and the vertices. For more detailed description see "Data-Base design" in Appendix B. A simple hidden surface removal technique is used to eliminate the hidden surfaces from the displayed image. Although this algorithm removes the hidden surfaces there are still some undesired hidden lines left. The hidden lines can sometimes cause confusion by cluttering the picture. This fact becomes obvious when there are many objects present in the workcell, or at certain views that create many undesirable partially hidden lines. Hidden line removal techniques are computationally intensive [18]. If these were used, the operational speed of the program would severely degenerate.

The best method for hidden surface/line removal for an application depends on many factors. Proper consideration of the requirements on the displayed image can help choosing the appropriate method. If the surfaces in a scene are spread out in the z direction a depth sorting method can be the best. If the surfaces are well separated in the x-y plane, a scan-line or area sub-division method might be the best. Factors such as the above mentioned ones can help in selecting the suitable method for the application. When wire frame images are used, hidden line removal is considered. A direct method to eliminate hidden lines involves comparing each line in the scene to each surface in the scene. This process is similar to clipping the lines against the window or viewport of the screen, except that in this case the part of the line inside the window is to be eliminated. If a hidden line method is to be used, there are some techniques that can improve the

efficiency of the algorithm. However, hidden line removal methods are all computationally intensive and require increasingly more time as the number of the objects increase. For complex scenes it will be preferable to use a scan-line method or if there is available memory on the machine, a z-buffer method can be used. These methods, unless implemented on the hardware, severely degenerate the performance of the program for animated scenes. Therefore it was decided not to use the hidden line removal algorithms.

The operational speed of the program is affected cumulatively by different factors. When a command is issued from the input device the new position is calculated. The Inverse kinematics solution provides the joint angles that can move the arm to this configuration. The animation routine then increments the joint angles and for each increment a picture frame is generated. The previous image is then deleted selectively by redrawing the image lines using the background color. Finally the new image is drawn. This procedure continues until the new position is achieved. Thus the operational speed of the program consists of the time required for calculating the new position and orientation, the time required for the animation routine to find the proper increment values and the time required for redrawing an image.

The process of generating a frame starts with the related transformation/rotation calculations for the arm links, world objects and grid lines representing the workcell. These transformations position the entities at their respective position in the world. Each transformed entity is further projected onto the two dimensional view plane by perspective projection. The hidden surface removal technique is applied to each entity. The previous image on the screen is deleted by redrawing the previous frame using the background color and then the new image is drawn on the screen. The frame generation time can be divided to two distinct components: the calculation time and the time required to transfer this image information onto the graphics screen. The speed by which the video information is refreshed depends on the hardware. Therefore, to speed up

frame generation the calculation time should be minimized. This fact prohibits the use of sophisticated methods for hidden line removal.

The **framing rate** is the number of the displayed images per second in the animation sequence. This rate depends mainly on the time required to generate a picture frame. If the picture generation is fast then the framing rate will be high and a smooth animation is achieved.

The animation is directly affected by the time that frame generation requires. To have real-time animation a high framing rate is necessary (more than 10 frames per second). Smoothness of the animation can be sacrificed in favor of achieving real-time performance. This is done using large increments in the animation routine. In this case the animation looks more abrupt and stepped. The increments can be adjusted through the software to control the apparent speed of the animated arm.

The best achievable 3-D representation on a two dimensional display screen is through perspective transformation. This transformation can create the illusion of depth in the displayed image. To further enhance the depth recognition, different colors are used to distinguish the arm and the workcell.

Stereo image processing can enhance depth recognition. A number of (expensive) systems are available which can produce stereo image pairs. This simulation technique in its simplest form generates the image from two different view points. Superimposing these images that are drawn in different colors, and using special polarized glasses can create 3-D images. However, this technique would decrease the framing rate by a factor of two.

### 3.4 Program Structure

Figure 4 shows the basic structure of the program. It would be ideal for the program to have access to a library of different robots, so that user can specify the type of the robot under study. The current version of the program is designed for *Excalibur* robot and relevant data and transformations are integrated into the program.

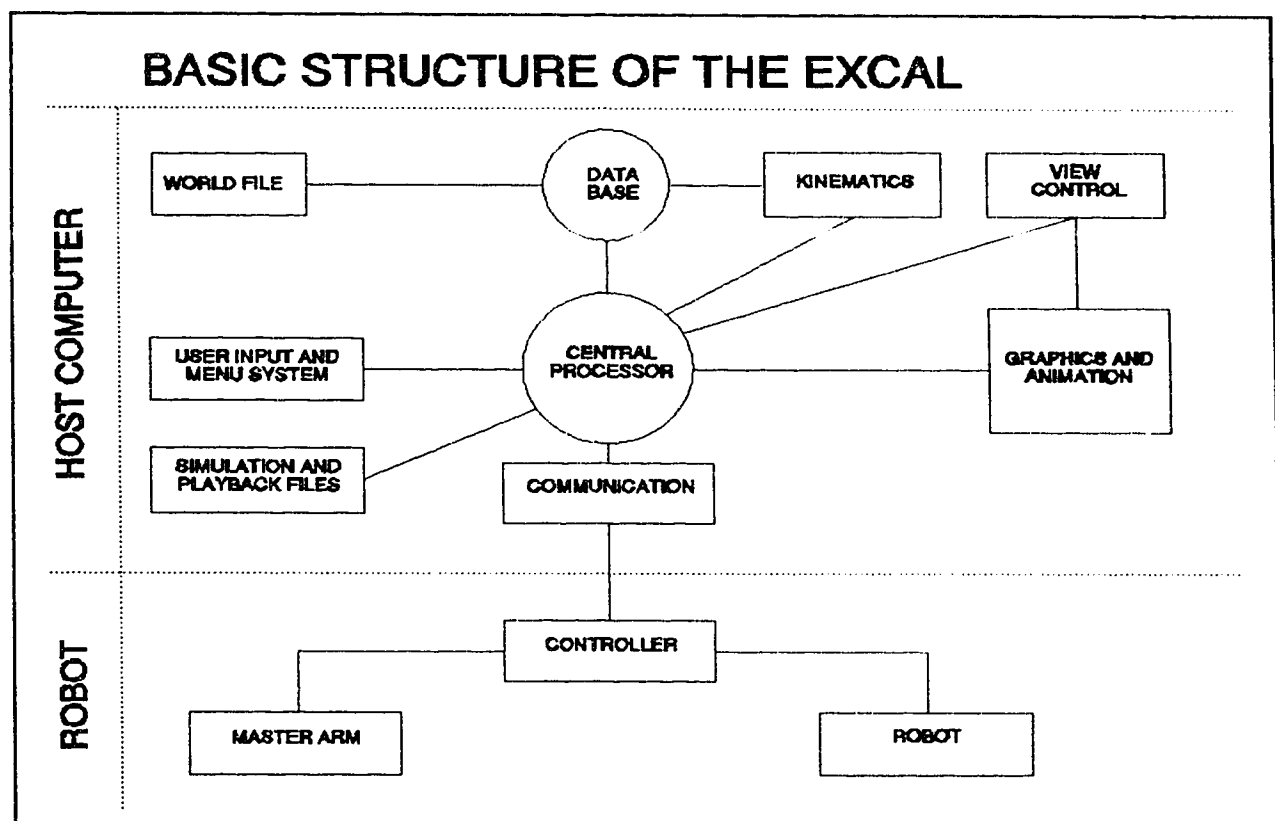


Figure 4 - Excalibur's program structure

Data describing the world objects (ie: the objects present in the workcell) is read from disk files. The user can add or replace an existing world object. There is currently no sensory feedback from world objects into the program, which assumes that the real world is arranged as the data files indicate. The user has to make sure that the world objects are the way they are defined in world data files (or vice versa). It would be possible to

use an advanced vision/sensory system and update the data structures of the program, but this requires a vision system and probably another computer to calculate the position of the world objects.

Once the data structures and global variables are set, the program displays the arm and main menu. The *Excalibur* is displayed as a wire frame image in the default home position (Figure 5). If the arm is turned on then the program reads the arm position and the displayed arm moves to that configuration. From this screen the user can select the sub-functions available and perform the desired tasks.

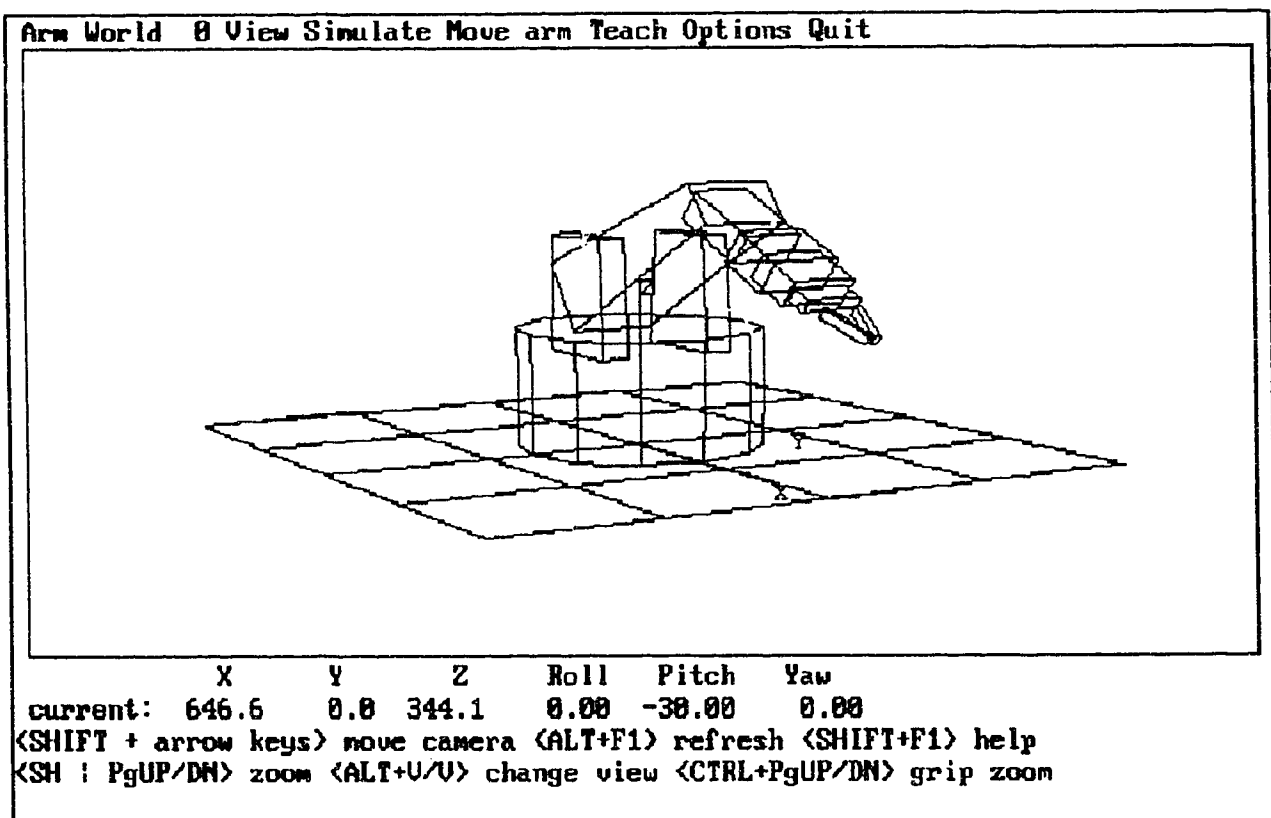


Figure 5 - Main screen of the EXCAL

## **Communication with the arm**

Through the communication menu ("Arm"), the user establishes or discontinues communication with the robot controller via the two way asynchronous serial data line.

To keep track of the events happening to the robotic arm, the controller is designed to send messages whenever something happens to the robot. These messages can refer to errors in communication or when the robot is unable to achieve the requested move. Since the controller can send messages at any time, the host computer must handle the communication regardless of the execution state of the main program. Therefore an interrupt driven communication module is used. In this method, whenever data is ready to be transmitted or a message is sent from the controller, an interrupt signal halts the execution of the main program and the interrupt handler routine takes care of the transmission. In the case of an incoming message, the transmission is read and stored in an internal buffer. For an outgoing message to the controller, the message is read from the internal memory buffer and sent to the controller. This procedure allows the program to continue its operation until there is a need to communicate. The communication routines of the program (such as READCH(), WRITECH()) read and write to the internal memory buffers, allocated for serial data storage, and not directly to the communication ports.

Other controls provided to the user are setting the communication port number and the operating mode of the controller (ie: manual, host modes). Also a global variable determines if the actions taken by the program should affect the robot or not.

It is valuable to isolate the program and the robot manipulator at will. This enables the user to simulate the desired moves and eventually download the move command to the robot manipulator when the user is satisfied.

### **World object data file**

The configuration of world objects is defined in data files. These files contain information on the size and shape of the objects and their positions in the workcell. The objects can be assigned different colors. The program assigns a number to each object as it is read into the program.

There are two types of objects that the program recognizes: Fixed and moveable. The fixed objects are identified by a negative color value in the data file, and they become a part of the robot workcell. The objective of having fixed objects is to allow the user to introduce existing obstacles to the simulated workcell. Also these fixed objects can be used as target locations for Off-Line task programming.

### **View considerations**

It is important for the operator to have a clear and informative view of the robot. Even if the robot is in direct view of the operator, it is unlikely that the operator can have a proper view of the robot at all the times and for every position of the robot in the workcell. For example, there will be instances when the view of the gripper is blocked or when during the approach of the gripper to an object, view direction and approach directions are normal to each other. In these cases the visual information is not enough for the operator to easily guide the arm through the task and the operator must physically move to the robot site or in more advanced cases a mobile camera can be guided to the site. It is possible to use video cameras and TV screens to provide view for the command level. In either case it is not very convenient and the speed of the whole operation can be impaired.

Achieving proper visual feedback becomes more difficult for remote manipulators. For these the only visual information is through the video cameras. This may require many cameras to be present on the site or an efficient mechanism to be implemented to move



the camera to the proper position and orientation. However, there are many instances where the video cameras will be of limited or no use. A clear video display requires proper lighting and mobility capabilities. There might be cases where there is no clear and transparent media for the camera to function, such as under water or in dusty environments. Video information also requires wide bandwidth for transmission, which can be a problem on the whole system. Another problem is with space applications where there is considerable time delay between transmissions. In real time operations, this factor can render the whole view system practically useless.

Graphical simulation does not have any of these physical limitations for its cameras, and such an interface can view the workcell from any view point and any direction required. Since the computer database holds all the information about the manipulator arm, workcell and the objects in the cell, it is easy to generate simulated or synthetic views of the robot and workcell from any direction and view point.

One advantage of such a program is that as many virtual cameras as necessary can be implemented and thus the most advantageous view is generated when required. This can be of great help to the operator in his/her efforts in guiding the end effector. For example, the proper view will assist the operator to avoid a collision with the objects in the cell. One goal of the present work is to investigate and find out the most favourable view points and cameras for most of the commonly performed tasks.

Two virtual cameras were implemented as a standard part of the program. These are the **main camera** and the **gripper camera**. The gripper camera views the world (i.e. robot/workcell) in the approach direction of the gripper. This camera facilitates the guiding of the gripper during approach to objects. These cameras have zooming capabilities so that the user can zoom in and out as required. The perspective transformation is used to generate a two dimensional representation of the 3-D objects. The parameters of the perspective transformation can be used to set the zoom factor and the degree of perceived depth perspective.

## View cameras

As already mentioned there are two virtual cameras available to the program. The main (default) camera, provides the user with a full view of the workcell (Figure 6). The user can view the robot and workcell from any angle and magnification.

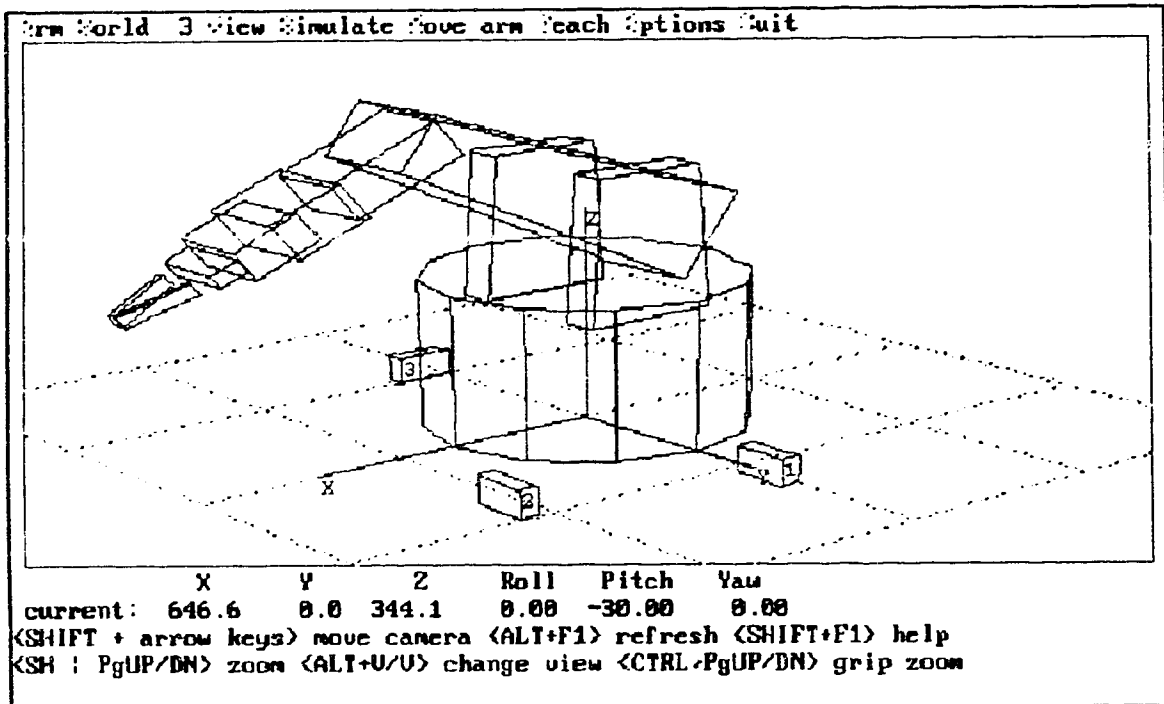


Figure 6 - View of the workcell using the main camera

The virtual camera replaces a real video camera with the difference that the virtual camera can move to any point and there are no physical barriers. This has both benefits and limitations. As it is observed in the performance of the program, certain positions of the camera can confuse the user. It takes a small time for an average user to reestablish a proper understanding of the position of the robot. An example is when the camera is moved under the surface of the workcell. If the program was designed to remove all the hidden lines this cases would pose no problem in creating a proper mental picture of the position of the robot. However, the current limitations on the program are such that hidden line removal techniques cannot be utilized. Still, the main camera provides very useful display and the ease of control and flexible placement distinguishes

this virtual camera from any real video camera. The main camera can lock on the gripper and follow it along its path. This mode offers continuous focused view of the gripper which is difficult to achieve with a real camera.

