Software for Control and Dynamic Simulation of
UNIMATE PUMA 560 ROBOT

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment of the
Requirements for the Degree
Master of Science

By
Sandeep Anand
Department of Mechanical Engineering
June 1993
Acknowledgements

I take this opportunity to acknowledge the support of my advisor, Dr. Sunil Agrawal, during the course of this research. Working with him has been a delightful, exciting and educating experience for me.

A vote of special thanks to the other members on the committee, Dr. Jay Gunasekera and Dr. Larry Snyder for extending their time and perusal on the thesis.

I would also like to thank Mr. Ronald B. Buck for his help provided during the course of research.

Finally, I would like to deeply thank my family back home for their tremendous support during my stay in the U.S.A..
# Table of Contents

Introduction ................................................................. 7  
  1.1 Background .......................................................... 7  
  1.2 Literature Review .................................................. 9  
  1.3 Motivation of the Present Study ................................. 11  
  1.4 Summary .............................................................. 12  

Forward Kinematics ........................................................ 14  
  2.1 Introduction .......................................................... 14  
  2.2 Forward Kinematics ................................................ 14  
  2.3 Denavit- Hartenberg Representation .............................. 17  

Inverse Kinematics ........................................................ 21  
  3.1 Introduction .......................................................... 21  
  3.2 Solvability ............................................................. 22  
  3.3 Pieper’s Solution ..................................................... 23  

Equations of Motion and Dynamics ...................................... 30  
  4.1 Introduction .......................................................... 30  
  4.2 Lagrangian Formulation of Manipulator Dynamics ............ 30  
  4.3 Equations of Motion ................................................ 32  
  4.4 The Real and Approximate Dynamic Models .................... 36  
  4.5 Dynamic Simulation ................................................ 37  
  4.6 Animation ............................................................. 38  
  4.7 Verification ........................................................... 39  
  4.8 Results ................................................................. 39
D-7 Computer Program to get Joint Motor Torque(s) for Controlling the Robot based on the Approximate Model ........................................ 132
D-8 Computer Program to get Forces(s) on the End Effector based on the Real Model ................................................................. 141
D-9 Computer Program to get Forces(s) on the End Effector based on the Approximate Model ...................................................... 163
D-10 Computer Program for Graphical Simulation of PUMA 560 ........ 180
D-11 Computer Program for Graphical Simulation of Pieper's Solution 204
D-12 The main routines for running the PUMA robot with the Trident Robotics and Research, Inc. interface boards. ......................... 228
D-13 The header file containing defined constants for use when using the interface boards from Trident Robotics and Research, Inc. ...... 235
D-14 An example user-code file for controlling the PUMA robot with the Trident Robotics and Research, Inc. interface boards. .......... 237
D-15 The main header file for utilizing functions provided for accessing the PUMA robot using the interface boards ......................... 242
D-16 File for calibration of the robot using the Trident Robotics cards. 243

Appendix E .......................................................................................... 245
E-1 The Approximate Model ............................................................... 245
Chapter 1
Introduction

1.1 Background

The use of industrial robots became identifiable as unique devices in the 1960's, along with computer aided design (CAD) systems, and computer aided manufacturing (CAM) systems. Presently, the robots characterize the latest trends in automation of the manufacturing process [60].

New disciplines of engineering, such as manufacturing engineering, applications engineering, and knowledge engineering are beginning to deal with the complexity of the field of robotics and the larger area of factory automation. Within a few years, it is possible that robotics engineering will stand on its own as a distinct engineering discipline.

A robot can be considered as a computer controlled industrial manipulator, operating under some degree of autonomy. Such devices are extremely complex electromechanical systems whose analytic description requires advanced methods which present many challenging and interesting research problems. The official definition of such a robot comes from the Robot Institute of America (RIA): A robot is a reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks. The key element in this definition is the reprogrammability of robots. It is the computer brain that gives the robot its utility and adaptability. The so-called robotics revolution is, in fact, part of the larger computer revolution.

There are many other applications of robots in areas where the use of humans is impractical or undesirable. Among these are undersea and planetary exploration, satellite retrieval and repair, the defusing of explosive devices, and work in radioactive environments. Finally, prostheses, such as artificial limbs, are themselves robotic
INTRODUCTION

devices requiring methods of analysis and design similar to those of industrial manipulators [19,25].

This thesis will focus mainly on the kinematics, dynamics and control of Unimation PUMA 560 robot. Figure 1.1 shows the PUMA 560, and Figure 1.2 shows its degrees of joint rotation [68]. On the basis of its geometry, the PUMA 560 robot is a six revolute jointed articulated manipulator [52].

Figure 1.1 The Unimation PUMA 560 (Adapted from [68]).
1.2 Literature Review

Se-Young Oh and Moon-Jeung Joe have developed a dynamic controller for a six degree of freedom manipulator (tested and simulated on PUMA 560) using backpropagation neural networks [51]. H. Chuang has examined model uncertainty and effects on compliant motion in design and simulation of the PUMA robot arm [15]. By adding a roll joint to the shoulder of the PUMA geometry, redundancy resolution has been studied to accomplish singularity avoidance in this human arm like manipulator [13]. The PUMA 560 robot manipulator control hardware structures have been sufficiently modified to facilitate manipulator motion control research [26]. A motion planner has been implemented that plans collision free motions for a PUMA 560 manipulator arm among stationary obstacles [29]. A computer simulation using a PUMA
INTRODUCTION

560 manipulator demonstrates generation of optimal time trajectory for two cooperating robots [14]. A stochastic optimization problem has been formulated to obtain the optimal production speed of robots based on measured geometric and non-geometric errors [45]. Progress in extending "Soar" architecture to tasks that involve interaction with external environments have been reported [38]. For space based robotic applications, a fast computer architecture for the control of robots coupled with closed loop position control and force/torque feedback has been developed [62]. Li and Sankar have presented a scheme for fast inverse dynamic computation in real time robot control [40]. Boudor has developed an exact kinematic model of PUMA 560 [9]. An energy based indirect adaptive controller for robots has also been presented by Serafi and Khalil [24]. The generation of joint space trajectories for robotic manipulators using different curve fitting techniques have been discussed with results applied to PUMA manipulator following circular paths in space [5]. A visual feedback control scheme called image based visual servo has been proposed for manipulators with cameras on their hands (experiments have been validated on PUMA 560) [30]. Dynamic modelling of geared and flexible jointed manipulators (investigated using simulation on PUMA 560) has been presented [47]. The feasibility of applying the principles of direct adaptive control to the industrial trajectory tracking problem was rigorously investigated using PUMA 560 [39]. The trajectory tracking performance of the non-linear feedback controller based on differential geometric control theory was experimentally studied and tested on PUMA 560 arm [66].

A method allowing one to determine the feasibility of displacement trajectories of a class of robot manipulators, namely with a wrist of intersecting axes (with PUMA 560 in particular) has been developed [71]. Singularity redundancy resolution with task priority has been implemented (derivation based on PUMA geometry) using extended Jacobian technique with weighted damped least squares [23]. The discrete time theory that linearizes the nonlinear dynamics of the PUMA 560 robot arm has been systematically developed [27]. A modular hierarchial model targeted for research and development to control robots has also been studied [6]. Li introduced an approach for fast mapping obstacles in the configuration space of the robot to plan collision free paths for industrial robots (Calculation results were proved by graphical simulations of PUMA
560 arm) [41]. The potential for using robotic techniques in the development of a surgeon robot was investigated in a preliminary feasibility study using a six axis PUMA robot [20]. A systematic procedure was presented to efficiently enumerate all possible singular configurations for robotic manipulators [42]. An active end effector based force control system for robotic deburring was successfully implemented using a PUMA 560 robot [10].

1.3 Motivation of the Present Study

We define an off-line programming (OLP) system as a robot programming language which has been sufficiently extended, generally by means of computer graphics, so that the development of robot programs can take place without access to the robot itself. Off-line programming systems are important both as aids in programming present-day industrial automation as well as platforms for robotics research. Numerous issues must be considered in the design of such systems. In this chapter, a discussion of these issues is presented using the design of PUMA 560. The topics include kinematics, dynamic simulation, path planning of the robot and, spatial representation of solids and their graphical representation.

In the development of programming systems for robots, advances in the power of programming techniques seem directly tied to the sophistication of the internal model referenced by the programming language. Early joint space "teach by showing" robot systems employed a limited world model, and there were very limited ways in which the system could aid the programmer in accomplishing a task. Slightly more sophisticated robot controllers included kinematic models so that the system could at least aid the user in moving the joints so as to accomplish Cartesian motions. Robot programming languages (RPLs) evolved which support many different data types and operations which the programmer may use as needed to model attributes of the environment and compute actions of the robot. Some RPLs support world modeling primitives such as affixments, data types for forces and moments, and other features [46].

Although this thesis focuses to some extent on the particular problem of robot
programming, the notion of an OLP system extends to any programmable device on the factory floor. A common argument raised in their favor is that an OLP system will not tie up production equipment when it needs to be reprogrammed, and hence, automated factories can stay in production mode a greater percentage of the time. They also serve as a natural vehicle to tie computer aided design (CAD) data bases used in the design phase of a product's development to the actual manufacturing of the product. In some applications, this direct use of CAD design data can dramatically reduce the programming time required for the manufacturing machinery.

Off-line programming of robots offers other potential benefits which are just beginning to be appreciated by industrial robot users. Some of the problems associated with robot programming are attributed to the fact that an external, physical workcell is manipulated by the robot program. Programming of robots in simulation offers a way of keeping the bulk of the programming work strictly internal to a computer - until the application is nearly complete. Thus, many of the problems peculiar to robot programming tend to diminish.

1.4 Summary

Chapter 2 describes the background on representation of coordinate systems and transformation among various coordinate systems. It leads to the forward kinematics problem, which is to determine the position and orientation of the end-effector or tool given the joint variables. Chapter 3 discusses the problem of inverse kinematics, i.e., given the position and orientation of the end-effector, calculate the possible sets of joint angles which attain this given position and orientation. In order to analyze manipulators in motion, the concept of mapping velocities in joint space to velocities in Cartesian space by the Jacobian matrix is presented in Chapter 4. This Chapter also deals with the forces and moments required to cause motion to a manipulator. Here, we develop techniques and algorithms based on Lagrangian dynamics for deriving the equations of motion for PUMA 560. This aspect of developing equations of motion is used to control the motion of the manipulator. To simulate how the manipulator would move under the
application of a set of actuator torques, the dynamic equations are reformulated so that the acceleration is computed as a function of actuator torques. Chapter 5 is concerned with describing motion of the manipulator in terms of trajectories through space. The conclusions and the scope of future work is presented in Chapter 6.
Chapter 2
Forward Kinematics

2.1 Introduction

Kinematics is the description of motion without regard to the forces that cause it. It deals with the study of position, velocity, acceleration, and higher derivatives of the position variables.

The Unimation PUMA 560 robot is an example of an all rotary joint manipulator with 6 \textit{dof} (i.e., it is a 6R mechanism).

2.2 Forward Kinematics

The forward kinematics problem can be stated as follows: Given the joint variables of the robot, determine the position and orientation of the end-effector. Since each joint has a single degree of freedom, the action of each joint can be described by a single number, i.e., the angle of rotation in the case of a revolute joint. The objective of forward kinematic analysis is to determine the \textit{cumulative} effect of the joint variables.

Suppose a robot has \( n+1 \) links numbered from 0 to \( n \) starting from the base of the robot, which is taken as link 0. The joints are numbered from 1 to \( n \), and \( \hat{z}_i \) is a unit vector along the axis in space about which the links \( i-1 \) and \( i \) are connected. The \( i \)-th joint variable is denoted by \( q_i \). In case of a revolute joint, \( q_i \) is the angle of rotation, and in the case of a prismatic joint, \( q_i \) is the joint translation. Next, a coordinate frame is attached rigidly to each link. To be specific, we choose frames 1 through \( n \) such that the frame \( i \) is rigidly attached to link \( i \). Figure 2.1 illustrates the idea of attaching frames rigidly to links in the case of a PUMA 560 robot. \( T_{i}^{i-1} \) is a homogenous matrix which is defined to transform the coordinates of a point from frame \( i \) to frame \( i-1 \) [7]. The matrix \( T_{i}^{i-1} \) is not constant, but varies as the configuration of the robot is changed. However, the assumption that all joints are either revolute or prismatic means that \( T_{i}^{i-1} \)
Figure 2.1 The PUMA 560 in the Zero Position with Attached Coordinate Frames Shown (Adapted from [3]).
FORWARD KINEMATICS

is a function of only a single joint variable, namely \( q_i \). In other words,

\[
T_{i}^{i-1} = T_{i}^{i-1}(q_i)
\]

The homogenous matrix that transforms the coordinates of a point from frame \( i \) to frame \( j \) is denoted by \( T_{i}^{j} \) (\( i > j \)).

Denoting the position and orientation of the end-effector with respect to the inertial or the base frame by a three dimensional vector \( d_n^0 \) and a 3x3 rotation matrix \( R_n^0 \), respectively, we define the homogenous matrix

\[
T_n^0 = \begin{bmatrix}
R_n^0 & d_n^0 \\
0 & 1
\end{bmatrix}
\]

(2.2)

Then the position and orientation of the end-effector in the inertial frame are given by

\[
T_n^0(q_1, q_2, ..., q_n) = T_{i}^0(q_j) T_{2}^i(q_2) ... T_{n}^{i-1}(q_n)
\]

(2.3)

Each homogenous transformation \( T_{i}^{i-1} \) is of the form

\[
T_{i}^{i-1} = \begin{bmatrix}
R_{i}^{i-1} & d_{i}^{i-1} \\
0 & 1
\end{bmatrix}
\]

(2.4)

Hence

\[
T_{i}^{j} = T_{j}^{j}...T_{i}^{i-1} = \begin{bmatrix}
R_{i}^{j} & d_{i}^{j} \\
0 & 1
\end{bmatrix}
\]

(2.5)
The matrix $R^i_j$ expresses the orientation of frame $i$ relative to frame $j$ ($i > j$) and is given by the rotational parts of the $T^i_j$-matrices ($i > j$) as

$$R^i_j = R^i_{j-1} \cdots R^{i-1}_i$$

(2.6)

The vectors $d^i_j$ ($i > j$) are given recursively by the formula

$$d^i_j = d^i_{j-1} + R^i_{j-1}d^{j-1}_i$$

(2.7)

### 2.3 Denavit- Hartenberg Representation

A commonly used convention for selecting frames of reference in robotic applications is the Denavit-Hartenberg or D-H convention [21]. In this convention each homogenous transformation $T^i_{i-1}$ is represented as a product of "four" basic transformations

$$T^i_{i-1} = \begin{align*}
\text{Rot}(x, \alpha_{i-1}) &\text{Trans}(x, a_{i-1}) \text{Rot}(z, \theta_i) \text{Trans}(z, d_i)
\end{align*}$$

(2.8)

where the notation $\text{Rot}(x, \alpha_{i-1})$ stands for rotation about $\hat{x}_i$ axis by $\alpha_{i-1}$, $\text{Trans}(x, a_{i-1})$ is translation along $\hat{x}_i$ axis by a distance $a_{i-1}$, $\text{Rot}(z, \theta_i)$ stands for rotation about $\hat{z}_i$ axis by $\theta_i$, and $\text{Trans}(z, d_i)$ is the translation along $\hat{z}_i$ axis by a distance $d_i$. 

$$T^i_{i-1} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & c_{\alpha_{i-1}} & -s_{\alpha_{i-1}} & 0 \\
0 & s_{\alpha_{i-1}} & c_{\alpha_{i-1}} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
c_\theta & -s_\theta & 0 & 0 \\
s_\theta & c_\theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
a_{i-1} \\
s_{a_{i-1}} \\
c_{a_{i-1}} \\
1
\end{bmatrix}$$

(2.9)
FORWARD KINEMATICS

\[
T_{i-1}^i = \begin{bmatrix}
  c_{\theta_i} & -s_{\theta_i} & 0 & a_{i-1} \\
  s_{\theta_i}c_{a_{i-1}} & c_{\theta_i}c_{a_{i-1}} & -s_{a_{i-1}} & -s_{a_{i-1}}d_i \\
  s_{\theta_i}s_{a_{i-1}} & c_{\theta_i}s_{a_{i-1}} & c_{a_{i-1}} & c_{a_{i-1}}d_i \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]  

(2.10)

where the four quantities \( \theta_i, a_i, d_i, \alpha_i \) are the parameters of link \( i \) and joint \( i \). Figure 2.2 illustrates the link frames attached so that frame \( \{i\} \) is attached rigidly to link \( i \).

![Figure 2.2 Coordinate Frames Satisfying D-H Assumptions (Adapted from [18])](image)

The various parameters in Eq. (2.10) are given the following names:

- \( a_i \) (length), is the distance from \( \hat{z}_i \) to \( \hat{z}_{i+1} \) measured along \( \hat{x}_i \);
- \( \alpha_i \) (twist), is the angle between \( \hat{z}_i \) and \( \hat{z}_{i+1} \) measured about \( \hat{x}_i \);
- \( d_i \) (offset), is the distance from \( \hat{x}_{i-1} \) to \( \hat{x}_i \) measured along \( \hat{z}_i \); and
- \( \theta_i \) (angle), is the angle between \( \hat{x}_{i-1} \) and \( \hat{x}_i \) measured about \( \hat{z}_i \).
In the usual case of a revolute joint, $\theta_i$ is called the joint variable, and the other three quantities are the fixed link parameters. For the PUMA 560 robot, 18 numbers are required to completely describe the fixed portion of its kinematics. These 18 numbers are in the form of six sets of $(a_i, \alpha_i, d_i)$.

Table 2.1 shows the sets in the form of D-H table for PUMA 560:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\alpha_{i-1}$</th>
<th>$a_{i-1}$</th>
<th>$d_i$</th>
<th>$\theta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\theta_1$</td>
</tr>
<tr>
<td>2</td>
<td>-90</td>
<td>0</td>
<td>$d_2$</td>
<td>$\theta_2$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>$a_2$</td>
<td>$d_3$</td>
<td>$\theta_3$</td>
</tr>
<tr>
<td>4</td>
<td>90</td>
<td>$a_3$</td>
<td>$d_4$</td>
<td>$\theta_4$</td>
</tr>
<tr>
<td>5</td>
<td>-90</td>
<td>0</td>
<td>0</td>
<td>$\theta_5$</td>
</tr>
<tr>
<td>6</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>$\theta_6$</td>
</tr>
</tbody>
</table>

Table (2.1) Link Parameters of the PUMA 560

Using the above values of the link parameters the individual link transformation matrices $T_{i}^{i-1}$ are computed. Then, the link transformations can be concatenated (multiplied together) to find the single transformation that relates frame {6} to frame {0}:

\[
T_6^0 = T_1^0 T_2^1 T_3^2 ... T_6^5
\]  

(2.11)

The transformation given by Eq. (2.11) is a function of all 6 joint variables. From the robot’s joint position sensors, the Cartesian position and orientation of the last link may be computed using Eq. (2.11). Appendix A lists the individual link transformations $T_{i}^{i-1}$ and the final link transformation matrix $T_6^0$ as given by Equations (2.10) and (2.11).
These transformations were derived using the above D-H table for PUMA 560. Appendix D shows the computer program to evaluate the forward kinematics i.e., $T_0^0$ of the PUMA 560.
Chapter 3
Inverse Kinematics

3.1 Introduction

The Inverse Kinematics problem is contrasted with the forward (direct) kinematics problem as follows: Given a desired position and orientation for the end-effector of the robot, determine a set of joint variables that achieves the desired position and orientation.

![Figure 3.1 The Standard Frames (Adapted from [18])]({})

The solution of the problem of finding the required joint angles is achieved by decoupling the inverse kinematics problem into two simpler problems. First, the position of the intersection of the wrist axes \( \{W\} \) is found, and then we find the orientation of the wrist (See Figure 3.1 and Figure 3.2).
3.2 Solvability

The equations for solving the inverse kinematics of a manipulator are nonlinear. Given the numerical value of the transformation matrix, we find the values $\theta_1, \theta_2, \ldots, \theta_n$. Considering the equations for $T^0_6$ given in Appendix A for the PUMA 560 manipulator, the statement of our problem is: Given the transformation matrix as $T^0_6$, solve for the six joint angles $\theta_1$ through $\theta_6$. For PUMA 560 with 6 dof, corresponding to equations in Appendix A, we have twelve equations and six unknowns. However, among the nine equations, arising from the rotation part of the transformation matrix $T^0_6$, only three equations are independent. These added with the three equations from the position part of $T^0_6$ gives six equations with six unknowns. These equations are nonlinear transcendental equations.

A manipulator is considered solvable if all the sets of joint variables associated with a given position and orientation can be determined [59].

We restrict our attention to closed form solution methods. In this context "closed
form" means a solution based on analytic expressions or on the solution of a polynomial of degree 4 or less, such that noniterative calculations suffice to arrive at a solution.

### 3.3 Pieper’s Solution

Pieper studied manipulators with six degrees of freedom in which three consecutive axes intersect at a point [54,55]. A sufficient condition that a manipulator will have a closed form solution is that three consecutive joint axes intersect at a point. The last three axes i.e., 4, 5, and 6 of PUMA 560 intersect.

When the last three axes intersect, the origins of link frames \{4\}, \{5\}, and \{6\} are located at this point of intersection. This point is given in base coordinates as

\[
P_{4ORG}^0 = T_1^0 T_2^1 T_3^2 P_{4ORG}^3 \quad (3.1)
\]

Using the fourth column of Eq. (2.10) for \(i = 4\),

\[
P_{4ORG}^0 = T_1^0 T_2^1 T_3^2 \begin{bmatrix}
a_3 \\
-d_4 s_3 a_3 \\
d_4 c_3 a_3 \\
1
\end{bmatrix} \quad (3.2)
\]

or,

\[
P_{4ORG}^0 = T_1^0 T_2^1 \begin{bmatrix}
f_1(\theta_3) \\
f_2(\theta_3) \\
f_3(\theta_3) \\
1
\end{bmatrix} \quad (3.3)
\]

where
Using Eq. (2.10) for \( T^2_3 \) in Eq. (3.4) yields the following expressions for \( f_i \):

\[
\begin{align*}
f_1 &= a_2 c_3 + d_4 s_3 s_3 + a_2 \\
f_2 &= a_2 c_2 s_3 - d_4 s_3 c_3 - d_4 s_2 c_3 - d_3 s_3 \\
f_3 &= a_2 s_2 s_3 - d_4 s_3 s_2 c_3 + d_4 c_2 c_3 + d_3 c_2.
\end{align*}
\]

(3.5)

Using Eq. (2.10) for \( T^0_1 \) and \( T^2_2 \) in Eq. (3.3) we obtain

\[
P_{^0_{40RG}} = \begin{bmatrix}
  c_1 g_1 - s_1 g_2 \\
  s_1 g_1 + c_1 g_2 \\
  g_3 \\
  1
\end{bmatrix}
\]

(3.6)

where

\[
\begin{align*}
g_1 &= c_2 f_1 - s_2 f_2 + a_1, \\
g_2 &= s_2 c_1 f_1 + c_2 c_1 f_2 - s_1 f_3 - d_2 s_1, \\
g_3 &= s_2 s_1 f_1 + c_2 s_1 f_2 + c_1 f_3 + d_2 c_1.
\end{align*}
\]

(3.7)

We now write an expression for the magnitude squared of \(^0 p_{40RG}\), which is seen from Eq. (3.6) to be

\[
r = g_1^2 + g_2^2 + g_3^2
\]

(3.8)

or, using Eq. (3.7) for the \( g_i \), we have
\[ r = f_1^2 + f_2^2 + f_3^2 + a_1^2 + d_2^2 + 2d_3f_3 + 2a_1(c_2f_1 - s_2f_2). \]  

(3.9)

We now write this equation, along with the \( z \) component equation from Eq. (3.6), as a system of two equations in the form

\[
\begin{align*}
r &= (k_1c_2 + k_2s_2)2a_1 + k_3, \\
z &= (k_1s_2 - k_2c_2)s\alpha_1 + k_4,
\end{align*}
\]

(3.10)

where

\[
\begin{align*}
k_1 &= f_1, \\
k_2 &= -f_2, \\
k_3 &= f_1^2 + f_2^2 + f_3^2 + a_1^2 + d_2^2 + 2d_3f_3, \\
k_4 &= f_3c\alpha_1 + d_2c\alpha_1.
\end{align*}
\]

(3.11)

Eq. (3.10) is useful because dependence on \( \theta_1 \) has been eliminated, and dependence on \( \theta_2 \) takes a simple form.

For our case of PUMA 560, since we have chosen the D-H parameters in such a way that \( a_i \) is zero (see Table (2.1)), therefore after making the substitution

\[
\begin{align*}
u &= \tan\frac{\theta}{2}, \\
\cos\theta &= \frac{1 - u^2}{1 + u^2}, \\
\sin\theta &= \frac{2u}{1 + u^2}.
\end{align*}
\]

(3.12)

we have \( r = k_3 \) where \( r \) is known. The right-hand side \( (k_3) \) is a function of \( \theta_3 \) only. This is an equation of degree 2 in \( u \) which may be solved for \( \theta_3 \).
Having solved for $\theta_3$, we may solve Eq. (3.10) for $\theta_2$, and Eq. (3.6) for $\theta_1$.

To complete our solution, we need to solve for $\theta_4$, $\theta_5$, and $\theta_6$. Since these axes intersect, these joint angles affect the orientation of only the last link. We can compute them based only upon the rotation portion of the specified goal, $R_6^g$. Having obtained $\theta_1$, $\theta_2$, and $\theta_3$, we can compute $R_3^g$, the orientation of link frame {3} relative to the base frame. The desired orientation of {6} differs from this orientation by only the action of the last three joints. Since the problem was specified given $R_6^g$, we can compute

$$R_6^g = (R_3^g)^{-1} R_6^o.$$  \hspace{1cm} (3.13)

For many manipulators, these last three angles can be solved by using the Z-Y-Z Euler angle solution given in Appendix B applied to $R_6^g$. For any manipulator (with intersecting axes 4, 5, and 6), the last three joint angles can be solved as a set of appropriately defined Euler angles. Since there are always two solutions for these last three joints, the total number of solutions for the manipulator will be twice the number found for the first three joints. As shown in Table (3.1) below, corresponding to two values for $\theta_3$, (obtained from quadratic equation (3.10) by using substitution of Eq. (3.12)), we have two values of $\theta_2$, the equation for later being quadratic again. Using this chain we have corresponding values for $\theta_1$, $\theta_4$, $\theta_5$, and $\theta_6$. Thus, we will have four sets of joint solutions for $\theta_1$, $\theta_2$, ..., $\theta_6$.

<table>
<thead>
<tr>
<th>$\theta_3(\text{+})$</th>
<th>$\theta_3(\text{+})$</th>
<th>$\theta_3(-)$</th>
<th>$\theta_3(-)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_4$</td>
<td>$\theta_2(-)$</td>
<td>$\theta_2(+)$</td>
<td>$\theta_2(-)$</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>$\theta_1$</td>
<td>$\theta_1$</td>
<td>$\theta_1$</td>
</tr>
<tr>
<td>$\theta_5$</td>
<td>$\theta_5$</td>
<td>$\theta_5$</td>
<td>$\theta_5$</td>
</tr>
<tr>
<td>$\theta_4$</td>
<td>$\theta_4$</td>
<td>$\theta_4$</td>
<td>$\theta_4$</td>
</tr>
<tr>
<td>$\theta_6$</td>
<td>$\theta_6$</td>
<td>$\theta_6$</td>
<td>$\theta_6$</td>
</tr>
</tbody>
</table>

Table (3.1) Set of Joint Solutions for PUMA 560
Using the Pieper's solution technique the expressions for respective joint angles for PUMA 560 are as follows:

$$\theta_1 = \text{Atan2}(Y_e, X_e) - \text{Atan2}(g_2, g_1)$$

$$\theta_2 = \text{Atan2}(-k_1, k_2) \pm \text{Atan2}(\sqrt{k_1^2 + k_2^2 - Z_e^2}, Z_e)$$

$$\theta_3 = \text{Atan2}(d_4, a_3) \pm \text{Atan2}(\sqrt{a_3^2 + d_4^2 - KC^2}, KC)$$

$$\theta_4 = \text{Atan2}[r_{3i}/\sin\theta_5, r_{13}/\sin\theta_5]$$

$$\theta_5 = \text{Atan2}\left[\sqrt{r_{21}^2 + r_{22}^2}, -r_{23}\right]$$

$$\theta_6 = \text{Atan2}\left[-r_{22}/\sin\theta_5, r_{23}/\sin\theta_5\right]$$

where $X_e, Y_e$ and $Z_e$ are the end effector position coordinates corresponding to the 4th column of the matrix $T_{\text{desired}}$, and "KC" is given by

$$KC = (r - 2d_2d_3 - d_3^2 - a_2^2 - d_4^2 - d_5^2 - a_5^2)/2a_2$$

The elements $r_i$ correspond to the R.H.S of Eq. (3.13)

If $\theta_5 = 0.0$ then,

$$\theta_5 = 0.0,$$

$$\theta_3 = 0.0,$$

and

$$\theta_6 = \text{Atan2}[-r_{12}, r_{11}]$$

If $\theta_5 = \pi$ then,

$$\theta_5 = \pi,$$

$$\theta_3 = 0.0,$$

and

$$\theta_6 = \text{Atan2}[r_{12}, -r_{11}]$$

The computer program in Appendix D uses the Pieper's solution approach to evaluate the above expressions for the inverse kinematics solution of PUMA 560. This program is easily verified by checking the match of joint angles ($\theta_1, \theta_2, \ldots, \theta_6$) input by forward kinematics and the output obtained through inverse kinematics solution. Using
the set of solutions obtained above a graphic simulation computer program is developed to show the set of 4 possible configurations of the PUMA 560 (Appendix D). Figure 3.3 shows the 4 solutions i.e., arm up, arm down, elbow left, and elbow right configurations of the PUMA 560 using the graphic simulation.
Figure 3.3 Four Solutions of the PUMA 560.
Chapter 4
Equations of Motion and Dynamics

4.1 Introduction

In this chapter we consider the equations of motion for the PUMA 560 i.e., the way in which the manipulator moves due to torques applied by the actuators, or from external forces applied to the manipulator.

We begin by deriving the so-called Euler-Lagrange equations. In order to determine these equations in a specific situation, one has to form the Lagrangian of the system. The Lagrangian dynamic formulation is an "energy based" approach to dynamics, using the kinetic and potential energy of the system [65].

There are two problems related to the dynamics of the manipulator that we wish to solve. The first problem requires to find the joint torques, \( \tau \) given a trajectory \( \theta, \dot{\theta} \) and \( \ddot{\theta} \). This formulation of dynamics is useful for controlling the manipulator [4]. The second problem is to calculate how the mechanism will move under application of joint torques. That is, given a torque vector, \( \tau \), calculate the resulting motion of the manipulator, \( \theta, \dot{\theta} \) and \( \ddot{\theta} \). This is useful for simulating the manipulator.

4.2 Lagrangian Formulation of Manipulator Dynamics

We start by developing an expression for the kinetic energy of a manipulator. The kinetic energy of the \( i \)th link, \( K_i \), can be expressed as

\[
K_i = \frac{1}{2} m_i V_{ci}^T V_{ci} + \frac{1}{2} \omega_i^T (I_i) \omega_i
\]

(4.1)

where the first term is the translational kinetic energy, and the second term is the rotational kinetic energy. \( V_{ci} \) is the linear velocity of the link's center of mass, \( \omega_i \) is the
angular velocity of the link, and $I_i$ is the inertia dyadic. The total kinetic energy of the manipulator is the sum of kinetic energy of the individual links i.e.,

$$K = \sum_{i=1}^{n} K_i$$

Equation (4.2)

Since the $V_{Ci}$ and $\dot{\theta}_i$ in Eq. (4.1) are the functions of $\theta$ and $\dot{\theta}$, we see that the kinetic energy of a manipulator can be described by a scalar formula as a function of joint position and velocity, $K(\theta, \dot{\theta})$. In fact, the kinetic energy of a manipulator is given by

$$K(\theta, \dot{\theta}) = \frac{1}{2} \dot{\theta}^T M(\theta) \dot{\theta},$$

Equation (4.3)

where $M(\theta)$ is the nxn symmetric and positive definite manipulator mass matrix. An expression of the form of Eq. (4.3) is known as a quadratic form, since when expanded out, the resulting scalar equation is composed solely of terms whose dependence on the $\theta_i$ is quadratic [50]. Eq. (4.3) can be seen to be analogous to the familiar expression for the kinetic energy of a point mass,

$$K = \frac{1}{2} m \dot{V}^2.$$

Equation (4.4)

The potential energy of the $i$th link, $U_i$, can be expressed as

$$U_i = -m_i \dot{\theta}_i^T P_{Ci},$$

Equation (4.5)

where $\dot{\gamma}$ is the 3x1 gravity vector, $P_{Ci}$ is the vector locating the center of mass of the $i$th link. The total potential energy stored in the manipulator is the sum of the potential energy in the individual links i.e.,

$$U = \sum_{i=1}^{n} U_i$$

Equation (4.6)

Since, the $P_{Ci}$ in (4.5) are functions of $\theta$, we see that the potential energy of a manipulator can be described by a scalar formula as a function of joint position, $U(\theta)$.

The Lagrangian dynamic formulation provides a means for deriving the equations
of motion from a scalar function called the Lagrangian, which is defined as the difference between the kinetic and potential energy of a mechanical system. In our notation, the Lagrangian of a manipulator is

\[ L(\theta, \dot{\theta}) = K(\theta, \dot{\theta}) - U(\theta). \] (4.7)

### 4.3 Equations of Motion

The equations of motion for the manipulator are then given by

\[ \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = \tau \] (4.8)

where \( \tau \) is the \( n \times 1 \) vector of actuator torques.

In the case of a manipulator, this equation becomes

\[ \frac{d}{dt} \frac{\partial K}{\partial \dot{\theta}} - \frac{\partial K}{\partial \theta} + \frac{\partial U}{\partial \theta} = \tau, \] (4.9)

The expanded structure of the terms corresponding to Eq. (4.9) is as follows:

Corresponding to first term we have,

\[ \frac{\partial L}{\partial \theta_k} = \sum_j M_{ki}(\theta) \dot{\theta}_j \] (4.10)

and

\[ \frac{d}{dt} \frac{\partial L}{\partial \theta_k} = \sum_j M_{ki}(\theta) \ddot{\theta}_j + \sum_j \frac{d}{dt} M_{ki}(\theta) \dot{\theta}_j \]

\[ = \sum_j M_{ki}(\theta) \ddot{\theta}_j + \sum_{i,j} \frac{\partial M_{ki}}{\partial \theta_i} \dot{\theta}_i \dot{\theta}_j \] (4.11)
Also

\[ \frac{\partial L}{\partial \dot{\theta}_k} = \frac{1}{2} \sum_{i,j} \frac{\partial M_{ij}}{\partial \theta_k} \dot{\theta}_i \dot{\theta}_j - \frac{\partial U}{\partial \theta_k} \]  

(4.12)

Thus the Euler-Lagrange equations can be written

\[ \sum_j M_{kj}(\theta) \ddot{\theta}_j + \sum_i \left( \frac{\partial M_{ij}}{\partial \theta_i} - \frac{1}{2} \frac{\partial M_{ij}}{\partial \theta_k} \right) \dot{\theta}_i \dot{\theta}_j + \frac{\partial U}{\partial \theta_k} = \tau_k \]  

(4.13)

\[ k = 1, \ldots, n \]

In the above Eq. (4.13), there are three types of terms. The first involve the second derivative of the generalized coordinates. The second are quadratic terms in the first derivatives of \( \theta \), where the coefficients may depend on \( \theta \). These are further classified into two types. Terms involving a product of the type \( \dot{\theta}_i^2 \) are called centrifugal, while those involving a product of the type \( \dot{\theta}_i \dot{\theta}_j \) where \( i \neq j \) are called Coriolis terms. The third type of terms are those involving only \( \theta \) but not its derivatives. Clearly the latter arise from differentiating the potential energy. It is common to write Eq. (4.13) in joint space as [57]

\[ M(\theta) \ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta) = \tau \]  

(4.14)

The fictitious forces acting on the end-effector, \( F \), could in fact be applied by the actuators at the joints using the relationship

\[ \tau = J^T(\theta)F, \]  

(4.15)

where the Jacobian, \( J(\theta) \), is written in the same frame as \( F \), usually the tool frame, \( \{T\} \).

The Jacobian is a multidimensional form of the derivative. In the field of robotics the Jacobians are time-varying linear transformations which relate joint velocities to Cartesian velocities of the tip of the arm [18,64]. Mathematically, the forward kinematic equations define a function between the space of Cartesian positions and orientations and the space of joint positions. The velocity relationships are then determined by the Jacobian of this function. The Jacobian is a matrix valued function and can be thought
of as the vector version of the ordinary derivative of a scalar function. For example
\[ \ddot{X} = J(\theta) \dot{\theta} \]  

(4.16)
where \( \theta \) is the vector of joint angles of the manipulator, and \( \dot{X} \) is a vector of linear and angular velocities of the end-effector. For any given configuration of the manipulator, joint rates are related to velocity of the tip in a linear fashion. This is only an instantaneous relationship.

For the PUMA 560, the Jacobian is 6x6, \( \dot{\theta} \) is 6x1, and \( \dot{X} \) is 6x1. This 6x1 Cartesian velocity vector is the 3x1 linear velocity vector and the 3x1 rotational velocity vector stacked together:

\[ \dot{X} = \begin{bmatrix} \dot{V} \\ \dot{\omega} \end{bmatrix} \]  

(4.17)
The dimension of the Jacobian is defined such that the number of rows equals the number of degrees of freedom in the Cartesian space being considered. The number of columns in a Jacobian is equal to the number of joints of the manipulator.

It may be desirable to express the dynamics of a manipulator with respect to Cartesian variables [35]. This entails the formulation of the dynamic equations which relate the acceleration of the end-effector expressed in Cartesian space to Cartesian forces and moments acting at the end-effector. The general form of the Cartesian state space equation is given as

\[ F = M_x(\theta) \ddot{X} + V_x(\theta, \dot{\theta}) + G_x(\theta), \]  

(4.18)
where \( F \) is a force-torque vector acting on the end-effector of the robot, and \( X \) is an appropriate Cartesian vector representing position and orientation of the end-effector [36]. Analogous to the joint space quantities, \( M_x(\theta) \) is the Cartesian mass matrix, \( V_x(\theta, \dot{\theta}) \) is a vector of velocity terms in Cartesian space, and \( G_x(\theta) \) is a vector of gravity terms in Cartesian space.

We can derive the relationship between the term of Eq. (4.14) and those of Eq. (4.18) in the following way. First, we premultiply Eq. (4.14) by the inverse of the
Jacobian transpose to obtain

$$J^T \tau = J^T M(\theta) \ddot{\theta} + J^T V(\theta, \dot{\theta}) + J^T G(\theta), \quad (4.19)$$

or,

$$F = J^T M(\theta) \ddot{\theta} + J^T V(\theta, \dot{\theta}) + J^T G(\theta). \quad (4.20)$$

We next develop a relationship between the joint space and Cartesian acceleration, starting with the definition of the Jacobian,

$$\dot{X} = J \dot{\theta} \quad (4.21)$$

and differentiating to obtain

$$\ddot{X} = J \ddot{\theta} + J \dot{\theta}. \quad (4.22)$$

Solving Eq. (4.22) for joint space acceleration leads to

$$\ddot{\theta} = J^{-1} \ddot{X} - J^{-1} J \ddot{\theta}. \quad (4.23)$$

Substituting Eq. (4.23) into Eq. (4.20) we have

$$F = J^T M(\theta) J^{-1} \ddot{X} - J^T M(\theta) J^{-1} J \ddot{\theta} + J^T V(\theta, \dot{\theta}) + J^T G(\theta), \quad (4.24)$$

from which we derive the expressions for the terms in the Cartesian dynamics as

$$M_x(\theta) = J^{-T}(\theta) M(\theta) J^{-1}(\theta),$$

$$V_x(\theta, \dot{\theta}) = J^{-T}(\theta) (V(\theta, \dot{\theta}) - M(\theta) J^{-1}(\theta) \dot{J}(\theta) \dot{\theta}), \quad (4.25)$$

$$G_x(\theta) = J^{-T}(\theta) G(\theta).$$

The Jacobian appearing in equations (4.25) is written in the same frame as $F$ and $X$ in Eq. (4.18) (certain choices facilitate computation) though, the choice of this frame
is arbitrary. Appendix C describes the application of jacobians in the force domain. Appendices C and D respectively, describe the analytic expressions and algorithms used to obtain the $J$ and $\dot{J}$ matrix for PUMA 560.

4.4 The Real and Approximate Dynamic Models

The algorithms in Appendix D were based on the "Real" model i.e., the model based by actually deriving the equations of motion analytically using the Lagrangian formulation.

The "Approximate" model was a simplified model, abbreviated from the full explicit model [3]. The approximate model was an improved dynamic model, in which a PUMA 560 arm was disassembled; the inertial properties of the individual links were measured; and an explicit model incorporating all of the non-zero measured parameters was derived.

The procedure used to derive the approximate model entailed simplification of the Kinetic energy, Centrifugal and the Coriolis matrix elements by combining inertia constants that multiply common variable expressions. Also, three simplification assumptions were made for the analysis: The rigid body assumption; link 6 has been assumed to be symmetric, i.e., $I_{xx} = I_{yy}$; and only the mass moments of inertia are considered, i.e., $I_{xx}, I_{yy}$ and $I_{zz}$. All the terms of the Kinetic energy, Centrifugal and the Coriolis matrices were retained, but the terms which were less than 1% of the greatest term within the same equation, or less than 0.1% as great as the largest constant term applicable to the same joint were eliminated.

The link parameters required to calculate the elements of the Kinetic energy, Centrifugal and Coriolis matrices and the gravity vector comprise of: the mass, location of the center of gravity and the terms of the inertia dyadic. The wrist, upper arm, and the shoulder of a PUMA 560 were detached in order to measure these parameters (See Figure 1.1). The mass of each component was determined with a beam balance. The center of gravity was located by balancing each link on a knife edge, once orthogonal to each axis. The diagonal terms of the inertia dyadic were measured with a two wire
The key features of the approximate model were:

(i) Its compactness characterized by fewer calculations.
(ii) Requirement of less memory storage.
(iii) The approximate model was computationally efficient. When the computer program(s) for the verification of both the models were run, it was found that the approximate model took 1.7030297 seconds of the processor time and, the real model took 1.813187 seconds of the processor time (Appendix D). Thus, the results generated by the approximate model were about 6.08% faster than the real model.
(iv) Efficient formulation since the results were significantly close to the real model.

The approximate model would save run time when the robot is being physically moved. This would be possible since the data from the computer could be produced as fast as possible in order to control the actual movement of the robot.

### 4.5 Dynamic Simulation

As mentioned above, a very important use of dynamic equations of motion is in simulation. By reformulating the dynamic equations so that acceleration is computed as a function of actuator torque, it is possible to simulate how a manipulator would move under application of a set of actuator torques.

To simulate the motion of PUMA 560, we make use of the model of dynamics developed above. Given the dynamics written in closed form as in Eq. (4.14), simulation requires solving the dynamic equation for acceleration:

\[
\ddot{\theta} = M^{-1}(\theta)[\tau - V(\theta, \dot{\theta}) - G(\theta)]
\]  

(4.26)

We may then apply any of the several known numerical integration techniques to
integrate the acceleration to compute future positions and velocities [16].

Given initial conditions on the motion of the manipulator, usually in the form:

\[
\theta(0) = \theta_0, \\
\dot{\theta}(0) = 0,
\]

we numerically integrate Eq. (4.26) forward in time steps of size \(\Delta t\). For the purpose of performing numerical integration, in our case the fourth order Runge-Kutta method was used [56,12].

For the purpose of illustration, we introduce a simple integration scheme, called Euler integration, which is accomplished as follows: Starting with \(t = 0\), iteratively compute

\[
\dot{\theta}(t + \Delta t) = \dot{\theta}(t) + \ddot{\theta}(t)\Delta t, \\
\theta(t + \Delta t) = \theta(t) + \dot{\theta}(t)\Delta t + \frac{1}{2}\ddot{\theta}(t)\Delta t^2,
\]

where for each iteration, Eq. (4.26) is computed to calculate \(\dot{\theta}\). In this way, the position, velocity, and acceleration of the manipulator caused by a certain input torque function can be computed numerically. The time step \(\Delta t\) is selected such that it is sufficiently small so that breaking continuous time into small increments is a reasonable approximation. Also, \(\Delta t\) should be sufficiently large so that excessive computer time is not required to compute a simulation.

4.6 Animation

A central element in OLP systems is the use of computer graphics to simulate the robot [58]. This requires that the robot be modelled as a three-dimensional object. Intergraph contains a facility to build 3-D CAD models. These models can be stored as wireframe models or solid models. The 3-D solid model for PUMA 560 was created and animated using Intergraph’s Engineering Modeling System (EMS) [32,31]. Each simulated entity such as the robot links are represented by an object. The object data
structure contains the model of the entity, and several other attributes (from Kinematics point of view). Each object can be referenced by a label because the object identification numbers are stored as graphic variables.

The individual movement of the robot links is achieved using Intergraph's Parametric Programming Language (PPL) feature [33]. PPL acts as the interface between the results of dynamic simulation and its animation. The program (Appendix D) reads the data file containing the time history of joint angles, $\theta_1, \theta_2, \ldots \theta_6$, (obtained from integration of Eq. (4.26)) and uses this as input to move the joint arms accordingly.

### 4.7 Verification

The computer programs based on the real and approximate model (Appendix D) were tested. One of the criterion used to test the accuracy of the programs was to assume that the PUMA 560 arm was in the "zero" configuration, as shown in Figure (2.1). The arm is then released and falls freely under gravity without the application of any external motor torques. At each successive time step, the value of kinetic energy, potential energy, total energy and the variation in individual joint angles and joint velocities were found. The analysis of output revealed the following results:

Starting from zero position the kinetic energy increased gradually and to compensate for the increase in kinetic energy the potential energy decreased, in accordance with the law of conservation of energy. Thus, the total energy of the system remained constant (Figure 4.5). As expected, maximum variation of joint angle(s) was seen in $\theta_2$, and minimum in $\theta_6$ (Figure 4.1 to Figure 4.4).

### 4.8 Results

The objective of the set of computer programs mentioned above is to determine how the mechanism will move under application of a set of joint torques. That is, given a torque vector, $\tau$, calculate the resulting motion of the manipulator, $\theta$, $\dot{\theta}$ and $\ddot{\theta}$. This is useful for simulating the manipulator. The output of joint angles from both the models was input to graphical simulation algorithm (Appendix D) and the behavior of the
manipulator was analyzed. Figure 4.6 and Figure 4.7 show plots obtained for some intermediate time steps for the real and approximate models respectively, when the algorithms were tested using the "conservation of energy" approach.

The above computer programs are modified to find the required vector of joint torques, $\tau$ given a trajectory, $\theta$, $\dot{\theta}$ and $\ddot{\theta}$. This formulation of dynamics is useful for the problem of controlling the manipulator. Various joint space schemes are employed to get the desired output. Chapter 5 gives the details on this aspect of trajectory generation.
Real vs. Approximate Models

- **Real Model**
- **Approx. Model**

Figure 4.1 Variation of Joint angles (degrees) for θ, w.r.t. Time
Figure 4.2 Variation of Joint angles (degrees) for $\theta_1$, $\theta_2$, w.r.t Time
Figure 4.3 Variation of Joint velocities ($\theta_1, -\theta_2$) w.r.t Time
Figure 4.4 Variation of Joint Velocities ($\theta - \dot{\theta}$) w.r.t. Time
Figure 4.5 Variation of Potential, Kinetic and Total Energies w.r.t Time
Figure 4.6 Plot showing Intermediate steps when the PUMA 560 falls under gravity (Real Model).
Figure 4.7 Plot showing intermediate steps when PUMA 560 falls under gravity (Approx. Model).
Chapter 5
Path Planning Schemes

5.1 Introduction

This chapter concerns with the methods of computing a trajectory in joint space which causes a desired motion of a manipulator. Here, trajectory refers to a time history of desired joint position, velocity, and acceleration for each degree of freedom.

This problem includes the human interface problem of how we wish to specify a trajectory or path through space. In order to make the description of manipulator motion easy for a user of a robot system, the user should not be required to write down complicated functions of space and time to specify the task. Rather, the specification of the trajectories must be done with simple descriptions of the desired motion. For example, the user may just specify the desired goal position and orientation of the end-effector, and let the system decide on the exact path, its duration, the velocity profile, etc..

5.2 Trajectory Interpolation

The simplest type of robot motion is the point to point motion. In this motion the robot is commanded to go from an initial configuration to a final configuration without specifying the intermediate path followed by the end-effector.

For robots that are commanded to perform a motion using teach pendants, there is no need for calculating the forward or the inverse kinematics. The desired motion is recorded as a set of joint angles (actually as a set of encoder values) and then the robot can be controlled entirely in joint space. For off-line programming, given the desired initial and final positions and orientations of the end-effector, the inverse kinematic solution must be evaluated to find the required initial and final joint variables.
5.3 Joint Space Schemes

We discuss here the problem of generating smooth trajectories in joint space i.e., we consider methods of path generation in which the path shapes (in space and time) are described in terms of functions of joint angles.

Each path point is usually specified in terms of a desired position and orientation of the tool frame, \( \{T\} \), relative to the station frame, \( \{S\} \) (See Figure's 3.1 and 3.2). Each of these via points is "converted" into a set of desired joint angles by application of inverse kinematics. Then a smooth function is found for each of the \( n \) joints which pass through the via points and end at the goal point. The time required for each segment is the same for each joint so that all joints reach the via point at the same time, thus resulting in the desired Cartesian position of \( \{T\} \) at each via point. Other than specifying the same duration for each joint, the determination of the desired joint angle function for a particular joint does not depend on the other joints.

What is required is a function for each joint whose value at \( t_0 \) is the initial position of the joint, and whose value at \( t_f \) is the desired goal position at that joint. There are many smooth functions for \( \theta(t) \), which might be used to interpolate the joint value.

For the joint space schemes to achieve the desired position and orientation at the via points, we then consider the problem of fitting cubic curves. These cubic curves connect the via points in a smooth way.

In making a single smooth motion, at least four constraints on \( \theta(t) \) are evident. Two constraints on the function's value come from the selection of initial and final values:

\[
\theta(0) = \theta_0, \\
\theta(t_f) = \theta_f
\]  

(5.1)

If desired velocities of the joints at the via points are known, then we can determine cubic polynomials with the velocity constraints as:
These four constraints can be satisfied by a polynomial of at least third degree. Since a cubic polynomial has four coefficients, it can be made to satisfy the four constraints given by Eq. (5.1) and Eq. (5.2). These constraints uniquely specify a particular cubic. A cubic has the form

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3,$$

and so the joint velocity and acceleration along this path are clearly

$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2,$$
$$\ddot{\theta}(t) = 2a_2 + 6a_3 t.$$

Combining Eq. (5.3) and Eq. (5.4) with the four desired constraints yields four equations in four unknowns:

$$\theta_0 = a_0,$$
$$\theta_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3,$$
$$\dot{\theta}_0 = a_1,$$
$$\dot{\theta}_f = a_1 + 2a_2 t_f + 3a_3 t_f^2.$$

Solving these equations for the $a_i$ we obtain
Using Eq. (5.6) we can calculate the cubic polynomial that connects the initial and final positions.

If we have the desired joint velocities at each via point, then we simply apply Eq. (5.6) to each segment to find the required cubics. One of the ways in which desired velocity at each of the via points might be specified is that the user inputs the desired velocity in terms of a Cartesian linear and angular velocity of the tool frame at that instant.

