CROBOTS

A CAD BASED ROBOT SIMULATION TOOL

JOHN JOSEPH O'LEARY

CROBOTS
A CAD Based Robot Simulation Tool

by

© John Joseph O'Leary, B.Eng.

A THESIS SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING

Faculty of Engineering and Applied Science
Memorial University of Newfoundland
St. John's, Newfoundland

August 1998

This thesis is dedicated to
my wife
Charlotte
whose persistence and patience
helped me achieve this goal.

Abstract

This thesis proposes a new CAD based tool for robot simulation (CROBOTS) that can be used in the design, application, and programming of educational and industrial robots. The software is written using the AutoLisp programming language and runs as a third party application inside AutoCAD R14. CROBOTS combines the impressive graphics capability of AutoCAD with custom developed tools for revolute robot model creation, robot operation cycle planning, and robot controller simulation. Custom functions enable the automatic creation of a 3D solid model representation of robot geometry given dimensions, the teaching of point positions defining a desired robot trajectory, and the 3D graphical simulation of the manipulator's response to computed torque plus PID generated drive torques.

CROBOTS should prove an effective tool in the delivery of robotics related curriculum at both secondary and post-secondary educational institutions. The time required to become proficient in the use of CROBOTS should be less than that required to learn stand alone commercial software packages for robot simulation. AutoCAD's inquiry tools, for example, may be used to facilitate the teaching of robot design concepts. Furthermore, CROBOTS can assist with the development of robot programs offline and should allow for more efficient use of limited robot hardware resources at educational institutions.

Future developments of CROBOTS should allow modeling of additional robot mechanical configurations, the recording of significant production data, and a task level programming interface. CROBOTS should also be tested in both industrial and educational settings to determine its robustness and to identify needs for additional functionality.

Acknowledgement

I extend my sincere appreciation to Dr. Michael Hinchey, my supervisor, for "knowing everything about robotics and control" and, in particular, for his understanding of how challenging it is to achieve this goal when employed full-time.

Nomenclature

(p_x, p_y, p_z)	Tool center point position with respect to the robot base.
α	Robot base rotation angle [rad].
β	Robot shoulder rotation angle [rad].
ϵ	Robot elbow rotation angle [rad].
κ	$\beta + \epsilon$ [rad].
E	Robot base height [m].
G	Length of link 1 [m].
H	Length of link 2 [m].
(p_{xo}, p_{yo}, p_{zo})	Wrist position with respect to the robot base.
P	End effector pitch angle [rad].
R	End effector roll angle [rad].
Y	End effector yaw angle [rad].
a	End effector approach vector.
o	End effector orientation vector.
n	End effector normal vector
$\dot{p}$	End effector rate [rad/s]
$\dot{q}$	Joint rate [rad/s].
δW	Virtual work [J].
δq	Virtual displacement [m].
τ_i	Joint load [N.m].

q_i	Joint displacement [rad]
T	Total kinetic energy [J].
V	Total potential energy [J].
L	Lagrangian, T – V [J].
M	Mass [kg]
ϕ	Base torque [N.m]
φ	Shoulder torque [N.m]
ω	Elbow torque [N.m]
M(q)	Inertia matrix
$V(q,\dot{q})$	Velocity torque matrix
G(q)	Gravity torque matrix
F(q)	Friction torque matrix
τ_d	Disturbance torque [N.m]
e(t)	Joint tracking error [rad]
K_p	Proportional gain [N.m\rad]
K_v	Derivative gain [N.m.s\rad
K_i	Integral gain [N.m.\rad.s]

Table of Contents

List of Illustrations

Chapter 1

1.0 Introduction

In an effort to remain competitive in a dynamic global marketplace, today's manufacturers are challenged to reduce the lead-time and cost required to bring a product from the concept stage to the consumer. Flexible manufacturing systems (FMS) provide a number of advantages over traditional labor intensive techniques which may help meet this need. FMS can result in increased productivity, shorter production time for new products, reduction of inventory parts in the plant, savings in labor cost, and improved product quality (Biekert, R. et al ,1991).

Industrial robots represent a major component in FMS due to their capability to be used for a number of different applications. The main applications of robots include parts handling, assembly, machine loading and unloading, spray painting, welding, and inspection (Ryan, D., 1994). The world's industrial-robot population was estimated at 570,000 units at the end of 1992. Japan accounted for about 60% of the world's stock operational in 1992. The worldwide stock of industrial robots in 1992 increased only by 8%, compared with 16% in 1991. Robot orders, however, jumped by 40% in the record-setting first half of 1993, lead by the auto industry in spot-welding,

coating and material handling. About 48,000 robots are at work in American factories, the second largest robot user behind Japan. But Japan installs more robots each year than the total that the U.S. has installed in the past 32 years. It is estimated that the number of industrial robots operating in Japan will reach 880,000 units by the year 2000 (National Robot Society, 1994).

The expected growth in industrial demand for robotics technology will invariably lead to a need for more efficient engineering design practices, increased productivity in robot programming, and improved resources in support of research, development, education and training. This thesis presents a new (C)AD based tool for (Robot) (S)imulation or CROBOTS that has been designed to help meet these needs.

A brief review of robotics fundamentals is provided beginning with a discussion on industrial robot configurations. Secondly, conventional programming techniques are reviewed and the potential advantages of offline programming are discussed. A detailed overview of the requirements for mathematical modeling of robotic systems follows. This section includes development of the direct kinematics, inverse kinematics, and dynamics models for a revolute manipulator. The application of these models in the development of numerical simulations for robots is then provided. The capabilities of the CROBOTS software are then highlighted and a discussion on the development of CROBOTS is presented. This is followed by an overview of the effectiveness of using this software in robot design and motion planning. Finally, recommendations for the future enhancement of the CROBOTS software are presented.

1.1 Industrial Robot Programming Techniques

The reduced time for product changeovers afforded by FMS is due primarily to the capability to pre-program robots to perform a variety of different tasks. There are three primary forms of robot programming now being used in industry: (1) lead-through programming, (2) teach pendant programming, and (3) offline programming (Greenwood, F., 1989). In lead-through programming the operator sets the robot controller to programming mode, grasps the manipulator, and leads it through the various point moves which make up the operation cycle. After the operator pushes a button, the controller records the positions to which the end effector is moved. During playback the controller is able to duplicate the defined robot moves. Alternatively, a hand-held teach pendant which enables real time programming of the robot is used. Using the pendant, the operator is able to jog each robot axes to position the end effector where desired. After the operator pushes a button on the pendant, the points are recorded by the controller. The teach pendant also enables the operator to develop the overall robot program which, in addition to point moves, may include instructions to control gripper actuation, conditional logic, instructions for communication with external sensors and other machines through input/output (I/O) modules, and most other commands included with the robot control programming software. Finally, offline programming enables development of the preliminary robot control program on a PC or workstation without the requirement to take the robot out of service. During on-line programming using lead through or teach pendant methods, the robot is not available for production and manufacturers must incur the cost of this downtime.

Industrial robots generally provide the operator with the capability to develop programs offline using an ASCII text editor and then download the program to the robot controller. In addition, proprietary software is available

from commercial robot suppliers which enables debugging of the ASCII form of the program before download. These programs, however, typically do not provide a 3D graphical simulation of the operation cycle and the operator is, therefore, still required to test the robot interaction with its environment online. Fortunately, there are a number of third party commercial software packages available which allow offline programming and 3D graphical simulation of the robot operation cycle. The capital cost of these software programs and supporting hardware may in some cases exceed the cost of robot hardware and is, therefore, difficult to economically or financially justify for industrial use. Educational and R&D institutions have even more limited budget funds and purchase of such commercial software is generally not feasible.

1.2 Commercial Offline Programming Software

There are a wide variety of 3D CAD based commercial offline programming software packages currently available in the marketplace. Workspace (version 3.2), an industrial robot simulation software package, is a graphic simulation system and a means of off-line programming a robot cell. The software package will create and simulate robot programs in the native language of the robot. For example, users of Fanuc robots may write robot programs in Karel, and ABB robot users may write robot programs in ARLA. There is therefore no need for postprocessors to translate from a simulation language to the robot language: the full power of the robot language is available to the user through off-line programming. It is also possible to transfer existing robot programs from the robot controller back into Workspace for optimization, so off-line programming is a two-way process. The full structure of the robot languages is implemented, including typed variables, teachpoints, subroutines, looping, branching on condition, signals, and condition handler interrupts. All the main industrial and

educational robot languages are implemented, and a library of over 140 robot models is available to the user (though it is also possible for users to create their own robots).

Robotica is a collection of robotics problem solving functions for the Mathematica software package. It has the capability of reading external simulation (e.g., SIMNON) output files and displaying the motion of the robot when subjected to the sequence of joint variables. It requires Mathematica and X-windows.

Deneb Inc. offers a number of software packages for offline robotics programming and simulation including UltraArc - The Simulation and Programming Tool for Robotic Arc Welding, UltraFinish - The Simulation and Analysis Tool for Robot Deburring, Grinding, Polishing and Buffing, UltraPaint - The Simulation and Analysis Tool for Robotic Painting, and UltraSpot - The Simulation and Programming Tool for Spot Welding.

Working Model 3D is a PC based software package which allows kinematic and dynamic simulation of a variety of mechanical systems. The program is not specifically designed for robotics applications but it does provide the utilities necessary to build and simulate manipulator operation.

In each of these cases, the cost of the software, required hardware, and related training may be difficult to justify for industry and educational customers who have already dedicated extensive in-house resources to CAD software such as AutoCAD.

1.3 Related Previous Work

Development of the CROBOTS software required consideration of a wide array of topics including robotics design fundamentals, direct kinematics, inverse kinematics, dynamics modeling, inverse dynamics, numerical controller simulation, offline robot programming, and robotics related CAD applications. Several previous research efforts related to these individual topic areas were identified and sourced in the development of this thesis.

Development of the direct and inverse kinematics model has been well documented by a number of sources. The basic parameters used to describe a manipulator were first presented by Denavit and Hartenberg (1955) and further detailed by Paul (1981). Pieper (1968) presented a closed form solution for the inverse kinematics for simple manipulators which Paul (1981) later expanded and applied to industrial robots such as the Unimate PUMA.

Development of control schemes have been numerous as well. Whitney (1969) proposed the use of resolved motion rate control (RMRC) for robots. Given a desired path of the tool endpoint in cartesian co-ordinates, RRMC can be designed by relating the joint rates to the tool endpoint rate through the manipulator Jacobian. RRMC, however, gives rise to an accumulation in tracking errors in position since position is established indirectly by taking an integral over a specified time interval. The Computed Torque model was originally proposed by Bejczy (1974) and again by Vukobratovic (1982). Unfortunately, the Computed Torque varies from the actual torque due to inaccuracies in the dynamics model resulting from unknown friction, payload and inertia parameters. Control schemes based on linearization of Lagrange's equations by non-linear feedback were proposed by Freund (1982). This

scheme encountered problems in discrepencies between required and calculated torques similar to the Computed Torque method. Takegaki and Arimoto (1981) showed that Proportional plus Derivative (PD) controllers can provide stable manipulator control if the gravity term can be carefully compensated for using counterweights. Alternatively, an integral control term resulting in a PID scheme may be used to compensate for the gravity term. This scheme still assumes knowledge of manipulator dynamics, which invariably are inaccurately modeled.

The use of CAD to aid in the design and programming of robots has been the subject of several research efforts. Wu (1984) proposed a mathematical formulation that could be used to build a CAD tool which would improve the accuracy of kinematic models and the development of offline programs. In this case, calibration of the actual robot hardware is still necessary online since the geometry of the manufactured components differs from the CAD model. The robotics facilities of the CAD/CAM CATIA system were proposed by Borrel, P. et al (1982). This program was developed to run on computer workstations. Hornick and Ravani (1986) proposed a CAD tool called STAR (Simulation Tool for Automation and Robotics) for offline robot motion planning and programming. This program was developed to run on computer workstations and relied on separate programs for the geometric model and the dynamics model. Dutt (1991) utilized AutoCAD to generate geometrical information required for offline robot programming. In this case, BASIC programming was used to extract geometrical information characterizing the manipulator position from the Drawing Interchange Format (DXF) file output of AutoCAD. This could only be done one position at a time. Ryan (1994) proposed the use of AutoCAD for simulation of robotic and automated systems. AutoCAD is not customized in this case and essentially this proposal is an overview of native AutoCAD commands that may be used for a static analysis of position geometry.

Chapter 2

2.0 Development of the Robot Model

The Robot Institute of America defines a robot as:

"A robot is a reprogrammable, multifunctional manipulator designed to move materials, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks."

As this definition implies, robots are available in a wide variety of mechanical configurations but, in general, may be classified in one of five different classifications. These include cartesian, cylindrical, spherical, revolute and SCARA (Critchlow, A., 1985).

2.1 Robot Classifications

Cartesian Configuration

This configuration characterizes robots that have linear or prismatic motion capability that can be measured in the familiar XYZ cartesian co-ordinates as shown in Figure 1. The manipulator can be moved linearly up or down the vertical Z axis and positioned in the horizontal plane through linear

motion along both the X and Y-axes. These robots are easiest to program because of the complete independence of their joints and there is no coupling between all axes for a rigid and frictionless structure. This motion defines a rectangular workspace that is suitable for relatively simple applications such as machine loading/unloading and inspection operations.

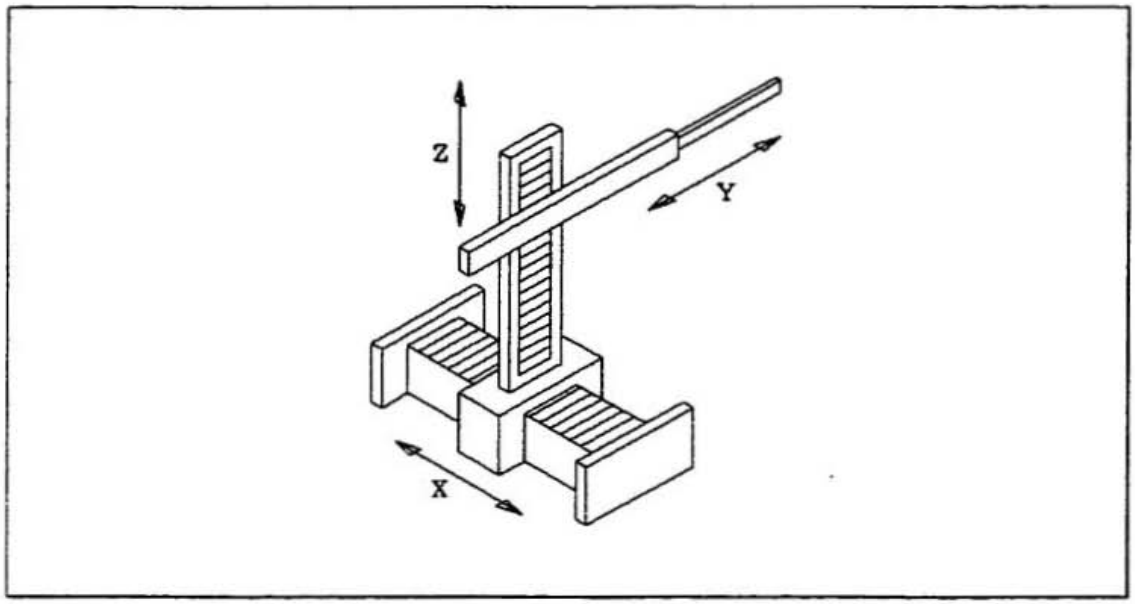

Figure 1: Cartesian Configuration

Cylindrical Configuration

This configuration applies to robots that combine vertical and radial prismatic motion capability with the ability to rotate about the vertical axis as shown in figure 2. These combined motions define a cylindrical workspace volume. The cylindrical configuration is more versatile than the rectangular configuration and it is used for a variety of production applications. Since there is telescopic radial motion, these units require relatively small space and reduced dynamics since initial rotation can be executed with the manipulator retracted. This configuration is well suited to the majority of pick and place operations. One disadvantage of the cylindrical configuration

is that the manipulator cannot reach below the bed of the structure. Redundant degrees of freedom at the wrist may be used to overcome this.

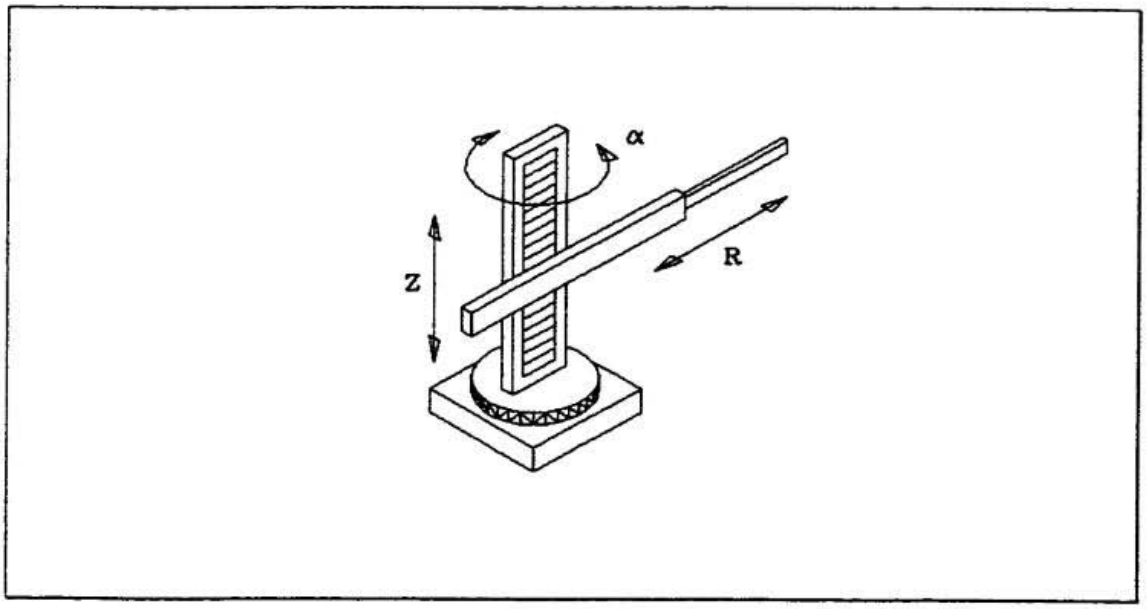

Figure 2: Cylindrical Configuration

Spherical Configuration

The spherical configuration retains the telescopic prismatic radial motion and the capability for rotation about the vertical axis of the cylindrical configuration. In addition it employs the ability to rotate about a horizontal axis through the base as shown in figure 3. The resulting motion capability defines a hemispherical workspace bounded by an inner and outer hemisphere. This configuration requires more sophisticated control systems but it is able to service a larger workspace.

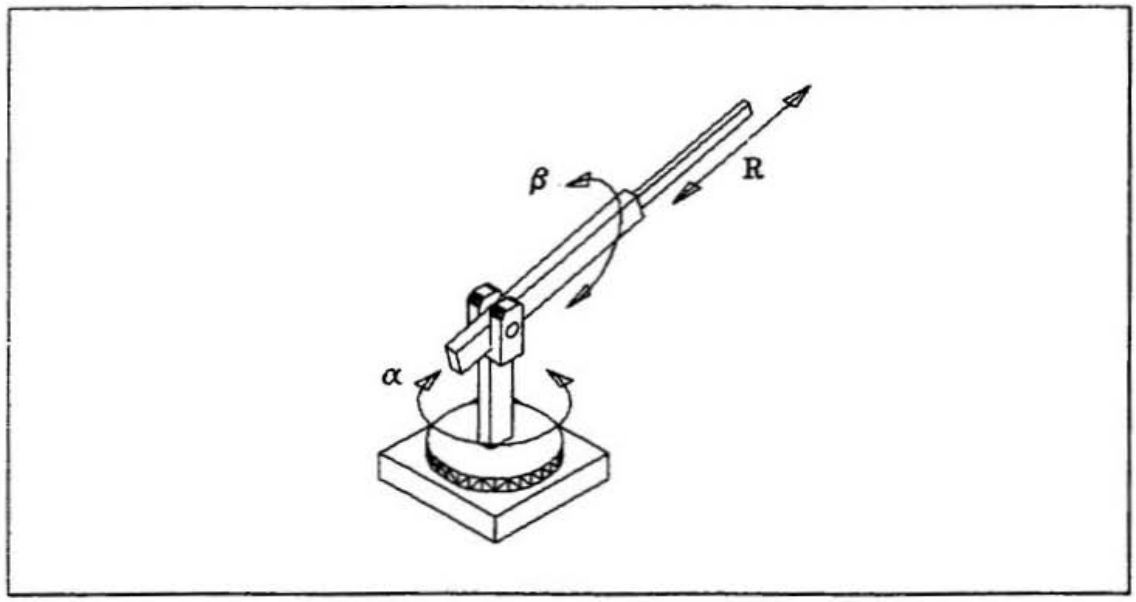

Figure 3: Spherical Configuration

Revolute Configuration

The revolute manipulator employs rotary motion capability about the base, the shoulder and the elbow as shown in figure 4. Three additional rotational degrees of freedom are normally added at the wrist resulting in a 6-axis manipulator for most industrial units. This configuration provides the most dexterity but also demands the most sophisticated controllers. Their small size relative to their workspace capability, ease of installation, high reliability, and ability to easily work in enclosed spaces make them well suited for a variety of industrial applications. A disadvantage of this configuration is the degenerating behavior of the manipulator near the workspace boundaries. In addition, because the shoulder and elbow axes are parallel and orthogonal to the waist axis, these units have relatively low stiffness. This makes them unsuited for many high precision applications.