However, one should not deny the value of the presence of real cameras. It is always a possibility that something will go wrong during normal operation of the robot. In this case, the amount of information returned from the robot (i.e. joint angles) is not adequate to reestablish the correct situation of the cell such as world object positions. For example the position of the world objects is assumed in the data file. The presence of a video camera can assist the operator to judge the real situation in the workcell and take necessary actions. This is specially important for the remote manipulators.

The second virtual camera is "mounted" on the gripper and views the workcell in the approach direction of the gripper. This camera provides the user with valuable information on the relative position of the gripper with the objects (Lower right corner in Figure 7 shows the two fingers centred on the object number 1).

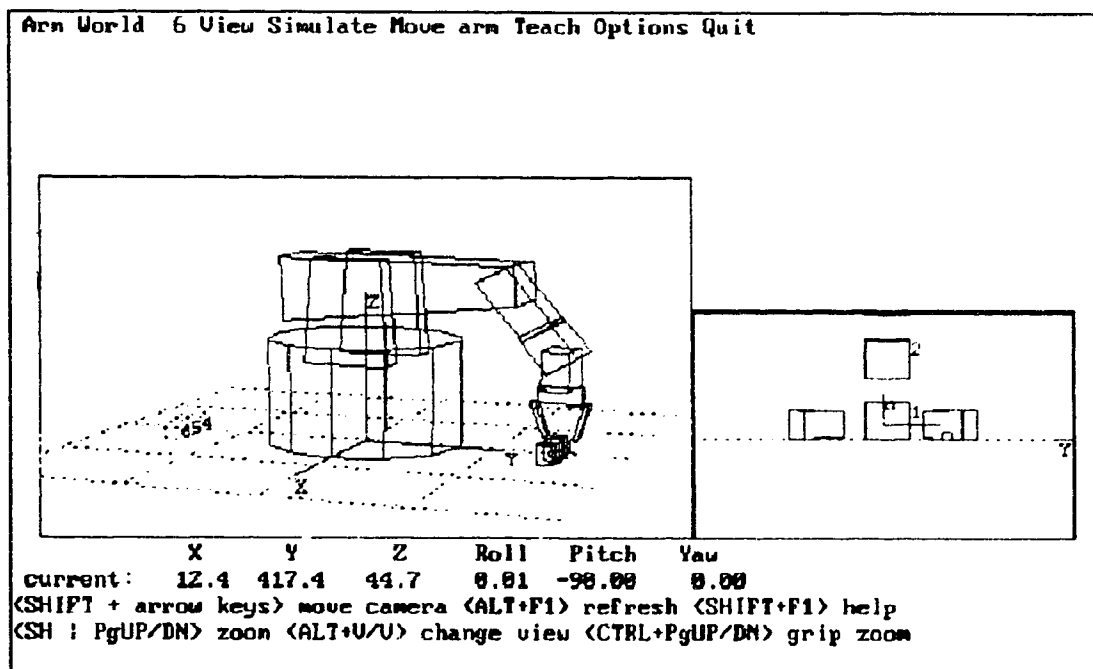


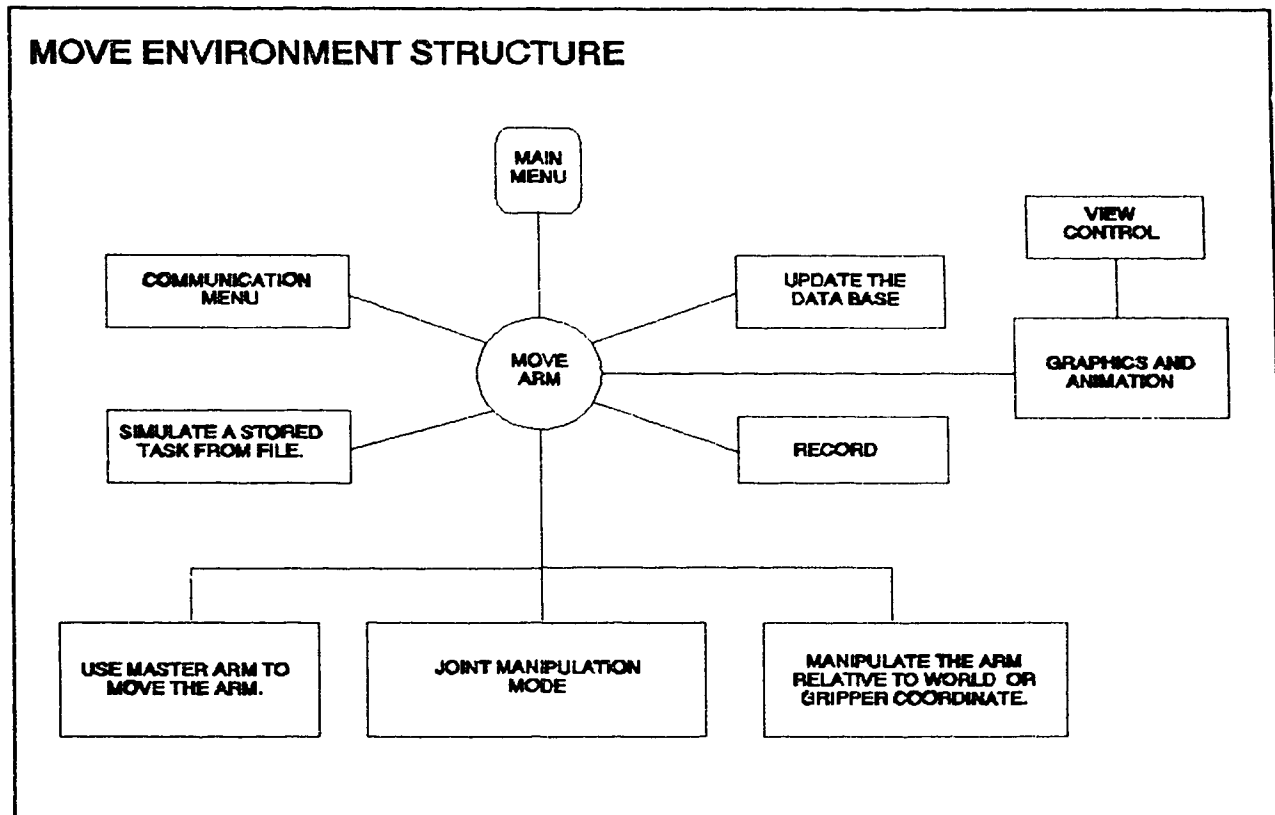
Figure 7 - Virtual camera mounted on the gripper

This camera is extremely useful in positioning the gripper during final approach to an object. The automatic updating of the two views provides the user with two view screens, and generates information on the relative position and orientation of the gripper from the object. The two screens are not updated simultaneously (due to computational constraints). The program updates the second active screen every few frames of animation. The primary active camera can be toggled between the two camera so that the view control commands can affect the desired camera.

Proper viewing of the robot/workcell and easy control of the view are of prime importance to the graphical simulation interface. Therefore the **view control** is offered as a menu, and also as keyboard commands. Since it is very important to have easy access to different views the shortcut keys are available to the user at any time. These keys control the view angle, view camera and zoom, during all programming and animation sequences.

## "Move arm" mode

This environment is the main working environment of the program. Figure 8 is a schematic of this environment. The program provides the user with elementary tools to command each individual joint of the manipulator. Using the inverse kinematic routines, the position of the end effector is manipulated as a whole. The inverse kinematic commanding tools are more sophisticated in terms of the specific type of move they can generate.



**Figure 8 - Move arm environment**

The moves can be recorded into disk files or played back and simulated. These features enable the user to generate off-line programs for the robot. Also by storing the path in a file the operator can preplan a task for on-line commanding. The communication mode

can be set to off-line and a task can be generated and stored in a temporary task file. If the generated motion is satisfactory then the file can be downloaded to the arm.

The master arm can be accessed and used as input for joint angles. The master arm has the advantage of being more natural in the sense that it physically resembles the manipulator arm. The master arm together with the simulated graphical arm can be an efficient commanding tool.

### **Command tools**

The main idea behind the design of these tools was to make the command procedure as easy and as natural as possible. The command tools can be accessed through the "move arm" environment.

The traditionally available systems are limited to the use of teach pendant with master-slave relationship. The teach pendant or master arm generates a position which is duplicated by the manipulator. In practice, the operator performs the necessary moves on the master arm and the robot manipulator follows. This is a good input device in the sense that all the joint angles are generated simultaneously. The physical resemblance of the master arm to manipulator arm ("spatial correspondence") is also of great help to the operator in the commanding stage.

However, having the master arm as the sole means of commanding imposes certain limitations. For example it is difficult to generate specific types of moves such as moving the gripper in a straight line or keeping its orientation constant during a move, or guiding the gripper during a grasping process. Also, it is difficult to orient the present master arm using one hand only, therefore the operator will require the use of both of his/her hands.

The command tools can be thought of as high level program commands. These commands are made available to the user through the interface and are developed in a way that will be useful together with the graphical simulation interface. Practically there is no limitation in developing these commands. Examples of command tools are the commands that will enable the user to move the gripper in one of the world coordinate axes keeping its orientation constant or altering its orientation without changing the position of the end effector. A similar commanding environment is developed in the gripper coordinates, which can be extremely useful at the grasping stage in an operation. One of the goals of this work was to evaluate several of these alternatives for command structure.

Since the real world robot can interact with the world objects and manipulate them, the program should reflect the changes imposed on the world by the actions of the robot. Therefore, the program should simulate grasping of the objects and moving them around in the workcell (pick and place operation). The benefits and shortcomings of these tools are discussed in the following sections.

### **Individual joint control**

The single key-strokes allow the user to change the joint angle of each joint. This is very similar to the use of a standard industrial robot teach pendant. This rather low level control allows for certain moves that can be useful in testing and calibration of the arm. The individual command of each separate joint requires understanding of the kinematic structure of the arm, and overall, it is an extremely slow command tool. The user has to set each joint angle separately and using the simulated arm, take the necessary steps to position the robotic arm.

A similar command mode is through the use of the **master arm**, which can act as an input source for the joint angles. The use of the master arm provides the user with a fast tool to set a new position for the arm. All the joint angles are read from the master arm

simultaneously and the simulation follows the master arm. In this way, the new position of the arm can be established almost instantaneously. The simulation should follow the arm instantly, but the serial communication slows down the joint angle data acquisition process. Hence, there is a small but noticeable delay between altering the configuration of the master arm and the subsequent simulation of the arm. To use the master arm, of course, the robot controller has to be available and on-line.

### **Inverse kinematics**

It is possible to move the end effector or gripper to a new position and orientation without having to go through tedious manual setting of the joint angles. The host computer, given the desired final position and orientation of the gripper, calculates the joint angles of the robot. If the angles are valid, the robotic arm assumes the final position. These command tools are based on the inverse solution of the robot kinematics. (Appendix A) The accuracy of the final position for the real robot depends on the accuracy of the geometric parameters used in the kinematic equations. (For more information on the inherent limitations on the accuracy of the inverse kinematic solution see Appendix A.)

*EXCAL* contains four separate command tools for specifying the world position and orientation of the gripper:

- Directly specifying the position and orientation in world coordinates
- Using a 3D cursor to position the end effector
- Relative moves in world coordinates
- Relative moves in gripper coordinates

A discussion of this set of tools follows.



### **Directly specifying the position and orientation**

In this mode, the user supplies the necessary information for the final position (X-Y-Z) and orientation RPY (Roll, Pitch, Yaw) of the gripper. The program finds the appropriate joint angles for the requested position and if the position is reachable the arm follows. If the position is out of reach or is denied for the reason of safety, a message lets the user know why the move is not performed.

The positive aspect for this command tool is that the user can specify a known position and orientation. This can be very useful in the design of certain types of tasks. This mode will have limited use in real-time on-line operation of the arm. Most people tend to avoid using this mode because of the more mathematically involved nature of the process. This might be due to the more abstract mental model required to analyze the position and orientation. Specifying the orientation using RPY is particularly awkward, since these angles are hard to visualize. The user has to enter the position and orientation data manually through the keyboard, which limits the speed of the whole operation.

### **Using 3-D cursor to position the end effector**

To overcome some of the problems of the previous tool, a 3-D graphical cursor is introduced. This cursor is of the size and shape of the gripper. The reason for choosing a similar shape for the cursor is to help the user visualize the final position of the gripper. The user can set the position and orientation of the cursor using the cursor keypad of the keyboard. This tool is actually an extension of the previous mode. The position of the cursor is displayed numerically and is updated as the cursor moves. The user can either view the cursor and guide it purely based on the screen display or move the cursor until all the position and orientations are what they should be numerically. This is a quick way of positioning the arm. If the position is achievable, the arm proceeds to the cursor position.

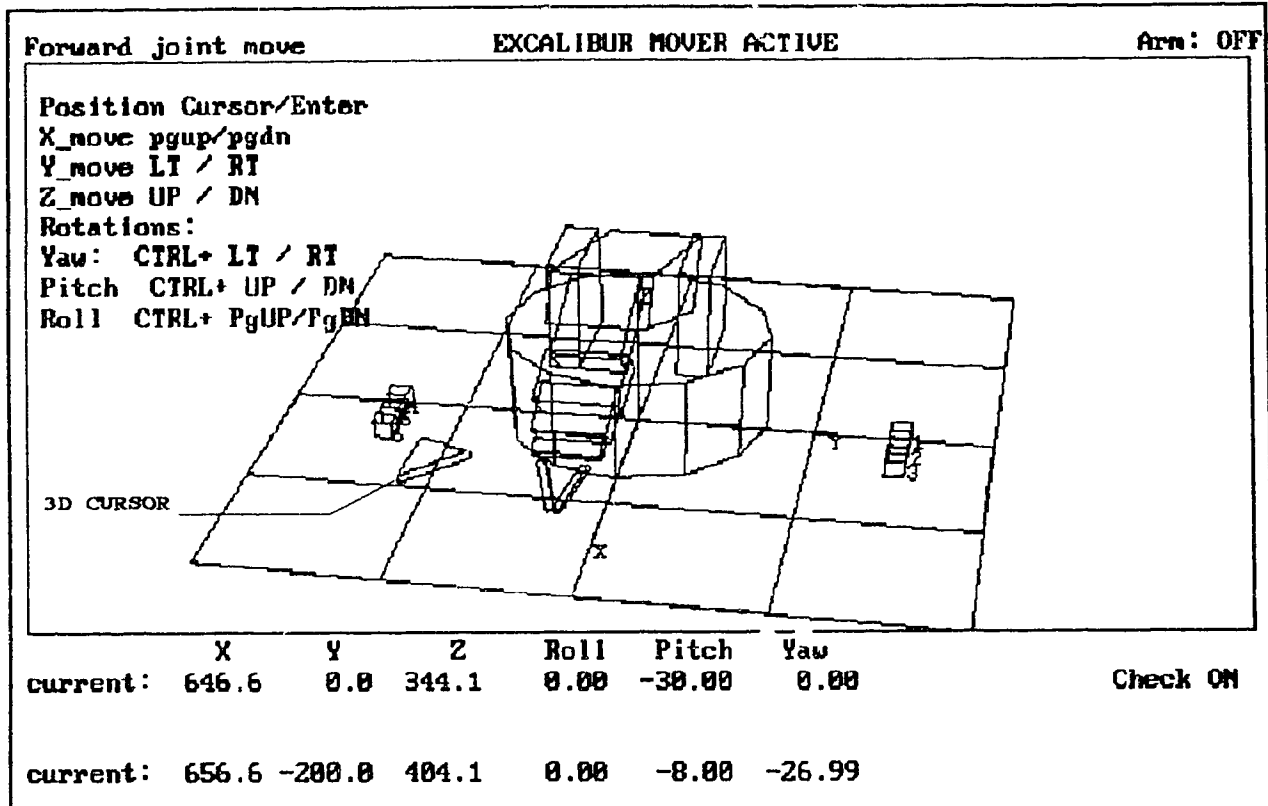


Figure 9 - Inverse mode with 3D cursor

Theoretically this cursor could be replaced by the whole arm. A rubber band link between keyboard and arm can be utilized to move the arm (cursor). These moves are not final and the main screen arm is left at its original position and can be used as the reference. This mode is perhaps preferable to the 3D cursor, however, current hardware limitations do not allow implementation of this mode. The cursor moves are to be fast otherwise the user will be frustrated. The main obstacle is the speed by which graphics on the screen can be updated.

### Relative moves in world coordinates

In certain operations, the operator is required to move the gripper in one world direction while other parameters are held constant. This tool and the one following are designed

to accommodate this need. The world coordinate is the coordinate located at the base of the arm. This tool enables the user to move the gripper in one of the world coordinate principle axes (X-Y-Z) relative to the current position of the gripper, while all other parameters of the gripper are held constant.