The Cartesian desired velocities at the via points are "mapped" to desired joint rates using the inverse Jacobian of the manipulator evaluated at the via point.

5.4 Results

We developed algorithm(s) corresponding to the real and approximate dynamic models for generating cubic trajectories given arbitrary initial and final conditions (Appendix D):

Given initial and final times, \( t_0 \) and \( t_f \), respectively, with the set of four initial conditions, (Eq. (5.1) and Eq. (5.2)) the required cubic polynomial \( \theta(t) \) can be computed from Eq. (5.3).
A sequence of moves can be planned using equations (5.3) and (5.6) by using the end conditions \( \theta_i \), \( \dot{\theta}_i \) of the \( i \)-th move as initial conditions for the \( i+1 \)-th move. Thus the input of \( \theta(t), \dot{\theta}(t) \) and \( \ddot{\theta}(t) \) (Eq. (5.3) and Eq. (5.4)) is applied to Eq. (4.14) to get the vector of joint torques \( \tau \) \([57] \). With a similar analogy, \( X(t), \dot{X}(t) \), and \( \ddot{X}(t) \) can be applied to the Cartesian state space equation (Eq. (4.18)) to get the force-torque vector \( F \) acting on the end-effector of the robot, and \( X \), is an appropriate Cartesian vector representing position and orientation of the end-effector \([36] \).

As an example, Figure 5.1 shows the joint torques (real and approximate model, respectively) for a specific instant at \( t=3 \) seconds, with:

- \( \theta_0 = 15^\circ \)
- \( \theta_f = 75^\circ \)
- \( \dot{\theta}_0 = 0.0 \text{ rad/sec} \)
- \( \dot{\theta}_f = 0.0 \text{ rad/sec} \)

and,

- \( t_f = 3 \) seconds.

Using the above analogy Figure 5.2 shows the forces for the real and approximate model, respectively.
### Figure 5.1
An example plot showing the joint torques

### Figure 5.2
An example plot showing the forces
Chapter 6
Conclusions and Future Work

Off-line programming systems are useful in present-day industrial applications and can serve as a basis for continuing robotics research and development. A large motivation in developing OLP systems is to fill the gap between the explicitly programmed systems available today and the task level systems of tomorrow.

In the last decade the growth of the industrial robot market has not been nearly as rapid as predicted. One primary reason for this is that robots are still too difficult to use. A great deal of time and expertise is required to install a robot in a particular application and bring the system to production readiness.

There are many factors that make robot programming a difficult task. Most of these special problems arise from the fact that a robot manipulator interacts with its physical environment [28]. Even simple programming systems maintain a "world model" of this physical environment in the form of various objects and have "knowledge" about presence and absence of various objects encoded in the program strategies. During development of a robot program it is necessary to keep the internal model maintained by the programming system in correspondence with the actual state of the robot's environment. Interactive debugging of programs with a manipulator requires frequent manual resetting of the state of the robot's environment i.e., parts, tools, etc., must be moved back to their initial locations. Such state resetting becomes especially difficult (and sometimes costly) when the robot performs a irreversible operation on one or more parts (eg., drilling, routing, etc.).

Although difficulties exist in maintaining an accurate internal model of the manipulator's environment, there seems no question that great benefits result from doing so. Whole ares of sensor research, perhaps most notably computer vision, focus on developing techniques by which world models may be verified, corrected, or discovered. Clearly, in order to apply any computational consideration algorithm to the robot
command-generation problem, the algorithm needs access to a model of the robot and its surroundings.

At the other end of the spectrum, are the so called task level programming (TLP) systems in which the programmer may state high-level goals such as "insert the bolt". Such systems use techniques from artificial intelligence research to automatically generate motion and strategy plans. However, these task level languages do not exist yet, although various pieces of such systems are under development by researchers [44]. Off-line programming systems should serve as the natural growth path from explicit programming systems to task level programming systems. The simplest OLP system is merely a graphical extension to a robot programming language, but from there it can be extended toward a task level programming system. This gradual extension is accomplished by providing automated solutions to various subtasks so as to explore options in the simulated environment. If we take this view, an OLP system serves as an important basis for research and development of task level planning systems. Hence, OLP systems should be a useful tool in research as well as an aid in current industrial practice.

Two potential benefits of OLP systems relate directly to the language translation. Most proponents of OLP systems note that one universal interface which enables users to program a variety of robots solves the problems of learning and dealing with several automation languages. The second benefit stems from economic considerations in future scenarios in which hundreds of robots fill factories. The cost associated with a powerful programming environment (such as a language and graphical interface) may prohibit placing this at the site of each robot installation. Rather, it seems to make economic sense to place very simple, but cheap controller with each robot, and have it downloaded from a powerful, "intelligent" OLP system which is located in an office environment.

Some of the advanced features which could be integrated into "baseline" OLP systems could be used to accomplish automated planning of an industrial application. Research on the planning of collision-free paths [43, 11, 48] and the planning of time-optimal paths [8, 61, 34] are natural candidates for inclusion in an OLP system. For example, consider the problem of using a six degree of freedom robot for an arc welding
task whose geometry specifies only five degrees of freedom. Automatic planning of the redundant degree of freedom can be used to avoid collisions and singularities of the robot [17].

In many arc welding situations, details of the process require a certain relationship between the workpiece and the gravity vector to be maintained during the weld. This results in a two or three degree of freedom orienting system on which the part is mounted operating simultaneously with the robot in a coordinated fashion. In such a system there may be nine or more degrees of freedom to coordinate. Such systems are generally programmed today using teach pendant techniques. A planning system that could automatically synthesize the coordinated motions for such a system might be quite valuable [17, 1].

In a simulated world in which objects are represented by their surfaces, it is possible to investigate the simulation of manipulator force-control strategies. This task involves the difficult problem of modeling some surface properties and expanding the dynamic simulator to deal with the constraints imposed by various contacting situations. In such an environment it might be possible to assess various force-controlled assembly operations for feasibility [53].

Planning automatic schedules for interacting processes is a difficult problem and an area of research [37, 2]. An OLP system would serve as an ideal test bed for such research, and would be immediately enhanced by any useful algorithms in this area.

An OLP system might be given some of the capabilities in modelling positioning errors sources and the effect of data from imperfect sensors [63, 22]. The world model could be made to include various error bounds and tolerancing information, and the system could assess the likelihood of success of various positioning or assembly tasks.

One of the basic tasks which could be extended to the OLP systems is the quick determination of the workcell layout in a simulated world as compared to the actual physical cell. Automatic placement (manipulator(s) collision free i.e., feasible reach to all of the required workpoints) can be computed by direct search, or by heuristic guided search techniques. Other criterion such as measure of manipulability in which the robot could reach in well-conditioned configurations could be applied [17, 49].
Currently, we have TRC004, a PUMA Interface Card, which is a general purpose interface board for servo applications [67]. The TRC004 is ideally suited for high-performance robotic applications including force-controlled manipulators and dextrous hands. It was specifically designed to replace the LSI/11 VAL computer and servo cards in the Unimate PUMA 560 manipulator to allow high speed, direct access to joint motor torques and positions [69]. When used as an interface to a Unimate PUMA 560 robotic manipulator, the TRC004, in conjunction with the user’s real-time computer, replaces the entire LSI/11 VAL computer and the joint servo cards (Figure 6.1).

Physically, the CPU, RAM, EPROM, serial controller, interface cards, and joint servos are all removed from the controller backplane. In their place, the TRC004, a single twelve-inch by twelve-inch (approximately) multi-purpose I/O card, is mounted and wired point-to-point to the backplane. The TRC004’s encoder inputs are connected to the joint encoders of the PUMA 560 for position acquisition. The analog outputs are connected to the power amplifiers for commanding motor torque. The analog inputs are used to measure the potentiometers during calibration. Finally, the discrete inputs and outputs are connected to various housekeeping functions including enabling arm power, controlling the pneumatic hand valves, and monitoring joint thermal sensors.

It depends on the user’s desire to replace as much of the functionality of the VAL operating system which includes, but is not limited to:

- power-on arm calibration
- occasional calibration of the joint potentiometers
- low-level joint position control
- arm kinematics
- trajectory generation
- joint limit monitoring

The teach-pendant, which is a serial peripheral of the LSI/11, becomes non-functional.
CONCLUSIONS AND FUTURE WORK

Figure 6.1 A block Diagram of the PUMA 560 with New Controller and Sensors.

Appendix D shows a few routines available for running and controlling the PUMA 560 with the TRC004 interface board [70].
References


REFERENCES


REFERENCES

Conference on Geometric Aspects of Industrial Design, Wright-Patterson AFB, OH, April 1990.


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


[60] Roth, R., "Principles of Automation", Future Directions in Manufacturing Technology, Based on the Unilever Research and Engineering Division Symposium held at Port Sunlight, April 1983, Published by Unilever Research, UK.


REFERENCES


Appendix A

A-1 Link Transformations for PUMA 560

\[ T_1^0 = \begin{bmatrix} c_{\theta_1} & -s_{\theta_1} & 0 & 0 \\ s_{\theta_1} & c_{\theta_1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]  \hspace{1cm} (A-1)

\[ T_2^1 = \begin{bmatrix} c_{\theta_2} & -s_{\theta_2} & 0 & 0 \\ s_{\theta_2} & c_{\theta_2} & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]  \hspace{1cm} (A-2)

\[ T_3^2 = \begin{bmatrix} c_{\theta_3} & -s_{\theta_3} & 0 & a_2 \\ s_{\theta_3} & c_{\theta_3} & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]  \hspace{1cm} (A-3)

\[ T_4^3 = \begin{bmatrix} c_{\theta_4} & -s_{\theta_4} & 0 & a_3 \\ 0 & 0 & -1 & -d_4 \\ s_{\theta_4} & c_{\theta_4} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]  \hspace{1cm} (A-4)
APPENDIX A

The final transformation matrix $T_5^6$ for PUMA 560 is:

$$
T_5^6 = \begin{bmatrix}
  c_{\theta_5} & -s_{\theta_5} & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  -s_{\theta_5} & c_{\theta_5} & 0 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
$$

(A-5)

The final transformation matrix $T_6^0$ for PUMA 560 is:

$$
T_6^0 = \begin{bmatrix}
  r_{11} & r_{12} & r_{13} & d_x \\
  r_{21} & r_{22} & r_{23} & d_y \\
  r_{31} & r_{32} & r_{33} & d_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
$$

(A-7)

where

$$
\begin{align*}
  r_{11} &= (c_1c_2c_4 + s_1s_4)c_5c_6 - c_1s_2s_5s_6 - c_1c_2s_4s_6 - s_1c_4s_6 \\
  r_{12} &= -(c_1c_2c_4 - s_1s_4)c_5s_6 + c_1s_2s_5s_6 - c_1c_2s_4s_6 - s_1c_4s_6 \\
  r_{13} &= (c_1c_2c_4 - s_1s_4)s_5 + c_1s_2c_5 \\
  r_{21} &= (s_1c_2c_4 + c_1s_4)c_5c_6 - s_1s_2s_5s_6 - s_1c_2s_4s_6 + c_1c_4s_6 \\
  r_{22} &= -(s_1c_2c_4 + c_1s_4)s_5s_6 + s_1s_2s_5s_6 + s_1c_2s_4s_6 + c_1c_4s_6 \\
  r_{23} &= (s_1c_2c_4 + c_1s_4)s_5 + s_1s_2c_5
\end{align*}
$$
APPENDIX A

\[ r_{31} = -s_2c_4c_5c_6 - c_2s_5c_6 + s_2s_5s_6 \]
\[ r_{32} = s_2c_4c_5s_6 + c_2s_5s_6 + s_2s_5c_6 \]
\[ r_{33} = -s_2c_5s_5 + c_2c_5 \]

\[ d_x = c_4(c_2a_3 + s_2d_4 + c_2a_2) - s_4d_{23} \]
\[ d_y = s_4(c_2a_4 + s_2d_4 + c_2a_2) + c_4d_{23} \]
\[ d_z = -s_2a_3 + c_2d_4 - s_2a_2 \]
Appendix B

B-1 Z-Y-Z Euler Angles

Starting with a frame coincident with a known frame \{A\}, if a frame \{B\} is first rotated about \( \hat{Z} \) by an angle \( \alpha \), then rotated about \( \hat{Y} \) by an angle \( \beta \), and then rotated about \( \hat{Z} \) by an angle \( \gamma \) then the equivalent rotation matrix is given as:

\[
R_{\hat{z}'/\hat{x}'(\alpha,\beta,\gamma)}^i = \begin{bmatrix}
    c_{\alpha}c_{\beta}c_{\gamma} - s_{\alpha}s_{\gamma} & -c_{\alpha}c_{\beta}s_{\gamma} - s_{\alpha}c_{\gamma} & c_{\alpha}s_{\beta} \\
    s_{\alpha}c_{\beta}c_{\gamma} + c_{\alpha}s_{\gamma} & -s_{\alpha}c_{\beta}s_{\gamma} + c_{\alpha}c_{\gamma} & s_{\alpha}s_{\beta} \\
    -s_{\beta}c_{\gamma} & s_{\beta}s_{\gamma} & c_{\beta}
\end{bmatrix}
\]  \hspace{1cm} (B-1)

The solution for extracting Z-Y-Z Euler angles from a rotation matrix is stated below:

Given

\[
R_{\hat{z}'/\hat{x}'(\alpha,\beta,\gamma)}^i = \begin{bmatrix}
    r_{11} & r_{12} & r_{13} \\
    r_{21} & r_{22} & r_{23} \\
    r_{31} & r_{32} & r_{33}
\end{bmatrix}
\]  \hspace{1cm} (B-2)

If \( \sin\beta \neq 0 \), then

\[
\beta = \text{Atan2}(\sqrt{r_{31}^2 + r_{32}^2}, r_{33}) ,
\]

\[
\alpha = \text{Atan2}(r_{23}/s_\beta, r_{13}/s_\beta) ,
\]

\[
\gamma = \text{Atan2}(r_{32}/s_\beta, -r_{31}/s_\beta) .
\]  \hspace{1cm} (B-3)
If $\beta = 0.0$, then a solution may be calculated as

$$\beta = 0.0,$$
$$\alpha = 0.0,$$  \hspace{1cm} (B-4)
$$\gamma = \text{Atan2}(-x_{12}, x_{11}).$$

If $\beta = 180.0^\circ$, then a solution may be calculated as

$$\beta = 180.0^\circ,$$
$$\alpha = 0.0,$$  \hspace{1cm} (B-5)
$$\gamma = \text{Atan2}(x_{12}, -x_{11}).$$
Appendix C

C-1 Jacobians in the Force Domain

There are joint torques that exactly balance forces at hand in the static situation. When forces act on a mechanism, work (in the technical sense) is done if the mechanism has a certain displacement. Work, a scalar quantity, is defined as a force acting through a distance. The principle of virtual work allows us to make certain statements about the static case by allowing the amount of this displacement to go to an infinitesimal. Specifically, we can equate the work done in Cartesian terms with the work done in joint space terms. In the multidimensional case, work is the dot product of a vector force or torque and a vector displacement. Thus we have,

$$ F \cdot \delta X = \tau \cdot \delta \theta, \quad (C-1) $$

where $F$ is a 6x1 Cartesian force-moment vector acting at the end-effector, $\delta X$ is a 6x1 infinitesimal Cartesian displacement of the end-effector, $\tau$ is a 6x1 vector of torques at the joints, $\delta \theta$ is a 6x1 vector of infinitesimal joint displacements. Expression (C-1) can also be written as

$$ F^T \delta X = \tau^T \delta \theta. \quad (C-2) $$

The definition of Jacobian is

$$ \delta X = J \delta \theta, \quad (C-3) $$

and so we may write

$$ F^T J \delta \theta = \tau^T \delta \theta, \quad (C-4) $$

which must hold for all $\delta \theta$, and so we have

$$ F^T J = \tau^T. \quad (C-5) $$
Transposing both sides yields the result

\[ \tau = J^T F. \]  

(C-6)

Thus, the Jacobian transpose maps Cartesian forces acting at the hand into equivalent joint torques. Eq. (C-6) is very interesting relationship in that it allows us to convert a Cartesian quantity into a joint space quantity without calculating any inverse kinematic functions. This is made use of in considering control problems.

C-2 The J and \( \dot{J} \) matrices for PUMA 560

The Jacobian matrix for PUMA 560 is:

\[
J = \begin{bmatrix} 
J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\
J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\
J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\
J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\
J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\
J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} 
\end{bmatrix}
\]  

(C-8)

where

\[
\begin{align*}
J_{11} &= -s_1(c_2d_3 + s_2d_4 + c_2a_2) - c_1d_23 \\
J_{12} &= c_1(-s_2d_3 + c_2d_4 - s_2a_2) \\
J_{13} &= c_1(-s_2d_3 + c_2d_4) \\
J_{14} &= 0.0 \\
J_{15} &= 0.0 \\
J_{16} &= 0.0
\end{align*}
\]
\text{APPENDIX C}

\[ J_{16} = 0.0 \]

\[ J_{21} = c_1(c_2s_3 + s_2d_4 + c_2a_3) - s_1d_{23} \]
\[ J_{22} = s_1(-s_2a_2 + c_2d_4 - s_2a_2) \]
\[ J_{23} = s_1(-s_2a_3 + c_2d_4) \]
\[ J_{24} = 0.0 \]
\[ J_{25} = 0.0 \]
\[ J_{26} = 0.0 \]

\[ J_{31} = 0.0 \]
\[ J_{32} = -(c_2a_2 + s_2d_4 + c_2a_2) \]
\[ J_{33} = -(c_2a_3 + s_2d_4) \]
\[ J_{34} = 0.0 \]
\[ J_{35} = 0.0 \]
\[ J_{36} = 0.0 \]

\[ J_{41} = 0.0 \]
\[ J_{42} = -s_1 \]
\[ J_{43} = -s_1 \]
\[ J_{44} = c_1s_{23} \]
\[ J_{45} = -c_1c_2s_4 - s_1c_4 \]
\[ J_{46} = (c_1c_2s_4 - s_1s_4)s_5 + c_1s_{23}c_5 \]

\[ J_{51} = 0.0 \]
\[ J_{52} = c_1 \]
\[ J_{53} = c_1 \]
\[ J_{54} = s_1s_{23} \]
\[ J_{55} = -s_1c_2s_4 + c_1c_4 \]
\[ J_{56} = (s_1c_2s_4 + c_1s_4)s_5 + s_1s_{23}c_5 \]
APPENDIX C

\[ J_{61} = 1.0 \]
\[ J_{62} = 0.0 \]
\[ J_{63} = 0.0 \]
\[ J_{64} = c_{23} \]
\[ J_{65} = s_{23}s_5 \]
\[ J_{66} = -s_{23}s_5 + c_{23}c_5 \]

The \( \mathbf{J} \) matrix for PUMA 560 is:

\[
\mathbf{J} = \begin{bmatrix}
J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\
J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\
J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\
J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\
J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\
J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66}
\end{bmatrix}
\]

where

\[
J_{11} = -c_1c_{23}a_3\dot{\theta}_1 + s_1s_{23}a_3\dot{\theta}_3 - c_1s_{23}d_3\dot{\theta}_2 + s_2a_2\dot{\theta}_2 + s_1d_2\dot{\theta}_1 \\
J_{12} = s_1s_{23}a_3\dot{\theta}_1 - c_1c_{23}a_3\dot{\theta}_3 - s_1s_{23}d_3\dot{\theta}_2 + s_2a_2\dot{\theta}_2 - c_1c_2\dot{\theta}_2 \\
J_{13} = s_1s_{23}a_3\dot{\theta}_1 - c_1c_{23}a_3\dot{\theta}_3 - s_1s_{23}d_3\dot{\theta}_2 - c_1s_2d_3\dot{\theta}_2 \\
J_{14} = 0.0 \\
J_{15} = 0.0 \\
J_{16} = 0.0 \\
J_{21} = -s_1c_{23}a_3\dot{\theta}_1 - c_1s_{23}a_3\dot{\theta}_3 - s_1s_{23}d_3\dot{\theta}_2 + c_1c_2d_3\dot{\theta}_3 - s_1s_2d_3\dot{\theta}_2 - c_1d_2\dot{\theta}_1 \\
J_{22} = -c_1s_{23}a_3\dot{\theta}_1 - s_1c_{23}a_3\dot{\theta}_3 + c_1c_2d_3\dot{\theta}_3 - s_1s_2d_3\dot{\theta}_2 - c_1s_2d_3\dot{\theta}_2 \\
J_{23} = -c_1s_{23}a_3\dot{\theta}_1 - s_1c_{23}a_3\dot{\theta}_3 + c_1c_2d_3\dot{\theta}_3 - s_1s_2d_3\dot{\theta}_2 \]
APPENDIX C

\[ J_{24} = 0.0 \]
\[ J_{25} = 0.0 \]
\[ J_{26} = 0.0 \]

\[ J_{31} = 0.0 \]
\[ J_{32} = s_2 a_2 \dot{\theta}_{23} - c_2 d_2 \dot{\theta}_{23} + s_2 a_2 \dot{\theta}_2 \]
\[ J_{33} = s_2 a_3 \dot{\theta}_{23} - c_2 d_2 \dot{\theta}_{23} \]
\[ J_{34} = 0.0 \]
\[ J_{35} = 0.0 \]
\[ J_{36} = 0.0 \]

\[ J_{41} = 0.0 \]
\[ J_{42} = -c_1 \dot{\theta}_1 \]
\[ J_{43} = -c_1 \dot{\theta}_1 \]
\[ J_{44} = -s_1 s_2 \dot{\theta}_1 + c_1 c_2 \dot{\theta}_{23} \]
\[ J_{45} = s_1 c_2 s_3 \dot{\theta}_1 + c_1 s_2 s_3 \dot{\theta}_{23} - c_1 c_2 c_3 \dot{\theta}_4 - c_1 c_4 \dot{\theta}_1 + s_1 s_4 \dot{\theta}_4 \]
\[ J_{46} = -s_1 c_2 c_3 s_3 \dot{\theta}_1 - c_1 s_2 c_3 s_3 \dot{\theta}_{23} - c_1 c_2 s_3 s_3 \dot{\theta}_4 + c_1 c_2 c_3 s_3 \dot{\theta}_5 - c_1 s_3 s_3 \dot{\theta}_1 - s_1 c_4 \dot{\theta}_4 - s_1 s_4 \dot{\theta}_5 \]
\[ -s_1 s_2 c_5 \dot{\theta}_1 - c_1 s_2 s_5 \dot{\theta}_{23} - c_1 c_2 s_5 \dot{\theta}_3 \]

\[ J_{51} = 0.0 \]
\[ J_{52} = -s_1 \dot{\theta}_1 \]
\[ J_{53} = -s_1 \dot{\theta}_1 \]
\[ J_{54} = c_1 c_2 \dot{\theta}_1 + s_1 c_2 \dot{\theta}_{23} \]
\[ J_{55} = -c_1 c_2 s_1 \dot{\theta}_1 + s_1 s_2 s_1 \dot{\theta}_{23} - s_1 c_2 s_1 \dot{\theta}_4 - s_1 c_4 \dot{\theta}_1 - c_1 s_4 \dot{\theta}_4 \]
\[ J_{56} = c_1 c_2 s_1 \dot{\theta}_1 - s_1 s_2 c_1 \dot{\theta}_{23} - s_1 c_2 s_1 \dot{\theta}_4 + s_1 c_2 c_1 \dot{\theta}_5 - s_1 s_4 c_1 \dot{\theta}_1 + c_1 c_4 s_1 \dot{\theta}_4 + c_1 s_4 c_1 \dot{\theta}_5 \]
\[ + c_1 c_2 c_1 \dot{\theta}_1 + s_1 c_2 c_1 \dot{\theta}_{23} - s_1 s_2 s_1 \dot{\theta}_5 \]

\[ J_{61} = 0.0 \]
\[ J_{62} = 0.0 \]
\[ J_{63} = 0.0 \]
\[ \dot{J}_{64} = -s_{23}\dot{\theta}_{23} \]
\[ \dot{J}_{65} = c_{23}s_{23}\dot{\theta}_{23} + s_{25}c_{4}\dot{\theta}_4 \]
\[ \dot{J}_{66} = -c_{23}s_{23}\dot{\theta}_{23} + s_{23}s_{23}\dot{\theta}_4 - s_{25}c_{4}s_{23}\dot{\theta}_5 - s_{25}c_{4}\dot{\theta}_{23} - c_{23}s_{23}\dot{\theta}_5 \]
Appendix D

D-1 Computer Program to Evaluate the Forward Kinematics, i.e., $T_e$ of the PUMA 560

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */

#define c1 (cos(theta[1]))
#define s1 (sin(theta[1]))

#define c23 (cos((theta[2])+theta[3]))
#define s23 (sin((theta[2])+theta[3]))

#define c2 (cos(theta[2]))
#define s2 (sin(theta[2]))

#define c4 (cos(theta[4]))
#define s4 (sin(theta[4]))

#define c5 (cos(theta[5]))
#define s5 (sin(theta[5]))

#define c6 (cos(theta[6]))
#define s6 (sin(theta[6]))

/* Defining fixed link parameters as per Denavit-Hartenberg notation */

#define a2 (0.4318)
#define a3 (-0.0203)
#define d4 (0.4331)

#define d2 (0.2435)
#define d3 (-0.0934)

#define pi (3.1415927)

float theta[7]; /* Vector of Joint Angles $\theta_1, \theta_2, \ldots, \theta_6$ */
float xe,ye,ze; /* Desired End-Effector position in Cartesian Coordinates */
int i;
float Tdesired[5][5]; /* 4x4 Matrix $T_{desired}$ for End-Effector orientation and position */

main()
{
    void frwd_matrix(); /* Returns $T_{desired}$ */
}
```
APPENDIX D

xe=ye=ze=0.0;

//for(i=0;i<=6;i++)

theta[1]=0.00*pi/180.0;theta[2]=0.00*pi/180.0;theta[3]=0.00*pi/180.0;theta[4]=0.00*pi/180.0;theta[5]=0.00*pi/180.0;theta[6]=0.00*pi/180.0;

frwd_matrix(Tdesired);

xe=Tdesired[1][4];
ye=Tdesired[2][4];
ze=Tdesired[3][4];

printf("xe ye ze \%f \%f \%f\n",xe,ye,ze);
return;
}

void frwd_matrix(Tdesired)
float Tdesired[5][5];
{

Tdesired[1][1] = (c1*c23*c4 - s1*s4)*c5*c6 - c1*s23*s5*c6 - c1*c23*s4*s6 - s1*c4*s6;
Tdesired[1][2] = -(c1*c23*c4 - s1*s4)*c5*s6 - c1*s23*s5*s6 - c1*c23*s4*c6 - s1*c4*c6;
Tdesired[1][3] = (c1*c23*c4 - s1*s4)*s5 + c1*s23*c5;
Tdesired[1][4] = c1*(c23*a3 + s23*d4 + c2*a2) - s1*(d2+d3);

Tdesired[2][1] = (s1*c23*c4 + c1*s4)*c5*c6 - s1*s23*s5*c6 - s1*c23*s4*s6 + c1*c4*s6;
Tdesired[2][2] = -(s1*c23*c4 + c1*s4)*c5*s6 - s1*s23*s5*s6 - s1*c23*s4*c6 + c1*c4*c6;
Tdesired[2][3] = (s1*c23*c4 + c1*s4)*s5 + s1*s23*c5;
Tdesired[2][4] = s1*(c23*a3 + s23*d4 + c2*a2) + c1*(d2+d3);

Tdesired[3][1] = -s23*c4*c5*c6 -c23*s5*c6 + s23*s4*s6;
Tdesired[3][2] = s23*c4*c5*s6 + c23*s5*s6 + s23*s4*c6;
Tdesired[3][3] = -s23*c4*s5 + c23*c5;
Tdesired[3][4] = -s23*a3 + c23*d4 - s2*a2;

Tdesired[4][1] = 0.0;
Tdesired[4][2] = 0.0;
Tdesired[4][3] = 0.0;
Tdesired[4][4] = 1.0;

return;
}
APPENDIX D

D-2 Computer Program to Evaluate the Inverse Kinematics Solution of PUMA 560

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */
#define c1 (cos(theta[1]))
#define s1 (sin(theta[1]))
#define c23 (cos((theta[2])+(theta[3])))
#define c2 (cos(theta[2]))
#define s23 (sin(theta[2]+theta[3]))
#define s2 (sin(theta[2]))
#define c4 (cos(theta[4]))
#define s4 (sin(theta[4]))
#define c5 (cos(theta[5]))
#define s5 (sin(theta[5]))
#define c6 (cos(theta[6]))
#define s6 (sin(theta[6]))

/* Defining fixed link parameters as per Denavit-Hartenberg notation */
#define a2 (0.4318)
#define a3 (-0.0203)
#define d4 (0.4331)
#define d2 (0.2435)
#define d3 (-0.0934)
#define pi (3.1415927)

float theta[7]; /* Vector of Joint Angles θ₁, θ₂, ..., θ₆ */
float xe,ye,ze; /* Desired End-Effector position in Cartesian Coordinates */
int i,j,k;
float Tdesired[5][5]; /* 4x4 Matrix Tdesired for End-Effector orientation and position */

/* Defining variable expressions and constants in Pieper's Solution */
float k11,k21,k31;
float k12,k22,k32;
float kc,kc1;
float r;
float g11,g2;
float g12;
float g13;
float g14;
float lhtrans[4][4]; /* 3x3 Matrix for Evaluating (R₀₋₁) */
```
APPENDIX D

float ihsmtrix[4][4]; /* 3x3 Matrix for Evaluating $(R_3)^{-1}T_{desired}$ for Z-Y-Z Euler Angles Solution */
float angles[7][5]; /* Array for 4 sets of Joint Solutions using Pieper's Solution approach */

main()
{
    void frwd_matrix(); /* Returns $T_{desired}$ */

    void zR3Trans(); /* Returns $(R_3)^{-1}$ */

    for(i=0;i<=7;i++)
        for(j=0;j<=5;j++)
            angles[i][j]=0.0;

    for(i=0;i<=6;i++)
        theta[i]=0.0;

    theta[1]=0.0*pi/180.0;theta[2]=30.0*pi/180.0;theta[3]=30.0*pi/180.0;theta[4]=0.0*pi/180.0;theta[5]=30.0*pi/180.0;
    theta[6]=0.0*pi/180.0;
    printf("the respective angles in DEGREES fed in follow\n");

    for(i=0;i<=7;i++)
        for(j=0;j<=5;j++)
            ihsmtrix[i][j]=0.0;

    for(i=0;i<=6;i++)
        printf("%f
",theta[i]*180.0/pi);

    for(i=0;i<=4;i++)
        for(j=0;i<=4;j++)
            ihsmtrix[i][i]=0.0;

    /*xe=0.4115;
    ye=0.1501;
    ze=0.4331;*/

    frwd_matrix(Tdesired);

    xc=Tdesired[1][4];
    yc=Tdesired[2][4];
    zc=Tdesired[3][4];

    r = xc*xc + yc*yc + zc*zc;
    printf("r %f\n",r);

    kc = (r - 2.0*d2*d3 - d2*d2 - a2*a2 - d3*d3 - d4*d4 - a3*a3)/(2.0*a2);
    kcl = (r - 2.0*d2*d3 - d2*d2 - a2*a2 - d3*d3 - d4*d4 - a3*a3);
    printf("%f %f %f\n",r,kc,kcl);

    **thetaa=2.0*atan((d4+sqrt(d4*d4 + a3*a3 - kc*kc))/(a3+kc));
    thetab=2.0*atan((d4-sqrt(d4*d4 + a3*a3 - kc*kc))/(a3+kc));****/

    angles[3][1]=atan2(d4,a3)+atan2(sqrt(a3*a3 + d4*d4 -kc*kc),kc);
    angles[3][2]=atan2(d4,a3)-atan2(sqrt(a3*a3 + d4*d4 -kc*kc),kc);
    printf("2 values of theta[3] (in deg):----> angles[3][1] angles[3][2] %f %f\n",angles[3][1]*180.0/pi,angles[3][2]*180.0/pi);

    ****************theta[3]=angles[3][2];**********************/

    k11 = a3*cos(angles[3][2]) + d4*sin(angles[3][2]) + a2;
APPENDIX D

\[
\begin{align*}
\text{k21} &= d4 \cdot \cos(\text{angles}[3][2]) - a3 \cdot \sin(\text{angles}[3][2]); \\
\text{k31} &= a3 \cdot a3 + d4 \cdot d4 + d3 \cdot d3 + a2 \cdot a2 + 2.0 \cdot a2 \cdot a3 \cdot \cos(\text{angles}[3][2]) + 2.0 \cdot a2 \cdot d4 \cdot \sin(\text{angles}[3][2]) + d2 \cdot d2 + 2.0 \cdot d2 \cdot d3; \\
\text{printf}(&"k11,k21,k31 \,<\, check k31 \,\text{is same as} \, r\,\, > \, \%f \, \%f \, \%f \, \n",k11,k21,k31);
\end{align*}
\]

**** CAN USE THIS CHECK TO FIND SUITABLE theta[3] AND PROCEED TO FIND theta[2] value(s)*****

\[
\begin{align*}
\text{angles}[2][1] &= \text{atan2}(-k12,k22) + \text{atan2}(\sqrt{k22^2+k12^2} - ze*ze),ze); \\
\text{printf}(&"2 \,\text{values of} \,\text{theta}[2] \,\text{in} \,\text{deg} \,(\text{from angles}[3][2]):----> \, \text{angles}[2][1] \,\text{angles}[2][2] \,\%f \, \%f \, \%f \, \n",\text{angles}[2][1]*180.0/pi,\text{angles}[2][2]*180.0/pi);
\end{align*}
\]

**********theta[2]= angles[2][1];***********/

\[
\begin{align*}
\text{k12} &= a3 \cdot \cos(\text{angles}[3][1]) + d4 \cdot \sin(\text{angles}[3][1]) + a2; \\
\text{k22} &= d4 \cdot \cos(\text{angles}[3][1]) - a3 \cdot \sin(\text{angles}[3][1]); \\
\text{k32} &= a3 \cdot a3 + d4 \cdot d4 + d3 \cdot d3 + a2 \cdot a2 + 2.0 \cdot a2 \cdot a3 \cdot \cos(\text{angles}[3][1]) + 2.0 \cdot a2 \cdot d4 \cdot \sin(\text{angles}[3][1]) + d2 \cdot d2 + 2.0 \cdot d2 \cdot d3; \\
\text{printf}(&"k12,k22,k32 \,<\, check k32 \,\text{is same as} \, r\,\, > \, \%f \, \%f \, \%f \, \n",k12,k22,k32);
\end{align*}
\]

**** CAN USE THIS CHECK TO FIND SUITABLE theta[3] AND PROCEED TO FIND theta[2] value(s)*****

\[
\begin{align*}
\text{angles}[2][3] &= \text{atan2}(-k12,k22) + \text{atan2}(\sqrt{k22^2+k12^2} - ze*ze),ze); \\
\text{printf}(&"2 \,\text{values of} \,\text{theta}[2] \,\text{in} \,\text{deg} \,(\text{from angles}[3][1]):----> \, \text{angles}[2][3] \,\text{angles}[2][4] \,\%f \, \%f \, \%f \, \n",\text{angles}[2][3]*180.0/pi,\text{angles}[2][4]*180.0/pi);
\end{align*}
\]

\[
\begin{align*}
\text{g11} &= \cos(\text{angles}[2][1])* (a3*\cos(\text{angles}[3][2]) + d4*\sin(\text{angles}[3][2]) + a2) - \sin(\text{angles}[2][1])* (a3*\sin(\text{angles}[3][2]) - d4*\cos(\text{angles}[3][2])); \\
\text{g2} &= d3 + d2; \\
\text{printf}(&"g11,g2 \,\%f \,\%f \,\n",\text{g11},\text{g2});
\end{align*}
\]

**********theta[1]=atan2(ye,xe)-atan2(g2,g1);***********/

\[
\begin{align*}
\text{angles}[1][1] &= \text{atan2}(\text{g11*ye-g2*xe},\text{g11*xe+g2*ye}); \\
\text{printf}(&"\text{angles}[1][1] \,\text{in} \,\text{DEGREES} \,\%f \,\%f,\text{angles}[1][1]*180.0/pi);
\end{align*}
\]

**********theta[1]=angles[1][1];***********/

\[
\begin{align*}
\text{g12} &= \cos(\text{angles}[2][2])* (a3*\cos(\text{angles}[3][2]) + d4*\sin(\text{angles}[3][2]) + a2) - \sin(\text{angles}[2][2])* (a3*\sin(\text{angles}[3][2])}
\]
APPENDIX D

- d4*cos(angles[3][2])

printf("g12 %fn", g12);

/****** theta[1] = atan2(ye, xe) - atan2(g2, g1); *******/

angles[1][2] = atan2(g12*ye - g2*xe, g12*xe + g2*ye);

printf("angles[1][2] in DEGREES

angles[1][3] = atan2(g13*ye - g2*xe, g13*xe + g2*ye);

printf("angles[1][3] in DEGREES

angles[1][4] = atan2(g14*ye - g2*xe, g14*xe + g2*ye);

printf("angles[1][4] in DEGREES

if(angles[3][2] || angles[2][1] || angles[1][1])
{
    theta[3] = angles[3][2];
    theta[2] = angles[2][1];
    theta[1] = angles[1][1];

    zR3Trans(lhstrans);

    for(i = 1; i <= 3; i++)
        for(j = 1; j <= 3; j++)
            for(k = 1; k <= 3; k++)
                lhstrans[i][j] = lhstrans[i][j] + lhstrans[i][k]*Tdesired[k][j];


    printf("angles[5][1] in DEGREES

    if(angles[5][1] < 1.0e-04)
APPENDIX D

\{angles[5][1]=0.0
\;angles[4][1]=0.0
\;angles[6][1]=atan2( -lhsmtixo[1][2],lhsmtixo[1][1]
\} ;
\}
else
if(angles[5][1]==pi)
\{angles[5][1]=pi;
\;angles[4][1]=0.0;
\;angles[6][1]=atan2( lhmtrix[1][2],-lhsmtixo[1][1]
\} ;
\}
else
\}
angles[4][1]=atan2( lhmtrix[3][3]/sin(angles[5][1]),lhsmtixo[1][3]/sin(angles[5][1])
\;
angles[6][1]=atan2( -lhsmtixo[2][2]/sin(angles[5][1]),lhsmtixo[2][1]/sin(angles[5][1])
\};
printf("angles[4][1] , angles[5][1] , angles[6][1] in DEG \%f \%f \\
%\%f",angles[4][1]*180.0/pi,angles[5][1]*180.0/pi,angles[6][1]*180.0/pi);

theta[5]=angles[5][1];
\;
\;theta[4]=angles[4][1];
\;
\;theta[6]=angles[6][1];
\;
\}
\}
\;
\}
\}\}
\}\}
\;printf("the respective angles in DEGREES follow\n")
\;for(i=1;i<=6;i++)
\;printf("%f\n",theta[i]*180.0/pi);
\;
\}
if(angles[3][2] || angles[2][2] || angles[1][2])
\{\n\;theta[3]=angles[3][2];
\;\;theta[2]=angles[2][2];
\;\;theta[1]=angles[1][2];
\;
\;zR3Trans(lhstrans);
\;
\;for(i=1;i<=3;i++)
\;for(j=1;j<=3;j++)
\;for(k=1;k<=3;k++)
\;lhmtrix[i][j]=lhmtrix[i][j]+lhstrans[i][j]*Tdesired[k][j];
\;
\};
\}
\;
\;printf("angles[5][2] in DEGREES \%f\n",angles[5][2]*180.0/pi);
\;
\{
\;if(angles[5][2]<=1.0e-04)
\;\{angles[5][2]=0.0;
\;\;angles[4][2]=0.0;
\;\;angles[6][2]=atan2( -lhsmtixo[1][2],lhsmtixo[1][1]
\};
\}
else
\{angles[5][2]=pi;
\;\;angles[5][2]=0.0;
\;\;angles[6][2]=atan2( lhmtrix[1][2],-lhsmtixo[1][1]
\};
else
\);
APPENDIX D

\begin{verbatim}
printf("angles[4][2], angles[5][2], angles[6][2] in DEG \%f \%f \%fn",
    angles[4][2]*180.0/pi, angles[5][2]*180.0/pi, angles[6][2]*180.0/pi);

theta[5] = angles[5][2];
theta[4] = angles[4][2];
theta[6] = angles[6][2];

printf("the respective angles in DEGREES follow\n");
for(i=1;i<=6;i++)
    printf("%fn",theta[i]*180.0/pi);
}

if(angles[3][1] || angles[2][3] || angles[1][3])
{
    theta[3] = angles[3][1];
    theta[2] = angles[2][3];
    theta[1] = angles[1][3];

    zR3Trans(lhstrans);
    for(i=1;i<=3;i++)
        for(j=1;j<=3;j++)
            for(k=1;k<=3;k++)
                lhstrmtrix[i][j] = lhstrmtrix[i][j] + lhstrans[i][j]*Tdesired[k][j];


    printf("angles[5][3] in DEG \%fn", angles[5][3]*180.0/pi);

    if(angles[5][3] <= 1.0e-04)
        { angles[5][3] = 0.0; angles[4][3] = 0.0; angles[6][3] = atan2(-lhstrmtrix[1][2], lhstrmtrix[1][1]); }
    else
        if(angles[5][3] == pi)
            { angles[5][3] = pi; angles[4][3] = 0.0; angles[6][3] = atan2(lhstrmtrix[1][2], -lhstrmtrix[1][1]); }
        else
               angles[4][3] = atan2( lhstrmtrix[3][3]/sin(angles[5][3]), lhstrmtrix[1][3]/sin(angles[5][3]));
               angles[6][3] = atan2(-lhstrmtrix[2][2]/sin(angles[5][3]), lhstrmtrix[2][1]/sin(angles[5][3])); }

    printf("angles[4][3], angles[5][3], angles[6][3] in DEG \%f \%f \%fn",
        angles[4][3]*180.0/pi, angles[5][3]*180.0/pi, angles[6][3]*180.0/pi);

    theta[5] = angles[5][3];
\end{verbatim}
APPENDIX D

\[
\text{theta}[4] = \text{angles}[4][3];
\text{theta}[6] = \text{angles}[6][3];
\]

printf("the respective angles in DEGREES follow\n");
\text{for}(i = 1; i < 6; i++)
printf("\%f\n", \text{theta}[i]*180.0/\text{pi});
\}

if(\text{angles}[3][1] \mid \mid \text{angles}[2][4] \mid \mid \text{angles}[1][4])
{\
\text{theta}[3] = \text{angles}[3][1];
\text{theta}[2] = \text{angles}[2][4];
\text{theta}[1] = \text{angles}[1][4];
}
\text{ZR3Trans(}\text{lhstrans});
\}
\text{for} (i = 1; i < 3; i++)
\text{for} (j = 1; j < 3; j++)
\text{for} (k = 1; k < 3; k++)
\text{lhstrix}[i][j] = \text{lhstrix}[i][j] + \text{lhstrans}[i][k]*T\text{desired}[k][j];

\text{angles}[5][4] = \text{atan}2\left(\text{sqrt} \left( \text{lhstrix}[2][1]*\text{lhstrix}[2][1] + \text{lhstrix}[2][2]*\text{lhstrix}[2][2]\right), -\text{lhstrix}[2][3]\right);}
\text{printf("angles[5][4] in DEGREES \%fn", angles[5][4]*180.0/\text{pi});}
\}
\text{if}(\text{angles}[5][4] < \text{1.0e-04})
{\text{angles}[5][4] = 0.0;
\text{angles}[4][4] = 0.0;
\text{angles}[6][4] = \text{atan}2\left(-\text{lhstrix}[1][2], \text{lhstrix}[1][1]\right);}
\}
\text{else}
\text{if}(\text{angles}[5][4] = \text{pi})
{\text{angles}[5][4] = \text{pi};
\text{angles}[4][4] = 0.0;
\text{angles}[6][4] = \text{atan}2\left(\text{lhstrix}[1][2], -\text{lhstrix}[1][1]\right);}
\}
\text{else}
{\text{angles}[5][4] = \text{atan}2\left(\sqrt{\text{lhstrix}[2][1]*\text{lhstrix}[2][1] + \text{lhstrix}[2][2]*\text{lhstrix}[2][2]}, -\text{lhstrix}[2][3]\right);
\text{angles}[4][4] = \text{atan}2\left(\text{lhstrix}[3][3]/\text{sin(angles[5][4])}, \text{lhstrix}[1][3]/\text{sin(angles[5][4])}\right);
\text{angles}[6][4] = \text{atan}2\left(-\text{lhstrix}[2][2]/\text{sin(angles[5][4])}, \text{lhstrix}[2][1]/\text{sin(angles[5][4])}\right);}
\}
\text{printf("angles[4][4], angles[5][4], angles[6][4] in DEG \%f \%f \%f \%fn", angles[4][4]*180.0/\text{pi}, angles[5][4]*180.0/\text{pi}, angles[6][4]*180.0/\text{pi});}
\}
\text{theta}[5] = \text{angles}[5][4];
\text{theta}[4] = \text{angles}[4][4];
\text{theta}[6] = \text{angles}[6][4];
\}
\text{printf("the respective angles in DEGREES follow\n");}
\text{for}(i = 1; i < 6; i++)
\text{printf("\%f\n", \text{theta}[i]*180.0/\text{pi});}
\}
\}
\text{return;}
\}
void fwd_matrix(Tdesired)
float Tdesired[5][5];
{
    Tdesired[1][1]=(c1*c23*c4 - s1*s4)*c5*c6 - c1*s23*s5*s6 - c1*c23*s4*s6 - 1*c4*s6;
    Tdesired[1][2]=-(c1*c23*c4 - s1*s4)*c5*s6 + c1*s23*s5*s6 - c1*c23*s4*c6 - s1*c4*c6;
    Tdesired[1][3]=(c1*c23*c4 - s1*s4)*s5 + c1*s23*c5;
    Tdesired[1][4]=c1*(c23*a3 + s23*d4 + c2*a2) - s1*(d2+d3);
    Tdesired[2][1]=(s1*c23*c4 + c1*s4)*c5*c6 - s1*s23*s5*s6 - c1*c23*s4*s6 + c1*c4*s6;
    Tdesired[2][2]=-(s1*c23*c4 + c1*s4)*c5*s6 + s1*s23*s5*s6 - s1*c23*s4*c6 + c1*c4*c6;
    Tdesired[2][3]=(s1*c23*c4 + c1*s4)*s5 + s1*s23*c5;
    Tdesired[2][4]=s1*(c23*a3 + s23*d4 + c2*a2) + c1*(d2+d3);
    Tdesired[3][1]=s23*c4*c5*c6 - c23*s5*s6 + s23*s4*s6;
    Tdesired[3][2]=s23*c4*c5*s6 + c23*s5*s6 + s23*s4*c6;
    Tdesired[3][3]=s23*c4*s5 + c23*c5;
    Tdesired[3][4]=s23*a3 + c23*d4 - s2*a2;
    Tdesired[4][1]=0.0;
    Tdesired[4][2]=0.0;
    Tdesired[4][3]=0.0;
    Tdesired[4][4]=1.0;
}

return;
}

void zR3Trans(lhstrans)
float lhstrans[4][4];
{
lhstrans[1][1]=c1*c23;
lhstrans[1][2]=s1*c23;
lhstrans[1][3]=s23;
lhstrans[2][1]=-c1*s23;
lhstrans[2][2]=s1*s23;
lhstrans[2][3]=-c23;
lhstrans[3][1]=-s1;
lhstrans[3][2]=c1;
lhstrans[3][3]=0.0;
}

return;
D-3 Computer Program to Evaluate J and \( \dot{J} \) matrices for PUMA 560

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */
#define s1 (sin(theta[1]))
#define c1 (cos(theta[1]))
#define c2 (cos(theta[2]))
#define s23 (sin(theta[2]+theta[3]))
#define s2 (sin(theta[2]))
#define c23 (cos((theta[2])+(theta[3])))
#define s4 (sin(theta[4]))
#define s5 (sin(theta[5]))
#define c4 (cos(theta[4]))
#define c5 (cos(theta[5]))