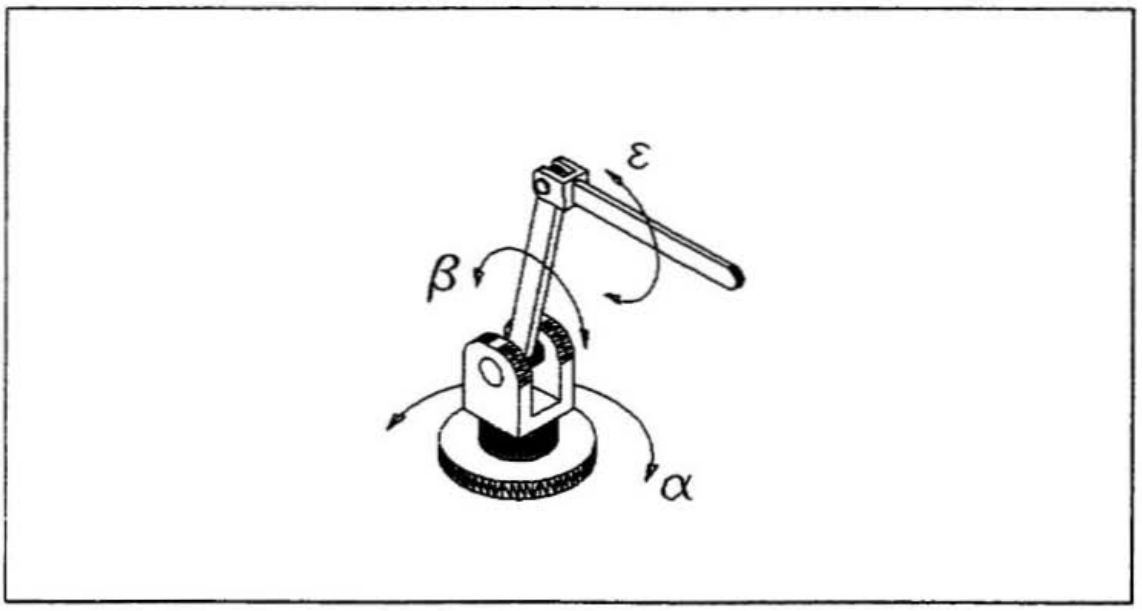

Figure 4: Revolute Configuration

SCARA Configuration

The SCARA (Selective Compliance Assembly Robot Arm), which is specifically designed for high precision assembly and drilling operations, overcomes the low stiffness problem of the revolute configuration but offers limited mobility. In this configuration shown in figure 5, the shoulder and elbow axes are parallel to the waist axis. The SCARA has four degrees of freedom: limited rotations about the shoulder and elbow to position the tool, rotation about the base to locate the shoulder and elbow workspaces, and vertical prismatic motion of the end effector. This motion capability makes it well suited for the majority of assembly operations in industry.

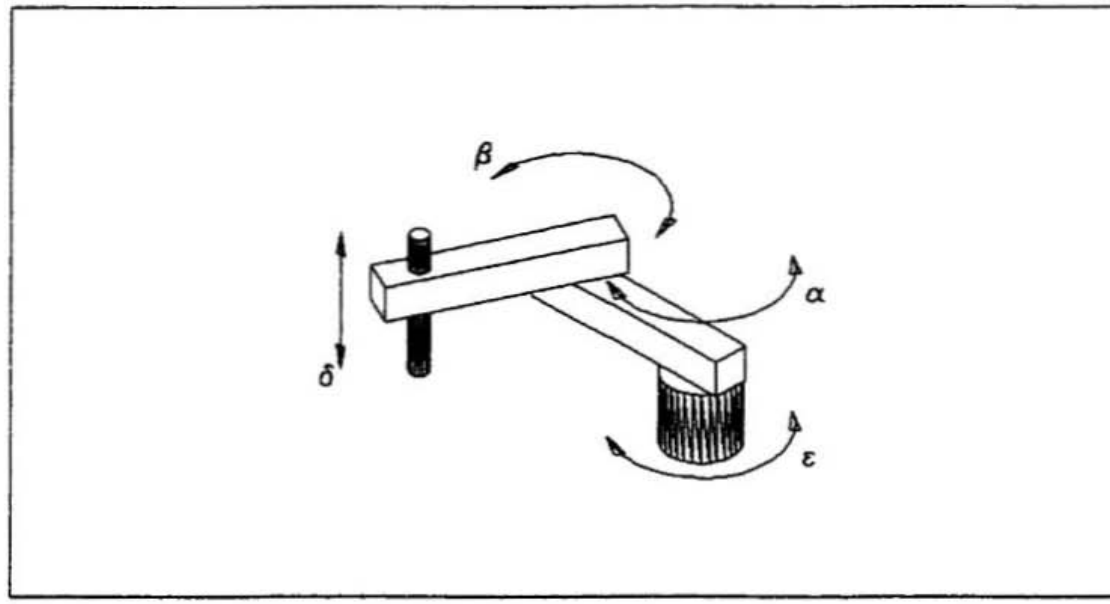

Figure 5: SCARA Configuration

2.2 Mathematical Modeling

Industrial robots are basically positioning and handling devices. To effectively complete typical operations such as welding, the robot must be able to control its motion, at a minimum, and, in many cases, the forces it applies to its environment. Control of the end effector demands an accurate analysis of the characteristics of the mechanical structure, actuators and sensors. Mathematical modeling of a robot manipulator is, therefore, a necessary pre-requisite to developing a successful controller (Canudas de Wit, 1996). Modeling of robot manipulators requires consideration of both kinematics and dynamics.

2.3 Kinematics

Kinematic modeling concerns the description of the manipulator motion with respect to a reference frame without consideration of the forces and torques that cause the motion of the structure. This formulation of the kinematic relationship allows study of both direct kinematics and inverse kinematics. Direct kinematics enables the description of the end effector motion as a function of joint motion. Inverse kinematics consists of transforming the desired end effector motion in the workspace into the corresponding joint motion.

2.3.1 Direct Kinematics

A serial link manipulator consists of a sequence of links connected together by actuated joints. For an n degree of freedom manipulator, there will be n links and n joints. The relationship between links of a two link articulated robot can be developed after assigning co-ordinate frames to each link as shown in figure 6. The tool center point (TCP) can be measured locally with respect to: (1) moving co-ordinate frame uvw which has its origin at the wrist, (2) moving co-ordinate frame nmh which has its origin at the elbow, (3) moving co-ordinate frame pqr which has its origin at the shoulder, (4) moving co-ordinate frame xyz which has its origin at the base, or (5) fixed world co-ordinate frame XYZ which also has its origin at the base. The direct kinematics (DK) solution for this case allows the determination of the Cartesian position and orientation of the end effector when given the joint co-ordinates.

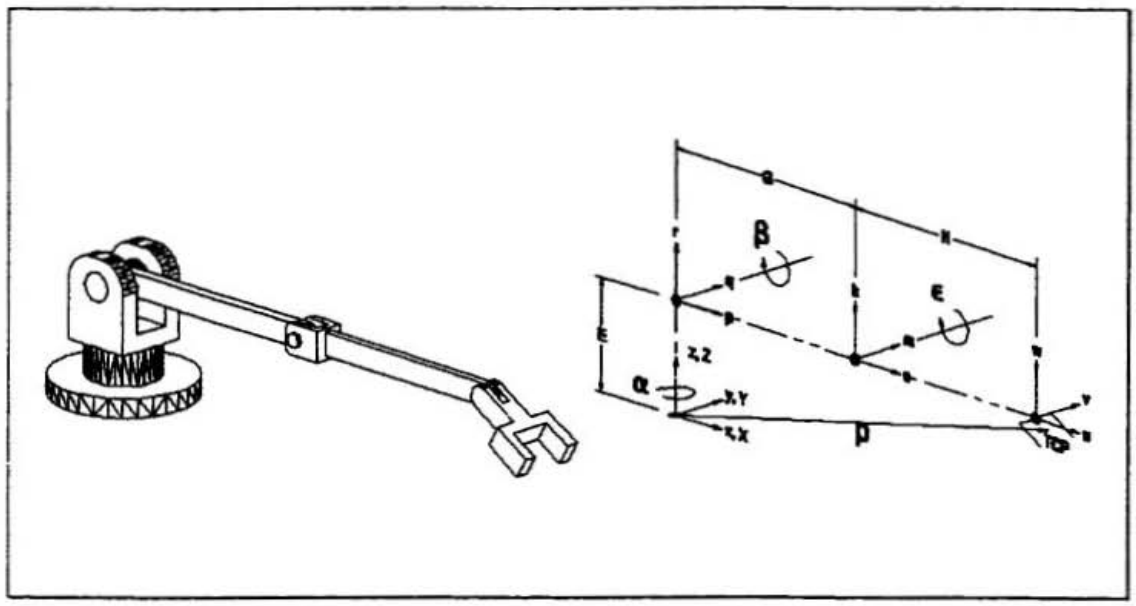

Figure 6: Co-ordinate Frames for a Two Link Revolute Robot

For a locked wrist case, the end effector tool center point (TCP) to base transformation is (Hinchey, M.J., 1994):

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} C\alpha & -S\alpha & 0 & 0 \\ S\alpha & C\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\beta & 0 & S\beta & 0 \\ 0 & 1 & 0 & 0 \\ -S\beta & 0 & C\beta & G \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\varepsilon & 0 & S\varepsilon & 0 \\ 0 & 1 & 0 & 0 \\ -S\varepsilon & 0 & C\varepsilon & H \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ E+w \\ 1 \end{bmatrix} \quad (1)$$

where (p_x, p_y, p_z) is the cartesian position of the TCP with respect to the base frame X,Y,Z in [m].

(u, v, w) is the cartesian position of the TCP with respect to the wrist frame uvw in [m].

E is the base height in [m].

G is the length of link 1 in [m].

H is the length of link 2 in [m].

$C\alpha$ and similar notations are used to abbreviate cosine/sine of respective joint angles in [rad].

At the wrist, u = v = w = 0 and the wrist to base transformation is therefore:

$$\begin{bmatrix} p_{xo} \\ p_{yo} \\ p_{zo} \\ 1 \end{bmatrix} = \begin{bmatrix} C\alpha & -S\alpha & 0 & 0 \\ S\alpha & C\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\beta & 0 & S\beta & 0 \\ 0 & 1 & 0 & 0 \\ -S\beta & 0 & C\beta & G \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S\epsilon E \\ 0 \\ H + C\epsilon E \\ 1 \end{bmatrix} \tag{2}$$

where (p_{xo}, p_{yo}, p_{zo}) represents the position of the wrist or origin of the end effector frame with respect to the base reference frame.

Matrix multiplication yields:

$$\begin{bmatrix} p_{xo} \\ p_{yo} \\ p_{zo} \\ 1 \end{bmatrix} = \begin{bmatrix} C\alpha & -S\alpha & 0 & 0 \\ S\alpha & C\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\beta S\epsilon E + S\beta(H + C\epsilon E) \\ 0 \\ G - S\beta S\epsilon E + C\beta(H + C\epsilon E) \\ 1 \end{bmatrix} \tag{3}$$

From which it follows:

$$p_{xo} = C\alpha(C\beta S\epsilon E + S\beta(H + C\epsilon E)) \tag{4}$$

$$p_{yo} = S\alpha(C\beta S\epsilon E + S\beta(H + C\epsilon E)) \tag{5}$$

$$p_{zo} = G - S\beta S\epsilon E + C\beta(H + C\epsilon E)) \tag{6}$$

Letting $\kappa = \beta + \epsilon$, these equations reduce to:

$$p_{xo} = C\alpha(S\kappa E + S\beta H) \tag{7}$$

$$p_{yo} = S\alpha(S\kappa E + S\beta H) \tag{8}$$

$$p_{zo} = G + C\beta H + C\kappa E \tag{9}$$

It should be noted that inspection of geometry yields the same equations. For a two link revolute robot with a Roll Pitch Yaw (RPY) wrist for end effector orientation, the tool center point (TCP) to base transformation is:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} C\alpha & -S\alpha & 0 & 0 \\ S\alpha & C\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\beta & 0 & S\beta & 0 \\ 0 & 1 & 0 & 0 \\ -S\beta & 0 & C\beta & G \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Ce & 0 & Se & 0 \\ 0 & 1 & 0 & 0 \\ -Se & 0 & Ce & H \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & E \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} CPCY & -CPSY & SP & 0 \\ CRSY + SRSPCY & CRCY - SRSPSY & -SRCP & 0 \\ SRSY - CRSPCY & SRCY + CRSPSY & CRCP & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} \tag{10}$$

where P is the pitch angle in [rad].
Y is the yaw angle in [rad].
R is the roll angle in [rad].

Multiplication gives an equation of the form:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_{xo} \\ n_y & o_y & a_y & p_{yo} \\ n_z & o_z & a_z & p_{zo} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} \tag{11}$$

The TCP of the end effector may, therefore, be expressed in terms of base co-ordinates through a rotation matrix $\mathbf{R} = [\mathbf{n}\ \mathbf{o}\ \mathbf{a}]$ representing the orientation of the end effector and a position vector $\mathbf{p}_o$ representing the position of the wrist. These values are denoted collectively as the arm **T** matrix. As shown in figure 7, the approach vector **a** represents the reach direction. The orientation vector **o** represents the direction specifying the orientation of the end effector from fingertip to fingertip. The normal vector **n** is chosen to complete the definition of a right-handed co-ordinate system.

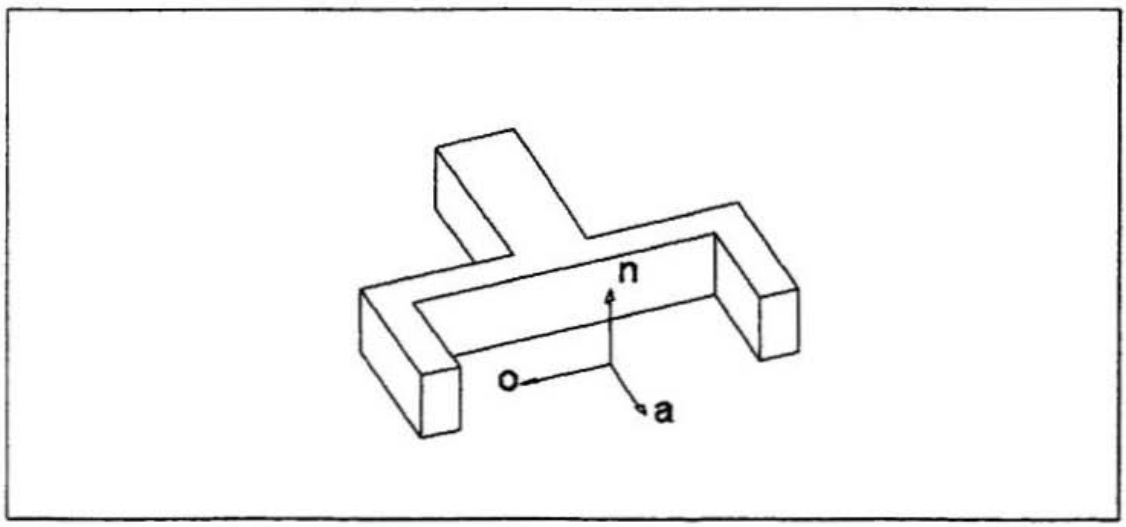

Figure 7: Robot End Effector Orientation Vectors

2.3.2 Inverse Kinematics

The DK solution allows the determination of the TCP position with respect to the base frame given the wrist position and the end effector orientation. While this solution is an important part of mathematical modeling of robots, the inverse kinematics (IK) solution offers more practical information that can be used in the development of controller strategies. The IK solution for the revolute robot is aimed at determining the joint angles

necessary to produce a given position and orientation of the end effector. The IK solution for the wrist of the revolute robot, previously shown in Figure 1, allows the determination of the joint angles α, β, and ϵ corresponding to a given (p_{xo}, p_{yo}, p_{zo}) location.

Manipulation of equations (4) and (5) yields:

$$p_{xo}S\alpha = S\alpha C\alpha(S\kappa E + S\beta H) \tag{12}$$

$$p_{yo}C\alpha = C\alpha S\alpha(S\kappa E + S\beta H) \tag{13}$$

Subtracting equation (13) from (12) gives:

$$p_{yo}C\alpha - p_{xo}S\alpha = 0 \tag{14}$$

Solving for α implies:

$$\alpha = \text{atan2}(p_{yo}/p_{xo}) \text{ and } \alpha = \text{atan2}(-p_{yo}/-p_{xo}) \tag{15}$$

where atan2 is a computer representation of atan which accounts for the fact that two possible quadrants provide the same solution for α.

Additional manipulation of equations (12) and (13) yields:

$$p_{xo}C\alpha = C\alpha C\alpha(S\kappa E + S\beta H) \tag{16}$$

$$p_{yo}S\alpha = S\alpha S\alpha(S\kappa E + S\beta H) \tag{17}$$

Summation gives:

$$p_{xo}C\alpha + p_{yo}S\alpha = S\kappa E + S\beta H \tag{18}$$

Manipulation of this equation yields:

$$S\kappa = (p_{xo}C\alpha + p_{yo}S\alpha - S\beta H)/E = P + RS\beta \tag{19}$$

Manipulation of equation (9) gives:

$$C\kappa = (p_{zo} - G - C\beta H)/E = Q + RC\beta \tag{20}$$

Squaring both sides of equations (19) and (20) yields:

$$S\kappa S\kappa + C\kappa C\kappa = (P + RS\beta)^2 + (Q + RC\beta)^2 = 1 \tag{21}$$

Expanding this gives an equation of the form:

$$IC\beta + JS\beta = K \tag{22}$$

General analytical inverse kinematics formula can be used to show that:

$$\beta = a\tan 2(J/I) + a\tan 2(\pm\sqrt{I^2 + J^2 - K^2}/K \tag{23}$$

The P, Q, R equations are of the form:

$S\kappa = V \qquad \text{and} \quad C\kappa = W$

where with α and β known, V and W are also known. V and W can then be used to determine:

$$\kappa = \text{atan2}(V/W) \tag{24}$$

$$\epsilon = \kappa - \beta \tag{25}$$

Examination of equations (15), (23), (24) and (25) reveals that the IK solution is not unique. There are two possible solutions for α. There are in turn, for each α, two possible solutions for β and ϵ. Thus, there may be four admissible solutions obtained for the wrist position according to the values of α, β, and ϵ. These include two elbow-up positions and two elbow-down positions as illustrated in figure 8.

