Order Number 1339225

Adaptive control and simulation of the PUMA 560 robot

Potocki, Jon Kyle, M.S.

The University of Arizona, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

ADAPTIVE CONTROL AND SIMULATION OF THE PUMA 560 ROBOT

by

Jon Kyle Potocki

A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1 9 8 9

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under the rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interested of scholarship. In all other instances, however, permission must be obtained by the author.

SIGNED: J. Kyle Ritola

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Hal L. Tharp — 11/7/89

Dr. H. Tharp
Professor of Electrical Engineering

Date

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Hal S. Tharp for his technical insight on control theory, and for procuring financial support. I would also like to thank Dr F. Cellier and Dr. R. Bahr for their critical readings of this manuscript.

Special thanks goes to my parents Kenneth and Donna Potocki, my grandparents Kyle and Mary Jane Anderson, and my brother and sister Dawn and Casey. Their love and support gave me the strength to pursue a graduate degree.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table page

ABSTRACT

The computed-torque algorithm is a popular model-based robot trajectory control scheme. Adding an adaptive mechanism to this scheme can improve the error tracking capabilities of the robot controller. This thesis describes the algorithms necessary to develop a computer simulation for the PUMA 560 robot arm. Several robot controllers are outlined with an emphasis on the computed-torque scheme. The PUMA simulation is used to analyze the error tracking capabilities of an adaptive computed-torque controller. Consideration is given to parameter mismatch, unmodeled friction, unknown loading, and path excitation. This thesis shows that even with inaccurate load knowledge the adaptive algorithm enhances the tracking capabilities of the controller.

CHAPTER ONE

Introduction

The word robot was first used in the satirical play *Rossum's Universal Robots* by Karel Capek and originated from the Czech word robotnic meaning serf. The play opened in Prague in 1921 and reflected Capek's reaction to the mechanized destruction of Europe during World War I. The romantic definition for robot is a machine that resembles a man in form and performs mechanical tasks upon verbal command. The majority of today's robots neither resemble humans nor perform mechanical tasks upon verbal command. Instead these robots are huge, slow moving arms that perform tasks such as spot and arc welding, spray painting, milling, as well as a variety of pick and place tasks. In addition, modern robots cannot formulate their own tasks and require complex programming to perform even the most mundane of jobs.

Robotics is an intensive research field involving disciplines such as physics, mathematics, mechanical engineering, electrical engineering, and systems engineering. Current robotics research areas include medical prosthesis and rehabilitation, space applications, vision and inspection systems, mobility, motion planning and collision avoidance, sensor integration, and decision making under uncertainty. A common theme within robotics research is the ability to better control the actions of the robot. Improved motion planning, vision systems, computer and sensor integration, decision making and dynamic modeling all result in stronger control schemes. Better control schemes expand the number of useful tasks that the robot is able to perform. The thrust of this thesis addresses topics necessary to simulate and control the PUMA 560 robot arm.

The Programmable Universal Machine for Assembly 560 or more commonly the PUMA 560 is a popular industrial robot. Built by Westinghouse Unimation, the PUMA line represented the third largest American robot line built in 1987 by supplying over 11 percent of the national market (Wolovich). Due to the large number of PUMA's on the market, the PUMA appears regularly in the robotics

research journals and conference papers. In addition, the dynamic structure of the PUMA 500 and 600 lines are equivalent which allows research work to cover a wide variety of arms. For these reasons the PUMA 560 arm was chosen for the simulation developed for this thesis.

Currently, model-based adaptive controllers dominate the robot control literature. Model-based implies that a full or reduced dynamic model of the robot is used to generate the control signal. Adaptive implies that the controller attempts to estimate some system parameters to improve the performance of the controller. J.J. Craig developed a model-based adaptive control scheme that is derived from stability theory (Craig 1988, Craig et. al. 1987). This study analyzes the feasibility of Craig's scheme compared with a non-adaptive model based scheme when simulated on a PUMA 560 robot arm.

Outline of Thesis

This thesis can be divided into two sections. Chapters Two, Three and Four discuss the topics needed to program a rigid body simulation for the PUMA 560. The information in these three chapters is tailored to the needs of the PUMA but many routines are broad enough for application to other robots. Chapters Five and Six develop the control algorithms and analyze the simulation results. A more detailed breakdown of the chapters is given below.

Chapter Two develops the basics of robot kinematics and the spatial relationships of the PUMA arm. This second Chapter describes the basic transformations that relate the six joints of the PUMA arm. In addition, a solution for calculating the joint angles from a desired hand position is given in Chapter Two. Chapter Three describes path planning basics and develops several simple path planning schemes. Path planning schemes are used to define trajectories the robot must follow to perform a task in freespace. The Fourth Chapter develops two algorithms used to describe the dynamics of robot arms. These two algorithms are the Lagrange-Euler method and the Newton-Euler method. This Chapter also contains the dynamic theory needed to program a dynamic robot simulation and implement

a model-based controller. Chapter Five outlines fundamental robot control design including classical, modern, model-based and adaptive techniques. The Fifth Chapter focuses on the popular model-based scheme called the computed-torque method. In addition, this Chapter also develops the adaptive computed-torque scheme developed by Craig. The PUMA simulation was used to compare the trajectory tracking capabilities of Craig's adaptive and the non-adaptive computed-torque controllers. The Sixth Chapter presents and discusses the results comparing the non-adaptive and adaptive computed torque schemes discussed in Chapter Five. These simulations consider the effects of parameter mismatch, unknown loading, friction, and persistent excitation of the path. Chapter Seven concludes the thesis. Appendix A gives the equations that solve the forward and inverse kinematics for the PUMA 560. Appendix B summarizes the programs written for the simulation. The final section lists the references used in this thesis.

The conventions used in this thesis are consistent with the conventions used in the robotics literature. The joint angles are represented in the text as θ's and q's in the simulation. The notation used in describing the algorithms found in this thesis reflect the Denavit-Hartenberg notation, as opposed to the spatial notation (Denavit and Hartenberg 1955). For information on robot algorithms derived using the spatial notation see Featherstone 1983a and 1987, Rodriguez and Kreutz 1988, and Kreutz and Lokshin 1988.

Finally, the simulation was written on a Sun 3/60 workstation as a library of Matrix_x functions. These functions were written as a set of subroutines in the Matrix_x language. These functions can be activated by using the 'define' command within the Matrix_x environment. Matrix_x was used because it reduced the time required to develop the simulation software and it provided fast graphing capabilities. The drawback was the speed required to simulate even the shortest trajectories. Future simulations would run faster if the Matrix_x files were transcribed into FORTRAN files to be appended to the Matrix_x library of functions.

CHAPTER TWO

Kinematics

Outine of Chapter

Kinematics is a branch of mechanics that deals with pure motion without regard for the forces and torques required to create that motion. In robotics a large portion of kinematics deals with finding the transformations needed to relate the various link coordinate frames. The kinematic problem can be subdivided into a forward problem and an inverse problem. In the forward problem transformations are found that relate joint angles with the Cartesian coordinates of the end-effector. In the inverse problem a set of joint angles are found that achieve a desired end-effector Cartesian position. This Chapter will give a general kinematic description of the PUMA 560, as well as explain methods for solving the forward and inverse problems.

General Kinematic Description of the PUMA 560

The PUMA 560 is a six degree of freedom revolute robot. A six degree of freedom robot has seven links, including the base link, and six joints connecting these links. Revolute means that each joint is rotational in nature as opposed to prismatic or sliding. The links are ordered numerically from base to tip. Thus, the zero link is the base, the first link connects this base with the second link and so on. Figure 2.1 shows the PUMA 560. Notice that the first three links control the position of the wrist, and the final three joints are found in the wrist itself. Consequently, it is possible to think of the PUMA arm as a human arm in that there are two degrees of freedom in the shoulder area, one in the elbow and three in the wrist. The last three joints in the wrist are described as connections between three zero length links.

There are several common coordinate frames and spaces used in describing the position of the PUMA arm. The base frame is the coordinate frame placed at

the base of the robot. The tool frame is the coordinate frame associated with the tip of a tool held in the robot's hand. In general, a coordinate frame is placed at each joint. In addition, coordinate frames are often placed upon objects that the robot might pick up, work with, or avoid. Sometimes if the robot has a vision system, a coordinate frame is located at the camera and the robot's motion is related to this camera frame. The type of coordinates (Cartesian, spherical, or cylindrical) specified for these frames depends upon the resulting ease of computation as defined by the robot structure. In the case of the PUMA 560, Cartesian coordinates require the least computations and are used to specify the frame coordinates.

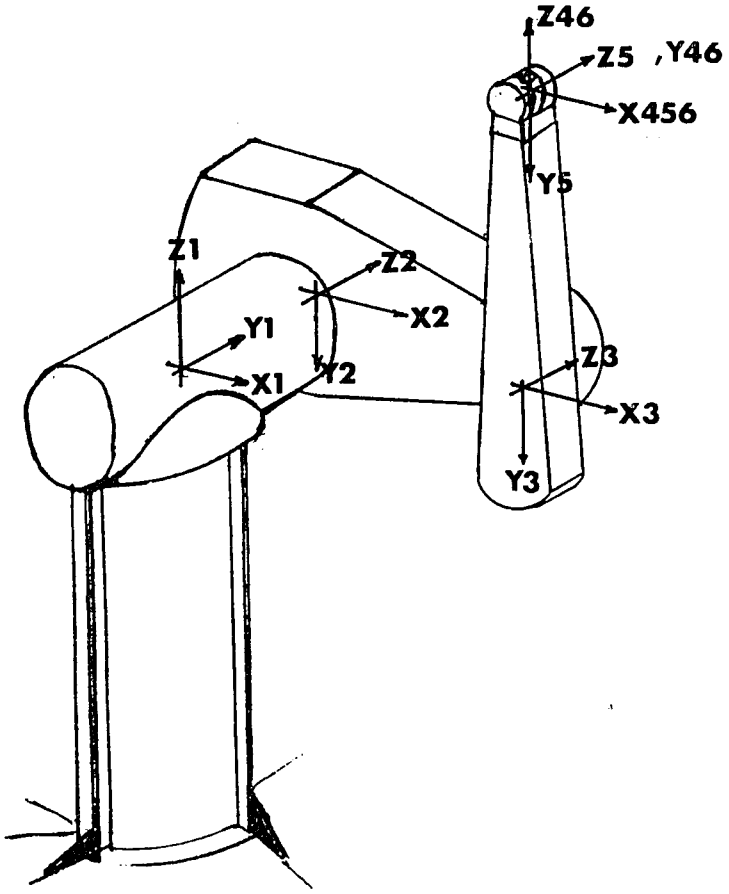

Figure 2.1 PUMA 560 robot arm

The workspace is the actual space throughout which the robot performs its tasks. Within the workspace, specific bounds exist that limit the allowed motion of the robot. For example, the arm can only reach out a finite distance from the base and each joint can only rotate a finite number of radians. Tables, walls or other objects within reach of the robot workspace represent regions the robot must avoid. These limitations and confinements define some of the boundaries that make up the workspace.

Three other important spaces are the actuator, joint and Cartesian spaces. The actuator space relates the configuration of the robot to the actuator or motor rotations needed to achieve that position. Because of the gearing or the positioning of the actuators with respect to the joints, the actuator values tend to disagree with joint angles. The joint space relates the configuration of the robot with the actual joint angles. Thus, if a desired robot position occurs when the joint angles are 1 radian, due to gearing, that 1 radian might correspond to 10 radians at the actuators. The Cartesian space defines the position of the robot wrist with respect to the base in terms of Cartesian coordinates. For the forward and inverse kinematics routines defined here, transformations are found that relate the joint and Cartesian spaces. The actuator space is not considered further, but in actual implementation, knowledge of gearing and actuator position is needed to relate desired joint torques with actuator torques.