/* Defining fixed link parameters as per Denavit-Hartenberg notation */
#define a2 (0.4318)
#define a3 (-0.0203)
#define d4 (0.4331)
#define d2 (0.2435)
#define d3 (-0.0934)
#define pi (3.1415927)

int i;

float theta[7]; /* Vector of Joint Angles \( \theta_1, \theta_2, \ldots, \theta_6 \) */
float jac[7][7]; /* (6x6) Jacobian matrix */
float thdot[7]; /* Vector of \( \dot{\theta} \) */
float jacdot[7][7]; /* (6x6) \( \dot{J} \) matrix */

main()
{
    void jacobian(); /* Returns the (6x6) Jacobian Matrix */
    void jacobdot(); /* Returns the (6x6) \( \dot{J} \) Matrix */

    for(i=0;i<=6;i++)
    {
        theta[i]=0.0;
        thdot[i]=0.0;
    }

    jacobian(jac);
    for(i=1;i<=6;i++)
    printf("jac main %f %f %f %f %f %f
", jac[i][1], jac[i][2], jac[i][3], jac[i][4], jac[i][5], jac[i][6]);
}
```
jacobdot(jacd0t);
for(i=1;i<=6;i++)
printf("jacdot main %f %f %f %f %f %f
",jacd0t[i][1],jacd0t[i][2],jacd0t[i][3],jacd0t[i][4],jacd0t[i][5],jacd0t[i][6]);

return;
}

void jacobian(jac)
float jac[7][7];
{
jac[1][1] = -s1*(c23*a3 + s23*d4 + c2*a2) -c1*(d2+d3);
jac[1][2] = c1*(-s23*a3 + c23*d4 - s2*a2);
jac[1][3] = c1*(-s23*a3 + c23*d4);
jac[1][4] = 0.0;
jac[1][5] = 0.0;
jac[1][6] = 0.0;
jac[2][1] = c1*(c23*a3 + s23*d4 + c2*a2) -s1*(d2+d3);
jac[2][2] = s1*(-s23*a3 + c23*d4 -s2*a2);
jac[2][3] = s1*(-s23*a3 + c23*d4);
jac[2][4] = 0.0;
jac[2][5] = 0.0;
jac[2][6] = 0.0;
jac[3][1] = 0.0;
jac[3][2] = -(c23*a2 + s23*d4 + c2*a2);
jac[3][3] = -(c23*a3 + s23*d4);
jac[3][4] = 0.0;
jac[3][5] = 0.0;
jac[3][6] = 0.0;
jac[4][1] = 0.0;
jac[4][2] = -s1;
jac[4][3] = -s1;
jac[4][4] = c1*s23;
jac[4][5] = -c1*c23*s4 -s1*c4;
jac[4][6] = (c1*c23*c4 -s1*s4)*s5 +c1*s23*c5;
jac[5][1] = 0.0;
jac[5][2] = c1;
jac[5][3] = c1;
jac[5][4] = s1*s23;
jac[5][5] = -s1*c23*s4 +c1*c4;
jac[5][6] = (s1*c23*c4 +c1*s4)*s5 +s1*s23*c5;
jac[6][1] = 1.0;
jac[6][2] = 0.0;
jac[6][3] = 0.0;
jac[6][4] = c23;
jac[6][5] = s23*s4;
jac[6][6] = -s23*c4*s5 +c23*c5;
void jacobdot(jacdot)
float jacdot[7][7];
{
    jacdot[1][4] = 0.0;
    jacdot[1][5] = 0.0;
    jacdot[1][6] = 0.0;
    jacdot[2][4] = 0.0;
    jacdot[2][5] = 0.0;
    jacdot[2][6] = 0.0;
    jacdot[3][1] = 0.0;
    jacdot[3][4] = 0.0;
    jacdot[3][5] = 0.0;
    jacdot[3][6] = 0.0;
    jacdot[4][1] = 0.0;
    jacdot[4][2] = -c1*thdot[1];
    jacdot[4][3] = -c1*thdot[1];
    jacdot[5][1] = 0.0;
    jacdot[5][2] = -s1*thdot[1];
    jacdot[5][3] = -s1*thdot[1];
APPENDIX D

for(i=1;i<=6;i++)
printf("jacdot subr %f %f %f %f %f %f\n",jacdot[i][1],jacdot[i][2],
jacdot[i][3],jacdot[i][4],jacdot[i][5],jacdot[i][6]);

return;
}
APPENDIX D

D-4 Computer Program to Verify the Equations of Motion (Using "Conservation of Energy Approach") and find the computer processor time taken for the Real Model

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */

#define cc2 (cos(theta[2])*cos(theta[2]))
#define ss23 (sin(theta[2])*sin(theta[3]))
#define c2 (cos(theta[2]))
#define s23 (sin(theta[2]+theta[3]))
#define s2 (sin(theta[2]))
#define c23 (cos((theta[2])+(theta[3])))
#define s3 (sin(theta[3]))
#define c3 (cos(theta[3]))
#define s4 (sin(theta[4]))
#define s5 (sin(theta[5]))
#define c4 (cos(theta[4]))
#define c5 (cos(theta[5]))
#define sc2 (sin(theta[2])*cos(theta[2]))
#define sc23 (sin(theta[2])*cos(theta[3]))
#define ss4 (sin(theta[4])*sin(theta[4]))
#define ss5 (sin(theta[5])*sin(theta[5]))
#define cc4 (cos(theta[4])*cos(theta[4]))
#define sc5 (sin(theta[5])*cos(theta[5]))
#define cc3 (cos(theta[2])*cos(theta[3]))
#define sc4 (sin(theta[4])*cos(theta[4]))
#define cc5 (cos(theta[5])*cos(theta[5]))
#define ss2 (sin(theta[2])*sin(theta[2]))

/* Defining Expressions for the Inertial Constants Appearing in the Equations of Forces of Motion */

#define i1 (-0.0142)
#define i2 (-0.0110)
#define i3 (-3.79e-03)
#define i4 (-0.134)
#define i5 (0.0238)
#define i6 (-0.213)
#define i7 (1.43)
#define i8 (1.75)
#define i9 (1.38)
#define i10 (0.690)
#define i11 (0.372)
#define i12 (0.333)
#define i13 (2.98)
#define i14 (-0.134)
#define i15 (0.0238)
#define i16 (-0.213)
APPENDIX D

#define ggl (-37.2)
#define gg2 (-8.44)
#define gg3 (1.02)
#define gg4 (0.249)
#define gg5 (-0.0282)

/* Defining Expressions for the Gravitational Constants Appearing in the Equations of Forces of Motion */

#define gg1 (-37.2)
#define gg2 (-8.44)
#define gg3 (1.02)
#define gg4 (0.249)
#define gg5 (-0.0282)

/* Defining fixed link parameters as per Denavit-Hartenberg notation */

#define TINY 1.0e-20;
#define N (6)

/****for 6x6 "A" ie to find inv. of "ar" matrix as "yy" matrix **********/

float **a,**yy,d,*col;
int i,j,*indx;

float theta[7]; /*(6x1) Vector of Joint Angles \(\theta_1, \theta_2, \ldots, \theta_6\)*/

float ar[7][7],br[7][16],CR[7][7],gr[7]; /* A(6x6), B(6x15), and C(6x6) matrices and the g(6x1) vector */

float **y,*xx=0;

float vel[7],posn[7]; /* Vector of Joint Velocities and Joint Positions */

float kecons,pecons,totenrgy; /* The Scalars Kinetic Energy, Potential Energy and the Total Energy */
main()
{
    void armat(); /* Returns the expressions giving the elements of the Kinetic Energy matrix */
    void brmat(); /* Returns the expressions giving the elements of the Coriolis matrix */
    void CRmat(); /* Returns the expressions giving the elements of the Centrifugal matrix */
    void grmat(); /* Returns the Gravity terms */
    void rk4(),rkdumb(); /* Runge-Kutta integration */
    float vstart[13]; //,dydx[13],yout[13];
    int nvar,nstep;
    float x1,x2;
    void (*derive)();
    void derivative(); /* Returns dy/dx */
    void ludcmp(),lubksb();
    float **matrix();
    float *vector();

    int k;
    float iden[7][7];

    clock_t start,end; /* Defining system clock variables */
    time_t first,second; /* Defining system time variables */

    theta[0]=0.0;

    start = clock(); /* get the no. of system clock ticks per second */
    first=time(NULL); /* get the system time */

    armat(ar);
    brmat(br);
    CRmat(CR);
    grmat(gr);

    derivs=&derivative; /* Passing the address of derivative() as an argument */

    x1=0.0; /* Range */
    x2=1.0;
    nvar=12; /* Number of variables */
    nstep=10; /* Number of steps */

    /* Defining Boundary Conditions and Initial Conditions */

    vstart[1]=theta[1];
    vstart[3]=theta[2];
    vstart[5]=theta[3];
    vstart[7]=theta[4];
    vstart[9]=theta[5];
APPENDIX D


col = vector(1,N);
a = matrix(1,N,1,N);
yy = matrix(1,N,1,N);

for (i = 1; i <= 6; i++)
    for (j = 1; j <= 6; j++)
        a[i][j] = ar[i][j];

ludemp(a,N,indx,&d);
for (j = 1; j <= N; j++) {
    for (i = 1; i <= N; i++)
        col[i] = 0.0;
    col[j] = 1.0;
    lubksb(a,N,indx,col);
    for (i = 1; i <= N; i++)
        yy[i][j] = col[i];
}

for (i = 1; i <= 6; i++)
    printf("matrix yy %f %f %f %f %f %f\n", yy[i][1], yy[i][2], yy[i][3], yy[i][4], yy[i][5], yy[i][6]);

for (i = 0; i <= 6; i++)
    for (j = 0; j <= 6; j++)
        iden[i][j] = 0.0;

for (i = 1; i <= 6; i++)
    for (j = 1; j <= 6; j++)
        for (k = 1; k <= 6; k++)
            iden[i][j] = iden[i][j] + yy[i][k] * ar[k][i];

for (i = 1; i <= 6; i++)
    printf("iden %f %f %f %f %f %f\n", iden[i][1], iden[i][2], iden[i][3], iden[i][4], iden[i][5], iden[i][6]);

rkdump(vstart,nvar,x1,x2,nstep,derivs);

for (i = 0; i <= 7; i++)
    { vel[i] = 0.0;
      posn[i] = 0.0; }

for (k = 1; k <= nstep; k++)
{
    kecons = 0.0;
    pecons = 0.0;
    totenrgy = 0.0;
    for (i = 1; i <= 6; i++)
        for (j = 1; j <= nvar; i++ j = j + 2)
APPENDIX D

\{vel[i]=y[j][k+1];
posn[i]=y[j-1][k+1];\}

printf("xx \%fn",xx[k+1]);
for(i=1;i<=nvar;i++)
printf(" i, k+1, y \%d \%d \%fn",i,k+1,y[i][k+1]);

for(i=1,j=2;i<=6,j++
printf("y1,posn,y2,vel \%f \%f \%f \%f \%f \%f \%fn",y[i-1][k+1],posn[i],y[i][k+1],vel[i]);

printf("Angles \%f \%f \%f \%f \%f \%f \%fn",posn[1]*180/pi,posn[2]*180/pi,
pson[3]*180/pi,posn[4]*180/pi,posn[5]*180/pi,posn[6]*180/pi);

printf("ke,pe,tc before \%f \%f \%f \%f \%fn",kecons,pecons,totenrgy);
for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
kecons = kecons + 0.5*(vel[i]*ar[i][j]*vel[j]);

printf(" kecons = \%f\%n",kecons);
pecons = 9.81*(17.40*(-sin(posn[2])*0.068-cos(posn[2])*0.006) +
+ 17.40*(-sin(posn[2])*0.068-cos(posn[2])*0.006) + 6.04*(
+ posn[3]) * sin ( p o s n [ 4 ] ) * c o s ( p o s n [ 6 ] ) ) * - 0 . 1 4 3
);

printf(" pecons = \%f\%n",pecons);
totenrgy = kecons + pecons;
printf("totenrgy = \%f\%n",totenrgy);

}
end = clock(); /* again get the system clock time after the running of processor */
second=time(NULL); /* again get the system time after the running of processor */
printf("Time elapsed as per system clock was : \%f sec.\n",(end-start)/CLK_TCK);
printf("The elapsed time by difftime is \%f sec.\n",difftime(second,first));
return;
}

void armat(ar)
float ar[7][7];
{
  int i,j;
for(i=0;i<=7;i++)
  for(j=0;j<=7;j++)
    ar[i][j]=y[i][j];
}
APPENDIX D

for(i=0;i<=7;i++)
    ar[i][i]=0.0;

ar[1][1]=(im1+i1+i3*cc2+i7*ss23+i10*sc23+i11*sc2+i20*(ss5*(ss23*(1+cc4))-2.0*ss33*cc4*ss5)+i21*ss23*cc4+2.0*(i5*cc2*ss23+i12*cc2*cc4*ss5+i15*(ss23*ss5+ss23*cc4*ss5)+i16*cc2*(ss23*ss5+cc4*ss5)+i18*ss4*ss5+i22*(ss23*ss5+cc23*cc4*ss5));
ar[1][2]=(i4*ss2+i8*ss2+i9*ss2+i13*cc2-i15*cc2*ss4*ss5+i16*ss2*ss4*ss5+i18*(ss23*cc4*ss5*cc23*cc5)+i19*ss23*ss4+120*ss4*(ss23*cc4*ss5+cc23*ss5)+i22*ss23*ss4*ss5);
ar[1][3]=(i4*cc2+i13*cc2+115*ss23*ss4*ss5+i19*cc2*ss4*ss5+i22*ss23*ss4*ss5);
ar[1][4]=(i15*ss23*ss4*ss5+i16*cc4*ss4*ss5+i17*ss23*ss4*ss4*ss5+i18*(ss23*ss5*cc23*cc4*ss5)+i22*ss23*ss4*ss5);
ar[1][5]=(i15*ss23*ss4*ss5+i16*cc2*ss4*ss5+i17*ss23*ss4*ss5+i18*(ss23*ss5*cc23*cc4*ss5)+i22*ss23*ss4*ss5);
ar[2][1]=(i2*i2+i6+i6+i20*ss*ss5+i21*ss4*ss4+2.0*(i5*ss3+i12*ss3+i15*ss5+i16*(ss3*ss5+cc4*ss5)+i22*cc4*ss5));
ar[2][2]=(i5*ss3+i6+i12*ss3+i16*ss3*ss3*ss3*ss5+i20*ss4*ss5+i21*ss4+2.0*(i15*ss5+i22*cc4*ss5));
ar[2][3]=(i15*cc4*ss5+i16*ss3*ss5+ss3*cc4*ss5+i17*cc4+i22*ss5);
ar[2][4]=(i15*cc4*ss5+i16*ss3*ss5+ss3*cc4*ss5+i17*cc4+i22*ss5);
ar[2][5]=(i15*ss4*ss5);
ar[3][1]=(i21*ss4*ss5+i20*ss4*ss5+i21*ss4*ss4*2*(i15*cc5+i22*cc4*ss5));
ar[3][2]=(-i15*ss4*ss5+i20*ss4*ss5);
ar[3][3]=(i15*cc4*ss5+i17*cc4+i22*ss5);
ar[3][4]=(i23*ss4*ss5);
ar[4][1]=(i4+i14-i20*ss5);
ar[4][2]=(0.0);
ar[4][3]=(i23*cc5);
ar[4][4]=(i23*cc5);
ar[5][1]=(i5+i17);
ar[5][2]=(0.0);
ar[5][3]=(i6+i23);

for(i=0;i<=7;i++)
    for(j=0;j<=7;j++)
        ar[i][j]=ar[i][i];

for(i=1;i<=7;i++)
    printf("ar : armat %f %f %f %f %f %f %f",ar[i][1],ar[i][2],ar[i][3],ar[i][4],ar[i][5],ar[i][6]);

return;
}

void brmat(br)
float br[7][16];
{
    int i,j;
for (i = 0; i < 7; i++)
for (j = 0; j < 16; j++)
br[i][j] = 0.0;

br[1][1] = (2.0*(-13*s23 + 15*c23 + 17*s23 - 12*s23 + 115*(2.0*s23*c5 + (1-2.0*s23)*c4*s5) + 116*(c23*c5 - s23*c4*s5) + i12*s23*c4 + i20*((1+c4)*s23*s5*(1-2.0*s23)*c4*s5) + i22*((1-2.0*s23)*c5 - 2.0*s23*c4*s5)) + i10*(1 - 2.0*s23) + i11*(1 - 2.0*s23));
br[1][2] = (2.0*(i5*c23 + 17*s23 - 12*c23 + 15*(2.0*s23*c5 + (1-2.0*s23)*c4*s5) + i16*c23*c5 - s23*c4*s5) + i21*s23*c4 + i20*((1+c4)*s23*s5*(1-2.0*s23)*c4*s5) + i22*((1-2.0*s23)*c5 - 2.0*s23*c4*s5)) + i10*(1 - 2.0*s23));
br[1][3] = (2.0*(-i15*s23*c4*s5 - i16*c23*c4*s5 + i18*c4*s5 - i20*(s23*s5*sc4 - s23*c4*s5) - i22*c23*c4*s5 - i21*s23*c4*s5));
br[1][4] = (2.0*i20*(s23*c4*(1 - c23) - c23) - c23*c4*(1 - 2.0*s23)) - i15*(s23*s5 - s23*c4*s5) - i16*c23*s5 - c23*c4*s5) + i18*c4*s5 + i22*(c23*c4*c5 - sc23*c4*s5));
br[1][5] = (0.0);
br[1][6] = (2.0*(-i8*s23 + i13*c23 + i15*s23*c4*s5 + i18*c23*c4*s5 + s23*c5) + i19*c23*c4 + i20*s4*(c23*c4*c5 - s23*c4*s5) + i22*c23*c4*s5));
br[1][7] = (-i18*2*s23*c4*s5 + i19*s23*(1 - 2.0*s23)) + i20*s23*(1-2.0*s23*c4*c5) - i14*s23;
br[1][8] = (i17*s23*c4 + i18*2.0*s23*c4*c5 + c23*s5) + i20*s4*c23*(1 - 2.0*s23) - s23*c4*2.0*s23);
br[1][9] = (-i23*s23*c5 + c23*c4*s5);}
br[1][10] = (br[1][7]);
br[1][11] = (br[1][8]);
br[1][12] = (br[1][9]);
br[1][13] = (2.0*(i15*s23*c4*c5 + i16*c23*c4*c5 + i18*s23*c4*c5 + i22*c23*c4*c5) + i17*s23*c4*i20*(s23*c4*(1 - 2.0*s23) + 2.0*c23*c4*c5));
br[1][14] = (23*s23*c4*s5);
br[1][15] = (-i23*(c23*c5 + s23*c4*c5));

br[2][1] = (0.0);
br[2][2] = (0.0);
br[2][3] = (i14*s23 + i19*s23*(1 - 2.0*s23)) + 2.0*(-i15*c23*c4*s5 + i16*s2*c4*s5 + i20*(s23*c4*c5 - c23*c4*s5) + i22*s23*c4*s5));
br[2][4] = (2.0*(i15*s23*c4*c5 + i22*s23*c4*c5 + i16*s23*s4*c5) - i17*s23*c4 + i20*(c23*c4*(1 - 2.0*s23) - 2.0*c23*c4*s5));
br[2][5] = (-br[1][9]);
br[2][6] = (2.0*(i12*s3 + i5*c3 + i16*(c3*c5 - s3*c4*s5)));
br[2][7] = (2.0*(i16*c4*s4*s5 + i20*s4*c23*s5 + i21*c4 - i22*s4*s5));
br[2][8] = (2.0*(-i15*s5 + i16*(c3*c4*c5 - s3*s5) + i20*s4*c23*s5 + i22*c4*c5));
br[2][9] = (0.0);
br[2][10] = (br[2][7]);
br[2][11] = (br[2][8]);
br[2][12] = (0.0);
br[2][13] = (2.0*(-i15*s4*c5 - i16*s3*c4*c5) - i17*s4 + i20*s4*(1 - 2.0*s23));
br[2][14] = (23*c4*s5);
br[2][15] = (23*c4*s5);

br[3][1] = (0.0);
br[3][2] = (0.0);
br[3][3] = (2.0*(-i15*c23*c4*c5 + i22*s23*c4*c5 + i20*(s23*c4*c5 - 0.5) + c23*c4*c5)) + i14*s23 + i19*s23*(1 - 2.0*s23));
br[3][4] = (2.0*(-i15*c23*c4*c5 + i22*s23*c4*c5) - i17*c23*c4 + i20*s4*c23*(1 - 2.0*s23) - 2.0*s23*c4*c5));
br[3][5] = (-br[1][12]);
APPENDIX D

\[
\begin{align*}
br[3][6] &= (0.0); \\
br[3][7] &= (2.0*(i20*sc4*ss5 + i21*sc4 - i22*sc4*s5)); \\
br[3][8] &= (2.0*(-i15*s5 + i20*ss4*sc5 + i22*sc4*c5)); \\
br[3][9] &= (0.0); \\
br[3][10] &= (br[3][7]); \\
br[3][11] &= (br[3][8]); \\
br[3][12] &= (0.0); \\
br[3][13] &= (-i15*2.0*sc4*c5 - i17*s4 + i20*s4*(1.0 - 2.0*ss5)); \\
br[3][14] &= (br[2][14]); \\
br[3][15] &= (br[2][15]); \\
br[4][1] &= (-br[2][3]); \\
br[4][2] &= (-br[3][3]); \\
br[4][3] &= (0.0); \\
br[4][4] &= (-i20*(s23*sc4*(1.0 - 2.0*ss5) + 2.0*s23*sc5) - i17*s23*c4); \\
br[4][5] &= (-br[1][14]); \\
br[4][6] &= (-br[3][7]); \\
br[4][7] &= (0.0); \\
br[4][8] &= (i17*s4 + i20*s4*(1.0 - 2.0*ss5)); \\
br[4][9] &= (-br[2][14]); \\
br[4][10] &= (0.0); \\
br[4][11] &= (br[4][8]); \\
br[4][12] &= (-br[3][14]); \\
br[4][13] &= (-i20*2.0*sc5); \\
br[4][14] &= (0.0); \\
br[4][15] &= (-i23*s5); \\
br[5][1] &= (-br[2][4]); \\
br[5][2] &= (-br[3][4]); \\
br[5][3] &= (-br[4][4]); \\
br[5][4] &= (0.0); \\
br[5][5] &= (-br[1][15]); \\
br[5][6] &= (-br[3][8]); \\
br[5][7] &= (-br[4][8]); \\
br[5][8] &= (0.0); \\
br[5][9] &= (-br[2][15]); \\
br[5][10] &= (br[5][7]); \\
br[5][11] &= (0.0); \\
br[5][12] &= (-br[3][15]); \\
br[5][13] &= (0.0); \\
br[5][14] &= (-br[4][15]); \\
br[5][15] &= (0.0); \\
br[6][1] &= (br[1][9]); \\
br[6][2] &= (br[1][12]); \\
br[6][3] &= (br[1][14]); \\
br[6][4] &= (br[1][15]); \\
br[6][5] &= (0.0); \\
br[6][6] &= (0.0); \\
br[6][7] &= (br[2][14]); \\
br[6][8] &= (br[2][15]); \\
br[6][9] &= (0.0); \\
br[6][10] &= (br[6][7]);
\end{align*}
\]
void CRmat(CR)
float CR[7][7];
{
int i,j;
extern float br[7][16];

for(i=0;i<=7;i++)
  for(j=0;j<=7;j++)
    CR[i][j]=0.0;

CR[1][1]=(-0.5*br[1][1]);
CR[2][2]=(-0.5*br[2][2]);
CR[3][3]=(0.5*br[2][6]);
CR[2][4]=(-i15*o4*s5 - i16*s3*c4*s5 + i20*c4*s5 );
CR[2][5]=(-i15*o4*s5 + i16*(c3*c5 - s3*c4*s5) + i22*c5 );
CR[2][6]=(0.0);

CR[3][1]=(-0.5*br[1][2]);
CR[3][2]=(-CR[2][3]);
CR[3][3]=(0.0);
CR[3][4]=(-i15*o4*s5 + i20*c4*s5 );
CR[3][5]=(-i15*o4*s5 + i22*c5 );
CR[3][6]=(0.0);

CR[4][1]=(-0.5*br[1][3]);
CR[4][2]=(-0.5*br[2][7]);
CR[4][3]=(0.5*br[4][6]);
CR[4][4]=(0.0);
CR[4][5]=(0.0);
CR[4][6]=(0.0);
for(i=1;i<7;i++)
printf("CR %f %f %f %f %f
",CR[i][1],CR[i][2],CR[i][3],CR[i][4],CR[i][5],CR[i][6]);
for (i=1;i<=6;i++)
return;
}

void grmat(gr)
float gr[7];
{
int i;
gr[1]= (0.0);
gr[2]= (gg1*c2 + gg2*s23 + gg3*s2 + gg4*c23 + gg5*(s23*c5 + c23*c4*s5 ));
gr[3]= (gg2*s23 + gg4*c23 + gg5*(s23*c5 + c23*c4*s5 ));
gr[4]= (-gg5*s23*s4*s5);
gr[5]= (gg5*(c23*s5 + s23*c4*c5 ));
gr[6]= (0.0);
for(i=1;i<=6;i++)
printf("gr %f",gr[i]);
return;
}

void rk4(y,dydx,n,x,h,yout,derivs)
float y[],dydx[],x,h,yout[];
void (*derivs)();

int n;
{
int i;
float xh, hh, h6,*dym, *dyy, *yt, *vector();
void free_vector();

dym = vector(1,n);
dyy = vector(1,n);
yt = vector(1,n);
hh = h*0.5;
h6 = h/6.0;
\[ x_h = x + hh; \]
for \( i = 1 ; i \leq n ; i++ \) \[ y[t][i] = y[i] + hh \cdot dydx[i]; \]
(*derivs)(xh,yt,dyt);
for \( i = 1 ; i \leq n ; i++ \) \[ y[t][i] = y[i] + hh \cdot dyt[i]; \]
(*derivs)(xh,yt,dym);
for \( i = 1 ; i \leq n ; i++ \) {
  \[ y[t][i] = y[i] + h \cdot dym[i]; \]
  dym[i] += dyt[i];
}
(*derivs)(x + h,yt,dyt);
for \( i = 1 ; i \leq n ; i++ \) \[ yout[i] = y[i] + h^6 \cdot (dydx[i] + dyt[i] + 2.0 \cdot dym[i]); \]
free_vector(yt,1,n);
free_vector(dyt,1,n);
free_vector(dym,1,n);

float *vector(nl,nh)
int nl,nh;
{
  void nrerror();
  float *v;
  v=(float *)malloc((unsigned) (nh-nl+1) * sizeof(float));
  if (!v) nrerror ("allocation failure in vector()");
  return v-nl;
}

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
  free((char*) (v+nl));
}

void nrerror(error_text)
char error_text[];
{
  void exit();
  fprintf(stderr,"Numerical Recipies run-time error ...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
}

void rkdumb(vstart,nvar,x1,x2,nstep,derivs)
int nvar,nstep;
float vstart[],x1,x2;
void (*derivs)();
{
  int i,k;
  float x,h;
  float *v,*vout,*dv,*vector(),**matrix();
void rk4(), nrerror(), free_vector();
float kemat[7];

v = vector(1, nvar);
vout = vector(1, nvar);
dv = vector(1, nvar);
x = vector(1, nvar);
y = matrix(1, nvar, 1, nvar);

for (i = 0; i <= 7; i++)
kemat[i] = 0.0;
for (i = 1; i <= nvar; i++) {
    v[i] = vstart[i];
    y[i][1] = v[i];
}
x[1] = x1;
x = x1;
h = (x2 - x1) / nstep;
for (k = 1; k <= nstep; k++) {
    (*derivs)(x, v, dv);
rk4(v, dv, nvar, x, h, vout, derivs);
    if (((float)(x + h) == x) nrerror("Step size too small in routine RKDUMB");
    x += h;
x[k + 1] = x;
    for (i = 1; i <= nvar; i++) {
        v[i] = vout[i];
y[i][k + 1] = v[i];
    }
}
free_vector(dv, 1, nvar);
free_vector(vout, 1, nvar);
free_vector(v, 1, nvar);

float **matrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
int i;
float **m;
m = (float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in matrix()");
m -= nrl;
for (i = nrl ; i <= nrh; i++) {
    m[i] = (float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
    if (!m[i]) nrerror("allocation failure 2 in matrix()");
m[i] -= ncl;
}
return m;
This evaluates $\dot{\theta} = A^{-1}(\tau - B(\theta)\dot{\theta} - C(\theta)\dot{\theta}^2) - g(\theta)$, which is integrated by RK-4 method to yield joint velocities i.e., $\dot{\theta}$ and joint positions i.e., $\theta$.

void derivative(x,y,dydx)
float x,y[],dydx[];
{
    void armat();
    void brmat();
    void CRmat();
    void grmat();

    int i,j;

    for(i=1,j=2;i<=2*N-1,j<=2*N;i+=1,j+=2)
    dydx[j]=y[i];
    dydx[1]=y[2];
    dydx[3]=y[4];
    dydx[5]=y[6];
    dydx[7]=y[8];
    dydx[9]=y[10];
}

for(i=1,j=2;i<=2*N-1,j<=2*N;i+=1,j+=2)
            CR[i][6]*y[12]*y[12] + gr[i]);
    //confusion also wrong m2 y1sq. y2sq. for CR
    dydx[]=-(br[i][1]*y[1]*y[3] + br[i][2]*y[1]*y[5] +
    // printf("dydx-first %f %f %f %f %f\n",dydx[2],dydx[4],dydx[6],dydx[8],dydx[10],dydx[12]);
            yy[1][6]*dydx[12];
            yy[2][6]*dydx[12];
            yy[3][6]*dydx[12];
            yy[4][6]*dydx[12];
             yy[5][6]*dydx[12];
                yy[6][6]*dydx[12];
    // printf("dydx-second %f %f %f %f %f\n",dydx[2],dydx[4],dydx[6],dydx[8],dydx[10],dydx[12]);
APPENDIX D

/* for(i=1;i<=6;i++)
    printf("dydx matrix yy %f %f %f %f %f %f \n",yy[i][1],yy[i][2],yy[i][3],yy[i][4],yy[i][5],yy[i][6]); */
}

void ludcmp(a,n,indx,d)
int n,*indx;
float **a,*d;
{
    int i,imax,j,k;
    float big,dum,temp;
    float *vv,*vector();
    void nrerror(),free_vector();
    vv=vector(1,n);
    *d=1.0;
    for(i=1;i<=n;i++) { 
        big=0.0;
        for(j=1;j<=n;j++)
            if ((temp=fabs(a[i][j])) > big) big=temp;
        if (big == 0.0) nrerror("Singular matrix in routine LUDCMP");
        vv[i]=1.0/big;
    }
    for(j=1;j<=n;j++) { 
        for(i=1;i<j;i++)
            sum=a[i][i];
        for (k=1;k<i;k++)
            sum -= a[i][k]*a[k][j];
        a[i][i]=sum;
        if ((dum=vv[i]*fabs(sum)) >= big) {
            big=dum;
            imax=i;
        }
    }
    if (j != imax) {
        for(k=1;k<=n;k++)
            dum=a[imax][k];
        a[imax][k]=a[i][k];
        a[i][k]=dum;
    }
    *d = -*d;
    vv[imax]=vv[i];
}
indx[j]=imax;
if (a[j][j] == 0.0) a[j][j]=TINY;
if(j != n) {
    dum=1.0/(a[j][j]);
void lubksb(a,n,indx,b)
float **a,b[];
int n,*indx;
{
int i,ii=0,ip,j;
float sum;
for(i=1;i<=n;i++) {
   ip=indx[i];
   sum=b[ip];
   b[ip]=b[i];
   if (ii)
      for(i=ii;i<i-ii;i++) sum -= a[i][ip]*b[i];
   else if (sum)
      ii=i;
   b[i]=sum;
}
for(i=n;i>=1;i--){
   sum=b[i];
   for(j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
   b[i]=sum/a[i][i];
}
}
APPENDIX D

D-5 Computer Program to Verify the Equations of Motion (Using "Conservation of Energy Approach") and find the computer processor time taken for the Approximate Model

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define TINY 1.0e-20;
#define N (6)  /*for 6X6 A" ie to find inv. of "ap" matrix as "yy" matrix ***********/

float **a,**yy,d,*col;
int i,j,*indx;

float theta[6];  /* (6x1) Vector of Joint Angles \theta_1, \theta_2, \ldots, \theta_6 */
float ap[6][6],bp[6][16],CP[6][7],gp[7];  /* A(6x6), B(6x15), and C(6x6) matrices and the g vector */
float **y,**xx=0;

float vel[7],posn[7];  /* Vector of Joint Velocities and Joint Positions */
```
APPENDIX D

float kecons,pecons,totenrgy; /* The Scalars Kinetic Energy, Potential Energy, and the Total Energy */

main()
{
  void apmat(); /* Returns the expressions giving the elements of the Kinetic Energy matrix */
  void bpmat(); /* Returns the expressions giving the elements of the Coriolis matrix */
  void CPmat(); /* Returns the expressions giving the elements of the Centrifugal matrix */
  void gpmat(); /* Returns the Gravity vector */

  void rk4(),rkdumb(); /* Runge-Kutta Integration */
  float vstart[13]; //=dydx[13],yout[13];
  int nvar,nstep;
  float x1,x2;
  void (*derivs)();
  void derivative(); /* Returns dy/dx */

  void ludcmp(),lubksb();
  float **matrix();
  float *vector();

  int k;
  float iden[7][7];

clock_t start,end; /* Defining system clock variables */
time_t first,second; /* Defining system time variables */

  theta[0]=0.0;

  start = clock(); /* get the no. of system clock ticks per second */
  first=time(NULL); /* get the system time */

  apmat(ap);
  bpmat(bp);
  CPmat(CP);
  gpmat(gp);

  derivs=&derivative; /* Passing the address of derivative() as an argument */

  x1=0.0; /* Range */
  x2=1.0;
  nvar=12; /* Number of Variables */
  nstep=10; /* Number of Steps */

  /* Defining Boundary Conditions and Initial Conditions */

  vstart[1]=theta[1];
  vstart[3]=theta[2];
APPENDIX D

vstart[5]=theta[3];
vstart[7]=theta[4];
vstart[9]=theta[5];


col=vector(1,N);
a=matrix(1,N,1,N);
yy=matrix(1,N,1,N);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
a[i][j]=ap[i][j];

ludcmp(a,N,indx,&d);
for(j=1;j<=N;j++){
for(i=1;i<=N;i++)col[i]=0.0;
col[j]=1.0;
lubksb(a,N,indx,col);
for(i=1;i<=N;i++)yy[i][j]=col[i];
}

for(i=1;i<=6;i++)
printf("matrix yy %f %f %f %f %f %f \n",yy[i][1],yy[i][2],yy[i][3],yy[i][4],yy[i][5],yy[i][6]);

for(i=0;i<=6;i++)
for(j=0;j<=6;j++)
iden[i][j]=0.0;

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
for(k=1;k<=6;k++)
iden[i][j]=iden[i][j]+yy[i][k]*ap[k][j];

for(i=1;i<=6;i++)
printf("iden %f %f %f %f %f %f \n",iden[i][1],iden[i][2],iden[i][3],iden[i][4],iden[i][5],iden[i][6]);

rkdump(vstart,nvar,x1,x2,ntstep,derivs);

for(i=0;i<=7;i++)
{vel[i]=0.0;
posn[i]=0.0;}

for(k=1;k<=ntstep;k++)
{
kecons=0.0;
pecons=0.0;
APPENDIX D

\[ \text{tetenrgy} = 0.0; \]

\[
\text{for}(i=1; i<6; i++) \quad \{ \text{vel}[i] = y[i][k+1]; \\
\text{posn}[i] = y[i-1][k+1]; \}
\]

\[ \text{printf("\%f\n",xx[k+1]);}
\]

\[
\text{for}(i=1; i<\text{nvar}; i++) \quad \text{printf(" i, k+1, y \%d \%d \%f \%f \%f \%f \%f \%f\n", i, k+1, y[i][k+1]);}
\]

\[
\text{for}(i=1; i<6; i++) \quad \{ \text{kecons} = \text{kecons} + 0.5*(\text{vel}[i]*\text{ap}[i][i]*\text{vel}[i]); \\
\text{pecons} = 9.81*0.740*(-\sin(\text{posn}[2])*0.068-\cos(\text{posn}[2])*0.006)+4.8*(-\cos(\text{posn}[2]+\\text{posn}[3])-0.070\sin(\text{posn}[2])]*a2) + \\
0.02*(\cos(\text{posn}[2]+\\text{posn}[3])*-0.019-\sin(\text{posn}[2]+\\text{posn}[3])*a3 + \cos(\text{posn}[2]+\\text{posn}[3])*d4 -\sin(\text{posn}[2])*a2) + \\
0.34*(-\sin(\text{posn}[2]+\\text{posn}[3])*a3 + \cos(\text{posn}[2]+\\text{posn}[3])*d4 -\sin(\text{posn}[2])*a2) + \\
0.09*(-\sin(\text{posn}[2]+\\text{posn}[3])*\cos(\text{posn}[4])*\sin(\text{posn}[5]) + \cos(\text{posn}[2]+\\text{posn}[3])*\cos(\text{posn}[5])*0.032 -\sin(\text{posn}[2]+\\text{posn}[3])*a3 + \cos(\text{posn}[2]+\\text{posn}[3])*d4 -\sin(\text{posn}[2])*a2); \\
\text{printf(" pecons = \%f\n",pecons);}
\]

\[ \text{tetenrgy} = \text{kecons} + \text{pecons}; \]

\[ \text{printf("tetenrgy = \%f\n",tetenrgy);} \]

\]

\text{end = clock(); /* again get the system clock time after the running of processor */}

\text{second = time(NULL); /* again get the system time after the running of processor */}

\[ \text{printf("Time elapsed as per system clock was : \%f secs.\n",(end-start)/CLK_TCK);}
\]

\[ \text{printf("The elapsed time by diffiime is \%f secs.\n",diffiime(second,first));}
\]

\text{return;}
\]

void apmat(ap)
float ap[7][7];
{ 
int i,j;

\[
\text{for}(i=0; i<7; i++) \quad \{ \text{ap}[i][i] = 0.0; \\
\}
\]
```c
APENDIX D

float bp[7][16];
{
    int i,j;
    for(i=0;i<7;i++)
        for(j=0;j<16;j++)
            bp[i][j]=0.0;

    bp[1][1] = (-2.70*sc2 + 0.744*c223 + 0.60*sc23 -0.0213*(1-2*ss23) );
    bp[1][2] = (0.744*c2*c23 +0.60*sc23 + 0.0222*c2*ss23 -0.0213*(1-2*ss23) );
    bp[1][3] = (-2.5e-03*sc23*s4*s5 + 8.60e-04*c4*s5 -2.48e-03*c2*c23*s4*s5);  
    bp[1][4] = (-2.5e-03*(ss23*s5-ssc23*c4*c5) - 2.48e-03*c2*(s23*s5-c23*c4*c5) + 8.60e-04*s4*c5 );
    bp[1][5] = (0.0);  
    bp[1][6] = (0.267*s23 -7.58e-03*c23 );
```
\begin{verbatim}
bp[1][7] = (0.0);
bp[1][8] = (0.0);
bp[1][9] = (0.0);
bp[1][10] = (bp[1][7]);
bp[1][11] = (bp[1][8]);
bp[1][12] = (bp[1][9]);
bp[1][13] = (0.0);
bp[1][14] = (0.0);
bp[1][15] = (0.0);

bp[2][1] = (0.0);
bp[2][2] = (0.0);
bp[2][3] = (1.64e-03*s23 - 2.50e-03*c23*c4*s5 + 2.48e-03*s2*c4*s5 + 0.3e-03*s23*(1-(2*ss4)));
bp[2][4] = (-2.50e-03*c23*s4*c5 + 2.48e-03*s2*s4*c5 - 6.42e-04*c23*s4);
bp[2][5] = (-bp[1][9]);
bp[2][6] = (0.0220*s3 + 0.7444*e3);
bp[2][7] = (-2.48e-03*c3*s4*s5);
bp[2][8] = (-2.50e-03*s5 + 2.48e-03*(c3*c4*c5*s3*s5));
bp[2][9] = (0.0);
bp[2][10] = (bp[2][7]);
bp[2][11] = (bp[2][8]);
bp[2][12] = (0.0);
bp[2][13] = (0.0);
bp[2][14] = (0.0);
bp[2][15] = (0.0);

bp[3][1] = (0.0);
bp[3][2] = (0.0);
bp[3][3] = (-2.50e-03*c23*c4*s5 + 1.64e-03*s23 + 0.3e-03*s23*(1-2*ss4));
bp[3][4] = (-2.50e-03*c23*s4*c5 - 6.42e-04*c23*s4);
bp[3][5] = (-bp[1][12]);
bp[3][6] = (0.0);
bp[3][7] = (0.0);
bp[3][8] = (-2.50e-03*s5);
bp[3][9] = (0.0);
bp[3][10] = (bp[3][7]);
bp[3][11] = (bp[3][8]);
bp[3][12] = (0.0);
bp[3][13] = (-2.5e-03*s4*c5);
bp[3][14] = (bp[2][14]);
bp[3][15] = (bp[2][15]);
\end{verbatim}
APPENDIX D

for (i=1; i<6; i++)
{ printf("br %f %f %f %f %f %f\n", bp[i][1], bp[i][2], bp[i][3], bp[i][4], bp[i][5], bp[i][6], bp[i][7]);
 printf(" %f %f %f %f %f %f\n", bp[i][8], bp[i][9], bp[i][10], bp[i][11], bp[i][12], bp[i][13], bp[i][14], bp[i][15]);
} // printf("from br br11 = \%fn", bp[1][1]);

return;
}

void CPmat( CP)
float CP[7][7];
    
    int i, j;
extern float bp[7][16];

for(i=0;i<7;i++)
for(j=0;j<7;j++)
CP[i][j]=0.0;

//printf("from inside CR br11 = %f\n",bp[i][j]);

CP[1][1]=(0.0);
CP[1][2]=(0.69*e2 + 0.134*s23 -0.0238*s2 );
CP[1][3]=(0.5*bp[1][6]);
CP[1][4]=(0.0);
CP[1][5]=(0.0);
CP[1][6]=(0.0);

CP[2][1]=(-0.5*bp[1][1]);
CP[2][2]=(0.0);
CP[2][3]=(0.5*bp[2][6]);
CP[2][4]=(0.0);
CP[2][5]=(0.0);
CP[2][6]=(0.0);

CP[3][1]=(-0.5*bp[1][2]);
CP[3][2]=(-CP[2][3]);
CP[3][3]=(0.0);
CP[3][4]=(-1.25c-03*c4*s5);
CP[3][5]=(CP[3][4]);
CP[3][6]=(0.0);

CP[4][1]=(-0.5*bp[1][3]);
CP[4][2]=(-0.5*bp[2][7]);
CP[4][3]=(0.5*bp[4][6]);
CP[4][4]=(0.0);
CP[4][5]=(0.0);
CP[4][6]=(0.0);

CP[5][1]=(-0.5*bp[1][4]);
CP[5][2]=(-0.5*bp[2][8]);
CP[5][3]=(0.5*bp[5][6]);
CP[5][4]=(-0.5*bp[4][13]);
CP[5][5]=(0.0);
CP[5][6]=(0.0);

CP[6][1]=(0.0);
CP[6][2]=(0.0);
CP[6][3]=(0.0);
CP[6][4]=(0.0);
CP[6][5]=(0.0);
CP[6][6]=(0.0);

for(i=1;i<7;i++)
printf("CR %f %f %f %f %f %fn",CP[i][1],CP[i][2],CP[i][3],CP[i][4],CP[i][5],CP[i][6]);
return;
void gpmat(gp)
float gp[7];
{
int i;
gp[1]= (0.0);
gp[2]= (-37.2*c2 - 8.4*s23 + 1.02*s2);
gp[3]= (-8.4*s23 + 0.25*c23);
gp[4]= (2.8e-02*s23*s4*s5);
gp[5]= (-2.8e-02*(c23*s5 + s23*c4*c5));
gp[6]= (0.0);
for(i=1;i<=6;i++)
printf("gr %fun",gp[i]);
return;
}

void rk4(y,dydx,n,x,h,yout,derivs)
float y[],dydx[],x,h,yout[];
void (*derivs)();

int n;
{
int i;
float xh, hh, h6,*dym,*dyt,*yt,*vector();
void free_vector();
dym=vector(1,n);
dyt=vector(1,n);
yt=vector(1,n);

hh=h*0.5;
h6=h/6.0;
xh=x+hh;
for (i=1;i<=n;i++)
yt[i]=y[i]+hh*dydx[i];
(*derivs)(xh,yt,dyt);
for (i=1;i<=n;i++)
yt[i]=y[i]+hh*dyt[i];
(*derivs)(xh,yt,dym);
for (i=1;i<=n;i++){
yt[i]=y[i]+h*dym[i];
dym[i] += dyt[i];
}
(*derivs)(x+h,yt,dyt);
for (i=1;i<=n;i++)
yout[i]=y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]);
free_vector(yt,1,n);
free_vector(dyt,1,n);
free_vector(dym,1,n);
}

float *vector(nl,nh)
int nl,nh;
APPENDIX D

{  
  void nrerror();  
  float *v;  
  v = (float *)malloc((unsigned) (nh-nl+1)*sizeof(float));  
  if (!v) nrerror ("allocation failure in vector()");  
  return v-nl;  
}

void free_vector(v,nl,nh)  
float *v;  
int nl,nh;  
{  
    free((char*) (v+nl));  
}

void nrerror(error_text)  
char error_text[];  
{  
  void exit();  
  fprintf(stderr,"Numerical Recipes run-time error ...
");  
  fprintf(stderr,"%s",error_text);  
  fprintf(stderr,"...now exiting to system...
");  
  exit(1);  
}

void rkdumb(vstart,nvar,x1,x2,nstep,derivs)  
int nvar,nstep;  
float vstart[],x1,x2;  
void (*derivs)();  
{  
  int i,k;  
  float x,h;  
  float *v,*vout,*dv,*vector(),**matrix();  
  void rk4(),nrerror(),free_vector();  
  float kemat[7];  

  v = vector(1,nvar);  
  vout = vector(1,nvar);  
  dv = vector(1,nvar);  
  xx = vector(1,nvar);  
  y = matrix(1,nvar,1,nvar);  

  for(i=0;i <= 7;i++)  
    kemat[i] = 0.0;  
  for(i=1;i <= nvar;i++)  
    {  
    v[i] = vstart[i];  
    y[i][1] = v[i];  
    }  
  xx[1] = x1;  
  x = x1;  
  h = (x2-x1)/nstep;
for (k=1;k <= nstep;k++) {
    (*derivs)(x,v,dv);
    rk4(v,dv,nvar,x,h,vout,derivs);
    if (((float)(x+h) == x) nrerror("Step size too small in routine RKDUMB");
    x += h;
    xx[k+1]=x;
    for (i=1;i <= nvar;i++) {
        v[i]=vout[i];
        y[i][k+1]=v[i];
    }
}
free_vector(dv,1,nvar);
free_vector(vout,1,nvar);
free_vector(v,1,nvar);
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i;
    float **m;
    m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;
    for(i=nrl;i <= nrh;i++) {
        m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}

/* This evaluates \[ \dot{\theta} = A^{-1}(r - B(\dot{\theta}) - C(\dot{\theta})^2 - g(\theta)) \], which is integrated by RK-4 method to yield joint velocities i.e., \( \dot{\theta} \) and joint positions i.e., \( \theta \) */

void derivative(x,y,dydx)
float x,y[],dydx[];
{
    void apmat();
    void bpmat();
    void CPmat();
    void gpmat();
    int i,j;
    for(i=1;i <= 2*N-1;i++) {j=i+2;j=j+2)
        dydx[i]=y[j];
APPENDIX D

for(i=1;j=1;i<2*N-1;j=1+i+j=1+2)
dydx[i]=-(b[i][i]y[i][2]*y[i][4]+b[i][i]y[i][2]*y[i][6]+b[i][i]y[i][3]*y[i][2]*y[i][8]+b[i][i]y[i][4]*y[i][2]*y[i][10]+b[i][i]y[i][5]*y[i][2]*y[i][12]+b[i][i]y[i][6]*y[i][2]*y[i][6]+b[i][i]y[i][7]*y[i][4]*y[i][8]+b[i][i]y[i][8]*y[i][4]*y[i][10]+b[i][i]y[i][9]*y[i][4]*y[i][12]+b[i][i]y[i][10]*y[i][6]*y[i][8]+b[i][i]y[i][11]*y[i][6]*y[i][10]+b[i][i]y[i][12]*y[i][6]*y[i][12]+b[i][i]y[i][13]*y[i][8]*y[i][10]+b[i][i]y[i][14]*y[i][8]*y[i][12]+b[i][i]y[i][15]*y[i][10]*y[i][12]+CP[i][i]*y[i][2]*y[i][2]+CP[i][i]*y[i][4]*y[i][4]+CP[i][i]*y[i][6]*y[i][6]+CP[i][i]*y[i][8]*y[i][8]+CP[i][i]*y[i][10]*y[i][10]+CP[i][i]*y[i][12]*y[i][12]+gp[i]);
//confusion also wrong m2 y1sq. y2sq. for CR dydx[j]=-b[i][i]y[i][1]*y[i][3]+b[i][i]y[i][2]*y[i][5]+b[i][i]y[i][3]*y[i][1]*y[i][7]+b[i][i]y[i][4]*y[i][1]*y[i][9]+b[i][i]y[i][5]*y[i][1]*y[i][11]+b[i][i]y[i][6]*y[i][1]*y[i][5]+b[i][i]y[i][7]*y[i][3]*y[i][7]+b[i][i]y[i][8]*y[i][3]*y[i][9]+b[i][i]y[i][9]*y[i][3]*y[i][11]+b[i][i]y[i][10]*y[i][3]*y[i][11]+b[i][i]y[i][11]*y[i][3]*y[i][9]+b[i][i]y[i][12]*y[i][3]*y[i][9]+b[i][i]y[i][13]*y[i][3]*y[i][11]+b[i][i]y[i][14]*y[i][3]*y[i][11]+b[i][i]y[i][15]*y[i][3]*y[i][11]+CP[i][i]*y[i][1]*y[i][1]+CP[i][i]*y[i][2]*y[i][2]+CP[i][i]*y[i][3]*y[i][3]+CP[i][i]*y[i][4]*y[i][4]+CP[i][i]*y[i][5]*y[i][5]+CP[i][i]*y[i][6]*y[i][6]+gp[i]);
dydx[10]):-b[i][i]y[i][1]*y[i][3]+b[i][i]y[i][2]*y[i][5]+b[i][i]y[i][3]*y[i][1]*y[i][7]+b[i][i]y[i][4]*y[i][1]*y[i][9]+b[i][i]y[i][5]*y[i][1]*y[i][11]+b[i][i]y[i][6]*y[i][1]*y[i][5]+b[i][i]y[i][7]*y[i][3]*y[i][7]+b[i][i]y[i][8]*y[i][3]*y[i][9]+b[i][i]y[i][9]*y[i][3]*y[i][11]+b[i][i]y[i][10]*y[i][3]*y[i][11]+b[i][i]y[i][11]*y[i][3]*y[i][9]+b[i][i]y[i][12]*y[i][3]*y[i][9]+b[i][i]y[i][13]*y[i][3]*y[i][11]+b[i][i]y[i][14]*y[i][3]*y[i][11]+b[i][i]y[i][15]*y[i][3]*y[i][11]+CP[i][i]*y[i][1]*y[i][1]+CP[i][i]*y[i][2]*y[i][2]+CP[i][i]*y[i][3]*y[i][3]+CP[i][i]*y[i][4]*y[i][4]+CP[i][i]*y[i][5]*y[i][5]+CP[i][i]*y[i][6]*y[i][6]+gp[i]);
// printf("dydx-first %f %f %f %f %f\n",dydx[2],dydx[4],dydx[6],dydx[8],dydx[10],dydx[12]);


// printf("dydx-second \%f \%f \%f \%f \%f \%fn",dydx[2],dydx[4],dydx[6],dydx[8],dydx[10],dydx[12]);
}

void ludcmp(a,n,indx,d)
int n,*indx;
float **a,*d;
{
int i,imax,j,k;
float big,dum,sum,temp;
float *vv,*vector();
void nrerror(),free_vector();

vv=vector(1,n);
*d=1.0;

for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
for(k=1;k<=i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k++)
for(k=1;k<i;k+)
APPENDIX D

```c
APPENDIX D

    a[j][k] = dum;

}  
*d = -(*d);  
vv[imax] = vv[j];  
}  
indx[j] = imax;  
if (a[j][j] == 0.0) a[j][j] = TINY;  
if (j != n) {
    dum = 1.0 / (a[i][j]);  
    for (i = j + 1; i < n; i++) a[i][j] *= dum;  
}  
}  
free_vector(vv, 1, n);  
}

void lubksb(a, n, indx, b)  
float **a, b[];  
int n, *indx;  
{
    int i, ii = 0, ip, j;  
    float sum;  
    for (i = 1; i < n; i++) {  
        ip = indx[i];  
        sum = b[ip];  
        b[ip] = b[i];  
        if (ii)  
            for (j = ii; j < = i - 1; j++) sum -= a[i][j] * b[j];  
        else if (sum) ii = i;  
        b[i] = sum;  
    }  
    for (i = n; i > = 1; i--) {  
        sum = b[i];  
        for (j = i + 1; j < n; j++) sum -= a[i][j] * b[j];  
        b[i] = sum / a[i][i];  
    }
}
```
D-6 Computer Program to get Joint Motor Torque(s) for Controlling the Robot based on the Real Model

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */

#define cc2 (cos(theta[2])*cos(theta[2]))
#define ss23 (sin(theta[2])*sin(theta[3]))
#define c2  (cos(theta[2]))
#define s2  (sin(theta[2]))
#define c23 (cos((theta[2])+theta[3]))
#define s3  (sin(theta[3]))
#define c3  (cos(theta[3]))
#define s4  (sin(theta[4]))
#define s5  (sin(theta[5]))
#define c4  (cos(theta[4]))
#define c5  (cos(theta[5]))
#define sc2 (sin(theta[2])*cos(theta[2]) )
#define sc23 (sin(theta[2])*cos(theta[3]) )
#define ss4 (sin(theta[4])*sin(theta[4]))
#define ss5 (sin(theta[5])*sin(theta[5]))
#define cc4 (cos(theta[4])*cos(theta[4]))
#define cc5 (cos(theta[5])*cos(theta[5]))
#define cc23 (cos(theta[2])*cos(theta[3]))
#define cc4 (cos(theta[4])*cos(theta[4]))
#define cc5 (cos(theta[5])*cos(theta[5]))
#define ss2 (sin(theta[2])*sin(theta[2]) )

/* Defining Expressions for the Inertial Constants Appearing in the Equations of Forces of Motion */