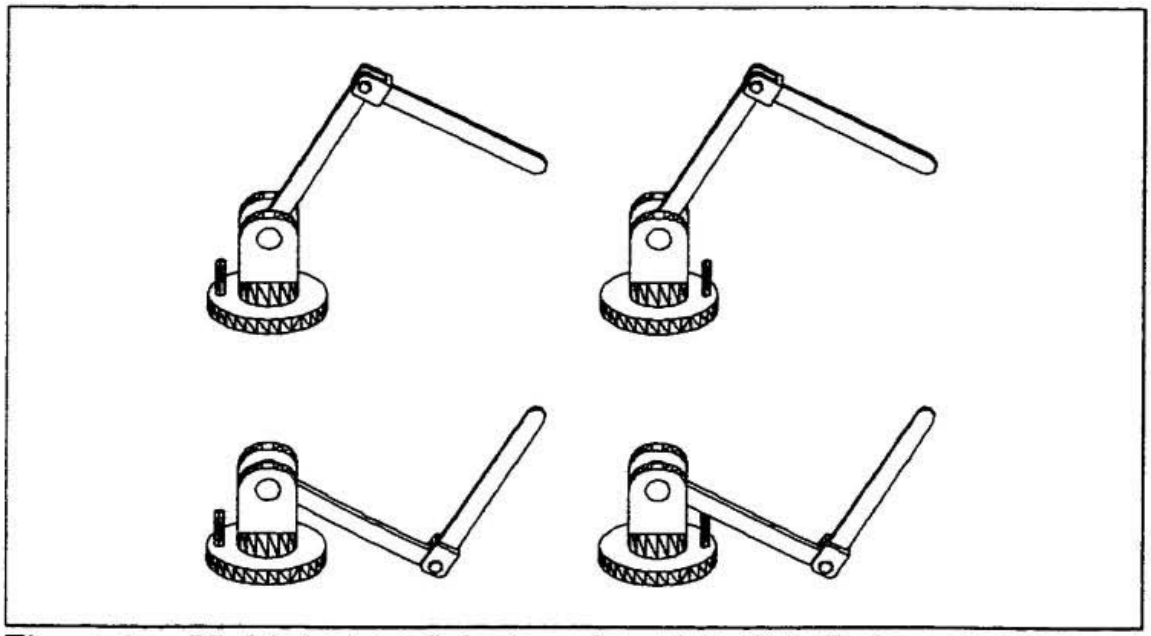

Figure 8: Multiple Arm Solutions for a Revolute Robot

2.4 Singularity

For certain configurations of a manipulator, finite end effector rates require infinite joint rates. These configurations are said to be singular. Joint loads generated by a controller are usually excessive near singular configurations. To determine singular configurations for the wrist of the revolute robot, consider the wrist position equations previously developed:

$$p_{xo} = C\alpha(S\kappa E + S\beta H) \tag{7}$$

$$p_{yo} = S\alpha(S\kappa E + S\beta H) \tag{8}$$

$$p_{zo} = G + C\beta H + C\kappa E \tag{9}$$

Differentiation with respect to time gives:

$$\dot{p}_{xo} = -S\alpha(S\kappa E + S\beta H)\dot{\alpha} + C\alpha(C\kappa E + C\beta H)\dot{\beta} + C\alpha C\kappa E\dot{\varepsilon} \tag{26}$$

$$\dot{p}_{yo} = C\alpha(S\kappa E + S\beta H)\dot{\alpha} + S\alpha(C\kappa E + C\beta H)\dot{\beta} + S\alpha C\kappa E\dot{\varepsilon} \tag{27}$$

$$\dot{p}_{zo} = -(S\beta H + S\kappa E)\dot{\beta} - S\kappa E\dot{\varepsilon} \tag{28}$$

Solving equation (28) for $\dot{\varepsilon}$:

$$\dot{\varepsilon} = [-\dot{p}_{zo} - (S\beta H + S\kappa E)\dot{\beta}]/S\kappa E \tag{29}$$

Substitution into equations (26) and (27) yields:

$$\dot{p}_{xo} = -S\alpha(S\kappa E + S\beta H)\dot{\alpha} + C\alpha(C\kappa E + C\beta H)\dot{\beta} + C\alpha C\kappa / S\kappa[-\dot{p}_{zo} - (S\beta H + S\kappa E)\dot{\beta}] \tag{30}$$

$$\dot{p}_{yo} = -C\alpha(S\kappa E + S\beta H)\dot{\alpha} + S\alpha(C\kappa E + C\beta H)\dot{\beta} + S\alpha C\kappa / S\kappa[-\dot{p}_{zo} - (S\beta H + S\kappa E)\dot{\beta}] \tag{31}$$

Manipulation gives:

$$S\kappa\dot{p}_{xo} + C\alpha C\kappa\dot{p}_{zo} = -S\alpha S\kappa(S\kappa E + S\beta H)\dot{\alpha} + [C\alpha S\kappa(C\kappa E + C\beta H) - C\alpha C\kappa(S\beta H + S\kappa E)]\dot{\beta} \tag{32}$$

$$S\kappa\dot{p}_{yo} + S\alpha C\kappa\dot{p}_{zo} = -C\alpha S\kappa(S\kappa E + S\beta H)\dot{\alpha} + [S\alpha S\kappa(C\kappa E + C\beta H) - S\alpha C\kappa(S\beta H + S\kappa E)]\dot{\beta} \tag{33}$$

Multiplying equation (32) by Cα and equation (33) by Sα and summing the results yields:

$$C\alpha(S\kappa\dot{p}_{xo} + C\alpha C\kappa\dot{p}_{zo}) + S\alpha(S\kappa\dot{p}_{yo} + S\alpha C\kappa\dot{p}_{zo}) = C\alpha[C\alpha S\kappa(C\kappa E + C\beta H) - C\alpha C\kappa(S\beta H + S\kappa E)]\dot{\beta}$$
$$+ S\alpha[S\alpha S\kappa(C\kappa E + C\beta H) - S\alpha C\kappa(S\beta H + S\kappa E)]\dot{\beta}$$

(34)

Equation (34) is of the form $A = B\dot{\beta}$. Infinite $\dot{\beta}$ occurs when B = 0:

$$C\alpha[C\alpha S\kappa(C\kappa E + C\beta H) - C\alpha C\kappa(S\beta H + S\kappa E)]$$
$$+ S\alpha[S\alpha S\kappa(C\kappa E + C\beta H) - S\alpha C\kappa(S\beta H + S\kappa E)] = 0$$

(35)

Expansion and simplification gives:

$$(C\alpha C\alpha + S\alpha S\alpha)S\kappa(C\kappa E + C\beta H) - (C\alpha C\alpha + S\alpha S\alpha)C\kappa(S\beta H + S\kappa E) = 0$$

(36)

$S\kappa C\beta - C\kappa S\beta = 0$

$S\kappa/C\kappa = S\beta/C\beta$

$\tan(\kappa) = \tan(\beta)$

(37)

Equation (37) has two solutions: $\kappa = \beta$ when $\epsilon = 0$ and $\kappa = \beta + \pi$ when $\epsilon = \pi$. These angles correspond to the outer and inner limits of the workspace. At these limits, radial motion is impossible and a degree of freedom is lost. It should be noted that in general end effector rates, $\dot{p}$, and joint rates, $\dot{q}$, are connected by a Jacobian matrix, J, of derivatives through the equation $\dot{p} = J\dot{q}$. Manipulation gives $\dot{q} = J^{-1}\dot{p}$. It turns out that the singular configurations occur where the determinant of J is zero because the determinant appears in the denominator of each component of J^{-1}.

Chapter 3

3.0 Dynamical Equations of Motion

Simulations based only on the kinematic model for a robot assume it will faithfully follow its commanded trajectory. In practice, industrial robots are usually required to move at high speeds to achieve higher productivity and dynamic effects may begin to dominate. A controller design based solely on the kinematic model may result in drive saturation. Nonlinear dynamic coupling and inertial effects can also make the robot deviate significantly from the design trajectory, cause excessive overshoot, and destabilize the system. Finally, the discrete-time and time-delay characteristics of the robot control computer sometimes interact with the robot dynamics to further degrade and destabilize the system's performance.

These problems must be considered in the robot design. The drive motors and circuits must be sized to produce the desired performance. The links must be constructed to minimize the dynamic effects, and the control system must be designed to ensure dynamic performance over the entire work profile of the robot. In order to accomplish these goals, good analytical dynamic models or simulations of the robot are required (Andeen, G. et al, 1988).

The first step in developing the dynamic simulation for a rigid link robot is to write the equations of motion for the system. Two formulations are mainly used to derive the dynamic model: the Lagrangian formulation and the Newton-Euler formulation. The Lagrangian formulation is simpler and more systematic while the Newton-Euler formulation is considered more efficient from a computational point of view (Canudas de Wit, S., 1996).

3.1 Lagrangian Equations of Motion

One approach to the dynamics problem is to consider the instantaneous equilibrium of every part of the system For this, a free body diagram is constructed separating each body from other parts of the system, and their influence is substituted by the initially unknown reaction forces. When a system consists of numerous bodies, however, this technique becomes very cumbersome and requires dealing with large systems of equations. In the dynamics analysis methods based on the principle of virtual work, the constraining effects on a body of other bodies in the system are not considered by the introduction of unknown reactions. Instead, imaginary infinitesimal displacements which could be given to various parts of the system without violating the constraint conditions are considered. These displacements are called virtual displacements.

Virtual work is defined as an increment of work which a force, F, acting on a particle might perform on a virtual displacement of a particle and is written as (Rivin, E., 1988):

$$\delta W = F\delta q\cos\alpha \qquad (38)$$

where δW is the virtual work

δq is the virtual displacement

α is the angle between directions of force and displacement

It can be shown that:

$$\sum \delta W = 0 \tag{39}$$

where $\sum \delta W$ is the sum of virtual works on any virtual displacement of the system.

This constitutes the principle of virtual work.

The Lagrangian formulation is based on the principle of virtual work and D'Alembert's principle which states that reaction loads due to inertia can be treated as static loads in the virtual work statement. Lagranges equations of motion for a general multi degree of freedom system are given by (Arimoto, S., 1996):

$$d(\partial L / \partial \dot{q}_i) / dt - \partial L / \partial q_i = \tau_i \tag{40}$$

where $L = T - V$ is known as the system Lagrangian

T is the total kinetic energy of the system

V is the total potential energy of the system

q_i represents the generalized displacements.

τ_i represents the generalized loads.

Friction loads can be added to the Lagrangian formulation using the Rayleigh Dissipation function although it is often easier to consider them part of the generalized load. By definition, the generalized load, τ, for a particular generalized displacement, q, is equal to the virtual work, δW, done by external loads during a virtual displacement, δq, divided by the virtual displacement:

$$\delta W = (\dot{j} - a\dot{q} - bsign(\dot{q}))\delta q \tag{41}$$

$$\tau = \delta W / \delta q = \dot{j} - a\dot{q} - bsign(\dot{q}) \tag{42}$$

where j is the joint load and a and b are constants associated with wet and dry friction respectively.

Joint/link compliance both add to V and this is often modeled using finite element discretization (Hinchey, M.J., 1994).

3.2 Equations of Motion for the Revolute Robot

Consider a rigid revolute robot structure with a concentrated payload mass, M, at the wrist. The kinetic energy of M is:

$$T = M/2([E^2 + H^2 + 2EHC\varepsilon]\dot{\beta}^2 + E^2\dot{\varepsilon}^2 + 2[E^2 + EHC\varepsilon]\dot{\beta}\dot{\varepsilon} + [HS\beta + ES\kappa]^2\dot{\alpha}^2) \tag{43}$$

Similarly, the potential energy of M is:

$$V = Mg[G + HC\beta + EC\kappa] \tag{44}$$

For a spherical payload with rotary inertia I, we would add to T:

$$I/2(\dot{\alpha}^2+(\dot{\beta}+\dot{\varepsilon})^2) \tag{45}$$

To simplify the presentation, the payload mass is assumed dominant and the I contributions to T are ignored. In this case, the Lagrangian is:

$$L = T - V$$

$$L = M/2([E^2+H^2+2EHC\varepsilon]\dot{\beta}^2+E^2\dot{\varepsilon}^2+2[E^2+EHC\varepsilon]\dot{\beta}\dot{\varepsilon}+[HS\beta+ES\kappa]^2\dot{\alpha}^2) - Mg[G+HC\beta+EC\kappa] \tag{46}$$

The equations of motion are:

$$d(\partial L/\partial\dot{\alpha})/dt-\partial L/\partial\alpha=\phi \tag{47}$$

$$d(\partial L/\partial\dot{\beta})/dt-\partial L/\partial\beta=\psi \tag{48}$$

$$d(\partial L/\partial\dot{\varepsilon})/dt-\partial L/\partial\varepsilon=\omega \tag{49}$$

where ϕ, φ, and ω are the joint loads. Manipulation of equation (47) gives:

$$\partial L/\partial\alpha=0$$

$$\partial L/\partial\dot{\alpha}=M(HS\beta+ES\kappa)^2\dot{\alpha}$$

$$d(\partial L/\partial\dot{\alpha})/dt = M(HS\beta + ES\kappa)^2\ddot{\alpha} + 2M(HS\beta + ES\kappa)(HC\beta + EC\kappa)\dot{\alpha}\dot{\beta} + 2M(HS\beta + ES\kappa)EC\kappa\dot{\alpha}\dot{\varepsilon}$$

Equation (47) becomes:

$$\phi = M(HS\beta + ES\kappa)^2\ddot{\alpha} + 2M(HS\beta + ES\kappa)(HC\beta + EC\kappa)\dot{\alpha}\dot{\beta} + 2M(HS\beta + ES\kappa)EC\kappa\dot{\alpha}\dot{\varepsilon} \quad (50)$$

where ϕ is the base torque

Note the gyroscopic terms $\dot{\alpha}\dot{\beta}$ and $\dot{\alpha}\dot{\varepsilon}$ in equation (50).

Manipulation of equation (48) gives:

$$\partial L/\partial\beta = Mg(HS\beta + ES\kappa) + M(HS\beta + ES\kappa)(HC\beta + EC\kappa)\dot{\alpha}^2$$

$$\partial L/\partial\dot{\beta} = M(E^2 + H^2 + 2EHC\varepsilon)\dot{\beta} + M(E^2 + EHC\varepsilon)\dot{\varepsilon}$$

$$d(\partial L/\partial\dot{\beta})/dt = M(E^2 + H^2 + 2EHC\varepsilon)\ddot{\beta} + M(E^2 + EHC\varepsilon)\ddot{\varepsilon} - MEHS\varepsilon\dot{\varepsilon}^2 - 2MEHS\varepsilon\dot{\beta}\dot{\varepsilon}$$

Equation (48) becomes:

$$\varphi = M(E^2 + H^2 + 2EHC\varepsilon)\ddot{\beta} + M(E^2 + EHC\varepsilon)\ddot{\varepsilon} - MEHS\varepsilon\dot{\varepsilon}^2 - 2MEHS\varepsilon\dot{\beta}\dot{\varepsilon} - Mg(HS\beta + ES\kappa) - M(HS\beta + ES\kappa)(HC\beta + EC\kappa)\dot{\alpha}^2 \quad (51)$$

where φ is the shoulder torque

Note the centrifugal terms $\dot{\varepsilon}^2$ and $\dot{\alpha}^2$ and the Coriolis term $\dot{\beta}\dot{\varepsilon}$.

Manipulation of equation (49) gives:

$$\partial L/\partial \varepsilon = -MEHS\varepsilon\dot{\beta}^2 - MEHS\varepsilon\dot{\beta}\dot{\varepsilon} + MgES\kappa + M(HS\beta + ES\kappa)EC\kappa\dot{\alpha}^2$$

$$\partial L/\partial \dot{\varepsilon} = ME^2\dot{\varepsilon} + M(E^2 + EHC\varepsilon)\dot{\beta}$$

$$d(\partial L/\partial \dot{\varepsilon})/dt = ME^2\ddot{\varepsilon} + M(E^2 + EHC\varepsilon)\ddot{\beta} - MEHS\varepsilon\dot{\beta}\dot{\varepsilon}$$

Equation (49) becomes:

$$\omega = ME^2\ddot{\varepsilon} + M(E^2 + EHC\varepsilon)\ddot{\beta} - MEHS\varepsilon\dot{\beta}^2 - MgES\kappa - M(HS\beta + ES\kappa)EC\kappa\dot{\alpha}^2 \qquad (52)$$

where ω is the elbow torque.

Note the centrifugal terms $\dot{\beta}^2$ and $\dot{\alpha}^2$.

Chapter 4

4.0 Simulation

Robot simulation is the solution of the equations of motion of the manipulator that yields the positions, velocities, and accelerations of the system elements as functions of time. In some cases, these simulations also yield the drive forces and torques required to produce motion and the resulting internal forces in the systems mechanical elements. The drive loads are important in the design of the system's actuators and control circuits. The internal forces are required for the design of the mechanical components of the manipulator. CAD based simulations enable generation of displays of the manipulator motions to aid in evaluation and interpretation of system performance.

In most cases, three levels of simulation are used to study a manipulator design. The first level, kinematic simulation, assumes the motion of the manipulator is determined by the commanded joint displacements. The positions and velocities of the manipulator are calculated using standard kinematic models and the dynamics of the manipulator and its control system are ignored. The second and more complex form of simulation uses a rigid link dynamic model of the arm. At this level of

simulation, the dynamic characteristics of the control systems, drive actuators, transmissions, and control computer may be considered. In this case, the equations of motion are coupled sets of nonlinear, algebraic, differential and difference equations whose solution requires the use of numerical forward integration techniques. This form of simulation is most useful in designing and evaluating the manipulator's control systems, which include its computer and its drive system. In the third and most complex form of simulation, the distributed mass and flexibility of the manipulator's mechanical elements are included in the model, in addition to the control and drive properties contained in the rigid link analysis. Finite element methods typically must be used to in order consider the links of the geometric complexity found in industrial manipulators (Andeen, G., 1988).

The basic structure of all three levels of simulation is shown in figure 9. In this structure, the data describing the system parameters and commands are input and all calculations that are not time dependent are performed. These data may be obtained directly from CAD models of the manipulator's elements. Based on user defined or default initial conditions for the end effector position vector, $\mathbf{p}(t_o)$, the equations of motion are then evaluated. The vector $\dot{\mathbf{p}}(t)$ is then integrated using standard numerical forward-integration algorithms to obtain the state of the system at $t_o + \Delta t$. This provides the new initial conditions to repeat the process, so the solution to the equations of motion advances in time in the manner of classical forward interaction (Korn, G., 1978).

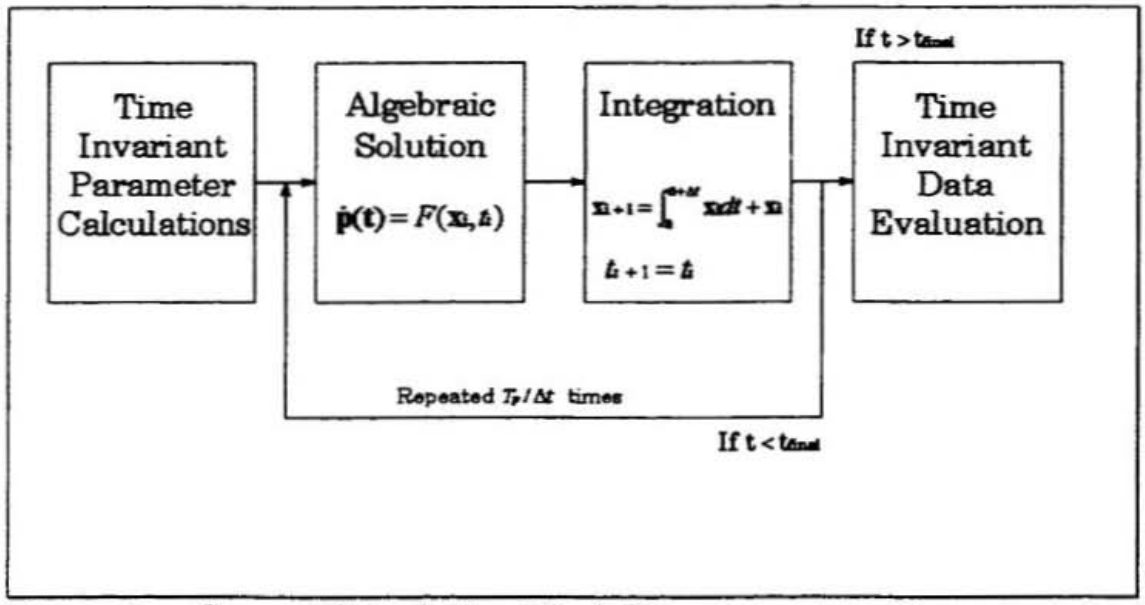

Figure 9: General Simulation Block Diagram

4.1 Control Torque Control

A basic problem in controlling robots is to make the manipulator follow a preplanned desired trajectory. In more advanced applications, the robot controller must enable trajectory planning which involves finding the prescribed path, collision avoidance, and control of actuator saturation. For point to point position control, there are number of control schemes which may be used to provide acceptable positioning accuracy (An, C. et al, 1988).

There have been numerous robot control schemes proposed which can be considered as special cases of the class of Computed Torque Controllers. To help develop this form of controller for a revolute robot, the Lagrangian equations of motion (50), (51) and (52) may be generalized in the following form (Lewis, F. et al, 1993):

$$M(q)\ddot{q} + V(q,\dot{q}) + G(q) = \tau$$

(53)

where M(q) is an n x n inertia matrix, q and its derivatives are n x 1 vectors of generalized co-ordinates, and $V(q,\dot{q})$, G(q) and τ are n x 1 vectors containing velocity-dependent torques, gravity torques, and input torques respectively.

To account for friction and distrurbances, this generalized robot dynamical equation becomes:

$$M(q)\ddot{q} + V(q,\dot{q}) + G(q) + F(q) + \tau_d = \tau$$

(54)

Alternatively, this equation may be written as:

$$M(q)\ddot{q} + N(q,\dot{q}) + \tau_d = \tau$$

(55)

where the nonlinear terms are represented by $N(q,\dot{q}) = V(q,\dot{q}) + G(q) + F(q)$.