Rotations, Translations and Transformations

Before analyzing the forward kinematic problem, a knowledge of how matrix transformations are used to relate robot links is needed. Figure 2.2 shows two coordinate frames and a single vector defined in the i+1 coordinate frame. The task is to transform the vector defined with the i+1 coordinates to a vector defined with the i coordinates. Conveniently, this transformation is broken up into a rotational part and a translational part.

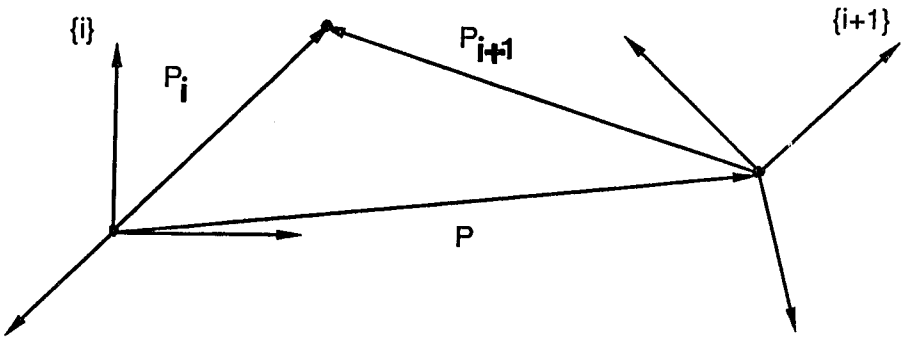

Figure 2.2 Transformation between two coordinate frames

Looking at the rotational part first, the rotation between the two coordinate frames is given by the matrix transformation:

$$P^i = R^i_{i+1} P^{i+1}. \tag{2.1}$$

Here P^i is the desired vector defined in the ith coordinate frame, R is a 3x3 matrix incorporating the rotation between the i and i+1 frames, and P^{i+1} is the vector in the i+1 frame. Three basic rotation matrices are used to define the possible Cartesian coordinate rotation described by R (Craig 1986, Wolovich 1986, Paul 1981a, Lee 1982). These three basic rotation matrices represent rotations about the x, y and z axis. These matrices are

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, \tag{2.2}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \tag{2.3}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.4}$$

From these three basic rotation matrices a wide variety of Cartesian rotations can be found. For example, a roll, pitch, and yaw rotation is given by a rotation of the i+1 frame about X_i by an angle γ, followed by the rotation of frame i+1 about Y_i by an angle β and finally, the rotation of frame i+1 about Z_i by an angle α. From equations 2.2-2.4 the roll, pitch, and yaw rotation matrix is given by (Craig 1986):

$$R_{rpy} = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

$$= \begin{pmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{pmatrix} \tag{2.5}$$

where $s\gamma = \sin\gamma, c\gamma = \cos\gamma$, and so on.

To complete the transformation illustrated in Figure 2.2, the translation between frame i and i+1 needs to be incorporated into the solution of equation (2.1). To facilitate the translation operation, equation (2.1) is augmented into a 4x4 transformation system. The R^i_{i+1} matrix is replaced with the homogeneous transformation matrix shown below (Lee 1982).

$$T^i_{i+1} = \begin{pmatrix} R^i_{i+1} & \hat{P}_{1x3} \\ f_{3x1} & s_{1x1} \end{pmatrix} \tag{2.6}$$

Here the subscripts describe the dimension ot each element, R^i_{i+1} is the rotation matrix from equation (2.1), $\hat{P}$ is the vector translation from frame i to frame i+1, f is a perspective transformation, and s is a scale factor. In robotics, the perspective transformation is always a zero vector and the scale factor is always one. To accommodate the change in system dimension, the P^{i+1} and P^i vectors have a one concatenated into the (4,1) position. The frame transformation is now given by:

$$P^i = T^i_{i+1}P^{i+1}. \tag{2.7}$$

Forward Kinematics

The forward kinematic problem is solved by finding the transformation matrices, T, that relate the various joint frames of the robot. This problem includes choosing the coordinate frames for each link, and the zero state of every frame. A convenient method for specifying frame position and finding the desired transformation matrices is given by the Denavit-Hartenberg representation (Denavit and Hartenberg 1955).

The Denavit-Hartenberg representation defines the link coordinate frames as follows. First, recall that the links are numbered from base to tip. Knowing this convention, the z_{i-1} axis are placed coincident with the i-1 joint axis, along the axis of motion of the ith frame. The x_{i-1} axis lies normal to z_{i-1} and in the direction of z_i. If z_{i-1} and z_i begin at the same point, as do the three links in the PUMA wrist, then the x_{i-1} is chosen normal to the plane between z_{i-1} and z_i. Finally, the y_i terms are chosen to complete the righthand coordinate rule. Figure 2.1 shows the coordinate frames and the zero position chosen for the PUMA arm. As each joint is revolute, the joint angles rotate about the z axis following the righthand rule convention.

The Denavit-Hartenberg representation gives four geometric quantities that completely describe the joint. These four parameters are (Craig 1986):

a_i = the distance from z_i to z_{i+1} measured along x_i;
α_i = the angle between z_i and z_{i+1} measured about x_i;
d_i = the distance from x_{i-1} to x_i measured along z_i;
θ_i = the angle between x_{i-1} and x_i measured about z_i.

Because the PUMA 560 is a revolute robot, the only parameters that change during motion are the θ_i joint angles. Table 2.1 gives one possible set of Denavit-Hartenberg parameters for the zero configuration given in Figure 2.1. (Armstrong et. al. 1986).

I	α_{I-1} (degrees)	θ_I	a_{I-1} (meters)	d_I (meters)
1	0	θ_1	0	0
2	-90	θ_2	0	0.2435
3	0	θ_3	0.4318	-0.0934
4	90	θ_4	-0.0203	0.4331
5	-90	θ_5	0	0
6	90	θ_6	0	0

Table 2.1 Denavit-Hartenberg parameters for PUMA 560

Once the Denavit-Hartenberg parameters are specified, finding the transformation matrices between joint frames becomes trivial. The transformation matrix changing i coordinates to i-1 coordinates is described as follows. First, there is a translation along z_i a distance of d_i. Second, there is a rotation of θ_i about z_i. Third, there is a translation of a distance a_{i-1} along x_i. Finally, there is a rotation of angle α_{i-1} about x_i. Equation (2.8) shows the generalized transformation matrix just described.

$$T_i^{i-1} = Rot_{x_i}(\alpha_{i-1})\, Trans_{x_i}(a_{i-1})\, Rot_{z_i}(\theta_i)\, Trans_{z_i}(d_i)$$

$$= \begin{pmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & c\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

From Table (2.1) and equation (2.8) the kinematic transformation matrices for the PUMA 560 are easily found. The transformation converting joint one coordinates to the zero frame is calculated as

$$T_1^0 = \begin{pmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.9}$$

The remaining joint transformations for the PUMA 560 are calculated in Appendix A.

Once the individual link transformations are known, all joints can be related using these individual transformations. The transformation converting joint 6 coordinates to the base frame coordinates is the product of all link transformations between link 6 and link 0.

$$T_6^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5 \tag{2.10}$$

Knowing T_6^0 allows the user to relate the specified joint angles θ_1 through θ_6 to the actual Cartesian coordinates of the wrist frame. This relation is the $\hat{P}$ vector contained within T_6^0 shown in (2.6). Thus, the Denavit-Hartenberg representation simplifies the forward kinematic problem.

Inverse Kinematics

While the forward problem is fairly simple, the inverse kinematic problem presents a multitude of solutions. In the inverse kinematic problem, one tries to find the joint angles necessary to achieve a desired wrist position and orientation. The problem is difficult because many solutions are available for a single wrist position. Figure 2.3 shows four possible inverse solutions for a single wrist position of the PUMA arm. The inverse solution is muddled further by the numerous Denavit-Hartenberg representations and mathematical solution techniques available. For example, Figure 2.1 represents only one possible zero state for the PUMA arm. Another zero state could have the wrist pointing parallel to link two or pointing down as opposed to up. Likewise, Table (2.1) gives only one possible set of Denavit-Hartenberg parameters for the PUMA. After solving the inverse kinematic equations for the PUMA, it becomes apparent that the d_2 and d_3 terms could be replaced with $\hat{d}_2 = d_2 + d_3$ and $\hat{d}_3 = 0$. Using these new $\hat{d}_2$ and $\hat{d}_3$ values will not change the overall forward or inverse kinematic solutions of the PUMA arm. There also exists a wide variety of mathematical techniques used in solving the PUMA's inverse kinematic problem. One result of these various techniques is that the literature provides a wide

variety of different but valid inverse kinematic solutions (Craig 1986, Lee 1982, Paul and Stevenson 1983).

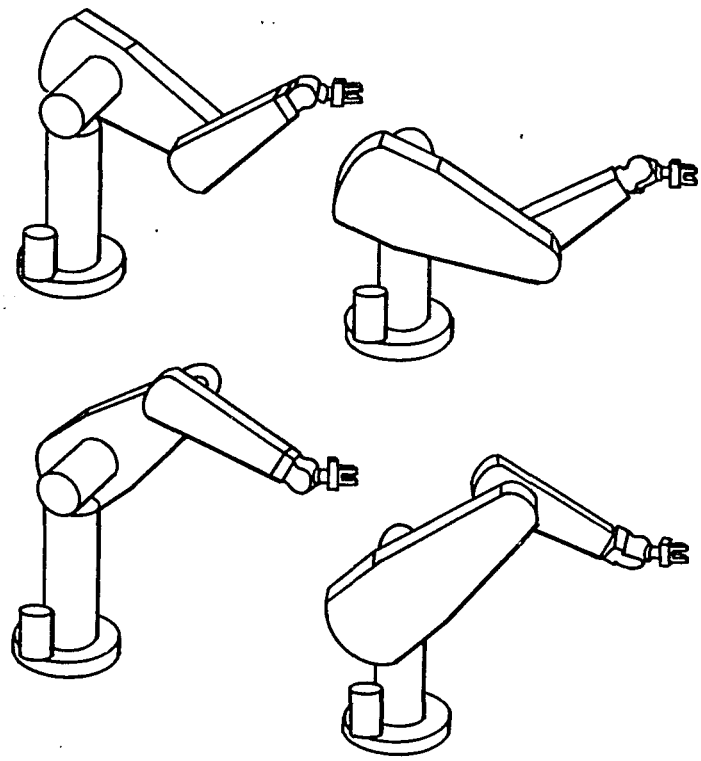

Figure 2.3 Four possible inverse solutions for PUMA

One method for solving the inverse problem for the PUMA requires decoupling the first three links from the last three links (Paul 1981b). The first three links control the Cartesian coordinates of the wrist, while the last three control the orientation of the wrist. From this decoupling a set of equations may be found that solve the inverse problem. The method given below simply uses matrix multiplications to arrive at the solution (Craig 1986). In the inverse kinematics problem the Cartesian position and orientation of the wrist are known, which imply the T_6^0

matrix is also known. From forward kinematics, the symbolic representations for all the transformation matrices are known. By equating the known T_6^0 matrix and the known symbolic transformation matrices, the inverse problem is solved by equating the proper terms. The first joint angle θ_1 can be found as follows. Assume that the desired Cartesian wrist position and rotation gives a transformation matrix with actual values