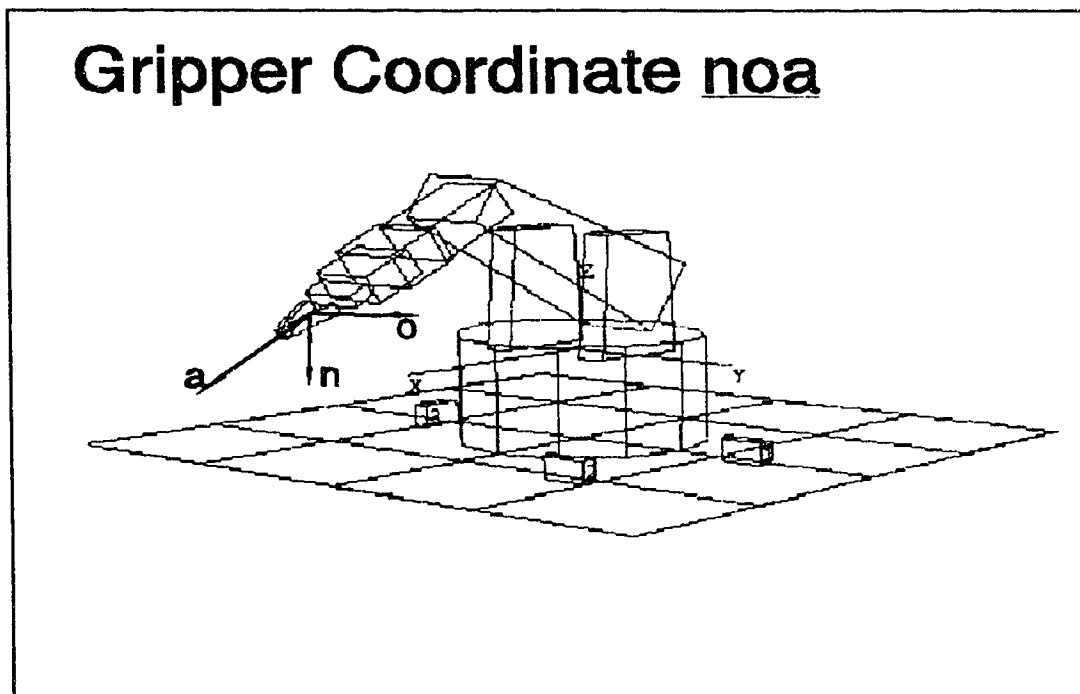


Figure 10 - Gripper coordinate frame

### Relative moves in gripper coordinates

This tool performs an action similar to the previously explained tool. The only difference is that these moves are performed relative to the  $\bar{n} \bar{o} \bar{a}$  coordinate frame (Figure 10) embedded in the gripper. This tool is very effective in guiding the robot in its final approach to an object. It functions well in combination with the virtual camera mounted on the gripper, since the  $\bar{n} \bar{o} \bar{a}$  frame is fixed relative to this view, i.e.  $\bar{n}$  is in the top/bottom and  $\bar{o}$  is in the right/left direction.

### **Problems with the inverse kinematic related tools**

When using the inverse kinematic tools, there are a few problems that the user must consider. First, the program does not have a general collision detection routine. The program does check the position of the gripper against the surface of the table ( $Z < 0$ ) and the base of the manipulator ( $R < R_{Base}$ ). The arm does not use any algorithm to establish a safe path between initial and the final position. If the outcome of these tools is an achievable position, the arm will attempt to reach that configuration. Therefore, it is quite possible that the arm may collide with an object. If this collision is left undetected then arm may be damaged. This of course will not damage the simulated arm and is the main reason for the use of a simulator. It is possible for the user to detect a possible collision in the preview of the move and insert intermediate "via" points which are collision free.

The other problem using these tools arises from the fact that the real robotic arm has limited accuracy. The accuracy of the robot depends mainly on the accuracy of the potentiometers and the resolution of the A/D convertor. Also the electrical calibration of the controller and the mechanical calibration of the arm has to be considered. Thus if the user commands the robot to a given position, the real arm will achieve that position with certain uncertainty involved. The program is designed to update the position of the simulated arm if the move is not achieved within a certain tolerance. Updating the arm position is also an option available to the user at any time. This problem does not pose any difficulty in real-time operation of the arm, but in off-line programming the accuracy of the positions remains a problem to be considered. This problem can be increasingly troublesome when the robot is out of calibration, mechanically or electronically.

In the on-line commanding the inaccuracies can create unexpected results. Because of the inaccuracies present in the position achieved by the arm, when relative moves are commanded (in world coordinate or gripper coordinate), the other parameters defining the position and orientation of the gripper, tend to change. The program attempts to keep

all other parameters constant while altering only one (direction or orientation) parameter (for example, in world relative mode, when a move in X direction is requested, the program keeps the Y, Z, roll, pitch and yaw values of the gripper constant). The actual arm usually cannot achieve the exact joint angles requested. By default, the program checks the position of the arm and compares it with the requested position and updates the joint angles based on the last inquiry made from the controller. The cumulative errors in joint angles eventually drive the gripper away from the initial fixed position or orientation. If repeated moves of this nature are requested with the arm connected and following, the cumulative small errors present in the joint angles become large and noticeable. Due to this the end effector position can go off the original alignment to an unacceptable degree.

A limited solution for this case is to disable the default position attaining check. This forces the program to assume that the robot has achieved the requested position. This will keep the moves close to what the user intends to achieve ( $\pm 0.3$  deg). However, if the robot does not achieve the requested position the user may not be aware of the difference with the real world robot and the simulation may be misleading.

### **Simulation**

The stored tasks are simulated using this feature in which the task is played back line by line or in continuous manner. The user can either forward or backward step in the task file. The objective is to provide enough flexibility so the motion of the robot can be visually verified against collision. As always, the view control can be used at any time.

### **Teach and playback**

Teach and playback introduced in this section are different from the record and simulate that have already been discussed. These features follow the exact protocol set by the

controller of the robot. ( For more information on these two functions consult the Excalibur's user manual [12]).

These functions are included for compatibility. The only difference with the teach and playback commands of the robot controller is that here the program uses the disk as external storage for the task files. The controller has limited memory of 24 KByte (about 80 seconds of recording at 10 Hz). The task files generated for continuous teaching mode can grow rapidly and they require large storage.

## **Options**

At start-up, the program initializes many of its parameters to the default values. The following parameters can be altered: display color, perspective transformation constants, help file path, serial communication delay values, text lines mode and animation adjustment. The **User Manual** for the *EXCAL* program [11] explains the functions of the program in more detail.

## **Use of a mouse as an alternative input device**

In this section the possibility of using the mouse to command the arm is explored. A two button mouse can be easily programmed to generate six controllable variables. This can be done with a combination of the x-y moves of the mouse on the mouse pad with its buttons. Two x and y variables are controlled by mouse moves on the mouse pad. The third z variable is signalled by holding down one of the mouse buttons. To generate the next set of variables the same procedure is utilized and holding down the other mouse button signals that next set of variables.

These two set of variables can be used to alter the position and orientation of the end effector. The idea would be to establish a "rubber band" linkage between the mouse and the end effector. A test routine was written and the mouse moves were used to command

the arm. However at the testing stage it became obvious that the unnatural spatial relation between mouse and arm prohibits the mind from establishing a meaningful relationship between the hand movements and the arm positioning. Therefore at this stage the idea of using the mouse was dropped.

### **Communication concerns**

Regardless of the type of operation, (ie: remote manipulation or on-site manipulation) the host computer and robot controller have to communicate with each other. The interface available with the existing equipment is serial communication with ASCII text format through an RS-232c line. Of prime importance is the reliability of the data transfer. A continuing problem with the existing system is that the hardware for communication differs on the controller and the host computer. It is easy to control and set the communication protocol of the host computer based on the settings of the controller side of communication hardware. However, it was very difficult to debug the communication problems. Therefore, the host computer program had to make sure that the communication was without error or that proper action was taken in case communication broke down.

As a direct consequence of human factors engineering considerations, the serial communication error handling should be transparent to the user. When communication fails the spatial correspondence between real world robot and simulated arm is questionable. For example if improper or incomplete position information is received by the robot then the robot can be in position where the next move request could cause a collision. In these cases the user is informed of communication failure. The user then can interact and take the necessary actions such as resetting the program or the controller of the robot.

## Chapter 4

### Program evaluation

#### 4.1 Introduction

This chapter discusses the overall performance of the *EXCAL* program, that is, whether it meets its goals and satisfies the users. Also the test user's comments are addressed. Based on the discussion, some conclusions are drawn on the effectiveness of such micro computer based interfaces.

#### 4.2 Performance of the Program

The two distinct cases of off-line programming and on-line commanding of the manipulator are evaluated separately. Although these two parts have many aspects in common, the behavior of the program can be considerably different when the arm is connected and the program has to communicate with the robot controller constantly.

#### Expectations

The expectations from a typical robot commanding interface have already been described in detail. The program is evaluated here to see if these expectations have been met and satisfied.

Throughout this work, two distinct subjects are studied. One is related to the interface, its appearance and the human factors engineering aspects of the interface. The general expectations from a good interface were described in chapter one. The second study is



the general behavior of the interface and its command tools. The latter is more specific to robot programming and commanding and distinguishes this interface from other computer interfaces. A brief discussion on the requirements of the program for off-line and on-line commanding follows.

### **On-line commanding**

The interface should provide a natural and safe command environment. The view of the arm and workcell should allow the user to guide the arm visually and solely based on the simulation. The simulation and the real arm should be spatially correspondent. The animated arm should be in phase with the real arm and satisfy the real time nature of the interface.

To avoid user frustration, a fast system response time is essential. The user requires adequate information on the status of the arm and the objects in the workcell. Transparent error handling can increase the effectiveness of the whole interface.

### **Off-line programming**

The off-line programming requires the same features as the on-line commanding. Due to the absence of the real arm, the simulation has to be exact in terms of the spatial path that the arm follows between two points. The path planning and recording stage requires trial and error sequences and ease of task generation and manipulation is very important. The requirements on the real-time performance are relaxed but an accurate task performance timing can be useful.

## ***EXCAL* evaluation**

### **Interface analysis**

The interface is task oriented and accommodates the needs of the user. The menu driven interface with on-line help makes the interface friendly and easy to use. The menu names are suggestive of their usage and require minimum knowledge of the robotic systems. The environment dependent help is available and the function of each menu item or command key is explained. The help text explains the function of each tool and not where the tool should be used. The interface design assumes that the user knows what he/she wants to do and therefore the help should suffice.

The response time of the program running on a 486/33 MHZ computer is very fast and the response to commands is almost instantaneous. The menu responses are equally as fast as the short cut key responses. The menu system has only one level of nesting. The simple menu structure eases menu selection. The commands not represented in the menu system are displayed at the bottom of the screen.

The program always displays messages indicating the state of its execution. In some cases the program cannot process the keys as fast as would be desirable. The reason for this is that the arm moves require a finite time to accomplish (in the physical world and in the simulation). Thus pressing and holding a command key can overflow the keyboard buffer, which can adversely affect the program. On the computer used for testing, once the keyboard buffer fills, the program execution halts and operating system beeps to indicate this fact. This has little effect on the performance of the program in off-line programming mode, but if the program communicates with the arm, this affects the communication line and communication breaks down.

Each environment of the program is devised to perform a specific task. By limiting the choices that each environment offers the user can focus on the task and function in an organized manner.

The task division between the computer and the user is such that the user is required to have a basic knowledge of robotics and the commonly performed tasks. The reason for this is that the current state of AI in microcomputer environment does not provide adequate algorithms to reduce the decision making requirements on the user.

Color is used extensively in the display. Links of the arm are drawn in different colors. This allows each link to be distinguished from its adjacent links and other objects present in the workcell. Color coding is used for different messages. Blinking is used to attract the user's attention and to indicate that the program is processing and the user has to wait. Tests indicated that the user responds to color coding much faster than to text.

The status of different elements of the system is displayed. These include the state of the arm (on or off), the mode of the arm controller, accuracy of the moves, timeout value of the controller and whether position checking is in effect or not.

When the program operates in off-line mode there are no errors that can occur. Once the arm is turned on, the communication status is extensively checked and line errors are corrected by resetting the port. The controller is reset to the exact conditions prior to the error. If the error persists the user is asked to take an action.

### **View capabilities**

The simulation provides satisfactory image information for object manipulation. The main view camera covers the workcell and can be positioned with ease to cover different view angles. The gripper view provides valuable information on the relative position of the gripper and other parts of the workcell. This view is crucial in grasping operations and

allows very easy positioning of the gripper. Two simultaneous views are available and each view is updated for every move performed.

Shortcut keys provide quick view control. Also, the main camera can be set to follow the gripper and keep the gripper at the centre of its focus. This mode is useful in object manipulation. The zoom feature provides close ups when required.

Automatic updating of the two views was not originally implemented, but tests indicated that if the views are updated without explicit interaction the user needs are better accommodated. The two views are updated for every task point in the path.

### **Animation and framing rate**

Quality of the animation and the degree of virtual reality it can create were two of the main concerns in the design and implementation of the interface. The simplified wire frame images allow the interface to achieve an acceptable animation effect. The animation quality determined by the framing rate is affected by many factors. The framing rate is not constant and depends on the complexity of the screen in terms of the number of objects present, the type of view camera and the number of the view cameras active.

Table 2 shows the CPU time for different drawing modes used for creating polygons using the Microsoft C library functions. The results show that drawing an object using filled polygons at its best, requires at least twice the time as compared to the use of wire frame polygons. The time required for filled polygon rises sharply with the size of the polygon (ten times more CPU time is required if the polygon is very large, %80 of the screen size). The table is compiled for the windowed screen coordinate and vertices are supplied as double floating point numbers.

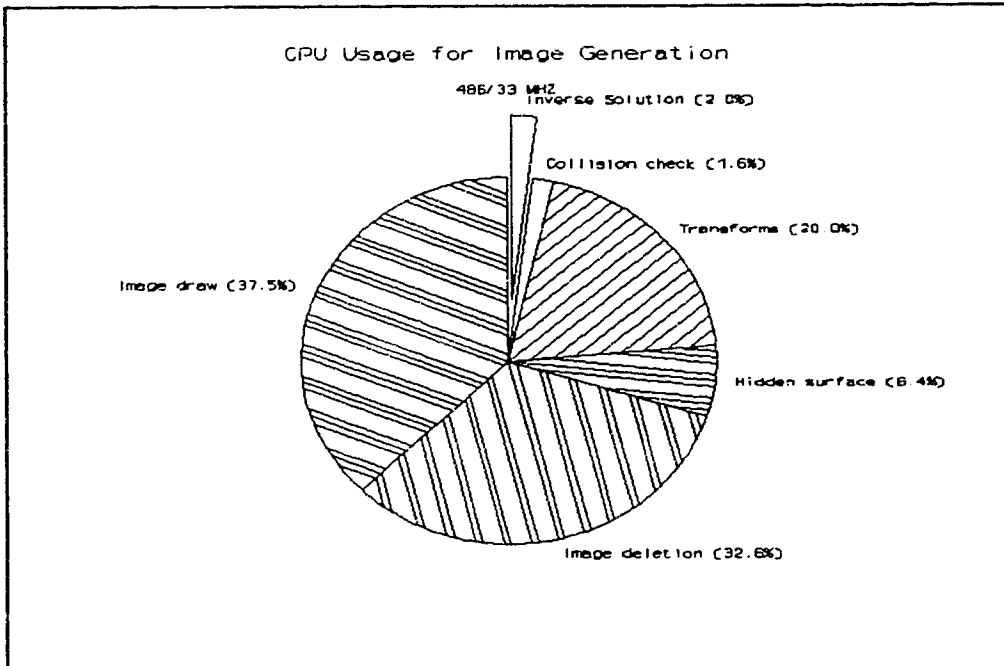
Table 2 - Time comparison for different polygon drawing methods

Rectangle Draw Time (seconds), 486/33 MHZ Microsoft C Library			
Case (Windowed screen coordinates)	Line draw	Wire frame	Filled
Small (aligned along the window axes)	0.0010	0.0011	0.0021
Medium (aligned along the window axes)	0.0012	0.0013	0.0044
Large (aligned along the window axes)	0.0023	0.0024	0.0234
Small (rotated in the window)	0.0012	0.0013	0.0023
Medium (rotated in the window)	0.0015	0.0016	0.0047
Large (rotated in the window)	0.0039	0.0040	0.0290

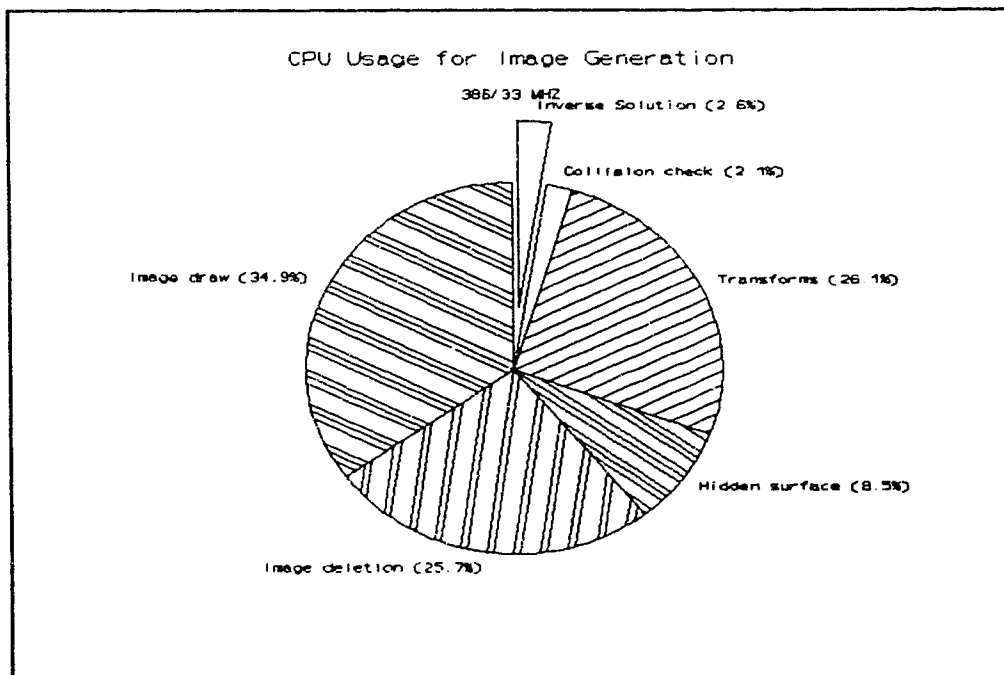
In order to determine the maximum framing rate achievable, the time used by different operations in generating one image on the screen is displayed in table 3. The results on 486/33 MHZ and 386/33 MHZ machines, also different in magnitude, follow a similar pattern. The graphs shown in Figure 11 and Figure 12 show the percentage of time used with different operation in image generation. In general, about one third of the time is used for calculation (transformation, inverse solution and hidden surface removal) and the remainder of the time (two third of the overall time) is used by line drawing functions. The results indicate the limits for animation on the PC machines are mainly due to the time required by the screen I/O operation.