#define i1  (1.43)
#define i2  (1.75)
#define i3  (1.38)
#define i4  (0.690)
#define i5  (0.372)
#define i6  (0.333)
#define i7  (0.298)
#define i8  (-0.134)
#define i9  (0.0238)
#define i10 (-0.0213)

#define i11 (-0.0142)
#define i12 (-0.0110)
#define i13 (-3.79e-03)
#define i14  (1.64e-03)
#define i15  (1.25e-03)
```
APPENDIX D

#define i16 (1.24e-03)
#define i17 (6.42e-04)
#define i18 (4.31e-04)
#define i19 (3.00e-04)
#define i20 (2.02e-04)
#define i21 (-1.00e-04)
#define i22 (-5.80e-05)
#define i23 (4.00e-05)
#define i1 (1.14)
#define i2 (4.71)
#define i3 (0.827)
#define i4 (0.200)
#define i5 (0.179)
#define i6 (0.193)

/* Defining Expressions for the Gravitational Constants Appearing in the Equations of Forces of Motion */

#define g1 (-37.2)
#define g2 (-8.44)
#define g3 (1.02)
#define g4 (0.249)
#define g5 (-0.0282)

/* Defining fixed link parameters as per Denavit-Hartenberg notation */

#define a2 (0.4318)
#define a3 (-0.0203)
#define a4 (0.4331)

#define pi (3.1415927)

int i,j;

float theta[7]; /* (6x1) Vector of Joint Angles \( \theta_1, \theta_2, \ldots, \theta_6 \) */

float ar[7][7],br[7][16],cr[7][7],gr[7]; /* A(6x6), B(6x15), and C(6x6) matrices and the g vector */

float coeff[4]; /* Vector of Cubic Coefficients */
float th0,th1,thd0,thdf; /* \( \theta_0, \dot{\theta}_0, \dot{\theta}_0, \ddot{\theta}_0 \) */
float thdot[7],thd1dlt[7]; /* (6x1) Vector \( \dot{\theta} \) and (6x1) Vector \( \dot{\theta} \) */
float t; /* Time variable */
float qq[16],qsquare[7]; /* (1x15) Vector of velocity products i.e., \( \dot{\theta} \dot{\theta} \), and (1x6) Vector of squared velocities i.e., \( \dot{\theta}^2 \) */
float term1[7],term2[7],term3[7];
float torque[7]; /* (6x1) Torque Vector */

main()
{
    void armat(); /* Returns the expressions giving the elements of the Kinetic Energy matrix */
void brmat(); /* Returns the expressions giving the elements of the Coriolis matrix */
void CRmat(); /* Returns the expressions giving the elements of the Centrifugal matrix */
void grmat(); /* Returns the Gravity vector */

float **matrix();
float *vector();

void cubic_coeff(); /* Returns the Vector of cubic coefficients */

int k;

theta[0]=0.0;

for(i=0;i<=7;i++)
    {theta[i]=0.0*pi/180.0;
thdot[i]=0.0;
thdbld[i]=0.0;
torque[i]=0.0;
qsquare[i]=0.0;
term1[i]=0.0;
term2[i]=0.0;
term3[i]=0.0;
}

th0=15.0*pi/180.0;
thf=75.0*pi/180.0;
thd0=0.0;
thdf=0.0;
tf=3.0;
t=3.0;

cubic_coeff(coeff);

for(i=1;i<=6;i++)
    {theta[i]=coeff[0]+coeff[1]*t+coeff[2]*t*t+coeff[3]*t*t*t;
thdot[i]=coeff[1]+2.0*coeff[2]*t+3.0*coeff[3]*t*t;
thdbld[i]=2.0*coeff[2]+6.0*coeff[3]*t;
}

armat(ar);
brmat(br);
CRmat(CR);
grmat(gr);

/*for(i=1;i<=6;i++)
    {thdot[i]=(float)(i)*t;/
for(i=1;i<=6;i++)
    printf("thdot %f\n",thdot[i]);
}
for(i=1,j=2;i < =5,j < =6;i++ j++)
qq[i]=thdot[1]*thdot[j];
for(i=6,j=3;i < =9 j < =6;i++ j++)
qq[i]=thdot[2]*thdot[j];
for(i=10,j=4;i < =12 j < =6;i++ j++)
qq[i]=thdot[3]*thdot[j];
for(i=13,j=5;i < =14 j < =6;i++ j++)
qq[i]=thdot[4]*thdot[j];
qq[15]=thdot[5]*thdot[6];

for(i=1;i < =15;i++)
printf("qq %d %fn",i,qq[i]);
printf("thdbldt*** %f %f %f %f %f %fn",thdbldt[1],thdbldt[2],thdbldt[3],thdbldt[4],thdbldt[5],thdbldt[6]);

for(i=1;i < =6;i++)
qsquare[i]=thdot[i]*thdot[i];
for(i=1;i < =6;i++)
printf("qsquare %f",qsquare[i]);

//* Evaluating individual terms of the state space equation i.e., A(\theta)(\dot{\theta}) + B(\theta)(\ddot{\theta}) + C(\theta)(\dot{\theta}^2) + g(\theta) = \tau */

for(i=1;i < =6;i++)
for(j=1;j < =6;i++ j++)
term1[i]=term1[i] + arf[i][j]*thdbldt[j];

for(i=1;i < =6;i++)
for(j=1;j < =15 j++)
term2[i]=term2[i] + br[i][j]*qq[j];

for(i=1;i < =6;i++)
for(j=1;j < =6 j++)
term3[i]=term3[i] + CR[i][j]*qsquare[j];

for(i=1;i < =6;i++)
printf("thdbldt,terms %f %f %f %f %f %fn",thdbldt[i],term1[i],term2[i],term3[i]);

for(i=1;i < =6;i++)
torque[i] = term1[i] + term2[i] + term3[i] + grf[i];

for(i=1;i < =6;i++)
printf("torque %d %fn",i,torque[i]);

return;
}

void armat(ar)
APPENDIX D

void brmat(br)
{

for(i=0;i<=7;i++)
for(j=0;j<=7;j++)
ar[i][j]=0.0;

    float ar[7][7];
    {
        int i,j;

        for(i=0;i<=7;i++)
for(j=0;j<=7;j++)
ar[i][j]=0.0;

    a r [ 1 ] [ 1 ] = ( i m 1 + i 1 + i 3 * c c 2 + i 7 * s s 2 3
+ i10*sc23 + i11*sc2 + i20*(ss5*ss23*(1+cc4))-1.2*sc23*cc4+2.0*(i5*c2*s23+i12*c2*c23+i15*(ss23
+ c5+sc23*cc4*ss)+i16*cc23*ss5+i22*(ss23*cc5+cc23*cc4*ss5));
    ar[1][2] = ( i4*s2 + i8*c23 + i9*s2 + i13*sc23 - i15*c23*ss4+ss5 + i16*s2*ss4+ss5 + i18*(ss23*ss4+ss5+c23*ss5) +
i19*sc23*ss4 + i20*ss4*(ss23*ss4+cc5+c23*ss5) + i22*ss23*ss4+ss5 );
    ar[1][3] = ( i8*c23 + i13*ss23 - i15*ss23*ss4+ss5 + i19*ss23*ss4 + i18*(ss23*ss4+ss5-c23*ss5) +
i20*ss4*(ss23*ss4+cc5+c23*ss5) );
    ar[1][4] = ( i14*c23 + i15*ss23*ss4+ss5 + i16*ss2*ss4+ss5 + i18*(ss23*ss4+ss5-c23*ss5) +
i20*ss4*(ss23*ss4+cc5+c23*ss5) );
    ar[1][5] = ( i15*ss23*ss4+ss5 + i16*ss2*ss4+ss5 + i17*ss23*ss4 + i18*(ss23*ss5-c23*ss4+ss5) +
i22*ss23*ss4+ss5 );
    ar[1][6] = ( i23*(ss23*ss5-c23*ss4+ss5) );

    a r [ 2 ] [ 2 ] = ( i m 2 + i 2 + i 6 + i 2 0 * s s 4 * s s 5 + i 2 1 * s s 4
+2.0*(i5*ss3+i12*ss5+i15*ss5-i16*(ss3*ss5+i22*ss4+ss5) ));
    ar[2][3] = ( i5*ss3 + i6 + i12*ss3 + i16*(ss3*ss5+i22*ss4+ss5) + i20*ss4+ss5 + i21*ss4 + 2.0*(i15*ss5+i22*ss4+ss5) );
    ar[2][4] = ( i14*ss4+ss5 + i16*ss4+ss5 + i20*ss4*ss5 );
    ar[2][5] = ( i15*ss4+ss5 + i16*(ss3*ss5 + s3*ss4+ss5) + i17*ss4 + i22*ss5 );
    ar[2][6] = ( i23*ss4+ss5 );

    ar[3][3] = ( im3 + i6 + i20*ss4+ss5 + i21*ss4+2*(i15*ss5 + i22*ss4+ss5) );
    ar[3][4] = ( i15*ss4+ss5 + i20*ss4*ss5 );
    ar[3][5] = ( i15*ss4+ss5 + i17*ss4 + i22*ss5 );
    ar[3][6] = ( i23*ss4+ss5 );

    ar[4][4] = ( im4+i14-i20*ss5);
    ar[4][5] = ( 0.0);
    ar[4][6] = ( i23*ss5);

    ar[5][5] = ( im5+i17);
    ar[5][6] = ( 0.0);
    ar[6][6] = ( im6+i23);

    for(i=0;i<=7;i++)
for(j=0;j<=7;j++)
ar[i][j]=ar[i][j];

    for(i=1;i<=7;i++)
printf("ar : armat %f %f %f %f %f %f n",ar[i][1],ar[i][2],ar[i][3],ar[i][4],ar[i][5],ar[i][6]);

    return;
}

void brmat(br)
APPENDIX D

```c
float br[7][16];
{
  int i, j;

  for(i=0; i<7; i++)
    for(j=0; j<16; j++)
      br[i][j]=0.0;

  br[1][1] = (2.0*(-i3*s23 + i5*c23 + i7*s23 - i12*s223 + i15*(2.0*s23*c5+(1-2.0*s23)*c4*s5) + i16*(c223*c5 - s223*c4*s5) + i21*s23*c4 + i20*((1+c4)*s23*s5-(1-2.0*s23)*c4*s5) + i22*((1-2.0*s23)*c5 - 2.0*s23*c4*s5) + i10*(1-2.0*s23) + i11*(1-2.0*s23));
  br[1][2] = (2.0*(i5*c23 + i7*s23 - i12*s23 + i15*(2.0*s23*c5+(1-2.0*s23)*c4*s5) + i16*c23*c23 + i21*s23*c4 + i20*((1+c4)*s23*s5-(1-2.0*s23)*c4*s5) + i22*((1-2.0*s23)*c5 - 2.0*s23*c4*s5) + i10*(1-2.0*s23));
  br[1][3] = (2.0*(-i15*s23*c4*s5 - i16*c23*c4*s5 + i18*c4*s5 - i20*(s23*s5*c4 - s23*c4*s5) - i22*s23*c4*s5 - i21*s23*c4*s5));
  br[1][4] = (2.0*(i10*(s23*c4*(1 - c23) - c23) - c23*c4*(1 - 2.0*s5)) - i15*(s23*s5 - s23*c4*s5) - i16*(c23*s4 - c23*c4*c5 + i18*c4*s5 + i22*(c23*c4*c5 - s23*s5)));
  br[1][5] = (0.0);
  br[1][6] = (2.0*(-i8*s23 + i13*s23 + i15*s23*c4*s5 + i18+c23*c4*s5 + s23*c5) + i19*c23*s4 + i20*s4*(c23*c4*c5 - s23*s5) + i22*s23*c4*s5));
  br[1][7] = (-i18*s23*c4*s5 + i19*s23*(1 - (2*s4)) + i20*s23*(1-2.0*s4*c5) - i14*s23);
  br[1][8] = (i17*c23*s4 + i18*2.0*(s23*c4*c5 + c23*s5) + i20*s4*(c23*(1 - 2.0*s5) - s23*c4*2.0*c5));
  br[1][9] = (-i23*(s23*c5 + c23*c4*s5));
  br[1][10] = (br[1][7]);
  br[1][11] = (br[1][8]);
  br[1][12] = (br[1][9]);
  br[1][13] = (2.0*(i15*s23*c4*c5 + i16*c23*c4*c5 + i18+c23*s4*c5 + i22*c23*c4*c5) + i17*s23*c4 -i20*(s23*c4*(1 - 2.0*s5) + 2.0*c23*s5));
  br[1][14] = (i23*s23*c4*s5);
  br[1][15] = (-i23*(c23*s5 + s23*c4*c5));

  br[2][1] = (0.0);
  br[2][2] = (0.0);
  br[2][3] = (i4*s23 + i9*s23*(1.0 - (2.0*s4)) + 2.0*(-i15*c23*c4*s5 + i16*s23*c4*s5 + i20*(s23*c4*c5 - c4*s5) + i22*s23*c4*s5));
  br[2][4] = (2.0*(-i15*c23*s4*c5 + i22*s23*c4*c5 + i16*s23*c4*c5 - i17*s23*c4 + i20*(c23*s4*(1.0 - 2.0*s5) - 2.0*s23*c4*c5));
  br[2][5] = (-br[1][9]);
  br[2][6] = (2.0*(-i12*s3 + i5*c3 + i16*(c3*c5 - s3*c4*s5)));
  br[2][7] = (2.0*(-i16*c3*s4*s5 + i20*s4*c5 + i21*s4 - i22*s4*c5));
  br[2][8] = (2.0*(-i15*s5 + i16*(c3*c4*c5 - s3*c5) + i20*s4*s5 + i22*c4*c5));
  br[2][9] = (0.0);
  br[2][10] = (br[2][7]);
  br[2][11] = (br[2][8]);
  br[2][12] = (0.0);
  br[2][13] = (2.0*(-i15*s4*c5 - i16*s3*s4*c5) - i17*s4 + i20*s4*(1 - 2.0*s5));
  br[2][14] = (i23*c4*s5);
  br[2][15] = (i23*c4*s5);

  br[3][1] = (0.0);
  br[3][2] = (0.0);
  br[3][3] = (2.0*(-i15*c23*c4*s5 + i22*s23*c4*s5 + i20*(s23*c4*c5 - c4*s5) + c23*c4*s5)) + i14*s23 +
```
APPENDIX D

\begin{verbatim}
i9*s23*(1.0 - (2.0*s4)));
br[3][4] = (2.0*(i15*s23*s4*c5 + i22*s23*s4*c5) - i17*s23*s4 + i20*s4*(c23*(1.0 - 2.0*s5) - 2.0*s23*c4*s5)));
br[3][5] = (-br[1][12]);
br[3][6] = (0.0);
br[3][7] = (2.0*(i20*s4*c5 + i21*s4 - i22*s4*s5));
br[3][8] = (2.0*(i15*s5 + i20*s4*c5 + i22*s4*c5));
br[3][9] = (0.0);
br[3][10] = (br[3][7]);
br[3][11] = (br[3][8]);
br[3][12] = (0.0);
br[3][13] = (-i15*2.0*s4*c5 - i17*s4 + i20*s4*(1.0 - 2.0*s5));
br[3][14] = (br[2][14]);
br[3][15] = (br[2][15]);
br[4][1] = (-br[2][3]);
br[4][2] = (-br[3][3]);
br[4][3] = (0.0);
br[4][4] = (-i20*s23*s4*(1.0 - 2.0*s5) + 2.0*c23*s5) - i17*s23*c4);
br[4][5] = (-br[1][14]);
br[4][6] = (-br[3][7]);
br[4][7] = (0.0);
br[4][8] = (i17*s4 + i20*s4*(1.0 - 2.0*s5));
br[4][9] = (-br[2][14]);
br[4][10] = (0.0);
br[4][11] = (br[4][8]);
br[4][12] = (-br[3][14]);
br[4][13] = (-i20*2.0*s5);
br[4][14] = (0.0);
br[4][15] = (-i23*s5);
br[5][1] = (-br[2][4]);
br[5][2] = (-br[3][4]);
br[5][3] = (-br[4][4]);
br[5][4] = (0.0);
br[5][5] = (-br[1][15]);
br[5][6] = (-br[3][8]);
br[5][7] = (-br[4][8]);
br[5][8] = (0.0);
br[5][9] = (-br[2][15]);
br[5][10] = (br[5][7]);
br[5][11] = (0.0);
br[5][12] = (-br[3][15]);
br[5][13] = (0.0);
br[5][14] = (-br[4][15]);
br[5][15] = (0.0);
br[6][1] = (br[1][9]);
br[6][2] = (br[1][12]);
br[6][3] = (br[1][14]);
br[6][4] = (br[1][15]);
br[6][5] = (0.0);
br[6][6] = (0.0);
\end{verbatim}
APPENDIX D

for (i=1; i<=6; i++)
{printf("%f %f %f %f %f %f %f 
",br[i][l],br[i][2],br[i][3],br[i][4],br[i][5],br[i][6],br[i][7],br[i][8]);
 printf("%f %f %f %f %f %f %f 
",br[i][9],br[i][10],br[i][11],br[i][12],br[i][13],br[i][14],br[i][15]);}
//printf("from br br11 = %f",br[1][1]);
return;
}

void CRmat(CR)
float CR[7][7];
{
  int i,j;
  extern float br[7][16];

  for(i=0;i<=7;i++)
    for(j=0;j<=7;j++)
      CR[i][j]=0.0;
    //printf("from inside CR br11 = %f",br[1][1]);

  CR[1][1]=(0.0);
  CR[1][2]=(i4*c2 - i8*s23 - i9*s2 + i13*c23 + i15*s23*s4*s5 + i16*c2*s4*s5 + i18*(c23*c4*s5 + s23*c5) +
            i19*s23*s4 + i20*s4*(c23*c4*s5 - s23*c5) + i22*c23*s4*s5);
  CR[1][3]=(0.5*br[1][6]);
  CR[1][4]=(-i15*c23*s4*s5 - i16*c2*s4*s5 + i18*c23*c4*s5 - i22*c23*s4*s5);
  CR[1][5]=(-i15*c23*s4*s5 - i16*c2*s4*s5 + i18*(s23*c5 + c23*c4*s5) - i22*c23*s4*s5);
  CR[1][6]=(0.0);

  CR[2][1]=(-0.5*br[1][1]);
  CR[2][2]=(0.0);
  CR[2][3]=(0.5*br[2][6]);
  CR[2][4]=(-i15*c4*s5 - i16*s3*c4*s5 + i20*c4*s5);
  CR[2][5]=(-i15*c4*s5 + i16*(c3*c5 - s3*c4*s5) + i22*c5);
  CR[2][6]=(0.0);

  CR[3][1]=(-0.5*br[1][2]);
  CR[3][2]=(-CR[2][3]);
  CR[3][3]=(0.0);
  CR[3][4]=(-i15*c4*s5 + i20*c4*s5);
  CR[3][5]=(-i15*c4*s5 + i22*c5);
  CR[3][6]=(0.0);

  CR[4][1]=(-0.5*br[1][3]);
  CR[4][2]=(-0.5*br[2][7]);
APPENDIX D

CR[4][3] = (0.5*br[4][6]);
CR[4][4] = (0.0);
CR[4][5] = (0.0);
CR[4][6] = (0.0);

CR[5][1] = (-0.5*br[1][4]);
CR[5][2] = (-0.5*br[2][8]);
CR[5][3] = (0.5*br[3][6]);
CR[5][4] = (-0.5*br[4][13]);
CR[5][5] = (0.0);
CR[5][6] = (0.0);

CR[6][1] = (0.0);
CR[6][2] = (0.0);
CR[6][3] = (0.0);
CR[6][4] = (0.0);
CR[6][5] = (0.0);
CR[6][6] = (0.0);

for (i = 1; i < 7; i++)
    printf("CR %f %f %f %f %f
", CR[i][1], CR[i][2], CR[i][3], CR[i][4], CR[i][5], CR[i][6]);
for (i = 1; i <= 6; i++)
    return;

int i;

void grmat(gr)

float gr[7];
{
    for (i = 1; i <= 6; i++)
        printf("gr %fn", gr[i]);
    return;
}

void cubic_coeff(coeff)

float coeff[4];
{
    for (i = 1; i <= 6; i++)
        printf("gr %fn", gr[i]);
    return;
}

    for (i = 1; i < 7; i++)
        printf("CR %f %f %f %f %f
", CR[i][1], CR[i][2], CR[i][3], CR[i][4], CR[i][5], CR[i][6]);
for (i = 1; i <= 6; i++)
    return;

int i;

void grmat(gr)

float gr[7];
{
    for (i = 1; i <= 6; i++)
        printf("gr %fn", gr[i]);
    return;
}

void cubic_coeff(coeff)

float coeff[4];
{
    for (i = 1; i <= 6; i++)
        printf("gr %fn", gr[i]);
    return;
}
for(i=0;i<=3;i++)
  printf("coeff %fn",coeff[i]);

  return;
}

float *vector(nl,nh)
int nl,nh;
{ void nrerror();
  float *v;
  v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
  if (!v) nrerror ("allocation failure in vector()");
  return v-nl;
}

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
  free((char*) (v+nl));
}

void nrerror(error_text)
char error_text[];
{
  void exit();
  fprintf(stderr,"Numerical Recipies run-time error ...
");
  fprintf(stderr,"%s",error_text);
  fprintf(stderr,"...now exiting to system...
");
  exit(1);
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
  int i;
  float **m;
  m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
  if (!m) nrerror("allocation failure 1 in matrix()");
  m -= nrl;
  for(i=nrl;i<=nrh;i++) {
    m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
    if (!m[i]) nrerror("allocation failure 2 in matrix()");
    m[i] -= ncl;
  }
  return m;
}
D-7 Computer Program to get Joint Motor Torque(s) for Controlling the Robot based on the Approximate Model

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */

#define cc2 (cos(theta[2])*cos(theta[2]))
#define ss23 (sin(theta[2])*sin(theta[3]))
#define cc2 (cos(theta[2]))
#define ss23 (sin(theta[2])*sin(theta[3]))
#define cc2 (cos(theta[2]))
#endif
#define ss23 (sin(theta[2])*sin(theta[3]))
#define cc2 (cos((theta[2]+theta[3])))
#define ss23 (sin((theta[2]+theta[3])))
#define cc2 (cos((theta[2]+theta[3])))
#define ss23 (sin((theta[2]+theta[3])))
#endif
#define ss4 (sin(theta[4]))
#define cc4 (cos(theta[4]))
#define cc5 (cos(theta[5]))
#define ss5 (sin(theta[5]))
#define cc6 (cos(theta[6]))
#define ss6 (sin(theta[6]))

/* Defining fixed link parameters as per Denavit-Hartenberg notation */

#define a2 (0.4318)
#define a3 (-0.0203)
#define d4 (0.4331)
#define pi (3.1415927)

int i,j;

float theta[7]; /* (6x1) Vector of Joint Angles \theta_j, \theta_p, ..., \theta_s */

float ap[7][7][16],cp[7][7],gp[7]; /* A(6x6), B(6x15), and C(6x6) matrices and the g vector */

float coeff[4]; /* Vector of Cubic Coefficients */
float th0,th,th0d,thdf,tf; /* \theta_0, \theta, \dot{\theta}_0, \dot{\theta}_p, t_f */
float thdot[7],shdhd[7]; /* (6x1) Vector \dot{\theta} and (6x1) Vector \ddot{\theta} */
float t; /* Time variable */
float qq[16],qsquare[7]; /* (1x15) Vector of velocity products i.e., \theta \dot{\theta}, and (1x6) Vector of squared velocities i.e., \theta^2 */
float term1[7],term2[7],term3[7];
float torque[7]; /* (6x1) Torque Vector */

main()
```
APPENDIX D

```c
{
    void apmat(); /* Returns the expressions giving the elements of the Kinetic Energy matrix */
    void bpmat(); /* Returns the expressions giving the elements of the Coriolis matrix */
    void CPmat(); /* Returns the expressions giving the elements of the Centrifugal matrix */
    void gpmat(); /* Returns the Gravity vector */

    float **matrix();
    float *vector();

    void cubic_coeff(); /* Returns the Vector of cubic coefficients */
    int k;

    theta[0] = 0.0;

    for(i=0;i<=7;i++)
    {
        theta[i] = 0.0*pi/180.0;
        thdot[i] = 0.0;
        thdddot[i] = 0.0;
        torque[i] = 0.0;
        qsquare[i] = 0.0;
        term1[i] = 0.0;
        term2[i] = 0.0;
        term3[i] = 0.0;
    }

    th0 = 15.0*pi/180.0;
    thf = 75.0*pi/180.0;
    thd0 = 0.0;
    thdf = 0.0;
    tf = 3.0;
    t = 3.0;

    cubic_coeff(coeff);
    for(i=0;i<=3;i++)
    {
        printf("coeff_f\n",coeff[i]);
    }

    for(i=1;i<=6;i++)
    {
        theta[i] = coeff[0] + coeff[1]*t + coeff[2]*t*t + coeff[3]*t*t*t;
        thdot[i] = coeff[1] + 2.0*coeff[2]*t + 3.0*coeff[3]*t*t;
        thddot[i] = 2.0*coeff[2] + 6.0*coeff[3]*t;
    }

    apmat(ap);
    bpmat(bp);
    CPmat(CP);
    gpmat(gp);

    /*for(i=1;i<=6;i++)
    {
        thdot[i] = (float)(i);*/
    for(i=1;i<=6;i++)
    {
        printf("thdot_f\n",thdot[i]);
    }

    for(i=1;i=2;i<=5;i++)
    {
        printf("thddot_f\n",thddot[i]);
    }
}
```
APPENDIX D

qq[i]=thdot[1]*thdot[j];
for(i=6;i<9; i++)
qq[i]=thdot[2]*thdot[j];
for(i=10;i<12; i++)
qq[i]=thdot[3]*thdot[j];
for(i=13;i<14; i++)
qq[i]=thdot[4]*thdot[j];
qq[15]=thdot[5]*thdot[6];

for(i=1;i<16; i++)
printf("qq %d %f %n",i,qq[i]);

/* Evaluating individual terms of the state equation i.e.,
A(θj)θj + B(θj)θjθj + C(θj)θjθj + g(θ) = τ */

for(i=1;i<6; i++)
qsquare[i]=thdot[i]*thdot[i];
for(i=1;i<6; i++)
printf("qsquare %f%n",qsquare[i]);
for(i=1;i<6; i++)
for(j=1;j<6; j++)
term1[i]=term1[i] + ap[i][j]*thdbldt[j];
for(i=1;i<6; i++)
for(j=1;j<15; j++)
term2[i]=term2[i] + bp[i][j]*qq[j];
for(i=1;i<6; i++)
for(j=1;j<6; j++)
term3[i]=term3[i] + CP[i][j]*qsquare[j];
for(i=1;i<6; i++)
torque[i] = term1[i] + term2[i] + term3[i] + gp[i];
for(i=1;i<6; i++)
printf("thdbldt,terms %f %f %f %fn",thdbldt[i],term1[i],term2[i],term3[i]);
for(i=1;i<6; i++)
printf("torque %d %f%n",i,torque[i]);
return;

void apmat(ap)
float ap[7][7];
{
int i,j;
for(i=0;i<7;i++)
for(j=0;j<7; j++)
ap[i][j]=0.0;
```
APPENDIX D

void bpmat(float bp[7][16])
{

    int i,j;

    for(i=0;i<=7;i++)
        for(j=0;j<=7;j++)
            bp[i][j]=ap[i][j];

    for(i=1;i<=7;i++)
        printf("ap %f %f %f %f %f
",ap[i][1],ap[i][2],ap[i][3],ap[i][4],ap[i][5],ap[i][6]);

    return;
}

float bp[7][16];

ap[1][1] = (2.57+1.38*(c2)+0.3*(s23)+7.44*0.1*(c2)*(s23));
ap[1][2] = (0.69*(s2)-0.134*(c2)+1.253*(c2));
ap[1][3] = (-0.134*c23 + -3.97e-03*s23);
ap[1][4] = (0.0);
ap[1][5] = (0.0);
ap[1][6] = (0.0);

bp[1][1] = (2.57+1.38*(c2)+0.3*(s23)+7.44*0.1*(c2)*(s23));
bp[1][2] = (0.69*(s2)-0.134*(c2)+1.253*(c2));
bp[1][3] = (-0.134*c23 + -3.97e-03*s23);
bp[1][4] = (0.0);
bp[1][5] = (0.0);
bp[1][6] = (0.0);

bp[2][1] = (6.79 + 0.744*s3);
bp[2][2] = (0.333 + 0.372*s3 -0.0110*c3);
bp[2][3] = (0.0);
bp[2][4] = (0.0);
bp[2][5] = (0.0);
bp[2][6] = (0.0);

bp[3][1] = (1.16);
bp[3][2] = (-1.25e-03*s4*s5);
bp[3][3] = (1.25e-03*c4*c5);
bp[3][4] = (0.0);
bp[3][5] = (0.0);
bp[3][6] = (0.0);

bp[4][1] = (0.20);
bp[4][2] = (0.0);
bp[4][3] = (0.0);
bp[4][4] = (0.0);
bp[4][5] = (0.0);
bp[4][6] = (0.0);

bp[5][1] = (0.18);
bp[5][2] = (0.0);
bp[5][3] = (0.0);
bp[5][4] = (0.0);
bp[5][5] = (0.0);
bp[5][6] = (0.0);

bp[6][1] = (0.19);

for(i=0;i<=7;i++)
    for(j=0;j<=7;j++)
        bp[i][j]=ap[i][j];

for(i=1;i<=7;i++)
    printf("ap %f %f %f %f %f
",ap[i][1],ap[i][2],ap[i][3],ap[i][4],ap[i][5],ap[i][6]);

return;
}
```
bp[1][7] = (0.0);
bp[1][8] = (0.0);
bp[1][9] = (0.0);
bp[1][10] = (bp[1][7]);
bp[1][11] = (bp[1][8]);
bp[1][12] = (bp[1][9]);
bp[1][13] = (0.0);
bp[1][14] = (0.0);
bp[1][15] = (0.0);

bp[2][1] = (0.0);
bp[2][2] = (0.0);
bp[2][3] = (1.64e-03*s23 - 2.50e-03*c23*c4*s5 + 2.48e-03*s2*c4*s5 + 0.3e-03*s23*(1-(2*s4)));
bp[2][4] = (-2.50e-03*c23*s4*c5 + 2.48e-03*s2*s4*c5 -6.42e-04*c23*s4);
bp[2][5] = (-bp[1][9]);
bp[2][6] = (0.0220*s3 + 0.744*c3);
bp[2][7] = (-2.48e-03*c3*s4*s5);
bp[2][8] = (-2.50e-03*s5 +2.48e-03*(c3*c4*c5-s3*s5));
bp[2][9] = (0.0);
bp[2][10] = (bp[2][7]);
bp[2][11] = (bp[2][8]);
bp[2][12] = (0.0);
bp[2][13] = (0.0);
bp[2][14] = (0.0);
bp[2][15] = (0.0);

bp[3][1] = (0.0);
bp[3][2] = (0.0);
bp[3][3] = (-2.50e-03*c23*c4*s5 +1.64e-03*s23 +0.30e-03*s23*(1-2*s4));
bp[3][4] = (-2.50e-03*c23*s4*c5 -6.42e-04*c23*s4);
bp[3][5] = (-bp[1][12]);
bp[3][6] = (0.0);
bp[3][7] = (0.0);
bp[3][8] = (-2.50e-03*s4 );
bp[3][9] = (0.0);
bp[3][10] = (bp[3][7]);
bp[3][11] = (bp[3][8]);
bp[3][12] = (0.0);
bp[3][13] = (-2.5e-03*s4*c5);
bp[3][14] = (bp[2][14]);
bp[3][15] = (bp[2][15]);

bp[4][1] = (-bp[2][3]);
bp[4][2] = (-bp[3][3]);
bp[4][3] = (0.0);
bp[4][4] = (-6.42e-04*s23*c4);
bp[4][5] = (-bp[1][14]);
bp[4][6] = (-bp[3][7]);
bp[4][7] = (0.0);
bp[4][8] = (6.42e-04*s4);
bp[4][9] = (-bp[2][14]);
APPENDIX D

for (i=1; i<6; i++)
    {printf("br %f %f %f %f %f %f %f
",bp[i][1],bp[i][2],bp[i][3],bp[i][4],bp[i][5],bp[i][6],bp[i][7],bp[i][8]);
     printf("%f %f %f %f %f %f\n\n",bp[i][9],bp[i][10],bp[i][11],bp[i][12],bp[i][13],bp[i][14],bp[i][15]);
    //printf("from br br\n\n",bp[1][1]);
}

return;
}

void CPmat(CP)
float CP[7][7];
{
int i,j;
for (i=1; i<6; i++)
    {printf("br %f %f %f %f %f %f %f
",bp[i][1],bp[i][2],bp[i][3],bp[i][4],bp[i][5],bp[i][6],bp[i][7],bp[i][8]);
     printf("%f %f %f %f %f %f\n\n",bp[i][9],bp[i][10],bp[i][11],bp[i][12],bp[i][13],bp[i][14],bp[i][15]);
    //printf("from br br\n\n",bp[1][1]);
}

return;
}

void CPmat(CP)
float CP[7][7];
{
int i,j;
for (i=1; i<6; i++)
    {printf("br %f %f %f %f %f %f %f
",bp[i][1],bp[i][2],bp[i][3],bp[i][4],bp[i][5],bp[i][6],bp[i][7],bp[i][8]);
     printf("%f %f %f %f %f %f\n\n",bp[i][9],bp[i][10],bp[i][11],bp[i][12],bp[i][13],bp[i][14],bp[i][15]);
    //printf("from br br\n\n",bp[1][1]);
}

return;
}
extern float bp[7][16];

for(i=0;i<7;i++)
for(j=0;j<7;j++)
CP[i][j]=0.0;

//printf("from inside CR brll = %f\n",bp[1][1]);

CP[1][1]=(-0.5*b[1][1]);
CP[1][2]=(-0.5*b[1][2]);
CP[1][3]=(-1.25e-03*b[1][3]);
CP[1][4]=(-1.25e-03*b[1][4]);
CP[1][5]=(0.0);
CP[1][6]=(0.0);

CP[2][1]=(-0.5*b[2][1]);
CP[2][2]=(-0.5*b[2][2]);
CP[2][3]=(0.5*b[2][3]);
CP[2][4]=(0.5*b[2][4]);
CP[2][5]=(0.0);
CP[2][6]=(0.0);

CP[3][1]=(-0.5*b[3][1]);
CP[3][2]=(-1.25e-03*b[3][2]);
CP[3][3]=(0.0);
CP[3][4]=(0.0);
CP[3][5]=(0.0);
CP[3][6]=(0.0);

CP[4][1]=(-0.5*b[4][1]);
CP[4][2]=(-0.5*b[4][2]);
CP[4][3]=(-1.25e-03*b[4][3]);
CP[4][4]=(0.5*b[4][4]);
CP[4][5]=(0.0);
CP[4][6]=(0.0);

CP[5][1]=(-0.5*b[5][1]);
CP[5][2]=(-1.25e-03*b[5][2]);
CP[5][3]=(0.5*b[5][3]);
CP[5][4]=(-0.5*b[5][4]);
CP[5][5]=(0.0);
CP[5][6]=(0.0);

CP[6][1]=(-0.5*b[6][1]);
CP[6][2]=(-0.5*b[6][2]);
CP[6][3]=(-1.25e-03*b[6][3]);
CP[6][4]=(0.5*b[6][4]);
CP[6][5]=(0.0);
CP[6][6]=(0.0);

for(i=1;i<7;i++)
printf("CR %f %f %f %f %f %f\n",CP[i][1],CP[i][2],CP[i][3],CP[i][4],CP[i][5],CP[i][6]);
return;
void gpmat(gp)
float gp[7];
{
    int i;
    gp[1] = (0.0);
    gp[2] = (-37.2*c2 - 8.4*s23 + 1.02*s2);
    gp[3] = (-8.4*s23 + 0.25*c23);
    gp[4] = (2.8e-02*s23*s4*s5);
    gp[5] = (-2.8e-02*(c23*s5 + s23*c4*c5));
    gp[6] = (0.0);

    for (i = 1; i <= 6; i++)
        printf("gr %f\n", gp[i]);

    return;
}

void cubic_coeff(coeff)
float coeff[4];
{
    int i;

    coeff[0] = th0;
    coeff[1] = thd0;
    coeff[2] = 3.0/(tf*tf)*(thf-th0)-2.0/(tf*thd0)*1.0/(tf*thdf);
    coeff[3] = -2.0/(tf*tf)*1.0/(thdf+thd0);

    for (i = 0; i <= 3; i++)
        printf("coeff %f\n", coeff[i]);

    return;
}

float *vector(nl, nh)
int nl, nh;
{
    void nrerror();
    float *v;
    v = (float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
}

void free_vector(v, nl, nh)
float *v;
int nl, nh;
{
    free((char*) (v + nl));
}
void nrerror(error_text)
char error_text[];
{
void exit();
fprintf(stderr,"Numerical Recipes run-time error ...
");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"...now exiting to system...
");
exit(1);
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
int i;
float **m;

m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in matrix()");
m -=nrl;

for(i=nrl;i <= nrh;i + + ) {
    m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
    if (!m[i]) nrerror("allocation failure 2 in matrix()");
    m[i] -=ncl;
}
return m;
}
D-8 Computer Program to get Forces(s) on the End Effector based on the Real Model

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */

#define ill (-0.0142)
#define cc2 (cos(theta[2])*cos(theta[2]))
#define ss23 (sin(theta[2])*sin(theta[3]))
#define c2 (cos(theta[2]))
#define s23 (sin(theta[2]+theta[3]))
#define s2 (sin(theta[2]))
#define c23 (cos((theta[2])+theta[3]))
#define s3 (sin(theta[3]))
#define c3 (cos(theta[3]))
#define s4 (sin(theta[4]))
#define s5 (sin(theta[5]))
#define c4 (cos(theta[4]))
#define c5 (cos(theta[5]))
#define sc2 (sin(theta[2])*cos(theta[2]))
#define sc23 (sin(theta[2])*cos(theta[3]))
#define ss4 (sin(theta[4])*sin(theta[4]))
#define ss5 (sin(theta[5])*sin(theta[5]))
#define cc4 (cos(theta[4])*cos(theta[4]))
#define sc5 (sin(theta[5])*cos(theta[5]))
#define cc23 (cos(theta[2])*cos(theta[3]))
#define sc4 (sin(theta[4])*cos(theta[4]))
#define cc5 (cos(theta[5])*cos(theta[5]))
#define ss2 (sin(theta[2])*sin(theta[2]))

/* Defining Expressions for the Inertial Constants Appearing in the Equations of Forces of Motion */

#define i1 (1.43)
#define i2 (1.75)
#define i3 (1.38)
#define i4 (0.690)
#define i5 (0.372)
#define i6 (0.333)
#define i7 (0.298)
#define i8 (-0.134)
#define i9 (0.0238)
#define i10 (-0.0213)
#define i11 (-0.0142)
```
APPENDIX D

#define i12 (-0.0110)
#define i13 (-3.79e-03)
#define i14 (1.64e-03)
#define i15 (1.25e-03)
#define i16 (1.24e-03)
#define i17 (6.42e-04)
#define i18 (4.31e-04)
#define i19 (3.00e-04)
#define i20 (-2.02e-04)

#define i21 (-1.00e-04)
#define i22 (-5.80e-05)
#define i23 (4.00e-05)

#define im1 (1.14)
#define im2 (4.71)
#define im3 (0.827)
#define im4 (0.2000)
#define im5 (0.179)
#define im6 (0.193)

/* Defining Expressions for the Gravitational Constants Appearing in the Equations of Forces of Motion */

#define g1 (-37.2)
#define g2 (-8.44)
#define g3 (1.02)
#define g4 (0.249)
#define g5 (-0.0282)

/* Defining fixed link parameters as per Denavit-Hartenberg notation */

#define a2 (0.4318)
#define a3 (-0.0203)
#define d4 (0.4331)

#define d2 (0.2435)
#define d3 (-0.0934)

#define pi (3.1415927)

int i,j;

float theta[7]; /* (6x1) Vector of Joint Angles \theta_p, \theta_p, \ldots, \theta_p */

float ar[7][7],br[7][16],cr[7][7],gr[7]; /* A(6x6), B(6x15), and C(6x6) matrices and the g vector */

float coeff[4]; /* Vector of Cubic Coefficients */
float th0,thf,thd0,thdf,tf; /* \theta_0, \theta_f, \dot{\theta}_0, \dot{\theta}_f */
float thdot[7],thddot[7]; /* (6x1) Vector \dot{\theta} and (6x1) Vector \ddot{\theta} */
float t; /* Time Variable */
APPENDIX D

float qq[16], qsquare[7]; /* (1x15) Vector of velocity products i.e., \( \dot{\theta}^{\dot{\theta}} \), and (1x6) Vector of squared velocities i.e., \( \dot{\theta}^{2} \)*/ float term1[7], term2[7], term3[7]; float torque[7]; /* (6x1) Torque Vector */

FILE *fpwrite;
int count;

main()
{
    void armat(); /* Returns the expressions giving the elements of the Kinetic Energy matrix */
    void brmat(); /* Returns the expressions giving the elements of the Coriolis matrix */
    void Crmat(); /* Returns the expressions giving the elements of the Centrifugal matrix */
    void grmat(); /* Returns the Gravity Vector */

    float **matrix();
    float *vector();

    void cubic_coeffs(); /* Returns the Vector of cubic coefficients */
    int k;

    theta[0] = 0.0;

    for(i = 0; i <= 7; i++)
    {
        theta[i] = 0.0 * pi/180.0;
        thdof[i] = 0.0;
        thdbld[i] = 0.0;
        torque[i] = 0.0;
        qsquare[i] = 0.0;
        term1[i] = 0.0;
        term2[i] = 0.0;
        term3[i] = 0.0;
    }

    th0 = 15.0 * pi/180.0;
    thf = 75.0 * pi/180.0;
    thdo = 0.0;
    thdf = 0.0;
    tf = 3.0;
    t = 3.0;
    cubic_coeffs(coeffs);

    /* Evaluating individual terms of the state space equation i.e., \[ A(\theta)\dot{\theta} + B(\theta)\ddot{\theta} + C(\theta)\dot{\theta}^{2} + g(\theta) = \tau \]*/
    for(i = 1; i <= 6; i++)
    {
        theta[i] = coeffs[0] + coeffs[1]*t + coeffs[2]*t^2 + coeffs[3]*t^3;
        thdo[i] = coeffs[1] + 2.0*coeffs[2]*t + 3.0*coeffs[3]*t^2;
        thdbld[i] = 2.0*coeffs[2] + 6.0*coeffs[3]*t;
    }
armat(ar);
brmat(br);
CRmat(CR);
grmat(gr);

/*for(i=1;i<=6;i++)
thdot[i]=(float)(i);
*/
for(i=1;i<=6;i++)
printf("thdot %fn",thdot[i]);

for(i=1;j=2;i<6;i++)
for(j=3;i<9;i++)
qq[i]=thdot[1]*thdot[j];
for(i=6;j=4;i<12;i++)
qq[i]=thdot[2]*thdot[j];
for(i=10;j=5;i<14;i++)
qq[i]=thdot[3]*thdot[j];
for(i=13;j=6;i<15;i++)
qq[i]=thdot[4]*thdot[j];
qq[15]=thdot[5]*thdot[6];

for(i=1;i<=15;i++)
printf("qq %dn",i,qq[i]);
printf("thdbldt*** %f %f %f %f %f %fn",thdbldt[1],thdbldt[2],thdbldt[3],thdbldt[4],thdbldt[5],thdbldt[6]);

for(i=1;i<=6;i++)
qsquare[i]=thdot[i]*thdot[i];
for(i=1;i<=6;i++)
printf("qsquare %fn",qsquare[i]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
term1[i]=term1[i]+ar[i][j]*thdbldt[j];
for(i=1;i<=6;i++)
for(j=1;j<=15;j++)
term2[i]=term2[i]+br[i][j]*qq[j];
for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
term3[i]=term3[i]+CR[i][j]*qsquare[j];
for(i=1;i<=6;i++)
printf("thdbldt,terms %f %f %f %f %f %fn",thdbldt[i],term1[i],term2[i],term3[i]);
APPENDIX D

for(i=1;i<=6;i++)
    torque[i] = term1[i] + term2[i] + term3[i] + gr[i];

for(i=1;i<=6;i++)
    printf("torque%d %f
",i,torque[i]);

if ((fpwrite=fopen("rlmfrc5.dat","w")) == NULL)
{
    printf("Error opening output data file rlmfrc5.dat\n");
    exit(0);
}

for(count=1;count<=6;count++)
{
    fprintf(fpwrite,"%f %f %f %f %f
",theta[count],thdot[count],term2[count],term3[count],gr[count]);
    fprintf("output_1 %f %f %f %f %f
",theta[count],thdot[count],term2[count],term3[count],gr[count]);
}

for(count=1;count<=6;count++)
{
    fprintf(fpwrite,"%f %f %f %f %f %f
",ar[count][1],ar[count][2],ar[count][3],ar[count][4],ar[count][5],ar[count][6]);
    fprintf("output_2 %f %f %f %f %f %f
",ar[count][1],ar[count][2],ar[count][3],ar[count][4],ar[count][5],ar[count][6]);
}

fclose(fpwrite);

return;
}

void armat(ar)
float ar[7][7];
{
    int i,j;

    for(i=0;i<=7;i++)
    for(j=0;j<=7;j++)
        ar[i][j]=0.0;

    a r [ 1 ] [ 1 ] = ( i m 1 + i 1 + i 3 * c c 2 + i 7 * s s 2 3 + i 1 0 * s c 2 3 + i 1 1 * s c 2 + i 2 0 * ( s s 3 * ( s s 3 * ( 1 + c c 4 ) - 1 ) - 2.0 * s c 2 3 * c c 2 4 ) + i 2 1 * s s 2 3 * c c 4 + 2.0 * ( i s 5 * s s 2 3 + i 1 2 * c c 2 3 + i 1 5 * s s 2 3 * c c 4 * s s 5 ) + i 1 6 * c c 2 3 * c c 4 * s s 5 + i 1 8 * s s 4 * s s 5 + i 2 2 * ( s c 2 3 * c c 5 + c c 2 3 * c c 4 * s s 5 ) );
    a r [ 1 ] [ 2 ] = ( i 4 * s s 2 + i 8 * c c 2 + i 9 * c c 2 + i 1 3 * s c 2 3 - i 1 5 * c c 3 * s s 4 * s s 5 + i 1 6 * s s 2 3 * c c 4 + 2.0 * ( i s 5 * s s 2 3 * i 1 2 * c c 2 3 + i 1 5 * ( s s 3 * s s 5 + s s 2 3 * c c 4 * s s 5 ) + i 1 6 * c c 2 3 * c c 4 * s s 5 + i 1 8 * s s 4 * s s 5 + i 2 2 * ( s c 2 3 * c c 5 + c c 2 3 * c c 4 * s s 5 ) ));
    a r [ 1 ] [ 3 ] = ( i 8 * c c 3 + i 1 3 * s c 2 3 - i 1 5 * c c 3 * s s 4 * s s 5 + i 1 6 * s s 2 3 * c c 4 + 2.0 * ( s c 2 3 * c c 4 * s s 5 + i 1 9 * s s 4 * c c 5 + i 2 0 * s s 5 * c c 4 * s s 5 + i 2 2 * s c 2 3 * c c 4 * s s 5 ) );
    a r [ 1 ] [ 4 ] = ( i 1 4 * c c 3 + i 1 5 * s c 2 3 + i 1 6 * s c 2 3 * c c 4 + i 1 7 * s s 3 * s s 4 + i 1 8 * s s 3 * s s 4 * c c 5 + i 2 0 * s s 5 * c c 4 * s s 5 + i 2 2 * s s 5 * s c 4 * s s 5 );
    a r [ 1 ] [ 5 ] = ( i 1 5 * s s 3 * s s 4 * s c 5 + i 1 6 * s c 2 3 * s s 4 * c c 5 + i 1 7 * s s 3 * s s 4 + i 1 8 * s s 3 * s s 4 * c c 5 + i 1 9 * s s 4 * s s 5 + i 2 0 * s s 5 * c c 4 * s s 5 + i 2 2 * s c 2 3 * c c 4 * s s 5 );
    a r [ 1 ] [ 6 ] = ( i 2 3 * c c 3 * s s 5 * s s 4 * s s 5 );

    a r [ 2 ] [ 2 ] = ( i m 2 + i 2 + i 6 + i 2 0 * s s 4 * s s 5 + i 2 1 * s s 4 + 2.0 * ( i s 5 * s s 2 3 + i 1 2 * c c 3 + i 1 5 * c c 5 + i 1 6 * s s 3 * c c 4 * s s 5 ) + i 2 2 * c c 4 * s s 5 );
    a r [ 2 ] [ 3 ] = ( i s 5 * s s 3 + i 1 2 * c c 3 + i 1 6 * s s 3 * c c 4 * s s 5 )
    a r [ 2 ] [ 4 ] = ( -i 1 5 * s s 4 * s s 5 - i 1 6 * s c 2 3 * s s 4 * s s 5 + i 2 0 * s s 5 * c c 4 * s s 5 );
    a r [ 2 ] [ 5 ] = ( i 1 5 * c c 4 * c c 5 + i 1 6 * c c 3 * s s 5 + s s 4 * c c 5 ) + i 1 7 * c c 4 + i 2 2 * s s 5 );
    a r [ 2 ] [ 6 ] = ( i 2 3 * s s 4 * s s 5 );
return;
}

void brmat(br)
float br[7][16];
{
 int i, j;

for(i=0;i<=7;i++)
for(j=0;j<=16;j++)
br[i][j]=0.0;

br[1][1] = (2.0*(-i*sc2 + i5*c223 + i7*sc23 - i12*s223 + i15*(2.0*sc23*c5 + (1-2.0*ss23)*c4*s5) + i16*(c223*c5 - s223*c4*s5) + i21*sc23*c4 + i26*(1+c4)*sc23*s5 -(1-2.0*ss23)*c4*s5) + i22*((1-2.0*ss23)*c5 - 2.0*sc23*c4*s5) + i10*(1-2.0*ss23) + i1*(1-2.0*ss23));

br[1][2] = (2.0*(i5*c23 + i7*sc23 - i12*c23 + i15*(2.0*sc23*c5 + (1-2.0*ss23)*c4*s5) + i16*c23*c4 - 2.0*sc23*c4*s5) + i21*sc23*c4 + i26*(1+c4)*sc23*s5 -(1-2.0*ss23)*c4*s5) + i22*((1-2.0*ss23)*c5 - 2.0*sc23*c4*s5) + i10*(1-2.0*ss23));

br[1][3] = (2.0*(-i15*sc23*s4*s5 - i16*c23*c4*s4 + i18*c4 + i20*(ss23*s5*sc4 - sc23*ss5*c4) + i12*sc23*c4*s5) + i10*(1-2.0*ss23)*c4*s5) + i22*(1-2.0*ss23)*c5 - 2.0*sc23*c4*s5) + i10*(1-2.0*ss23));

br[1][4] = (2.0*(i20*(ss5*cc4*(1 - sc23) - cc23) - sc3*c4*(1 - 2.0*ss5)) - i15*(ss23*s5 - sc23*c4*c5) - i16*c23(s23*s5 - c23*c4*c5) + i18*s4*c4 + i22*(cc23*c4*c5 - sc23*ss5)));

br[1][5] = (0.0);

br[1][6] = (2.0*(-i8*s23 + i13*c23 + i15*s23*c4*s5 + i18*c23*c4*s5 + s23*c5) + i19*c23*sc4 + i20*s4*(c23*c4*cc5 - s23*sc5) + i22*c23*c4*s5));

br[1][7] = (-i18*s23*c4*s5 + i19*c23*(1 - (2*ss4)) + i20*s23*(1-2.0*ss4*cc5) - i14*s23);

br[1][8] = (i17*s23*c4 + i18*2.0*(s23*c4*c5 + c23*s5) + i20*s4*(c23*(1 - 2.0*ss5) - s23*c4*2.0*sc5));

br[1][9] = (-i23*(s23*c5 + c23*4*c5));

br[1][10] = (br[1][7]);

br[1][11] = (br[1][8]);

br[1][12] = (br[1][9]);
APPENDIX D

\[
\begin{align*}
br[1][13] &= (2.0*(i15*s23*c4*s5 + i16*e2*c4*e5 + i18*e23*s4*e5 + i22*e23*c4*e5) + i17*s23*c4*i20*(s23*c4*(1 - 2.0*s85) + 2.0*e23*s85)); \\
br[1][14] &= (i23*s23*s84*s5); \\
br[1][15] &= (-i23*(s23*s85 + s23*c4*e5)); \\
br[2][1][0] &= (0.0); \\
br[2][2][0] &= (0.0); \\
br[2][3] &= (i14*s23 + i19*s23*(1.0 - (2.0*s84)) + 2.0*(-i15*c23*c4*s5 + i16*s2*c4*s5 + i20*(s23*(s84*s4 - 0.5) + c23*c4*s85) + i22*s23*c4*s85)); \\
br[2][4] &= (2.0*(-i15*c23*s4*c5 + i22*s23*s4*c5 + i16*s2*s4*c5) - i17*c23*s4 + i20*(c23*s4*(1.0 - 2.0*s85) - 2.0*s23*s84*s85)); \\
br[2][5] &= (-br[1][9]); \\
br[2][6] &= (2.0*(-i12*s3 + i5*e3 + i16*(c3*e5 - s3*c4*e5))); \\
br[2][7] &= (2.0*(-i16*c3*c4*s5 + i20*s3*c4*s85 + i21*c4 - i22*c4*s5)); \\
br[2][8] &= (2.0*(-i15*s5 + i16*(c3*c4*c5 - s3*s5) + i20*s84*s85 + i22*c4*e5)); \\
br[2][9] &= (0.0); \\
br[2][10] &= (br[2][7]); \\
br[2][11] &= (br[2][8]); \\
br[2][12] &= (0.0); \\
br[2][13] &= (2.0*(-i15*s4*c5 - i16*s3*c4*e5) - i17*s4 + i20*s4*(1 - 2.0*s85)); \\
br[2][14] &= (i23*c4*s85); \\
br[2][15] &= (i23*s4*c5); \\
br[3][1][0] &= (0.0); \\
br[3][2][0] &= (0.0); \\
br[3][3] &= (2.0*(-i15*c23*c4*s5 + i22*s23*c4*s5 + i20*(s23*(s84*s4 - 0.5) + c23*c4*s85)) + i14*s23 + i19*s23*(1.0 - (2.0*s84))); \\
br[3][4] &= (2.0*(-i15*c23*s4*c5 + i22*s23*s4*c5) - i17*c23*s4 + i20*s4*(c23*(1.0 - 2.0*s85) - 2.0*s23*c4*s85)); \\
br[3][5] &= (-br[1][12]); \\
br[3][6] &= (0.0); \\
br[3][7] &= (2.0*(i20*s84*s85 + i21*c4 - i22*s4*e5)); \\
br[3][8] &= (2.0*(-i15*s5 + i20*s84*s85 + i22*c4*e5)); \\
br[3][9] &= (0.0); \\
br[3][10] &= (br[3][7]); \\
br[3][11] &= (br[3][8]); \\
br[3][12] &= (0.0); \\
br[3][13] &= (-i15*2.0*s4*c5 - i17*s4 + i20*s4*(1.0 - 2.0*s85)); \\
br[3][14] &= (br[2][14]); \\
br[3][15] &= (br[2][15]); \\
br[4][1] &= (-br[2][3]); \\
br[4][2] &= (-br[3][3]); \\
br[4][3] &= (0.0); \\
br[4][4] &= (-i20*(s23*c4*(1.0 - 2.0*s85) + 2.0*c23*s85) - i17*s23*c4); \\
br[4][5] &= (-br[1][14]); \\
br[4][6] &= (-br[3][7]); \\
br[4][7] &= (0.0); \\
br[4][8] &= (i17*s4 + i20*s4*(1.0 - 2.0*s85)); \\
br[4][9] &= (-br[2][14]); \\
br[4][10] &= (0.0); \\
br[4][11] &= (br[4][8]); \\
br[4][12] &= (-br[3][14]);
\[ \begin{align*} 
\text{for } (i = 1; i <= 6; i++) \\
\{ \text{printf("br }%f%f%f%f%f%f%f\n",br[i][1],br[i][2],br[i][3],br[i][4],br[i][5],br[i][6],br[i][7],br[i][8]);} \\
\text{printf(" }%f%f%f%f%f%f%f\n",br[i][9],br[i][10],br[i][11],br[i][12],br[i][13],br[i][14],br[i][15]);} \\
\//\text{printf("from br br11 = }%f\n",br[1][1]);} \\
\text{return;}} \\
\} \\
\text{void CRmat(CR)} \\
\text{float CR[7][7];} \\
\{ \\
\text{int } i,j; \\
\text{extern float br[7][16];} \\
\text{for}(i=0;i<=7;i++) \\
\text{for}(j=0;j<=7;j++) \\
\text{CR[i][j]} = 0.0; \\
\} \]
APPENDIX D

//printf("from inside CR br\n\n= %fin",br[1][l]);

CR[1][1]=(0.0);
CR[1][2]=(i4*c2 - i8*s23 - i9*s2 + i13*c23 + i15*s23*s4*s5 + i16*c2*s4*s5 + i18*(c23*c4*s5 + s23*c5) + i19*c23*s4 + i20*s4*(c23*c4*cc5 - s23*sc5) + i22*c23*s4*s5);
CR[1][3]=(0.5*br[1][6]);
CR[1][4]=(-i15*s23*s4*s5 - i16*c2*s4*s5 + i18*c23*c4*s5 + i20*s23*s4*sc5 - i22*c23*s4*s5);
CR[1][5]=(-i15*s23*s4*s5 - i16*c2*s4*s5 + i18*(s23*c5 + c23*c4*s5) - i22*c23*s4*s5);
CR[1][6]=(0.0);

CR[2][1]=(-0.5*br[1][1]);
CR[2][2]=(0.0);
CR[2][3]=(0.5*br[2][6]);
CR[2][4]=(-i15*c4*s5 - i16*s3*c4*s5 + i20*c4*sc5);
CR[2][5]=(-i15*c4*s5 + i16*(c3*c5 - s3*c4*s5) + i22*c5);
CR[2][6]=(0.0);

CR[3][1]=(-0.5*br[1][2]);
CR[3][2]=(-CR[2][3]);
CR[3][3]=(0.0);
CR[3][4]=(-i15*c4*s5 + i20*c4*sc5);
CR[3][5]=(-i15*c4*s5 + i22*c5);
CR[3][6]=(0.0);

CR[4][1]=(-0.5*br[1][3]);
CR[4][2]=(-0.5*br[2][7]);
CR[4][3]=(0.5*br[4][6]);
CR[4][4]=(0.0);
CR[4][5]=(0.0);
CR[4][6]=(0.0);

CR[5][1]=(-0.5*br[1][4]);
CR[5][2]=(-0.5*br[2][8]);
CR[5][3]=(0.5*br[5][6]);
CR[5][4]=(-0.5*br[4][13]);
CR[5][5]=(0.0);
CR[5][6]=(0.0);

CR[6][1]=(0.0);
CR[6][2]=(0.0);
CR[6][3]=(0.0);
CR[6][4]=(0.0);
CR[6][5]=(0.0);
CR[6][6]=(0.0);

for(i=1;i<7;i++)
printf("CR \%f \%f \%f \%f \%f \%f\n",CR[i][1],CR[i][2],CR[i][3],CR[i][4],CR[i][5],CR[i][6]);
for (i=1;i<=6;i++)

return;
}

void grmat(gr)
APPENDIX D