$$T_6^0 = \begin{pmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.11}$$

Let $\hat{T}_1^0$ and $\hat{T}_6^0$ be the symbolic transformation matrices found in equations (2.9) and (2.10), respectively. Multiplying the inverse of the $\hat{T}_1^0$ transformation with both the actual and symbolic transformation matrices gives

$$[\hat{T}_1^0]^{-1} T_6^0 = \hat{T}_6^1. \tag{2.12}$$

Equating the (2,4) element on both sides of equation (2.12) gives

$$-s\theta_1 p_x + c\theta_1 p_y = d_2 + d_3. \tag{2.13}$$

Equation (2.13) is a single equation with a single unknown, the solution of which is found by converting to polar coordinates. Let

$$p_x = \rho \cos\phi,$$

$$p_y = \rho \sin\phi,$$

where

$$\rho = \sqrt{p_x^2 + p_y^2},$$

$$\phi = \arctan\frac{p_y}{p_x}.$$

Combining these polar representations with equation (2.13) gives

$$c\theta_1 s\phi - s\theta_1 c\phi = \frac{d_2 + d_3}{\rho}.$$

Trigonometric angle identities are applied to give

$$\sin(\phi - \theta_1) = \frac{d_2 + d_3}{\rho}$$

$$\cos(\phi - \theta_1) = \pm\sqrt{1 - \frac{(d_2 + d_3)^2}{\rho^2}}$$

Now the inverse kinematic equation for the first joint angle is found as

$$\theta_1 = \arctan\frac{p_y}{p_x} - \arctan\frac{d_2 + d_3}{\pm\sqrt{p_x^2 + p_y^2 - (d_2 + d_3)^2}}. \tag{2.14}$$

Notice that two θ_1 values exist in (2.14) depending upon the sign chosen in the argument of the second arctan function. This choice of sign will effect all following inverse solutions, and gives rise to the multiple inverse solutions shown in Figure 2.3.

Here the arctan function refers to the ATAN2 function used commonly in robotics (see Craig 1986). The ATAN2 function is an arctangent function that preserves the quadrant of the output angle. If the quadrant of the angle is not preserved by the arctangent function, the resulting angle could be off by 180 degrees. The remainder of the inverse kinematic solution for the PUMA 560 is given in Appendix A.

CHAPTER THREE

Path Planning

Outline of Chapter

One criterion for measuring the usefulness of a robot arm is the arm's ability to follow a desired trajectory. Path planning deals with the positions, velocities, and accelerations needed to move from an initial point to a final point. In general, one considers the path of the tool frame with respect to the base frame. This trajectory is described in either the joint space or the Cartesian space. Path planning is an important topic as very little information is available from the manufacturer on most industrial trajectory generators (Tondu and El-Zorkany 1986).

Path planning requires more than simply writing equations that join sets of points in a smooth manner. Environmental constraints, manipulator constraints, and time-varying obstacle avoidance are but three topics within path planning (Lee 1989). Environmental constraints address the problems of time limitations, motion priority, collision constraints and path constraints. The time constraint is simply the amount of time the arm has to complete the desired motion. In motion priority a decision is made as to whether the obstacle or the robot arm must change motion to avoid collision. Collision constraints assume that the wrist is modeled as a sphere and tests whether another object will intersect with that sphere. Path constraints put a limitation on the maximum deviation a path may take from the predetermined path (Lee 1989).

Manipulator constraints deal with the smoothness of motion and torque limitations at high speeds. Jerky motion puts unwanted strain and wear on the robot arm. Therefore, it is desirable to move in a smooth fashion, where smooth implies continuous first and possibly second derivatives. The robot motors require a minimum torque to achieve motion, and have a maximum torque that they cannot exceed. Thus, the path must not require a greater torque than the allowed maximum or the robot will saturate (Lee 1989).

This Chapter explains the basics of joint and Cartesian path planning and introduces some of the algorithms used in the simulation. As the simulation does not use advanced planning schemes, obstacle avoidance, environmental constraints, and manipulator constraints are not considered further. For more information on obstacle avoidance see (Lee 1989, and Lozano-Perez and Wesley 1979). The persistent excitation of a path describes the amount of parameter information that an estimator can extract from the desired path. If the path is not persistently exciting, then parameter estimation schemes will not receive enough information to converge to the actual parameter values. The question of the persistent excitation of a path is addressed in Chapter Five.

Joint Space Schemes

In joint space path planning the trajectory is described by a set of joint angles, velocities, and accelerations. The end points of the path are specified in the Cartesian space. Inverse kinematics is used to convert these Cartesian points into joint angles. From the resulting joint angles, smooth functions are derived to form a path connecting the endpoints. In other cases a set of points might be specified in the Cartesian space as "via points". Now the robot must pass through or near all of the desired via points when following its trajectory. Once again inverse kinematics is used to convert all of the via points into joint angles. However, care must be taken in assuring that the path remains smooth in the vicinity of the via points (Craig 1986). The advantage of joint space path planning is that the results are in the joint space and ready for direct use in the dynamic calculations. The disadvantage of working in the joint space is that a simple motion in joint space is usually very complex in the Cartesian space. Thus, it is hard to visualize joint space trajectories in terms of robot motion.

Paul (1972) gives a list of constraints that should be met in order to successfully operate in the joint space. These constraints are: initial position, velocity, acceleration; lift-off position; set-down position; final position, velocity, acceleration; and time constraints due to motor characteristics. In addition, the extrema

for all joint trajectories must lie within the physical limits of each joint. Paul showed that each joint required an eighth order polynomial to satisfy the given set of constraints. As it is difficult to find the extreme point of an eighth order polynomial, other compound trajectories where devised. The idea is to split the trajectory up into several lower order polynomials. Three such schemes are the 4-3-4, the 3-5-3, and the 5-cubic trajectories. However, other authors suggest less complex schemes which achieve the desired results (Craig 1986, and Wolovich 1986). Two such schemes use quintic polynomials and linear functions with parabolic blends.

The solution of the quintic polynomial path is simply one of finding the six constant coefficients shown in equation (3.1).

$$\theta_i(t) = a_o + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{3.1}$$

To find these six constants, the initial position, velocity, and acceleration and the final position, velocity, and acceleration must be known. Call the initial and final positions, velocities, and accelerations θ_i, $\dot{\theta}_i$, $\ddot{\theta}_i$, $\theta_f, \dot{\theta}_f$, $\ddot{\theta}_f$, respectively, and call the total time t_f. Then the constants are given as (Craig 1986):

$$\begin{aligned}
a_0 &= \theta_i, \\
a_1 &= \dot{\theta}_i, \\
a_2 &= \frac{\ddot{\theta}_i}{2}, \\
a_3 &= \frac{20\theta_f - 20\theta_i - (8\dot{\theta}_f + 12\dot{\theta}_i)t_f - (3\ddot{\theta}_i - \ddot{\theta}_f)t_f^2}{2t_f^3}, \\
a_4 &= \frac{30\theta_i - 30\theta_f + (14\dot{\theta}_f + 16\dot{\theta}_i)t_f + (s\ddot{\theta}_i - 2\ddot{\theta}_f)t_f^2}{2t_f^4}, \\
a_5 &= \frac{12\theta_f - 12\theta_i - (6\dot{\theta}_f + 6\dot{\theta}_i)t_f - (\ddot{\theta}_i - \ddot{\theta}_f)t_f^2}{2t_f^5}.
\end{aligned} \tag{3.2}$$

If θ_f is the final destination point, then usually the final velocity and acceleration are zero. If, however, θ_f is actually the next via point, one must choose the velocities and accelerations to maintain a smooth curve. Also notice that differentiation of

equation (3.1) gives the desired joint velocities and joint accelerations for all points on the path.

A straight line is one of the simplest ways of connecting two points in space. The method of linear functions with parabolic blends uses straight lines with parabolic shaping near via points to maintain velocity continuity. The shape of the function is symmetric about the halfway point between the two end points. The actual shape of the region is a function of the desired acceleration specified by the user. Define θ_h and t_h as the halfway point's position and time values and θ_b and t_b as the blend position and time. Maintaining velocity continuity at the blend point requires

$$\ddot{\theta} t_b = \frac{\theta_h - \theta_b}{t_h - t_b}, \tag{3.3}$$

where $\ddot{\theta}$ is the acceleration near the endpoints of the blend. The value of the blend position is

$$\theta_b = \theta_i + \frac{1}{2}\ddot{\theta} t_b^2. \tag{3.4}$$

Knowing that $t_f = 2t_h$ and combining equations (3.3) and (3.4) gives

$$\ddot{\theta} t_b^2 - \ddot{\theta} t_b t_f + (\theta_f - \theta_i) = 0. \tag{3.5}$$

Bounds on t_b and $\ddot{\theta}$ are found as

$$t_b = \frac{t_f}{2} - \frac{\sqrt{\ddot{\theta}^2 t_f^2 - 4\ddot{\theta}(\theta_f - \theta_i)}}{2\ddot{\theta}}$$

$$\ddot{\theta} \geq \frac{4(\theta_f - \theta_i)}{t_f^2}. \tag{3.6}$$

The equation of motion become

$$\theta_i(t) = \begin{cases} \theta_i + \frac{\ddot{\theta}}{2} t^2 & 0 \leq t \leq t_b \\ \frac{\theta_f + \theta_i - \dot{\theta} t_f}{2} + \dot{\theta} t & t_b \leq t \leq t_f - t_b \\ \theta_f - \frac{\ddot{\theta} t_f^2}{2} + \ddot{\theta} t_f t - \frac{\ddot{\theta}}{2} t^2 & t_f - t_b \leq t \leq t_f. \end{cases} \tag{3.7}$$

Cartesian Path Planning

In Cartesian path planning the path is specified in Cartesian space and converted to the joint space for use in the dynamic routines. The advantage of the Cartesian schemes is the easy visualization of the robot trajectory. In obstacle avoidance it is easier to plan a trajectory in Cartesian space than from the obscure results produced in the joint space. The disadvantages of Cartesian plans are increased computations and the inability to convert all Cartesian points into the joint space. If the path is computed online, then the control computer must convert the Cartesian result to the joint space at the sampling rate of the system. The most common method of converting from the joint space to the Cartesian space makes use of the Jacobian. The Jacobian is a matrix used to convert joint coordinates into Cartesian coordinates. To convert the Cartesian path into the joint space, the Jacobian has to be inverted. When singularities occur in the Jacobian matrix it becomes impossible to convert to the joint space.

The quintic polynomial and linear function with parabolic blends methods are both valid Cartesian path planning schemes (Wolovich 1986). The difficulty with Cartesian schemes is the added burden of having to convert the result to the joint space. The following matrix transformation is used to convert between joint and Cartesian space.

$$\dot{X} = J(\theta)\,\dot{\theta} \tag{3.8}$$

Where $J(\theta)$ is termed the Jacobian and represents the change in variable from the joint space to the Cartesian space. From equation (3.8) and the inverse kinematics solution the necessary results for path planning are found by inverting the Jacobian.

$$\dot{\theta} = J(\theta)^{-1}\dot{X} \tag{3.9}$$

By differentiating equation (3.9) the desired joint accelerations are found as

$$\ddot{\theta} = \dot{J}(\theta)^{-1}\dot{X} + J(\theta)^{-1}\ddot{X} \tag{3.10}$$

There are several methods for arriving at the Jacobian matrix (Orin and Schrader 1984, and Paul and Stevenson 1983). However, the Jacobian is a function of θ and

does lose rank under certain conditions. Two types of singularities, also called knot points, are workspace boundary singularities and workspace interior singularities (Craig 1986). Boundary singularities occur when the arm is fully extended or folded back upon itself. Interior singularities occur when two or more joints line up causing a loss of freedom in the Cartesian space. Since the alignment of two or more joints can cause singularity in the Jacobian, there exist huge numbers of possible knot points in a path.