Table 3 - CPU time usage for image generation on the microcomputers

CPU Usage for Image Generation						
CPU type	Inverse solution	Collision check	Transformations	Hidden surface	Image deletion	Image redraw
486 (sec)	0.00116	0.0010	0.013	0.00389	0.020	0.023
386 (sec)	0.00284	0.0023	0.028	0.00928	0.028	0.038



**Figure 11** - Time usage on 486/33



**Figure 12** - Time usage on 386/33

In the simplest operating mode, with a reasonable number of objects (five or six) and using the main camera, a satisfactory real-time animation is achieved. The framing rate

decreases once the two view screens are activated or the main camera is set to follow the gripper. The framing rate decrease as the cameras zoom in where the size of the polygons drawn increases. On the average, the framing rate is fifteen images per second (EGA/VGA mode 16). If the simulation falls behind, then the number of steps for the move can be adjusted to satisfy the real-time nature of the interface. This of course will reduce the quality of the animation in terms of reduced continuity between images. The following two tables and associated graphs show the performance of the *EXCAL* program in terms of the time required to display an image on the screen. The data is obtained from the tests of the program on two different machines (486/33C and 386/33C).

Figure 13 shows the display time vs. the number of world objects in the workcell for different operating modes of the screen cameras. The display time increases as the number of world objects increases. The world objects affect the calculations required as well as the number of polygons to be redrawn. The worst case is for the gripper follow mode where all the entities on the screen are updated for every image. Figure 14 compares the display time on 486/33C and 386/33C based machines. The framing rate on the 386 machine is about two third of the framing rate on the 486 machine.

Table 4 - Display time of an image for different number of world objects

Display Time (Seconds) - 486/33 MHZ EXCAL Program					
No. of Objects	Normal view	Gripper view	Zoomed In View	Two Cameras	Gripper Follow Mode
0	0.04	0.030	0.049	0.037	0.0766
2	0.04	0.0334	0.051	0.04	0.0834
4	0.041	0.0393	0.054	0.044	0.0932
6	0.042	0.045	0.057	0.048	0.0941
8	0.045	0.050	0.058	0.0511	0.103
12	0.052	0.0618	0.064	0.0598	0.137
18	0.061	0.0775	0.073	0.0716	0.179
24	0.07	0.093	0.081	0.083	0.22

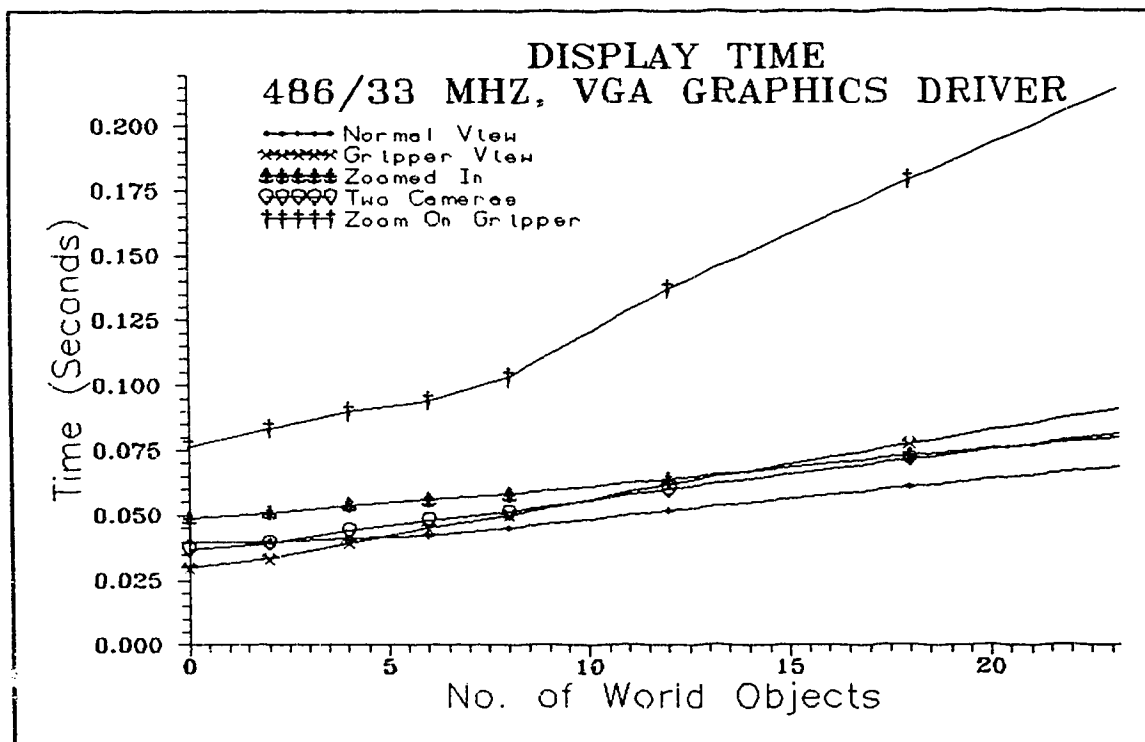


Figure 13 - Display time for the EXCAL program on 486/33C machine



Table 5 - Display time of an image for different number of world objects

Display Time (Seconds) 386/33 MHZ		
No of Objects	Normal View	Zoomed In View
0	0.061	0.074
6	0.066	0.083
12	0.078	0.092
18	0.089	0.103
24	0.099	0.115

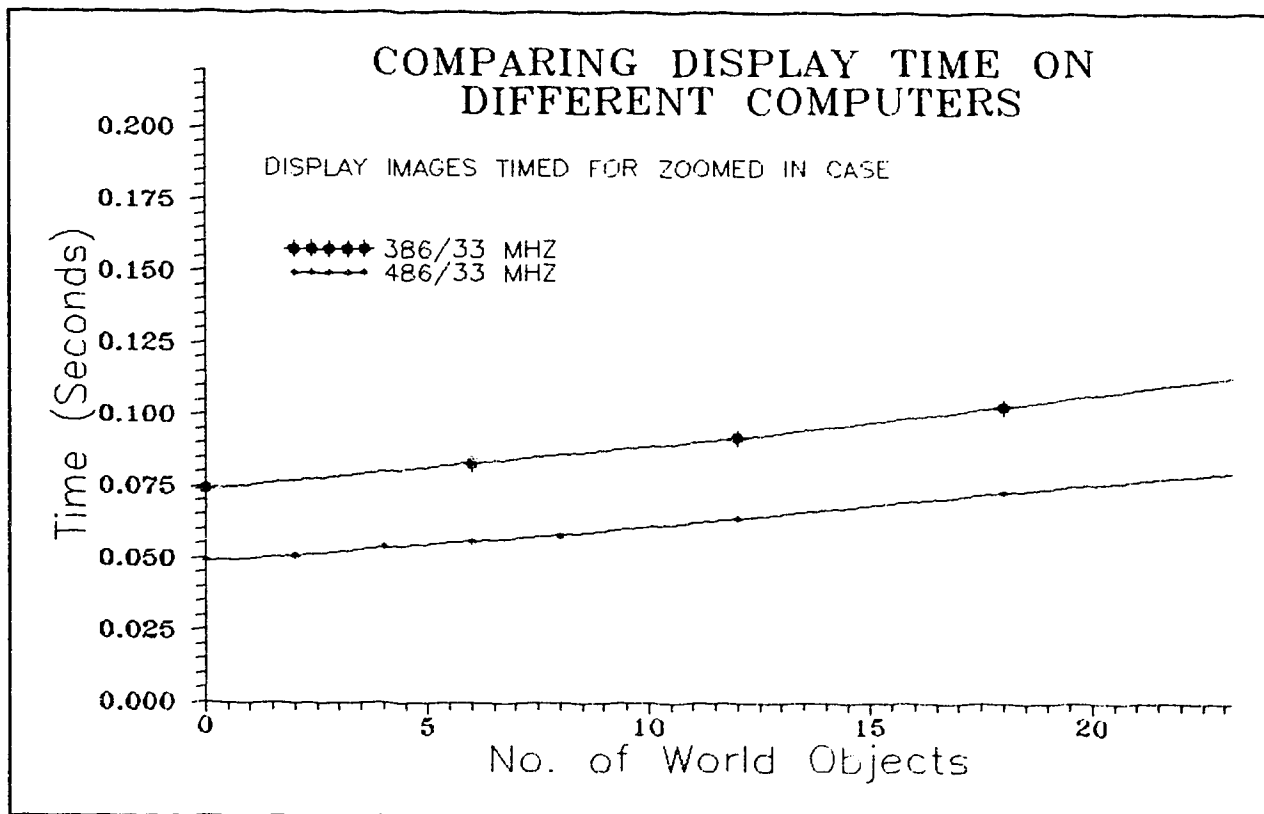


Figure 14 - Display time on 386/33C and 486/33C machines

Due to the absence of technical information on torque-velocity characteristic of the joint motors or of the controller specifications, the dynamic behavior of the arm is not simulated. It is assumed that if the simulation can accomplish a move at least as fast as the real arm, then the real-time requirements of the on-line commanding interface can be satisfied.

### **On-line commanding**

The on-line tasks are either pick and place type or object manipulation. These tasks differ from off-line programming in the accuracy requirements and the tolerance in position errors. In either case the arm is required to perform a grasping action. The moves consist of two parts, the coarse movement and the final fine adjustments.

To be consistent with the previous practice of the robotic systems the program offers a joint manipulation feature much like the teach pendant. Using this mode, the user performs much of decision making on the individual joint moves to configure the arm properly. This has proven to be inefficient and slow. However this is the preferred mode for performing coarse moves.

The possibility of manipulating the arm relative to its current position allows fine moves and orientation corrections. The coordinate system at the base of the robot known as world coordinate allows relative moves in one of the principle directions of the coordinate and is useful in linear moves. The second coordinate frame is the gripper coordinate. This coordinate is associated with the orientation of the gripper. This tool has proven to be very useful at the grasping stage and allows very fine adjustment of the gripper position when used in conjunction with the gripper view.

These two relative tools allow the user to alter the orientation of the gripper while its tip position is held constant. This enables the user to align the gripper before closing its

fingers on the object. Without the gripper view feature, the relative mode tools would be of diminished effectiveness.

The program by default obtains the current position of the arm from the controller and compares it with the requested position. The position checking can be used to signal the completion of one move and allow the program to proceed to the next step. This can keep the simulation synchronized with the real arm. However, due to the slow communication interface, the position check feature slows the on-line operation. For example it is not possible to achieve smooth and continuous moves. Use of this feature with the current communication system is questionable especially when the robot controller checks the requested positions. The user can become very frustrated by slow responses from the arm when the delay between consecutive moves is long. One possible solution to overcome this limitation and achieve smooth continuous moves, is to turn off the position checking feature of the program and decrease the accuracy setting of the controller. However these actions violate the safety requirements of the robotic arm operation.

To preview a move and check for collisions the arm can be set to off-line mode. In this mode, the simulated moves are not automatically downloaded to the arm. Once satisfied with a move then a single key stroke will move the actual arm to its corresponding screen position. Visual collision avoidance is possible and the tests performed confirm this. The tests were performed with simple rectangular prisms representing obstacles. A more complex workcell has yet to be investigated.

The controller side of communication is slower in data handling than the host computer. The communication board on the controller does not keep up with the rate of incoming characters (the controller does not keep up with its published baud rate). This is mainly due to the CPU speed of the controller and hardware modification is required to overcome this problem. This often causes communication errors. The frequency of error occurrence increases with more stringent position checking. The communication errors

are handled without user interaction. If software fails to correct the communication line or if the controller does not respond, then the user has to manually reset the controller. This shortcoming can disrupt the on-line commanding to an unacceptable degree.

The availability of the master arm is beneficial and allows easy coarse moves. The master arm, by solving the inverse kinematics of the arm mechanically, reduces the computational load of the program and theoretically the animation should benefit. However due to slow communication, there is a noticeable lag between the moves of the master arm and the simulation.

### **Off-line programming**

There is no basic difference in the features offered in the off-line programming and the on-line commanding. The simulation performs all the moves possible by the real arm and detects potentially dangerous situations such as collision with the table and prevents the move. The simulation generates accurate joint angles to be recorded and provides record and playback facilities.

It is very easy to record the position of the arm. A single key stroke stores the position in a file. The stored path can be played back and new positions can be inserted or deleted from the task file.

The traditional programming using a teach pendant is awkward and time consuming. This package provides an affordable and very competitive off-line programming utility. The graphical interface allows the tasks to be verified by simulating the robot motion.

### **Compatibility with the controller**

To fulfil the compatibility requirements of the interface with the robot controller, the program emulates the controller teach and playback mode. In these modes the program

provides external storage for the controller. The task files generated in this mode can be played back and simulated. The only difference in the task files generated in this mode is the stored value for the gripper which is either open or closed. The normal record feature of the program stores the steps for open/close of the gripper relative to its current aperture. The simulation is designed based on the relative open/close values and thus the simulation can behave differently if these teach files are simulated. Otherwise the file formats are exactly the same and cannot be distinguished by their appearance.

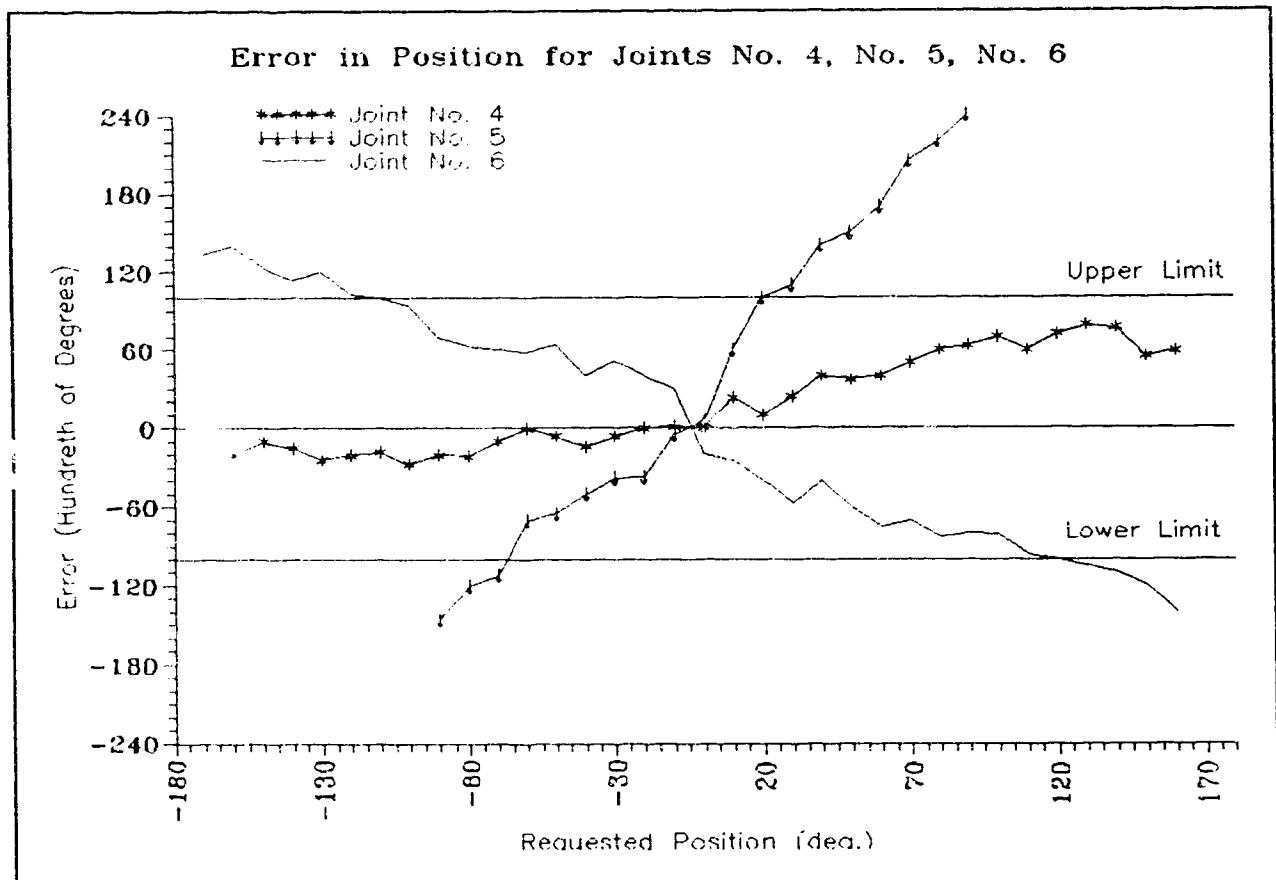
### **4.3 Program testing**

#### **Calibration notes**

The accuracy of the robot (that is the accuracy by which a requested position in world coordinates can be achieved) depends on the proper electrical and mechanical calibration. The possible problems related to calibration are discussed. Small errors in the joint angles can produce large position errors for the end effector. Therefore the robot must be calibrated so that the robotic system will operate properly.