```c
float gr[7];
{
    int i;
    gr[1] = (0.0);
    gr[2] = (gg1*c2 + gg2*s23 + gg3*s2 + gg4*c23 + gg5*(s23*c5 + c23*c4*s5));
    gr[3] = (gg2*s23 + gg4*c23 + gg5*(s23*c5 + c23*c4*s5));
    gr[4] = (-gg5*s23*s4*s5);
    gr[5] = (gg5*(c23*s5 + s23*c4*c5));
    gr[6] = (0.0);

    for(i=1;i<=6;i++)
        printf("gr %fn", gr[i]);

    return;
}

void cubic_coeff(coeff)
float coeff[4];
{
    int i;
    coeff[0]=th0;
    coeff[1]=thd0;
    coeff[2]=3.0/(tf*tf)*(thf-th0)-2.0/tf*thdf-1.0/tf*thdf;
    coeff[3]=-2.0/(tf*tf*tf)*(thf-th0)+1.0/(tf*tf)*(thdf+thd0);

    for(i=0;i<=3;i++)
        printf("coeff %fn", coeff[i]);

    return;
}

float *vector(nl,nh)
int nl,nh;
{
    int nrerror();
    float *v;
    v = (float*)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror ("allocation failure in vector()");
    return v-nl;
}

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void nrerror(error_text)
char error_text[];
```
APPENDIX D

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define s1 (sin(theta[1]))
#define c1 (cos(theta[1]))
#define c23 (cos((theta[2])+(theta[3])))
#define s23 (sin(theta[2])+(theta[3]))
#define c2 (cos(theta[2]))
#define s2 (sin(theta[2]))
#define s4 (sin(theta[4]))
#define c4 (cos(theta[4]))
#define s5 (sin(theta[5]))
#define c5 (cos(theta[5]))

/* Defining factorizations for trigonometric terms that are functions of joint angles */

/* Defining fixed link parameters as per Denavit-Hartenberg notation */

#define a2 (0.4318)
#define a3 (-0.0203)
#define d4 (0.4331)

#define d2 (0.2435)
#define d3 (-0.0934)
```
#define pi (3.1415927)

#define TINY 1.0e-20;
#define N (6) /**< for 6x6 "jactrans" ie to find inv. of "jactrans" matrix as "jactrinv" matrix **********/

FILE *fpread;

float **a,**jactrinv,d,*col;
int *indx;
float **jcinvrse;

float theta[7]; /**< (6x1) Vector of Joint Angles \theta_1, \theta_2, ..., \theta_6 */
float thdot[7]; /**< (6x1) Vector \dot{\theta} */
int i,j,k;
float jact[7][7]; /**< (6x6) Jacobian Matrix */
float jactdot[7][7]; /**< (6x6) J matrix */

float ar[7][7],gr[7]; /**< A(6x6) matrix and the g(6x1) vector */
float term2[7],term3[7];

float jactrans[7][7]; /**< (6x6) \bold{J} */
float iden[7][7]; /**< (6x6) Identity matrix */
float fretemp1[7],fretemp2[7][7],fretemp3[7][7][7],fretemp4[7][7][7][7],fretemp5[7][7];
float xdotdot[7],freterm1[7],freterm2[7],freterm3[7],freterm4[7][7]; /**< xdotdot: (6x1) Vector of \ddot{x} */
float term23[7];
float chkterm2[7],chkterm3[7];
float force[7]; /**< (6x1) Force-Torque Vector acting on the End-Effector */

int count;

main()
{
    void ludcmp(),lubksb();

    float **matrix();
    float *vector();

    void jacobian(); /**< Returns the (6x6) Jacobian Matrix */
    void jacobdot(); /**< Returns the (6x6) J Matrix */

    theta[0]=0.0;

    for(i=0;i<=7;i++)
    {
        theta[i]=0.0*pi/180.0;
APPENDIX D

\[ \text{thdot}[i] = 0.0; \]
\[ \text{term2}[i] = 0.0; \]
\[ \text{term3}[i] = 0.0; \]
\[ \text{xdotdot}[i] = 0.0; \]
\[ \text{frcterm1}[i] = 0.0; \]
\[ \text{frcterm2}[i] = 0.0; \]
\[ \text{frcterm3}[i] = 0.0; \]
\[ \text{frcterm4}[i] = 0.0; \]
\[ \text{frctemp1}[i] = 0.0; \]
\[ \text{frctemp5}[i] = 0.0; \]
\[ \text{term23}[i] = 0.0; \]
\[ \text{chkterm2}[i] = 0.0; \]
\[ \text{chkterm3}[i] = 0.0; \]
\[ \text{force}[i] = 0.0; \]
Evaluating the individual terms for the Cartesian state space equation i.e.,

\[ F = J^T A(\theta) J^1 \dot{x} - J^T A(\theta) J^1 J \theta + J^T B(\theta, \theta) + J^T G(\theta) \]

```c
for(i=0;i<6;i++)
{theta[i]=90.0*pi/180.0;
thdot[i]=1.0;
xdddot[i]=5.0;}

jacobian(jac);
for(i=1;i<6;i++)
printf("jac main %f %f %f %f %f %f\n",jac[i][1],jac[i][2],jac[i][3],jac[i][4],jac[i][5],jac[i][6]);

jacobdot(jacdot);
for(i=1;i<6;i++)
printf("jacdot main %f %f %f %f %f %f\n",jacdot[i][1],jacdot[i][2],jacdot[i][3],jacdot[i][4],jacdot[i][5],jacdot[i][6]);

jactrinv=matrix(1,N,1,N);
for(i=1;i<6;i++)
for(j=1;j<6;j++)
jactrans[i][j]=jac[i][j];

for(i=1;i<6;i++)
for(j=1;j<6;j++)
a[i][j]=jactrans[i][j];

for(i=1;i<6;i++)
printf(" jactrans %f %f %f %f %f %f %f\n",jactrans[i][1],jactrans[i][2],jactrans[i][3],jactrans[i][4],jactrans[i][5],jactrans[i][6]);
```

ludcmp(a,N,indx,&d);
for(j=1;j<6;j++)
{for(i=1;i<=N;i++)col[i]=0.0;
col[j]=1.0;
APPENDIX D

lubksb(a,N,indx,col);
for(i=1;i<=N;i++) jactrinv[i][j] = col[i];
}

for(i=1;i<=6;i++)
printf(" matrix jactrinv \%f \%f \%f \%f \%f
%%\n",jactrinv[i][1],jactrinv[i][2],jactrinv[i][3],jactrinv[i][4],jactrinv[i][5],jactrinv[i][6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
for(k=1;k<=6;k++)
iden[i][j] = iden[i][j] + jactrinv[i][k]*jactrans[k][j];

for(i=1;i<=6;i++)
printf(" iden_1 \%f \%f \%f \%f \%f \%f
%%\n",iden[i][1],iden[i][2],iden[i][3],iden[i][4],iden[i][5],iden[i][6]);

for(i=0;i<6;i++)
for(j=0;j<=6;j++)
iden[i][j] = 0.0;

col = vector(1,N);
a = matrix(1,N,1,N);
jcinvrse = matrix(1,N,1,N);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
a[i][j] = jac[i][j];

ludcmp(a,N,indx,&d);
for(j=1;j<=N;j++){
    for(i=1;i<=N;i++)
        if(i == N)
            col[i] = 0.0;
        else
            col[i] = 1.0;
    lubksb(a,N,indx,col);
    for(i=1;i<=N;i++)
        jcinvrse[i][j] = col[i];
}

for(i=1;i<=6;i++)
printf(" matrix jcinvrse \%f \%f \%f \%f \%f \%f
%%\n",jcinvrse[i][1],jcinvrse[i][2],jcinvrse[i][3],jcinvrse[i][4],jcinvrse[i][5],jcinvrse[i][6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
for(k=1;k<=6;k++)
iden[i][j] = iden[i][j] + jcinvrse[i][k]*jac[k][j];

for(i=1;i<=6;i++)
APPENDIX D

printf("iden_2 %s %f %f %f %f %fn",iden[i][1],iden[i][2],iden[i][3],iden[i][4],iden[i][5],iden[i][6]);

for(i=1;i <= 6;i++)
for(j=1;j <= 6;j++)
frctempl[i] = frctemp1[i] + jeinvrse[i][j]*xdotdot[j];

printf("frctemp %f %f %f %f %f %fn",frctemp1[1],frctemp1[2],frctemp1[3],frctemp1[4],frctemp1[5],frctemp1[6]);

for(i=1;i <= 6;i++)
for(j=1;j <= 6;j++)
for(k=1;k <= 6;k++)
frctemp2[i][j] = frctemp2[i][j] + jactrinv[i][k]*ar[k][j];

for(i=1;i <= 6;i++)
for(j=1;j <= 6;j++)
frcterml[i] = frcterml[i] + frctemp2[i][j]*frctempl[i];

printf("frcterml %f %f %f %f %f %f %f %fn",frcterml[1],frcterml[2],frcterml[3],frcterml[4],frcterml[5],frcterml[6]);

for(i=1;i <= 6;i++)
for(j=1;j <= 6;j++)
for(k=1;k <= 6;k++)
frctemp3[i][j] = frctemp3[i][j] + jeinvrse[i][k]*jacect[k][j];

for(i=1;i <= 6;i++)
for(j=1;j <= 6;j++)
for(k=1;k <= 6;k++)
frctemp4[i][j] = frctemp4[i][j] + frctemp2[i][k]*frctemp3[k][j];

for(i=1;i <= 6;i++)
for(j=1;j <= 6;j++)
for(k=1;k <= 6;k++)
frctermt2[i] = frctermt2[i] + frctemp4[i][j]*thdot[j];

printf("frctermt2 %f %f %f %f %f %f %f %f %fn",frctermt2[1],frctermt2[2],frctermt2[3],frctermt2[4],frctermt2[5],frctermt2[6]);

for(i=1;i <= 6;i++)
for(j=1;j <= 6;j++)
frcterm3[i] = frcterm3[i] + jactrinv[i][j]*(term2[j] + term3[j]);
for(i=0;i<=6;i++)
term23[i]=term22[i]+term3[i];

printf("term23 %f %f %f %f %f
",term23[1],term23[2],term23[3],term23[4],term23[5],term23[6]);

for(i=0;i<=6;i++)
frcterm3[i]=0.0;

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
frcterm3[i]=frcterm3[i]+jactrinv[i][j]*term23[j];

printf("frcterm3 (Alternate 1) %f %f %f %f %f
",frcterm3[1],frcterm3[2],frcterm3[3],frcterm3[4],frcterm3[5],frcterm3[6]);

for(i=0;i<=6;i++)
frcterm3[i]=0.0;

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
chkterm2[i]=chkterm2[i]+jactrinv[i][j]*term2[i];

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
chkterm3[i]=chkterm3[i]+jactrinv[i][j]*term3[j];

for(i=0;i<=6;i++)
chkterm2[i]=chkterm2[i]+chkterm3[i];

printf("frcterm3 (Alternate 2) %f %f %f %f %f
",frcterm3[1],frcterm3[2],frcterm3[3],frcterm3[4],frcterm3[5],frcterm3[6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
frcterm4[i]=frcterm4[i]+jactrinv[i][j]*grf[j];

printf("frcterm4 %f %f %f %f %f
",frcterm4[1],frcterm4[2],frcterm4[3],frcterm4[4],frcterm4[5],frcterm4[6]);

for(i=1;i<=6;i++)
force[i]=frcterm1[i]+frcterm2[i]+frcterm3[i]+frcterm4[i];

.
for(i=1;i<=6;i++)
    printf("force %fn",force[i]);

return;

void jacobian(jac)
float jac[7][7];
{
    jac[1][1] = -s1*(c23*a3 + s23*d4 + c2*a2) -c1*(d2+d3);
    jac[1][2] = c1*(-s23*a3 + c23*d4 - s2*a2);
    jac[1][3] = c1*(-s23*a3 + c23*d4);
    jac[1][4] = 0.0;
    jac[1][5] = 0.0;
    jac[1][6] = 0.0;

    jac[2][1] = c1*(c23*a3 + s23*d4 + c2*a2) -s1*(d2+d3);
    jac[2][2] = s1*(-s23*a3 + c23*d4 -s2*a2);
    jac[2][3] = s1*(-s23*a3 + c23*d4);
    jac[2][4] = 0.0;
    jac[2][5] = 0.0;
    jac[2][6] = 0.0;

    jac[3][1] = 0.0;
    jac[3][2] = -(c23*a2 + s23*d4 + c2*a2);
    jac[3][3] = -(c23*a3 + s23*d4);
    jac[3][4] = 0.0;
    jac[3][5] = 0.0;
    jac[3][6] = 0.0;

    jac[4][1] = 0.0;
    jac[4][2] = -s1;
    jac[4][3] = -s1;
    jac[4][4] = c1*s23;
    jac[4][5] = -c1*c23*s4 -s1*e4;
    jac[4][6] = (c1*c23*c4 -s1*e4)*s5 +c1*s23*e5;

    jac[5][1] = 0.0;
    jac[5][2] = c1;
    jac[5][3] = c1;
    jac[5][4] = s1*s23;
    jac[5][5] = -s1*c23*s4 +c1*e4;
    jac[5][6] = (s1*c23*c4 +c1*e4)*s5 +s1*s23*e5;

    jac[6][1] = 1.0;
    jac[6][2] = 0.0;
    jac[6][3] = 0.0;
APPENDIX D

for(i=1; i< 6; i++)
printf("jac subr %f %f %f %f \n",jac[i][1],jac[i][2],jac[i][3],jac[i][4],jac[i][5],jac[i][6]);

return;
}

void jacobdot(jacdot)
float jacdot[7][7];
{
jacdot[1][4] = 0.0;
jacdot[1][5] = 0.0;
jacdot[1][6] = 0.0;

-jacdot[2][4] = 0.0;
jacdot[2][5] = 0.0;
jacdot[2][6] = 0.0;

jacdot[3][1] = 0.0;
jacdot[3][4] = 0.0;
jacdot[3][5] = 0.0;
jacdot[3][6] = 0.0;

jacdot[4][1] = 0.0;
jacdot[4][2] = -c1*thdot[1];
jacdot[4][3] = -c1*thdot[1];
+s1*s4*thdot[4];

jacdot[5][1] = 0.0;
jacdot[5][2] = -s1*thdot[1];
jacdot[5][3] = -s1*thdot[1];
for(i=1;i<=6;i++)
    printf("jacdot subr %f %f %f %f %f %f\n", jacdot[i][1], jacdot[i][2], jacdot[i][3], jacdot[i][4], jacdot[i][5], jacdot[i][6]);

return;

}

void ludcmp(a,n,indx,d)
int n,*indx;
float **a,*d;
{
    int i,imax,j,k;
floa big,dum,sum,temp;
float *vv,*vector();
void nrerror(),free_vector();

vv=vector(1,n);
*d=1.0;
for(i=1;i<=n;i++)
    big=0.0;
    for(j=1;j<=n;j++)
        if ((temp=fabs(a[i][j])) > big) big=temp;
    if (big == 0.0) nrerror("Singular matrix in routine LUDCMP");
vv[i]=1.0/big;
}
for(j=1;j<=n;j++)
for(i=1;i<j;i++)
    sum=a[i][j];
    for (k=1;k<j;k++) sum -= a[i][k]*a[k][j];
a[i][j]=sum;
}
big=0.0;
for(i=j;i<=n;i++)
    sum=a[i][j];
    for (k=1;k<j;k++)
        sum -= a[i][k]*a[k][j];
a[i][j]=sum;
    if ( (dum=vv[i]*fabs(sum)) >= big )

APPENDIX D

jacidot[6][1] = 0.0;
jacidot[6][2] = 0.0;
jacidot[6][3] = 0.0;
jacidot[6][5] = c23*c4*s4*(thdot[2]+thdot[3]) +s23*c4*thdot[4];
for(i=1;i<=6;i++)
    printf("jacdot subr %f %f %f %f %f %f\n", jacdot[i][1], jacdot[i][2], jacdot[i][3], jacdot[i][4], jacdot[i][5], jacdot[i][6]);
return;
}


\[ \text{big}=\text{dum}; \]
\[ \text{imax} = i; \]

\}

if (j != imax) {
  for(k=1;k <= n;k++) {
    \text{dum} = a[imax][k];
    a[imax][k] = a[j][k];
    a[j][k] = \text{dum};
  }
  \text{v} = \text{-(v)};
  v[i] = v[j];
}

indx[i] = imax;
if (a[i][i] == 0.0) a[i][i] = TINY;
if (j != n) {
  \text{dum} = 1.0/(a[j][j]);
  for(i=j+1;i <= n;i++) a[i][j] *= dum;
}

free_vector(v,1,n);

void lubksb(a,n,indx,b)
float **a,b[];
int n,*indx;
{
int i,ii=0,ip,j;
float sum;

for(i=1;i <= n;i++) {
  ip = indx[i];
  sum = b[ip];
  b[ip] = b[i];
  if (ii)
    for(j=i-1;j >= i-1;j++) sum -= a[i][j]*b[j];
  else if (sum)
    ii=i;
  b[i] = sum;
}
for(i=n;i > = 1;i--){
  sum = b[i];
  for(j=i+1;j <= n;j++) sum -= a[i][j]*b[j];
  b[i] = sum/a[i][i];
}

float *vector(nl,nh)
int nl,nh;
{
  \text{nrerror}();
}
APPENDIX D

```c
float *v;
v = (float *) malloc((unsigned) (nh-nl+1)*sizeof(float));
if (!v) nrerror("allocation failure in vector()");
return v-nl;
}

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void nrerror(error_text)
char error_text[];
{
    void exit();
    fprintf(stderr,"Numerical Recipies run-time error ...
");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"... now exiting to system ...
");
    exit(1);
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i;
    float **m;

    m = (float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for (i=nrl;i <= nrh;i++) {
        m[i] = (float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}
```
D-9 Computer Program to get Forces(s) on the End Effector based on the Approximate Model

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* Defining factorizations for trigonometric terms that are functions of joint angles */

#define s1 (sin(theta[1]))
#define c1 (cos(theta[1]))

#define cc2 (cos(theta[2])*cos(theta[2]))
#define ss23 (sin(theta[2])*sin(theta[3]))
#define c2 (cos(theta[2]))
#define s2 (sin(theta[2]))
#define c23 (cos((theta[2])+(theta[3])))
#define s3 (sin(theta[3]))
#define c3 (cos(theta[3]))
#define s4 (sin(theta[4]))
#define c4 (cos(theta[4]))
#define s5 (sin(theta[5]))
#define c5 (cos(theta[5]))
#define sc2 (sin(theta[2])*cos(theta[2]))
#define sc23 (sin(theta[2])*cos(theta[3]))
#define ss4 (sin(theta[4])*sin(theta[4]))

/* Defining fixed link parameters as per Denavit-Hartenberg notation */

#define a2 (0.4318)
define a3 (-0.0203)
define d4 (0.4331)
define d2 (0.2435)
define d3 (-0.0934)
define pi (3.1415927)
define TINY 1.0e-20;
define N (6) /*for 6X6 "jactrans" ie to find inv. of "jactrans" matrix as "jactrinv" matrix */

float **a,**jactrinv,d,*col;
int *indx;
float **jcinvrse;

int i,j;
float theta[7]; /* (6x1) Vector of Joint Angles $\theta_1, \theta_2, \ldots, \theta_6$ */

float ap[7][7], bp[7][16], CP[7][7], gp[7]; /* A(6x6), B(6x15), and C(6x6) matrices and the g vector */

float coeff[4]; /* Vector of Cubic Coefficients */
float th0, thdf, thdd, tf; /* $\theta_0, \dot{\theta}, \ddot{\theta}, \dddot{\theta}, \dddot{\theta}$ */
float thd[7], thdld[7]; /* (6x1) Vector $\dot{\theta}$ and (6x1) Vector $\ddot{\theta}$ */
float t; /* Time Variable */
float qq[16], qsquare[7]; /* (1x15) Vector of velocity products i.e., $\dot{\theta}^2$, and (1x6) Vector of squared velocities i.e., $\partial^2$ */

float torque[7]; /* (6x1) Torque Vector */
float jac[7][7]; /* (6x6) Jacobian Matrix */
float jact[7][7]; /* (6x6) J matrix */
float jactrans[7][7]; /* (6x6) $J^T$ */
float iden[7][7]; /* Identity Matrix */
float frctemp[1][7], frctemp2[7][7], frctemp3[7][7], frctemp4[7][7], frctemp5[7];
float xdotdot[7], frcterm1[7], frcterm2[7], frcterm3[7], frcterm4[7], frcterm5[7]; /* xdotdot: (6x1) Vector of $\ddot{x}$ */
float term23[7];
float chkterm2[7], chkterm3[7];
float force[7]; /* (6x1) Force-Torque Vector acting on the End-Effector */

main()
{
    void apmat(); /* Returns the expressions giving the elements of the Kinetic Energy matrix */
    void bpmat(); /* Returns the expressions giving the elements of the Coriolis matrix */
    void CPmat(); /* Returns the expressions giving the elements of the Centrifugal matrix */
    void gpmat(); /* Returns the Gravity vector */
    void ludecmp(), lubksb();

    float **matrix();
    float *vector();

    void cubic_coef(); /* Returns the Vector of cubic coefficients */
    int k;
    void jacobian(); /* Returns the (6x6) Jacobian Matrix */
    void jacobdot(); /* Returns the (6x6) $J$ Matrix */

    theta[0]=0.0;

    for(i=0; i<7; i++)
    {
        theta[i]=0.0*pi/180.0;
        thd[i]=0.0;
        thdld[i]=0.0;
        torque[i]=0.0;
        qsquare[i]=0.0;
        term1[i]=0.0;
        term2[i]=0.0;
        term3[i]=0.0;
        xdotdot[i]=0.0;
    }
}
APPENDIX D

frcterm1[i] = 0.0;
frcterm2[i] = 0.0;
frcterm3[i] = 0.0;
frcterm4[i] = 0.0;
frctemp1[i] = 0.0;
frctemp5[i] = 0.0;
term23[i] = 0.0;
chkterm2[i] = 0.0;
chkterm3[i] = 0.0;
force[i] = 0.0;

for(i = 0; i <= 6; i++)
for(j = 0; j <= 6; j++)
{jastrans[i][j] = 0.0;
 jacdot[i][j] = 0.0;
 jacf[i][j] = 0.0;
 iden[i][j] = 0.0;
 frctemp2[i][j] = 0.0;
 frctemp3[i][j] = 0.0;
 frctemp4[i][j] = 0.0;
}

th0 = 15.0*pi/180.0;
thf = 75.0*pi/180.0;
thd0 = 0.0;
thdf = 0.0;
tf = 3.0;
t = 3.0;

cubie_coeff(coef);
for(i = 0; i <= 3; i++)
printf("coeff %fn", coef[i]);

for(i = 1; i <= 6; i++)
{theta[i] = coef[0] + coef[1]*t + coef[2]*t*t + coef[3]*t*t*t;
 thdot[i] = coef[1] + 2.0*coef[2]*t + 3.0*coef[3]*t*t;
 thdld[i] = 2.0*coef[2] + 6.0*coef[3]*t;} 
apmat(ap);
bpmat(bp);
CPmat(CP);
gpmat(gp);

/*for(i = 1; i <= 6; i++)
 thdot[i] = (float)(i); */
for(i = 1; i <= 6; i++)
printf("thdot %fn", thdot[i]);

<!--[if !supportLists]--> /* Evaluating individual terms of the state space equation i.e., A(θ)(θ) + B(θ)(θ) + C(θ)(θ^2) + g(θ) = r */

for(i = 1; j = 2; i <= 5; j < = 6; i++ j++)
qq[i] = thdot[1]*thdot[j];
for(i = 6; j = 3; i < = 9; j < = 6; i++ j++)
qq[i] = thdot[2]*thdot[j];
for (i=1; i<=6; i++)
printf("qq %d %f\n", i, qq[i]);

for (i=1; i<=6; i++)
qsquare[i]=thdot[i]*thdot[i];

for (i=1; i<=6; i++)
printf("qsquare %f\n",qsquare[i]);

for (i=1; i<=6; i++)
for (j=1; j<=6; j++)
term1[i] = term1[i] + ap[i][j]*thdbldt[j];

for (i=1; i<=6; i++)
for (j=1; j<=6; j++)
term2[i] = term2[i] + bp[i][j]*qq[j];

for (i=1; i<=6; i++)
for (j=1; j<=6; j++)
term3[i] = term3[i] + CP[i][j]*qsquare[j];

for (i=1; i<=6; i++)
torque[i] = term1[i] + term2[i] + term3[i] + gp[i];

for (i=1; i<=6; i++)
printf("thdbldt,terms %f %f %f %f %f\n",thdbldt[i],term1[i],term2[i],term3[i]);

for (i=1; i<=6; i++)
printf("torque %d %f\n",i,torque[i]);

/**** BLUFF CHECKER *******/*

for (i=0; i<=6; i++)
{ theta[i] = 90.0*pi/180.0;
  thdot[i] = 1.0;
  xdotdot[i] = 5.0;} */

/* Evaluating the individual terms for the Cartesian state space equation i.e.,
F = J^T A(\theta) J J^T \dot{x} - J^T A(\theta) J J^T B(\theta, \dot{\theta}) + J^T G(\theta) */

jacobian(jac);
for (i=1; i<=6; i++)
printf("jac main %f %f %f %f %f %f\n", jac[i][1],jac[i][2],jac[i][3],jac[i][4],jac[i][5],jac[i][6]);

jacobdot(jacobdot);
for(i=1;i<=6;i++)
printf("jacdot main %f %f %f %f %f %f
",jacdot[i][1],jacdot[i][2],jacdot[i][3],jacdot[i][4],jacdot[i][5],jacdot[i][6]);

/*** ALITER CAN FIND jactrinv by finding jcinvrse and then taking its transpose ******/
col=vector(1,N);
a=matrix(1,N,1,N);
jactrinv=matrix(1,N,1,N);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
jactrans[i][j]=jac[i][j];

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
a[i][j]=jactrans[i][j];

for(i=1;i<=6;i++)
printf("matrix jactrans %f %f %f %f %f %f
",jactrans[i][1],jactrans[i][2],jactrans[i][3],jactrans[i][4],jactrans[i][5],jactrans[i][6]);

ludcmp(a,N,indx,&d);
for(i=1;j<=N;j++)
    for(i=1;i<=N;i++)col[i]=0.0;
col[j]=1.0;
lubksb(a,N,indx,col);
for(i=1;i<=N;i++) jactrinv[i][j]=col[i];
}

for(i=1;i<=6;i++)
printf(" matrix jactrinv %f %f %f %f %f %f
",jactrinv[i][1],jactrinv[i][2],jactrinv[i][3],jactrinv[i][4],jactrinv[i][5],jactrinv[i][6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
for(k=1;k<=6;k++)
iden[i][j]=iden[i][j] + jactrinv[i][k]*jactrans[k][j];

for(i=1;i<=6;i++)
printf("iden_1 %f %f %f %f %f
",iden[i][1],iden[i][2],iden[i][3],iden[i][4],iden[i][5],iden[i][6]);

for(i=0;i<=6;i++)
for(j=0;j<=6;j++)
iden[i][j]=0.0;

col=vector(1,N);
a=matrix(1,N,1,N);
jcinvrse=matrix(1,N,1,N);
for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
a[i][j]=jac[i][j];

ludemp(a,N,indx,&ld);
for(j=1;j<=N;j++)
{
  for(i=1;i<=N;i++)
    col[i]=0.0;
    col[j]=1.0;
    lubksb(a,N,indx,col);
    for(i=1;i<=N;i++)
      jcinvrse[i][j]=col[i];
}

for(i=1;i<=6;i++)
printf("matrix jcinvrse %f %f %f %f %f %f
%fn",jcinvrse[i][1],jcinvrse[i][2],jcinvrse[i][3],jcinvrse[i][4],jcinvrse[i][5],jcinvrse[i][6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
for(k=1;k<=6;k++)
iden[i][j]=iden[i][j]+jcinvrse[i][k]*jac[k][j];

for(i=1;i<=6;i++)
printf("iden_2 %f %f %f %f %f %f\n",iden[i][1],iden[i][2],iden[i][3],iden[i][4],iden[i][5],iden[i][6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
frcterml[i]=frcterml[i]+jcinvrse[i][j]*dotdot[j];

printf("frcterml %f %f %f %f %f %f\n",frcterml[1],frcterml[2],frcterml[3],frcterml[4],frcterml[5],frcterml[6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
for(k=1;k<=6;k++)
frctemp2[i][j]=frctemp2[i][j]+jactrinv[i][k]*ap[k][j];

for(i=1;i<=6;i++)
printf("frctemp 2 %f %f %f %f %f %f\n",frctemp2[i][1],frctemp2[i][2],frctemp2[i][3],frctemp2[i][4],frctemp2[i][5],frctemp2[i][6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
frterm1[i]=frterm1[i]+frctemp2[i][j]*frctemp1[j];

printf("frterm1 %f %f %f %f %f %f\n",frterm1[1],frterm1[2],frterm1[3],frterm1[4],frterm1[5],frterm1[6]);

for(i=1;i<=6;i++)
for(j=1;j<=6;j++)
for(k=1;k<=6;k++)
frterm2[i][j]=frterm2[i][j]+jactrinv[i][k]*ap[k][j];

printf("frterm2 %f %f %f %f %f %f\n",frterm2[1],frterm2[2],frterm2[3],frterm2[4],frterm2[5],frterm2[6]);
for(k = 1;k <= 6;k++)
frtemp3[i][j] = fretemp3[i][j] + jaclnrf[i][k] * jactds[k][j];

for(i = 1;i <= 6;i++)
printf(" fretemp3 % f % f % f % f % f
% f
", fretemp3[i][1], fretemp3[i][2], fretemp3[i][3], fretemp3[i][4], fretemp3[i][5], fretemp3[i][6]);

for(i = 1;i <= 6;i++)
for(j = 1;j <= 6;j++)
for(k = 1;k <= 6;k++)
frtemp4[i][j] = fretemp4[i][j] + fretemp2[i][k] * fretemp3[k][j];

for(i = 1;i <= 6;i++)
printf(" fretemp4 % f % f % f % f % f
% f
", fretemp4[i][1], fretemp4[i][2], fretemp4[i][3], fretemp4[i][4], fretemp4[i][5], fretemp4[i][6]);

for(i = 1;i <= 6;i++)
for(j = 1;j <= 6;j++)
frterm2[i] = freterm2[i] + fretemp4[i][j] * thdot[j];

printf("frterm2 % f % f % f % f % f
% f
", freterm2[1], freterm2[2], freterm2[3], freterm2[4], freterm2[5], freterm2[6]);

for(i = 1;i <= 6;i++)
for(j = 1;j <= 6;j++)
frterm3[i] = freterm3[i] + jaclnrf[i][j] * (term2[j] + term3[j]);

printf("frterm3 % f % f % f % f % f
% f
", freterm3[1], freterm3[2], freterm3[3], freterm3[4], freterm3[5], freterm3[6]);

for(i = 0;i <= 6;i++)
term23[i] = term2[i] + term3[i];

printf("term23 % f % f % f % f % f
% f
", term23[1], term23[2], term23[3], term23[4], term23[5], term23[6]);

for(i = 0;i <= 6;i++)
frterm5[i] = 0.0;

for(i = 1;i <= 6;i++)
for(j = 1;j <= 6;j++)
frterm3[i] = frterm3[i] + jaclnrf[i][j] * (term23[j]);

printf("freterm3 (Alter 1) % f % f % f % f % f
% f
", freterm3[1], freterm3[2], freterm3[3], freterm3[4], freterm3[5], freterm3[6]);

for(i = 0;i <= 6;i++)
frterm3[i] = 0.0;

for(i = 1;i <= 6;i++)

APPENDIX D

```c
void apmat(ap, cTI, A;

for(i=0; i<6; i++)
frcterm3[i]+=chkterm2[i]+chkterm3[i];

printf("frcterm 3 ( \text{Aliter} _ 2 ) \%f \%f \%f \%f \%f \%f
%f\n", frcterm3[1], frcterm3[2], frcterm3[3], frcterm3[4], frcterm3[5], frcterm3[6]);

```

```c
for(i=0; i<6; i++)
frcterm4[i]+=frcterm3[i]+frcterm4[i];

printf("frcterm4 \%f \%f \%f \%f \%f \%f\n", frcterm4[1], frcterm4[2], frcterm4[3], frcterm4[4], frcterm4[5], frcterm4[6]);

```

```c
void apmat(ap, 7;

for(i=0; i<7; i++)
ap[i][i]=0.0;

```

```c
for(i=0; i<7; i++)
for(j=0; j<7; j++)
ap[i][j]=
```

```c
for(i=1; i<6; i++)
for(j=1; j<6; j++)
chkterm2[i] = chkterm2[i] + jactinv[i][j]* ( term2[j] );

for(i=1; i<6; i++)
for(j=1; j<6; j++)
chkterm3[i] = chkterm3[i] + jactinv[i][j]* ( term3[j] );
```

```c
printf("frcterm 4 \%f \%f \%f \%f \%f \%f\n", frcterm4[1], frcterm4[2], frcterm4[3], frcterm4[4], frcterm4[5], frcterm4[6]);

```

```c
for(i=1; i<6; i++)
force[i] = frcterm1[i] + frcterm2[i] + frcterm3[i] + frcterm4[i];
```

```c
return;
}
```

```c
void apmat(ap, 7;

for(i=0; i<7; i++)
for(j=0; j<7; j++)
ap[i][j]=0.0;

```

```c
ap[1][1] = (2.57 + 1.38*(cc2) + 0.3*(ss23) + 7.44*0.1*(c2)*(s23));
ap[1][2] = (0.69*(s2) - 0.134*(c23) + 0.0238*(c2));
ap[1][3] = (-0.134*c23 + -3.97e-03*s23);
ap[1][4] = (0.0);
ap[1][5] = (0.0);
ap[1][6] = (0.0);
ap[2][2] = (6.79 + 0.744*s3);
ap[2][3] = (0.333 + 0.372*s3 -0.0110*c3);
ap[2][4] = (0.0);
ap[2][5] = (0.0);
ap[2][6] = (0.0);
```
APPENDIX D

```
ap[2][6] = (0.0);
ap[3][3] = (1.16);
ap[3][4] = (-1.25e-03*s4*s5);
ap[3][5] = (1.25e-03*c4*c5);
ap[3][6] = (0.0);
ap[4][4] = (0.20);
ap[4][5] = (0.0);
ap[4][6] = (0.0);
ap[5][5] = (0.18);
ap[5][6] = (0.0);
ap[6][6] = (0.19);

for(i=0;i<=7;i++)
    for(j=0;j<=7;j++)
        ap[i][j]=ap[i][j];

for(i=1;i<7;i++)
    printf("ap %f %f %f %f %f %f \n",ap[i][1],ap[i][2],ap[i][3],ap[i][4],ap[i][5],ap[i][6]);

return;
}

void bpmat(bp)
    float bp[7][16];
{
    int i,j;

    for(i=0;i<=7;i++)
        for(j=0;j<=16;j++)
            bp[i][j]=0.0;

    bp[1][1] = (-2.70*s2 + 0.744*c2*s23 + 0.60*s23 -0.0213*(1-2*s23));
    bp[1][2] = (0.744*c2*c23 +0.60*s23 + 0.022*c2*s23 -0.0213*(1-2*s23));
    bp[1][3] = (-2.50e-03*s23*s4*s5 + 8.60e-04*c4*s5 -2.48e-03*c2*c23*s4*s5);
    bp[1][4] = (-2.50e-03*(s23*s5-s23*c4*c5) - 2.48e-03*c2*(s23*s5-c23*c4*c5) + 8.60e-04*s4*c5);
    bp[1][5] = (0.0);
    bp[1][6] = (0.267*s23 -7.58e-03*c23);
    bp[1][7] = (0.0);
    bp[1][8] = (0.0);
    bp[1][9] = (0.0);
    bp[1][10] = (bp[1][7]);
    bp[1][11] = (bp[1][8]);
    bp[1][12] = (bp[1][9]);
    bp[1][13] = (0.0);
    bp[1][14] = (0.0);
    bp[1][15] = (0.0);
```
APPENDIX D

\[ \begin{align*}
bp[2][1] &= (0.0); \\
bp[2][2] &= (0.0); \\
bp[2][3] &= (1.64e-03*s23 - 2.50e-03*c23*c4*s5 + 2.48e-03*s2*c4*s5 + 0.3e-03*s23*(1-2*ss4)); \\
bp[2][4] &= (-2.50e-03*c23*s4*c5 + 2.48e-03*s2*s4*c5 - 6.42e-04*c23*s4); \\
bp[2][5] &= (-bp[1][9]); \\
bp[2][6] &= (0.0220*s3 + 0.744*e3); \\
bp[2][7] &= (-2.48e-03*c3*s4*s5); \\
bp[2][8] &= (-2.50e-03*s5 + 2.48e-03*(c3*c4*c5*s3*s5)); \\
bp[2][9] &= (0.0); \\
bp[2][10] &= (bp[2][7]); \\
bp[2][11] &= (bp[2][8]); \\
bp[2][12] &= (0.0); \\
bp[2][13] &= (0.0); \\
bp[2][14] &= (0.0); \\
bp[2][15] &= (0.0); \\
bp[3][1] &= (0.0); \\
bp[3][2] &= (0.0); \\
bp[3][3] &= (-2.50e-03*c23*c4*s5 + 1.64e-03*s23 + 0.30e-03*s23*(1-2*ss4)); \\
bp[3][4] &= (-2.50e-03*c23*s4*c5 - 6.42e-04*c23*s4); \\
bp[3][5] &= (-bp[1][12]); \\
bp[3][6] &= (0.0); \\
bp[3][7] &= (0.0); \\
bp[3][8] &= (-2.50e-03*s5); \\
bp[3][9] &= (0.0); \\
bp[3][10] &= (bp[3][7]); \\
bp[3][11] &= (bp[3][8]); \\
bp[3][12] &= (0.0); \\
bp[3][13] &= (-2.5e-03*s4*c5); \\
bp[3][14] &= (bp[2][14]); \\
bp[3][15] &= (bp[2][15]); \\
bp[4][1] &= (-bp[2][3]); \\
bp[4][2] &= (-bp[3][3]); \\
bp[4][3] &= (0.0); \\
bp[4][4] &= (-6.42e-04*s23*c4); \\
bp[4][5] &= (-bp[1][14]); \\
bp[4][6] &= (-bp[3][7]); \\
bp[4][7] &= (0.0); \\
bp[4][8] &= (6.42e-04*s4); \\
bp[4][9] &= (-bp[2][14]); \\
bp[4][10] &= (0.0); \\
bp[4][11] &= (bp[4][8]); \\
bp[4][12] &= (-bp[3][14]); \\
bp[4][13] &= (0.0); \\
bp[4][14] &= (0.0); \\
bp[4][15] &= (0.0); \\
bp[5][1] &= (-bp[2][4]); \\
bp[5][2] &= (-bp[3][4]); \\
bp[5][3] &= (-bp[4][4]); \\
bp[5][4] &= (0.0); 
\end{align*} \]
APPENDIX D

bp[5][5] = (bp[1][15]);
bp[5][6] = (bp[3][8]);
bp[5][7] = (bp[4][8]);
bp[5][8] = (0.0);
bp[5][9] = (bp[2][15]);
bp[5][10] = (bp[5][7]);
bp[5][11] = (0.0);
bp[5][12] = (bp[3][15]);
bp[5][13] = (0.0);
bp[5][14] = (bp[4][15]);
bp[5][15] = (0.0);

bp[6][1] = (bp[1][9]);
bp[6][2] = (bp[1][12]);
bp[6][3] = (bp[1][14]);
bp[6][4] = (bp[1][15]);
bp[6][5] = (0.0);
bp[6][6] = (0.0);
bp[6][7] = (bp[2][14]);
bp[6][8] = (bp[2][15]);
bp[6][9] = (0.0);
bp[6][10] = (bp[6][7]);

for (i = 1; i <= 6; i++)
{
printf("br %f %f %f %f %f %f
", bp[i][1], bp[i][2], bp[i][3], bp[i][4], bp[i][5], bp[i][6], bp[i][7], bp[i][8]);
printf("%f %f %f %f %f %f %f
", bp[i][9], bp[i][10], bp[i][11], bp[i][12], bp[i][13], bp[i][14], bp[i][15]);
}
//printf(" from br br11 = %f\n", bp[1][1]);

return;

void CPmat(CP)
float CP[7][7];
{
int i,j;
extern float bp[7][16];

for(i=0;i<7;i++)
for(j=0;j<7;j++)
CP[i][j]=0.0;

//printf(" from inside CR br11 = %f\n", bp[1][1]);

CP[1][1]=(0.0);
CP[1][2]=(0.69*e2 + 0.134*s23 -0.0238*s2);
CP[1][3]=(0.5*bp[1][6]);
CP[1][4]=(0.0);
APPENDIX D

for(i=1;i < 7;i++)
printf("CR %f %f %f %f %f %f\n",CP[i][1],CP[i][2],CP[i][3],CP[i][4],CP[i][5],CP[i][6]);
return;
}

void gpmat(gp)
float gp[7],
{
    int i;
    gp[1]= (0.0);
    gp[2]= (-37.2*c2 - 8.4*s23 + 1.02*s2 );
    gp[3]= (-8.4*s23 + 0.25*c23 );
    gp[4]= ( 2.8e-02*s23*s4*s5 );
    gp[5]= ( -2.8e-02*c23*s5 + s23*c4*c5 );
}
void cubic_coeff(coeff)
float coeff[4];
{
  int i;

  coeff[0]=th0;
  coeff[1]=thd0;
  coeff[2]=3.0/(tf*tf)*(thf-th0)-2.0/tf*thd0-1.0/tf*thdf;
  coeff[3]=-2.0/(tf*tf*tf)*(thf-th0)+1.0/(tf*tf)*(thdf+thd0);

  for(i=0;i<=3;i++)
    printf("coeff %fn",coeff[i]);

  return;
}

void jacobian(jac)
float jac[7][7];
{
  jac[1][1] = -s1*(c23*a3 + s23*d4 + c2*a2) -c1*(d2+d3);
  jac[1][2] = c1*(-s23*a3 + c23*d4 - s2*a2);
  jac[1][3] = c1*(-s23*a3 + c23*d4);
  jac[1][4] = 0.0;
  jac[1][5] = 0.0;
  jac[1][6] = 0.0;

  jac[2][1] = c1*(c23*a3 + s23*d4 + c2*a2) -s1*(d2+d3);
  jac[2][2] = s1*(-s23*a3 + c23*d4 -s2*a2);
  jac[2][3] = s1*(-s23*a3 + c23*d4);
  jac[2][4] = 0.0;
  jac[2][5] = 0.0;
  jac[2][6] = 0.0;

  jac[3][1] = 0.0;
  jac[3][2] = -(c23*a2 + s23*d4 + c2*a2);
  jac[3][3] = -(c23*a3 + s23*d4);
  jac[3][4] = 0.0;
  jac[3][5] = 0.0;
  jac[3][6] = 0.0;

  jac[4][1] = 0.0;
  jac[4][2] = -s1;
  jac[4][3] = -s1;
  jac[4][4] = c1*s23;
  jac[4][5] = -c1*c23*s4 -s1*c4;
jac[4][6] = (c1*c23*c4 - s1*s4)*s5 + c1*s23*c5;

jac[5][1] = 0.0;
jac[5][2] = c1;
jac[5][3] = c1;
jac[5][4] = s1*s23;
jac[5][5] = -s1*c23*s4 + c1*c4;
jac[5][6] = (s1*c23*c4 + c1*s4)*s5 + s1*s23*c5;

jac[6][1] = 1.0;
jac[6][2] = 0.0;
jac[6][3] = 0.0;
jac[6][4] = s23;
jac[6][5] = s23*s4;
jac[6][6] = -s23*s4*s5 + c23*c5;

for(i=1;i<=6;i++)
    printf("jac subr %f %f %f %f %f %fin", jac[i][1], jac[i][2], jac[i][3], jac[i][4], jac[i][5], jac[i][6]);

return;
}

void jacobdot(jacdot)
float jacdot[7][7];
{
    printf("BETWEEN %f %f %f %f %f %f", a2, a3, d2, d3, d4, c1, s1, c23, s23);
    jacdot[1][4] = 0.0;
    jacdot[1][5] = 0.0;
    jacdot[1][6] = 0.0;

    jacdot[2][4] = 0.0;
    jacdot[2][5] = 0.0;
    jacdot[2][6] = 0.0;
    jacdot[3][1] = 0.0;
    jacdot[3][4] = 0.0;
    jacdot[3][5] = 0.0;
    jacdot[3][6] = 0.0;
APPENDIX D

jacid[3][6] = 0.0;
jacid[4][1] = 0.0;
jacid[5][1] = 0.0;
jacid[5][2] = -s1*c23*(thdot[1]);
jacid[6][1] = 0.0;
jacid[6][2] = 0.0;
jacid[6][3] = 0.0;
jacid[6][5] = c23*s4*(thdot[2]+thdot[3]) + s23*c4*thdot[4];
jacid[6][6] = -c23*c4*s5*(thdot[2]+thdot[3]) + s23*s4*s5*thdot[4]-s23*c4*c5*thdot[5]-s23*c5*(thdot[2]+thdot[3]) - s23*s5*thdot[5];

for(i=1;i<=6;i++)
printf("jacdot subr %f %f %f %f %f
",jacid[1][1],jacid[1][2],jacid[1][3],jacid[1][4],jacid[1][5],jacid[1][6]);

return;
}

void ludcmp(a,n,indx,d)
int n,*indx;
float **a,*d;
{
int i,imax,j,k;
float big,dum,sum,temp;
float *vv,*vector();
void nrerror(),free_vector();

vv=vector(1,n);
*d=1.0;
for(i=1;i<=n;i++)
{
big=0.0;
for(j=1;j<=n;j++)

if ((temp=fabs(a[i][j])) > big) big=temp;
if (big == 0.0) nrerror("Singular matrix in routine LUDCMP");
vv[i]=1.0/big;
for(j=1;j<n;j++)
for(i=1;i<=j;i++)
{
    sum=a[i][j];
    for (k=1;k<i;k++)
    sum -= a[i][k]*a[k][j];
    a[i][j]=sum;
}
big=0.0;
for(i=j;i<=n;i++)
{
    sum=a[i][j];
    for (k=1;k<j;k++)
    sum -= a[i][k]*a[k][j];
    a[i][j]=sum;
    if ( (dum=vv[i]*fabs(sum)) >= big) {
        big=dum;
        imax=i;
    }
}
if (j != imax) {
    for(k=1;k<n;k++)
    
    dum=a[imax][k];
    a[imax][k]=a[j][k];
    a[j][k]=dum;