The second problem with Cartesian schemes is the increased computation time. Not only does the Jacobian have to be found, it has to be inverted and differentiated. In the case of the PUMA controller these computations need to occur every 0.028 seconds to keep up with the control scheme (Wolovich 1986). Some use interpolation schemes to escape the inversion problem (Whitney 1969). Others avoid the Jacobian computation and manipulate the PUMA kinematics equations to convert to the joint space (Featherstone 1983, and Elgazzar 1985). However, these last schemes still suffer from the same singularity problems as the Jacobian methods.

Straight Line Motion

Now what if the user wants the robot arm to follow a straight line path through Cartesian space? Obviously, a path can be found in Cartesian space and then converted to the joint space. If any singularities occur in the resulting path, it becomes unwise to use the Cartesian scheme. How then are the joint space schemes modified to achieve straight line motion? First, as the joint space schemes give results in the joint space, the singularities in the Cartesian space have no effect. To achieve straight line motion the path constraint about the line must be known. Next, the line is broken up into equally spaced via points and the joint space schemes are applied. If the maximum deviation exceeds the path constraint, then the number of via points is increased. Thus, a path is found in the joint space that approximates a straight line and avoids the singularities and computational problems of the Cartesian schemes (Taylor 1979).

CHAPTER FOUR

Dynamics

Outline of Chapter

Dynamics is the branch of mechanics that deals with the motion of bodies in conjunction with the forces that cause the motion. Dynamics plays a very important role in understanding the functioning of robots. First, knowledge of dynamics explains how forces and torques relate to desired joint positions, velocities, and accelerations. Second, a large number of control schemes use robot models to drive the robot arm along a desired trajectory.

Drawing the free body diagrams for the PUMA and solving the resulting forces on each link is very tedious, if not impossible. There are four general methods for solving the dynamic equations for a manipulator. These methods are the Lagrange-Euler, the Newton-Euler, the Recursive Lagrange-Euler (Hollerbach 1980), and the generalized d'Alembert (Lee et. al. 1983). For the purposes of this Chapter, only the Lagrange-Euler and Newton-Euler methods will be discussed. The Recusive Lagrange-Euler and generalized d'Alembert methods are not discussed as they add no additional theory needed to understand robot dynamics.

As with kinematics, there is a forward dynamic problem and an inverse dynamic problem. In the forward dynamics problem, torques are found that achieve a certain joint position, velocity, and acceleration. Forward dynamics is used in controls as a method for finding the torques necessary to follow a desired trajectory. In the inverse dynamics problem a set of torques is applied to the model and the resulting joint accelerations are found. The inverse dynamics model is used mostly in computer simulations to represent the manipulator.

This Chapter discusses the Lagrange-Euler and Newton-Euler methods for solving the forward and inverse dynamics problems. There is a short discussion on the use of explicit models in the case of the PUMA. Finally, the Chapter closes with a discussion of the friction models used in the simulation.

Lagrange-Euler Dynamic Solution

The Lagrange-Euler solution to manipulator dynamics results in a structure that is very useful for robot control design. The following equation gives the matrix results representing the PUMA's dynamic solution.

$$D(\theta)\ddot{\theta} + C(\dot{\theta}, \theta) + G(\theta) = \tau \tag{4.1}$$

In Equation (4.1), $D(\theta)$ is the manipulator mass matrix, $C(\theta, \dot{\theta})$ is the vector that represents the Coriolis and centrifugal terms, $G(\theta)$ is the vector of gravity terms, and τ is the vector of output torques. Since the PUMA is a six degree-of-freedom arm, τ is a six by one vector with one torque per joint. In this case τ is made up only of torques as the PUMA is revolute and forces only arise in prismatic joints.

One way of solving for the terms given in equation (4.1) is by Lagrangian dynamics (Paul 1981a, Lee 1982). The Lagrangian is defined as the difference between the kinetic energy Ke and the potential energy Pe

$$L = Ke - Pe. \tag{4.2}$$

Once the Lagrangian is known, the torque equation becomes

$$\tau_i = \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_i} - \frac{\partial L}{\partial \theta_i}. \tag{4.3}$$

The difficulty with the Lagrange-Euler method is the complexity and number of terms that result from solving equation (4.3).

Classically the kinetic energy of a system is given by $Ke = \frac{1}{2}mV^2$. For a manipulator, the position of a certain point is given by kinematics as

$$p = T_i\, p^i.$$

Differentiation gives the velocity of p.

$$\frac{dp}{dt} = \left(\sum_{j=1}^{i} \frac{\partial T_i}{\partial \theta_j} \dot{\theta}_j \right) p^i.$$

The kinetic energy needs the square of the velocity.

$$\left(\frac{dp}{dt}\right)^2 = Tr\left\{\sum_{j=1}^{i}\sum_{k=1}^{i}\frac{\partial T_i}{\partial\theta_j}p^i p^{i\prime}\frac{\partial T_i}{\partial\theta_k}\dot{\theta}_j\dot{\theta}_k\right\} \tag{4.4}$$

where Tr is the trace operator. Now the kinetic energy of a particle of mass dm is given as

$$dKe_i = \frac{1}{2}\left(\frac{dp}{dt}\right)^2 dm.$$

The total kinetic energy of the mass is found by integrating over the mass.

$$Ke_i = \int_m dKe_i = \frac{1}{2}Tr\left\{\sum_{j=1}^{i}\sum_{k=1}^{i}\frac{\partial T_i}{\partial\theta_j}\int_m p^i p^{i\prime}dm\frac{\partial T_i}{\partial\theta_k}\dot{\theta}_j\dot{\theta}_k\right\} \tag{4.5}$$

However, the integral of the p vectors over the mass is termed the inertia tensor and is given as

$$I_i = \int_m p^i p^{i\prime}dm,$$

such that,

$$Ke_i = \frac{1}{2}Tr\left\{\sum_{j=1}^{i}\sum_{k=1}^{i}\frac{\partial T_i}{\partial\theta_j}I_i\frac{\partial T_i}{\partial\theta_k}\dot{\theta}_j\dot{\theta}_k\right\}. \tag{4.6}$$

Equation (4.6) represents the individual link kinetic energies needed in the formulation of the Lagrangian. By summing over all of the links the entire kinetic energy Ke is found. The potential energy of the links is found from the knowledge of the link's center of mass, the link's mass and the link's position. It is found as

$$Pe_i = m_i g^T T_i \bar{p}_i, \tag{4.7}$$

where g_i is a vector representing the acceleration of gravity, and $\bar{p}_i$ is the center of gravity for link i. The total potential energy of the manipulator Pe is found by summing equation (4.7) over all links.

From equations (4.6) and (4.7) the Lagrangian is calculated and substituted into the torque equation (4.3). The result of performing the operations in equation (4.3) gives:

$$\tau_i = \sum_{j=i}^{6}\sum_{k=1}^{i} Tr\left(\frac{\partial T_j}{\partial\theta_k}I_j\frac{\partial T_j}{\partial\theta_i}\right)\ddot{\theta}_k$$

$$+\sum_{j=i}^{6}\sum_{k=1}^{j}\sum_{m=1}^{j} Tr\left(\frac{\partial^2 T_j}{\partial\theta_k\partial\theta_m} I_j \frac{\partial T_j}{\partial\theta_i}\right)\dot{\theta}_k\dot{\theta}_m - \sum_{j=i}^{6} m_j g' \frac{\partial T_j}{\partial\theta_i}\bar{p}_j \tag{4.8}$$

Equation (4.8) is the solution of the dynamic problem by use of the Lagrange-Euler formulation. Equation (4.8) can be put into matrix form by letting:

$$D(\theta)_{ij} = \sum_{p=max\ i,j}^{6} Tr\left(\frac{\partial T_p}{\partial\theta_j} I_p \frac{\partial T_{p'}}{\partial\theta_i}\right) \tag{4.9a}$$

$$C(\theta,\dot{\theta})_{ijk} = \sum_{p=max\ i,j,k}^{6} Tr\left(\frac{\partial T_p}{\partial\theta_j\partial\theta_k} I_p \frac{\partial T_{p'}}{\partial\theta_i}\right)\dot{\theta}_j\dot{\theta}_k \tag{4.9b}$$

$$G(\theta)_i = \sum_{p=i}^{6} -m_p g' \frac{\partial T_p}{\partial\theta_i}\bar{p}_p. \tag{4.9c}$$

Once the relations in equation (4.9) are known, the forward and inverse dynamic solutions are solved as follows. For the forward problem the desired θ, $\dot{\theta}$, and $\ddot{\theta}$ are known through the path planner or control scheme. These values are applied to give the same results as equation (4.1). For the inverse dynamics problem the applied torque, and previous θ and $\dot{\theta}$ values are assumed known. These values are used to solve the equation

$$\ddot{\theta} = D(\theta)^{-1}\left(\tau - C(\dot{\theta},\theta) - G(\theta)\right) \tag{4.10}$$

The new $\ddot{\theta}$ values are used to drive a numerical integration routine to update the θ and $\dot{\theta}$ values for the next iteration.

Although shown in equation form, a few words need to be said regarding the inertial characteristics shown above. First the mass manipulator matrix or kinetic energy matrix $D(\theta)$ is a symmetric positive definite matrix (Tourassis and Neuman 1985, Craig 1988). The D_{ij} terms represent the coupling between the i and j links, and the D_{ii} or diagonal terms represent the self-inertial coefficients. The degree of coupling between two joints is defined by the coefficient of coupling (Tourassis and Neuman 1985)

$$k_{ij} = \frac{|D_{ij}|}{D_{ii}D_{jj}}.$$

Since the mass matrix is positive definite, the coefficient of coupling falls between zero and one. As the coefficient of coupling tends towards one the joints become tightly coupled. As the coefficient tends towards zero the joints have very little inertial interaction. The values of the mass matrix depend on the kinematic arrangement of the robot. Thus, two links could be highly coupled in one configuration and totally decoupled in another. As a result, the most desirable manipulator mass matrix would be diagonal with constant self inertial terms. In this case the links are decoupled, the dynamics with respect to the joint acceleration would remain the same for all arm configurations, and the control algorithm would be very simple. Therefore, diagonal dominance of the mass matrix is a very desirable result. Diagonal dominance implies the coefficients of coupling are low as the self inertias are large. Robot design can produce a diagonally dominant mass matrix when the loading and kinematic factors are minimized. The loading factor is reduced when the link masses decrease from base to tip. The kinematic factor is reduced by proper placement of coordinate frames to reduce the net torque on a link (Tourassis and Neuman 1985). As a result of its large actuator inertias, the PUMA arm has a diagonally dominant mass matrix.

The Coriolis and centrifugal terms represent the velocity coupling between the joints of the manipulator. The Coriolis coefficients represent the velocity coupling of links j and k on link i, and are multiplied by $\dot{\theta}_j\dot{\theta}_k$. The centrifugal coefficients represent the velocity coupling of only link j on link i, and are multiplied by $\dot{\theta}_j\dot{\theta}_j$. Unlike the mass matrix, there are no centrifugal terms generated by link i appearing in the link i term. The Coriolis and centrifugal terms tend to be small and only become important when the arm moves at high speeds. In the case of the PUMA arm, tests show that the effects of the Coriolis and centrifugal terms are so small, with respect to the other torques in (4.1), that they may be ignored (Leahy et el. 1989).