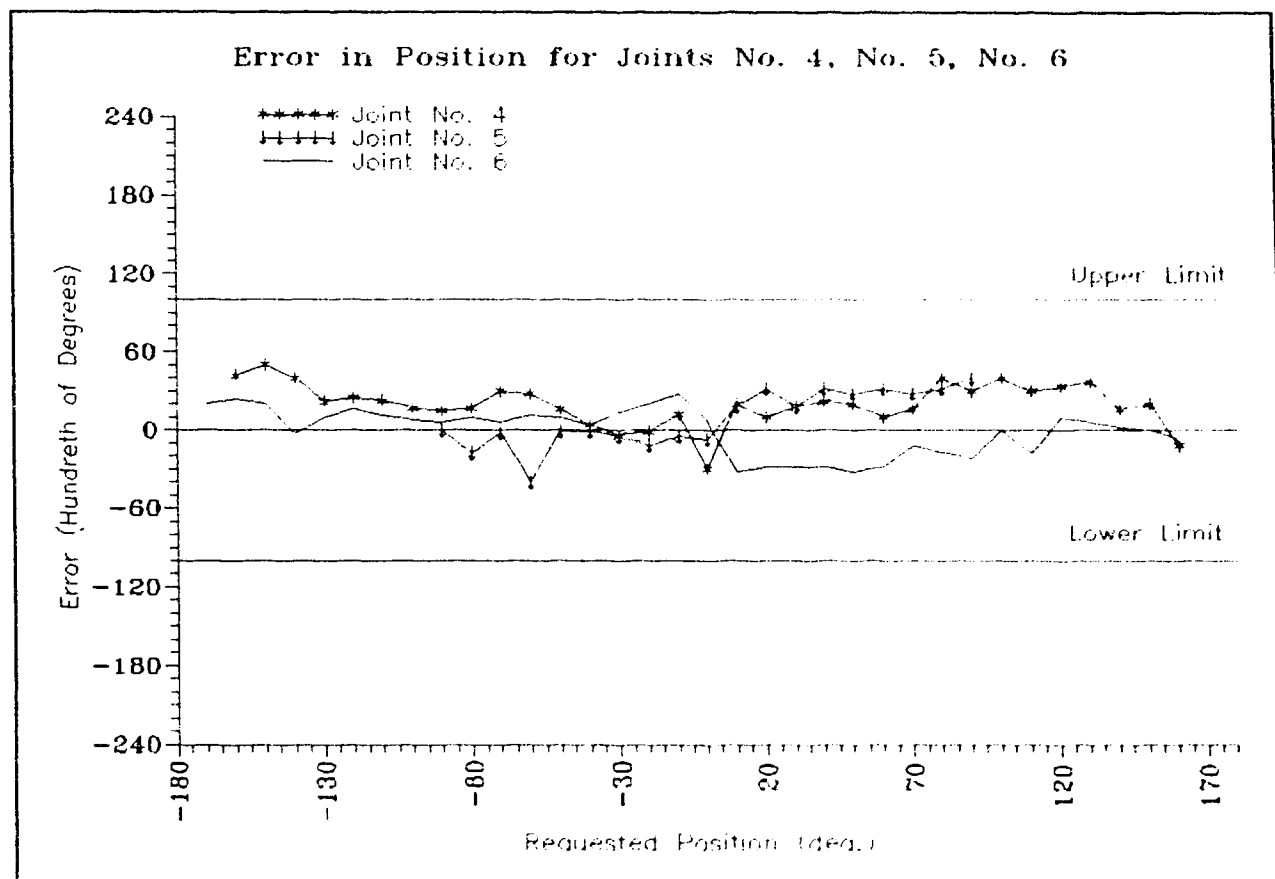
Usually, the electrical accuracy is fixed and depends on the quality of the position encoders and the resolution of the A/D, D/A conversions. This electrical circuitry is designed to produce an acceptable error level, and is quoted as the inaccuracy present in each joint angle. For the present system the electrical accuracy is  $\pm 0.30$  degrees for each joint angle. The uncertainty due to electrical inaccuracies is fixed and the software program and robot controller deal with these inaccuracies by introducing a tolerance in the accuracy by which each joint position is achieved. The default value for tolerance is set to 1.0 degree by the robot controller. This value is adapted by the host computer. The software can change this value as required.

The A/D and D/A boards may also require calibration. As an example of the possible electrical calibration problems the case for the excalibur's controller, encountered during the performance evaluation of the program, is discussed. The *EXCAL* program has the option of checking the achieved position against the requested position. During initial testing, it became obvious that there were discrepancies between the requested position and the reported position. The problem was not obvious since the default controller position check did not report any problems related to position achievement. To diagnose the problem the difference between requested position and the position reported after the move was completed was recorded for every joint along its range (Figure 15).



**Figure 15 - Errors in position before calibration**

The upper and lower limits are the tolerance setting of the *EXCAL* program. The program execution under this condition often resulted in over flowing the controller's buffer, and subsequent crash of the Excalibur arm. The previous graph shows a linear deviation from horizontal level (zero error) for each joint. Through adjusting the potentiometers on the A/D convertor board (zero and gain potentiometers) the slope of the line was brought to about zero and the zero offset control allowed minimizing the error for each joint. The following graph (Figure 16) displays the errors in position after the electrical calibration.



**Figure 16** - Errors in joint positions after electrical calibration

The mechanical inaccuracy on the other hand is a problem that can repeatedly occur. The mechanical zero position of each joint is set by the joint potentiometer to produce

electrical zero. As the robot is being operated, this setting can change. Therefore the mechanical calibration is a procedure that has to be performed from time to time. It would be preferable to have the computer program perform the calibration. One possible way to perform mechanical calibration through software is to find the difference between requested electrical zero angle and actual mechanical zero for each joint. The errors in each joint are then noted in the computer program and the necessary correction can be applied through software. This procedure requires an external sensory system to report the mechanical zero positions. The current version of the robot under study is not equipped with this kind of external sensors and manual mechanical calibration is the only possibility.

### **The workcell**

Because of the uncertainties present in the accuracy of the robot, the workcell has to be mapped to match the program. In doing so, a grid paper is used to map the floor of the workcell. The arm is moved to known positions and repeating this procedure for different positions in the world coordinates will enable the user to map the floor of the workcell with the program.

### **Testing procedure**

The objective of the study is to evaluate effectiveness of the interface. This evaluation consists of two distinct parts. First the interface has to be studied for the human factors engineering aspects. This includes the evaluation of screen setup, keyboard layout, menus and so on. Secondly, the interface has to be evaluated in terms of its effectiveness in real time commanding of the arm and in off-line task planning.

The available test users were mainly engineering students with some knowledge of computers but with no exposure to robotic systems. Some comments from more experienced computer programmers were collected. The user's views as reflected through



the questionnaire (Appendix C) were compiled and the points approved and the parts disliked are discussed below. Each test user on the average spent three hours to become familiar with the software and its capabilities. At this stage little or no help was offered. Once the users were familiarized with the setup of the program, they were asked to perform simple on-line tasks and off-line programming. To simulate remote manipulation the view of the robot was blocked and the graphical interface was the main visual information available to the users. Also there was a fixed video camera providing video information from the arm and the workcell. The use of the camera proved to be difficult to manage and very dependent on the lighting and view direction.

The on-line task performed was either picking a block and placing it in a prespecified position or moving the arm in a straight path (Figure 17). In the latter case, an external device would notify the user if the arm was off the desired course.

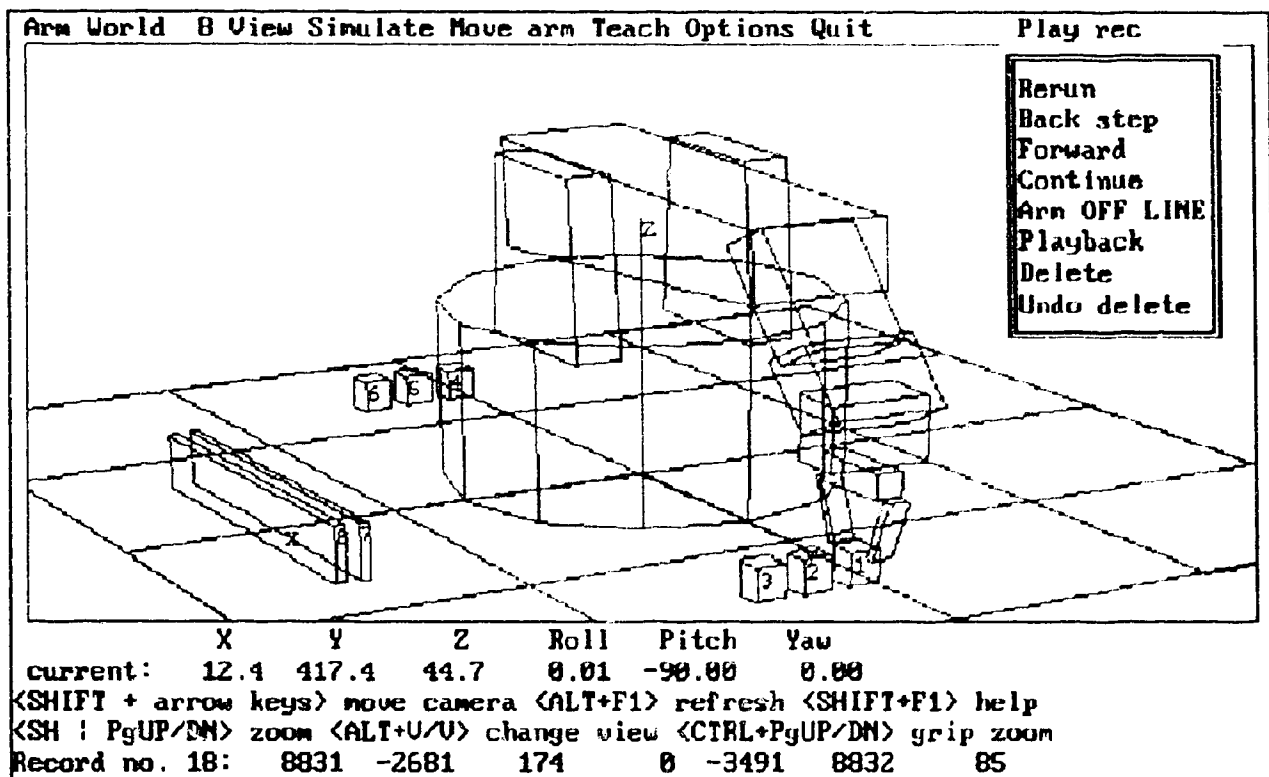


Figure 17 - Test user's workcell layout

In either case, the users had to make a world data file describing the setup of the workcell. Generating the description file is simple and requires little time. On the average users could make these files in five to ten minutes. The pick and place task (Figure 18) was relatively simple and could be performed in less than five minutes. Moving the arm in predefined paths (between the two parallel objects shown in Figure 17) proved to be difficult and several trials were necessary to be able to accomplish the task. The test users spent more than an hour to be able to do this task.

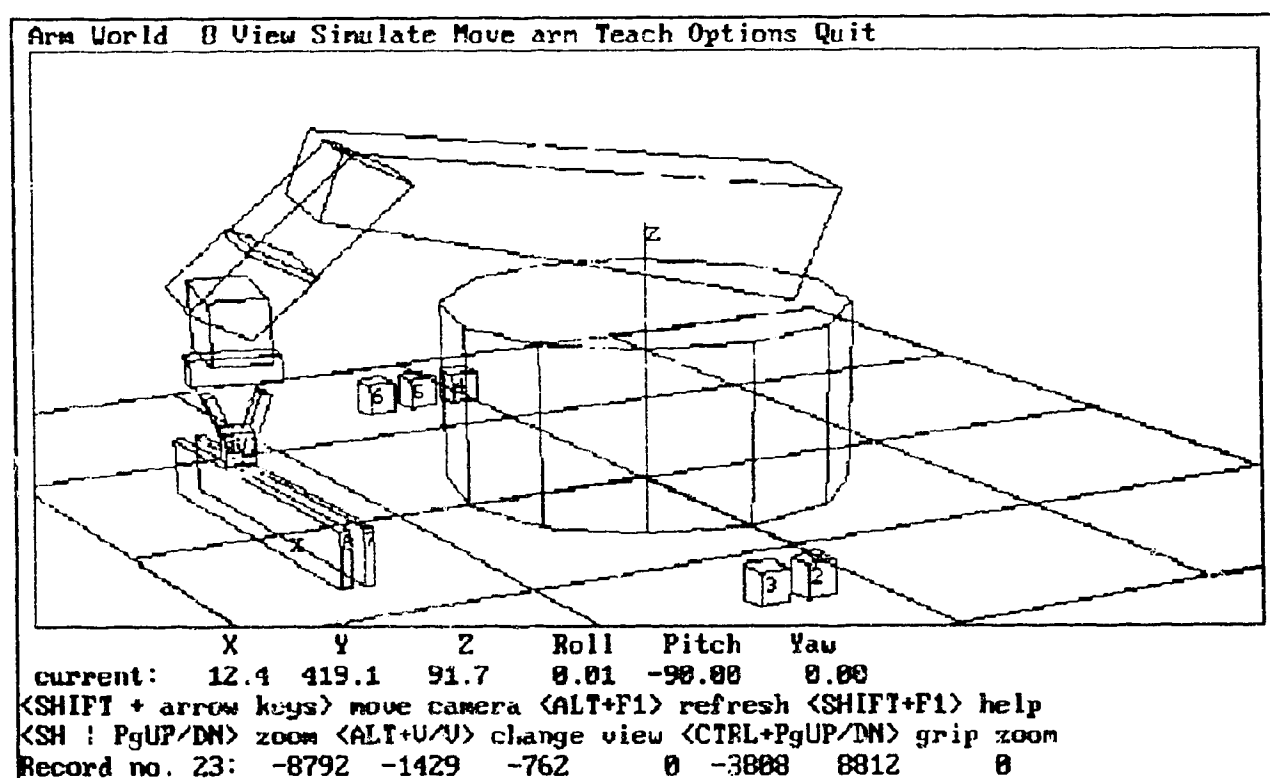


Figure 18 - Test user's generated task

The test for off-line programming consisted of 1) setting up a world description file for two or more blocks, 2) specifying the target position and 3) creating a task file to move the blocks. The users had to repeat the process of creating the task file a few times. Difficult aspects of this task were the orientation of the gripper during approach and the

amount of the gripper closure to grasp an object. The latter problem arises from the fact that the gripper's aperture is not fed back to the program and can be different from the simulation. Once the test users became aware of this deficiency they could create a task file in half an hour.

The familiarity level that the test users achieved can be rated as skilled in working with the software and novice in working with the arm.

### **User approved features**

Generally the users approved the performance of the program. The fast response of the program makes the software easy to use. The graphical simulation of the arm does raise concern at certain operations but overall, the information enables the user to perform the required tasks. The image can be confusing when the gripper approaches an object to grasp it.

The users were pleased with the dual view capability of the program and the automatic updating of the views allows the user to have a better picture of the relative position of the arm.

The relative move tools (world and gripper coordinates) proved to be very useful. The relative to gripper tool along with gripper view was very effective in accurate positioning of the arm relative to the objects.

The record and playback utilities of the program were easy to operate and task generation and manipulation were very easy.

The simulation is not dynamically exact and can be faster than the real arm in many instances and the real time requirements for on-line commanding were satisfied.

### **User disliked features**

There were concerns for the quality of the graphics and viewing information. When the gripper is very close to the objects or the table it is hard to judge the exact situation. For example it is difficult to visually determine if the tip of the gripper is in contact with an object or the surface of the table. If the numerical position display and color changes were not implemented it would be difficult even to judge if the gripper is below the table or not. These deficiencies are due to the nature of the wire frame images used in the simulation. For close object manipulation a more detailed graphical representation is required (solid surfaces with proper shading and shadow).

The joint manipulation mode of the move arm was quite slow and inefficient. The users preferred other means of control for coarse moves (eg: the master arm). Also the lack of high level commands (for example a command to move the arm to the grasping position of an object) raised users concern.

The slow serial communication interface was very frustrating at times. The program performs much faster when the arm is set to off-line. In the operation requiring constant communication with the arm controller, the performance of the program deteriorates. The performance of the software is limited by the capabilities of the controller side of the serial communication line.

## Chapter 5

### Conclusion

The *EXCAL* software seeks to provide a realistic micro computer based robot simulation package. The research program was developed for studying the graphical interface and tool development. The wire frame graphical representation was selected over solid modelling. This selection is based on several reasons:

The most restricting factor is the slow graphics hardware. The program requires real-time operation. Simulation based on wire frame representation allows for smooth animation in real-time.

Wire frame representation along with hidden surface removal provides satisfactory visual information for most of the applications.

Overall the program performance was satisfactory and very competent. Although the graphical simulation depends on the simulation model used, the program shows that it does not have to be overly complex.

The inexpensive programming interface for the PCs allows testing and evaluation of different programming and commanding tools (e.g. master arm, view cameras, etc.). High level command tools can be designed and implemented using this system.

The two views explored have proven to be very useful. The gripper view in particular, is very helpful and is essential to the system. With growing capabilities of the graphics hardware for the PCs, other view cameras can be added as required. One possibility is the simultaneous generation of stereo pairs for better depth perception.

A detailed image is required for the view of the gripper in close interaction with other objects (solid surface rendering, shading etc.).

The communication interface is very slow and hardware improvement is required.

The joint manipulation mode (simulating a teach pendant) is inefficient and very awkward.

The relative move modes are very useful especially for performing fine moves and accurate positioning of the arm near grasp points.

The master arm is still a preferred input device and very useful for coarse moves.

This interface offers easy and very competitive off-line programming utility.

The future plans call for an interface to CAD packages for workcell setup. Also an end effector selection option for testing different tools is desirable. More high level commands are required along with automatic and efficient path generation. These improvements would produce a package of considerable usefulness in actual production processes.

## References

- 1- **Stauffer, N. Robert**, "Robot System Simulation", The Robotics Today, volume 6, number 3, June 1984.
- 2- **Howie, Phil**, "Graphic Simulation for Off-line Robot Programming", The Robotics Today, volume 6, number 1, February 1984.
- 3- **Speed, R.**, "Off-line Programming for Industrial Robots", Proceedings of the 17<sup>th</sup> International Symposium on Industrial Robots and Robots 11, April 1987.
- 4- **Larson, G. and Donath, M.** "Animated Simulation of Intelligent Robot Workcells", Proceedings of the Robot-9 Conference, volume 2, June 1985.
- 5- **Zhang, H.**, Through private communication, University of Alberta, Department of Computer Science, February 1992.
- 6- **Yoffa, A. Nathan**, "Workcell Simulation Case Study: A Spot Welding Application", Proceedings of the 17<sup>th</sup> International Symposium on Industrial Robots and Robots 11, April 1987.
- 7- **Derby, Stephen**, "In Position: Simulating Robotic Workcells on a Micro", CIME, volume 5, number 2, September 1986.
- 8- **Gonschior, Martin and Schunke, Andreas**, "Graphisch Interaktive Robotersimulation auf Personal Computern", VDI-Z, volume 131, number 10, Okt. 1989, p. 51-54.
- 9- **Ravani, B.** CAD Based Programming for Sensory Robots, Young, K., Bennaton, J., "Off-line Programming of Robots using a 3D Graphical Simulation System", Berlin / Springer-Verlag, NY, c1988.
- 10- **Khatib, O., Craig, John J., Lozano-Perez, T.** The Robotics Review 1, Brady, M., "Problems in Robotics", The MIT Press, c1989.
- 11- **Simonian, A.** EXCAL's User's Manual, University of Alberta, 1991.
- 12- Excalibur User's Manual, Part A and B, Version 1.5, RSI Robotic System International Ltd., 1986.
- 13- C ASYNCH MANAGER, user reference manual, Blaise Computing INC., 1987.