For a desired trajectory, $q_d(t)$, an output tracking error may be defined to ensure trajectory tracking by the joint variable as follows:

$$e(t) = q_d(t) - q(t)$$

(56)

Differentiating twice yields:

$$\dot{e} = \dot{q}_d - \dot{q}$$

$$\ddot{e} = \ddot{q}_d - \ddot{q}$$

Substituting for $\ddot{q}$ in equation (55) gives:

$$\ddot{e} = \ddot{q}_d + M^{-1}(N + \tau_d - \tau) \tag{57}$$

Defining the control input function as:

$$u = \ddot{q}_d + M^{-1}(N - \tau) \tag{58}$$

This feedback linearizing transformation may be inverted to yield:

$$\tau = M(\ddot{q}_d - u) + N \tag{59}$$

This is referred to as the Computed Torque Control Law.

The use of the control input, u, has converted a complicated non-linear controls problem into a simple design for a linear system. The block diagram for a Computed Torque Control scheme shown in figure 10 illustrates that this form of control utilizes both an inner control loop and an outer control loop. The computed torque depends on the inversion of the robot dynamics and is sometimes called inverse dynamics control. τ(t) is computed in the inner loop by substituting $\ddot{q}_d - u$ for $\ddot{q}$ in equation (55). If the dynamic model is exact, the nonlinear dynamic perturbations are exactly canceled and what

is left is a decoupled linear system that can be controlled according to standard techniques (An, C. et al, 1988). Unfortunately, dynamic models are never exact and an outer loop feedback signal is therefore required to counteract trajectory drift or error growth. Independent joint control using proportional plus integral plus derivative (PID) control may be used to compute the corrective torque in the outer loop.

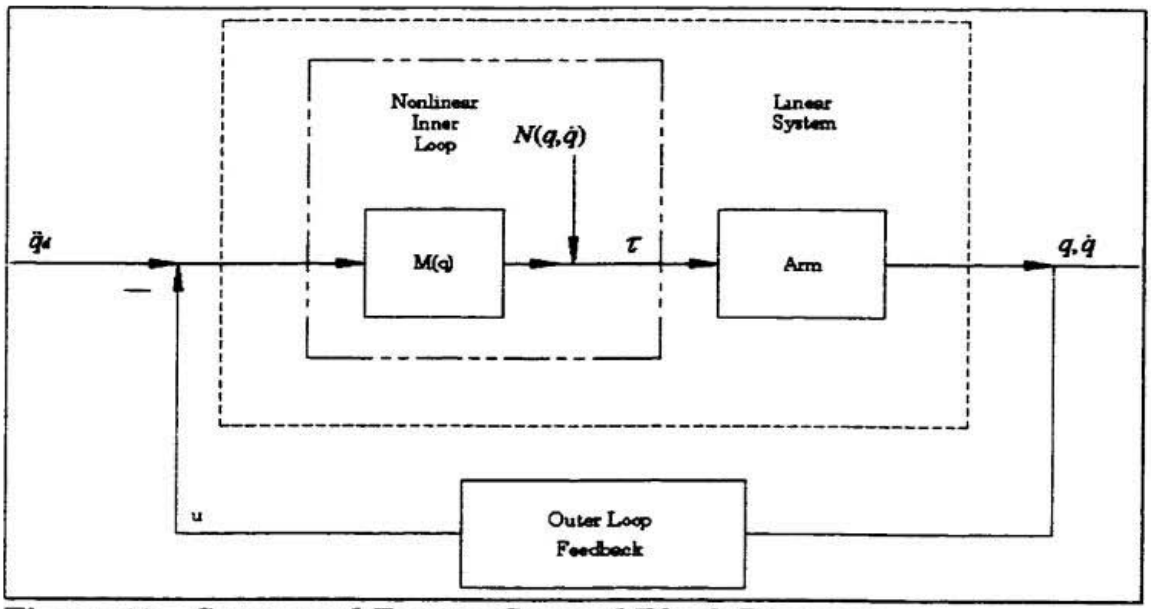

Figure 10: Computed Torque Control Block Diagram

4.2 PID Control

The outer loop control feedback signal may be generated using PID control in accordance with the equation:

$$u = -K_p e - K_v \dot{e} - Ki \int e dt \tag{60}$$

where K_p, K_v, and K_i are the proportional, derivative and integral gains repsectively.

The proportional control mode produces a change in the controller output proportional to the error signal. With this mode of control at steady state there is a residual error due to gravity. This would imply that the proportional gain should be as high as possible since increasing the gain should reduce the residual error required to produce a change in the controller output. Increasing the gain, however, increases the tendency for oscillation of the manipulator about the position setpoint. To eliminate the residual error, an integral control term is added. The integral mode changes the controller output by an amount proportional to the integral of the error signal. As long as there is an error, the integral mode will change the output at a rate proportional to the sum of the error over time. The derivative control mode may be added to limit oscillations. This mode of control changes the output of the controller proportional to the rate of change of the error signal. The derivative mode is an attempt to anticipate an error by observing how fast the error is changing, and using the rate of change to produce a control action that will reduce the expected error. Derivative control contributes to the output of the controller only when the error is changing

and is, therefore, always used in combination with the proportional, or proportional plus integral control modes (Bateson, R., 1996).

The PID gains must be selected for each joint of the manipulator separately since the Computed Torque controller does not result in a decoupled control strategy in the inner loop. Thus, information on all joint positions and velocities is needed to compute the control torque for any one joint. The PID gains may be selected based on the natural frequency of the manipulator and the desired damping in the system. The PD gains may be selected as (Lewis, F. et al, 1993):

$$K_{pi} = \omega_{ni}^2 \tag{61}$$

$$K_{vi} = 2\zeta\omega_{ni} \tag{62}$$

where ζ is the desired damping ratio.

ω_{ni} is the natural frequency for joint error i in [rad/s].

Since it is undesirable for the robot to exhibit overshoot, the PD gains are usually selected for critical damping $\zeta = 1$. In this case:

$$K_{vi} = 2\sqrt{K_{pi}} \tag{63}$$

It can be shown that for closed-loop stability, the integral gain is subject to the condition:

$$K_{ii} < K_{vi}K_{pi} \tag{64}$$

4.3 Zero Order Hold

While most controllers are designed in continuous time, they are implemented on robots digitally. In this case, the control signals are updated at discreet instants of time using a microprocessor. To verify that a controller will operate as expected, it is highly desirable to simulate it in its digitized form prior to actual implementation.

A generalized digital control scheme may be represented in simplified block diagram form as shown in figure 11 (Lewis, F., 1993). The plant or system to be controlled is a continuous-time system and K(z) is the dynamic digital controller where z is the Z-transform variable. The reference input r(t) is the desired trajectory that y(t) should follow, and e_k is the discrete tracking error.

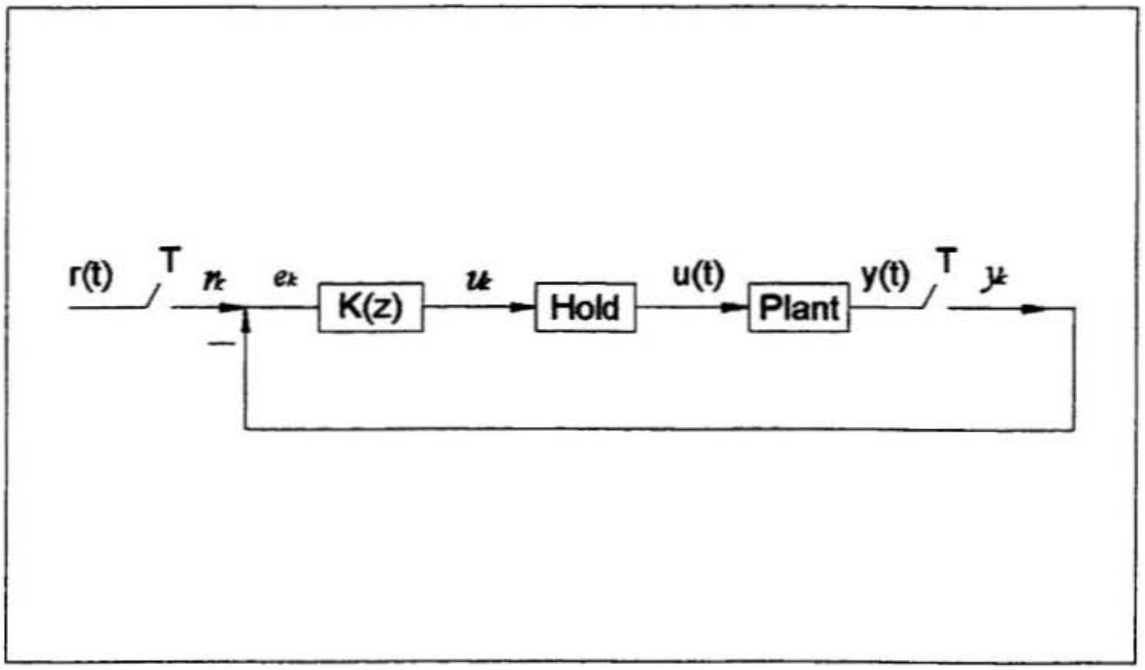

Figure 11: Digital Controller

The sampler with sample period T is an analog-to-digital (A/D) converter that takes the samples y(kT) of the output y(t) that are required by the software controller. In robot control, y(t) might represent the vector composed of q(t) and $\dot{q}(t)$. The hold device is a digital-to-analog (D/A) converter that converts the discrete control samples u_k computed by the software controller K(z) unto the continuous time control u(t) required by the plant.

Zero-order hold (ZOH) is generally used for controls purposes. For ZOH, the input u_k and the output u(t) are shown in figure 12. Note that u(t) is held continuous until updated at times kT.

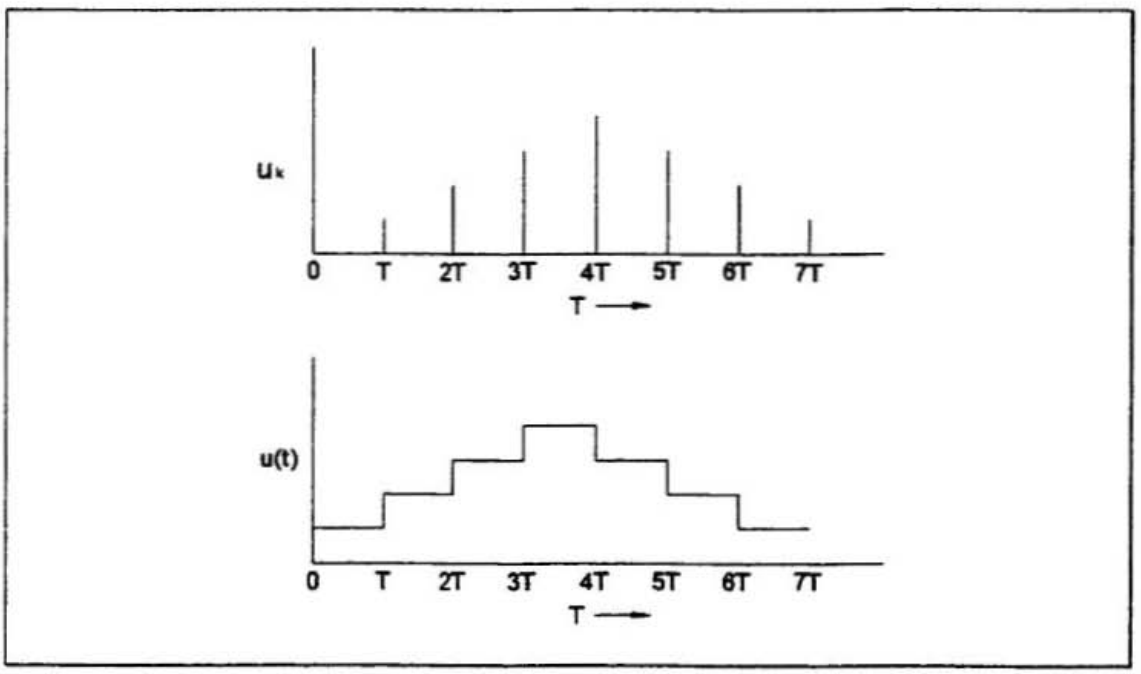

Figure 12: Zero Order Hold

4.4 Numerical Simulation

The equations of motion (50), (51), and (52) for the revolute robot can be expressed in matrix form as:

$$\begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & 0 \\ 0 & 0 & A_{33} \end{bmatrix} \begin{bmatrix} \ddot{\beta} \\ \ddot{\varepsilon} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} \tag{65}$$

where

$$A_{11} = M(E^2 + H^2 + 2EHC\varepsilon)$$

$$A_{12} = M(E^2 + EHC\varepsilon)$$

$$A_{21} = M(E^2 + EHC\varepsilon)$$

$$A_{22} = ME^2$$

$$A_{33} = M(HS\beta + ES\kappa)^2$$

$$B_1 = \varphi + M(HS\beta + ES\kappa)(HC\beta + EC\kappa)\dot{\alpha}^2 + Mg(HS\beta + ES\kappa) + 2MEHS\varepsilon\dot{\beta}\dot{\varepsilon} + MEHS\dot{\varepsilon}\dot{\varepsilon}^2$$

$$B_2 = \omega + M(HS\beta + ES\kappa)ECk\dot{\alpha}^2 + MgES\kappa - MEHS\varepsilon\dot{\beta}^2$$

$$B_3 = \phi - 2M(HS\beta + ES\kappa)(HC\beta + EC\kappa)\dot{\alpha}\dot{\beta} - 2M(HS\beta + ES\kappa)ECk\dot{\alpha}\dot{\varepsilon}$$

Multiplication of both sides of the matrix equation by the inverse of the A matrix gives equations for $\ddot{\alpha}$, $\ddot{\beta}$, and $\ddot{\varepsilon}$:

$$\ddot{\alpha} = \frac{B_3}{A_{33}} \tag{66}$$

$$\ddot{\beta} = \frac{A_{22}B_1 - A_{12}B_2}{A_{11}A_{22} - A_{12}A_{21}} \tag{67}$$

$$\ddot{\varepsilon} = \frac{A_{11}B_2 - A_{21}B_1}{A_{11}A_{22} - A_{12}A_{21}} \tag{68}$$

Letting $\ddot{\alpha} = K1$, $\ddot{\beta} = K2$, $\ddot{\varepsilon} = K3$, $a = \dot{\alpha}$, $b = \dot{\beta}$, and $c = \dot{\varepsilon}$, these become a set of six first order ordinary differential equations:

$$\dot{\alpha} = a$$

$$\dot{\beta} = b$$

$$\dot{\varepsilon} = c$$

$$\dot{a} = K1$$

$$\dot{b} = K2$$

$$\dot{c} = K3$$

Using a simple Euler one step integration scheme gives:

$$(\alpha_{new} - \alpha_{old})/\Delta t = a_{old}$$

$$(a_{new} - a_{old})/\Delta t = K1_{old}$$

$$(\beta_{new} - \beta_{old})/\Delta t = b_{old}$$

$$(b_{new} - b_{old})/\Delta t = K2_{old}$$

$$(\varepsilon_{new} - \varepsilon_{old})/\Delta t = c_{old}$$

$$(c_{new} - c_{old})/\Delta t = K3_{old}$$

where subscript old indicates values at the beginning of a time step and subscript new indicates values at the end of a time step.

Using this integration scheme, the equations of motion can be solved as a function of time and the manipulator motion can be simulated using the Computed Torque PID control model.

Chapter 5

5.0 CROBOTS: A CAD Based Tool for Robot Simulation

CROBOTS is a proposed software simulation tool that is designed to assist users in the design, application, and programming of educational and industrial robots. The software is written using the AutoLisp programming language and runs as a third party application inside AutoCAD R14. CROBOTS combines the impressive graphics capability of AutoCAD with custom developed tools for robot model creation, operation cycle planning, and simulation of robot controllers. While the software is primarily designed as an educational tool for robotics related curricula, it also has potential for commercial applications.

5.1 Commercial Applications

In most industrial applications, motion planning is performed by the programmer rather than a computer. The actual robot hardware in its work environment is used to plan a strategy for performing a task. This usually involves using a teach pendant or a high level programming language to facillitate the teaching or the programming of the robot. Once the

programming is completed for a task, the robot hardware is used in playback mode to test the program and the manipulation strategy. Debugging is accomplished through a teach-playback loop. The robot programmer can use CROBOTS in conjunction with native AutoCAD functionality to assess preliminary robot operation cycle designs. This could reduce unproductive programming of the actual robot hardware.

The robot's performance in response to various control strategies may also be graphically simulated using CROBOTS. This may aid designers in the preliminary stages of controller design.

5.2 Educational Applications

CROBOTS offers a number of potential benefits for educational institutions who are faced with the challenge of maintaining current and adequate resources related to the field of robotics with only limited budgets. Firstly, CROBOTS is written as a third party application for AutoCAD. AutoCAD has dominated the PC CAD market and is recognized as the industry standard. Instruction in the use of AutoCAD is a fundamental component of engineering technology and degree programs in many Colleges and Universities both nationally and internationally. The learning curve for CROBOTS is insignificant for students and educators who already have skills in AutoCAD. Robotics instruction utilizing CROBOTS may therefore be focused on the learning of key concepts and not on how to use the software. Secondly, the ease of programming in AutoLISP should facilitate the development of additional CROBOTS routines by students and educators. Students will also benefit from this opportunity to learn AutoLISP programming since this skill may be important for future related employment in industry. Furthermore, the graphical simulation capability of CROBOTS allows students to actually see the effects of manipulating

dynamic and control parameters on robot performance. This may help reinforce theoretical concepts delivered using traditional lectures. Finally, the cost of robot hardware is often prohibitive for most educational institutions. CROBOTS provides students with an offline tool which can be used to learn basic concepts before moving on to the actual robot hardware. In addition, since most engineering educational institutions already use AutoCAD, they should not have to incur the added cost of a graphics engine as would be the case with commercial offline robotics programming software.

5.3 CROBOTS Overview

CROBOTS includes customized menus and new AutoLisp functions which provide the user with specialized commands that facilitate the use of AutoCAD for robot geometric modeling, operation cycle planning, and controller simulation. These specialized functions are made available through customization of the AutoCAD menu file as shown in figure 13.

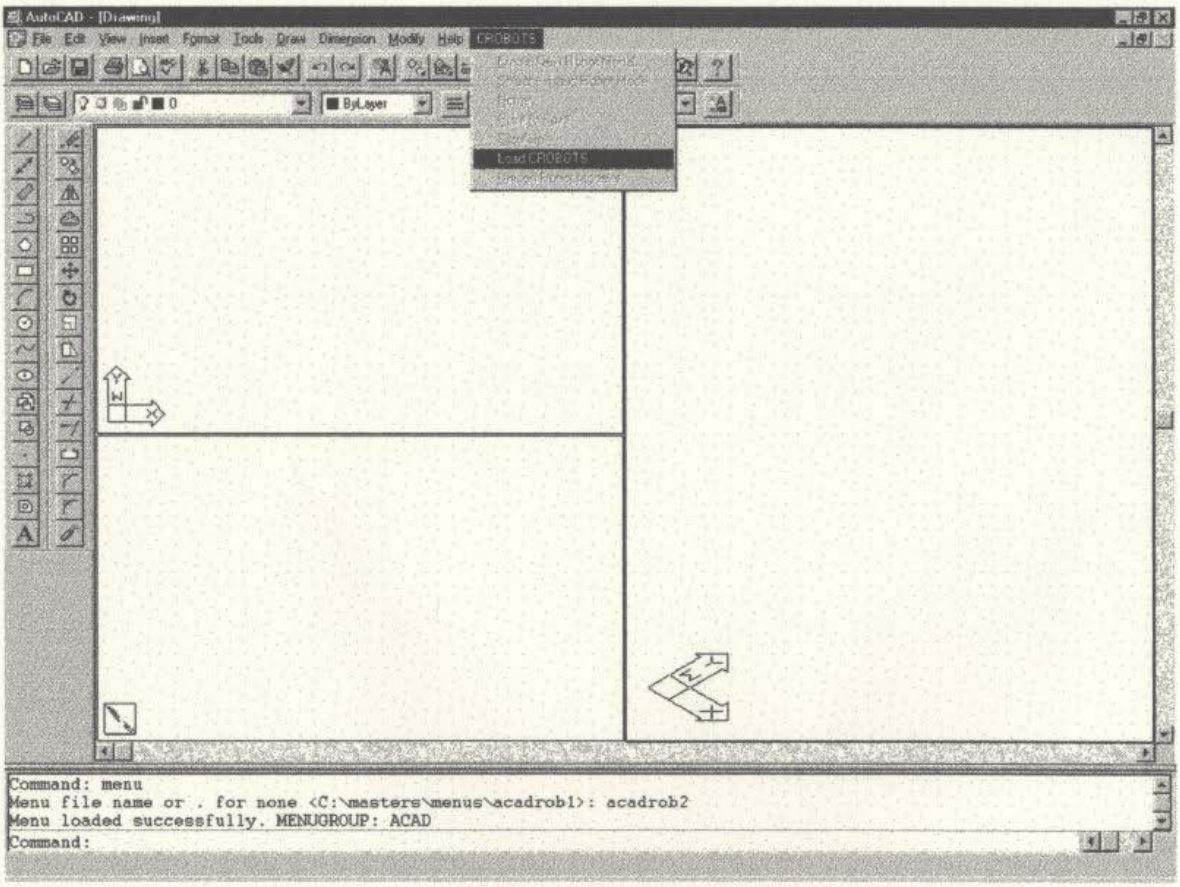

Figure 13: Customized Pull-Down Menu for CROBOTS

The modified menu file presents the user with a CROBOTS pull-down menu which initially provides only a "Load CROBOTS" option. Selection of this option loads a second custom menu which then makes all CROBOTS functions available to the user as shown in figure 14.