The gravity coefficients arise from the potential energy within the individual links. The coefficients vary with the configuration of the manipulator. In the case of the rotational joints of the PUMA, if the axis of rotation is parallel with the gravitational field, the resulting gravity coefficient is zero (Tourassis and Neuman

1985). This fact explains why the gravitational coefficient at link one of the PUMA is always zero.

Equation (4.9) gives dynamic results based on the assumption that no loads are attached to the arm. In the case of a PUMA arm, a load is considered an object held by the hand at the sixth link. The effects of a load on the dynamic coefficients is additive and thus the load may be considered separately from (4.9) (Izaguirre and Paul 1985). The load is considered in many cases as a point mass located in the grasp of the sixth link (see Izaguirre and Paul 1985). It can also be approximated as an increase in mass of the sixth link (Armstrong et. al. 1986). Instead of modeling the load separate from the hand, the load is held by the sixth link and is considered to be an extension of that link.

Since the PUMA arm is driven by gears, the affects of loading are less significant than in the case of direct-drive arms that have no gearing. Sensitivity analysis shows that the load sensitivity is inversly proportional to the gear ratio k (Asada and Hara 1986)

$$S_i^L = \frac{\partial \tau}{\partial M_L} = \frac{W_i^M}{k_i}.$$

Here M_L is the load mass, W_i^M are terms sensitive to the load mass and i represents the specific link. The PUMA arm has gearing (see Armstrong et. al. 1986 for values) and is therefore less sensitive to loading than a direct drive arm or an arm with less gearing.

Finally, a special note on how actuator inertia values are added into the dynamic solution. The actuator or motor inertias are decoupled values that act only at the joints. Thus, the manipulator mass matrix may be updated such that

$$D_{new}(\theta) = D_{old}(\theta) + I_a, \tag{4.11}$$

where I_a is a diagonal matrix of the joint motor inertias to be added. In the case of the PUMA, these actuator inertias tend to dominate the manipulator mass matrix resulting in strong diagonal dominance. These large actuator inertias are a direct consequence of the large gear ratios found in the PUMA.

Newton-Euler Dynamics

The advantage of using the Newton-Euler method for forward dynamics over the Lagrange-Euler is the speed of computation. If n is the number of links, then the Newton-Euler method requires order n computations denoted $O(n)$, and Lagrange-Euler $O(n^3)$ computations (Luh et.al. 1980a). This reduction in the total number of computations is a result of the recursive nature of the Newton-Euler equations (see Luh et.al 1980a, McInnis and Liu 1986, Craig 1986).

The Newton-Euler method has an outward and an inward iteration (Craig 1986). In the outward iteration the joint velocities, and accelerations are found along with the forces and torques about the link's center of mass. These outward iterations begin at the base of the arm and move towards the tip. Once the outward iteration is complete, the inward iteration begins and moves from the tip back to the base. In the inward iteration, the joint torques and forces are solved from the information found in the outward iteration. An advantage of the Newton-Euler method is that the information for each link is found in terms of the joint coordinates of each link. In the Lagrange-Euler method the solution requires differentiation of the transformation matrices relating all joints to a common reference. The Newton-Euler method requires only the rotations and translations relating successive coordinate frames. As a result, the Newton-Euler method is very simple and computationally efficient.

Newton's equation states that the force of an object is proportional to the mass times the acceleration of the center of mass of the object.

$$F = ma_c \tag{4.12}$$

Euler's equation relates the torque about the object's center of mass to the object's angular velocity, acceleration, and inertia.

$$N = I_c\dot{\omega} + \omega \times I_c\omega \tag{4.13}$$

Here I_c is the inertia tensor of the center of mass, ω is the angular velocity, $\dot{\omega}$ is the angular acceleration, and N is the resulting torque. Note that I_c is only 3x3 in this

case and is diagonal since it is referenced to the center of mass. From equations (4.12) and (4.13), the Newton-Euler forward dynamic solution, where i numbers the link, is given as follows (Craig, 1986).

Outward Iteration i: 0 to 5

$$\omega_{i+1} = R_i^{i+1}\omega_i + \dot{\theta}_{i+1}Z_{i+1}$$
$$\dot{\omega}_{i+1} = R_i^{i+1}\dot{\omega}_i + R_i^{i+1}\omega_i \times \dot{\theta}_{i+1}Z_{i+1} + \ddot{\theta}_{i+1}Z_{i+1}$$
$$\dot{v}_{i+1} = R_i^{i+1}(\dot{\omega}_i \times p_{i+1}^i + \omega_i \times (\omega_i \times p_{i+1}^i) + \dot{v}_i)$$
$$\dot{v}_{ci+1} = \omega_{i+1} \times p_{ci+1} + \omega_{i+1} \times (\omega_{i+1} \times p_{ci+1}) + \dot{v}_{i+1}$$
$$F_{i+1} = m_{i+1}\dot{v}_{ci+1}$$
$$N_{i+1} = I_{ci+1}\dot{\omega}_{i+1} + \omega_{i+1} \times I_{ci+1}\omega_{i+1}$$

Inward Iteration i: 6 to 1

$$f_i = R_{i+1}^i f_{i+1} + F_i$$
$$n_i = N_i + R_{i+1}^i n_{i+1} + p_{ci} \times F_i + p_{i+1}^i \times R_{i+1}^i f_{i+1}$$
$$\tau_i = n_i^T Z_i$$

The terms are assumed to be in the coordinates of the subscript unless superscripted differently. For example p_{i+1}^i is the translation from joint i to joint i+1 in coordinates of joint i. The subscript c indicates that the parameter is taken about the center of mass. Thus, p_{ci+1} is the center of mass of link i+1 in terms of the coordinates of joint i+1. The Z parameters are simply the vectors $[0\,0\,1]^T$ and the $R's$ are the rotations relating the joint coordinates specified. The ω and $\dot{\omega}$ terms represent the angular velocity and angular acceleration at joint i. The $\dot{\theta}$ and $\ddot{\theta}$ terms are scalar joint velocity and acceleration values, the angular velocity and angular acceleration is a vector. The f and n terms represent the final force and torque at each joint. For the PUMA, the τ term picks out the torques from n required to cause motion at each joint. To include the effects of gravity, the outward iteration is initialized with the $\ddot{\theta}_o = G$, with G being the gravity vector. By initializing the iteration this way the effects of gravity are included in the output torque. Note that in the inward iteration the f and n terms at link 6 depend on results for an

imaginary link 7. One way of representing an external load is to specify a force and torque at the imaginary link 7 and follow the iteration. If no load is specified n_7 and f_7 are set to zero.

The drawback of the Newton-Euler method is that it does not transform into the simple matrix form shown in equation (4.1). As a result, it is very difficult to see how the links relate during the recursion. Thus, the Newton-Euler method is fast but difficult to use in control applications. The forward dynamic problem is solved using the algorithm shown above. The inverse dynamic problem in terms of the Newton-Euler method is not solved as obviously. Walker and Orin give four methods for solving the inverse dynamics problem (Walker and Orin 1982). The simplest to understand, though computationally the weakest method, is their first method.

The first method of Walker and Orin requires a total of seven iterations of the Newton-Euler equations. The first iteration is used to find the Coriolis, centrifugal and gravity terms. The next six are used to form the manipulator mass matrix. Recall that in the inverse dynamics problem the torque and previous velocity and position are known and the new acceleration is desired. From equation (4.1) it follows that when $\ddot{\theta}$ is zero, only the Coriolis, centrifugal and gravity terms appear in the torque equation. Thus, by solving the Newton-Euler equations with all $\ddot{\theta}$ terms set to zero will give an output torque equal to the Coriolis, centrifugal and gravity terms. Call this torque b. The mass matrix depends only on the position θ and relates the joint accelerations $\ddot{\theta}$ to the output torque. The columns of the mass matrix are found as follows. First, set the velocity terms to zero, $\dot{\theta} = 0$, remove the gravitational effects, and exclude any external loads. Now solve the Newton-Euler equations with $\ddot{\theta}_i = 1$ for the joint equaling the column being solved for and let all other $\ddot{\theta}$ values equal zero. In other words, to find column 3 of the manipulator mass matrix, set $\ddot{\theta}_3 = 1$, set all other $\ddot{\theta}$'s equal to zero, solve the Newton-Euler equations under the above velocity restrictions, and the resulting torque is column 3 of the mass matrix. After solving for all the columns of the mass matrix the new acceleration is found by:

$$\ddot{\theta} = D(\theta)^{-1}(\tau - b). \tag{4.14}$$

As with the Lagrange-Euler method the actuator inertias are added into the Newton-Euler model externally. The actuator torque can be added in after the forward solution by multiplying the I_a and the $\ddot{\theta}$. In the inverse solution, the I_a terms should be added along the diagonal of the manipulator mass matrix before equation (4.14) is solved.

Explicit Models

Silver used tensor analysis to prove that the Lagrange-Euler and the Newton-Euler methods are equivalent (Silver 1982). This result is intuitively satisfying as a robot arm should not have a multiplicity of dynamic solutions. The difficulty is that the Lagrange-Euler method is computationally too difficult and the Newton-Euler method is poorly structured for control design. With the advent of symbolic manipulation software, a useful compromise has resulted. Either the Lagrange-Euler or Newton-Euler method can be used to find the dynamic equations for the desired manipulator. These equations are put into a symbolic generator that separates the terms into a form similar to equation (4.1). Next, geometric rules relating cosines and sines are used to reduce the equations into more compact forms. Full and reduced order PUMA 560 and 600 models can be found in Armstrong et. al. 1986, Neuman and Murray 1987, Paul et. al. 1983, Leahy et. al. 1986a, and Bejczy et. al. 1985a.

The Armstrong et. al. (1986) formulation was used here as it was a less complex full model and contained the necessary masses, inertias, and link sizes. The model was given as follows:

$$D(\theta)\ddot{\theta} + B(\theta)\dot{\theta}\dot{\theta} + C(\theta)\dot{\theta}^2 + G(\theta) = \tau. \tag{4.25}$$

Here the Coriolis and centrifugal terms have been broken up into two separate terms. The Coriolis or B terms were represented as six upper triangular 6x6 matrices. Each of these matrices represented one of the joints of the PUMA arm. The Coriolis terms were calculated as:

$$\tau_{B_i} = \dot{\theta}^T B_i(\theta)\,\dot{\theta}.$$

The centrifugal or C terms are condensed into only a single matrix and are multiplied by the vector of squared joint velocities. The advantage of using this model is that the results are given in a desirable form and the model requires only three-quarters as many computations as the Newton-Euler method. Simulation showed that the explicit model exactly matched the results of the Newton-Euler method.

Friction Effects

The results given by the Lagrange-Euler and Newton-Euler methods are both ideal with respect to the actual robot. These results reflect only the ideal rigid body dynamics of the PUMA arm. There exist non-rigid body effects like friction, gearing non-linearities, motor backlash, and link elasticity that result in errors between the model and actual arm. Friction can have a dramatic affect on the working of large industrial robot arms. In an attempt to make the simulation more realistic, a friction model was added into the PUMA's inverse dynamics programs.