    *d = -*d;
    vv[imax]=vv[j];
}
indx[i]=imax;
if (a[i][i] == 0.0) a[i][i]=TINY;
if(j != n) {
    dum=1.0/(a[j][j]);
    for(i=j+1;i<=n;i++) a[i][i] *= dum;
}
}
free_vector(vv,1,n);
}
void lubksb(a,n,indx,b)
float **a,b[];
int n,*indx;
{
int i,ii=0,ip,j;
float sum;
for(i=1;i<=n;i++)
{
    ip=indx[i];
    sum=b[ip];
    b[ip]=b[i];
    if (ii)
        for(j=ii;j<=i-1;j++)
            sum -= a[i][j]*b[j];
    else if (sum)
        ii=i;
    b[i]=sum;
}
} for(i=n;i>1;i--){
    sum=b[i];
    for(j=i+1;j<=n;j++) sum-= a[i][j]*b[j];
    b[i] = sum/a[i][i];
}
}

float *vector(nl,nh)
int nl,nh;
{
    void nrerror();
    float *v;
    v = (float *)malloc((unsigned)(nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
}

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void nrerror(error_text)
char error_text[];
{
    void exit();
    fprintf(stderr,"Numerical Recipies run-time error ...\n");
    fprintf(stderr,"%s\n",error_text);  
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i;
    float **m;

    m = (float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i] = (float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}
D-10 Computer Program for Graphical Simulation of PUMA 560

/* Defining axis of rotation for each link */

double lnk1_in[3],lnk1_out[3];
double lnk2_in[3],lnk2_out[3];
double lnk3_in[3],lnk3_out[3];
double lnk4_in[3],lnk4_out[3];
double lnk5_in[3],lnk5_out[3];
double lnk6_in[3],lnk6_out[3];
struct Fl_data_st formData;

/* Defining object variables for respective link id's */

GRobj link_1,link_2,link_3,link_4a,link_4b,link_4c,link_5;
GRobj link_6a,link_6b;
int number_found;
double theta[7];

int i;
double px,py,pz,qx,qy,qz,cx,cy,cz,d;
double temp_in[3],temp_out[3];
double theta_initial[7];

void zero_position()
{
  begincmd_key("EMPCyAxC");
APPENDIX D

"EMPCyAxR" "Place cylinder by axis and radius"

```c
begin cmd-key("EMPCyAxR");
    ci$put(cmd = "xy=0.0,0.0,-5.25,front");
    ci$put(cmd = "xy=0.0,0.0,-5.00,front");
    ci$put(string = "4.5");
    ci$put(response = RESET);
    ci$put(cmd = "xy=0.0,0.0,-0.50,front");
    ci$put(string = "2.00");
    ci$put(response = TERMINATE);
endcmd();
```

```
begin cmd-key("EMPCyAxR");
    ci$put(cmd = "xy=0,-1.25,0,front");
    ci$put(cmd = "xy=0.0,2.00,0.00,front");
    ci$put(string = "1.5");
    ci$put(response = TERMINATE);
endcmd();
```

```c
g$last_elements(pobj = &link_1,
     parents = 1,
     nb_wanted = 1,
     nb_read = &number_found);

if( number_found == 0 )
    write("link_1 not found.
           ");
else
    write("link_1 no. found = ",number_found," ID = ", link_1, ",\n      ");
```

```c
begin cmd-key("GRPLSgW");
    ci$put(cmd = "xy=-1.25,2.00,-1.00,front");
    ci$put(cmd = "xy=0.00,2.00,-1.00,front");
    ci$put(cmd = "xy=3.50,2.00,-0.75,front");
    ci$put(response = TERMINATE);
endcmd();
```

```
begin cmd-key("GRPArR2Pnt");
    /"GRPArR2Pnt" "Place Arc by Radius and 2 Points"
    ci$put(string = "1.000");
    ci$put(cmd = "xy=3.5,2.00,-0.75,front");
    ci$put(cmd = "xy=3.5,2.00,0.00,front");
    ci$put(cmd = "xy=3.5,2.00,0.75,front");
    ci$put(response = TERMINATE);
endcmd();
```

```c
begin cmd-key("GRPLSgW");
    ci$put(cmd = "xy=3.50,2.00,0.75,front");
```
APPENDIX D

beginmd_key("GRPArR2Pn");
// "GRPArR2Pn" "Place Arc by Radius and 2 Points"
ci$put( cmd = "xy = 0.00,2.00,1.00,front" );
ci$put( cmd = "xy = -1.25,2.00,1.00,front" );
ci$put( response = TERMINATE );
endcmd();

beginmd_key("EMPcMcCrAt");
// "EMPcMcCrAt" "auto link curves"
ci$put( string = "1.000" );
ci$put( cmd = "xy = -1.25,2.00,1.00,front" );
ci$put( cmd = "xy = -1.25,2.00,0.00,front" );
ci$put( cmd = "xy = -1.25,2.00,-1.00,front" );
ci$put( response = TERMINATE );
endcmd();

beginmd_key("EMPcPsPr");
// "EMPcPsPr" "Place solid of projection"
ci$put( cmd = "xy = -1.00,2.00,-1.00,front" );
ci$put( cmd = "xy = -1.25,2.00,-1.00,front" );
ci$put( cmd = "xy = -1.25,3.00,-1.00,front" );
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_2,
                 parents = 1,
                 nb_wanted = 1,
                 nb_read = &number_found );

if ( number_found == 0 )
  write( "link_2 not found.
else
  write( "link_2 no. found = ",number_found," ID = ", link_2, ",\n"

beginmd_key("GRPLSe");
ci$put( cmd = "xy = 3.75,2.00,0.00,front" );
ci$put( cmd = "xy = 3.50,2.00,5.00,front" );
ci$put( cmd = "xy = 2.50,2.00,5.00,front" );
ci$put( cmd = "xy = 2.25,2.00,0.00,front" );
ci$put( response = TERMINATE );
endcmd();

beginmd_key("GRPArR2Pn");
APPENDIX D

//"GRPaR2Pn" "Place Arc by Radius and 2 Points"
ci$sput(string = "0.75000");
ci$sput(cmd = "xy=2.25,2.00,0.00,front");
ci$sput(cmd = "xy=3.00,2.00,0.00,front");
ci$sput(cmd = "xy=3.75,2.00,0.00,front");
ci$sput(response = TERMINATE);
endcmd();

begin cmd_key("EMPCmCrAt");
//"EMPCmCrAt" "auto link curves"
ci$sput(string = "y");
ci$sput(cmd = "xy=3.00,2.00,5.00,front");
ci$sput(cmd = "xy=3.00,2.00,5.00,front");
ci$sput(response = RESET);
ci$sput(response = TERMINATE);
endcmd();

begin cmd_key("EMPSIPr");
//"EMPSIPr" "Place solid of projection"
ci$sput(cmd = "xy=3.625,2.00,3.50,front");
ci$sput(cmd = "xy=3.75,2.00,0.00,front");
ci$sput(cmd = "xy=3.75,1.00,0.00,front");
ci$sput(response = TERMINATE);
endcmd();

gr$last_elements( pobj = &link_3,
                   parents = 1,
                   nb_wanted = 1,
                   nb_read = &number_found);

if( number_found == 0 )
  write("link_3 not found.
M);
else
  write("link_3 no. found = ",number_found," ID = ", link_3, "]\nM);;

begin cmd_key("EMPSIBx2Pn");
ci$sput(cmd = "xy=2.5,2.0,5.00,front");
ci$sput(cmd = "xy=3.5,1.0,5.50,front");
ci$sput(response = TERMINATE);
endcmd();

gr$last_elements( pobj = &link_4a,
                   parents = 1,
                   nb_wanted = 1,
                   nb_read = &number_found);

if( number_found == 0 )
  write("link_4a not found.
M);
else
write( "link_4a no. found = ", number_found, " ID = ", link_4a, "\n");

begincmd_key("EMPCyAxR");
// "EMPCyAxR"  "Place cylinder by axis and radius"
ci$put( cmd = "xy=3.00,1.00,5.75,front");
ci$put( cmd = "xy=3.00,1.33,5.75,front");
ci$put( string = "1.00");
ci$put( response = TERMINATE");
endcmd();

gr$last_elements( pobj = &link_4b,
                  parents = 1,
                  nb_wanted = 1,
                  nb_read = &number_found");

if( number_found == 0 )
write( "link_4b not found.\n");
else
write( "link_4b no. found = ", number_found, " ID = ", link_4b, "\n");

begincmd_key("EMPCyAxR");
ci$put( cmd = "xy=3.00,1.33,5.75,front");
ci$put( cmd = "xy=3.00,1.66,5.75,front");
ci$put( string = "1.00");
ci$put( response = TERMINATE");
endcmd();

gr$last_elements( pobj = &link_5,
                  parents = 1,
                  nb_wanted = 1,
                  nb_read = &number_found");

if( number_found == 0 )
write( "link_5 not found.\n");
else
write( "link_5 no. found = ", number_found, " ID = ", link_5,"\n");

begincmd_key("EMPCyAxR");
ci$put( cmd = "xy=3.00,1.66,5.75,front");
ci$put( cmd = "xy=3.00,2.00,5.75,front");
ci$put( string = "1.00");
ci$put( response = TERMINATE");
endcmd();

gr$last_elements( pobj = &link_4c,
                  parents = 1,
                  nb_wanted = 1,
                  nb_read = &number_found");
APPENDIX D

```c
nb_read = &number_found;

if( number_found == 0 )
    write( "link_4c not found.\n");
else
    write( "link_4c no. found = ",number_found," ID= ", link_4c, "\n");

begincmd_key("EMPCyAxR");
ci$put( cmd = "xy=3.00,1.50,6.25,front");
ci$put( cmd = "xy=3.00,1.50,6.50,front");
ci$put( string = "0.25" );
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

if( number_found == 0 )
    write( "link_6a not found.\n");
else
    write( "link_6a no. found = ",number_found," ID= ", link_6a, "\n");

begincmd_key("EMPCyAxR");
ci$put( cmd = "xy=3.00,1.50,6.50,front" );
ci$put( cmd = "xy=3.00,1.50,6.75,front" );
ci$put( string = "0.50" );
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

if( number_found == 0 )
    write( "link_6b not found.\n");
else
    write( "link_6b no. found = ",number_found," ID= ", link_6b, "\n");

return;
}

main()
{
    zero_position(); /* Drawing the zero position configuration */
}
for(i=0; i <= 6; i=i+1)
  {theta[i]=0.0; theta_initial[i]=0.0;}

/* Reading the data file containing the set of joint angles i.e., θ1, θ2, ..., θ6 for each time step */

if((fptr=fopen("aprx.dat","r")) == NULL)
  {write("error opening file aprx.dat \n"); exit;}

i=0;

while(( fscanf(fptr,"%lf %lf %lf %lf %lf %lf ",&theta[1],&theta[2],
            &theta[3],&theta[4],&theta[5],&theta[6]) ) != EOF)
{

  write("data file",theta[1],theta[2],theta[3],theta[4],theta[5],theta[6],"\n");

  theta[1]=theta[1]-theta_initial[1];
  theta[3]=theta[3]-theta_initial[3];
  theta[4]=theta[4]-theta_initial[4];
  theta[5]=theta[5]-theta_initial[5];
  theta[6]=theta[6]-theta_initial[6];

  for(i=1; i < = 6; i=i+1)
    if(abs(theta[i])<1.0e-07)
      theta[i]=0.0; write("modified",theta[1],theta[2],theta[3],theta[4],theta[5],theta[6],"\n");

  lnk[1_in][0] = 0.00;
  lnk[1_in][1] = 0.00;
  lnk[1_in][2] = -5.00;

  lnk[1_out][0] = 0.00;
  lnk[1_out][1] = 0.00;
  lnk[1_out][2] = 0.50;

  lnk[2_in][0] = 0.00;
  lnk[2_in][1] = 2.00;
  lnk[2_in][2] = 0.00;

  lnk[2_out][0] = 0.00;
  lnk[2_out][1] = 3.00;
  lnk[2_out][2] = 0.00;

  lnk[3_in][0] = 3.00;
  lnk[3_in][1] = 1.00;
  lnk[3_in][2] = 0.00;
\textbf{APPENDIX D}

\begin{verbatim}

\begin{verbatim}

\text{lnk3\_out[0]} = 3.00;
\text{lnk3\_out[1]} = 2.00;
\text{lnk3\_out[2]} = 0.00;

\text{lnk4\_in[0]} = 3.00;
\text{lnk4\_in[1]} = 1.50;
\text{lnk4\_in[2]} = 5.00;

\text{lnk4\_out[0]} = 3.00;
\text{lnk4\_out[1]} = 1.50;
\text{lnk4\_out[2]} = 5.50;

\text{lnk5\_in[0]} = 3.00;
\text{lnk5\_in[1]} = 1.33;
\text{lnk5\_in[2]} = 5.75;

\text{lnk5\_out[0]} = 3.00;
\text{lnk5\_out[1]} = 1.66;
\text{lnk5\_out[2]} = 5.75;

\text{lnk6\_in[0]} = 3.00;
\text{lnk6\_in[1]} = 1.50;
\text{lnk6\_in[2]} = 6.25;

\text{lnk6\_out[0]} = 3.00;
\text{lnk6\_out[1]} = 1.50;
\text{lnk6\_out[2]} = 6.75;

\end{verbatim}

\begin{verbatim}

\{ 
    \text{begincmd\_key("GRSAn")};
    \text{ci}$\text{put ( value = theta[1] );}
    \text{endcmd();}

    \text{begincmd\_key("GRRtEAbAx");}
    \text{ci}$\text{put ( obj = link\_1 );}
    \text{ci}$\text{put ( point = lnk1\_in, window\_name = "front");}
    \text{ci}$\text{put ( point = lnk1\_out, window\_name = "front");}
    \text{ci}$\text{put ( response = TERMINATE );}
    \text{endcmd();}

    \text{gr}$\text{last\_elements( pobj = &link\_1,}
                \text{ parents = 1,}
                \text{ nb\_wanted = 1,}
                \text{ nb\_read = &number\_found );}

\end{verbatim}

\end{verbatim}

\end{verbatim}
begincmd_key("GRRtEAbAx");
ci$put( obj = link_2);
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_2,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_3);
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_3,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_4a);
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_4a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_4b);
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_4b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
APPENDIX D

```c
app$put( obj = link_4c );
capp( point = lnk1_in, window_name = "front");
capp( point = lnk1_out, window_name = "front");
capp( response = TERMINATE );
endcmd();

gapp$last_elements( pobj = &link_4c,
        parents = 1,
        nb_wanted = 1,
        nb_read = &number_found );

begincmd_key("GRRtEAbAx");
capp( obj = link_5 );
capp( point = lnk1_in, window_name = "front");
capp( point = lnk1_out, window_name = "front");
capp( response = TERMINATE );
endcmd();

gapp$last_elements( pobj = &link_5,
        parents = 1,
        nb_wanted = 1,
        nb_read = &number_found );

begincmd_key("GRRtEAbAx");
capp( obj = link_6a );
capp( point = lnk1_in, window_name = "front");
capp( point = lnk1_out, window_name = "front");
capp( response = TERMINATE );
endcmd();

gapp$last_elements( pobj = &link_6a,
        parents = 1,
        nb_wanted = 1,
        nb_read = &number_found );

begincmd_key("GRRtEAbAx");
capp( obj = link_6b );
capp( point = lnk1_in, window_name = "front");
capp( point = lnk1_out, window_name = "front");
capp( response = TERMINATE );
endcmd();

gapp$last_elements( pobj = &link_6b,
        parents = 1,
        nb_wanted = 1,
        nb_read = &number_found );
```
\[ \text{lnk2\_in}[0] = 0.00 \times \cos(\theta[1] \times \pi / 180.0) - 2.00 \times \sin(\theta[1] \times \pi / 180.0); \]
\[ \text{lnk2\_in}[1] = 0.00 \times \sin(\theta[1] \times \pi / 180.0) + 2.00 \times \cos(\theta[1] \times \pi / 180.0); \]
\[ \text{lnk2\_in}[2] = 0.00 \times 1.00; \]
\[ \text{lnk2\_out}[0] = 0.00 \times \cos(\theta[1] \times \pi / 180.0) - 3.00 \times \sin(\theta[1] \times \pi / 180.0); \]
\[ \text{lnk2\_out}[1] = 0.00 \times \sin(\theta[1] \times \pi / 180.0) + 3.00 \times \cos(\theta[1] \times \pi / 180.0); \]
\[ \text{lnk2\_out}[2] = 0.00 \times 1.00; \]

\[
\begin{align*}
\{ \\
\text{begin cmd\_key("GRSAn");} \\
\text{ci$put(value = theta[2]);} \\
\text{endcmd();} \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{begin cmd\_key("GRRtEAbAx");} \\
\text{ci$put(obj = link_2);} \\
\text{ci$put(point = lnk2\_in, window\_name = "front");} \\
\text{ci$put(point = lnk2\_out, window\_name = "front");} \\
\text{ci$put(response = TERMINATE);} \\
\text{endcmd();} \\
\text{gr$last\_elements( pobj = &link_2,} \\
\text{ parents = 1,} \\
\text{ nb\_wanted = 1,} \\
\text{ nb\_read = &number\_found ;) } \\
\end{align*}
\]

\[
\begin{align*}
\text{begin cmd\_key("GRRtEAbAx");} \\
\text{ci$put(obj = link_3);} \\
\text{ci$put(point = lnk2\_in, window\_name = "front");} \\
\text{ci$put(point = lnk2\_out, window\_name = "front");} \\
\text{ci$put(response = TERMINATE);} \\
\text{endcmd();} \\
\text{gr$last\_elements( pobj = &link_3,} \\
\text{ parents = 1,} \\
\text{ nb\_wanted = 1,} \\
\text{ nb\_read = &number\_found ;) } \\
\end{align*}
\]

\[
\begin{align*}
\text{begin cmd\_key("GRRtEAbAx");} \\
\text{ci$put(obj = link_4a);} \\
\text{ci$put(point = lnk2\_in, window\_name = "front");} \\
\text{ci$put(point = lnk2\_out, window\_name = "front");} \\
\text{ci$put(response = TERMINATE);} \\
\text{endcmd();} \\
\text{gr$last\_elements( pobj = &link_4a,} \\
\text{ parents = 1,} \\
\text{ nb\_wanted = 1,} \\
\text{ nb\_read = &number\_found ;) } \\
\end{align*}
\]
begin cmd-key("GRRtEAbAx");
ci$put( obj = link_4b );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_4b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin cmd-key("GRRtEAbAx");
ci$put( obj = link_4c );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_4c,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin cmd-key("GRRtEAbAx");
ci$put( obj = link_5 );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_5,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin cmd-key("GRRtEAbAx");
ci$put( obj = link_6a );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
gr$last_elements( pobj = &link_6a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin cmd-key("GRRtEAbAx");
ci$put( obj = link_6b );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

}

lnk3_in[0] = 3.00*cos(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0) - 1.00*sin(theta[1]*pi/180.0) + 0.00*cos(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_in[1] = 3.00*sin(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0) + 1.00*cos(theta[1]*pi/180.0) + 0.00*sin(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_in[2] = 3.00*cos(theta[1]*pi/180.0) + 0.00*1.00 - 0.00*cos(theta[2]*pi/180.0);

lnk3_out[0] = 3.00*cos(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0) - 2.00*sin(theta[1]*pi/180.0) + 0.00*cos(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_out[1] = 3.00*sin(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0) + 2.00*cos(theta[1]*pi/180.0) + 0.00*sin(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_out[2] = 3.00*sin(theta[2]*pi/180.0) + 0.00*2.00 - 0.00*cos(theta[2]*pi/180.0);

{

begincmd_key("GRSAn");
ci$put( value = theta[3] );
endcmd();

begincmd_key("GRRtEAbAx");
ci$put( obj = link_3 );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
APPENDIX D

gr$last_elements( pobj = &link_3,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
cisput( obj = link_4a );
cisput( point = lnk3_in, window_name = "front");
cisput( point = lnk3_out, window_name = "front");
cisput( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_4a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
cisput( obj = link_4b );
cisput( point = lnk3_in, window_name = "front");
cisput( point = lnk3_out, window_name = "front");
cisput( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_4b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
cisput( obj = link_4c );
cisput( point = lnk3_in, window_name = "front");
cisput( point = lnk3_out, window_name = "front");
cisput( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_4c,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
cisput( obj = link_5 );
cisput( point = lnk3_in, window_name = "front");
cisput( point = lnk3_out, window_name = "front");
cisput( response = TERMINATE );
endcmd();
APPENDIX D

\[
\begin{align*}
\text{begincmd_key("GRRtEAbAx");} \\
\text{cisput(obj = link}_6\text{a);} \\
\text{cisput(point = lnk}_3\text{, window_name = "front");} \\
\text{cisput(point = lnk}_3\text{, window_name = "front");} \\
\text{cisput(response = TERMINATE);} \\
\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
\text{begincmd_key("GRRtEAbAx");} \\
\text{cisput(obj = link}_6\text{b);} \\
\text{cisput(point = lnk}_3\text{, window_name = "front");} \\
\text{cisput(point = lnk}_3\text{, window_name = "front");} \\
\text{cisput(response = TERMINATE);} \\
\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
\text{begincmd_key("GRRtEAbAx");} \\
\text{cisput(obj = link}_6\text{b);} \\
\text{cisput(point = lnk}_3\text{, window_name = "front");} \\
\text{cisput(point = lnk}_3\text{, window_name = "front");} \\
\text{cisput(response = TERMINATE);} \\
\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
\text{lnk}_4\text{.in[0]} &= \cos(\theta[1]*\pi/180.0)^* \\
&\quad (3.00*\cos(\theta[2]*\pi/180.0)+5.00*\sin((\theta[2]+\theta[3])*\pi/180.0)) \\
&\quad -\sin(\theta[1]*\pi/180.0)*1.50; \\
\end{align*}
\]

\[
\begin{align*}
\text{lnk}_4\text{.in[1]} &= \sin(\theta[1]*\pi/180.0)^* \\
&\quad (3.00*\cos(\theta[2]*\pi/180.0)+5.00*\sin((\theta[2]+\theta[3])*\pi/180.0)) \\
&\quad +\cos(\theta[1]*\pi/180.0)*1.50; \\
\end{align*}
\]

\[
\begin{align*}
\text{lnk}_4\text{.in[2]} &= -3.00*\sin(\theta[2]*\pi/180.0)+5.00*\cos((\theta[2]+\theta[3])*\pi/180.0); \\
\end{align*}
\]

\[
\begin{align*}
\text{lnk}_4\text{.out[0]} &= \cos(\theta[1]*\pi/180.0)^* \\
&\quad (3.00*\cos(\theta[2]*\pi/180.0)+5.50*\sin((\theta[2]+\theta[3])*\pi/180.0)) \\
&\quad -\sin(\theta[1]*\pi/180.0)*1.50; \\
\end{align*}
\]

\[
\begin{align*}
\text{lnk}_4\text{.out[1]} &= \sin(\theta[1]*\pi/180.0)^* \\
&\quad (3.00*\cos(\theta[2]*\pi/180.0)+5.50*\sin((\theta[2]+\theta[3])*\pi/180.0)) \\
\end{align*}
\]
\[+ \cos(\theta[1]*\pi/180.0)*1.50;\]

\[\lnk4_{\text{out}}[2] = -3.00*\sin(\theta[2]*\pi/180.0) + 5.50*\cos((\theta[2]+\theta[3])\pi/180.0);\]

\[
\begin{align*}
&\text{GRSAn} \\
&\text{ci$\$put( value = \text{theta}[4] );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4a};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4b};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4c};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4d};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4e};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4f};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4g};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4h};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4i};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4j};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]

\[
\begin{align*}
&\text{GRRtEAbAx} \\
&\text{ci$\$put( obj = \text{link}_{4k};} \\
&\text{ci$\$put( point = \lnk4_{\text{in}}, window\_name = "front");} \\
&\text{ci$\$put( point = \lnk4_{\text{out}}, window\_name = "front");} \\
&\text{ci$\$put( response = TERMINATE );} \\
&\text{endcmd();}
\end{align*}
\]
APPENDIX D

begincmd_key("GRRtEAbAx");
ci$put( obj = link_5);
ci$put( point = lnk4_in, window_name = "front");
ci$put( point = lnk4_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_5,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_6a);
ci$put( point = lnk4_in, window_name = "front");
ci$put( point = lnk4_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_6b);
ci$put( point = lnk4_in, window_name = "front");
ci$put( point = lnk4_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

}

\[
\begin{align*}
\text{temp_in}[0] &= \cos(\theta_1)\pi/180.0) \\
& \quad \times (3.00\cos(\theta_2)\pi/180.0) + 5.75\sin((\theta_2 + \theta_3)\pi/180.0)) \\
& \quad - \sin(\theta_1)\pi/180.0) \times 1.33; \\
\text{temp_in}[1] &= \sin(\theta_1)\pi/180.0) \\
& \quad \times (3.00\cos(\theta_2)\pi/180.0) + 5.75\sin((\theta_2 + \theta_3)\pi/180.0)) \\
& \quad + \cos(\theta_1)\pi/180.0) \times 1.33; \\
\text{temp_in}[2] &= -3.00\sin(\theta_2)\pi/180.0) + 5.75\cos((\theta_2 + \theta_3)\pi/180.0); \\
\text{temp_out}[0] &= \cos(\theta_1)\pi/180.0) \\
& \quad \times (3.00\cos(\theta_2)\pi/180.0) + 5.75\sin((\theta_2 + \theta_3)\pi/180.0)) \\
& \quad - \sin(\theta_1)\pi/180.0) \times 1.66; \\n\end{align*}
\]
APPENDIX D

\[
temp\_out[1]=\sin(\theta_1\pi/180.0)\times
(3.00\cos(\theta_2\pi/180.0)+5.75\sin((\theta_2+\theta_3)\pi/180.0))
+\cos(\theta_1\pi/180.0)\times1.66;
\]

\[
temp\_out[2]=-3.00\sin(\theta_2\pi/180.0)+5.75\cos((\theta_2+\theta_3)\pi/180.0);
\]

\[
px=\cos(\theta_1\pi/180.0)\times
(3.00\cos(\theta_2\pi/180.0)+5.00\sin((\theta_2+\theta_3)\pi/180.0))
-\sin(\theta_1\pi/180.0)\times1.50;
\]

\[
py=\sin(\theta_1\pi/180.0)\times
(3.00\cos(\theta_2\pi/180.0)+5.00\sin((\theta_2+\theta_3)\pi/180.0))
+\cos(\theta_1\pi/180.0)\times1.50;
\]

\[
pz=-3.00\sin(\theta_2\pi/180.0)+5.00\cos((\theta_2+\theta_3)\pi/180.0);
\]

\[
write("px,py,pz",px,py,pz,\"\ln\")
\]

\[
qx=\cos(\theta_1\pi/180.0)\times
(3.00\cos(\theta_2\pi/180.0)+5.75\sin((\theta_2+\theta_3)\pi/180.0))
-\sin(\theta_1\pi/180.0)\times1.50;
\]

\[
qy=\sin(\theta_1\pi/180.0)\times
(3.00\cos(\theta_2\pi/180.0)+5.75\sin((\theta_2+\theta_3)\pi/180.0))
+\cos(\theta_1\pi/180.0)\times1.50;
\]

\[
qz=-3.00\sin(\theta_2\pi/180.0)+5.75\cos((\theta_2+\theta_3)\pi/180.0);
\]

\[
\text{/* Evaluating the direction cosines */}
\]

\[
cx=(qx-px)/\sqrt{(qx-px)^2+(qy-py)^2+(qz-pz)^2+qz-pz)};
\]

\[
cy=(qy-py)/\sqrt{(qx-px)^2+(qy-py)^2+(qz-pz)^2+qz-pz)}
\]

\[
cz=(qz-pz)/\sqrt{(qx-px)^2+(qy-py)^2+(qz-pz)^2+qz-pz)};
\]

\[
d=\sqrt{(cy*cy)+(cz*cz)};
\]

\[
write("cx,cy,cz,d",cx,ey,ez,d,\"\ln\")
\]

\[
write("xo",temp\_in[0],\"\ln\")
\]

\[
write("yo",temp\_in[1],\"\ln\")
\]

\[
write("zo",temp\_in[2],\"\ln\")
\]

\[
\text{/* Performing matrix concatenations for rotation about an arbitrary axis */}
\]

\[
lnk5\_in[0]=\text{temp\_in[0]}*1.0-px;
lnk5\_in[1]=\text{temp\_in[1]}*1.0-py;
lnk5\_in[2]=\text{temp\_in[2]}*1.0-pz;
\]

\[
\text{temp\_in[0]}=\text{lnk5\_in[0]}*1.0;
\]


APPENDIX D

\[ \text{temp} \_{\text{in}}[1] = \text{lnk}5\_\text{in}[1] \cdot \text{cz} / d - \text{lnk}5\_\text{in}[2] \cdot \text{cy} / d; \]
\[ \text{temp} \_{\text{in}}[2] = \text{lnk}5\_\text{in}[1] \cdot \text{cy} / d + \text{lnk}5\_\text{in}[2] \cdot \text{cz} / d; \]
\[ \text{lnk}5\_\text{in}[0] = \text{temp} \_{\text{in}}[0] \cdot d - \text{cx} \cdot \text{temp} \_{\text{in}}[2]; \]
\[ \text{lnk}5\_\text{in}[1] = \text{temp} \_{\text{in}}[1] * 1.0; \]
\[ \text{lnk}5\_\text{in}[2] = \text{temp} \_{\text{in}}[0] \cdot \text{cx} + d \cdot \text{temp} \_{\text{in}}[2]; \]
\[ \text{temp} \_{\text{in}}[0] = \cos(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}5\_\text{in}[0] \]
\[ - \sin(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}5\_\text{in}[1]; \]
\[ \text{temp} \_{\text{in}}[1] = \sin(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}5\_\text{in}[0] \]
\[ + \cos(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}5\_\text{in}[1]; \]
\[ \text{temp} \_{\text{in}}[2] = \text{lnk}5\_\text{in}[2] * 1.0; \]
\[ \text{lnk}5\_\text{in}[0] = \text{temp} \_{\text{in}}[0] \cdot d + \text{temp} \_{\text{in}}[2] \cdot \text{cx}; \]
\[ \text{lnk}5\_\text{in}[1] = \text{temp} \_{\text{in}}[1] * 1.0; \]
\[ \text{lnk}5\_\text{in}[2] = -\text{temp} \_{\text{in}}[0] \cdot \text{cx} + \text{temp} \_{\text{in}}[2] \cdot d; \]
\[ \text{temp} \_{\text{in}}[0] = \text{lnk}5\_\text{in}[0] * 1.0; \]
\[ \text{temp} \_{\text{in}}[1] = \text{lnk}5\_\text{in}[1] \cdot \text{cz} / d - \text{lnk}5\_\text{in}[2] \cdot \text{cy} / d; \]
\[ \text{temp} \_{\text{in}}[2] = -\text{lnk}5\_\text{in}[1] \cdot \text{cy} / d + \text{lnk}5\_\text{in}[2] \cdot \text{cz} / d; \]
\[ \text{lnk}5\_\text{in}[0] = \text{temp} \_{\text{in}}[0] * 1.0 + \text{px}; \]
\[ \text{lnk}5\_\text{in}[1] = \text{temp} \_{\text{in}}[1] * 1.0 + \text{py}; \]
\[ \text{lnk}5\_\text{in}[2] = \text{temp} \_{\text{in}}[2] * 1.0 + \text{pz}; \]

\[ \text{write}("x", \text{lnk}_5\_\text{in}[0], ";n"); \]
\[ \text{write}("y", \text{lnk}_5\_\text{in}[1], ";n"); \]
\[ \text{write}("z", \text{lnk}_5\_\text{in}[2], ";n"); \]

\[ \text{write}("x0", \text{temp}_\text{out}[0], ";n"); \]
\[ \text{write}("y0", \text{temp}_\text{out}[1], ";n"); \]
\[ \text{write}("z0", \text{temp}_\text{out}[2], ";n"); \]
\[ \text{lnk}_5\_\text{out}[0] = \text{temp}_\text{out}[0] * 1.0 - \text{px}; \]
\[ \text{lnk}_5\_\text{out}[1] = \text{temp}_\text{out}[1] * 1.0 - \text{py}; \]
\[ \text{lnk}_5\_\text{out}[2] = \text{temp}_\text{out}[2] * 1.0 - \text{pz}; \]
\[ \text{temp}_\text{out}[0] = \text{lnk}_5\_\text{out}[0] * 1.0; \]
\[ \text{temp}_\text{out}[1] = \text{lnk}_5\_\text{out}[1] \cdot \text{cz} / d - \text{lnk}_5\_\text{out}[2] \cdot \text{cy} / d; \]
\[ \text{temp}_\text{out}[2] = \text{lnk}_5\_\text{out}[1] \cdot \text{cy} / d + \text{lnk}_5\_\text{out}[2] \cdot \text{cz} / d; \]
\[ \text{lnk}_5\_\text{out}[0] = \text{temp}_\text{out}[0] * d - \text{cx} * \text{temp}_\text{out}[2]; \]
\[ \text{lnk}_5\_\text{out}[1] = \text{temp}_\text{out}[1] * 1.0; \]
\[ \text{lnk}_5\_\text{out}[2] = \text{temp}_\text{out}[0] * \text{cx} + d * \text{temp}_\text{out}[2]; \]
\[ \text{temp}_\text{out}[0] = \cos(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}_5\_\text{out}[0] \]
\[ - \sin(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}_5\_\text{out}[1]; \]
\[ \text{temp}_\text{out}[1] = \sin(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}_5\_\text{out}[0] \]
\[ + \cos(\text{theta}[4] \cdot \text{pi} / 180.0) \cdot \text{lnk}_5\_\text{out}[1]; \]
\[ \text{temp}_\text{out}[2] = \text{lnk}_5\_\text{out}[2] * 1.0; \]
APPENDIX D

lnk5\_out[0]=\text{temp\_out[0]}*d+\text{temp\_out[2]}*ex;
lnk5\_out[1]=\text{temp\_out[1]}*1.0;
lnk5\_out[2]=\text{temp\_out[0]}*ex+\text{temp\_out[2]}*d;

temp\_out[0]=lnk5\_out[0]*1.0;
temp\_out[1]=lnk5\_out[1]*ez/d+lnk5\_out[2]*cy/d;
temp\_out[2]=-lnk5\_out[1]*cy/d+lnk5\_out[2]*ez/d;

lnk5\_out[0]=\text{temp\_out[0]}*1.0+px;
lnk5\_out[1]=\text{temp\_out[1]}*1.0+py;
lnk5\_out[2]=\text{temp\_out[2]}*1.0+pz;

write("x",lnk5\_out[0],"in");
write("y",lnk5\_out[1],"in");
write("z",lnk5\_out[2],"in");

{}

\text{begin\:cmd\:key("GRSAn")};
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{value} = \text{theta}[5]);
\text{end\:cmd}();

\text{begin\:cmd\:key("GRRtEAbAx")};
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{obj} = \text{link\_5});
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{point} = \text{lnk5\_in}, \text{window\_name} = "front");
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{point} = \text{lnk5\_out}, \text{window\_name} = "front");
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{response} = \text{TERMINATE});
\text{end\:cmd}();

\text{gr$\!\!\!\!\!\!\!\!\!$last\_elements(} \text{pobj} = \&\text{link\_5},
  \text{parents} = 1,
  \text{nb\_wanted} = 1,
  \text{nb\_read} = \&\text{number\_found});

\text{begin\:cmd\:key("GRRtEAbAx")};
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{obj} = \text{link\_6a});
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{point} = \text{lnk5\_in}, \text{window\_name} = "front");
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{point} = \text{lnk5\_out}, \text{window\_name} = "front");
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{response} = \text{TERMINATE});
\text{end\:cmd}();

\text{gr$\!\!\!\!\!\!\!\!\!$last\_elements(} \text{pobj} = \&\text{link\_6a},
  \text{parents} = 1,
  \text{nb\_wanted} = 1,
  \text{nb\_read} = \&\text{number\_found});

\text{begin\:cmd\:key("GRRtEAbAx")};
ci$\!\!\!\!\!\!\!\!\!\!\!\!\!\!$put(\text{obj} = \text{link\_6b});
cisput( point = lnk5_in, window_name = "front");  
cisput( point = lnk5_out, window_name = "front");  
cisput( response = TERMINATE );  
endcmd();  
gr$last_elements( pobj = &link_6b,  
    parents = 1,  
    nb_wanted = 1,  
    nb_read = &number_found );  

px=lnk5_in[0];  
py=lnk5_in[1];  
pz=lnk5_in[2];  
qx=lnk5_out[0];  
qy=lnk5_out[1];  
qz=lnk5_out[2];  
write("px,py,pz",px,py,pz,\"in\");  
/* Evaluating the direction cosines */  
  cx = (qx-px)/sqrt((qx-px)*(qx-px) + (qy-py)*(qy-py) + (qz-pz)*(qz-pz));  
  cy = (qy-py)/sqrt((qx-px)*(qx-px) + (qy-py)*(qy-py) + (qz-pz)*(qz-pz));  
  cz = (qz-pz)/sqrt((qx-px)*(qx-px) + (qy-py)*(qy-py) + (qz-pz)*(qz-pz));  
  d = sqrt((cy*cy) + (cz*cz));  
write("cx,cy,cz,d",cx,cy,cz,d,\"in\");  

   temp_in[0]=cos(theta[1]*pi/180.0)*  
   (3.00*cos(theta[2]*pi/180.0) + 6.25*sin((theta[2]+theta[3])-pi/180.0))  
   -sin(theta[1]*pi/180.0)*1.50;  
   temp_in[1]=sin(theta[1]*pi/180.0)*  
   (3.00*cos(theta[2]*pi/180.0) + 6.25*sin((theta[2]+theta[3])-pi/180.0))  
   +cos(theta[1]*pi/180.0)*1.50;  
   temp_in[2]=-3.00*sin(theta[2]*pi/180.0) + 6.25*cos((theta[2]+theta[3])*pi/180.0);  

   temp_out[0]=cos(theta[1]*pi/180.0)*  
   (3.00*cos(theta[2]*pi/180.0) + 6.75*sin((theta[2]+theta[3])*pi/180.0))  
   -sin(theta[1]*pi/180.0)*1.50;  
   temp_out[1]=sin(theta[1]*pi/180.0)*
APPENDIX D

\[
(3.00 \cdot \cos(\theta_2 \cdot \pi / 180.0) + 6.75 \cdot \sin((\theta_2 + \theta_3) \cdot \pi / 180.0)) + \cos(\theta_1 \cdot \pi / 180.0) \cdot 1.50;
\]

\[
tem_{\text{p-out2}} = -3.00 \cdot \sin(\theta_2 \cdot \pi / 180.0) + 6.75 \cdot \cos((\theta_2 + \theta_3) \cdot \pi / 180.0);
\]

write("**** data for link_6 *****", "\n");

write("xo", temp_in[0], "\n");
write("yo", temp_in[1], "\n");
write("zo", temp_in[2], "\n");

/* Performing matrix concatenations for rotation about an arbitrary axis */

lnk6_in[0] = temp_in[0] \cdot 1.0 - px;
lnk6_in[1] = temp_in[1] \cdot 1.0 - py;

\[
tem_{\text{p-in0}} = lnk6_{\text{in0}} \cdot 1.0;
\]

\[
tem_{\text{p-in1}} = lnk6_{\text{in1}} \cdot cz/d - lnk6_{\text{in2}} \cdot cy/d;
\]

\[
tem_{\text{p-in2}} = lnk6_{\text{in1}} \cdot cy/d + lnk6_{\text{in2}} \cdot cz/d;
\]

lnk6_in[0] = temp_in[0] \cdot d - cx * temp_in[2];
lnk6_in[1] = temp_in[1] \cdot 1.0;
lnk6_in[2] = temp_in[0] \cdot cx + d * temp_in[2];

\[
tem_{\text{p-in0}} = \cos(\theta_5 \cdot \pi / 180.0) \cdot lnk6_{\text{in0}}
\]

\[
- \sin(\theta_5 \cdot \pi / 180.0) \cdot lnk6_{\text{in1}};
\]

\[
tem_{\text{p-in1}} = \sin(\theta_5 \cdot \pi / 180.0) \cdot lnk6_{\text{in0}}
\]

\[
+ \cos(\theta_5 \cdot \pi / 180.0) \cdot lnk6_{\text{in1}};
\]

\[
tem_{\text{p-in2}} = lnk6_{\text{in2}} \cdot 1.0;
\]

lnk6_in[0] = temp_in[0] \cdot d + temp_in[2] \cdot cx;
lnk6_in[1] = temp_in[1] \cdot 1.0;

\[
tem_{\text{p-in0}} = lnk6_{\text{in0}} \cdot 1.0;
\]

\[
tem_{\text{p-in1}} = lnk6_{\text{in1}} \cdot cz/d + lnk6_{\text{in2}} \cdot cy/d;
\]

\[
tem_{\text{p-in2}} = -lnk6_{\text{in1}} \cdot cy/d + lnk6_{\text{in2}} \cdot cz/d;
\]

lnk6_in[0] = temp_in[0] \cdot 1.0 + px;
lnk6_in[1] = temp_in[1] \cdot 1.0 + py;

write("x", lnk6_in[0], "\n");
write("y", lnk6_in[1], "\n");
write("z", lnk6_in[2], "\n");

write("xo", temp_out[0], "\n");
write("yo", temp_out[1], "\n");
write("zo", temp_out[2], "\n");
APPENDIX D

\begin{verbatim}
lnk6_out[0] = temp_out[0] * 1.0 - px;

temp_out[0] = lnk6_out[0] * 1.0;

lnk6_out[0] = temp_out[0] * d - cx * temp_out[2];
lnk6_out[1] = temp_out[1] * 1.0;
lnk6_out[2] = temp_out[0] * cx + d * temp_out[2];

temp_out[0] = cos(theta[5] * pi/180.0) * lnk6_out[0]
- sin(theta[5] * pi/180.0) * lnk6_out[1];
temp_out[1] = sin(theta[5] * pi/180.0) * lnk6_out[0]
+ cos(theta[5] * pi/180.0) * lnk6_out[1];

lnk6_out[0] = temp_out[0] * d + temp_out[2] * cx;
lnk6_out[1] = temp_out[1] * 1.0;

temp_out[0] = lnk6_out[0] * 1.0;

lnk6_out[0] = temp_out[0] * 1.0 + px;
lnk6_out[1] = temp_out[1] * 1.0 + py;

write("x", lnk6_out[0], "\n");
write("y", lnk6_out[1], "\n");
write("z", lnk6_out[2], "\n");

{
begincmd_key("GRSAn");
ci$put( value = theta[6] );
endcmd();

begincmd_key("GRRtEAbAx");
ci$put( obj = link_6a );
ci$put( point = lnk6_in, window_name = "front");
ci$put( point = lnk6_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

begincmd_key("GRRtEAbAx");
ci$put( obj = link_6a );
\end{verbatim}
APPENDIX D

\begin{verbatim}
ci$put( point = lnk6_in, window_name = "front");
ci$put( point = lnk6_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6a,
                   parents = 1,
                   nb_wanted = 1,
                   nb_read = &number_found );

begincmd_key("GRRtEAbAxA");
ci$put( obj = link_6b );
ci$put( point = lnk6_in, window_name = "front");
ci$put( point = lnk6_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6b,
                   parents = 1,
                   nb_wanted = 1,
                   nb_read = &number_found );
}

theta_initial[1]=theta[1];
theta_initial[2]=theta[2];
theta_initial[3]=theta[3];
theta_initial[4]=theta[4];
theta_initial[5]=theta[5];
theta_initial[6]=theta[6];

write("theta_initial",theta_initial[1],theta_initial[2],theta_initial[3],theta_initial[4],theta_initial[5],theta_initial[6],"\n");

i=i+1;}//END WHILE

close(fptr);
\end{verbatim}
D-11 Computer Program for Graphical Simulation of Pieper’s Solution

```c
#include "ciminimum.h"
#include "cimacros.hm"
#include "fi.h"
#include "grlastele.h"
#include "stdio.h"

#define pi (3.14157)

extern fscanf();

FILE *fptr;

double lnk1_in[3],lnk1_out[3];
double lnk2_in[3],lnk2_out[3];
double lnk3_in[3],lnk3_out[3];
double lnk4_in[3],lnk4_out[3];
double lnk5_in[3],lnk5_out[3];
double lnk6_in[3],lnk6_out[3];
struct FI_data_st formData;

GRobj link_1,link_2,link_3,link_4a,link_4b,link_4c,link_5;
GRobj link_6a,link_6b;
int number_found;
double theta[7];

void zero_position()
{
    beginmd_key(“EMPCyAxR”);
}```
APPENDIX D

_APPENDIX D_

IEMPCyAxR" Place cylinder by axis and radius"

ci$put( cmd = "xy=0.0,0.0,-5.25,front");
ci$put( cmd = "xy=0.0,0.0,-5.00,front");
ci$put(string = "4.5");
ci$put( response = RESET");
ci$put( cmd = "xy=0.0,0.0,-5.00,front");
ci$put( cmd = "xy=0.0,0.0,-0.50,front");
ci$put(string = "2.00");
ci$put( response = TERMINATE");
endcmd();

beginmd_key("EMPCyAxR");
// "EMPCyAxR" Place cylinder by axis and radius"
ci$put( cmd = "xy=0.0,1.25,0,front");
ci$put( cmd = "xy=0.0,1.25,0.00,front");
ci$put(string = "1.5");
ci$put( response = TERMINATE");
endcmd();

gr$last_elements( pobj = &link_1,
   parents = 1,
   nb_wanted = 1,
   nb_read = &number_found );

if( number_found == 0 )
write( "link_1 not found.\n");
else
write( "link_1 no. found = ",number_found," ID = ", link_1, "\n");

beginmd_key("GRPLSg");
ci$put( cmd = "xy=3.50,1.25,0.75,front");
endcmd();

beginmd_key("GRPArR2Pn");
// "GRPArR2Pn" Place Arc by Radius and 2 Points"
ci$put(string = "1.000");
ci$put( cmd = "xy=3.5,1.25,-0.75,front");
ci$put( cmd = "xy=3.5,1.25,0.00,front");
ci$put( cmd = "xy=3.5,1.25,0.75,front");
ci$put( response = TERMINATE");
endcmd();

beginmd_key("GRPLSg");
ci$put( cmd = "xy=3.50,1.25,0.75,front");
APPENDIX D

cisput(cfunc = "xy = 0.00,1.25,1.00,front");
cisput(cfunc = "xy = -1.25,1.25,1.00,front");
cisput(response = TERMINATE);
endcmd();

begincmd_key("GRPArR2Pn");
// "GRPArR2Pn"  "Place Arc by Radius and 2 Points"
cisput(string = "1.000");
cisput(cfunc = "xy = -1.25,1.25,1.00,front");
cisput(cfunc = "xy = -1.25,1.25,0.00,front");
cisput(cfunc = "xy = -1.25,1.25,-1.00,front");
cisput(response = TERMINATE);
endcmd();

begincmd_key("EMPCmCrAt");
// "EMPCmCrAt"    "auto link curves"
cisput(string = "y");
cisput(cfunc = "xy = -1.00,1.25,-1.00,front");
cisput(cfunc = "xy = -1.00,1.25,-1.00,front");
cisput(response = RESET);
cisput(response = TERMINATE);
endcmd();

begincmd_key("EMPSIPr");  // "EMPSIPr"    "Place solid of projection"
cisput(cfunc = "xy = -1.00,1.25,-1.00,front");
cisput(cfunc = "xy = -1.25,1.25,-1.00,front");
cisput(cfunc = "xy = -1.25,2.25,-1.00,front");
cisput(response = TERMINATE);
endcmd();

g$last_elements(pobj = &link_2,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found);

if(number_found == 0)
write("link_2 not found.
");
else
write("link_2 no. found = ",number_found," ID= ", link_2," ");

begincmd_key("GRPLSg");
cisput(cfunc = "xy = 3.75,1.25,0.00,front");
cisput(cfunc = "xy = 3.50,1.25,2.25,front");
cisput(cfunc = "xy = 2.50,1.25,2.25,front");
cisput(cfunc = "xy = 2.25,1.25,0.00,front");
cisput(response = TERMINATE);
endcmd();

begincmd_key("GRPArR2Pn");
APPENDIX D

"GRPAr2Pn" "Place Arc by Radius and 2 Points"
ci$put(string = "0.75000");
ci$put(cmd = "xy=2.25,1.25,0.00,front");
ci$put(cmd = "xy=3.00,1.25,0.00,front");
ci$put(cmd = "xy=3.75,1.25,0.00,front");
ci$put(response = TERMINATE);
endcmd();

begin_cmd_key("EMPCmCrAt");
"EMPCmCrAt" "auto link curves"
ci$put(string = "y");
ci$put(cmd = "xy=3.00,1.25,2.25,front");
ci$put(cmd = "xy=3.00,1.25,2.25,frontU");
ci$put(response = RESET);
ci$put(response = TERMINATE);
endcmd();

begin_cmd_key("EMPSIPr");
"EMPSIPr" "Place solid of projection"
ci$put(cmd = "xy=3.625,1.25,1.125,front");
ci$put(cmd = "xy=3.75,1.25,0.00,front");
ci$put(cmd = "xy=3.75,0.25,0.00,front");
ci$put(response = TERMINATE);
endcmd();

g$last_elements( pobj = &link_3,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );
if( number_found == 0 )
    write( "link_3 not found.
    else
    write( "link_3 no. found = ",number_found," ID= ", link_3, ",n");

begin_cmd_key("EMPSIBx2Pn");
ci$put(cmd = "xy=2.5,1.25,2.25,front");
ci$put(cmd = "xy=3.5,0.25,2.75,front");
ci$put(response = TERMINATE);
endcmd();

g$last_elements( pobj = &link_4a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );
if( number_found == 0 )
    write( "link_4a not found.
    else
APPENDIX D

begin cmd_key("EMPCyAxR");
/*EMPCyAxR* "Place cylinder by axis and radius"
ci$put( cmd = "xy=3.00,0.25,3.00,front");
ci$put( cmd = "xy=3.00,0.58,3.00,front");
ci$put( string = "1.00");
ci$put( response = TERMINATE );
endcmd();

gr$:last_elements( pobj = &link_4b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

if( number_found == 0 )
    write("link_4b not found.
");
else
    write("link_4b no. found = ",number_found," ID = ",link_4b,"\n");

begin cmd_key("EMPCyAxR");
ci$put( cmd = "xy=3.00,0.58,3.00,front");
ci$put( cmd = "xy=3.00,0.91,3.00,front");
ci$put( string = "1.00");
ci$put( response = TERMINATE );
endcmd();

gr$:last_elements( pobj = &link_5,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

if( number_found == 0 )
    write("link_5 not found.\n");
else
    write("link_5 no. found = ",number_found," ID = ",link_5,"\n");

begin cmd_key("EMPCyAxR");
ci$put( cmd = "xy=3.00,0.91,3.00,front");
ci$put( cmd = "xy=3.00,1.25,3.00,front");
ci$put( string = "1.00");
ci$put( response = TERMINATE );
endcmd();

gr$:last_elements( pobj = &link_4c,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );
APPENDIX D

```
nb_read = &number_found;

if( number_found == 0 )
    write( "link_4c not found.\n");
else
    write( "link_4c no. found = ",number_found," ID= ", link_4c, "\n");

begincmd_key("EMPCyAxR");
cisput( cmd = "xy=3.00,0.75,3.50,front");
cisput( cmd = "xy=3.00,0.75,3.75,front");
cisput(string = "0.25");
cisput( response = TERMINATE );
endcmd();
g$rlast_elements( pobj = &link_6a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

if( number_found == 0 )
    write( "link_6a not found.\n");
else
    write( "link_6a no. found = ",number_found," ID= ", link_6a, "\n");

begincmd_key("EMPCyAxR");
cisput( cmd = "xy=3.00,0.75,3.50,front");
cisput( cmd = "xy=3.00,0.75,4.00,front");
cisput(string = "0.50");
cisput( response = TERMINATE );
endcmd();
g$rlast_elements( pobj = &link_6b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

if( number_found == 0 )
    write( "link_6b not found.\n");
else
    write( "link_6b no. found = ",number_found," ID= ", link_6b, "\n");
return;
}
```

main()
{
    zero_position(); /* Drawing the zero position configuration */
for( i=0; i <= 6; i=i+1 )
{theta[i]=0.0;
 theta_initial[i]=0.0;}