- 14- **Sutcliffe, A.** Human-Computer Interface Design, MacMillan Education, 1988.
- 15- **Brown, C. Marlin** Human-Computer Interface Design Guidelines, ABLEX Pub. Co., c1988.
- 16- **Laurel, B.** The Art of Human Computer Interface Design, Addison-Wesley Pub. Co., c1990.
- 17- **Yuguang (Eugene), Cao**, "Thesis: Collision-free Motion Planning And Application to the PUMA 560 Manipulator", University of Alberta, October 1990.
- 18- **Hearn, D., Baker, M. Paulin**, Computer Graphics, Prentice-Hall Inc. c1986.
- 19- **Paul, Richard P.** Robot Manipulators, The MIT press, c1989.
- 20- **Stone Henry W.** Kinematic modelling, Identification, and Control of Robotic Manipulators, Kluwer Academic, Boston, c1987.



## Appendix A

### Kinematic Modelling

#### Introduction

Kinematic models are required for analysis and control of the robot manipulators. These models describe the relationship between the robotic arm parameters and a coordinate frame. Kinematic models relate the position and orientation of the end effector to the joint angles of the arm.

Usually these models are based on the manufacturer's specification of the robotic arm. Therefore real world factors such as gear backlash, friction, link compliance, encoder resolution, joint wobble, and manufacturing errors are neglected [19]. These affect the accuracy of the model, but ease the formulation of the analytical solution. Usually simplified models result in a closed form of inverse kinematic solution.

The **Denavit-Hartenberg** model is used, and this appendix will introduce the basic approach taken by this model [20]. The forward and inverse kinematic equations for the *EXCALIBUR* arm will be derived.

### Conventions used by Denavit-Hartenberg model

Manipulators consist of links connected in series by joints. Rigid body, homogenous transformations can describe the position of each link. Traditionally the transformation matrices describing the relation between one link and the next one are called A matrices. An A matrix is simply a transformation that defines the relationship between two coordinate systems at the two ends of a link. Thus  $A_n$  describes the position and orientation of the link n relative to link n-1. Using matrix multiplication the relation between different links can be established. For Example,

$$T_2 = A_1A_2$$

the  $T_2$  describes the position and orientation of the coordinate attached to the end of link 2 relative to the coordinate located at the beginning of link 1. Usually industrial robots have six links and the T matrix will be:

$$T_6 = A_1A_2A_3A_4A_5A_6$$

The present *Excalibur* arm is a six link manipulator with six degrees of freedom. The forward kinematic and inverse kinematic solutions for this manipulator are determined in closed form. The notation T is historically used to represent the product of A matrices.

### Notation

The  $T_6$  matrix is a homogenous rotation translation transformation matrix that contains the position and orientation of the end effector.

$$T_6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The following three matrices are the basic rotation matrices used to determine **A** matrices for each link of the arm. The last matrix is a general translation matrix.

$$Rot(x, \theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(y, \theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(z, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trans(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### "A" matrices

The **A** matrix is a homogenous transformation consisting of a series of rotations and translations.

### Joint coordinate convention

To be consistent the Denavit-Hartenberg model uses the following convention to assign coordinate frames to each link of the manipulator. For the case under study with revolute joints,  $\theta_n$  is the joint variable. The parameters used to determine the  $A$  matrices using Denavit-Hartenberg method (Figure 19) are defined as follows:

$d_n$  is the distance along  $x_n$  between the axis  $z_{n-1}$  and  $z_n$ .

$\alpha_n$  is a rotation about  $x_n$  that orients the  $z_n$  based on  $z_{n-1}$ .

$a_n$  is the distance between  $x_{n-1}$  and  $x_n$  along the  $z_{n-1}$  axis.

$\theta_n$  is the angle between  $x_{n-1}$  and  $x_n$  measured around  $z_{n-1}$ .

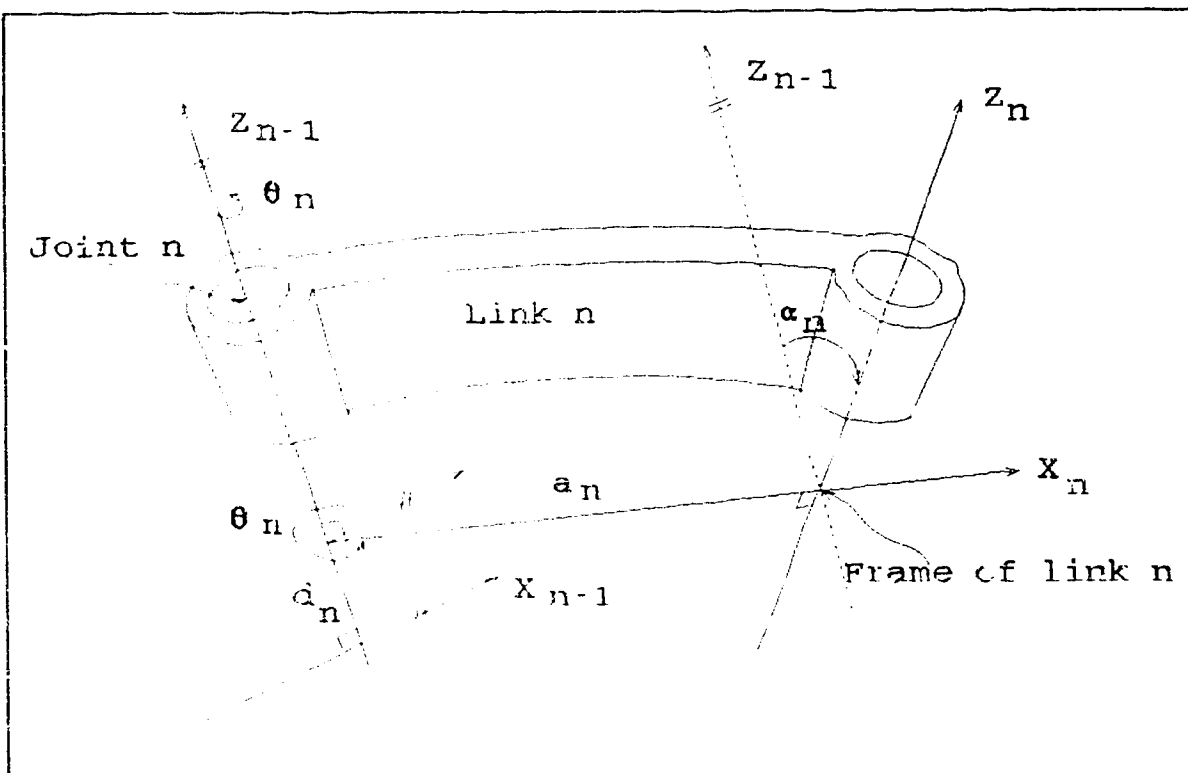


Figure 19 - Link parameters  $\alpha$ ,  $\theta$ ,  $d$  and  $a$

Following the above convention and the assigned coordinate frames to each link of the manipulator (Figure 20), the relationship between successive frames  $n-1$  and  $n$  is established by performing the following sequence of the rotations and translations:

- rotate about  $z_{n-1}$  by angle  $\theta_n$ ;
- translate along  $z_{n-1}$ , a distance  $d_n$ ;
- translate along rotated  $x_{n-1}$ , now parallel to  $x_n$ , a length  $a_n$ ;
- rotate about  $x_n$  by the twist angle  $\alpha_n$ ;

$$A_n = Rot(z, \theta) Trans(0, 0, d) Trans(a, 0, 0) Rot(x, \alpha)$$

$$A_n = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_n = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$A_n$  represents the composite transform for link  $n$ .

### Specification of $T_6$ in terms of $A$ matrices

The general relationship between the  $T$  and  $A$  matrices is:

$${}^{n-1}T_6 = A_n A_{n+1} \dots A_6$$

hence the end of manipulator with respect to the base,  $T_6$  is given by:

$$T_6 = A_1 A_2 A_3 A_4 A_5 A_6$$

For additional transformations the relation between homogenous rigid body transformation can be used to determine the position of the manipulator. Suppose that the

position of the manipulator with respect to the world coordinate frame is given by transformation matrix  $Z$ , and the position and orientation of the end of the attached tool with respect to the end of manipulator is given by  $E$ . Then the position and orientation of the end of the tool with respect to the world coordinate frame can be found using:

$$X = ZT_6E$$

### Kinematic Equations for *Excalibur* Manipulator

Figure 20 shows the attached coordinate frames for the *Excalibur* manipulator. Based on the assigned coordinates the Denavit-Hartenberg parameters are determined and shown in table 6.

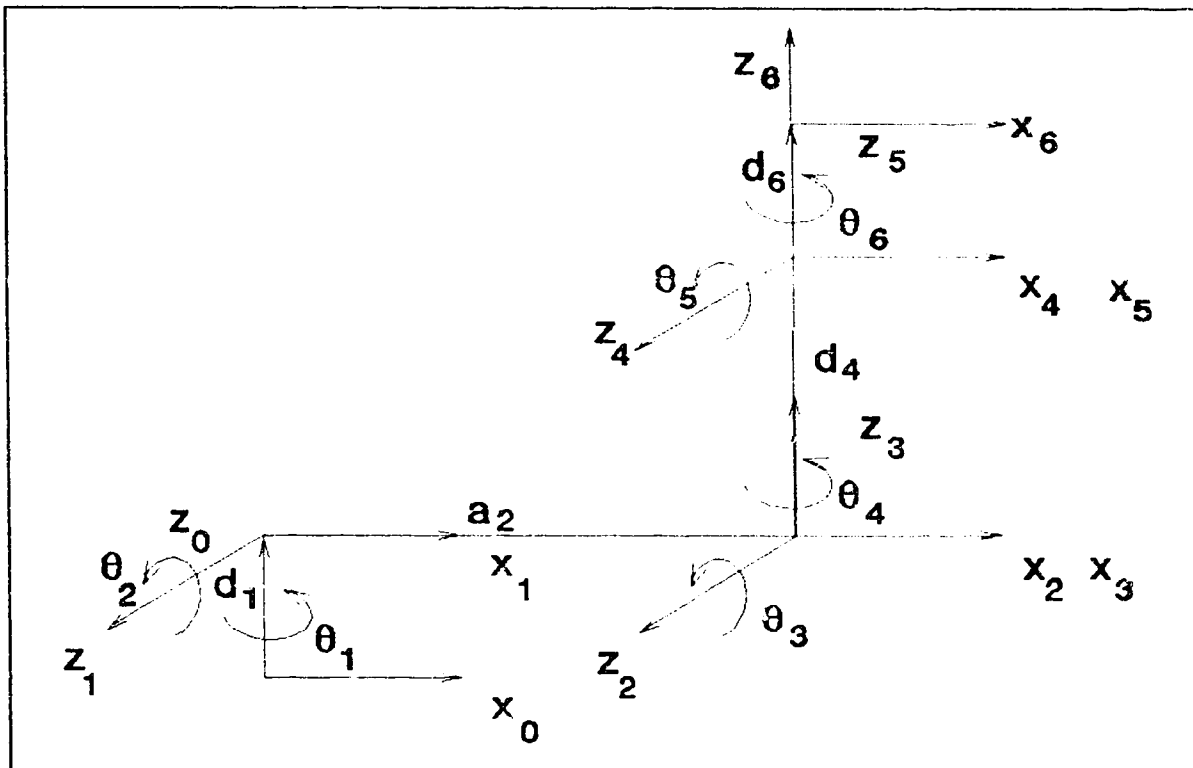


Figure 20 - Coordinate frames for the *Excalibur*

Table 6 - Denavit-Hartenberg parameters for *Excalibur*

Link	Variable $\theta$	$\alpha$	a	d
1	$\theta_1$	90	0	$d_1$
2	$\theta_2$	0	$a_2$	0
3	$\theta_3$	-90	0	0
4	$\theta_4$	90	0	$d_4$
5	$\theta_5$	-90	0	0
6	$\theta_6$	0	0	$d_6$

The **A** matrices are simplified using the following definitions:

$$\begin{aligned} C_1 &= \cos\theta_1 \\ S_1 &= \sin\theta_1 \\ \theta_{12} &= \theta_1 + \theta_2 \\ C_{12} &= \cos(\theta_1 + \theta_2) \end{aligned}$$

**A** - Matrices:

$$A_1 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2 C_2 \\ S_2 & C_2 & 0 & a_2 S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_3 = \begin{bmatrix} C_3 & 0 & -S_3 & 0 \\ S_3 & 0 & C_3 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_5 = \begin{bmatrix} C_5 & 0 & -S_5 & 0 \\ S_5 & 0 & C_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_6 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

Note: The zero joint angles viewed by the Excalibur controller are defined differently from the zero joint angles  $\theta_n$  of the kinematic solution. The relation between these two set of angles are as follows:

$$\begin{aligned} \theta_2 &= \theta_2^* + 30 \\ \theta_3 &= \theta_3^* - 150 \end{aligned}$$

\* indicates angles for Excalibur controller.

The software program supplies the conversion to account for the difference when a move is requested or the joint angles of the Excalibur are inquired.



### Forward Solution

$$T_6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_1 A_2 A_3 A_4 A_5 A_6$$

$$n_x = [(C_1 C_{23} C_4 - S_1 S_4) C_5 - C_1 S_{23} S_5] C_6 - (C_1 C_{23} S_4 + S_1 C_4) S_6$$

$$n_y = [(S_1 C_{23} C_4 + C_1 S_4) C_5 - S_1 S_{23} S_5] C_6 - (S_1 C_{23} S_4 - C_1 C_4) S_6$$

$$n_z = (S_{23} C_4 C_5 + C_{23} S_5) C_6 - S_{23} S_4 S_6$$

$$o_x = -[(C_1 C_{23} C_4 - S_1 S_4) C_5 - C_1 S_{23} S_5] S_6 - (C_1 C_{23} S_4 + S_1 S_4) C_6$$

$$o_y = -[(S_1 C_{23} C_4 + C_1 S_4) C_5 - S_1 S_{23} S_5] S_6 - (S_1 C_{23} S_4 - C_1 C_4) C_6$$

$$o_z = -(S_{23} C_4 C_5 + C_{23} S_5) S_6 - S_{23} S_4 C_6$$

$$a_x = -(C_1 C_{23} C_4 - S_1 S_4) S_5 - C_1 S_{23} C_5$$

$$a_y = -(S_1 C_{23} C_4 + C_1 S_4) S_5 - S_1 S_{23} C_5$$

$$a_z = -S_{23} C_4 S_5 + C_{23} C_5$$

$$p_x = [-(C_1 C_{23} C_4 - S_1 S_4) S_5 - C_1 S_{23} C_5] d_6 - C_1 S_{23} d_4 + a_2 C_1 C_2$$

$$p_y = [-(S_1 C_{23} C_4 + C_1 S_4) S_5 - S_1 S_{23} C_5] d_6 - S_1 S_{23} d_4 + a_2 S_1 C_2$$

$$p_z = [-S_{23} C_4 S_5 + C_{23} C_5] d_6 + C_{23} d_4 + a_2 S_2 + d_1$$

## Inverse Solution

This procedure is based on the method described by Paul, see reference [20].

Step 1:

$$A_1^{-1}T_6 = \begin{bmatrix} f_{11}(n) & f_{11}(o) & f_{11}(a) & f_{11}(p) \\ f_{12}(n) & f_{12}(o) & f_{12}(a) & f_{12}(p)-d_1 \\ f_{13}(n) & f_{13}(o) & f_{13}(a) & f_{13}(p) \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_2A_3A_4A_5A_6$$

where

$$f_{11} = C_1x + S_1y$$

$$f_{12} = z$$

$$f_{13} = S_1y - C_1y$$

from the right hand side of the  ${}^1T_6$ :

$$(3,3) \quad f_{13}(a) = S_1a_x - C_1a_y = S_4S_5$$

$$(3,4) \quad f_{13}(p) = S_1p_x - C_1p_y = S_4S_5d_6$$

Solving for  $\theta_1$ :

$$\frac{S_1}{C_1} = \frac{(p_y - a_yd_6)}{(p_x - a_xd_6)}$$

$$\theta_1 = ATAN2(p_y - a_yd_6, p_x - a_xd_6)$$

Note that the solution degenerates if  $S_4S_5$  is zero. See the discussion for this case and the similar cases at the end of this section.

Step 2:

Using the same matrices as step 1, we have:

$$\begin{aligned} (1,4) \quad f_{11}(p) &= f_{11}(a)d_6 - S_2d_4 + a_2C_2 \\ (2,4) \quad f_{12}(p) - d_1 &= f_{12}(a)d_6 + C_{23}d_4 + a_2S_2 \end{aligned}$$

Let:

$$\begin{aligned} P_1 &= f_{11}(P) - f_{11}(a)d_6 \\ P_2 &= f_{12}(P) - d_1 - f_{12}(a)d_6 \end{aligned}$$

Rearranging the above equation:

$$\begin{aligned} -S_2d_4 + a_2C_2 &= P_1 \\ C_{23}d_4 + a_2S_2 &= P_2 \end{aligned}$$

Now square both sides and adding, and using the trigonometric identity  $\sin(\theta_3 - \theta_2) = \sin\theta_3$  we get:

$$S_3 = -\frac{P_1^2 + P_2^2 - d_4^2 - a_2^2}{2a_2d_4}$$

We cannot find an independent expression to evaluate  $C_3$ . So we must use

$$C_3 = \pm\sqrt{1-S_3^2}$$

Note two points:

1. If the absolute value of  $S_3 > 1$ , the SQRT function fails. This is caused by the desired position given by  $T_6$  being illegal (out of the envelope of the arm).
2. Also we must choose the + or - value of the square root. To resolve this problem we will take the positive square root and check the computed joint angle against its limits. If the angle is out of range, we will adjust it by adding or subtracting 180

degrees from the angle ( $\theta_3 = \text{ATAN}(S_3, C_3)$ ). The computer program uses the corresponding value for the elbow up solution.