There are several different types of friction and a variety of ways of modeling there effects. Coulomb or dry friction is the friction between two solid surfaces. The level of joint lubrication plays a big role in how large the Coulomb friction will be. Viscous friction models the friction due to the motion of the joint. In all cases the effects of friction reduce the energy within the system and thus, reduce the influence of the applied torque. The model herein combines the Coulomb and the viscous friction and depends on the sign of the joint velocity (Canudas, et. al. 1986). The model is given as:

$$\tau_f = \begin{cases} \alpha_1\dot{\theta} + \beta_1 & \dot{\theta} > 0 \\ \alpha_2\dot{\theta} + \beta_2 & \dot{\theta} < 0 \end{cases} \tag{4.26}$$

Where the α terms represent the viscous friction and the β terms represent the Coulomb friction. The values of α and β were chosen to be one-tenth the size of the actuator inertia terms. Thus, to represent the friction in the joints, equation (4.26) was solved once at each joint. Since friction only occurs at the joints, decoupling the friction effects gives valid model results.

CHAPTER FIVE

Trajectory Control

Outline of Chapter

The forum of robot control can be divided into three parts. The first part is called trajectory control and addresses the problem of following a desired path in the workspace. The second part, force control, deals with the robot making contact with objects in the workspace. For example, for a robot to wash a window, its motions against the window must be constrained so as not to break the glass. The third part of robot control is a hybrid that combines trajectory and force control. Robots need to move through the workspace and make contact with tools and equipment to be useful. In hybrid robot control the free motion is governed by a trajectory controller and the constrained motion of contact is governed by a force controller. This Chapter addresses the problem of trajectory control. Trajectory control is examined because it is the simplest to simulate and lends itself easily to improved force and hybrid schemes.

Kinematics describes the transformations used to relate the various links of the robot arm. Path planning allows the user to specify a trajectory for the robot arm to trace. Dynamics relates the motion of the arm with the forces and torques required to achieve that motion. The final ingredient for robot motion is the creation of a trajectory control algorithm that keeps the robot moving along the desired path. Trajectory control addresses arm motion through space under various load conditions. It is also the basis for force control, which examines robot motion under contact constraints.

The object of trajectory control is to build a controller for the equation

$$\tau = D(\theta)\ddot{\theta} + C(\dot{\theta}, \theta) + G(\theta) + F(\theta) \tag{5.1}$$

under various load conditions. The path planning algorithms given in Chapter Three define the trajectories for the robot to follow. Three criterion for evaluating

the success of a trajectory control algorithm are stability, robustness, and accuracy. If a controller is stable then the arm will follow the desired path in a manner that drives the errors to zero or to a specified minimum. A controller that drives an arm along a certain path more accurately than another controller is not necessarily more stable. Input-Output stability implies that a bounded input will result in a bounded output. Robustness addresses the soundness of a control scheme to parameter mismatch and small changes in the system. For example, how well does a model based control scheme perform when all of the friction parameters are in error by fifty percent? Does a scheme remain stable under all load conditions and at all acceptable joint velocities? Since robots move at various speeds, with various loads, robustness is a very advantageous quality for a controller to possess. Finally, the most desirable aspect of a trajectory controller is accurate path following. If two schemes are stable and robust, then the more accurate is probably the best.

Equation (5.1) is nonlinear, multivariable, and coupled which makes it a challenge to control. Most trajectory controllers can be divided into the categories of non-adaptive and adaptive controllers. In a non-adaptive scheme the parameters in the controller usually remain constant at all times. For example, feedback gains are calculated, input to the controller, and never changed over the life of the robot. In adaptive control some sort of parameter estimation occurs that is used to update the controller. In an adaptive controller the feedback gains might change automatically on-line to conform more favorably with an updated parameter in the controller.

This Chapter contains four sections that examine the various aspects of non-adaptive and adaptive trajectory control. The first section looks at several non-adaptive controllers developed for robot trajectory control. The second section outlines the non-adaptive computed-torque method. The third section looks at three adaptive design techniques and several trajectory controllers developed from these techniques. Finally, the Chapter concludes with a section on the adaptive computed-torque method developed by Craig (Craig et. al. 1987, Craig 1988).

Non-Adaptive Trajectory Control Algorithms

All industrial robots have a position sensor and many times have velocity sensing in the form of a tachometer. The information gathered by the sensors and the planned path are combined to form joint position and velocity error signals. In general the error is defined as the difference between the planned value and the actual value. The earliest and simplest robot controllers ignore the complex dynamic structure of equation (5.1) and work directly with the joint errors. Proportional-derivative (PD) and proportional-integral-derivative (PID) controllers are used extensively where precise tracking and high speeds are not important. In these controllers the robot dynamics are assumed to be decoupled at each joint. The joint actuators are modeled and individual controllers are designed to minimize the position error at each joint. By minimizing the position error at each joint, the controller attempts to minimize the overall error in the arm. For the PD controller the applied joint torques are given by

$$\tau_i = k_{vi}\dot{e} + k_{pi}e, \tag{5.2}$$

where k_{vi} is the velocity feedback gain, $\dot{e}$ is the velocity error, k_{pi} the position feedback gain, and e the position error. Paul develops equations useful in choosing the k_v and k_p values with respect to the actuator model and joint inertias (Paul 1981a). The PID scheme includes an additional feedback term based on the integral of the error

$$\tau_i = k_{vi}\dot{e} + k_{pi}e + \int k_{ii}e\,dt. \tag{5.3}$$

The advantage of the PD controller is that position and velocity information is available making the scheme simple to implement. The PID controller requires an integrator that can result in problems with windup. However, the addition of the integral term provides better static tracking than with the PD controller.

Assuming the arm is decoupled and applying a PD or PID controller to each actuator gives acceptable results when the desired trajectory can be traced at a slow speed. More complex schemes are needed for tasks that require high speed or

high tracking accuracy. Much of today's work centers on finding control algorithms that are stable and accurate at high speeds.

In addition to PD and PID controllers, a variety of other classical and modern non-adaptive control schemes have been implemented on mechanical manipulators. Kahn and Roth applied optimal control theory to obtain time optimal control for a robot arm. The cost function chosen was simply the time necessary for a robot to complete a motion. Thus, their algorithm sought the control that minimized the time needed to complete a task (Kahn and Roth 1971). Whitney looked at controlling the robot by converting the desired Cartesian motion into joint motion. This method is called resolved rate control as the useful command directions are resolved into the necessary joint motion (Whitney 1969). The scheme uses the Jacobian when converting from the Cartesian to the joint space. Luh et. al. (1980b) take resolved rate control one step further to obtain resolved acceleration control. In resolved acceleration control the accelerations are also specified to allow continuity of the velocity. The joint accelerations are found from the Cartesian accelerations, the Jacobian, and the desired joint velocities. The resulting joint acceleration is used to drive an acceleration based control scheme like the computed-torque method (Luh et. al. 1980b). Chen suggests replacing PID controllers with lag-lead controllers designed in the frequency domain. As most robots have computer controllers, the control schemes must be converted from the continuous to the discrete domain. The lag-lead compensator offers more flexibility in this conversion than either PD or PID controllers (Chen 1989). Finally, Bejczy et. al. (1985a,b), Hemami and Camana (1986), and Spong and Vidyasagar (1985a,b), develop a variety of control algorithms based on nonlinear control theory.

Another approach to manipulator control is based upon accurate knowledge of the robot's dynamic model. Called model based control, a dynamic model of the robot arm is used to generate the control torque. A feedback signal is applied to smooth out errors due to differences between the controller model and PUMA arm. Two popular model based control schemes are the computed-torque method, discussed in the next section, and the feedforward method (An et. al. 1986, 1989, Khosla and Kanade 1986).

In the feedforward method the feedforward signal is given by the inverse dynamics and the feedback signal is simply a PD controller. Thus, the applied torque is generated from the equation

$$\tau = \hat{D}(\theta_d)\ddot{\theta}_d + \hat{C}(\dot{\theta}_d, \theta_d) + \hat{G}(\theta_d) + K_v\dot{E} + K_pE. \tag{5.4}$$

Here $\hat{D}$, $\hat{C}$, and $\hat{G}$ are estimates of the arm dynamics. The resulting error equation is given by combining (5.1) and (5.4) and ignoring the effects of friction. Assuming the controller model and the PUMA robot match exactly leads to the following error dynamics

$$D\ddot{E} + K_v\dot{E} + K_pE = \bar{0}.$$

The Computed-Torque Technique

The computed-torque technique is the most basic of the dynamically dependent control schemes (Leahy et. al. 1989). The computed-torque technique makes use of model based nonlinear feedback to decouple and linearize the dynamics of equation (5.1). The advantage of the computed-torque technique over the feedforward technique is that computed-torque provides more thorough compensation (Khosla and Kanade 1989).

The computed-torque technique requires joint position and velocity information to derive Coriolis, centrifugal, and gravity terms. These terms are fed back to the robot in an attempt to cancel their effects on the system. In addition, the desired joint acceleration plus an error equation are multiplied with the mass matrix to smooth out tracking errors. The input torque for the computed-torque technique is given as

$$\tau = \hat{D}(\theta)\ddot{\theta}^* + \hat{C}(\dot{\theta}, \theta) + \hat{G}(\theta) \tag{5.6}$$

$$\ddot{\theta}^* = \ddot{\theta}_d + K_v\dot{E} + K_pE. \tag{5.7}$$

Assuming that the modeled dynamics match the actual dynamics, ignoring the effects of friction, and combining equations (5.1) and (5.6) gives

$$D(\theta)[\ddot{E} + K_v\dot{E} + K_pE] = \bar{0}. \tag{5.8}$$

As $D(\theta)$ is positive definite, its inverse exists and the error equation becomes

$$\ddot{E} + K_v\dot{E} + K_pE = \bar{0}. \tag{5.9}$$

The augmented joint acceleration given in equation (5.7) can be interpreted as follows. First, the $\ddot{\theta}_d$ term will generate the desired torque for each joint if there are no modeling errors and the system parameters are known. However, deviations from the desired joint trajectory will arise due to backlash, gear friction, uncertainties in the inertial parameters, and time delay in the servo loop (Lee et. al. 1982). Second, the $K_v\dot{E} + K_pE$ terms represent a PD controller that will compensate for inertial loading, coupling effects, and gravity loading in the links (Lee et. al. 1982).

The K_p and K_v values are usually chosen to produce critical damping at each joint. Critical damping is achieved from the relation

$$k_{vi} = 2\sqrt{k_{pi}}.$$

Critical damping is a desirable feature as it produces the fastest tracking without overshoot. Faster tracking is possible with an underdamped system, but complex poles also produce overshoot and possibly oscillation. Overshoot and oscillation are undesirable qualities because they cause wear and tear on the actuators and gears.

Either the dynamics in (5.6) or the command joint acceleration in (5.7) can be changed to modify the computed-torque scheme. In the case of the dynamics, either the Lagrange-Euler, Newton-Euler or explicit model can be used to model the PUMA. Other dynamic considerations include the use of reduced mass matrices (Leahy et.al. 1986b), the inclusion of a friction model (Leahy and Saridis 1986), or the exclusion of the Coriolis and centrifugal terms (Leahy et. al. 1989). Dynamic reduction is usually used to speed up the sampling rate at which the control scheme is applied to the robot. For the PUMA mass matrix reduction can take a diagonal uncoupled form or a block diagonal form. In the block diagonal form the three arm joints are decoupled from the three wrist joints (Leahy et. al. 1986b). Since the

PUMA has a diagonally dominant mass matrix, full or partial decoupling of the mass matrix seems plausible.