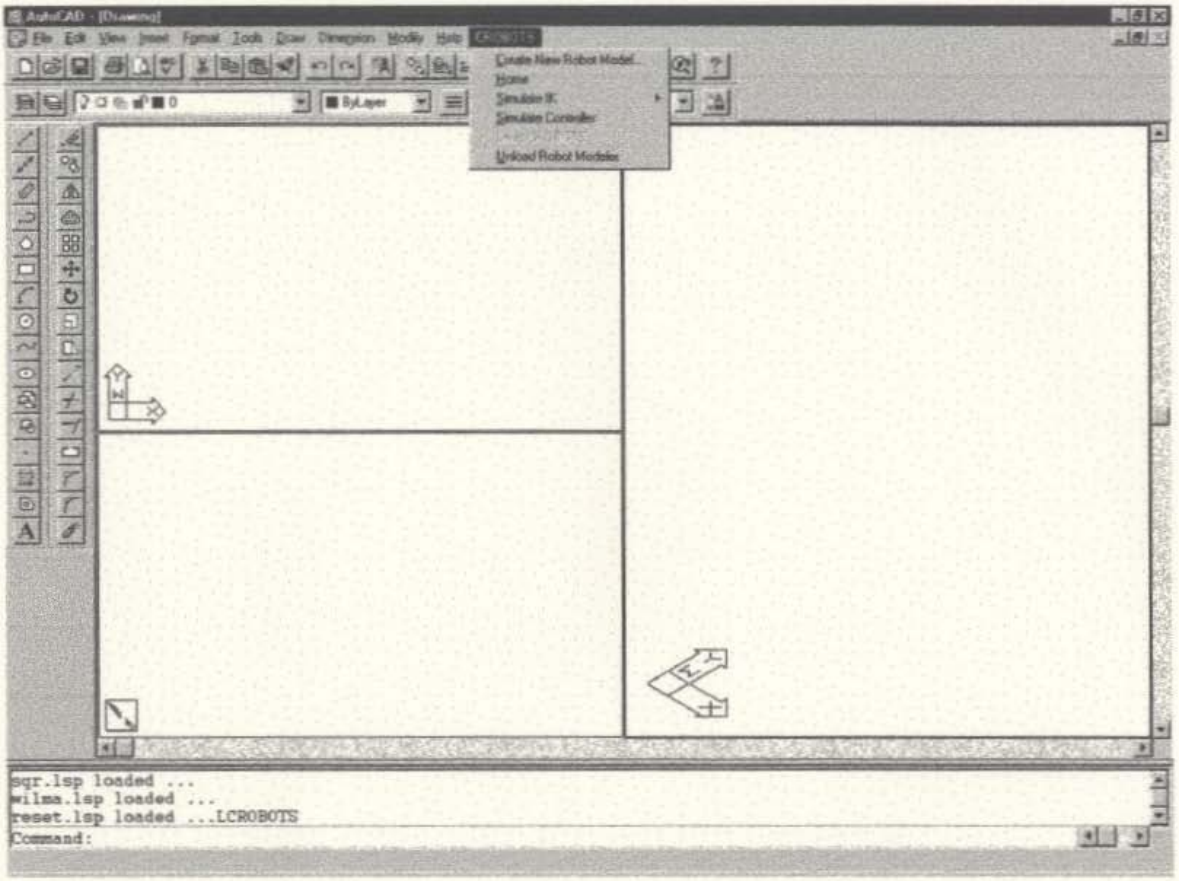

Figure 14: CROBOTS Main Menu

The main menu features five menu options including "Create New Robot Model", "Home", "Simulate IK", Simulate Controller", and "Unload Robot Modeler".

Create New Robot Model

This menu option provides a customized parametric drawing tool which allows the user to quickly generate a functional geometric model of the robot. Selection of this option displays an image box from which the user may choose the robot configuration desired. The image box includes options for all five mechanical configurations of robots including revolute, cartesian, cylindrical, spherical and SCARA as shown in figure 15.

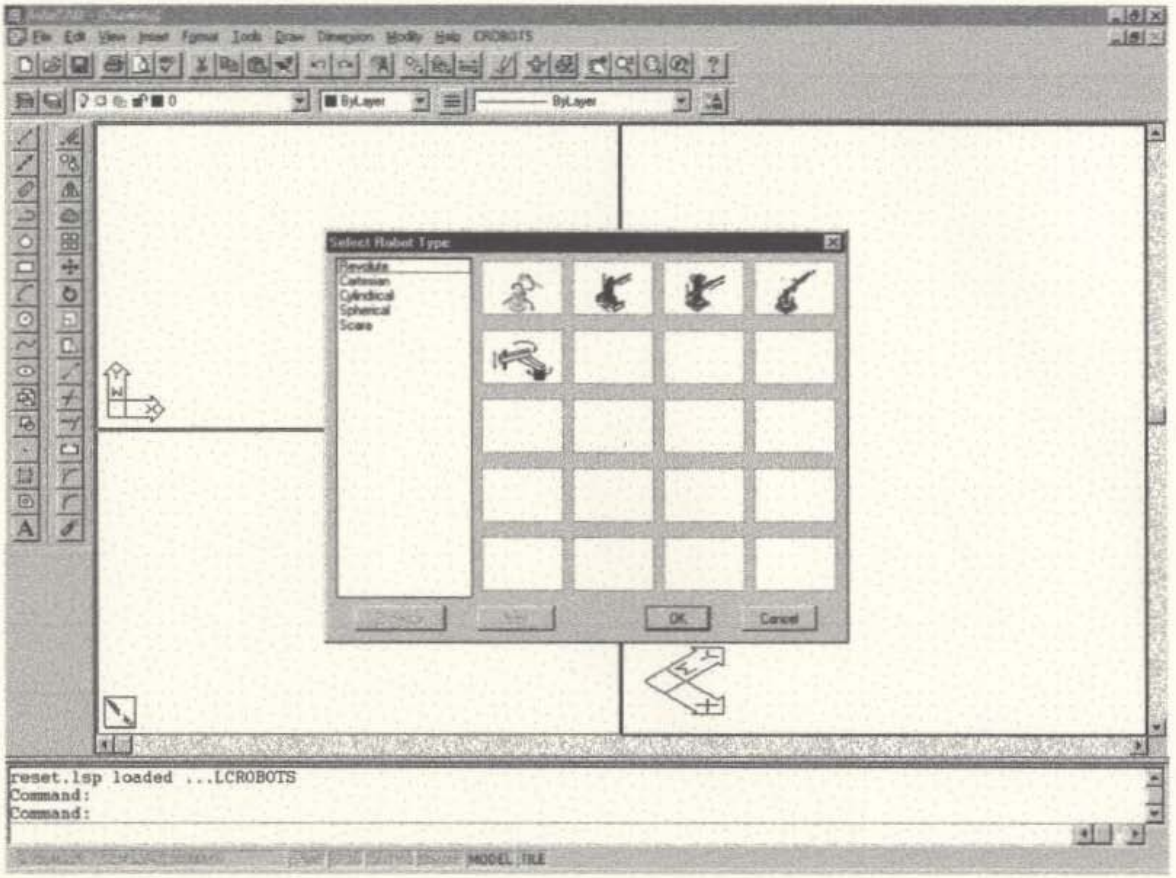

Figure 15: Image Menu for "Create New Robot Model"

Following selection of the configuration type, the user is only required to input the desired robot dimensions. In the case of the revolute robot, for example, the user is required to input the base height, the length of link 1, and the length of link 2. A 3D solid model of the robot geometry is then automatically generated based on this input as shown in figure 16.

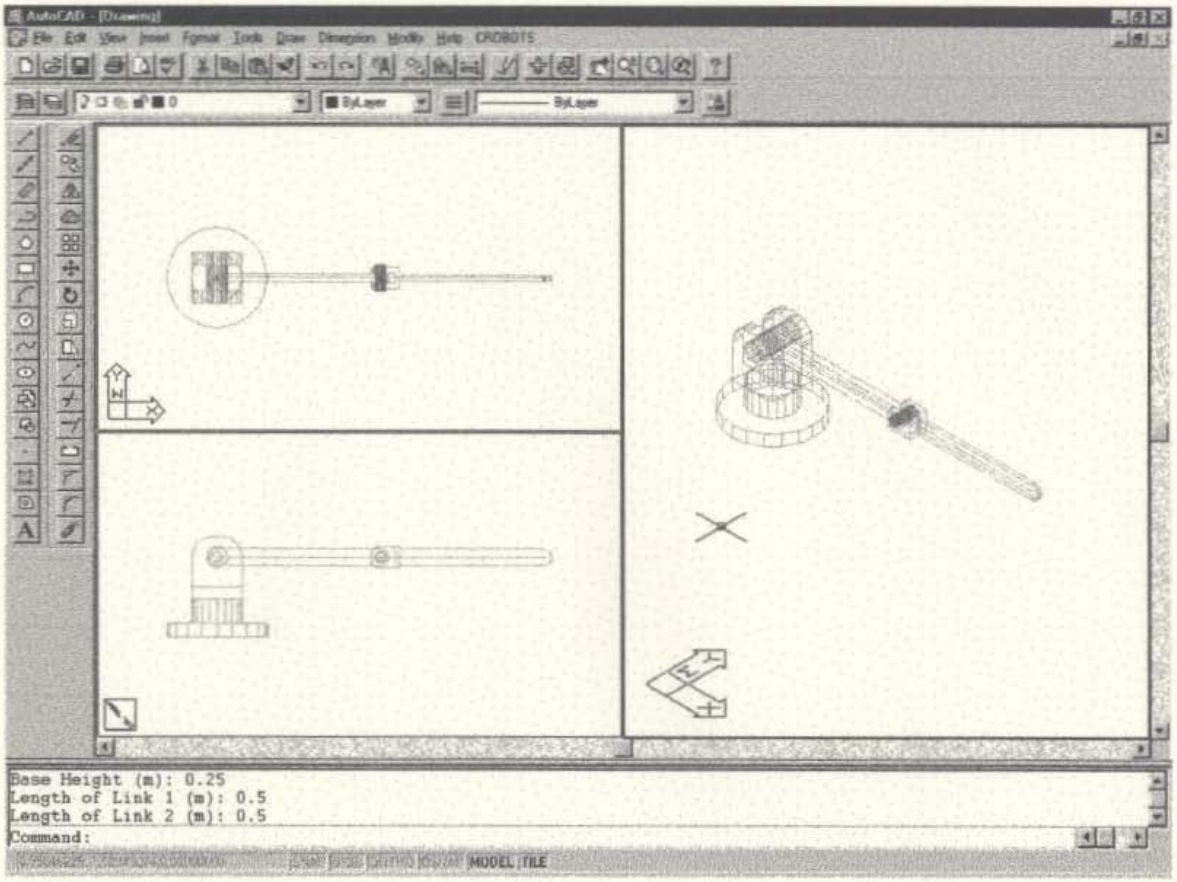

Figure 16: Create New Robot Model

Home

The home function simulates a standard capability of industrial robots to assume a reference or default start-up position. Industrial robots, which incorporate incremental optical encoders for position measurement, require this capability to allow reset of counters before programming. In the CROBOTS program, this function similarly initializes relevant AutoLisp variables and automatically re-positions the robot to its required startup position as shown in figure 17.

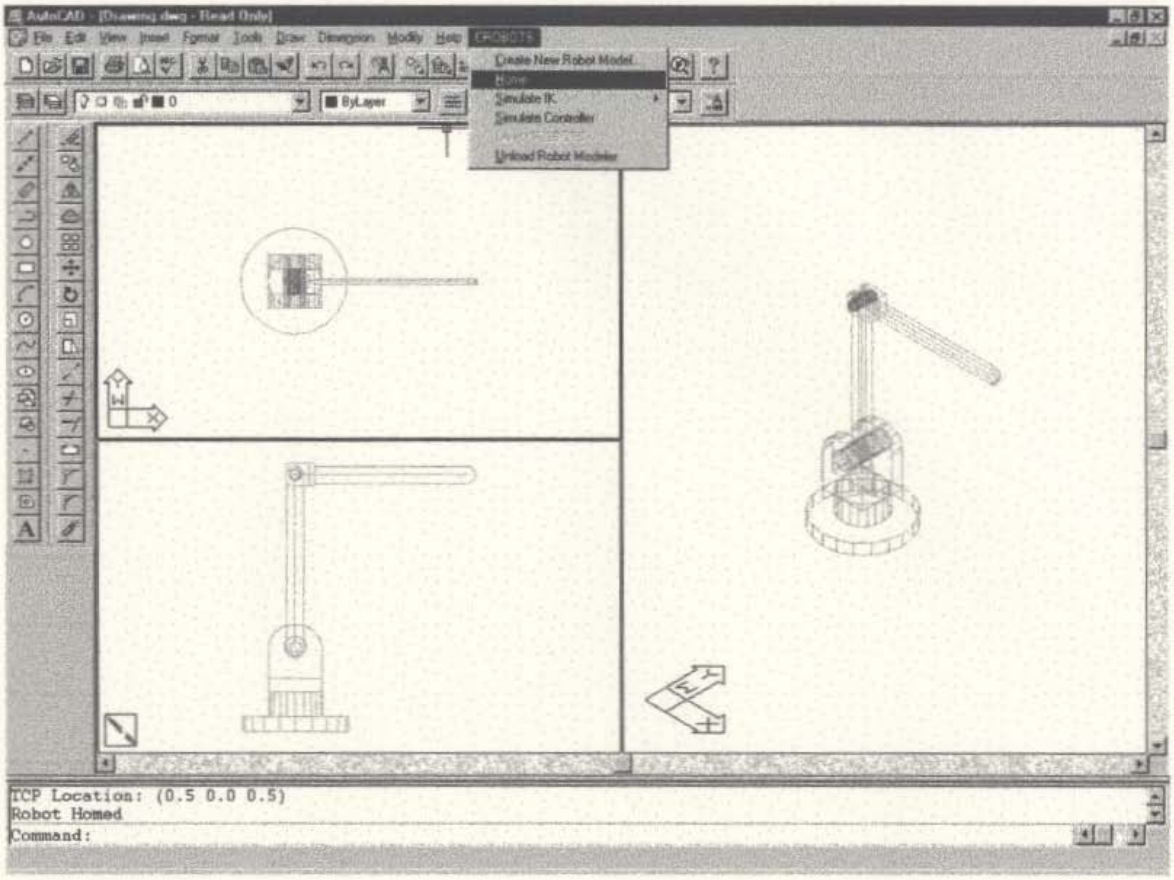

Figure 17: Home Position for the Robot

Simulate IK (Inverse Kinematics)

This menu option provides the user with two submenu options "Teach" and "Run Cycle". The "Teach" option allows the user to simulate the industrial on-line programming practice of teaching and recording a series of points which define the robot trajectory. This function computes the inverse kinematics solution for the manipulator which then allows automatic re-positioning of the robot TCP. Following selection of this option, the user specifies the desired point location by either "cursor picking" a location in the drawing window or by keyboard entry of the point co-ordinates at the AutoCAD command prompt. In either case, specification of the desired point repositions the manipulator TCP as shown in figure 18. The user is also provided with the option to record the point co-ordinates. Recorded points are

stored in an ASCII text file which may be later used in CROBOTS or for download to industrial robot controllers.

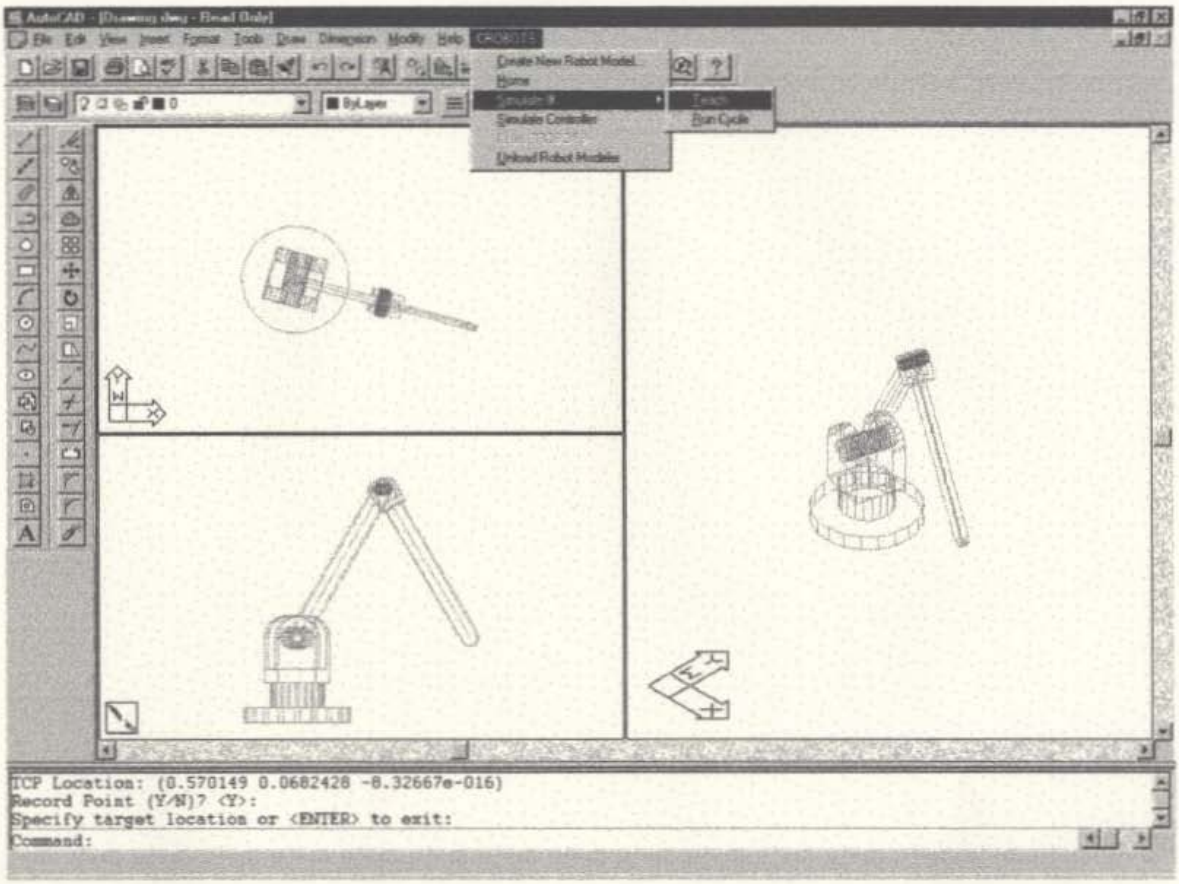

Figure 18: Teaching the Robot

The function may be used in conjunction with native AutoCAD commands to simulate path tracking by the robot. For example, if a circular path is required, AutoCAD can be used to draw the circle and then subdivide it into a number of segments. The teach function can then be used to approximate tracking of the circle with point to point moves from the beginning of a segment to the end of a segment. Figure 19 illustrates a segmented elliptical path with segment endpoints displayed. AutoCAD's object snap options enable the user to use the "Teach" function to individually record each point and reposition the manipulator accordingly.