Step 3:

Rewriting the above set of equations, and expanding the terms  $S_{23}$  and  $C_{23}$ , we obtain:

$$\begin{aligned} (-C_3d_4)S_2 + (a_2 - S_3d_4)C_2 &= p_1 \\ (a_2 - S_3d_4)S_2 + (C_3d_4)C_2 &= p_2 \end{aligned}$$

Solving this set of equations:

$$S_2 = \frac{p_1C_3d_4 - p_2(a_2 - S_3d_4)}{-(C_3d_4)^2 - (a_2 - S_3d_4)^2}$$

and

$$C_2 = \frac{-p_2C_3d_4 - p_1(a_2 - S_3d_4)}{-(C_3d_4)^2 - (a_2 - S_3d_4)^2}$$

Therefore:

$$\theta_2 = \text{ATAN2}(S_2, C_2)$$

Step 4:

Now we will obtain the matrix products on both sides of:

$$A_3^{-1}A_2^{-1}A_1^{-1}T_6 = \begin{bmatrix} f_{31}(n) & f_{31}(o) & f_{31}(a) & f_{31}(p) & -S_{23}d_1 - a_2C_3 \\ f_{32}(n) & f_{32}(o) & f_{32}(a) & f_{32}(p) & \\ f_{33}(n) & f_{33}(o) & f_{33}(a) & f_{33}(p) & -C_{23}d_1 + a_2S_3 \\ 0 & 0 & 0 & 1 & \end{bmatrix} = A_4A_5A_6$$

where:

$$\begin{aligned} f_{31} &= C_{23}(C_1x + S_1y) + S_{23}z \\ f_{32} &= -S_1x + C_1y \\ f_{33} &= -S_{23}(C_1x + S_1y) + C_{23}z \end{aligned}$$

From the elements (2,3) and (1,3) we obtain:

$$\begin{aligned} (2,3) \quad f_{32}(a) &= -S_4 S_5 \\ (1,3) \quad f_{31}(a) &= -C_4 S_5 \end{aligned}$$

which yields:

$$\begin{aligned} S_4 &= (-S_1 a_x + C_1 a_y) / S_5 \\ C_4 &= (C_{23}(C_1 a_x + S_1 a_y) + S_{23} a_z) / S_5 \\ \theta_4 &= ATAN2(S_4, C_4) \end{aligned}$$

Note that again, we have assumed that  $S_5$  is not zero. See the discussion at the end of this section.

Step 5:

Obtain the matrix product on both sides of:

$$A_4^{-1} A_3^{-1} A_2^{-1} A_1 \quad T_6 \quad = \quad \begin{bmatrix} f_{41}(n) & f_{41}(o) & f_{41}(a) & f_{41}(p) - S_{23} C_4 d_1 - a_2 C_3 C_4 \\ f_{42}(n) & f_{42}(o) & f_{42}(a) & f_{42}(p) - C_{23} d_1 + a_2 S_3 - d_4 \\ f_{43}(n) & f_{43}(o) & f_{43}(a) & f_{43}(p) - S_{23} S_4 d_1 - a_2 C_3 C_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_5 A_6$$

where:

$$\begin{aligned} f_{41} &= C_{23} C_4 (C_1 x + S_1 y) - S_4 (S_1 x - C_1 y) + S_{23} C_4 z \\ f_{42} &= -S_{23} (C_1 x + S_1 y) + C_{23} z \\ f_{43} &= -C_{23} S_4 (C_1 x + S_1 y) + C_4 (S_1 x - C_1 y) + S_{23} S_4 z \end{aligned}$$

From elements (1,3) and (2,3) we obtain:

$$\begin{aligned} (1,3) \quad f_{41}(a) &= -S_5 \\ (2,3) \quad f_{42}(a) &= C_5 \end{aligned}$$

So that

$$\theta_5 = ATAN2(S_5, C_5)$$

Step 6:

From the same matrix product as step 5, we get:

$$\begin{aligned} f_{43}(n) &= -S_6 = C_{23}S_4(C_1n_x + S_1n_y) - C_4(S_1n_x - C_1n_y) - S_{23}S_4n_z \\ f_{43}(o) &= -C_6 = -C_{23}S_4(C_1o_x + S_1o_y) - C_4(S_1o_x - C_1o_y) - S_{23}S_4o_z \end{aligned}$$

So that

$$\theta_6 = ATAN2(S_6, C_6)$$

Procedure to handle the case when  $\theta_5 = 0$ :

In deriving the inverse kinematic equations we assumed that  $S_5$  was not zero. In implementing this method therefore we must deal with two problems:

1. How to detect when  $\theta_5$  is zero?
2. What to do about it!

If  $\theta_5$  is zero, this means that the  $\bar{a}$  vector points directly away from the robot base.

Examining the (3,3) and (3,4) elements of step 1, using  $S_5 = 0$  gives the results:

$$\begin{aligned} (3,3) \quad f_{13}(a) &= S_1a_x - C_1a_y = 0 \\ (3,4) \quad f_{13}(p) &= S_1p_x - C_1p_y = 0 \end{aligned}$$

So that

$$\frac{S_1}{C_1} = \frac{p_y}{p_x} = \frac{a_y}{a_x}$$

This relation  $p_y / p_x$  and  $a_y / a_x$  can be checked as soon as the inverse solution is called with the input  $T_6$  matrix. If this condition is true, then  $S_5$  is zero.  $\theta_1$  can be determined using either of the equations above.

Furthermore, if  $\theta_5$  is zero, the axes of joints 4 and 6 coincide so that the contribution of each joint to the roll orientation is indeterminate. We can arbitrarily set  $\theta_4 = 0$  and let joint 6 take all of the roll.

## Appendix B

### Implementation of the program in the 'C' language

#### Introduction

This appendix covers the programming concepts and actual implementation of different routines. The data-base of the program and graphics techniques used for animation and menu display are discussed. This material with the comments in the actual code can be used for future modification of the code.

#### Frame coordinates

In the following material there are occasional referrals to frame coordinates. In the development of the program two coordinate frames are used. These frames are called kinematic (world) coordinate and the screen coordinate (Figure 21).

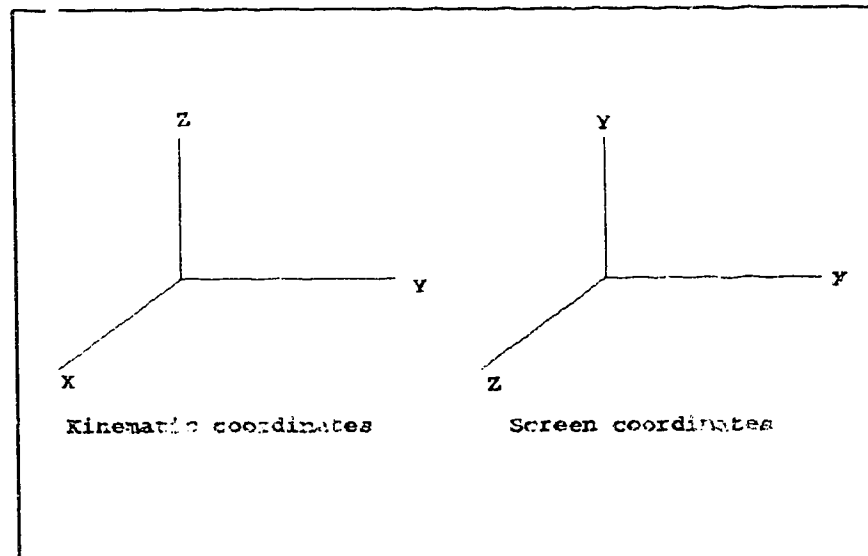


Figure 21 - Frame coordinates



## Graphic representation of the objects

### Data-Base design

Data for each three dimensional object is stored in a two dimensional array. These arrays store the coordinates of the rectangular prisms representing the arm links and the objects in the workcell. The objects can be classified in two groups. One set is the rectangular prisms used to construct the arm, which is composed of six links, two fingers and the base. Except for the base of the arm, all parts are represented using rectangular prisms. A twelve sided polygon prism is used to represent the cylindrical base.

The other set of three dimensional objects contains the world objects. Any object introduced to the workcell as a foreign object is known as world object. These objects are stored in data structures as they are read into the program. The arm links could be stored in similar data structures. However, since computer calculation with data of type "structure" requires more time it was decided to use arrays.

The relationship between vertices and surfaces for each object type is stored in two dimensional arrays. The first subscript in these arrays represents the surface (poly side) and the second dimension holds the indices of the vertices composing that surface.

The program *EXCAL* stores the world objects in internal data structures defined in the following format:

```
struct world_cube{
    REAL xyz[OBJECT_SIZE/3];
    REAL txyz[3];
    REAL tpyr[3];    /* not used */
    int vertices;
    char obj_num;
    char g;          /* set to yes if object picked */
    int color;
    struct world_cube *next_object;
};
```

The program dynamically allocates memory for the data structure as more objects are read into the program. This structure holds information on the number of vertices, coordinates of the object, transfer vector that defines the position of the object, color value for the object, object number, a flag that indicates whether the object is grasped or not and a pointer to the next data structure of its own kind. The final item is used to create a linked list of the objects.

This structure can hold data for as many vertices as required. However, the current version of the program limits the number of vertices to eight for rectangular prisms. Some functions use the explicit orthogonal characteristics of the prisms.

### **Transformations**

To construct the arm each component of the arm (arm link) is transformed to its corresponding position. The transformation matrix for each link  $T_n$  (Appendix A) for link  $n$  is applied to the coordinates of the link in kinematic coordinates. This procedure is repeated for each link and the arm is constructed.

To increase the efficiency of the calculations a dedicated function is developed for each link. Two additional transformations are required to correct the position of the projected objects for changes in the view point. The default view point lies on the  $z$  axis of the world coordinate as defined in Figure 22. A combination of pitch and yaw rotations correct the position of the objects for any other view point. After the new position of a link is calculated the vertices are projected onto the two dimensional screen plane using perspective projection.

Shown here is a sample code that transforms the coordinates of link no. 2 .

```
void transform2(int no_of_points,REAL abs_rot_pt[PNTS],REAL rot_pt[3],\
               REAL pt[3],REAL vxy[])
{
  int i;
  pitch_rot(no_of_points,abs_rot_pt,pt,scr_arm_ang[1]+SHOULDER_HOME);
  for (i=0; i<no_of_points; i++)
    abs_rot_pt[i][1] += L1;
  yaw_rot(no_of_points,abs_rot_pt,abs_rot_pt,scr_arm_ang[0]+camera_pyr[1]);
  pitch_rot(no_of_points,abs_rot_pt,abs_rot_pt,camera_pyr[0]);
  perspective(no_of_points, abs_rot_pt, rot_pt, vxy);
}
```

The names of the functions called in this function are self explanatory. Notice that the *camera\_pyr[]* array holds the pitch and yaw angle for the main camera's position. These two angles relocate the camera on the surface of an imaginary sphere that covers the arm and the workcell.

The first parameter passed to the function is the number of vertices and determines the number of points to be transformed by the function. The original points defining the coordinates of the link n are passed to the function by *pt[]* array. The array *abs\_rot\_pt[]* hold the coordinates of the transformed object. The array *rot\_pt[]* is used for the internal calculation in the function. The array *vxy[]* holds the projected screen coordinates. Perspective projection generates the coordinates for screen images.

For each link a similar function is called with the appropriate transformation sequence. The functions *pitch\_rot*, *yaw\_rot*, and *roll\_rot* do not perform full matrix multiplication. Many elements of each rotation matrix are zeros and full matrix multiplication would be a waste of processor time.

## Perspective projection

To obtain a two dimensional representation of a three dimensional object, perspective projection method can be used. Perspective projection creates the illusion of depth. Figure 22 shows the basic arrangement used for this projection. The program performs this transformation using the following formulas:

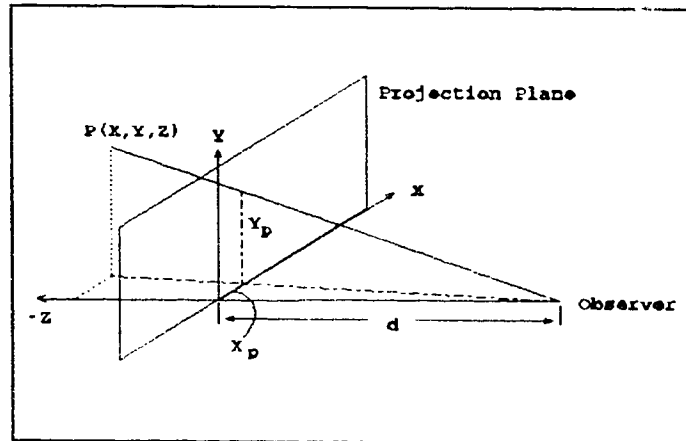


Figure 22 - Perspective projection parameters

$$x_p = x \left( \frac{d}{z+d} \right)$$

$$y_p = y \left( \frac{d}{z+d} \right)$$

$$z_p = 0$$

The actual code performing this transformation is shown here.

```
void perspective(int no_of_points, REAL abs_rot_pt[ ][PNTS], REAL rot_pt[ ][PNTS], REAL vxy[ ])
{
    int i;
    for (i=0; i < no_of_points; i++)
    {
        rot_pt[i][0] = abs_rot_pt[i][0] + mx;
        rot_pt[i][1] = abs_rot_pt[i][1] + my;
        rot_pt[i][2] = abs_rot_pt[i][2] + mz;
        vxy[2*i] = (d*rot_pt[i][0])/(d-rot_pt[i][2]) + off_x; /*-d*x/z + off_x;*/
        vxy[2*i+1] = (d*rot_pt[i][1])/(d-rot_pt[i][2]) + off_y; /*-d*y/z + off_y;*/
    }
}
```

The  $mz$  is added to the  $z$  axis of the object to translate the object to negative  $z$  coordinates. The distance between observer and projection plane is controlled by parameter  $d$ . The parameters  $mx$  and  $my$  are used to control the position of the projected image on the screen (pan). If  $mx$  and  $my$  are set to zero then the image is centred on the screen.

At this stage the screen coordinates are ready and the image can be put on the screen. Before calling line/polygon drawing functions the hidden surfaces have to be determined.

### Hidden surface removal

An object-space based hidden surface removal technique determines the visibility index for each surface. This method uses the plane equation:

$$Ax + By + Cz + D = 0$$

The method calculates the dot product of the view vector and the outward normal vector to the plane. (Figure 23) The dot product determines the angle between view vector and normal to the plane. If the result of dot product is negative the surface is visible. The function *surface\_test* shown below returns the result of this dot product.

```
REAL surface_test(REAL x1,REAL my_y1,REAL z1, REAL x2,REAL y2,REAL z2,REAL x3,REAL y3,REAL
z3)
{
  REAL stest,a,b,c;
  a = (view_dir_xyz[0][0]-x1)*(my_y1*(z2-z3) + y2*(z3-z1) +y3*(z1-z2));
  b = (view_dir_xyz[0][1]-my_y1)*(z1*(x2-x3) + z2 *(x3-x1) + z3*(x1-x2));
  c = (view_dir_xyz[0][2]-z1)*(x1*(y2-y3)+x2 *(y3-my_y1) + x3*(my_y1-y2));
  stest = a+b+c; /* dot product of view_vec and normal of plane */
  return stest;
}
```

This functions uses the three points on the plane to find the normal vector to that plane. Components of the normal vector to the plane are found from the cross product of the two lines in the plane. The dot product of the view vector and the normal vector to the plane generates the visibility index of that surface.

$$\bar{n} = \bar{u1} \times \bar{u2}$$

$$stest = \bar{n} \cdot \bar{v}$$

Note that vector  $\bar{v}$  is directed from view point to the first vertex of the surface under study. If *stest* returns a negative value then the surface is marked as visible. The visibility information of each polyhedron is stored in an array.

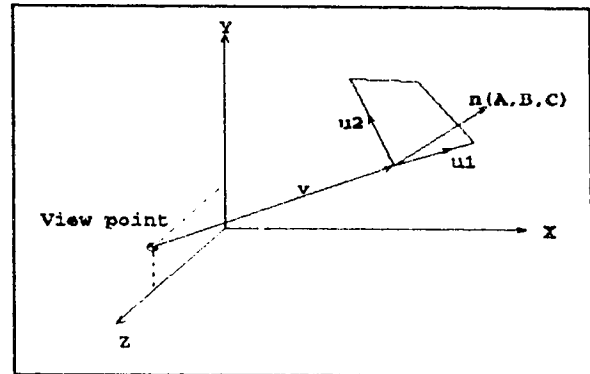


Figure 23 - Hidden surface removal

Once the screen x and y coordinates and visibility information for all the links are determined, the screen image can be generated. The *polygon()* function of the Quick-C library is used.

### Drawing a frame

The following material covers the necessary steps in generating one complete frame. There are five types of objects in each frame. First the grid representing the table is drawn. Then the base of the arm is drawn followed by a frame axis. The world objects laying on the table are plotted next. The last item is the moving part of the frame "the Excalibur arm".

## **Grid**

The grid coordinates are generated at the initialization stage of the program. The grid is five by five, and only the coordinates for the outer points are generated. These grid coordinates go through the necessary transformations for the view camera. The grid plotting function generates the grid by plotting five rectangular polygons. This method takes less time than plotting ten separate lines. The current plotting coordinates are saved for later calls to this function.

## **Arm base**

Excalibur's base is a truncated cone very close in shape to an upright cylinder. The base is approximated by a twelve sided polygon prism. The procedure for generating the plot coordinates is the same for any other objects.

## **Axis**

An axis frame is plotted to show the x-y-z conventions used by the program. To plot this frame, four points are defined and transformed (rotated) by the camera angles.