Some researchers have looked at the effects of varying the command joint acceleration $\ddot{\theta}^*$. These other command accelerations include:

$$\ddot{\theta}^* = k_p(\theta_d - \theta) - k_v\dot{\theta}, \tag{5.10}$$

$$\ddot{\theta}^* = k_p(\theta_d - \theta) + k_v(\dot{\theta}_d - \dot{\theta}), \tag{5.11}$$

$$\ddot{\theta}^* = k_p e + k_v\dot{e} + \int k_i e dt. \tag{5.12}$$

Equations (5.10) and (5.11) represent situations where the desired trajectory is not fully defined for the joint velocity and joint acceleration, respectively (Khosla and Kanade 1989). Equation (5.12) represents a PID controller rather than the PD controller (5.7) (Leahy and Saridis 1986). The PD controller in (5.7) provides a controller with faster response than either (5.10) or (5.11) (Khosla and Kanade 1989). The PD controller tends to perform slightly better at high speeds than the PID controller. In general, the PD and PID controllers both reject disturbances due to unmodeled dynamics. However, the PD controller is simpler than the PID and does not suffer from any integrator windup problems (Leahy and Saridis 1986).

The question of the computed-torque technique's robustness to errors depends upon the author and robot you choose. Egeland shows that the computed-torque method is robust to errors that occur in the Coriolis, centrifugal, and gravitational terms. Egeland goes on to show that very small errors in the computed mass matrix may result in instability (Egeland 1986). Craig shows that the computed-torque method is very sensitive to errors in the mass matrix. However, Craig points out that theory and experimentation do not match on the robustness issue. He states that acceptable results occur when the inertial terms in the mass matrix are 50 to 100 percent in error (Craig 1989). Leahy et. al. (1989) explain that for the PUMA, modeling errors produced by overcompensating for the impact of inertial coupling produces higher tracking errors, not instability. The robust nature of the computed-torque method with respect to the PUMA is a consequence of high gear ratios. The high gear ratios magnify the actuator inertias and viscous friction thus

enhancing the robustness of the PUMA arm (Leahy et. al. 1989). A direct drive arm, like the CMU DD Arm II (Khosla and Kanade 1989), does not duplicate these gearing traits and is less robust to parameter uncertainty. As a result, the robustness of the computed-torque method depends on the robot being modeled. The technique tends to be more robust on highly geared industrial robots than on direct drive arms. Leahy et. al. showed the best tracking results for the PUMA occurred when the mass matrix was fully decoupled (Leahy et. al. 1989). These results would indicate that the computed-torque scheme is very robust when applied to the PUMA arm.

Adaptive Robot Control

Due to the presence of nonlinearities and uncertainty in robot dynamic models, adaptive control is recognized as an effective technique used in robot control (Hsia 1986). In adaptive control an updating occurs that attempts to reduce the effects of nonlinearities and parameter uncertainty influencing the controller. This updating makes adaptive controllers more desirable than many of the previously discussed non-adaptive control schemes.

Three design techniques, model reference adaptive control (MRAC), self-tuning regulators (STR), and linear perturbation adaptive control (LPAC), categorize robot adaptive control. MRAC controllers are designed in continuous time and attempt to equate the robot closed-loop response with a desired model response. STR's are designed in the discrete time domain and use parameter estimation techniques to equate the robot with the model. In LPAC the applied torque combines a model based nominal torque with a differential torque derived from perturbation analysis.

Some early goals of adaptive control were: insensitivity to parameter uncertainties; insensitivity to unknown load variations; decoupled joint response; and low demand for on-line computations (Hsia 198 1986). Many early non-adaptive robot control schemes failed to give desired results because they required too many

computations to effectively update the torque at high speeds. Adaptive control offered a controller design that could produce acceptable results at computationally acceptable update rates. The increasing power and the decreasing cost of today's computers has allowed researchers to design more powerful adaptive schemes. One example is Craig's computed-torque method discussed in the final section of this Chapter (Craig et. al. 1987, Craig 1988).

This section addresses some of the basics of MRAC controllers, STR's, LPAC controllers, and several resulting adaptive robot schemes. The adaptive schemes developed by Slotine based on sliding modes is not discussed here (see Slotine and and Li 1987, Slotine 1985). Slotine's method is very similar to Craig's adaptive scheme with the exception of a sliding mode term.

The object of model reference control is to make the closed-loop transfer function as close as possible to a desired model's transfer function. The adaptation adjusts the feedback gains in an attempt to minimize the error between the controller dynamic model and actual robot output. There are three basic design techniques used to equate the closed-loop system with the model system. These techniques are known as the gradient approach, the Lyapunov approach, and the hyperstability approach.

Debowsky and DesForges'(1979) application of MRAC to robot manipulators represents one of the first attempts to apply adaptive control to a robot. In this application the reference model dynamics at each joint were modeled as second order differential equations

$$a_i\ddot{y}_i + b_i\dot{y}_i + y_i = r_i(t). \tag{5.13}$$

Here $r(t)$ is the reference input, $a_i = \frac{1}{\omega_{n_i}^2}$, and $b_i = \frac{2\xi_i}{\omega_{n_i}}$. The desired closed-loop dynamic equations with PD feedback can be written as

$$\frac{M_i}{k_{m_i}k_p}\ddot{x}_i + \frac{k_v}{k_p}\dot{x}_i + x_i = r_i(t), \tag{5.14}$$

where k_m is a motor constant and M_i is the effective mass at link i. Making a change of variables for (5.14) gives

$$\alpha_i\ddot{x}_i + \beta_i\dot{x}_i + x_i = r_i(t). \tag{5.15}$$

The question now is how to update k_v and k_p to drive (5.15) to (5.13). Debowsky and DesForges chose to use the gradient approach to find the feedback gains. In the gradient approach the parameters are updated with respect to the gradient of the error equation. The α_i and β_i terms are updated according to the following error equation

$$f_i(\epsilon_i) = \frac{1}{2}(q_0\epsilon_i + q_1\dot{\epsilon}_i + \ddot{\epsilon}_i)^2$$

$$\dot{\alpha}_i = -\frac{\partial f_i}{\partial \alpha_i} \approx \frac{\partial f_i}{\partial a_i}$$

$$\dot{\beta}_i = -\frac{\partial f_i}{\partial \beta_i} \approx \frac{\partial f_i}{\partial b_i}$$

where $\epsilon_i = y_i - x_i$ and the q's are chosen to stabilize f_i. Once $\dot{\alpha}_i$ and $\dot{\beta}_i$ are known the k_v and k_p terms are updated by differentiating the definitions of α_i and β_i and letting a_i approximate α_i.

$$\dot{k}_{pi} = -\frac{\dot{\alpha}_i k_{pi}^2 k_m}{M_i} = -\frac{\dot{\alpha}_i k_{pi}}{a_i},$$

$$\dot{k}_{vi} = k_{pi}\dot{\beta}_i - \frac{k_{vi}}{a_i}\dot{\alpha}_i.$$

Here it is assumed that the effective masses M_i vary slowly with respect to the adaptation (Debowsky and DesForges 1979).

The Lyapunov and hyperstability approaches represent more complex MRAC design techniques than the gradient approach. Craig's adaptive computed-torque represents a MRAC controller based upon the Lyapunov approach and is discussed in the next section. The hyperstability method achieves results similar to the the Lyapunov method. The difference is that the Lyapunov approach uses state space representations and the hyperstability approach uses transfer functions. For examples of MRAC robot controller based on the hyperstability approach see Hsia (1986), and Asare and Wilson (1987). For Lyapunov approaches other than Craig's, see Hsia (1986), and Lim and Eslami (1986).

The most obvious difference between MRAC controllers and self-tuning regulators is the time domain used in the controller design. In the STR the plant is

modeled as a discrete time system or converted to a discrete time system. The design of the STR requires a sizing step, an estimation step, and a controller step. In the sizing step the degree of the plant polynomial has to be decided upon for use in the estimation and controller steps. In the estimation step the unknown parameters are estimated using such techniques as recursive least squares, maximum likelihood or least mean squares (Haykin 1986). In the final step the estimated parameters are used to update the feedback gains of a controller design such as a pole placement routine, a linear quadratic regulator, or a model follower. Either the estimates are injected into the controller indirectly or directly. In an indirect STR the plant parameters are estimated and then used to update the regulator. In a direct STR the model is reparameterized so that the regulator feedback parameters are estimated directly from the error equations. The difficulty with STR's is proving the stability of the system regardless of indirect or direct estimation (Åström and Wittenmark 1988).

The algorithms developed by Koivo and Guo, and Walters and Bayoumi both use the indirect STR approach (Koivo and Guo 1983, Walters and Bayoumi 1982). Both algorithms use a second order autoregressive moving average (ARMA) structure to model the individual joint of the robot. Such a second order structure can be modeled as

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_1 u(k-1) + b_2 u(k-2) \qquad (5.16)$$

where y is the output, u is the input and k the kth sample which occurs at t=kT with T being the sampling rate. The most common method used to estimate the a and b terms in (5.16) is the recursive least squares technique. Let Θ be the vector of coefficients to be estimated (not to be confused with the joint angles), and Φ the regression vector made up of input and output functions.

$$\Theta = [a_1, a_2, b_1, b_2]^T$$

$$\Phi = [-y(k-1), -y(k-2), u(k-1), u(k-2)]^T$$

Using recursive least squares to estimate the Θ vector gives (Åström and Wittenmark 1988)

$$\Theta(k) = \Theta(k-1) + P(k)\Phi(k)[y(k) - \Phi^T(k)\Theta(k-1)]$$

$$P(k) = P(k-1) - P(k-1)\Phi(k)[I - \Phi^T(k)P(t-1)\Phi(k)]^{-1}\Phi^T(k)P(k-1).$$

From the Θ vector the coefficients of equation (5.16) are available and can be used to update the controller.

Koivo and Guo apply recursive least squares to a second order (ARMA) structure similar to equation (5.16). They use the estimated coefficients to drive an optimal controller based on linear programming (Koivo and Guo 1983). Walters and Bayoumi use the estimated coefficients of a second order (ARMA) model to drive a pole placement algorithm (Walters and Bayoumi 1982).

In structure the linear perturbation adaptive controller is similar to the model based feedforward design technique. Both have an input torque that combines a model based feedforward torque with a differential torque used to smooth model errors. In the case of the LPAC, the differential torque is based upon perturbation analysis and not PD control. The input of the LPAC takes the form

$$\tau = \hat{D}(\theta_d)\ddot{\theta}_d + \hat{C}(\dot{\theta}_d, \theta_d) + \hat{G}(\theta_d) + \delta\tau. \tag{5.17}$$

Lee and Chung parameterize the dynamic equations into the state space form

$$\dot{X}(t) = A(X(t), t)X(t) + B(X(t), t)\tau(t). \tag{5.18}$$

Here the states $X(t)$ are the joint velocities and accelerations, and the inputs $\tau(t)$ are the joint torques. A Taylor series expansion of equation (5.18) about a nominal trajectory gives the linear perturbation model

$$\delta\dot{X}(t) = A(X(t), t)\delta X(t) + B(X(t), t)\delta\tau(t). \tag{5.19}$$

The objective becomes to find the $\delta\tau(t)$ that drives the $\delta X(t)$ to zero in equation (5.19). The recursive least squares algorithm is used to find the unknown parameters

in $A(t)$ and $B(t)$. Once these parameters are known a linear quadratic controller is used to find the desired $\delta\tau(t)$ values. This $\delta\tau(t)$ is then added to the nominal model torque as described in equation (5.17) (Lee and Chung 1982).