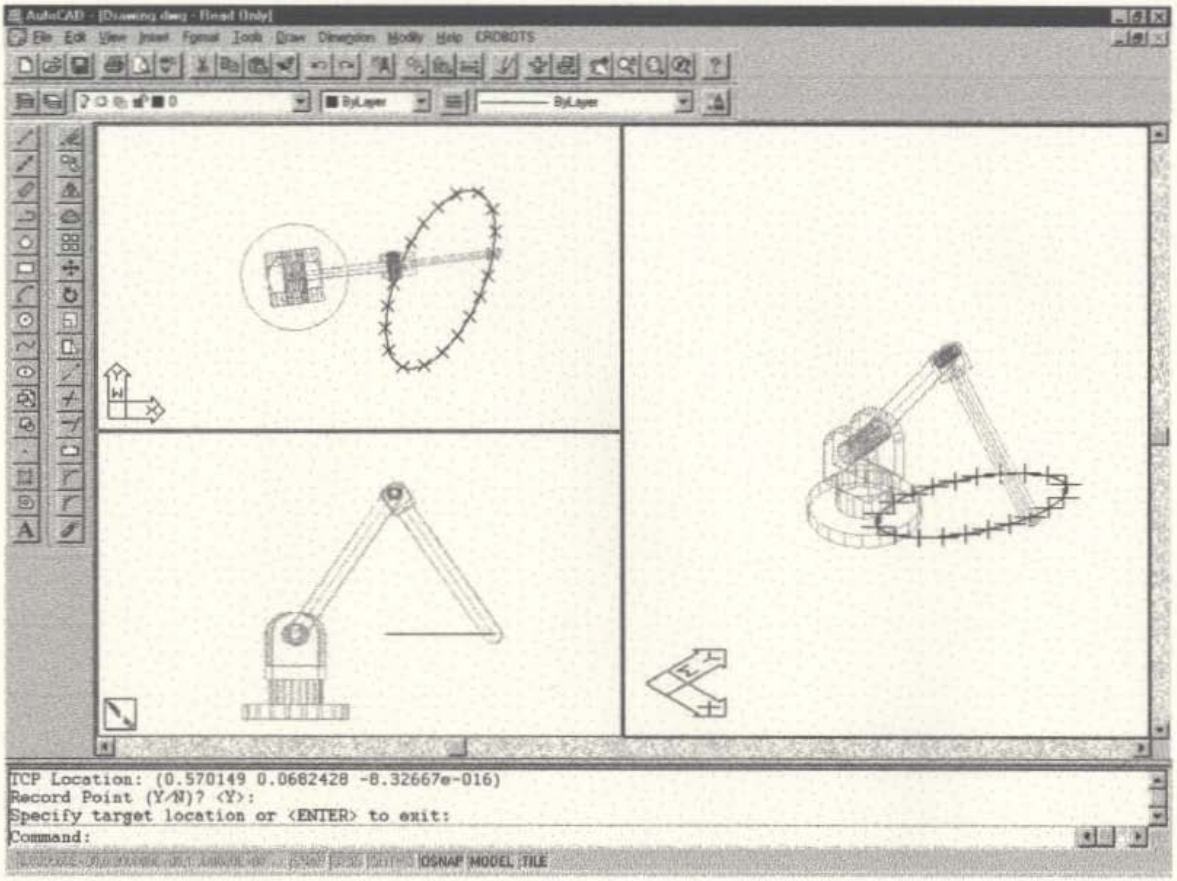

Figure 19: Trajectory Approximation using the "Teach" Function

The "Run Cycle" function enables the user to "playback" the repositioning of the manipulator defined through the series of points recorded in the ASCII text file. In effect, this function allows graphical simulation of the PTP motion of the manipulator along the approximated trajectory.

Simulate Controller

This menu option enables numerical simulation of a computed torque with PID controller and graphical simulation of the corresponding manipulation for PTP motion of the robot. Following selection of this option, the user is prompted for gain values, cycle time, and command angles as shown in figure 20. The function then provides a graphical simulation of the manipulator motion in response to torques generated based on the modified

computed torque model. The overall output torque at each joint is calculated based on the dynamical equations of motion , the PID control signal, the relay control signal, and the use of zero order hold, if desired.

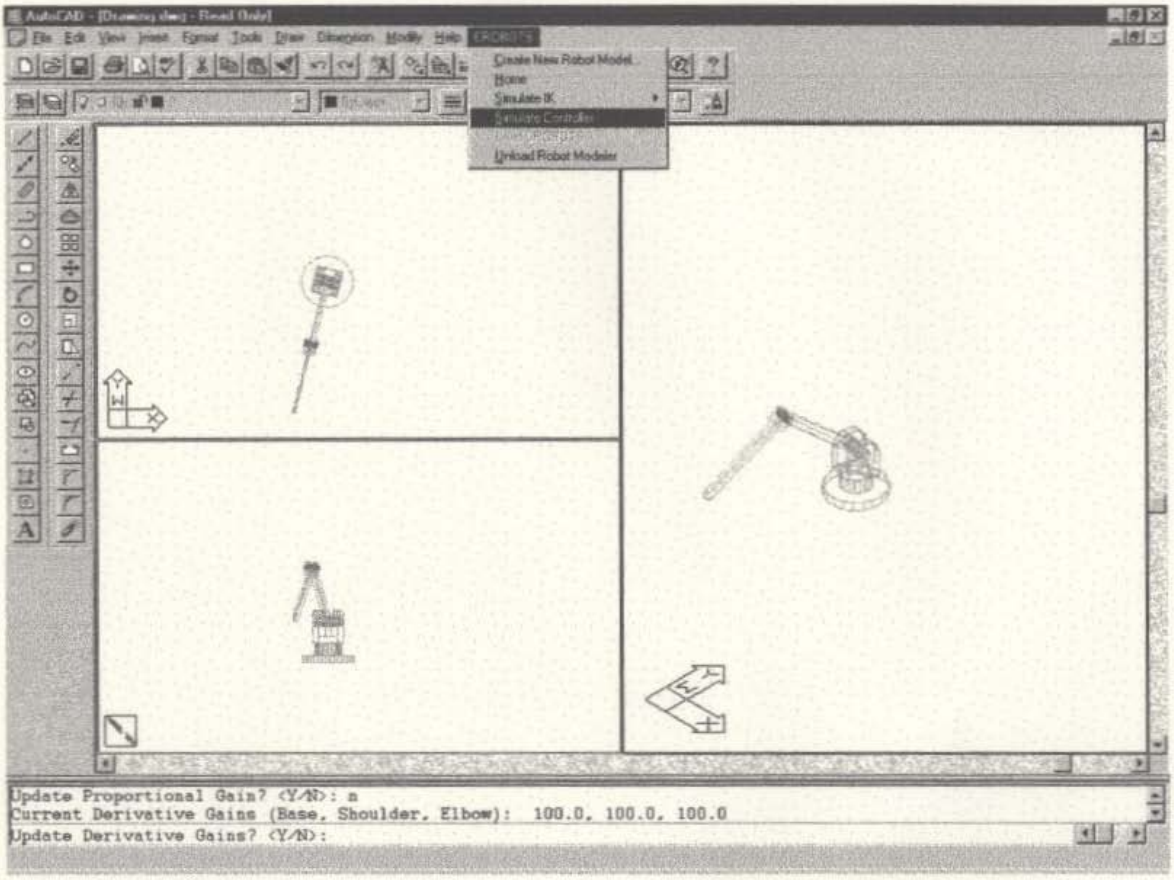

Figure 20: Simulate Controller

The user is also provided with a utility which allows automatic generation of a 3D polyline path which graphically summarizes the manipulator motion for the given control parameters. This capability may be used to compare the effects of changing control parameters such as the ZOH sampling rate.

Unload Robot Modeler

This menu option is included to allow the user to unload CROBOTS and restore the standard AutoCAD menu display.

5.4 Overview of AutoCAD Customization for CROBOTS

The development of CROBOTS involved the modification of the AutoCAD R14 menu file, the creation of a template drawing file, the creation of slide files, the creation of blocks, and the creation of new AutoLisp functions. Two modified menu files were created to provide the CROBOTS pull-down menu: crobots1.mnu and crobots2.mnu. Secondly, a template drawing file with the required layers and viewport settings was developed. Finally, Autolisp functions were created to enable the menu options "Create New Robot Model", "Teach", "Run Cycle", and "Simulate Controller".

5.4.1 AutoCAD R14

AutoCAD R14 is a general purpose Computer Aided Design (CAD) drafting application for the PC. The software provides an open architecture that permits the user to customize and extend many AutoCAD features to suit the particular requirements at hand. This has led to the development of numerous third party software applications which are specifically designed to improve the productivity of CAD drafting in particular technical disciplines such as Architectural Design. AutoCAD customization capability has improved with each new release and many past custom user developed functions have become part of the standard tools offered in following releases of AutoCAD (AutoDesk, 1997).

AutoCAD customization capability enables users to:

- Develop custom menus
- Program their own dialogue boxes
- Create scripts to automate repetitive command sequences

- Define your own text fonts
- Define your own line types
- Define your own hatch patterns
- Create custom symbols and parts libraries
- Create template drawings with custom default settings
- Export/Import DXF files to share drawing geometry with other applications
- Generate slides or postscript files
- Utilize the Windows OLE capabilities
- Use Autolisp, Diesel , and ARX (AutoCAD Runtime Extension) programming languages to perform calculations, automate repetitive tasks, and create new AutoCAD commands
- Generate 3D solid models and extract engineering data

These customization capabilities coupled with the already impressive graphics creation and editing capabilities of AutoCAD made it well suited for the development of CROBOTS. In addition, since AutoCAD is the accepted industry standard for CAD on the PC, the time and cost that students, educators, robot designers, and robot programmers incur to become proficient in the use of CROBOTS and relevant AutoCAD commands should be reduced. Students of robotics should also benefit further since they may acquire new AutoCAD skills, such as customization, that potential employers may desire. In addition, since most students have already used AutoCAD for Engineering Graphics courses, its use in robotics courses will allow them to focus on the learning of key robotics concepts and not the learning of a new graphics software package.

5.4.2 The Template Drawing File

Templates are drawing files with pre-established settings for new drawings. These files allow users to pre-define defaults for numerous drawing parameters including layers, text styles, dimension styles, symbol libraries, and a variety of system variables which control the performance of AutoCAD. The CROBOTS template, revsetup.dwt, pre-defines the layer structure and viewport setup as shown in figure 21. This layer setup allows the automatic selection of the manipulator geometry in a number of CROBOTS AutoLisp functions. The viewport setup is pre-defined to automatically provide the user with a top view display, a front view display, and a 3D view display of the manipulator.

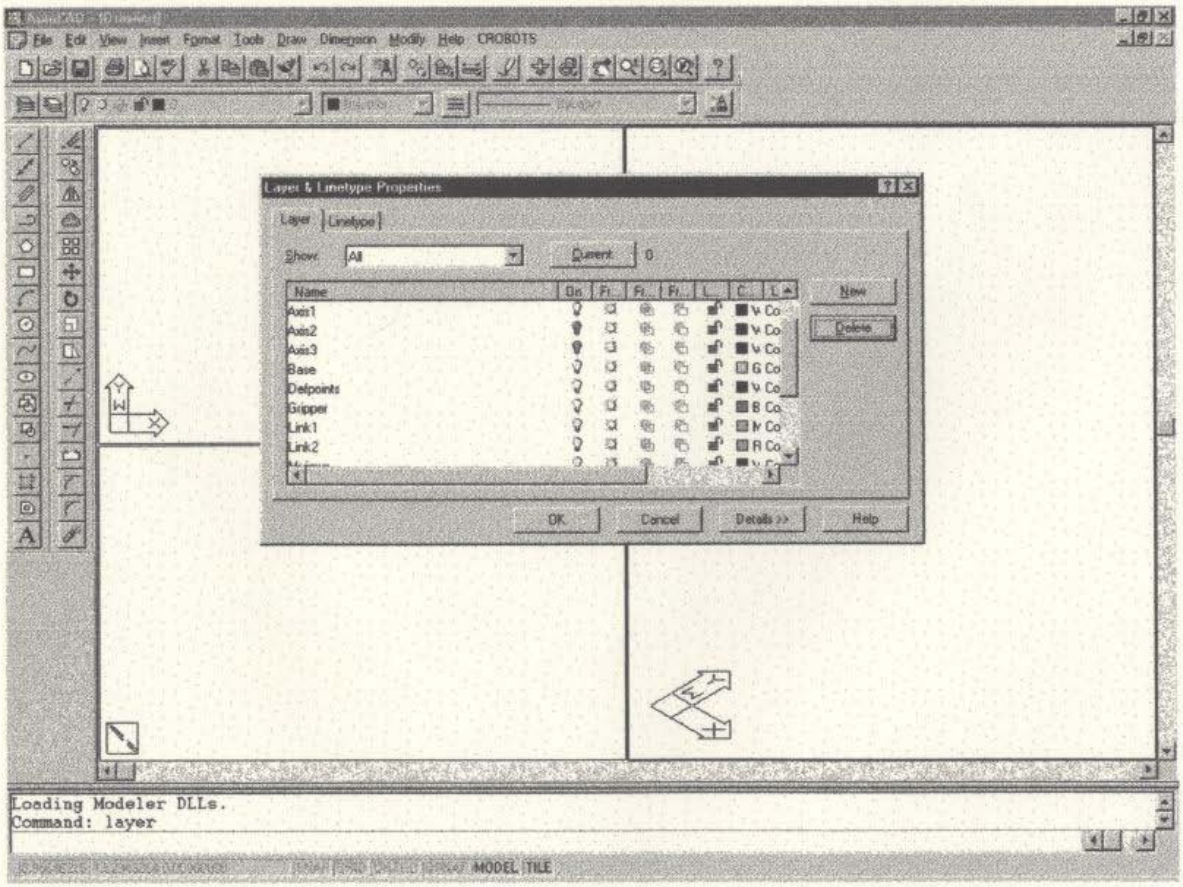

Figure 21: Layer Dialogue Box for Template Revsetup.dwt

5.4.3 The Modified AutoCAD Menu File

The AutoCAD main menu file, acad.mnu, is an ASCII text file that can be edited using a suitable text editor to provide custom menu options. The user is able to develop custom pull-down menus, image menus, menu toolbars, button menus, and the screen menu by editing the appropriate section of acad.mnu.

The CROBOTS functions appear as pull-down menu options in a new pull-down menu that has been created by modifying the standard AutoCAD menu file. Two modified menu files, crobots1.mnu and crobots2.mnu, were created to provide the CROBOTS menu interface. A partial listing of the main CROBOTS menu, acadrob1.mnu, is presented in Appendix A.

The main CROBOTS menu file, acadrob1.mnu, includes the following additions to provide the new pull-down menu option:

```
***POP11***
ID_CROBOTS   [CROBOTS]
ID_Create    [&Create New Robot Model...]^c^c$I=*ID_Robot
                                          $I=ACAD.image_robot
                                          $I=ACAD.*
ID_Home      [&Home]^c^c^p(if (not home) (load "home"))(home);^p
ID_TEACH     [->&Simulate IK]
                  [&Teach]^c^c^p(if (not ikrevo) (load
                                 "ikrevo4"));ikrevo;^p
                  [<-&Run Cycle]^c^c^p(if (not playback) (load
                                       "playback"))(playback);^p
ID_Simulate [&Simulate Controller]^c^c^p(if (not rcontrol)(load
                                       "rcontrol"))(rcontrol);^p
ID_Load      [~Load CROBOTS]^c^cmenu;acadrob1;^p(if (not lcrobots)(load
                                 "lcrobots"))(lcrobots);^p
ID_Unload    [&Unload Robot Modeler]^c^cmenu;crobots2;
```

The ***POP11*** header identifies this menu definition as pull-down menu number 11 and, accordingly, it appears in this position in AutoCAD.

The [CROBOTS] label is used to identify the pull-down option on the pull-down menu bar. A similar approach is used to label menu options which appear when CROBOTS is selected.

The [&Create New Robot Model] listing essentially calls the image menu from which users select the robot configuration type. The image menu listing shown below relies on previously created AutoCAD slide files to generate the robot configuration images which appear to the user.

```
**image_robot
[Select Robot Type]
[revolute,Revolute]^c^c^p(if (not revcreat) (load
"revcreat"))(princ);(revcreat)^p;
[~cartesia,Cartesian]
[~cylindri,Cylindrical]
[~spherica,Spherical]
[~scara,Scara]
```

The [&Home] listing relies on a menu macro to load, if necessary, and activate the AutoLisp function which executes the homing of the robot. The conditional statement (if (not home) (load "home")) first checks to see if the Autolisp function, home.lsp, is loaded with the (not home) AutoLisp expression. If it is not loaded, the (load "home") expression loads the function; otherwise this expression is ignored. Once loading is verified, the function is activated with the (home) expression. A similar approach is used for the remaining CROBOTS pull-down menu options.

5.4.4 The Custom AutoLisp Functions

Custom Autolisp functions were created to enable the menu options "Create New Robot Model", "Teach", "Run Cycle", and "Simulate Controller". In addition, other functions were created to aid in system management. Appendix B includes a complete listing of all related AutoLisp functions.

Create New Robot Model

The "Create New Robot Model" function enables the automatic generation of the geometry of a revolute manipulator following user input of the base height and the link lengths. The AutoLisp file, revcreat.lsp, was developed for this purpose. A complete listing of this file is presented as Appendix B.0. The function sequence consists of updating AutoCAD system variables, prompting the user, generating the robot model, establishing rotation axes, and adjusting the viewports display.

Home

The "Home" function enables the automatic repositioning of the manipulator to a pre-defined reference position. This home position is important for the proper execution of the "Simulate IK" and "Simulate Controller" menu options. The AutoLisp file, home.lsp, was developed for this purpose. A complete listing of this file is presented as Appendix B.1. This function involves updating AutoCAD system variables, selection of the robot links, establishing rotation axes, defining the home position, computation of the IK solution, robot repositioning and output of link angles.

Teach

The "Teach" function enables the user to select and record TCP locations to which the robot is automatically repositioned. The AutoLisp file, ikrevo.lsp, was developed for this purpose. A complete listing of this file is presented in Appendix B.2. The function sequence consists of opening an ASCII file for output, prompting the user, selection of the robot links, computation of the IK solution, robot repositioning, and output of link angles.

Run Cycle

The "Run Cycle" function enables the user to simulate the point to point positioning of the robot through the series of points previously recorded using "Teach". The AutoLisp file, playback.lsp, was developed for this purpose. A complete listing of this file is presented in Appendix B.3. The function operation consists of prompting the user, opening the "teach" ASCII file, robot repositioning, and output of link angles.

Simulate Controller

The "Simulate Controller" function enables the user to simulate a computed torque robot controller that may employ, if desired, an auxiliary control signal based on PID, Relay and ZOH control modes. The AutoLisp file, control.lsp, was developed for this purpose. A complete listing of this file is presented in Appendix B.4. The function operation is depicted in flowchart form in figure 22.

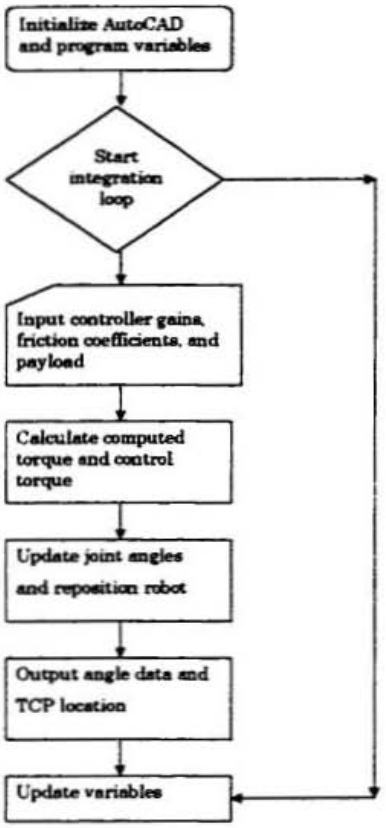

Figure 22: Program Flowchart for Simulate Controller

Chapter 6

6.0 Conclusions and Observations

The CROBOTS functions "Create New Robot Model", "Teach", "Run Cycle" and "Simulate Controller" are the primary functions which may be used by robot designers, educators and students to aid in the design and motion planning of robots. It should be noted, however, that proficiency in the use of the standard AutoCAD tools is a necessary pre-requisite to effectively using these custom tools to assist with these tasks.

The "Create New Robot Model" function is a tool which can facilitate the engineering analysis of robots and improve user productivity. Creating a model requires only input of the manipulator dimensions and users thereby avoid the time and cost associated with building the model from basic AutoCAD entities. Secondly, since the model is constructed from 3D solids, important engineering analysis data, such as centroid location and moments of inertia, can be easily extracted using the standard AutoCAD inquiry tools. Finally, the use of AutoCAD blocks enables users to change the geometry of the model generated by simply modifying the drawing files that constitute each block.

The " Teach" function provides users with a tool that can be used to develop preliminary operation cycle designs before the online programming is done. A 3D model of the workspace can be created using the standard AutoCAD toolkit and the robot's interaction with this environment during trajectory following can then be tested using this function. In addition, the TCP position coordinates that are recorded in an ASCII text file can be used to reduce online programming time since these may be downloadable as a point array to the actual robot controller hardware. This can reduce the online programming time by limiting the programmer's task to fine positioning adjustments.

The "Run Cycle" function enables users to "playback" the manipulator repositioning through the series of recorded points and, subsequently, generate a 3D graphical simulation for the operation cycle. This can be used to optimize workspace layout in the field before online programming begins.