## **World objects**

World objects are read into the program whenever the user wants to add/replace objects. The objects can be either moveable or non-moveable. This is determined by the color value of the object when loaded. Negative value for color marks the object as fixed and the program assigns the grid color to these objects. The gripper cannot move these objects. Moveable objects can be attached to the gripper and moved around. The world objects are drawn after the axis frame is plotted. If an object is attached to the gripper it is not plotted at this stage. The grasped object located between the gripper's fingers will be plotted following the arm.

The objects set on the table undergo the viewing transformations (rotations). The object (only one) grasped by the gripper has to be transferred to the gripper's current position and orientation. The grasped object is plotted in red color.

### **Arm**

Six rectangular prisms representing the links of the arm and two more prisms as the gripper fingers compose the simulation of the arm. The coordinates for each link are stored in a static array in the function that plots the arm. Using the global values of the joint angles, each link is transformed to its appropriate position and then plotted. The transformation for the gripper's fingers, accounts for the fact that the tip of the gripper moves back and forth as the gripper opens and closes. (ie: the distance from the wrist to the finger tips increases as the fingers are closed).

A global flag indicates if an object is grasped by the gripper. If there is a grasped object the value of this flag points to the number of the grasped object. A set of transformations are applied to this object to position the object according to the gripper's position and orientation.

### **Transformation of the grasped object**

The forward kinematic solution generates the  $T_6$  in the kinematic coordinates.

$$T_6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



The objects are defined in screen coordinate. To transform the grasped object to gripper's configuration the final transform should be applicable to screen coordinates. The following formula can be used for coordinate transformation.

$$T_6 = C T_6^* \quad \text{thus} \quad T_6^* = C T_6^{-1}$$

C is the transform between world coordinate and screen coordinate. The  $T_6^*$  in the screen coordinate is:

$$T_6^* = C^{-1} T_6 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ n_x & o_x & a_x & p_x \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This transformation will transfer any point (in screen coordinate) to the position and orientation of the gripper. To keep track of the relative configuration of the grasped object with respect to the gripper, the procedure initially applies the inverse of the above transformation to the object when it is first grasped. This lets the program keep the current relative configuration of the object with the gripper. This transformation is applied only once at the time when the gripping command is issued. The inverse transformation is:

$$[T_6^*]^{-1} = \begin{bmatrix} n_y & n_z & n_x & -n.p \\ o_y & o_z & o_x & -o.p \\ a_y & a_z & a_x & -a.p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The forward kinematic transformation at later stages positions the object in its correct configuration.

To reduce the computational load, the program does not check for the presence of an object at the gripper. Therefore to make a logical connection between the gripper and the object an explicit request is required. Once the command is issued the program has to confirm the presence of the object between the gripper's fingers and recalculates the new configuration of the object in the data-base list.

### Object grasping procedure

To confirm the presence of an object at proper grasping position the routine finds the closest object to the gripper. This is done by calculating the distance between the tip of the gripper and the objects present in the workcell. A more detailed check follows once the closest object is found. This check will determine if the gripper's coordinates fall inside this object. The procedure assigns a coordinate frame to one corner of the object. Then a transformation based on the following formula is applied to view the tip of the gripper in the newly established coordinate.

$$\bar{x} = \bar{c} + Q\bar{r}$$

$$\bar{r} = Q^{-1}(\bar{x} - \bar{c})$$

The coordinates of  $\bar{r}$  are evaluated in the constructed frame (Figure 24). The  $Q$  transformation (rotation) in the above equation is generated using the coordinates of the object (right angle prisms only). The  $\bar{x}$  represents the position of the gripper and  $\bar{c}$  is the coordinates of the base of the constructed frame (this is available in the data-base of the program.). If the  $\bar{r}$  has positive coordinates

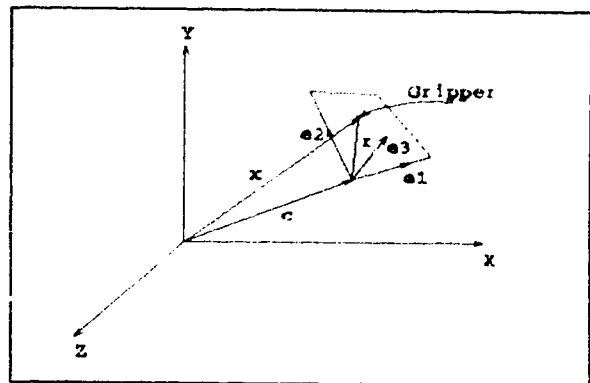


Figure 24 - Relative position of an object and the gripper

then this procedure is repeated for the diagonally opposite corner of the object. If again  $\bar{r}$  has positive coordinates it means that gripper is inside the object and can be picked up.

This procedure is very fast and there are no transcendental calculations involved. The only problem is that it can only be effective for rectangular prisms since its successful operation depends on the construction of orthogonal coordinate frames.

### Object dropping procedure

The program uses a simple algorithm to approximate the position and orientation of the object once the gripper opens its fingers. The position of the dropped object is calculated by first rotating the object to bring the bottom surface of the object to horizontal x-z plane, and then translating it directly under its rotated position (Figure 25).

To find the bottom surface of the grasped object the normal vector of each surface is dotted with the unit vector of the vertical direction. This is equivalent of calculating the B component of the  $N(A,B,C)$ , the normal vector to each surface. The closest to 1 (if normalized vectors are used) determines the bottom surface.

To minimize the required calculation

the program simply finds the surface with the largest B component.

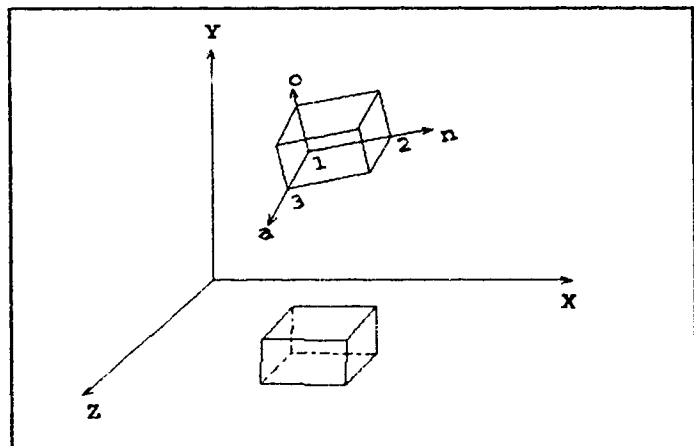


Figure 25 - Dropped object

The following transformation [T] positions the object under the current coordinates of the gripper (the projection of the gripper on the table).

$$[T] = [Tr1][R1][R2][Tr2]$$

where:

$$Tr1 = \begin{bmatrix} 1 & 0 & 0 & p_y \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & p_x \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This translation locates the object at its position on the table based on the coordinates of the gripper's projection on the table.

$$R1 = \begin{bmatrix} n_x & 0 & -n_z & 0 \\ 0 & 1 & 0 & 0 \\ n_z & 0 & n_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This rotation matrix is based on the projection of the  $\bar{n}$  vector on the horizontal plane x-z (Figure 25). This rotation determines the final orientation of the object on the table. The components of the n direction vector are normalized. The second column is the vertical direction and the third column is the cross product of the first and second columns.

The [R2] transform rotates the object to the horizontal plane. The rotation is found by first constructing an orthogonal frame based on the coordinates of three points on the bottom surface. The components of this frame represent a rotation. The inverse of this rotation is [R2]. The components of  $\bar{\rho}$  are found from the cross product of the other two coordinates. The coordinates of the three points on the bottom surface are passed to the function from the calling function.

$$R2 = \begin{bmatrix} x_2-x_1 & y_2-y_1 & z_2-z_1 & 0 \\ a_x & a_y & a_z & 0 \\ x_3-x_2 & y_3-y_2 & z_3-z_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} X_1 & X_2 & X_3 & 0 \\ Y_1 & Y_2 & Y_3 & 0 \\ Z_1 & Z_2 & Z_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This translation transfers the object to the base of the coordinate before the rotation is applied.

$$Tr2 = \begin{bmatrix} 1 & 0 & 0 & -p_y \\ 0 & 1 & 0 & -p_z \\ 0 & 0 & 1 & -p_x \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The final transform matrix [T] is shown here. The 'p' represents the position of the gripper. The  $\bar{n}, \bar{o}$  and  $\bar{a}$  vectors are the direction vectors shown in Figure 25.

$$T = \begin{bmatrix} n_x X_1 - n_z Z_1 & n_x X_2 - n_z Z_2 & n_x X_3 - n_z Z_3 & P_x^* \\ Y_1 & Y_2 & Y_3 & P_y^* \\ n_z X_1 + n_x Z_1 & n_z X_2 + n_x Z_2 & n_z X_3 + n_x Z_3 & P_z^* \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_x^* = -p_y(n_x X_1 - n_z Z_1) - p_z(n_x X_2 - n_z Z_2) - p_x(n_x X_3 - n_z Z_3) + p_y$$

$$P_y^* = -p_y Y_1 - p_z Y_2 - p_x Y_3$$

$$P_z^* = -p_y(n_z X_1 + n_x Z_1) - p_z(n_z X_2 + n_x Z_2) - p_x(n_z X_3 + n_x Z_3) + p_x$$

## **Animation**

The animation routine produces the simulated motion of the arm between the initial and final position of the arm whenever the position of the arm changes. The joint angles of the arm are stored in a global array. These values are accessed by the transformation routines. Once a move is requested, the new joint angles are calculated and animation routine is called. This routine keeps the values of the joint angles from the last animation session and the new joint angles are available in the global array. To find the appropriate number of frames between initial and final positions, the program calculates the largest angle change of the joint angles and multiplies this number by a parameter that controls the animation rate (fine or coarse steps in animation). The result is the number of frames between the initial and final position. If this number is less than one only the final position is drawn, otherwise the incremental value for each joint is calculated and for each step the old image is deleted and the new image is drawn.

Smooth animation requires a high framing rate. The framing rate depends on the amount of calculation required to generate a frame and the time required to update the graphics on the screen. To create a proper animation effect the old images are deleted and the new ones are drawn. It is possible to issue a clearscreen command to completely clear the screen but this command is slow. If the changes to each frame are limited to a small part of the screen or a few polygons then it is much faster to selectively delete the parts of the old image. This is done by redrawing the old images in the background color.

## **View camera transformations**

The arm can be viewed using two different view screens. The default camera views the arm and workcell from a fixed view point. The transformations for this mode are already explained. The second camera is a moving view point attached to the gripper. Simulation of this case requires an additional transformation of the arm and workcell to the coordinates of the gripper. From calculations for the default camera all the data points

(coordinates of the objects) are positioned properly and available to the new view camera. To achieve the desired view all these points have to be transferred to (viewed from) the gripper coordinate. Hence the  $\bar{n}\bar{o}$  plane of the gripper coordinate is now the view plane for the screen perspective projection and the view point resides on the direction of the  $\bar{a}$  axis.

The  $[T_6]$  transformation can transform any point to the position and orientation of the gripper. Thus the  $[T_6]^{-1}$  matrix can perform the necessary transformations for the new view screen. For the final perspective projection two additional transforms are required. One is to find the mirror image of the transformed objects with respect to the view plane. This is required because the camera is looking to the world in the  $\bar{a}$  direction of the gripper. Without this transformation camera would see the world and arm as if looking into the gripper and not from the gripper. The next transformation is a 90 degree rotation that corrects the alignment of the transformed objects for viewing purposes. This transformation is applied to object coordinates expressed in screen coordinate frames.

$$[T_{grripper}] = [\text{Rot } z, 90^\circ][\text{mirror about x-y plane}][T_6]^{-1}$$

$$[T_{grripper}] = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_y & n_z & n_x & -n.p \\ o_y & o_z & o_x & -o.p \\ a_y & a_z & a_x & -a.p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{grripper}] = \begin{bmatrix} -o_y & -o_z & -o_x & n.p \\ -n_y & -n_z & -n_x & a.p \\ -a_y & -a_z & -a_x & o.p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## **Relative moves**

### **Moves relative to world coordinate**

The term relative signifies that moves are performed relative to the current position or orientation of the gripper. The world coordinate is the orthogonal coordinate at the base of the arm. For convenience the coordinate frame is positioned at the workbench level (grid level). In this mode the gripper can move in one of the principle directions of the world coordinate where its orientation and other coordinates remain constant.

The program reads its input from the keyboard and increments the position based on the specific key entry. This new position is the input for the inverse kinematic solution and thus the new configuration of the arm is calculated.

Also, this mode allows changing the orientation of the gripper in the same way it alters the position of the gripper. The orientation change is signalled through keyboard and the new value of one of the principle angles (pitch, yaw or roll) defining the orientation of the arm is altered respectively. The angles and the current position of the gripper generate the new  $T_6$  matrix. The inverse kinematic solution recalculates the necessary joint angles for the new configuration.

### **Moves relative to gripper**

Moves relative to gripper are similar to the moves relative to world. The only difference is the coordinate frame used as reference for the moves. As the name implies, the gripper coordinate is the reference coordinate. The availability of the  $T_6$  matrix enables easy calculation of the new position in the world coordinate. Once the required moves are found in the world coordinate the rest of the routine behaves exactly the same way it does in the relative moves in world coordinates.



The  $T_6$  provides the directional vectors for the three principle directions of the gripper coordinate  $\bar{n} \ \bar{o} \ \bar{a}$ . Using these directions the position of the gripper can be altered in one of the principle directions of the gripper. Once a move is requested the increment in the requested direction is calculated by multiplying the unit vector representing that direction with the requested move step size. The result is added to the current position of the gripper in the world coordinate. The rest is as explained for the relative to world moves.

### **Menu**

The menu function generates a pop-up menu and enables the user to choose a menu item either by entering the highlighted word of the menu items or by moving the menu bar and selecting. Most menu libraries are developed for menus in the text mode screens. In the *EXCAL* program, the screen is in graphics mode as soon as the program is loaded.

The menu function saves the content of the screen where it pops up. The content of the screen is saved to the second graphics page memory. The second page is not used and its memory is always available. The menu function can pose compatibility problems if the graphics display is incapable of supplying two pages of graphics screen. It is possible to overcome this problem using the dynamic memory allocation techniques. However, the limited availability of large contiguous memory prevents effective implementation of this method.

## **Appendix C**

### ***EXCAL* program verification/testing data sheet**

The following table is compiled based on the "Human Factors Engineering" concepts, applicable to software design for engineering systems. This table was used during test procedures and users were asked to comment on their experience with the program. The obtained information was then used to improve the program as discussed in chapter 6.

The information in this questionnaire is for statistical analysis and evaluation of the software program "EXCAL" for the robotic arm Excalibur.

Test User Information

Name:.....

Education:.....

Familiarity with computers:.....

Familiarity with robotic systems:.....

Time spent learning the program:.....

Level of familiarity achieved:.....

What is your opinion about the view capabilities of the program?

What do you think about the on-line controlling capabilities of the program?

What do you think about off-line capabilities of the program?

What would you suggest to improve the program?

In completing the questionnaire use appropriate wording to describe your acceptance or dislike of each field. Suggested words are:

Poor --- did not like, disapprove

Fair --- acceptable

Good --- liked the usage, feature

Very good --- completely approving the method

<i>EXCAL</i> program verification/testing sheet Human-computer interface		
Case	Rating	Additional comment
<b>Reserved Display Areas:</b> Evaluate the display locations and the consistent use of fixed display locations.		
Fixed fields		
Consistent conventions		
consistent use of terms		
<b>Alphabetic Data:</b> Evaluate ease of understanding and also readability of data.		
Justification		
Data grouping		
Clarity of text		
Essentiality of information		
<b>Numeric Data</b>		
Decimal numbers		
Data order		
Location		
Can user differentiate between data and instruction?		

<i>EXCAL</i> program verification/testing sheet		
Human-computer interface		
Case	Rating	Additional comment
Units		
Clutter		
Spacing		
<b>Highlighting</b>		
Use of highlighting		
Brightness highlighting		
Two color highlighting		
<b>Effective wording</b>		
Abbreviation consistency		
Clarity of abbreviation		
Length of abbreviation		
Familiar wording		
Consistent wording		
<b>Color</b>		
Use of color, overuse?		
Color as highlight		
Color for status		

<b>EXCAL program verification/testing sheet</b>		
<b>Human-computer interface</b>		
<b>Case</b>	<b>Rating</b>	<b>Additional comment</b>
Color coding (error, warning)		
consistency of color coding		
Color visibility		
Background color		
Blinking		
<b>Color Graphics</b>		
Coloring		
Consistent coloring		
Color changes		
<b>Graphics and Animation</b>		
Display density		
Multi_window screen		
Visual interface		
<b>Information and status display</b>		
Intermediate feedback		
Input acknowledgment		

<b>EXCAL</b> program verification/testing sheet		
Human-computer interface		
Case	Rating	Additional comment
Inactive screen		
Mode designator		
System messages		
Error messages		
Warning messages		
Visibility		
<b>Menus</b>		
Main menu accessibility		
Menu bypass		
Menu wording		
Consistent titles		
Menu order		
Inactive menu options		
<b>Commands</b>		
Command wording		
Abbreviated commands		
Command consistency		



<b>EXCAL program verification/testing sheet</b>		
<b>Human-computer interface</b>		
<b>Case</b>	<b>Rating</b>	<b>Additional comment</b>
Compatible commands		
Distinctive commands		
Command clarity		
<b>System response time</b>		
Delay in responding an action		
Interactive delay		
File delay		
<b>Control and Input Devices</b>		
Function key setup		
Cursor key setup		
Alt key setup		
Control key setup		
Shift key setup		
Original key setup		
Consistency of key use		
Common key usage		

<b>EXCAL program verification/testing sheet</b>		
Human-computer interface		
Case	Rating	Additional comment
Programmable keys		
Master arm availability		
Preferences		
<b>Error Messages and On-line Assistance</b>		
Program response		
Error recovery		
Error message display		
Old message removal		
Auditory error signals		
Usefulness of error messages		
Phrasing of messages		
Length of messages		
Message content		
Help on error		
Effect of error on system		
System protection on error		

<i>EXCAL</i> program verification/testing sheet		
Human-computer interface		
Case	Rating	Additional comment
<b>On-line Guidance</b>		
On-line assistance		
Help key		
Help content		
Help on commands		
Help in context		
Concise help		