/* Reading the data file containing a set of 4 set of joint angles i.e., θ1, θ2, ..., θ6 from Pieper's Solution */

if((fptr=fopen("pieper.dat","r")) == NULL)
{write("error opening file pieper.dat \n");
 exit;}

i=0;

while(( fscanf(fptr,"%lf %lf %lf %lf %lf %lf \n",&theb[1],&theta[2],
        &theta[3],&theta[4],&theta[5],&theta[6]) != EOF))
{

write("datafile",theta[1],theta[2],theta[3],theta[4],theta[5],theta[6],"\n");

theta[1]=theta[1]-theta_initial[1];
theta[3]=theta[3]-theta_initial[3];
theta[4]=theta[4]-theta_initial[4];
theta[5]=theta[5]-theta_initial[5];
theta[6]=theta[6]-theta_initial[6];

for(i=1;i <= 6;i=i+1)
if(abs(theta[i])<1.0e-07)
theta[i]=0.0;

write("modified",theta[1],theta[2],theta[3],theta[4],theta[5],theta[6],"\n");

lnk1_in[0]= 0.00;
lnk1_in[1]= 0.00;
lnk1_in[2]= -5.00;

lnk1_out[0]= 0.00;
lnk1_out[1]= 0.00;
lnk1_out[2]= -0.50;

lnk2_in[0]= 0.00;
lnk2_in[1]= 1.25;
lnk2_in[2]= 0.00;

lnk2_out[0]= 0.00;
lnk2_out[1]= 2.25;
lnk2_out[2]= 0.00;

lnk3_in[0]= 3.00;
lnk3_in[1]= 0.25;
lnk3_in[2]= 0.00;
\textbf{APPENDIX D}

\begin{verbatim}
lnk3_out[0] = 3.00;
lnk3_out[1] = 1.25;
lnk3_out[2] = 0.00;

lnk4_in[0] = 3.00;
lnk4_in[1] = 0.75;
lnk4_in[2] = 2.25;

lnk4_out[0] = 3.00;
lnk4_out[1] = 0.75;
lnk4_out[2] = 2.75;

lnk5_in[0] = 3.00;
lnk5_in[1] = 0.58;
lnk5_in[2] = 3.00;

lnk5_out[0] = 3.00;
lnk5_out[1] = 0.91;
lnk5_out[2] = 3.00;

lnk6_in[0] = 3.00;
lnk6_in[1] = 0.75;
lnk6_in[2] = 3.50;

lnk6_out[0] = 3.00;
lnk6_out[1] = 0.75;
lnk6_out[2] = 4.00;

{
    beincmd_key("GRSAn");
    ci$put( value = theta[1] );
    endcmd();

    beincmd_key("GRRtEAbAx");
    ci$put( obj = link_1 );
    ci$put( point = lnk1_in, window_name = "front");
    ci$put( point = lnk1_out, window_name = "front");
    ci$put( response = TERMINATE );
    endcmd();

    gr$last_elements( pobj = &link_1, parents = 1, nb_wanted = 1, nb_read = &number_found );
\end{verbatim}
APPENDIX D

begincmd_key("GRRtEAbAx");
ci$put( obj = link_2 );
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

g$last_elements( pobj = &link_2,
   parents = 1,
   nb_wanted = 1,
   nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_3 );
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

g$last_elements( pobj = &link_3,
   parents = 1,
   nb_wanted = 1,
   nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_4a );
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

g$last_elements( pobj = &link_4a,
   parents = 1,
   nb_wanted = 1,
   nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_4b );
ci$put( point = lnk1_in, window_name = "front");
ci$put( point = lnk1_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

g$last_elements( pobj = &link_4b,
   parents = 1,
   nb_wanted = 1,
   nb_read = &number_found );

begincmd_key("GRRtEAbAx");
Appendix D

```
ci$put(obj = link_4c);
ci$put(point = lnkl_in, window_name = "front");
ci$put(point = lnkl_out, window_name = "front");
ci$put(response = TERMINATE);
endcmd();

g$last_elements(pobj = &link_4c,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found);

begincmd_key("GRRtEAbAx");
ci$put(obj = link_5);
ci$put(point = lnkl_in, window_name = "front");
ci$put(point = lnkl_out, window_name = "front");
ci$put(response = TERMINATE);
endcmd();

g$last_elements(pobj = &link_5,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found);

begincmd_key("GRRtEAbAx");
ci$put(obj = link_6a);
ci$put(point = lnkl_in, window_name = "front");
ci$put(point = lnkl_out, window_name = "front");
ci$put(response = TERMINATE);
endcmd();

g$last_elements(pobj = &link_6a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found);

begincmd_key("GRRtEAbAx");
ci$put(obj = link_6b);
ci$put(point = lnkl_in, window_name = "front");
ci$put(point = lnkl_out, window_name = "front");
ci$put(response = TERMINATE);
endcmd();

g$last_elements(pobj = &link_6b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found);
```
\texttt{APPENDIX D}

\begin{verbatim}
\texttt{
\textbf{link2\_in}[0]=0.00*\cos(\theta[1]*\pi/180.0) - 1.25*\sin(\theta[1]*\pi/180.0);
\texttt{link2\_in}[1]=0.00*\sin(\theta[1]*\pi/180.0) + 1.25*\cos(\theta[1]*\pi/180.0);
\texttt{link2\_in}[2]=0.00*1.00;

\texttt{link2\_out}[0]=0.00*\cos(\theta[1]*\pi/180.0) - 2.25*\sin(\theta[1]*\pi/180.0);
\texttt{link2\_out}[1]=0.00*\sin(\theta[1]*\pi/180.0) + 2.25*\cos(\theta[1]*\pi/180.0);
\texttt{link2\_out}[2]=0.00*1.00;

\{
  \begin{verbatim}
  \texttt{beginmd_key("GRSAn");
  ci$put( value = \theta[2] );
  \texttt{endcmd();

  beginmd_key("GRRtEAbAx");
  ci$put( obj = \texttt{link\_2} );
  ci$put( point = \texttt{lnk2\_in}, window\_name = "front");
  ci$put( point = \texttt{lnk2\_out}, window\_name = "front");
  ci$put( response = \texttt{TERMINATE} );
  \texttt{endcmd();

  gr$last_elements( pobj = &\texttt{link\_2},
    parents = 1,
    nb\_wanted = 1,
    nb\_read = \&number\_found );

  beginmd_key("GRRtEAbAx");
  ci$put( obj = \texttt{link\_3} );
  ci$put( point = \texttt{lnk2\_in}, window\_name = "front");
  ci$put( point = \texttt{lnk2\_out}, window\_name = "front");
  ci$put( response = \texttt{TERMINATE} );
  \texttt{endcmd();

  gr$last_elements( pobj = &\texttt{link\_3},
    parents = 1,
    nb\_wanted = 1,
    nb\_read = \&number\_found );

  beginmd_key("GRRtEAbAx");
  ci$put( obj = \texttt{link\_4a} );
  ci$put( point = \texttt{lnk2\_in}, window\_name = "front");
  ci$put( point = \texttt{lnk2\_out}, window\_name = "front");
  ci$put( response = \texttt{TERMINATE} );
  \texttt{endcmd();

  gr$last_elements( pobj = &\texttt{link\_4a},
    parents = 1,
    nb\_wanted = 1,
    nb\_read = \&number\_found );
\end{verbatim}

\end{verbatim}
}
\end{verbatim}
begin_cmd_key("GRRt5AbAx");
ci$put( obj = link_4b );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = 'TERMINATE' );
endmd();
gr$last_elements( pobj = &link_4b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin_cmd_key("GRRt5AbAx");
ci$put( obj = link_4c );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = 'TERMINATE' );
endmd();
gr$last_elements( pobj = &link_4c,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin_cmd_key("GRRt5AbAx");
ci$put( obj = link_5 );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = 'TERMINATE' );
endmd();
gr$last_elements( pobj = &link_5,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin_cmd_key("GRRt5AbAx");
ci$put( obj = link_6a );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = 'TERMINATE' );
endmd();
gr$last_elements( pobj = &link_6a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begin_cmd_key("GRRt5AbAx");
APPENDIX D

```c
appendix( obj = link_6b );
ci$put( point = lnk2_in, window_name = "front");
ci$put( point = lnk2_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

g$rlast_elements( pobj = &link_6b,
parents = 1,
 nb_wanted = 1,
 nb_read = &number_found );

}

lnk3_in[0]=3.00*cos(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0)
-0.25*sin(theta[1]*pi/180.0)
+0.00*cos(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_in[1]=3.00*sin(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0)
+0.25*cos(theta[1]*pi/180.0)
+0.00*sin(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_in[2]=3.00*sin(theta[2]*pi/180.0) + 0.00*0.25
-0.00*cos(theta[2]*pi/180.0);

lnk3_out[0]=3.00*cos(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0)
-1.25*sin(theta[1]*pi/180.0)
+0.00*cos(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_out[1]=3.00*sin(theta[1]*pi/180.0)*cos(theta[2]*pi/180.0)
+1.25*cos(theta[1]*pi/180.0)
+0.00*sin(theta[1]*pi/180.0)*sin(theta[2]*pi/180.0);

lnk3_out[2]=3.00*-sin(theta[2]*pi/180.0) + 0.00*1.25
-0.00*cos(theta[2]*pi/180.0);

{
begincmd_key("GRSAn");
ci$put( value = theta[3] );
endcmd();

begincmd_key("GRRtEAbAx");
ci$put( obj = link_3 );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
```
APPENDIX D

gr$last_elements( pobj = &link_3,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_4a );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_4a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_4b );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_4b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_4c );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_4c,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_5 );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();
APPENDIX D

```
gr$last_elements( pobj = &link_5,  
  parents = 1,  
  nb_wanted = 1,  
  nb_read = &number_found );

beginmd_key("GRRtEAbAx");
ci$put( obj = link_6a );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6a,  
  parents = 1,  
  nb_wanted = 1,  
  nb_read = &number_found );

beginmd_key("GRRtEAbAx");
ci$put( obj = link_6b );
ci$put( point = lnk3_in, window_name = "front");
ci$put( point = lnk3_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6b,  
  parents = 1,  
  nb_wanted = 1,  
  nb_read = &number_found );
```

```
APPENDIX D

\[ + \cos(\theta[1] \cdot \pi / 180.0) \cdot 0.75; \]

\[ \text{lnk4\_out}[2] = -3.00 \cdot \sin(\theta[2] \cdot \pi / 180.0) + 2.75 \cdot \cos((\theta[2] + \theta[3]) \cdot \pi / 180.0); \]

\{
begin\text{cmd\_key}("GRSAn");
ci$put( \text{value} = \theta[4] );
end\text{cmd}();

\begin\text{cmd\_key}("GRRtEAbAx");
ci$put( \text{obj} = \text{lnk\_4a} );
ci$put( \text{point} = \text{lnk4\_in}, \text{window\_name} = \text{"front"});
ci$put( \text{point} = \text{lnk4\_out}, \text{window\_name} = \text{"front"});
ci$put( \text{response} = \text{TERMINATE} );
end\text{cmd}();

\text{gr$last\_elements}( \text{pobj} = &\text{lnk\_4a},
\begin{align*}
\text{parents} & = 1, \\
\text{nb\_wanted} & = 1, \\
\text{nb\_read} & = &\text{number\_found} \;
\end{align*}
);

\begin\text{cmd\_key}("GRRtEAbAx");
ci$put( \text{obj} = \text{lnk\_4b} );
ci$put( \text{point} = \text{lnk4\_in}, \text{window\_name} = \text{"front"});
ci$put( \text{point} = \text{lnk4\_out}, \text{window\_name} = \text{"front"});
ci$put( \text{response} = \text{TERMINATE} );
end\text{cmd}();

\text{gr$last\_elements}( \text{pobj} = &\text{lnk\_4b},
\begin{align*}
\text{parents} & = 1, \\
\text{nb\_wanted} & = 1, \\
\text{nb\_read} & = &\text{number\_found} \;
\end{align*}
);

\begin\text{cmd\_key}("GRRtEAbAx");
ci$put( \text{obj} = \text{lnk\_4c} );
ci$put( \text{point} = \text{lnk4\_in}, \text{window\_name} = \text{"front"});
ci$put( \text{point} = \text{lnk4\_out}, \text{window\_name} = \text{"front"});
ci$put( \text{response} = \text{TERMINATE} );
end\text{cmd}();

\text{gr$last\_elements}( \text{pobj} = &\text{lnk\_4c},
\begin{align*}
\text{parents} & = 1, \\
\text{nb\_wanted} & = 1, \\
\text{nb\_read} & = &\text{number\_found} \;
\end{align*}
);
APPENDIX D

```
begincmd_key("GRRtEAbAx");
ci$put( obj = link_5);
ci$put( point = lnk4_in, window_name = "front");
ci$put( point = lnk4_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_5,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_6a);
ci$put( point = lnk4_in, window_name = "front");
ci$put( point = lnk4_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6a,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

begincmd_key("GRRtEAbAx");
ci$put( obj = link_6b);
ci$put( point = lnk4_in, window_name = "front");
ci$put( point = lnk4_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );
}

temp_in[0] = cos(theta[1]*pi/180.0)*
(3.00*cos(theta[2]*pi/180.0) + 3.00*sin((theta[2]+theta[3])*pi/180.0))
-sin(theta[1]*pi/180.0)*0.58;

temp_in[1] = sin(theta[1]*pi/180.0)*
(3.00*cos(theta[2]*pi/180.0) + 3.00*sin((theta[2]+theta[3])*pi/180.0))
+ cos(theta[1]*pi/180.0)*0.58;

temp_in[2] = -3.00*sin(theta[2]*pi/180.0) + 3.00*cos((theta[2]+theta[3])*pi/180.0);

temp_out[0] = cos(theta[1]*pi/180.0)*
(3.00*cos(theta[2]*pi/180.0) + 3.00*sin((theta[2]+theta[3])*pi/180.0))
-sin(theta[1]*pi/180.0)*0.91;
```
temp_out[1] = sin(theta[1]*pi/180.0)*
    (3.00*cos(theta[2]*pi/180.0) + 3.00*sin((theta[2]+theta[3])*pi/180.0))
    + cos(theta[1]*pi/180.0)*0.91;

temp_out[2] = -3.00*sin(theta[2]*pi/180.0) + 3.00*cos((theta[2]+theta[3])*pi/180.0);

px = cos(theta[1]*pi/180.0)*
    (3.00*cos(theta[2]*pi/180.0) + 2.25*sin((theta[2]+theta[3])*pi/180.0))
    -sin(theta[1]*pi/180.0)*0.75;

py = sin(theta[1]*pi/180.0)*
    (3.00*cos(theta[2]*pi/180.0) + 2.25*sin((theta[2]+theta[3])*pi/180.0))
    + cos(theta[1]*pi/180.0)*0.75;

pz = -3.00*sin(theta[2]*pi/180.0) + 2.25*cos((theta[2]+theta[3])*pi/180.0);

write("px,py,pz",px,py,pz,"\n");

qx = cos(theta[1]*pi/180.0)*
    (3.00*cos(theta[2]*pi/180.0) + 3.00*sin((theta[2]+theta[3])*pi/180.0))
    -sin(theta[1]*pi/180.0)*0.75;

qy = sin(theta[1]*pi/180.0)*
    (3.00*cos(theta[2]*pi/180.0) + 3.00*sin((theta[2]+theta[3])*pi/180.0))
    + cos(theta[1]*pi/180.0)*0.75;

qz = -3.00*sin(theta[2]*pi/180.0) + 3.00*cos((theta[2]+theta[3])*pi/180.0);

/* Evaluating the direction cosines */

cx = (qx-px)/sqrt((qx-px)*(qx-px)+(qy-py)*(qy-py)+(qz-pz)*(qz-pz));
cy = (qy-py)/sqrt((qx-px)*(qx-px)+(qy-py)*(qy-py)+(qz-pz)*(qz-pz));
cz = (qz-pz)/sqrt((qx-px)*(qx-px)+(qy-py)*(qy-py)+(qz-pz)*(qz-pz));

d = sqrt((cy*cy)+(cz*cz));

write("cx,cy,cz,d",cx, cy, cz, d,"\n");

write("xo",temp_in[0],"\n");
write("yo",temp_in[1],"\n");
write("zo",temp_in[2],"\n");

/* Performing matrix concatenations for rotation about an arbitrary axis */

lnk5_in[0] = temp_in[0]*1.0-px;
lnk5_in[1] = temp_in[1]*1.0-py;
lnk5_in[2] = temp_in[2]*1.0-pz;

temp_in[0] = lnk5_in[0]*1.0;
APPENDIX D

temp_in[1] = lnk5_in[1]*cz/d-lnk5_in[2]*cy/d;
temp_in[2] = lnk5_in[1]*cy/d + lnk5_in[2]*cz/d;

lnk5_in[0] = temp_in[0]*d-cx*temp_in[2];
lnk5_in[1] = temp_in[1]*1.0;
lnk5_in[2] = temp_in[0]*cx + d*temp_in[2];

temp_in[0] = cos(theta[4]*pi/180.0)*lnk5_in[0]
-sin(theta[4]*pi/180.0)*lnk5_in[1];
temp_in[1] = sin(theta[4]*pi/180.0)*lnk5_in[0]
+ cos(theta[4]*pi/180.0)*lnk5_in[1];
temp_in[2] = lnk5_in[2]*1.0;

lnk5_in[0] = temp_in[0]*d + temp_in[2]*cx;
lnk5_in[1] = temp_in[1]*1.0;
lnk5_in[2] = -temp_in[0]*cx + temp_in[2]*d;

temp_in[0] = lnk5_in[0]*1.0;
temp_in[1] = lnk5_in[1]*cz/d + lnk5_in[2]*cy/d;
temp_in[2] = -lnk5_in[1]*cy/d + lnk5_in[2]*cz/d;

lnk5_out[0] = temp_out[0]*1.0+px;
lnk5_out[1] = temp_out[1]*1.0+py;
lnk5_out[2] = temp_out[2]*1.0+pz;

write("x",lnk5_in[0],"\n");
write("y",lnk5_in[1],"\n");
write("z",lnk5_in[2],"\n");

write("xo",temp_out[0],"\n");
write("yo",temp_out[1],"\n");
write("zo",temp_out[2],"\n");

lnk5_out[0] = temp_out[0]*1.0-px;
lnk5_out[1] = temp_out[1]*1.0-py;
lnk5_out[2] = temp_out[2]*1.0-pz;

temp_out[0] = lnk5_out[0]*1.0;
temp_out[1] = lnk5_out[1]*cz/d-lnk5_out[2]*cy/d;
temp_out[2] = lnk5_out[1]*cy/d + lnk5_out[2]*cz/d;

lnk5_out[0] = temp_out[0]*d-cx*temp_out[2];
lnk5_out[1] = temp_out[1]*1.0;
lnk5_out[2] = temp_out[0]*cx + d*temp_out[2];

temp_out[0] = cos(theta[4]*pi/180.0)*lnk5_out[0]
-sin(theta[4]*pi/180.0)*lnk5_out[1];
temp_out[1] = sin(theta[4]*pi/180.0)*lnk5_out[0]
+ cos(theta[4]*pi/180.0)*lnk5_out[1];
temp_out[2] = lnk5_out[2]*1.0;
\[ \text{link5\_out}[0]=\text{temp\_out}[0]\times d+\text{temp\_out}[2]\times cx; \]
\[ \text{link5\_out}[1]=\text{temp\_out}[1]\times 1.0; \]
\[ \text{link5\_out}[2]=-(\text{temp\_out}[0]\times cy+\text{temp\_out}[2]\times cz); \]
\[ \text{temp\_out}[0]=\text{link5\_out}[0]\times 1.0; \]
\[ \text{temp\_out}[1]=\text{link5\_out}[1]\times cz/d+\text{link5\_out}[2]\times cy/d; \]
\[ \text{temp\_out}[2]=-(\text{link5\_out}[1]\times cy/d+\text{link5\_out}[2]\times cz/d); \]
\[ \text{link5\_out}[0]=\text{temp\_out}[0]\times 1.0+px; \]
\[ \text{link5\_out}[1]=\text{temp\_out}[1]\times 1.0+py; \]
\[ \text{link5\_out}[2]=\text{temp\_out}[2]\times 1.0+pz; \]

write("x",link5\_out[0],"n");
write("y",link5\_out[1],"n");
write("z",link5\_out[2],"n");

{} 

beginmd_key("GRSAn");
ci$put( value = \theta[5] );
endmd();

beginmd_key("GRRtEAbAx");
ci$put( obj = \text{link\_5} );
ci$put( point = \text{link5\_in}, window\_name = "front" );
ci$put( point = \text{link5\_out}, window\_name = "front" );
ci$put( response = TERMINATE );
endmd();

gr$last\_elements( pobj = &\text{link\_5},
parents = 1,
 nb\_wanted = 1,
 nb\_read = &number\_found );

beginmd_key("GRRtEAbAx");
ci$put( obj = \text{link\_6a} );
ci$put( point = \text{link5\_in}, window\_name = "front" );
ci$put( point = \text{link5\_out}, window\_name = "front" );
ci$put( response = TERMINATE );
endmd();

gr$last\_elements( pobj = &\text{link\_6a},
parents = 1,
 nb\_wanted = 1,
 nb\_read = &number\_found );

beginmd_key("GRRtEAbAx");
ci$put( obj = \text{link\_6b} );
APPENDIX D

```c
    ci$put( point = lnk5_in, window_name = "front");
ci$put( point = lnk5_out, window_name = "front");
ci$put( response = TERMINATE );
endcmd();

gr$last_elements( pobj = &link_6b,
    parents = 1,
    nb_wanted = 1,
    nb_read = &number_found );

}

px=lnk5_in[0];
py=lnk5_in[1];
px=lnk5_in[2];

qx=lnk5_out[0];
qy=lnk5_out[1];
qz=lnk5_out[2];

write("px,py,pz",px,py,pz,"\n");

/* Evaluating the direction cosines */

cx=(qx-px)/sqrt((qx-px)*(qx-px)+(qy-px)*(qy-px)+(qz-px)*(qz-px));
cy=(qy-px)/sqrt((qx-px)*(qx-px)+(qy-px)*(qy-px)+(qz-px)*(qz-px));
cz=(qz-px)/sqrt((qx-px)*(qx-px)+(qy-px)*(qy-px)+(qz-px)*(qz-px));
d=sqrt((cx*cy)+(cz*cz));
write("cx,cy,cz,d",cx,cy,cz,d,"\n");

temp_in[0]=cos(theta[1]*pi/180.0)*
(3.00*cos(theta[2]*pi/180.0)+3.50*sin((theta[2]+theta[3])*pi/180.0))
-sin(theta[1]*pi/180.0)*0.75;

temp_in[1]=sin(theta[1]*pi/180.0)*
(3.00*cos(theta[2]*pi/180.0)+3.50*sin((theta[2]+theta[3])*pi/180.0))
+cos(theta[1]*pi/180.0)*0.75;

temp_in[2]=-3.00*sin(theta[2]*pi/180.0)+3.50*cos((theta[2]+theta[3])*pi/180.0);


temp_out[0]=cos(theta[1]*pi/180.0)*
(3.00*cos(theta[2]*pi/180.0)+4.00*sin((theta[2]+theta[3])*pi/180.0))
-sin(theta[1]*pi/180.0)*0.75;

temp_out[1]=sin(theta[1]*pi/180.0)*
```

**Evaluating the direction cosines**
(3.00*cos(theta[2]*pi/180.0) + 4.00*sin((theta[2] + theta[3])*pi/180.0))
+ cos(theta[1]*pi/180.0)*0.75;

temp_out[2] = -3.00*sin(theta[2]*pi/180.0) + 4.00*cos((theta[2] + theta[3])*pi/180.0);

write("**** data for link_6 ****", "n");

write("xo", temp_in[0], "n");
write("yo", temp_in[1], "n");
write("zo", temp_in[2], "n");

/* Performing matrix concatenations for rotation about an arbitrary axis */

lnk6_in[0] = temp_in[0]*1.0 - px;
lnk6_in[1] = temp_in[1]*1.0 - py;
lnk6_in[2] = temp_in[2]*1.0 - pz;

temp_in[0] = lnk6_in[0]*1.0;
temp_in[1] = lnk6_in[1]*ez/d - lnk6_in[2]*cy/d;

lnk6_in[0] = temp_in[0]*dx*temp_in[2];
lnk6_in[1] = temp_in[1]*1.0;
lnk6_in[2] = temp_in[0]*ex + d*temp_in[2];

temp_in[0] = cos(theta[5]*pi/180.0)*lnk6_in[0]
- sin(theta[5]*pi/180.0)*lnk6_in[1];
temp_in[1] = sin(theta[5]*pi/180.0)*lnk6_in[0]
+ cos(theta[5]*pi/180.0)*lnk6_in[1];
temp_in[2] = lnk6_in[2]*1.0;

lnk6_in[0] = temp_in[0]*d + temp_in[2]*cx;
lnk6_in[1] = temp_in[1]*1.0;
lnk6_in[2] = -temp_in[0]*ex + temp_in[2]*d;

temp_in[0] = lnk6_in[0]*1.0;
temp_in[1] = lnk6_in[1]*ez/d + lnk6_in[2]*cy/d;

lnk6_in[0] = temp_in[0]*1.0 + px;
lnk6_in[1] = temp_in[1]*1.0 + py;
lnk6_in[2] = temp_in[2]*1.0 + pz;

write("x", lnk6_in[0], "\n");
write("y", lnk6_in[1], "\n");
write("z", lnk6_in[2], "\n");

write("xo", temp_out[0], "\n");
write("yo", temp_out[1], "\n");
write("zo", temp_out[2], "\n");
\[ \text{temp\_out}[0] = \text{lnk\_6\_out}[0] \times 1.0; \]
\[ \text{temp\_out}[1] = \text{lnk\_6\_out}[1] \times \text{cz/d} - \text{lnk\_6\_out}[2] \times \text{cy/d}; \]
\[ \text{temp\_out}[2] = \text{lnk\_6\_out}[1] \times \text{cy/d} + \text{lnk\_6\_out}[2] \times \text{cz/d}; \]
\[ \text{lnk\_6\_out}[0] = \text{temp\_out}[0] \times \text{d} - \text{cx} \times \text{temp\_out}[2]; \]
\[ \text{lnk\_6\_out}[1] = \text{temp\_out}[1] \times 1.0; \]
\[ \text{lnk\_6\_out}[2] = \text{temp\_out}[0] \times \text{cx} + \text{d} \times \text{temp\_out}[2]; \]
\[ \text{temp\_out}[0] = \cos(\theta[5] \times \pi/180.0) \times \text{lnk\_6\_out}[0] - \sin(\theta[5] \times \pi/180.0) \times \text{lnk\_6\_out}[1]; \]
\[ \text{temp\_out}[1] = \sin(\theta[5] \times \pi/180.0) \times \text{lnk\_6\_out}[0] + \cos(\theta[5] \times \pi/180.0) \times \text{lnk\_6\_out}[1]; \]
\[ \text{temp\_out}[2] = \text{lnk\_6\_out}[2] \times 1.0; \]
\[ \text{lnk\_6\_out}[0] = \text{temp\_out}[0] \times \text{d} + \text{temp\_out}[2] \times \text{cx}; \]
\[ \text{lnk\_6\_out}[1] = \text{temp\_out}[1] \times 1.0; \]
\[ \text{lnk\_6\_out}[2] = -\text{temp\_out}[0] \times \text{cx} + \text{temp\_out}[2] \times \text{d}; \]
\[ \text{temp\_out}[0] = \text{lnk\_6\_out}[0] \times 1.0; \]
\[ \text{temp\_out}[1] = \text{lnk\_6\_out}[1] \times \text{cz/d} + \text{lnk\_6\_out}[2] \times \text{cy/d}; \]
\[ \text{temp\_out}[2] = -\text{lnk\_6\_out}[1] \times \text{cy/d} + \text{lnk\_6\_out}[2] \times \text{cz/d}; \]
\[ \text{lnk\_6\_out}[0] = \text{temp\_out}[0] \times 1.0 + \text{px}; \]
\[ \text{lnk\_6\_out}[1] = \text{temp\_out}[1] \times 1.0 + \text{py}; \]
\[ \text{lnk\_6\_out}[2] = \text{temp\_out}[2] \times 1.0 + \text{pz}; \]
\[ \text{write}("x", \text{lnk\_6\_out}[0], \\
\text{"n"}); \]
\[ \text{write}("y", \text{lnk\_6\_out}[1], \\
\text{"n"}); \]
\[ \text{write}("z", \text{lnk\_6\_out}[2], \\
\text{"n"}); \]

\{
    \begin{cmd}
      \put(value = \theta[6]);
    \end{cmd}
\}

\begin{cmd}
    \put( obj = \text{link\_6a});
    \put( point = \text{lnk\_6\_in}, window\_name = "front");
    \put( point = \text{lnk\_6\_out}, window\_name = "front");
    \put( response = \text{TERMINATE});
\end{cmd}

\begin{cmd}
    \put( obj = \text{link\_6a});
\end{cmd}
APPENDIX D

```c
  ci$put( point = lnk6_in, window_name = "front");
  ci$put( point = lnk6_out, window_name = "front");
  ci$put( response = TERMINATE );
  endcmd();

  gr$last_elements( pobj = &link_6a,
          parents = 1,
          nb_wanted = 1,
          nb_read = &number_found );

begincmd_key("GRRtEAbAx");
  ci$put( obj = link_6b );
  ci$put( point = lnk6_in, window_name = "front");
  ci$put( point = lnk6_out, window_name = "front");
  ci$put( response = TERMINATE );
  endcmd();

  gr$last_elements( pobj = &link_6b,
          parents = 1,
          nb_wanted = 1,
          nb_read = &number_found );
}

  theta_initial[1]=theta[1];
  theta_initial[2]=theta[2];
  theta_initial[3]=theta[3];
  theta_initial[4]=theta[4];
  theta_initial[5]=theta[5];
  theta_initial[6]=theta[6];

  write("theta_initial",theta_initial[1],theta_initial[2],theta_initial[3],theta_initial[4],theta_initial[5],theta_initial[6],"\n");
  i=i+1;}//END WHILE

fclose(fptr);
```
This file contains the main routines for running the PUMA robot with the Trident Robotics and Research, Inc. interface boards.

/* -- ROBOT.C
   — Initial coding by Richard Voyles, Vigilant Technologies. */

#include <stdio.h>
#include <dos.h>
#include "robot.h"

typedef void interrupt (*hndlrgtr)(void);

/* — Input frequency to the Programmable Interval Timer (8253 or
   — equivalent) on IBM compatible machines. */
#define TIMER_INPUT_FREQ 1192500.0

/* — This is the external, user-supplied task that is executed every
   — timer tick. */
extern void periodic_task();

/* — Global variables (all globals end in "G") */
long clockTicksG; /* number of timer interrupts since init */
float clockPeriodG; /* period, in seconds, of interrupts */
float clockFreqG; /* actual frequency, in hertz */
short installFlagG = 0; /* flag indicating handler installed */
hndl_ptr oldHandlerG; /* function pointer to old timer handler */
PUMAstatusT pumaVarsG; /* structure containing PUMA variables */

/* — The following values are valid for a PUMA 560 */
short motorScaleG[6] = {400, -250, 400, -700, -600, -600};
float encoderScaleG[6] = {0.00010035, -0.000073156, 0.000117,
-0.000082663, -0.000087376, -0.000081885};

short robot_fltrd(short option, short chan, float *val);
short robot_intwr(short option, short chan, short *val);
short robot_fltrd(short option, short chan, float *val);
short robot_longrd(short option, short chan, long *val);

/* — This function is the interrupt handler for timer 0 (interrupt 8).
   — Working code can be inserted here or a call to another
   — subroutine. */
void interrupt timer_handler()
{
    clockTicksG += ; /* Working code or function call goes here. */
    periodic_task(); /* End of working code. */

    /* This line is required to exit properly. */
    outportb(0x020, 0x20); /* Send non-specific EOI to 8259A */
}
/* — This function sets the frequency of the timer 0 interrupts.  
   — The float value FREQ is in hertz. The actual interrupt frequency  
   — in hertz is returned. */
float interrupt_rate(float freq)
{
    int timerCnt;

    timerCnt = (int)(TIMER_INPUT_FREQ / freq);
    outportb(0x043,0x36); /* init timer 0 for count load */
    outportb(0x040,(char)timerCnt); /* load timer 0 count LSB */
    outportb(0x040,(char)(timerCnt > > 8)); /* load timer 0 count MSB */
    /* — Set the global variables that keep track of interrupt rate. */
    clockFreqG = TIMER_INPUT_FREQ / (float)timerCnt;
    clockPeriodG = (float)timerCnt / TIMER_INPUT_FREQ;
    return clockFreqG;
}

/* — This function installs the new timer 0 interrupt handler. It  
   — grabs the old timer handler for restoration upon quitting the  
   — program. */
void install_handler()
{
    if (installFlagG == 0){
        oldHandlerG = getvect(8); /* Save old timer interrupt handler */
        installFlagG = 1; /* set flag to indicate handler installed */
    } /* endif */
    clockTicksG = 0;
    setvect(8,timer_handler); /* Install new timer handler */
}

/* — This function provides an orderly shutdown by restoring the original  
   — timer interrupt handler. It should also reset the system time from  
   — the time-of-day clock. */
void shutdown()
{
    pumaVarsG.discrete = pumaVarsG.discrete & ~POWER_BIT;
    robot_intwr(STATUS, 0, &pumaVarsG.discrete);
    interrupt_rate(18.2);
    setvect(8,oldHandlerG);
    installFlagG = 0;
}

/* — This function initializes the system for robot control. It installs  
   — the interrupt handler and sets the desired interrupt frequency, in  
   — the proper order. It also initializes variables.  
   — It returns the actual interrupt frequency, in hertz. */
float system_init(float freq)
{
    int i;

    /* — First initialize the PUMA structure. */
    /* — Start by resetting everything but the <Joint 6 Overflow> and  
       — <Calibration Status> bits and setting the discrete variable. */
pumaVarsG.status = inport(TRC_BASE + STATUS_REG);
pumaVarsG.discrete = (pumaVarsG.status & 0x0C00) >> 8;
outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
pumaVarsG.status = inport(TRC_BASE + STATUS_REG);
/* -- Now set position variables appropriately */
if ((pumaVarsG.status & CALIB_MASK) == 0) {
    /* -- Arm IS NOT calibrated so set everything to zero. */
    for (i = 0; i < 6; i++) {
        pumaVarsG.pos[i] = 0L;       /* reset the variable */
        robot_intwr(POSITION, i, 0); /* reset the encoder */
    } /* endfor */
pumaVarsG.discrete = 0;       /* make sure all bits are cleared */
outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
} else {
    /* -- Arm IS calibrated */
    pumaVarsG.oldPos5 = 0;
    /* -- This is not quite complete -- must deal with bit 17 */
    robot_longrd(POSITION6, 0, &pumaVarsG.pos[0]);
} /* endif */

/* -- These are required for initializing the interrupt handler */
install_handler();
return interrupt_rate(freq);

/* -- This function writes 1 or 6 float values to the robot. The particular
   value is chosen with the OPTION parameter, which is one out of a
   list of enumerated ioOptionsT. CHAN ranges from 0 to 5 (corresponding
   to PUMA joints 1-6). It returns one of the enumerated ioErrorsT. */
short robot_floatw(short option, short chan, float *val)
{
    short intVal,
    i;

    switch (option) {
    case TORQUE6:       /* Output to all DACs -- val is in N-m */
        for (i = 0; i < 6; i++) {
            intVal = (int)(val[i] * motorScaleG[i]) + 0x0800;
            outport(TRC_BASE + DAC_BASE + 2*i, intVal);
        } /* endfor */
        return OK;
    case TORQUE:         /* Output to single DAC -- val is in N-m */
        intVal = (int)(*val * motorScaleG[chan]) + 0x0800;
        outport(TRC_BASE + DAC_BASE + 2*chan, intVal);
        return OK;
    case VOLTAGE:        /* Output to single DAC -- val is in volts */
        if (*val >= 10.0) {
            intVal = 0;
        } else if (*val <= -9.995) {
            intVal = 0x0FFF;
        } else {
            intVal = (int)(*val * DAC_BITS_PER_VOLT) + 0x0800;
        } /* endif */
    }
APPENDIX D

outport(TRC_BASE + DAC_BASE + 2*chan, intVal);
return OK;
case POSITION6:  /* Preset all encoders -- val is in radians */
    for (i=0; i<6; i++){
        intVal = (int)(val[i] / encoderScaleG[i]);
        outport(TRC_BASE + ENC_LOAD_BASE + 2*i, intVal);
    } /* endfor */
    return OK;
case POSITION:  /* Preset a single encoder -- val is in radians */
    intVal = (int)(val / encoderScaleG[chan]);
    outport(TRC_BASE + ENC_LOAD_BASE + 2*chan, intVal);
    return OK;
default:
    return BAD_OPTION;
} /* endswitch */

/* — This function writes 1 or 6 integer values to the robot. The particular — value is chosen with the OPTION parameter, which is one out of a — list of enumerated ioOptionsT. CHAN ranges from 0 to 5 (corresponding — to PUMA joints 1 - 6). It returns one of the enumerated ioErrorsT. */
short robot_intwr(short option, short chan, short *val)
{
    short i;

    switch (option) {
    case TORQUE6:  /* Output to all DACs */
        for (i=0; i<6; i++){
            outport(TRC_BASE + DAC_BASE + 2*i, val[i]);
        } /* endfor */
        return OK;
    case TORQUE:  /* Output to single DAC */
        outport(TRC_BASE + DAC_BASE + 2*chan, *val);
        return OK;
    case POSITION6:  /* Preset all encoders */
        for (i=0; i<6; i++){
            outport(TRC_BASE + ENC_LOAD_BASE + 2*i, val[i]);
        } /* endfor */
        return OK;
    case POSITION:  /* Preset a single encoder */
        outport(TRC_BASE + ENC_LOAD_BASE + 2*chan, *val);
        return OK;
    case STATUS:  /* Write the entire status word */
        outport(TRC_BASE + DISCRETE_REG, *val);
        return OK;
    default:
        return BAD_OPTION;
    } /* endswitch */
}

/* — This function reads 1 or 6 float values from the robot. The particular — value is chosen with the OPTION parameter, which is one out of a — list of enumerated ioOptionsT. CHAN ranges from 0 to 5 (corresponding
short robot_fltrd(short option, short chan, float *val)
{
    short tempPos,
    i;

    switch (option) {
    case POSITION6: /* Read all encoders -- val is in radians */
        for (i=0; i<5; i++) {
            val[i] = encoderScaleG[i] * import(TRC_BASE + ENC_COUNT_BASE + 2*i);
        } /* endfor */
        /* Joint 6 must be treated differently because it needs 17 bits. */
        tempPos = import(TRC_BASE + ENC_COUNT_BASE + 10);
        pumaVarsG.pos[5] += (long)(tempPos - pumaVarsG.oldPos5);
        pumaVarsG.oldPos5 = tempPos;
        /* Set the j6 overflow bit appropriately. */
        if (pumaVarsG.pos[5] < 0) {
            pumaVarsG.discrete = pumaVarsG.discrete | JT6_OVRFLO_BIT;
        } else {
            pumaVarsG.discrete = pumaVarsG.discrete & (~JT6_OVRFLO_BIT);
        } /* endif */
        outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
        return OK;
    case POSITION: /* Read a single encoder */
        if (chan != 5) {
            *val = encoderScaleG[chan] * import(TRC_BASE + ENC_COUNT_BASE + 2*chan);
        } else {
            /* Joint 6 must be treated differently because it needs 17 bits. */
            tempPos = import(TRC_BASE + ENC_COUNT_BASE + 10);
            pumaVarsG.pos[5] += (long)(tempPos - pumaVarsG.oldPos5);
            *val = encoderScaleG[5] * pumaVarsG.pos[5];
            pumaVarsG.oldPos5 = tempPos;
            /* Set the j6 overflow bit appropriately. */
            if (pumaVarsG.pos[5] < 0) {
                pumaVarsG.discrete = pumaVarsG.discrete | JT6_OVRFLO_BIT;
            } else {
                pumaVarsG.discrete = pumaVarsG.discrete & (~JT6_OVRFLO_BIT);
            } /* endif */
            outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
        } /* endif */
        return OK;
    case POT: /* Read a single ADC -- val is in volts */
        outport(TRC_BASE + ADC_MUX_SELECT, chan); /* select the MUX channel */
        import(TRC_BASE + ADC_START); /* trigger the conversion */
        /* Enable the ADC conversion complete bit. */
        pumaVarsG.discrete = pumaVarsG.discrete | ADC_MASK_BIT;
        outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
        /* Wait for the conversion complete flag. */
        while (! (import(TRC_BASE + STATUS_REG) & ADC_STAT_MASK));
        *val = 0.0196 * (import(TRC_BASE + ADC_VALUE) & 0x00FF); /* Convert 8-bit value to single 16-bit */
        /* Disable and clear the ADC conversion complete bit. */
        pumaVarsG.discrete = pumaVarsG.discrete & (~ADC_MASK_BIT);
This function writes 1 or 6 LONG values to the robot. The particular
value is chosen with the OPTION parameter, which is one out of a
list of enumerated ioOptionsT. CHAN ranges from 0 to 5 (corresponding
to PUMA joints 1 - 6). It returns one of the enumerated ioErrorsT. */
short robot_longrd(short option, short chan, long *val)
{
    short i,
    tempPos;

    switch (option) {
    case POSITION6: /* Read all encoders */
        for (i=0; i<5; i++) {
            val[i] = (long)inport(TRC_BASE + ENC_COUNT_BASE + 2*i);
        } /* endfor */
        /* Joint 6 must be treated differently because it needs 17 bits. */
        tempPos = inport(TRC_BASE + ENC_COUNT_BASE + 10);
        val[5] += (long)(tempPos - pumaVarsG.oldPos5);
        pumaVarsG.oldPos5 = tempPos;
        /* -- Set the JT6 overflow bit appropriately. */
        if (val[5] < 0)
            pumaVarsG.discrete = pumaVarsG.discrete | JT6_OVRFLO_BIT;
        else
            pumaVarsG.discrete = pumaVarsG.discrete & (~JT6_OVRFLO_BIT);
        } /* endif */
        outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
        return OK;
    case POSITION: /* Read a single encoder */
        if (chan != 5) {
            *val = (long)inport(TRC_BASE + ENC_COUNT_BASE + 2*chan);
        } else {
            /* Joint 6 must be treated differently because it needs 17 bits. */
            tempPos = inport(TRC_BASE + ENC_COUNT_BASE + 10);
            *val += (long)(tempPos - pumaVarsG.oldPos5);
            pumaVarsG.oldPos5 = tempPos;
            /* -- Set the JT6 overflow bit appropriately. */
            if (*val < 0)
                pumaVarsG.discrete = pumaVarsG.discrete | JT6_OVRFLO_BIT;
            else
                pumaVarsG.discrete = pumaVarsG.discrete & (~JT6_OVRFLO_BIT);
        } /* endif */
        outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
        } /* endif */
        return OK;
    case STATUS: /* Read the entire status word into long int */
        *val = (long)((unsigned short)inport(TRC_BASE + STATUS_REG));
        return OK;
    default:
        return BAD_OPTION;
    } /* endswitch */
} /* endswitch */
APPENDIX D

```c
case POT:    /* Read a single ADC */
    outport(TRC_BASE + ADC_MUX_SELECT, chan); /* select the MUX channel */
    inport(TRC_BASE + ADC_START);            /* trigger the conversion */
    /* -- Enable the ADC conversion complete bit. */
    pumaVarsG.discrete = pumaVarsG.discrete | ADC_MASK_BIT;
    outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
    /* -- Wait for the conversion complete flag. */
    while ((inport(TRC_BASE + STATUS_REG) & ADC_STAT_MASK) != 0);
    *val = (long)(inport(TRC_BASE + ADC_VALUE) & 0x00FF);
    /* -- Disable and clear the ADC conversion complete bit. */
    pumaVarsG.discrete = pumaVarsG.discrete & (~ADC_MASK_BIT);
    outport(TRC_BASE + DISCRETE_REG, pumaVarsG.discrete);
    return OK;
    default:
        return BAD_OPTION;
} /* endswitch */
```
APPENDIX D

D-13 This header file contains defined constants for use in both ROBOT.C and ROBOTUSR.C when using the interface boards from Trident Robotics and Research, Inc.

/* — ROBOT.H
   — Initial coding by Richard Voyles, Vigilant Technologies */

/* — Enumerated types defining return codes for the I/O routines and
   — options for the I/O routines.
   — All typedefs end in "T". */
typedef enum { OK,
   BAD_OPTION
 } ioErrorsT;

typedef enum { TORQUE,        /* W torque to a single joint motor */
   TORQUE6,        /* W torques to all joint motors */
   VOLTAGE,        /* W voltage to a single joint motor */
   POSITION,       /* R/W position of a single joint */
   POSITION6,      /* R/W position of all joints */
   STATUS,         /* R/W discrete status/control bits */
   POT,            /* R ADC value */
 } ioOptionsT;

/* — Structure that holds the PUMA information */
typedef struct{
   short status;
   short discrete;
   long pos[6];
   short oldPos5;
 } PUMAstatusT;

/* — Base I/O address for the TRC006. */
#define TRC_BASE 0x7000

/* — Address offsets for functional groups of utilities. */
#define ENC_INDEX_BASE 0x0000
#define STATUS_REG 0x000C
#define ENC_COUNT_BASE 0x0010
#define ADC_VALUE 0x001C
#define ADC_START 0x001E
#define ENC_LOAD_BASE 0x0020
#define ADC_MUX_SELECT 0x002C
#define DISCRETE_REG 0x002E
#define DAC_BASE 0x0030

/* — Masks for the status register. */
#define CALIB_MASK 0x4000
#define ADC_STAT_MASK 0x0040

/* — Masks for the discrete register. */
#define POWER_BIT 0x0001
#define HAND_OPEN_BIT 0x0002
#define HAND_CLOSED_BIT 0x0004
APPENDIX D

```
#define CALIB_BIT     0x0040
#define JT6_OVRFLO_BIT 0x0080
#define ADC_MASK_BIT  0x4000

/* — Reciprocal of DAC gain (gain is set by resistors) */
#define DACBITS_PER_VOLT -204.8
```
D-14 This is an example user-code file for controlling the PUMA robot with the Trident Robotics and Research, Inc. interface boards.

/* — ROBOTUSR.C
   — Initial coding by Richard Voyles, Vigilant Technologies */

#include <stdio.h>
#include <math.h>
#include "robotusr.h"

extern int calibrate(int chan);

/* — Global variables */
float KPosG[6], /* position error gain for PD cntrlr */
     KVelG[6]; /* velocity error gain for PD cntrlr */
float posRefG[6], /* joint reference positions */
     velRefG[6]; /* joint reference velocity */
int cntlFlagG; /* control start/stop flag */

/* — Simple PD controller with gravity compensation on joints 2 and 3. */
void pdgControl(float pos[6], float vel[6], float torque[6])
{
    float gravity[3];

    torque[0] = KPosG[0]*(posRefG[0] - pos[0]) + KVelG[0]*(velRefG[0] - vel[0]);

    /* calculate gravity compensation for joints 2 and 3 */
    gravity[2] = 0.75 * sin(pos[1] - pos[2]);
    gravity[1] = -2.0 * sin(pos[1]) - gravity[2];

    torque[1] += gravity[1];
    torque[2] += gravity[2];
}

/* — This function must appear and must be named exactly as shown.
   — It is executed every timer tick. */
void periodic_task()
{
    float posNew[6],
         *tempPtr,
         velTemp[6],
         outTorq[6],
         factor;
    static float posOld[2][6];
APPENDIX D

/* --- WARNING!!! The numeric coprocessor state is NOT saved. If this
--- interrupt can clobber a floating point calculation, you must
--- save state and restore upon exit. */

factor = 2.0 * clockFreqG;
if (cntrlFlagG){
    robot_fltrd(POSITION6, 0, posNew); /* get current robot pos */
    /* calculate velocities */
    velTemp[0] = (posNew[0] - posOld[0][0]) * factor;
    velTemp[1] = (posNew[1] - posOld[1][1]) * factor;
    pdgControl(posNew, velTemp, outTorq); /* calculate control law */
    robot_fltrd(TORQUE6, 0, outTorq); /* write torque values */
} /* endif */

/* --- Time shift the position values */
posOld[1][0] = posOld[0][0];
posOld[0][0] = posNew[0];
posOld[1][1] = posOld[0][1];
posOld[0][1] = posNew[1];
posOld[1][2] = posOld[0][2];
posOld[0][2] = posNew[2];
posOld[1][3] = posOld[0][3];
posOld[0][3] = posNew[3];
posOld[1][4] = posOld[0][4];
posOld[0][4] = posNew[4];
posOld[1][5] = posOld[0][5];
posOld[0][5] = posNew[5];
}

void print_menu()
{
    printf("Menu options - > \n");
    if (cntrlFlagG) {
        printf("c	Controller toggle off \n");
    } else {
        printf("c	Controller toggle on \n");
    } /* endif */
    printf("t	Controller toggle on \n");
    printf("C	Calibrate \n");
    printf("t	Disable arm power \n");
    printf("tD	Disable output \n");
    printf("tE	Enable arm power \n");
    printf("tP	Position display \n");
    printf("tP	Position reference display \n");
    printf("tq	Quit \n");
    printf("tR	Reference position \n");
    printf("tS	Status read \n");
    printf("tW	Write a DAC voltage \n");
    printf("tW	Write a DAC value in hex \n");
    printf("tZ	Zero the DACs \n");
}
APPENDIX D