The "Simulate Controller" function provides users with a tool that can be used to test the robot performance for proposed controller designs. The effect of changing controller parameters such as PID gains and sampling rates can be tested using this function. Similarly, the effect of adding or removing a particular control mode can be tested. Furthermore, changes in payload and the resulting effect on robot performance can be assessed using this function. Perhaps, the most significant benefit of using this function is that it enables users to visually observe the manipulator response within its environment. This can reduce the time taken to analyse the proposed controller's performance since the time history graphs of torque, error and so on that are typically used for this purpose are not necessary at the preliminary analysis stage.

During testing of CROBOTS, the software proved to function as designed. The primary objective of the program design was to produce a 3D graphical simulation tool which would facilitate the design and motion planning of robots. Accordingly, the functionality of CROBOTS is best observed through actual use of CROBOTS on a PC.

The overall effectiveness of this software will depend on its application. Industry, educators and students may all benefit from the use of AutoCAD as the core program for a number of reasons. Firstly, since AutoCAD is the industry standard for the PC, a number of users already have access to this CAD package. AutoCAD is a standard component of the engineering graphics curriculum for a number of educational institutions and, accordingly, the cost required to implement the use of CROBOTS should be less than that required for commercial software. Educational institutions may improve the delivery of robotics related curriculum through the use of CROBOTS. Since many students already have the pre-requisite skills in AutoCAD, they will not be required to learn new software and should, therefore, be able to focus on the learning of robotics concepts. In addition, CROBOTS can easily be enhanced through the development of new AutoLisp functions; students should benefit from the acquisition of the programming skills required to achieve this.

Chapter 7

7.0 Recommendations

As previously discussed, CROBOTS offers a number of potential benefits to users including industry, educators, and students. Based on the observations made throughout the development and testing of the software, however, the following recommendations are made:

(1) CROBOTS should be expanded to include a function which allows users to specify a variety of end effectors which are typically used in industry. This will require the expansion of the DK, IK, and dynamics mathematical models to include the orientation of the end effector. The dynamics model should also be enhanced to include the link mass and associated inertia terms;

(2) CROBOTS should be expanded to include an extensive error trapping routine to "clean up" the user interface and retain system parameters in the event of a function error;

(3) CROBOTS AutoLisp functions should be optimized from a programming point of view. That is, similar code which appears in several functions should be converted to a generalized subroutine that is available to all functions;

(4) CROBOTS should be enhanced to allow the creation of Flics or AVI files so that graphical simulations can be recorded and replayed at a later time. This may be achieved by integrating AutoCAD with an "off the shelf" software such as HyperCAM which has this capability;

(5) CROBOTS should be expanded to allow modeling of all robot configuration types. This will again require generalization of the programming code to ensure programming efficiency;

(6) CROBOTS should be expanded to enable the recording of relevant production data such as operation cycle time;

(7) CROBOTS should be enhanced to provide users with a task level programming interface for robot motion planning that emulates those typically used in industry;

(8) CROBOTS should be tested in an educational setting to determine its robustness and to identify need for additional functionality;

(9) CROBOTS should be enhanced to allow improved offiine programming tools. This will require determining how the actual robot controller software may be interfaced with AutoCAD;

References

An, C. et al, 1988, *Model Based Control of a Robot Manipulator*, MIT Press, USA.

Andeen, G.B., 1988, *Robot Design Handbook*, McGraw Hill, New York.

Arimoto, S., 1996, *Control Theory of Non-Linear Mechanical Systems*, Clarendon Press, Oxford.

AutoDesk, 1997, *AutoCAD R14 User's Guide*, AutoDesk Inc., USA.

Bateson, R., 1996, *Introduction to Control System Technology*, Prentice Hall, New Jersey.

Bejczy, A. K., 1974, "Robot Arms Dynamics and Control", TM:33-69, Jet Propulsion Laboratory.

Biekert, R. et al, 1991, *CIM Technology Fundamentals and Applications*, The Goodheart Wilcox Company, Inc.,Illinois.

Borrel, P. et al., 1982, "The Robotics Facilities Available in CAD/CAM CATIA System", *Developments in Robotics*, December.

Canudas de Wit, C. et al, 1996, *Theory of Robot Control*, Springer, London.

Critchlow, A., 1985, *Introduction to Robotics*, MacMillan, New York.

Denavit, J., and Hartenberg, R. B., 1955, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices", ASME, *Journal of Applied Mechanics*, vol. 23, pp. 215-221.

References (Cont.'d)

Freund, E., 1982, "Fast Non-linear Control with Arbitrary Pole-Placement for Industrial Robots and Manipulators", *International Journal of Robotics Research*, Vol. 1, pp. 65-78.

Greenwood, F., 1988, *Introduction to Computer Integrated Manufacturing*, HBJ, New York.

Hinchey, M. J., 1994, "Numerical Simulation for the Puma Arm", *Professional Development Series*, CSME, Montreal.

Hornick, M., and Ravani, B., 1986, "Computer Aided Offline Planning and Programming of Robot Motion", *The International Journal of Robotics Research*, Vol. 4, pp. 18-31.

Korn, G.A., 1978, *Digital Continuous System Simulation*, Prentice Hall, New Jersey.

Lewis, F. et al, 1993, *Control of Robot Manipulators*, MacMillan, New York.

National Robot Society, 1994, Annual Statistics

Paul, R. P., 1981, *Robot Manipulators: Mathematics, Programming and Control*, Cambridge, MIT Press.

Pieper, D. L., 1968, "The Kinematics of Manipulators under Computer Control", *Stanford Artificial Intelligence Project*, Stanford, CA, Memo.

Rivin, E., 1988, *Mechanical Design of Robots*, McGraw Hill, New York.

Ryan, D., 1994, *Robotic Simulation*, CRC Press, London.

References (Cont.'d)

Sanjib, D., 1991, "Offline Programming of Robots Using CAD for Geometrical Information", Thesis, Lamar University – Beaumont.

Takegaki, M., and Arimoto, S., 1981, "An Adaptive Trajectory Control for Manipulators", *International Journal of Control*, Vol. 34, pp. 219-230.

Vukobratovic, M., 1982, *Scientific Fundamentals of Robotics*, Vol. 1 and 2, Springer-Verlag, Berlin.

Whitney, D. E., 1969, "Resolved Motion Rate Control of Manipulators and Human Prostheses", *IEEE Transactions on Man-Machine Systems*, Vol. 10, pp. 47-53.

Wu, C., 1984, "A Kinematic CAD Tool for the Design and Control of a Robot Manipulator", *The International Journal of Robotics Research*, Vol. 3, pp. 58-67.

Appendix A

A.1 CROBOTS1.mnu modified menu file

CROBOTS1.mnu

```
***POP11
ID_CROBOTS  [CROBOTS]
ID_Create   [&Create New Robot Model...]^c^c$I=*ID_Robot
$I=ACAD.image_robot $I=ACAD.*
ID_Home     [&Home]^c^c^p(if (not home) (load "home"))(home);^p
ID_TEACH    [->&Simulate IK]
                  [&Teach]^c^c^p(if (not ikrevo) (load
"ikrevo4"));ikrevo;^p
                  [<-&Run Cycle]^c^c^p(if (not playback) (load
"playback"))(playback);^p
ID_Simulate [&Simulate Controller]^c^c^p(if (not wilma7)(load
"wilma7"))(wilma7);^p
ID_Load     [~Load CROBOTS]^c^cmenu;acadrob1;^p(if (not lcrobots)(load
"lcrobots"))(lcrobots);^p
ID_Unload   [&Unload Robot Modeler]^c^cmenu;acadrob2;


**image_robot
[Select Robot Type]
[revolute,Revolute]^c^c^p(if (not revcreat) (load
"revcreat"))(princ);(revcreat)^p;
[~cartesia,Cartesian]
[~cylindri,Cylindrical]
[~spherica,Spherical]
[~scara,Scara]
```

Appendix B

Listing of Custom AutoLisp Functions

B.0 Revcreat.lsp

```
(defun revcreat ()
;
;       SET SYSTEM VARIABLES
;
        (setq cmdecho (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (setq pdmode (getvar "pdmode"))
        (setvar "pdmode" 0)
        (setq cvport (getvar "cvport"))
        (setq isolines (getvar "isolines"))
        (setvar "isolines" 20)
        (setq facetres (getvar "facetres"))
        (setvar "facetres" 2)
;
;       INPUT GEOMETRY
;
        (princ " Input Robot Geometry ....")(terpri)
        (command "layer" "s" "base" "")
        (setq ptbase (list 0.0 0.0 0.0))
        (setq gg (getdist "Base Height (m): "))
        (command "insert" "base" ptbase gg "" "0")
        (princ)
        (command "layer" "s" "link1" "")
        (setq hh (getdist "Length of Link 1 (m): "))
        (command "insert" "link1" ptbase hh "" "0")
        (princ)
        (command "layer" "s" "link2" "")
        (setq ee (getdist "Length of Link 2 (m): "))
        (setq pt12 (list hh 0.0 0.0))
        (command "insert" "link2" pt12 ee "" "0")
        (princ)
        (setq tcp (list (+ ee hh) 0.0 0.0))
;
;       DRAW CONTROL AXES
;
        (command "layer" "s" "tcp" "")
        (command "line" tcp "@0,-0.005" "")
        (command "layer" "s" "axis2" "")
        (command "line" ptbase "@0,-0.005" "")
        (command "layer" "s" "axis3" "")
        (command "line" pt12 "@0,-0.005" "")
        (command "layer" "s" "0" "")
        (command "layer" "off" "tcp" "")
        (command "layer" "off" "axis2" "")
        (command "layer" "off" "axis3" "")
;
;       ADJUST VIEWPORT ZOOMS
;
        (command "zoom" "e")
        (command "zoom" "0.75x")
        (command "regen")
        (setvar "cvport" 3)
        (command "zoom" "e")
        (command "zoom" "0.75x")
```

```
      (command "regen")
      (setvar "cvport" 4)
      (command "zoom" "e")
      (command "zoom" "0.75x")
      (command "regen")
      (setvar "cvport" 2)
;
;     INITIALIZE JOINT ANGLES
;
      (setq aold 0.0 bold 0.0 cold 0.0)
;
;     RESTORE SYSTEM VARIABLES
;
      (setvar "cmdecho" cmdecho)
      (setq "pdmode" pdmode))
      (setq "cvport" cvport)
      (setq "isolines" isolines)
      (setq "facetres" facetres)
;
      (princ)
);defun
```

B.1 Home.lsp

```
(defun home ()
;
;       Initialize functions
;
        (setq cmdecho (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (if (not rotate3d) (arxload "geom3d"))
        (princ "\nHoming Robot ...")
;
;       Select links
;
        (setq ssbase (ssget "X" '((8 . "base")))
        )
        (setq ssl1 (ssget "X" '((8 . "link1")))
        )
        (setq ssl2 (ssget "X" '((8 . "link2")))
        )
        (setq sstcp (ssget "X" '((8 . "tcp")))
        )
        (setq ssgrip (ssget "X" '((8 . "gripper")))
        )
;       (setq ssaxis1 (ssget "X" '((8 . "axis1")))
;       )
        (setq ssaxis2 (ssget "X" '((8 . "axis2")))
        )
        (setq ssaxis3 (ssget "X" '((8 . "axis3")))
        )
        (command "SELECT" ssbase ssl1 ssl2 ssaxis2 ssaxis3 sstcp "")
        (setq ssrobot (ssget "p")
        )
        (command "SELECT" ssl1 ssl2 ssaxis3 sstcp "")
        (setq sslinks (ssget "p")
        )
        (command "SELECT" ssl2 sstcp "")
        (setq ssl2 (ssget "p")
        )
        (terpri)
;
;       Find Axis Points for Rotation
;
        (setq ax2list (entget (ssname ssaxis2 0))
        )
        (setq ax3list (entget (ssname ssaxis3 0))
        )
        (setq p1 (cdr (assoc 10 ax2list))
        )
        (setq p2 (cdr (assoc 11 ax2list))
        )
        (setq p3 (cdr (assoc 10 ax3list))
        )
        (setq p4 (cdr (assoc 11 ax3list))
        );
;       Specify target point location
;
```

```
      (setq pwx ee
            pwy 0.0
            pwz hh
      )(terpri)
;
;     Calculate anew
;
;if2
      (if (= pwy 0.0)
            (if (= pwx 0.0) (setq arad 0.0)
                  (if (< pwx 0.0) (setq arad pi) (setq arad 0.0)
                  )
            )
            (if (= pwx 0.0)
                  (if (< pwy 0.0)
                        (progn (setq arad (* 1.5 pi)))
                        (progn (setq arad (/ pi 2)))
                  )
                  (if (> pwx 0.0)
                        (progn (setq arad (atan (/ pwy pwx))))
                        (progn (setq arad (+ (atan (/ pwy pwx)) pi)))
                  )
            )
      );if2
      (setq anew (rtd arad)
      )
;
;     Calculate bnew
;
      (setq alpha (+ (* (cos arad) pwx) (* (sin arad) pwy))
      )
      (setq beta (/ (+ (sqr pwz) (sqr alpha) (sqr hh) (- (sqr ee))) (* 2
hh))
      )
      (setq numlbnew (sqrt (+ (sqr pwz) (sqr alpha) (- (sqr beta)))))
      )
      (setq v (/ (+ pwz numlbnew) (+ alpha beta))
      )
      (setq brad (* 2 (atan v))
      )
      (setq bnew (rtd brad)
      )
;
;     Calculate cnew
;
(if (= pwz (+ hh ee)) (setq cnew 0.0)

(progn (setq bcrad (atan (/ (- pwz (* hh (sin brad))) (- alpha (* hh
(cos brad))))))
      )
      (setq bcnew (rtd bcrad)
      )
      (setq crad (- bcrad brad)
      )
      (setq cnew (rtd crad)
      )
);progn
```

```
);if
);else for f1
);if1
;
;     Rotate Links
;
      (setq crot (- cnew cold)
      )
      (setq brot (- bnew bold)
      )
      (setq arot (- anew aold)
      )
      (rotate3d ssl2 p3 p4 crot)
      (rotate3d sslinks p1 p2 brot)
      (rotate3d ssrobot "z" "" arot)
      (setq sstcplist (entget (ssname sstcp 0))
      )
      (setq tcp (cdr (assoc 10 sstcplist))
      )
      (prompt "TCP Location: ")(princ tcp)(terpri)
;
;     RESET JOINT ANGLES
;
      (setq aold anew)(princ)
      (setq bold bnew)(princ)
      (setq cold cnew)(princ)
      (prompt "Robot Homed")
      (setvar "cmdecho" cmdecho)
;
;     Print Output to Screen
;
      (prompt "anew = ")(princ anew)(terpri)
      (prompt "bnew = ")(princ bnew)(terpri)
      (prompt "cnew = ")(princ cnew)(terpri)

(princ)
);defun
```

B.2 Ikrevo.lsp

```
(defun c:ikrevo ()
;
;     Initialize functions
;
      (setq cmdecho (getvar "cmdecho"))
      (setvar "cmdecho" 0)
      (if (not rotate3d) (arxload "geom3d"))
      (if (not rtd) (load "rtd"))
      (if (not dtr) (load "dtr"))
      (if (not sqr) (load "sqr"))
      (setq countik 0.0)
;
;     OPEN FILE FOR RECORDING
;
      (setq fpik (open "teach.txt" "w"))
;
;     BEGIN TEACH LOOP
;
(while
;
;     Specify target point location
;
      (setq p (getpoint "Specify target location or <ENTER> to exit: ")
            pwx (car p)
            pwy (cadr p)
            pwz (caddr p)
      )(terpri)
      (setq countik (+ countik 1))
;
;     Select links
;
      (setq ssbase (ssget "X" '((8 . "base")))
      )
      (setq ssl1 (ssget "X" '((8 . "link1")))
      )
      (setq ssl2 (ssget "X" '((8 . "link2")))
      )
      (setq sstcp (ssget "X" '((8 . "tcp")))
      )
      (setq ssgrip (ssget "X" '((8 . "gripper")))
      )
      (setq ssaxis1 (ssget "X" '((8 . "axis1")))
      )
      (setq ssaxis2 (ssget "X" '((8 . "axis2")))
      )
      (setq ssaxis3 (ssget "X" '((8 . "axis3")))
      )
      (command "SELECT" ssbase ssl1 ssl2 ssaxis2 ssaxis3 sstcp "")
      (setq ssrobot (ssget "p")
      )
      (command "SELECT" ssl1 ssl2 ssaxis3 sstcp "")
      (setq sslinks (ssget "p")
      )
      (command "SELECT" ssl2 sstcp "")
```

```
      (setq ssl2 (ssget "p")
      )
      (terpri)
;
;     Find Axis Points for Rotation
;
      (setq ax2list (entget (ssname ssaxis2 0))
      )
      (setq ax3list (entget (ssname ssaxis3 0))
      )
      (setq p1 (cdr (assoc 10 ax2list))
      )
      (setq p2 (cdr (assoc 11 ax2list))
      )
      (setq p3 (cdr (assoc 10 ax3list))
      )
      (setq p4 (cdr (assoc 11 ax3list))
      )
;
;     Establish Constraints on Motion
;
      (setq k (sqrt (+ (sqr pwx) (sqr pwy)))
      )
      (setq l pwz)
;if1
(if (> (+ (sqr k) (sqr l)) (sqr (+ hh ee)))
(progn (setq anew aold bnew bold cnew cold) (alert "Point Outside
Working Range!!"))
(progn
;
;     Calculate anew
;
;if2
      (if (= pwy 0.0)
            (if (= pwx 0.0) (setq arad 0.0)
                  (if (< pwx 0.0) (setq arad pi) (setq arad 0.0)
                  )
            )
            (if (= pwx 0.0)
                  (if (< pwy 0.0)
                        (progn (setq arad (* 1.5 pi)))
                        (progn (setq arad (/ pi 2)))
                  )
                  (if (> pwx 0.0)
                        (progn (setq arad (atan (/ pwy pwx))))
                        (progn (setq arad (+ (atan (/ pwy pwx)) pi)))
                  )
            )
      );if2
      (setq anew (rtd arad)
      )
;
;     Calculate bnew
;
      (setq alpha (+ (* (cos arad) pwx) (* (sin arad) pwy))
      )
```

```
        (setq beta (/ (+ (sqr pwz) (sqr alpha) (sqr hh) (- (sqr ee))) (* 2
hh))
        )
        (setq numlbnew (sqrt (+ (sqr pwz) (sqr alpha) (- (sqr beta))))
        )
        (setq v (/ (+ pwz numlbnew) (+ alpha beta))
        )
        (setq brad (* 2 (atan v))
        )
        (setq bnew (rtd brad)
        )
;
;       Calculate cnew
;
(if (= pwz (+ hh ee)) (setq cnew 0.0)

(progn (setq bcrad (atan (/ (- pwz (* hh (sin brad))) (- alpha (* hh
(cos brad))))))
        )
        (setq bcnew (rtd bcrad)
        )
        (setq crad (- bcrad brad)
        )
        (setq cnew (rtd crad)
        )
);progn
);if
);else for f1
);if1
;
;       Rotate Links
;
        (setq crot (- cnew cold)
        )
        (setq brot (- bnew bold)
        )
        (setq arot (- anew aold)
        )
        (rotate3d ssl2 p3 p4 crot)
        (rotate3d sslinks p1 p2 brot)
        (rotate3d ssrobot "z" "" arot)
        (setq sstcplist (entget (ssname sstcp 0))
        )
        (setq tcp (cdr (assoc 10 sstcplist))
        )
        (prompt "TCP Location: ")(princ tcp)(terpri)
        (setq aold anew)
        (setq bold bnew)
        (setq cold cnew)
;
;       RECORD POINT
;
        (setq reply (strcase (getstring "Record Point (Y/N)? <Y>: ")))
        (if (or (= reply "") (= reply "Y")) (progn (princ tcp fpik)(princ
(chr 10) fpik)))
);while
(close fpik)
```

```
;      Print Output to Screen
;
       (prompt "anew = ")(princ anew)(terpri)
       (prompt "bnew = ")(princ bnew)(terpri)
       (prompt "cnew = ")(princ cnew)(terpri)

(princ)
);defun
```