```c
void main (argc, argv, envp)
    int argc;
    char *argv[];
    char *envp;
{
    short keepGoing = 1,
        jtNum, value, va1[6],
        ans;
    long longVal;
    float actualFreq,
         volts,
         posTemp[6];
    
    cntrlFlagG = 0;  /* turn off controller */
    /* -- This call sets up the timer interrupt. */
    actualFreq = system_init(300.0);
    printf("Actual interrupt rate is %8.3f Hz \n",actualFreq);
    
    KPosG[0] = 30.0;
    KPosG[1] = 300.0;
    KPosG[2] = 35.0;
    KPosG[3] = 30.0;
    KPosG[4] = 25.0;
    KPosG[5] = 15.0;
    KVelG[0] = 0.0;
    KVelG[1] = 0.0;
    KVelG[2] = 0.0;
    KVelG[3] = 0.0;
    KVelG[4] = 0.0;
    KVelG[5] = 0.0;
    
    while(keepGoing){
        if (kbhit()){
            ans = getch();
            switch (ans) {
                case 'c':    /* Control flag toggle */
                    if (cntrlFlagG){
                        cntrlFlagG = 0;
                    } else {
                        cntrlFlagG = 1;
                    } /* endif */
                    break;
                case 'C':
                    printf("Enter joint number (1 - 6): ");
                    scanf("%d",&jtNum);
                    if ((jtNum >= 1) && (jtNum <= 6)){
                        calibrate(jtNum-1);
                    } else {
```
APPENDIX D

```c
  printf("Joint number out of range \n");
} /* endif */
break;
case 'd': /* Disable arm power */
pumaVarsG.discrete = pumaVarsG.discrete & ~POWER_BIT;
robot_intwr(STATUS, 0, &pumaVarsG.discrete);
break;
case 'D': /* Disable arm power */
  printf("Discretes are: %04X enter new value in hex: ",
pumaVarsG.discrete);
  scanf("%x",&pumaVarsG.discrete);
  robot_intwr(STATUS, 0, &pumaVarsG.discrete);
break;
case 'e': /* Enable arm power */
pumaVarsG.discrete = pumaVarsG.discrete | POWER_BIT;
robot_intwr(STATUS, 0, &pumaVarsG.discrete);
break;
case 'p':
  robot_longrd(POSITION6, 0, &pumaVarsG.pos[0]);
  printf("Positions: \n");
  printf("%6ld %6ld %6ld \n",pumaVarsG.pos[0], pumaVarsG.pos[1],
pumaVarsG.pos[2]);
  printf("%6ld %6ld %6ld \n",pumaVarsG.pos[3], pumaVarsG.pos[4],
pumaVarsG.pos[5]);
  printf("%5.3f %5.3f %5.3f \n",encoderScaleG[0]*pumaVarsG.pos[0],
    encoderScaleG[1]*pumaVarsG.pos[1],
    encoderScaleG[2]*pumaVarsG.pos[2]);
  printf("%5.3f %5.3f %5.3f \n",encoderScaleG[3]*pumaVarsG.pos[3],
    encoderScaleG[4]*pumaVarsG.pos[4],
    encoderScaleG[5]*pumaVarsG.pos[5]);
  break;
case 'P':
  printf("Position references: \n");
  printf("%5.3f %5.3f %5.3f \n",posRefG[0] ,posRefG[1] ,posRefG[2]);
  printf("%5.3f %5.3f %5.3f \n",posRefG[3] ,posRefG[4] ,posRefG[5]);
  break;
case 'q': /* Quit */
  keepGoing = 0;
  break;
case 'r': /* Reference position */
  printf("Enter joint number (1 - 6): ");
  scanf("%d",&jtNum);
  if (((jtNum >= 1) && (jtNum <= 6))
    printf("Enter reference position in radians: ");
    scanf("%f",&posRefG[jtNum-1]);
  else {
    printf("Joint number out of range \n");
  } /* endif */
  break;
case 's':
  robot_longrd(STATUS, 0, &longVal);
  value = longVal;
  printf("Status = %04X \n",value);
```

APPENDIX D

break;
case 'w':  /* Write DAC value */
    printf("Enter joint number (1 - 6): ");
    scanf("%d", &jtNum);
    if ((jtNum >= 1) && (jtNum <= 6)) {
        printf("Enter voltage in volts: ");
        scanf("%f", &volts);
        robot_fltwr(VOLTAGE, jtNum - 1, &volts);
    } else {
        printf("Joint number out of range \n");
    } /* endif */
    break;
case 'W':  /* Write DAC value */
    printf("Enter joint number (1 - 6): ");
    scanf("%d", &jtNum);
    if ((jtNum >= 1) && (jtNum <= 6)) {
        printf("Enter value in hex: ");
        scanf("%x", &value);
        robot_intwr(TORQUE, jtNum - 1, &value);
    } else {
        printf("Joint number out of range \n");
    } /* endif */
    break;
case 'z':  /* Zero the DACs */
    robot_intwr(TORQUE6, 0, val);
    break;
case 'Z':  /* Zero the encoder counters */
    robot_intwr(POSITION6, 0, val);
    break;
case '?':
    print_menu();
    break;
default:
    printf("Bad command, try again\n");
    print_menu();
} /* endswitch */
} /* endif */
} /* endwhile */

/* — Make sure PUMA arm power is off */
pumaVarsG.discrete = pumaVarsG.discrete & ~POWER_BIT;
robot_intwr(STATUS, 0, &pumaVarsG.discrete);

/* !!! IMPORTANT !!!
— This call is required to restore the system clock */
shutdown();
}
APPENDIX D

D-15 This is the main header file for utilizing functions provided for accessing the PUMA robot using the interface boards from Trident Robotics and Research, Inc.

/* — ROBOTUSR.H
   — Initial coding by Richard Voyles, Vigilant Technologies */

#include "robot.h"

/* — Global variables defined in ROBOT.C (all globals end in "G") */
extern long clockTicksG; /* number of timer interrupts since init */
extern float clockPeriodG; /* period, in seconds, of interrupts */
extern float clockFreqG; /* actual frequency, in hertz */
extern PUMAstatusT pumaVarsG; /* structure containing PUMA variables */

extern short motorScaleG[6];
extern float encoderScaleG[6];

/* — This function sets up the timer interrupt */
extern float system_init(float freq);

/* — This function re-installs the original timer interrupt handler. */
extern void shutdown();

/* — These functions access the TRC006 */
extern short robot_flwr(short option, short chan, float *val);
extern short robot_intwr(short option, short chan, short *val);
extern short robot_fltrd(short option, short chan, float *val);
extern short robot_longrd(short option, short chan, long *val);

/* — The following functions are generally not called directly by the
   — user. The user should place a call to system_init(), which calls
   — the appropriate routines as necessary. */

/* — This function sets a new interrupt rate. */
extern float interrupt_rate(float freq);

/* — This function installs a standard timer interrupt handler. */
extern void install_handler();
APPENDIX D

D-16 File for calibration of the robot using the Trident Robotics cards.

/* -- CALIB.C
   -- Initial coding by Richard Voyles, Vigilant Technology */

#include "robotusr.h"

/* -- These global variables are defined in ROBOTUSR.C */
extern float posRefG[6];  /* joint reference positions */
extern float velRefG[6];  /* joint reference velocity */

/* -- This routine calibrates a single joint, selected by CHAN (0-5).
   -- A joint position controller must be running and it must use
   -- the global variables referenced above. */
int calibrate(int chan)
{
    int  keepGoing = 0;
    short discreteMask = 0x0100,
         statusMask = 0x0001;
    long  clockNow;
    float posInc[6] = {0.001, 0.001, 0.001, 0.001, 0.001, 0.001};

    if ((chan >= 0) && (chan < 5)) {
        keepGoing = 1;
        velRefG[chan] = 0.0;  /* Set velocity ref to zero */
        discreteMask = discreteMask << chan;
        statusMask = statusMask << chan;
        /* -- Clear the index flag by disabling then enabling */
        pumaVarsG.discrete = pumaVarsG.discrete & ~discreteMask;
        robot_intwr(STATUS, 0, &pumaVarsG.discrete);  /* disable */
        pumaVarsG.discrete = pumaVarsG.discrete | discreteMask;
        robot_intwr(STATUS, 0, &pumaVarsG.discrete);  /* enable */
        /* -- Always move the joint toward the zero position. */
        if (posRefG[chan] > 0.0) {
            posInc[chan] = -posInc[chan];
        } /* endif */
        clockNow = clockTicksG;
    } /* endif */

    while (keepGoing) {
        if (clockTicksG != clockNow) {  /* update posRef each clock tick */
            clockNow = clockTicksG;
            posRefG[chan] += posInc[chan];
        } /* endif */
        if ((inport(TRC_BASE + STATUS_REG) & statusMask) == 0) {
            /* -- An index pulse was found. */
            keepGoing = 0;
        } /* endif */
        if (posInc[chan] <= 0) {
            keepGoing = 0;
        } /* endif */
    } /* end while */
    return posInc[chan];
} /* calibrate */
APPENDIX D

/* endwhile */
/* — Disable the index flag bit for this encoder channel */
pumaVarsG.discrete = pumaVarsG.discrete & ~discreteMask;
robot_intwr(STATUS, 0, &pumaVarsG.discrete);
return 0;
}
Appendix E

E-1 The Approximate Model

The Explicit Dynamic Model and Inertial Parameters of the PUMA 600 Arm

Brian Armstrong, Oussama Khatib, Joel Burdick

Stanford Artificial Intelligence Laboratory
Stanford University

Abstract

To provide COSMOS, a dynamic model based manipulator control system, with an improved dynamic model, a PUMA 600 arm was disassembled; the inertial properties of the individual links were measured; and an explicit model incorporating all of the non-zero measured parameters was derived. The explicit model of the PUMA arm has been obtained with a derivation procedure comprised of several heuristic rules for simplification. A simplified model, abbreviated from the full explicit model with a 1% significance criterion, can be evaluated with 305 calculations, one fifth the number required by the recursive Newton-Euler method. The procedure used to derive the model is laid out; the measured inertial parameters are presented, and the model is included in an appendix.

1. Introduction

The implementation of dynamic control systems for manipulators has been hampered because the models are difficult to derive and computationally expensive, and because the needed parameters of the manipulator are generally unavailable. Recursive methods for computing the dynamic forces have been available for several years [Luh, Walker and Paul 1980a; Hollerbach 1980]. Several authors have proposed and simulated the use of RNE in control systems [Luh, Walker and Paul 1980b; Kim and Shin 1985]; and [Valavanis, Leahy and Sardis 1985] have used the RNE to control a PUMA - 600 arm. The RNE algorithm has also found use in the computation of forward dynamics for simulation [Walker and Orin 1982; Kozekanani, et al. 1983], and nominal trajectory control [Vukobratovic and Kirtzanski 1984]. The RNE meets the need for calculation of dynamic forces in these applications, but does not offer several advantages available provided by an explicit model. The explicit model allows of the calculation decomposition based on a significance criterion or other criteria, and provides a more direct solution for dynamic simulations. The tremendous size of an explicit dynamic model is the greatest barrier to its realization. Correspondingly, a considerable portion of the effort spent investigating dynamic models for control has been directed toward efficient formulation and automatic generation of the manipulator equations of motion. Programs for automatic generation of manipulator dynamics are reported in [Liégeois et al. 1976; Megahed and Renaud 1982; Cesare, F. Nicolò and S. Niccolò 1984; Murray and Newman 1984; Renaud 1984; Aldon and Liégeois 1984; Aldon et al. 1985]. The size of the models generated by these programs varies widely; and there is little consensus on the question of whether the explicit models can be made sufficiently compact to be used for control. Aldon and Liégeois [1984] present an algorithm for obtaining efficient dynamic models, but none-the-less recommend the use of recursive algorithms for real time control, claiming that the complete results are too complicated for real-time control of robots.

As we show, explicit dynamic models of manipulators that are more computationally efficient than the alternative recursive algorithms can be obtained. The computational cost of the RNE algorithm, the full explicit PUMA model, and the explicit PUMA model abbreviated with a 1% significance criterion are presented in Table 1. The method presented here for factoring the dynamic equations has yielded a dynamic model of the PUMA 600 arm.

Table 1. Calculations Required to Compute the Forces of Motion by 3 Methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Calculations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursive Newton-Euler</td>
<td>1560</td>
</tr>
<tr>
<td>Evaluation of the Full Explicit PUMA Model</td>
<td>1165</td>
</tr>
<tr>
<td>Evaluation of the Abbreviated Explicit PUMA Model</td>
<td>305</td>
</tr>
</tbody>
</table>

that requires 1165 calculations (139 multiplications and 426 additions), 25% fewer than the 1560 calculations required by the 6 dof RNE. With the application of a 1% sensitivity criterion, the explicit model can be evaluated with one fifth the count of calculations required by the recursive algorithm. Furthermore, this formulation of the explicit model is not optimally compact; factorizations that were discovered and employed during the model derivation have been expanded out to present explicit expressions for each component of the dynamic model. Renaud and Burdick both report automatic generation of 6 dof manipulator models that are more compact than that presented here [Renaud 1984; Burdick 1985]. Their models incorporate nested factorizations, which were not used here.

The count of 1165 calculations for the full PUMA model is the total required to evaluate the model presented in the appendix and equation (1) below. This total and other totals presented do not include the calculations required to evaluate the sines and cosines.

2. Derivation of the Dynamic Model

The dynamic model used for this analysis follows from [Liégeois et al. 1976]. It is:
APPENDIX E

\[ A(q)\dot{q} + B(q)\ddot{q} + C(q)\dot{q}^2 + g(q) = \Gamma; \]  

where \( A(q) \) is the \( n \times n \) kinetic energy matrix;
\( B(q) \) is the \( n \times n(n-1)/2 \) matrix of Coriolis torques;
\( C(q) \) is the \( n \times n \) matrix of centrifugal torques;
\( g(q) \) is the \( n \)-vector of gravity torques;
\( \dot{q} \) is the \( n \)-vector of accelerations;
\( \Gamma \) is the generalized joint force vector.

The symbols \([qq]\) and \([\dot{q}]^2\) are notation for the \( n \times 1 \) vector of velocity products and the \( n \times \) vector of squared velocities. \([qq]\) and \([\dot{q}]^2\) are given by:

\[
[qq] = [\dot{q}_1\dot{q}_2 \dot{q}_3 \ldots \dot{q}_n, \dot{q}_2\dot{q}_3 \ldots \dot{q}_n \ldots \dot{q}_n - \dot{q}_n\dot{q}_n, \dot{q}_n - \dot{q}_n\dot{q}_n]^T, \\
[\dot{q}]^2 = [\dot{q}_1^2, \dot{q}_2^2 \ldots \dot{q}_n^2]^T.
\]

The procedure used to derive the dynamic model entails four steps:

1. Symbolic Generation of the kinetic energy matrix and gravity vector elements by performing the summation of either Lagrange's or the Gibbs-Alembert formulation.
2. Simplification of the kinetic energy matrix elements by combining inertia constants that multiply common variable expressions.
3. Expression of the Coriolis and centrifugal matrix elements in terms of partial derivatives of kinetic energy matrix elements; and reduction of these expressions with four relations that hold on these partial derivatives.
4. Formation of the needed partial derivatives, expansion of the Coriolis and centrifugal matrix elements in terms of the derivatives, and simplification by combining inertia constants as in 2.

The first step was carried out with a LISP program, named EMDEG, which symbolically generates the dynamic model of an articulated mechanism. EMDEG employs Kane's dynamic formulation [Kane 1968], and produced a result comparable in form and size to that of ARM [Murry and Neuman 1984]. Three simplifying assumptions were made for this analysis: the rigid body assumption; link 6 has been assumed to be symmetric, that is \( I_{xx} = I_{yy} \); and only the mass moments of inertia are considered, that is \( I_{zz}, I_{xx}, \) and \( I_{yy} \). The original output of EMDEG, including Coriolis and centrifugal terms, required 15,000 multiplications and 45,500 additions. This step might also have been performed with the momentum theorem method used in [Itazguirre and Paul 1985].

In the second step of this procedure, the kinetic energy matrix elements are simplified by combining inertia constants that multiply common variable expressions. This is the greatest source of computational efficiency. Looking to the dynamic model of a 3 degree-of-freedom manipulator presented in [Murry and Neuman 1984], we see that the kinetic energy matrix element \( a_{11} \) is given by:

\[
a_{11} = J_{xx} \cos^2(\theta_2 + \theta_3) + J_{yy} \sin^2(\theta_2 + \theta_3) + J_{zz} + d_3^2 m_2^2 + 2 M_{xx} \cos(\theta_2 + \theta_3) \cos(\theta_2 + \theta_3) + d_3^2 m_3^2 \cos(\theta_2) + 2 M_{yy} \cos(\theta_2 + \theta_3) \cos(\theta_2 + \theta_3) + d_3^2 m_4^2 \cos(\theta_2) + 2 M_{zz} \cos^2(\theta_2) + 2 d_2 d_3 m_3 + 2 M_{xx} \cos(\theta_2) \cos(\theta_2) + d_2 d_3 m_4 + J_{xx} + J_{yy} + J_{zz}.
\]

Calculations required: 37 multiplications, 18 additions.

By combining inertia constants with common variable terms and expanding \( \sin^2(\theta_2) \) into \((1 - \cos^2(\theta_2))\), equation (2) can be reduced to:

\[
a_{11} = I_1 + I_2 \cos^2(\theta_2) + I_3 \cos(\theta_2) \cos(\theta_2 + \theta_3) + I_4 \cos^2(\theta_2 + \theta_3)
\]

Calculations required: 3 multiplications, 3 additions.

where \( I_1 = d_3^2 m_2 + d_3^2 m_3 + 2 d_3 d_4 m_5 + d_4^2 m_2 + J_{xy} + J_{zz} + J_{yy} + J_{xy} + J_{yz} + J_{xz} \), etc.

Creating \( I_1 \) through \( I_4 \), which are constants of the mechanism, leads to a reduction from 35 to 3 multiplications and from 18 to 3 additions. Computing the constant \( I_1 \) involves 18 calculations. Since the simple parameters required for the calculation of \( I_1 \) are the input to the RNE, the RNE will effectively carry out the calculation of \( I_1 \) on every pass, producing considerable unnecessary computation. Thirty four lumped constants are needed by the full PUMA model, 8 fewer than the count of 42 simple parameters required to describe the arm.

In the third step the elements of the Coriolis matrix, \( b_{ij} \), and of the centrifugal matrix, \( c_{ij} \), are written in terms of the Christoffel symbols of the first kind [Corben and Stehle 1950; Liegeois et al. 1970] giving:

\[
b_{ij} = 2 \beta^{i*} \dot{k} \\
c_{ij} = \dot{g}^{i*} \dot{k}
\]

where \((\dot{q}_k \cdot \dot{q}_i)\) is the \( j^{th} \) velocity product in the \([qq]\) vector, and where

\[
\beta^{i*} = \frac{1}{2} \left( \frac{\partial q_{ik}}{\partial \theta_j} + \frac{\partial q_{jk}}{\partial \theta_i} - \frac{\partial q_{ij}}{\partial \theta_k} \right)
\]

is the Christoffel symbol.

The number of unique non-zero Christoffel symbols required by the PUMA model can be reduced from 126 to 39 with four equations that hold on the derivatives of the kinetic energy matrix elements. The first two equations are general; the last two are specific to the PUMA 560. The equations are:

\[
\frac{\partial A_{il}}{\partial \theta_l} = \delta A_{il} \qquad \forall_{i,j,k} \\
\frac{\partial A_{ij}}{\partial \theta_1} = 0 \\
\frac{\partial A_{ij}}{\partial \theta_2} = 0 \\
\frac{\partial A_{ij}}{\partial \theta_3} = 0 \\
\frac{\partial A_{ij}}{\partial \theta_4} = \delta A_{ij} \delta A_{ij} = \delta A_{ij}
\]

The reduction of Equation (7) arises from the symmetry of the kinetic energy matrix. Equation (8) obtains because the kinetic energy imparted by the velocity of a joint is independent of the configuration of the prior joints. Equation (9) results from the symmetry of the sixth and terminal link of the PUMA arm. And equation (10) holds because the second and third axes of the PUMA arm are parallel. Of the reduction from 126 to 39 unique Christoffel symbols, 61 eliminations are obtained with the general equations, 14 more with (9) and a further 12 with (10).

Step four requires differentiating the mass matrix elements with respect to the configuration variables. The means to carry out differentiation automatically have been available for some time.*

* The French authors seem to assume the use of Cristoffel symbols, while the American authors seem unaware of them. Corben and Stehle, in the 1950 edition of their text, derive the results required here; but the derivation is largely omitted from their 1960 edition.
time [Léger et al. 1976; MIT Matlab Group 1983]. Only the derivatives required after the simplification of step 3 need to be formed. Of the 126 derivatives possible when n = 6, 46 are required by the model of the PUMA arm. After the needed derivatives are formed and expanded into the Christoffel symbols, inertial constants that multiply common variable expressions are again combined.

Our method of model derivation is able to simplify to manageable form the complex sum-of-product expressions that are produced by symbolically carrying out the summations of Lagrange's equations. Simplification is in general a non-deterministic task that grows very rapidly with the number of terms in an equation; but the procedure presented is deterministic, with a cost that grows most rapidly as \( p^5 \), where \( p \) is the number of sum-of-product expressions in the largest individual kinetic energy matrix element. Our procedure has the virtue of producing explicit expressions for each component of the dynamic model: a result that is very useful for design analysis and that allows straight forward simplification by application of a sensitivity criterion.

Steps 2 through 4 of the above procedure were carried out by hand, requiring five weeks of rather tedious labor. To discover errors, the explicit solution was numerically checked against the RNE algorithm, extended to give \( B \) and \( C \) matrix elements individually in a manner similar to that of Walker and Orin. Over a range of configurations, the explicit solution of the PUMA dynamics agrees exactly with the RNE calculation. It is instructive to observe that the RNE algorithm was coded in 5 hours, 2% of the time required to develop the full explicit model.

3. Several Advantages Obtained from Decomposition of the Explicit Model

The explicit solution of PUMA dynamics shows two structural properties that can be used to advantage: a tremendous range between the largest and the smallest contributing terms within most equations, and the depend solely upon configuration of the \( A \), \( D \) and \( C \) matrix elements. Using the measured PUMA parameters an abbreviated dynamic model has been formed. This model is derived from the full PUMA model by eliminating all terms that are less than 1% as great as the greatest term within the same equation, or less than 0.1% as great as the largest constant term applicable to the joint. All of the elements of the \( A \), \( D \) and \( C \) matrices are retained: the significance test is applied on an equation by equation basis. The reduction in required calculations achieved via the significance test is roughly a factor of four, as shown in Table 1 above.

Observing that the \( A \), \( D \) and \( C \) matrix elements depend only on configuration, it is possible to decompose the calculation into configuration dependent and velocity or acceleration dependent components. Because configuration changes more slowly than velocity or acceleration, the configuration dependent components may be computed at a slower rate [Khatib 1985; Isaguirre and Paul 1985]. Shown in Table 2 is the evaluation rate of the PUMA 560 dynamics that can be achieved with 100,000 floating point operations per second, the approximate speed of a PDP-11. In the first case the entire model is recomputed in each pass; in the second case the \( A \), \( D \) and \( C \) matrix elements are computed only once for every four iterations of the multiplication by velocity and acceleration vectors. This partitioning of the dynamic calculation reduces the pace of computing the configuration dependent terms by one third; but increases the pace of computing the velocity and acceleration dependent terms by a factor of two and one half. The advantage of this decomposition applies equally well to the calculation of forward dynamics for simulation, where tessellation is the step size rather than servo interval and the cost is run time rather than bounded computing power.

Table 2. PUMA 560 Dynamic Model Evaluation Rate Attainable with 100K FLOPS.

<table>
<thead>
<tr>
<th>Method</th>
<th>Rate of Evaluation of Configuration Dependent Terms</th>
<th>Rate of Computation of Torque</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation of the Full Model Each Iteration</td>
<td>78 Hz</td>
<td>78 Hz</td>
</tr>
<tr>
<td>Evaluation of the Configuration Dependent Terms once during every four Evaluations of the Velocity and Acceleration Dependent Terms</td>
<td>50 Hz</td>
<td>200 Hz</td>
</tr>
</tbody>
</table>

A final decomposition to be considered is that for multiprocessing, an issue likely to become more important. The recursive formulations are well suited to pipeline computation, but poorly suited to multiprocessor computation. For the recursive algorithms, the number of calculations that can be performed by cooperating processors is small in relation to the volume of communication that is required. Using an explicit model the blocks of parallel computation can be made much larger, and the ratio of computation to communication correspondingly higher. The decomposition into configuration dependent and velocity or acceleration dependent components is particularly suitable for multiprocessing and has been implemented at the Stanford Artificial Intelligence Laboratory [Khatib 1985].

4. The Utility of an Explicit Model for Dynamic Simulation

Walker and Orin have demonstrated the use of the RNE algorithm in the calculation of forward dynamics for simulation. By taking advantage of the symmetry of the kinetic energy matrix they have reduced the model order that must be considered in successive applications of the RNE [Walker and Orin 1983]. The RNE algorithm has also been used to compute dynamics for simulation in fields outside of robotics [Benati et al. 1986; Kooskekanani et al. 1983]. Presented in table 3 are the number of calculations required to compute the elements of the kinetic energy matrix using Walker and Orin's method, using the full PUMA model, using the simplified model reported in [Isaguirre and Paul 1985], and using the abbreviated (1% significance criterion) model. The analytic models all show a tremendous advantage over the RNE algorithm.

Table 3. Calculations Required to determine the Kinetic Energy Matrix Elements for a PUMA 560 Arm.

<table>
<thead>
<tr>
<th>Method</th>
<th>Calculations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walker and Orin</td>
<td>2737</td>
</tr>
<tr>
<td>Full Explicit Model</td>
<td>278</td>
</tr>
<tr>
<td>Isaguirre and Paul Simplified Model</td>
<td>58</td>
</tr>
<tr>
<td>Abbreviated Explicit Model</td>
<td>25</td>
</tr>
</tbody>
</table>

5. Measurement of the PUMA 560 Dynamic Parameters

The link parameters required to calculate the elements of \( A \), \( B \), \( C \) and \( g \) in equation (1) are mass, location of the center...
of gravity and the terms of the inertia dyadic. The wrist, link three and link two of a PUMA 560 arm were detached in order to measure these parameters. The mass of each component was determined with a beam balance; the center of gravity was located by balancing each link on a knife edge, once orthogonal to each axis; and the diagonal terms of the inertia dyadic were measured with a two wire suspension.

The motor and drive mechanism at each joint contributes to the inertia about that joint an amount equal to the inertia of the rotating pieces magnified by the gear ratio squared. The drives and reduction gears were not removed from the links, so the total motor and drive contribution at each joint was determined by an identification method. This contribution is considered separately from the \( I_{ix} \) term of the link itself because the motor and drive inertia seen through the reduction gear does not contribute to the inertial forces at the other joints in the arm. The motors were left installed in links two and three when the inertia of these links were measured, so the effect of their mass as the supporting links move is correctly considered. The gyroscopic forces imparted by the rotating motor armatures is neglected in the model, but the data presented below include armature inertia and gear ratios, so these forces can be determined.

The parameters of the wrist links were not directly measured. The wrist itself was not disassembled. But the needed parameters were estimated using measurements of the wrist mass and the external dimensions of the individual links. To obtain the inertial terms, the wrist links were modeled as thin shells.

**Measurement of Rotational Inertia**

The two wire suspension shown in Figure 1 was used to measure the \( I_{ix} \), \( I_{iy} \), and \( I_{iz} \) parameters of links two and three. With this arrangement a rotational pendulum is created about an axis parallel to and halfway between the suspension wires. The link’s center of gravity must lie on this axis. The two wire suspension method of measuring the rotational inertia requires knowledge of parameters that are easily measured: the mass of the link, the location of the center of gravity, the distance from the wire attachment points to the axis of rotation, the length of the wires, and the period of rotational oscillation. The inertia about each axis is measured by configuring the link to swing about that axis. Rotational oscillation is started by twisting and releasing the link. If one is careful when releasing the link, it is possible to start fundamental mode oscillation without visibly exciting any of the other modes. The relationship between measured properties and rotational inertia is:

\[
I = M_g r^2 \omega^2 + I
\]

where:
- \( I \) is the inertia about the axis of rotation;
- \( M_g \) is the weight of the link;
- \( r \) is the distance from each suspension wire to the axis of rotation;
- \( \omega \) is the oscillation frequency in radians per second;
- \( I \) is the length of the supporting wires.

**Measurement of the Motor and Drive Inertia**

A parameter identification method was used to learn the total rotational inertia at each joint. This inertia includes the effective motor and drive inertia and the contribution due to the mass of the arm. To make this measurement our control system was configured to command a motor torque proportional to displacement, effecting a torsional spring. By measuring the period of oscillation of the resultant mass-spring system, the total rotational inertia about each joint was determined. By subtracting the arm contributions, determined from direct measurements, from the measured total inertia, the motor and drive inertial contributions were found.

**Measurement Tolerance**

A tolerance for each direct measurement was established as the measurement was taken. The tolerance values are derived from the precision or smallest graduation of the measuring instrument used, or from the repeatability of the measurement itself. The tolerances are reported where the data are presented. The tolerance values assigned to calculated parameters were determined by RMS combination of the tolerance assigned to each direct measurement contributing to the calculation. The inertia dyadic and center of gravity parameters of link 3 were measured with the wrist attached; the values reported for link 3 alone have been obtained by subtracting the contribution of the wrist from the total of link 3 plus wrist. Tolerance values are reported with the values for link 3 plus wrist, as these are the original measurements.

**6. The Measured PUMA 560 Parameters**

The mass of links 2 through 6 of the PUMA arm are reported in Table 4; the mass of link 1 is not included because that link was not removed from the base. Separately measured mass and inertia terms are not required for link one because that link rotates only about its own Z axis.

**Table 4. Link Masses (kilograms; ±0.01 + 1%)**

<table>
<thead>
<tr>
<th>Link</th>
<th>Mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link 2</td>
<td>17.40</td>
</tr>
<tr>
<td>Link 3</td>
<td>4.80</td>
</tr>
<tr>
<td>Link 4*</td>
<td>0.82</td>
</tr>
<tr>
<td>Link 5*</td>
<td>0.34</td>
</tr>
<tr>
<td>Link 6*</td>
<td>0.09</td>
</tr>
<tr>
<td>Link 3 with Complete Wrist</td>
<td>6.04</td>
</tr>
<tr>
<td>Detached Wrist</td>
<td>2.24</td>
</tr>
</tbody>
</table>

* Values derived from external dimensions; ±25%.

The positions of the centers of gravity are reported in Table 5. The dimensions \( r_x \), \( r_y \) and \( r_z \) refer to the x, y and z coordinates...
of the center of gravity in the coordinate frame attached to the link. The coordinate frames used are assigned by a modified Denavit-Hartenberg method [Craig 85]. In this variant of the Denavit-Hartenberg method, frame \( i \) is attached to link \( i \), and axis \( Z_i \) lies along the axis of rotation of joint \( i \). The coordinate frame attachments are shown in Figure 2; they are located as follows:

- Link 1: \( Z \) axis along the axis of rotation, \( +Z \) up; \( +Y1 \) \( \parallel \) \( +Z2 \).
- Link 2: \( Z \) axis along the axis of rotation, \( +Z \) away from the base; \( X-Y \) plane in the center of the link, with \( +X \) toward link 3.
- Link 3: \( Z3 \parallel Z2 \); \( X-Y \) plane is in the center of link 3; \( +Y \) is away from the wrist.
- Link 4: The origin is at the intersection of the axes of joints 4 and 6; \( +Z4 \) is along the axis of rotation and directed away from link 2; \( +Y4 \parallel +Z3 \) when joint 4 is in the zero position.
- Link 5: The origin coincides with that of frame 4; \( +Z5 \) is directed away from the base; \( +Y5 \) is directed toward link 2 when joint 5 is in the zero position.
- Link 6: The origin coincides with that of frame 4; when joints 5 and 6 are in the zero position frame 6 is aligned with frame 4.
- Wrist: The dimensions are reported in frame 4.

The inertia dyadic and effective motor and drive inertia terms are reported in Table 6. For each link, the coordinate frame for the inertia dyadic terms is placed at the center of gravity, parallel to the attached frame used in Table 5. The tolerances assigned to these measurements are shown in parentheses. No tolerance is associated with the value of \( I_{xx} \) for link one because this value was not directly measured; it was computed backwards from the measured total joint inertia. It is not important to distinguish \( I_{xx} \) from the \( m1 \times r1^2 \) term or from the motor and drive inertia at joint one because these contributions are neither configuration dependent nor appear in any term other than \( \alpha_i \). The total link 1 inertia measured by the identification method is the sum of \( I_{xx1} \) and \( I_{motor} \) in Table 6.

Table 5. Centers of Gravity. (meters ±0.003)  
<table>
<thead>
<tr>
<th>Link</th>
<th>( r_x )</th>
<th>( r_y )</th>
<th>( r_z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link 2</td>
<td>0.068</td>
<td>0.006</td>
<td>-0.016</td>
</tr>
<tr>
<td>Link 3</td>
<td>0</td>
<td>-0.070</td>
<td>0.014</td>
</tr>
<tr>
<td>Link 3 With Wrist</td>
<td>0</td>
<td>-0.143</td>
<td>0.014</td>
</tr>
<tr>
<td>Link 4*</td>
<td>0</td>
<td>0</td>
<td>-0.019</td>
</tr>
<tr>
<td>Link 5*</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Link 6*</td>
<td>0</td>
<td>0</td>
<td>0.032</td>
</tr>
<tr>
<td>Wrist</td>
<td>0</td>
<td>0</td>
<td>-0.004</td>
</tr>
</tbody>
</table>

* Values derived from external dimensions; ±25%.

The effective torsional spring method of inertia measurement was applied at each joint. The motor and drive inertia, \( I_{motor} \), were found by subtracting the inertial contribution due to the arm dynamics, known from direct measurements, from the total inertia measured. The uncertainty in the total inertia measurement is somewhat higher at joint one because of the larger friction at that joint. It was necessary to add positive velocity feedback (damping factor -0.1) to cause joint one to oscillate for several cycles.

The gear ratios, maximum motor torque, and break away torque for each joint of the PUMA is reported in Table 7. The maximum motor torque and break away torque values have been taken from data collected during our motor calibration process. The current amplifiers of the Unimate controller are driven by 12 bit D/A converters, so the nominal torque resolution can be obtained by dividing the reported maximum joint torque by 2048.

Table 7. Motor and Drive Parameters

<table>
<thead>
<tr>
<th>Link</th>
<th>Joint 1</th>
<th>Joint 2</th>
<th>Joint 3</th>
<th>Joint 4</th>
<th>Joint 5</th>
<th>Joint 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gear Ratio</td>
<td>62.81</td>
<td>107.82</td>
<td>83.01</td>
<td>83.01</td>
<td>79.91</td>
<td>79.75</td>
</tr>
<tr>
<td>Maximum Torque [Nm]</td>
<td>97.60</td>
<td>100.44</td>
<td>88.42</td>
<td>74.32</td>
<td>30.13</td>
<td>31.38</td>
</tr>
<tr>
<td>Break Away Torque [Nm]</td>
<td>0.32</td>
<td>0.54</td>
<td>0.31</td>
<td>0.26</td>
<td>1.30</td>
<td>1.21</td>
</tr>
</tbody>
</table>

7. Conclusion

Explicit dynamic models of complex manipulators are attainable. The PUMA 560 arm is as complex as any 6 dof arm with a spherical wrist, yet a deterministic simplification procedure has produced an explicit model that is more economical than the
References


J. Burdick, Private Communication.


APPENDIX

The Full Expressions for the Forces of Motion of a Puma 600 Arm

In the following tables the expressions for the elements of the A, B and C matrices and the g vector are presented. These expressions are made in terms of constants which have units of inertia or torque, and trigonometric terms that are functions of the joint angles. We have abbreviated the trigonometric functions by writing $S2$ to mean $\sin(q_2)$ and $C5$ to mean $\cos(q_5)$. When a trigonometric operation is applied to the sum of several angles we write $C23$ to mean $\cos(q_2 + q_3)$ and $S223$ to mean
\[ \sin(q_2 + q_3 + q_4) \]. And when a product of several trigonometric operations on the same joint variable appears we write CC2 to mean \( \cos(q_2) \cdot \cos(q_3) \) and CC4 to mean \( \cos(q_1) \cdot \sin(q_2) \). These final abbreviations, CC5 etc., are considered to be factorizations; and the cost of computing these terms is included in the totals reported above.

The position of zero joint angles and coordinate frame attachment to the PUMA arm are shown in Figure 2 above. The modified Denavit-Hartenberg parameters, assigned according to the method presented in [Craig 85], are listed in Table A1.

### Table A1. Modified Denavit–Hartenberg Parameters

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \alpha_{i-1} ) (degrees)</th>
<th>( \beta_i )</th>
<th>( \theta_{i-1} ) (meters)</th>
<th>( d_i ) (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>( q_1 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-90</td>
<td>( q_2 )</td>
<td>0</td>
<td>2.435</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>( q_3 )</td>
<td>.4318</td>
<td>-0.0934</td>
</tr>
<tr>
<td>4</td>
<td>90</td>
<td>( q_4 )</td>
<td>-.0203(( \bar{q}_5 ))</td>
<td>.4331</td>
</tr>
<tr>
<td>5</td>
<td>-90</td>
<td>( q_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>90</td>
<td>( q_6 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The equations of the PUMA model constants are presented in Table A2; these constants appear in the dynamic equations of Tables A4 through A7. \( I_{xx} \) and \( m_i \) refer to the second moment of link \( i \) about the \( x \)-axis of frame \( i \) and the mass of link \( i \) respectively. The terms \( \alpha_i \) and \( d_i \) are the Denavit-Hartenberg parameters. Terms of the form \( r_{ii} \) are the offsets to the center of gravity of link \( i \) in the \( i^{th}\) coordinate frame. In Table A3 the values of the model constants are listed. The terms \( I_{xx} \) are the motor and drive train contribution to inertia at joint \( i \).

The equations for the elements of the kinetic energy matrix, \( A(q) \), are presented in Table A4. \( A(q) \) is symmetric, so only equations for elements on and above the matrix diagonal are presented.

The equations for the elements of the Coriolis matrix, \( B(q) \), are presented in Table A5. The Coriolis terms have been left in the form of a three dimensional array, with a convention for the indices that matches that of the Christoffel symbols. Element \( \beta_{ij} \) multiplies \( \dot{q}_i \) and \( \dot{q}_j \) to give a contribution to the torque at joint \( i \). The Coriolis matrix may also be written as a 6 \( \times \) 15 array, where the 15 columns correspond to the 15 possible combinations of joint velocities. The equations for the elements of the centrifugal matrix, \( C(q) \), are presented in Table A6. And the equations for the terms of the gravity vector, \( g(q) \), are presented in Table A7.

A load may be represented in this model by attaching it to the \( 6^{th} \) link. In the model the \( 6^{th} \) link is assumed to have a center of gravity on the axis of rotation, and to have \( I_{xx6} = \rho_{xx6} \); these restrictions extend to a load represented by changing the \( 6^{th} \) link parameters. A more general, though computationally more expensive, method of incorporating a load in the dynamic calculation is presented in [Ishiguro and Paul 1985].

### Table A2. Expressions for the Constants Appearing in the Equations of Forces of Motion.

#### Part I. Inertial Constants

\[ I_1 = \sum_{i=1}^{7} m_i r_{i,2}^2 + (m_i + m_{i+1} + m_{i+2}) \dot{q}_i^3 + 2m_i r_{i,2} (m_i + m_{i+1} + m_{i+2}) \dot{q}_i \dot{q}_{i+1} + 2m_i (d_i + d_{i+1}) \dot{q}_i + \sum_{i=1}^{7} I_{xx} = \rho_{xx6} \]

### Table A3. Computed Values for the Constants Appearing in the Equations of Forces of Motion.

The inertial constants have units of kilogram meters-squared

<table>
<thead>
<tr>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
<th>( I_5 )</th>
<th>( I_6 )</th>
<th>( I_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.13</td>
<td>1.38</td>
<td>3.72x10^{-1}</td>
<td>0.31x10^{-1}</td>
<td>2.98x10^{-1}</td>
<td>0.29x10^{-1}</td>
<td>2.28x10^{-2}</td>
</tr>
</tbody>
</table>

### Table A4. Gravitational Constants

\[ g_1 = g \cdot (m_1 + m_2 + m_3) \cdot a_2 + g \cdot m_2 \cdot a_3 \]

### Table A5. Computed Values for the Gravitational Constants

The gravitational constants have units of newton meters

<table>
<thead>
<tr>
<th>( g_1 )</th>
<th>( g_2 )</th>
<th>( g_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-37.2</td>
<td>1.02</td>
<td>-2.25x10^{-2}</td>
</tr>
</tbody>
</table>

516
Table A.4. The expressions giving the elements of the kinetic energy matrix.
(The Abbreviated Expressions have units of kg-m².)

\[ a_{10} = I_{m1} + I_1 + I_2 + A_2 + I_3 + S\theta S3 + I_{10} + A\theta C3 + I_{11} + SC2 + I_{20} + S\theta S3 + \theta C4 + 1 + I_2 + A\theta C2 + SC3 + I_{22} + 11 + SC2 + I_{12} + I_1 + SC2 + I_{23} + I_{24} + SC3 + I_{25} + SC5 + C2 + C4 + S5 \]
\[ + I_{42} + A_2 + C\theta S3 + C\theta S5 + C\theta S4 + S5 + I_{11} + C\theta S3 + S\theta S4 + S5 + I_{11} + A_1 + SC2 + I_{23} + I_{24} + I_1 + SC2 + I_{25} + I_{10} + C2 + C4 \]
\[ + I_{12} + S\theta S3 + C\theta S5 + C\theta S4 + S5 \]
\[ \approx 2.37 + 13.8 + C2 + 0.30 + S\theta S3 + 7.44 \times 10^{-1} \times C2 + S2 + S3 \]

Table A.5. The expressions giving the elements of the Coriolis matrix.
(The Abbreviated Expressions have units of kg-m².)

\[ b_{112} = 2 \times (-I_{1} + S\theta C2 + I_{1} + C2 + I_{1} + SC2 + C2 + I_{1} + SC2 + C2 + I_{1} + SC2 + C2 + I_{1} + SC2 + C2) \]
\[ + I_{11} + (C2 + C5 + S\theta S3 + C\theta S4 + S5) + I_{11} + SC2 + C5 + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ + I_{11} + (1 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5) + I_{11} + (1 + C2 + C5 + S\theta S3 + C\theta S4 + S5) \]
\[ \approx -2.76 + S\theta C3 + 7.44 \times 10^{-1} \times C2 + SC3 + 0.60 \times SC2 + 2.13 \times 10^{-1} \times (1 + SC2 + S2 + S3) \]

\[ b_{113} = 2 \times (I_{1} + C2 + C3 + I_{1} + S\theta C3 - I_{1} + C2 + SC2 + I_{1} + SC2 + C2) \]
\[ + I_{11} + (2 + SC2 + C3 + 1 + 1 + C2 + C5 + S\theta S3 + C\theta S4 + S5) + I_{11} + C2 + C3 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ + I_{11} + (1 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5) + I_{11} + (1 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5) \]
\[ + I_{11} + (1 + C2 + C5 + S\theta S3 + C\theta S4 + S5) + I_{11} + (1 + C2 + C5 + S\theta S3 + C\theta S4 + S5) \]
\[ \approx 7.44 \times 10^{-1} \times C2 + C3 + 0.60 \times SC2 + 2.20 \times 10^{-1} \times C2 + C2 + SC2 - 2.13 \times 10^{-1} \times (1 + SC2 + S2 + S3) \]

\[ b_{114} = 2 \times (-I_{1} + S\theta S3 + C\theta S4 + S5 + I_{11} + C2 + C3 + S\theta S4 + S5 + I_{11} + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ - I_{11} + C2 + C3 + S\theta S4 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ - I_{11} + C2 + C3 + S\theta S4 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ \approx 2.50 \times 10^{-1} \times C2 + S\theta S4 + S5 + 8.0 \times 10^{-1} \times C4 + S5 \]
\[ - 2.48 \times 10^{-1} \times C2 + C3 + C4 + S4 \]

\[ b_{115} = 2 \times (I_{1} + S\theta S3 + C\theta S4 + S5 + I_{11} + C2 + C3 + S\theta S4 + S5 + I_{11} + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ - I_{11} + C2 + C3 + S\theta S4 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ - I_{11} + C2 + C3 + S\theta S4 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ \approx 2.50 \times 10^{-1} \times C2 + S\theta S4 + S5 + 8.0 \times 10^{-1} \times C4 + S5 \]
\[ - 2.48 \times 10^{-1} \times C2 + C3 + C4 + S4 \]

\[ b_{116} = 6.0 \]

\[ b_{117} = 2 \times (-I_{1} + S\theta S3 + I_{1} + S\theta S3 + C\theta S4 + S5 + I_{11} + C2 + C3 + S\theta S4 + S5 + I_{11} + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ + I_{11} + C2 + C3 + S\theta S4 + SC2 + SC2 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ + I_{11} + SC2 + C4 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ + I_{11} + SC2 + C4 + C5 + S\theta S3 + C\theta S4 + S5 \]
\[ \approx 2.67 \times 10^{-1} \times S\theta S3 \]

\[ b_{118} = -I_{11} + 2 \times S\theta S4 + S5 + I_{11} + S\theta S3 + (1 + S\theta S4) \]
\[ + I_{11} + S\theta S3 + (1 + S\theta S4) \]
\[ - I_{11} + SC2 + C4 + C5 + S\theta S3 + S\theta S4 \]
\[ + I_{11} + SC2 + C4 + C5 + S\theta S3 + S\theta S4 \]
\[ \approx 0.0 \]

\[ b_{119} = -I_{11} + (S\theta S3 + C\theta S4 + C\theta S4 + S5) \]
\[ \approx 0.0 \]

\[ b_{120} = -I_{11} + (C3 + S\theta S5 + C\theta S4 + C\theta S5) \]
\[ + I_{11} + S\theta S4 + S5 \]
\[ + I_{11} + S\theta S5 + S5 \]
\[ + I_{11} + S\theta S6 + S5 \]
\[ \approx 1.16 \times 10^{-1} \times S\theta S5 + S5 \]

\[ b_{121} = -I_{11} + S\theta S4 + S5 + I_{11} + S\theta S5 + S5 \]
\[ \approx 1.25 \times 10^{-1} \times S\theta S4 + S5 \]

\[ b_{122} = -I_{11} + S\theta S4 + S5 + I_{11} + S\theta S5 + S5 \]
\[ \approx 1.25 \times 10^{-1} \times S\theta S4 + S5 \]

\[ b_{123} = I_{11} + S\theta S4 + S5 \]
\[ \approx 0 \]

\[ b_{124} = 0 \]

\[ b_{125} = 0 \]

\[ b_{126} = 0 \]

\[ b_{127} = 0 \]

\[ b_{128} = 0 \]

\[ b_{129} = 0 \]

\[ b_{130} = 0 \]

\[ b_{131} = 0 \]

\[ b_{132} = 0 \]

\[ b_{133} = 0 \]

\[ b_{134} = 0 \]

\[ b_{135} = 0 \]

\[ b_{136} = 0 \]

\[ b_{137} = 0 \]
APPENDIX E

\[ b_{234} = 2 \cdot (-I_{14} \cdot C3 \cdot S4 \cdot S5 + I_{50} \cdot SC4 \cdot S5) \]
\[ = -2.48 \times 10^{-2} \cdot C3 \cdot S4 \cdot S5 \]
\[ b_{235} = 2 \cdot (-I_{15} \cdot S5 + I_{14} \cdot (C3 \cdot C4 \cdot C5 - S3 \cdot S5) \]
\[ + I_{50} \cdot SC4 + SC5 + I_{32} \cdot (C4 \cdot C5) \]
\[ = -2.50 \times 10^{-2} + 2.48 \times 10^{-2} \cdot (C3 \cdot C4 \cdot C5 - S3 \cdot S5) \]
\[ b_{236} = 0 \]
\[ b_{234} = b_{234} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]
\[ b_{235} = b_{235} \]

Table A6. The expressions for the terms of the centrifugal matrix.
(The Abbreviated Expressions have units of kg.m³)

\[ c_{11} = 0 \]
\[ c_{13} = -I_{14} \cdot C3 \cdot C4 \cdot C5 - S3 \cdot S5 + I_{50} \cdot SC4 + SC5 + I_{32} \cdot (C4 \cdot C5) \]
\[ = -2.50 \times 10^{-2} - 2.50 \times 10^{-2} \cdot S3 \cdot S5 \]
\[ c_{13} = -2.50 \times 10^{-2} + 2.50 \times 10^{-2} \cdot S5 \]
\[ c_{15} = -2.50 \times 10^{-2} + 2.50 \times 10^{-2} \cdot S3 \cdot S5 \]
\[ c_{17} = 0 \]

\[ c_{14} = c_{14} \]
\[ c_{16} = c_{16} \]
\[ c_{18} = c_{18} \]

Table A7. Gravity Terms.
(The Abbreviated Expressions have units of newton-meters)

\[ g_1 = 0 \]
\[ g_2 = g_{14} \cdot C3 \cdot C4 \cdot C5 \cdot S3 \cdot S5 + I_{50} \cdot SC4 + SC5 \]
\[ + I_{32} \cdot (C3 \cdot C4 \cdot C5 - S3 \cdot S5) \]
\[ = -37.2 \cdot C2 - 8.4 \cdot C3 \cdot S5 \]
\[ g_3 = g_{14} \cdot C3 \cdot C4 \cdot C5 \cdot S3 \cdot S5 + I_{50} \cdot SC4 + SC5 \]
\[ + I_{32} \cdot (C3 \cdot C4 \cdot C5 - S3 \cdot S5) \]
\[ = -8.4 \cdot C3 \cdot S5 \cdot 0.25 \cdot C2 \]

\[ g_4 = 0 \]