B.3 Playback.lsp

```
(defun playback ()
(prompt "Running Cycle ....")
;
;      Initialize functions
;

       (if (not rotate3d) (arxload "geom3d"))
       (if (not rtd) (load "rtd"))
       (if (not dtr) (load "dtr"))
       (if (not sqr) (load "sqr"))
       (setq delay (* 1000 (getint "Input Delay Time in Seconds: ")))
;
;      Start Loop
;
       (setq n 1)
       (setq fp (open "teach.txt" "r"))
(while (< n countik)
;      Select links
;
       (setq ssbase (ssget "X" '((8 . "base")))
       )
       (setq ssl1 (ssget "X" '((8 . "link1")))
       )
       (setq ssl2 (ssget "X" '((8 . "link2")))
       )
       (setq sstcp (ssget "X" '((8 . "tcp")))
       )
       (setq ssgrip (ssget "X" '((8 . "gripper")))
       )
       (setq ssaxis1 (ssget "X" '((8 . "axis1")))
       )
       (setq ssaxis2 (ssget "X" '((8 . "axis2")))
       )
       (setq ssaxis3 (ssget "X" '((8 . "axis3")))
       )
       (command "SELECT" ssbase ssl1 ssl2 ssaxis2 ssaxis3 sstcp "")
       (setq ssrobot (ssget "p")
       )
       (command "SELECT" ssl1 ssl2 ssaxis3 sstcp "")
       (setq sslinks (ssget "p")
       )
       (command "SELECT" ssl2 sstcp "")
       (setq ssl2 (ssget "p")
       )
       (terpri)
;
;      Find Axis Points for Rotation
;
       (setq ax2list (entget (ssname ssaxis2 0))
       )
       (setq ax3list (entget (ssname ssaxis3 0))
       )
       (setq p1 (cdr (assoc 10 ax2list))
       )
```

```
      (setq p2 (cdr (assoc 11 ax2list))
      )
      (setq p3 (cdr (assoc 10 ax3list))
      )
      (setq p4 (cdr (assoc 11 ax3list))
      );
;     Specify target point location
;
;
      (setq pt (read (read-line fp)))
      (princ pt)
      (setq pwx (car pt)
            pwy (cadr pt)
            pwz (caddr pt)
      )
      (terpri)
      (setq n (+ 1 n)
      )
      (princ)
;
;
;     Establish Constraints on Motion
;
      (setq k (sqrt (+ (sqr pwx) (sqr pwy)))
      )
      (setq l pwz)
;if1
(if (> (+ (sqr k) (sqr l)) (sqr (+ hh ee)))
(progn (setq anew aold bnew bold cnew cold) (alert "Point Outside
Working Range!!"))
(progn
;
;     Calculate anew
;
;if2
      (if (= pwy 0.0)
            (if (= pwx 0.0) (setq arad 0.0)
                  (if (< pwx 0.0) (setq arad pi) (setq arad 0.0)
                  )
            )
            (if (= pwx 0.0)
                  (if (< pwy 0.0)
                        (progn (setq arad (* 1.5 pi)))
                        (progn (setq arad (/ pi 2)))
                  )
                  (if (> pwx 0.0)
                        (progn (setq arad (atan (/ pwy pwx))))
                        (progn (setq arad (+ (atan (/ pwy pwx)) pi)))
                  )
            )
      );if2
      (setq anew (rtd arad)
      )
;
;     Calculate bnew
;
      (setq alpha (+ (* (cos arad) pwx) (* (sin arad) pwy))
```

```
      )
      (setq beta (/ (+ (sqr pwz) (sqr alpha) (sqr hh) (- (sqr ee))) (* 2
hh))
      )
      (setq numlbnew (sqrt (+ (sqr pwz) (sqr alpha) (- (sqr beta)))))
      )
      (setq v (/ (+ pwz numlbnew) (+ alpha beta))
      )
      (setq brad (* 2 (atan v))
      )
      (setq bnew (rtd brad)
      )
;
;     Calculate cnew
;
(if (= pwz (+ hh ee)) (setq cnew 0.0)

(progn (setq bcrad (atan (/ (- pwz (* hh (sin brad))) (- alpha (* hh
(cos brad))))))
      )
      (setq bcnew (rtd bcrad)
      )
      (setq crad (- bcrad brad)
      )
      (setq cnew (rtd crad)
      )
);progn
);if
);else for f1
);if1
;
;     Rotate Links
;
      (setq crot (- cnew cold)
      )
      (setq brot (- bnew bold)
      )
      (setq arot (- anew aold)
      )
;     (prompt "alpha =")(princ alpha)(terpri)
;     (prompt "beta =")(princ beta)(terpri)
      (prompt "arot =")(princ arot)(terpri)
      (prompt "brot =")(princ brot)(terpri)
      (prompt "crot =")(princ crot)(terpri)
      (rotate3d ssl2 p3 p4 crot)
      (rotate3d sslinks p1 p2 brot)
      (rotate3d ssrobot "z" "" arot)
      (setq sstcplist (entget (ssname sstcp 0))
      )
      (setq tcp (cdr (assoc 10 sstcplist))
      )
      (prompt "TCP Location: ")(princ tcp)(terpri)
      (setq aold anew)(princ)
      (setq bold bnew)(princ)
      (setq cold cnew)(princ)
      (command "delay" delay)
```

```
;
;      Print Output to Screen
;
       (prompt "anew = ")(princ anew)(terpri)
       (prompt "bnew = ")(princ bnew)(terpri)
       (prompt "cnew = ")(princ cnew)(terpri)
);while
(close fp)
(princ)
);defun
```

B.4 Control.lsp

```
;      Numerical Simulation of the revolute manipulator
;      Based on Lagrangian Energy Method Equations of Motion
;      Assumes link weights are negligible and payload is
;      a concentrated mass at wrist joint.
;      PID + Computed Torque + Relay + ZOH Control is employed to
;      provide drive torques.;
;      Written by: John O'Leary
;      Date:       November 23, 1997
;
;      Key Program Variables Include:
;
;      Old Joint Angles          aold, bold, cold
;      New Joint Angles          anew, bnew, cnew
;      Command Joint Angles      acom, bcom, ccom
;      Old Joint Rates           uold, vold, wold
;      New Joint Rates           unew, vnew, wnew
;      Drive torques             at, bt, ct
;
;
;      Data Input Includes:
;
;      Command Joint Angles      acom,bcom,ccom
;      Proportional Gains        pgb,pgs,pge
;      Derivative Gains          dgb,dgs,dge
;      Integral Gains            igb,igs,ige
;      Link Lengths              gg,hh,ee
;      Number of Cycles          nit
;      Plot Steps                nip
;      Payload Mass              skg
;      Joint Friction            wet, dry
;      Time step                 delt
;
;      REVOLUTE FUNCTION
;
(defun control ()
;
;      INITIALIZE VALUES
;
(setq uold 0.0
      vold 0.0
      wold 0.0
      at 0.0
      bt 0.0
      ct 0.0
      id 0
      propa 0.0
      deriva 0.0
      inta 0.0
      relaya 0.0
      propaa 0.0
      derivaa 0.0
      intaa 0.0
      relayaa 0.0
      propb 0.0
```

```
        derivb 0.0
        intb 0.0
        relayb 0.0
        propbb 0.0
        derivbb 0.0
        intbb 0.0
        relaybb 0.0
        propc 0.0
        derivc 0.0
        intc 0.0
        relayc 0.0
        propcc 0.0
        derivcc 0.0
        intcc 0.0
        relaycc 0.0
        count 0
        plot 0
        band 0.034
        errsuma 0.0
        errsumb 0.0
        errsumc 0.0
        arotsum 0.0
        brotsum 0.0
        crotsum 0.0
)
;
;       INPUT PROPORTIONAL GAINS OR ACCEPT DEFAULT
;
(princ)(terpri)
;
(if pgb (progn     (prompt "\nCurrent Proportional Gains (Base, Shoulder,
Elbow):  ")
        (princ pgb)(prompt ", ")(princ pgs)(prompt ", ")(princ pge)
        (setq progains (strcase (getstring "\nUpdate Proportional Gain?
<Y/N>: ")))
        )
        (progn       (setq pgb (getreal "\nInput Proportional Base Gain
(N.m/rad): "))
             (setq pgs (getreal "\nInput Proportional Shoulder Gain
(N.m/rad): "))
             (setq pge (getreal "\nInput Proportional Elbow Gain
(N.m/rad): "))
        );progn
);if
(if (= progains "Y") (progn   (setq pgb (getreal "\nInput Proportional
Base Gains (N.m/rad): "))
                         (setq pgs (getreal "\nInput Proportional
Shoulder Gain (N.m/rad): "))
                         (setq pge (getreal "\nInput Proportional Elbow
Gain (N.m/rad): "))
                    );progn
);if
;
;       INPUT DERIVATIVE GAINS OR ACCEPT DEFAULT
;
(princ)(terpri)
;
```

```
(if dgb (progn (prompt "\nCurrent Derivative Gains (Base, Shoulder,
Elbow):  ")
      (princ dgb)(prompt ", ")(princ dgs)(prompt ", ")(princ dge)
      (setq dergains (strcase (getstring "\nUpdate Derivative Gains?
<Y/N>: ")))
      )
      (progn      (setq dgb (getreal "\nInput Derivative Base Gains: "))
            (setq dgs (getreal "\nInput Derivative Shoulder Gain: "))
            (setq dge (getreal "\nInput Derivative Elbow Gain: "))
      );progn
);if
(if (= dergains "Y") (progn   (setq dgb (getreal "\nInput Derivative
Base Gain (N.m.s/rad): "))
                        (setq dgs (getreal "\nInput Derivative Shoulder
Gain (N.m.s/rad): "))
                        (setq dge (getreal "\nInput Derivative Elbow
Gain(N.m.s/rad) : "))
                  );progn
);if
;
;     INPUT INTEGRAL GAINS OR ACCEPT DEFAULT
;
(princ)(terpri)
;
(if igb (progn (prompt "\nCurrent Integral Gains (Base, Shoulder,
Elbow):  ")
      (princ igb)(prompt ", ")(princ igs)(prompt ", ")(princ ige)
      (setq intgains (strcase (getstring "\nUpdate Integral Gains?
<Y/N>: ")))
      )
      (progn      (setq igb (getreal "\nInput Integral Base Gains: "))
            (setq igs (getreal "\nInput Integral Shoulder Gain: "))
            (setq ige (getreal "\nInput Integral Elbow Gain: "))
      );progn
);if
(if (= intgains "Y") (progn   (setq igb (getreal "\nInput Integral Base
Gains (N.m/rad): "))
                        (setq igs (getreal "\nInput Integral Shoulder
Gain (N.m/rad): "))
                        (setq ige (getreal "\nInput Integral Elbow Gain
(N.m/rad): "))
                  );progn
);if
;
;     INPUT RELAY GAINS OR ACCEPT DEFAULT
;
(princ)(terpri)
;
(if rgb (progn (prompt "\nCurrent Relay Gains (Base, Shoulder, Elbow):
")
      (princ rgb)(prompt ", ")(princ rgs)(prompt ", ")(princ rge)
      (setq rgains (strcase (getstring "\nUpdate Relay Gains? <Y/N>:
")))
      )
      (progn      (setq rgb (getreal "\nInput Relay Base Gains: "))
            (setq rgs (getreal "\nInput Relay Shoulder Gain: "))
            (setq rge (getreal "\nInput Relay Elbow Gain: "))
```

```
      );progn
);if
(if (= rgains "Y") (progn      (setq rgb (getreal "\nInput Relay Base
Gains (N.m): "))
                        (setq rgs (getreal "\nInput Relay Shoulder Gain
(N.m): "))
                        (setq rge (getreal "\nInput Relay Elbow Gain
(N.m): "))
                  );progn
);if
;
;     INPUT ZERO ORDER HOLD OR ACCEPT DEFAULT
;
(princ)(terpri)
;
(setq zoh (strcase (getstring "Use Zero Order Hold? (Y/N): ")))
(if (= zoh "Y")
(progn      (if ns      (progn (prompt "\nCurrent Rates (Sampling,
Controller):  ")
      (princ ns)(prompt ", ")(princ nz)
            (setq zgains (strcase (getstring "\nUpdate Zero Order Hold
Values? <Y/N>: ")))
           );progn
           (progn      (setq ns (getreal "Input Sampling Rate: "))
           (setq nz (getreal "Input Controller Rate: "))
           );progn
      );if
      (if (= zgains "Y")      (progn      (setq ns (getreal "\nInput
Sampling Rate: "))
                        (setq nz (getreal "\nInput Controller Rate: "))
                        );progn
      );if
);progn
(progn (setq nz 1) (setq ns 1))
);if
;
;     INPUT FRICTION COEFFICIENTS
;
(setq wet (getreal "\nInput Wet Friction Value (N.m.s/rad): ")
      dry (getreal "\nInput Dry Friction Value (N.m.s/rad): ")
)
;
;     INPUT # OF CYCLES, PLOT CONTROL, AND ANGLE STEP
;
(setq nit (getreal "\n Input Number of Cycles: ")
      delt (getreal "\n Input Cycle Step Angle Increment (rad/s): ")
      nip (getreal "\n Input Plot Skip Value: ")
)
;
;     INPUT COMMAND ANGLES
;
(setq acom (getreal "\n Input Commanded Base Angle (degrees): ")
      bcom (getreal "\n Input Commanded Shoulder Angle (degrees): ")
      ccom (getreal "\n Input Commanded Elbow Angle (degrees): ")
)
```

```
;
;
;
;     INPUT PAYLOAD
;
(setq skg (getreal "\n Input Payload (kg): "))
;
;     PRESET DATA
;
(setq gravity 9.81
      atmax 2000.0
      btmax 2000.0
      ctmax 2000.0
)
;
;     CONVERT ANGLES TO RADIANS
;
(setq aold (dtr aold)
      bold (dtr (- 90.0 bold))
      cold (dtr (- cold))
)
(setq acom (dtr acom)
      bcom (dtr bcom)
      ccom (dtr ccom)
)
;
;     OPEN OUTPUT FILE
      (setq fpb (open "c:tbase.txt" "w"))
      (setq fps (open "c:tshoulde.txt" "w"))
      (setq fpe (open "c:telbow.txt" "w"))
;
;     INTEGRATION LOOP FOR EQUATIONS OF MOTION
;
(while (/= count nit)
      (setq id (+ id 1))
      (if (= id ns)
      (progn (setq erra (- acom aold) errb (- bcom bold) errc (- ccom
cold))
      );progn
      );if
      (setq count (+ count 1.0)
      )
      (setq plot (+ plot 1.0)
      )
      (setq sgnu 0.0 sgnv 0.0 sgnw 0.0)
      (if (not (= uold 0.0)) (progn (setq sgnu (/ uold (abs uold)))))
      (if (not (= vold 0.0)) (progn (setq sgnv (/ vold (abs vold)))))
      (if (not (= wold 0.0)) (progn (setq sgnw (/ wold (abs wold)))))
;
;     DYNAMICS
;
      (setq one (+ (* hh (sin bold)) (* ee (sin (+ bold cold))))
            two (+ (* hh (cos bold)) (* ee (cos (+ bold cold))))
            aii (* skg (+ (* ee ee) (* hh hh) (* 2.0 ee hh (cos cold))))
            aij (* skg (+ (* ee ee) (* ee hh (cos cold))))
            aji (* skg (+ (* ee ee) (* ee hh (cos cold))))
            ajj (* skg ee ee)
```

```
            akk (* skg one one)
      )
;
;     CONTROL TORQUE
;
(if (= id ns)
(progn      (setq propa (* pgb (- acom aold)))
            (setq propb (* pgs (- bcom bold)))
            (setq propc (* pge (- ccom cold)))
            (setq deriva (* dgb (- uold)))
            (setq derivb (* dgs (- vold)))
            (setq derivc (* dge (- wold)))
            (setq inta (* igb errsuma))
            (setq intb (* igs errsumb))
            (setq intc (* ige errsumc))
            (if (> (- acom aold) band) (progn (setq relaya rgb)))
            (if (> (- bcom bold) band) (progn (setq relayb rgs)))
            (if (> (- ccom cold) band) (progn (setq relayc rge)))
            (if (< (- acom aold) (- band)) (progn (setq relaya (-
rgb))))
            (if (< (- bcom bold) (- band)) (progn (setq relayb (-
rgs))))
            (if (< (- ccom cold) (- band)) (progn (setq relayc (-
rge))))
);progn
);if
(if   (= id nz)
      (progn      (setq propaa propa)
                  (setq propbb propb)
                  (setq propcc propc)
                  (setq derivaa deriva)
                  (setq derivbb derivb)
                  (setq derivcc derivc)
                  (setq intaa inta)
                  (setq intbb intb)
                  (setq intcc intc)
                  (setq relayaa relaya)
                  (setq relaybb relayb)
                  (setq relaycc relayc)
      );progn
      );if
      (setq at (- (+ propaa derivaa intaa relayaa) (+ (* dry sgnu) (*
wet uold))))
      (setq bt (- (+ propbb derivbb intbb relaybb) (+ (* dry sgnv) (*
wet vold))))
      (setq ct (- (+ propcc derivcc intcc relaycc) (+ (* dry sgnw) (*
wet wold))))
;
;     OVERALL TORQUE
;
      (setq bi (+ bt (* skg (+ (* 2.0 ee hh (sin cold) wold vold) (* ee
hh (sin cold) wold wold) (* one two uold uold) (* gravity one))))
            bj (- ct (* skg (- (* ee hh (sin cold) vold vold) (* gravity
ee (sin (+ bold cold))) (* one ee (cos (+ bold cold)) uold uold))))
            bk (- at (* 2.0 skg (+ (* one two uold vold) (* one ee (cos
(+ bold cold)) uold wold))))
      )
```

```
;
;       DETERMINANT
;
        (setq det (- (* aii ajj) (* aij aji)))
              (if (> (abs det) 0.1)
              (progn (setq      ri (/ (- (* ajj bi) (* aij bj)) det)
              rj (/ (- (* aii bj) (* aji bi)) det)
              rk (/ bk akk)
              )
              );progn
              );if
;
;       NEW VALUES
;
        (setq anew (+ aold (* delt uold))
              bnew (+ bold (* delt vold))
              cnew (+ cold (* delt wold))
              unew (+ uold (* delt rk))
              vnew (+ vold (* delt ri))
              wnew (+ wold (* delt rj))
        )
;
;       OUTPUT VALUES
;
        (setq crot (- (rtd (- cnew cold)))
        )
        (setq brot (rtd (- bnew bold))
        )
        (setq arot (rtd (- anew aold))
        )
        (setq arotsum (+ arot arotsum))
        (setq brotsum (+ brot brotsum))
        (setq crotsum (+ crot crotsum))
(while (or (eq plot nip) (eq plot 1.0))
;       (princ "Plot=") (princ plot)(terpri)
        (princ "anew=") (princ (rtd anew)) (prompt " ")(terpri)
;       (princ "aold=") (princ (rtd aold)) (prompt " ")(terpri)
        (princ "bnew=") (princ (rtd bnew)) (prompt " ")(terpri)
;       (princ "bold=") (princ (rtd bold)) (prompt " ")(terpri)
        (princ "cnew=") (princ (rtd cnew)) (prompt " ")(terpri)
;       (princ "cold=") (princ (rtd cold)) (prompt " ")(terpri)
        (princ " ") (terpri)
;
;       UPDATE GRAPHIC POSITION DISPLAY
;
;
;       Find Axis Points for Rotation
;
        (setq ax2list (entget (ssname ssaxis2 0))
        )
        (setq ax3list (entget (ssname ssaxis3 0))
        )
        (setq p1 (cdr (assoc 10 ax2list))
        )
        (setq p2 (cdr (assoc 11 ax2list))
        )
        (setq p3 (cdr (assoc 10 ax3list))
```

```
      )
      (setq p4 (cdr (assoc 11 ax3list))
      )
;
;     DETERMINE ROTATION ANGLES
;
;     (princ "arot=") (princ arot) (terpri)
;     (princ "brot=") (princ brot) (terpri)
;     (princ "crot=") (princ crot) (terpri)
;
;     ROTATE MANIPULATOR
;
      (rotate3d ssl2 p3 p4 crotsum)
      (rotate3d sslinks p1 p2 brotsum)
      (rotate3d ssrobot "z" "" arotsum)
      (setq plot (+ plot 1.0))
      (if (> plot nip) (progn (setq plot 0.0 arotsum 0.0 brotsum 0.0
crotsum 0.0)))
);while
;
;     INCREMENT VALUES
;
      (setq aold anew
            bold bnew
            cold cnew
            uold unew
            vold vnew
            wold wnew
      )
      (if (= id nz)
      (progn      (setq errsuma (+ errsuma (* nz delt erra)))
                  (setq errsumb (+ errsumb (* nz delt errb)))
                  (setq errsumc (+ errsumc (* nz delt errc)))
      );progn
      );if
      (if (= id nz) (setq id 0))
);while
(setq aold (rtd aold)
      bold (- 90.0 (rtd bold))
      cold (rtd (- cold))
)
      (setq sstcplist (entget (ssname sstcp 0))
      )
      (setq tcp (cdr (assoc 10 sstcplist))
      )
      (prompt "TCP Location: ")(princ tcp)(terpri)
;     (close fp)
(princ)
); defun
```