Guo and Angeles use a PD controller to solve for the differential torque $\delta\tau(t)$ in equation (5.18). In addition, they solve for the feedback gains directly using a least squares scheme and reparameterizing (5.18). Compared with the Lee and Chung parameter estimation scheme, the Guo and Angeles method is simpler and computationally less complex (Guo and Angeles 1989).

Adaptive Computed-Torque Control

One disadvantage of the adaptive schemes discussed in the previous section is the inability to prove the global stability of the controller. Debowsky and DesForges' algorithm requires slowly changing effective masses for the adaptation to occur. In addition, the Debowsky and DesForges' as well as the two STR algorithms model the robot as a set of decoupled linear second order equations. The LPAC controllers of Lee and Chung, and Guo and Angeles linearize the perturbation equations about a nominal trajectory. The problem is that all the stability proofs are based upon simplified linear robot models. Robots are nonlinear and effective masses can change very quickly with respect to the adaptation. As a result, the given stability proofs only hold for a specific set of constrained conditions and not for all robot motion. This section explains the adaptive computed-torque method developed by Craig (Craig 1988, Craig et.al. 1987). Unlike other adaptive schemes, Craig's method uses the full nonlinear dynamics and is developed from Lyapunov stability theory.

Recall that the applied torque from the computed-torque technique is

$$\tau = \hat{D}(\theta)[\ddot{\theta}_d + K_v\dot{E} + K_pE] + \hat{C}(\dot{\theta},\theta) + \hat{G}(\theta). \tag{5.20}$$

If there exist differences between the parameters in the modeled dynamics and the actual dynamics then the actual error equation becomes

$$\ddot{E} + K_v\dot{E} + K_pE = \hat{D}^{-1}[(D-\hat{D})\ddot{\theta} + (C-\hat{C}) + (G-\hat{G})], \tag{5.21}$$

where the arguments have been dropped for simplicity. Craig proposed calling the system parameters P and their estimates $\hat{P}$, so that

$$\Phi = P - \hat{P}.$$

The error equation (5.21) can now be written as

$$\ddot{E} + K_v\dot{E} + K_pE = \hat{D}^{-1}W(\theta, \dot{\theta}, \ddot{\theta})\Phi, \tag{5.22}$$

where the regression matrix W is the matrix of functions that are multiplied with the system parameters. An important aspect for the development of a stable algorithm is knowing some bound on the estimated parameters. The estimated parameters must maintain values that keep the manipulator mass matrix full rank.

Craig's adaptation algorithm begins with a filtering of the servo error to achieve a strictly positive real transfer function. The filtered error is given as

$$e_{1i} = \dot{e}_i + \psi_i e_i.$$

The ψ_i values are chosen to make the transfer function

$$G(s) = \frac{s + \psi_i}{s^2 + k_{vi}s + k_{pi}} \tag{5.23}$$

strictly positive real. A transfer function $G(s)$ is strictly positive real if $G(s)$ is real for all real s, and $Re\, G(s - \epsilon) \geq 0$ for $Re\, s > 0$ and some $\epsilon > 0$ (Åström and Wittenmark 1988). In the case of equation (5.23), the transfer function will be strictly positive real if ψ_i satisfies the relation $k_{vi} > \psi_i > 0$. The Kalman-Yakubovich (Lefschetz 1965) lemma states that if a transfer function $G(s) = C(sI - A)^{-1}B$ is strictly positive real then there exists positive definite matrices $\mathcal{P}$ and $\mathcal{Q}$ such that (Åström and Wittenmark 1988)

$$A^T\mathcal{P} + \mathcal{P}A = -\mathcal{Q}$$

and

$$\mathcal{P}B = C^T.$$

Since (5.23) is strictly positive real, there exist positive definite matrices $\mathcal{P}_i$ and $\mathcal{Q}_i$ such that

$$A_i^T \mathcal{P}_i + \mathcal{P}_i A_i = -\mathcal{Q}_i$$

$$\mathcal{P}_i B_i = C_i^T.$$

Here A_i,B_i and C_i are minimal realizations for the ith filtered error equation

$$\dot{x}_i = A_i x_i + B_i(\hat{D}^{-1} W \Phi),$$

$$e_{1i} = C_i x_i,$$

$$x_i = [e_i, \dot{e}_i]^T.$$

Now the overall system error equation becomes

$$\dot{X} = AX + B\hat{D}^{-1} W \Phi$$

$$E_1 = CX \tag{5.24}$$

where A, B, and C are block diagonal matrices of the individual error realizations. The overall system is still strictly positive real such that

$$A^T \mathcal{P} + \mathcal{P} A = -\mathcal{Q}$$

$$\mathcal{P} B = C^T$$

where $\mathcal{P}$ and $\mathcal{Q}$ are block diagonal matrices of the individual error realizations.

Craig uses Lyapunov stability theory to arrive at the desired update scheme for the system parameters. Craig chooses

$$V(X, \Phi) = X^T \mathcal{P} X + \Phi^T \Gamma^{-1} \Phi \tag{5.25}$$

as the Lyapunov candidate. Recall from Lyapunov stability theory that if $V(0,0) = 0$ for all t, V is differentiable in X and Φ, and V is positive definite then a sufficient condition for asymptotic stability is that $\dot{V}(X, \Phi) < 0$ (Lyapunov). The time derivative of (5.25) gives

$$\dot{V}(X, \Phi) = -X^T \mathcal{Q} X + 2\Phi^T (W^T \hat{D}^{-1} E_1 + \Gamma^{-1} \dot{\Phi}). \tag{5.26}$$

By choosing

$$\dot{\Phi} = -\Gamma W^T \hat{D}^{-1} E_1,$$

equation (5.26) becomes

$$\dot{V}(X, \Phi) = -X^T Q X \tag{5.27}$$

which is negative definite and therefore asymptotically stable. Recall that $\Phi = P - \hat{P}$ and P is a constant with respect to time. Thus, $\dot{\Phi} = -\dot{\hat{P}}$ and Craig's update rule becomes

$$\dot{\hat{P}} = \Gamma W^T \hat{D}^{-1} E_1. \tag{5.28}$$

Equation (5.28) is used to update the parameters within their predetermined bounds. Recall that the bounds were chosen to maintain the full rank of the modeled manipulator mass matrix $\hat{D}$. If the parameter exceeds one of the bounds, that parameter is reset to the boundary value. The inclusion of parameter resetting maintains the non-positiveness of $\dot{V}(X, \phi)$ and insures the overall system stability (Craig 1988).

The terms X, Φ, $\hat{D}^{-1}$, and W are all bounded and $\dot{X}$ is bounded as equation (5.24) is strictly positive real. This implies that X is uniformly continuous and from equation (5.27) $\dot{V}(X, \Phi)$ is also uniformly continuous. The convergence lemma states that if g is uniformly continuous and the integral

$$\int_0^\infty g(s)\, ds$$

exists then

$$\lim_{t \to \infty} g(t) = 0.$$

The boundedness of X and Φ implies the boundedness of V such that

$$\lim_{t \to \infty} V(X, \Phi) = V^*. \tag{5.29}$$

Combining (5.27) and (5.29) implies that

$$V^* - V(X_o, \Phi_o) = -\int_0^\infty X^T Q X dt - \sum_{j=1}^{q} \epsilon_j, \tag{5.30}$$

where ϵ is the error due to parameter resetting (Craig 1988).The left side of (5.30) is finite which implies the terms on the right side are also finite. Thus, $X^T QX$ is uniformly continuous, has a finite value when integrated over positive time, and satisfies the conditions of the convergence lemma. Therefore,

$$\lim_{t\to\infty} X^T QX = 0.$$

Since $X^T QX$ tends towards zero, this implies (Craig 1988)

$$\lim_{t\to\infty} E = 0$$

$$\lim_{t\to\infty} \dot{E} = 0.$$

These results imply the adaptation is stable and the trajectory errors E and $\dot{E}$ converge to zero (Craig 1988).

The convergence of the parameters $\hat{P}$ to P depends upon the persistent excitation of the path chosen. The complete system can be written as

$$\begin{bmatrix} \dot{X} \\ \dot{\Phi} \end{bmatrix} = \begin{bmatrix} A & BU^T \\ -\Gamma UC & 0 \end{bmatrix} \begin{bmatrix} X \\ \Phi \end{bmatrix} \tag{5.31}$$

where $U = (\hat{D}^{-1}W)^T$. Equation (5.31) is asymptotically stable if U satisfies the persistent excitation condition (Craig 1988)

$$\alpha^* I_r \leq \int_{t_o}^{t_o+\rho} UU^T dt \leq \beta^* I_r. \tag{5.32}$$

If U satifies this persistent excitation criterion, the parameters will converge to their actual values. Since $\hat{D}$ is invertible for all $\hat{P}$ the right side of equation (5.32) holds as W is bounded. The problem becomes one of showing for $\alpha^* > 0$ or $\alpha > 0$ that

$$\alpha^* I_r \leq \int_{t_o}^{t_o+\rho} UU^T dt,$$

or equally

$$\alpha I_r \leq \int_{t_o}^{t_o+\rho} W^T W dt. \tag{5.33}$$

The object of robot trajectory control is to drive the arm along the desired path. Therefore, it is not surprising that the persistent excitation condition in equation (5.33) is tantamount to the condition

$$\alpha I_r \leq \int_{t_o}^{t_o+\rho} W_d^T W_d dt \tag{5.34}$$

where W_d is the regression matrix related to the desired trajectory (Craig 1988).

The difficulty of proving the persistent excitation of a path for a robot is the result of the nonlinear dynamics involved. Nonlinear systems are usually statistically non-stationary so that the normal methods based on stationary systems fail (Armstrong 1987). Armstrong showed that the persistent excitation of a path depends upon the condition number of the squared regression matrix

$$M = W_d^T W_d.$$

The squared regression matrix M must be full rank and also well conditioned for persistent excitation. In addition, α must not only exceed zero but it should be fairly large (Armstrong 1987). Khosla adds that the estimated parameters can be classified into three categories. These three categories are uniquely identifiable, identifiable in linear combinations, and unidentifiable. These categories are dictated by the kinematic structure of the robot and the input trajectory. If the user has the freedom to choose the desired path, knowing the parameter categorization of the robot can help in finding the path that excites the desired parameters (Khosla 1989).

Craig's scheme is robust to parameter mismatch and bounded external disturbances. External disturbances are assumed to be uncorrelated with the states of the control system. Internal disturbances or disturbances depending linearly upon the states and unmodeled dynamics represent cases where the robustness is unknown (Craig 1988). Assume that the external disturbance is ν and that it is bounded by ν_{max}. The system error equation becomes

$$\dot{X} = AX + B\hat{D}^{-1}W\Phi + \nu$$

$$E_1 = CX. \tag{5.34}$$

The Lyapunov candidate function remains as in equation (5.25), but using the same adaptation gives

$$\dot{V}(X, \Phi) = -X^T \mathcal{Q} X + 2X^T \mathcal{P} \nu. \tag{5.35}$$

Thus, for the system to remain stable X is bounded by

$$X < 2\mathcal{Q}^{-1}\mathcal{P}\nu.$$

This is equivalent to bounding the X within the hypersphere

$$\|X\|_2 < 2\frac{\lambda_{pmax}}{\lambda_{qmin}}\nu_{max}. \tag{5.36}$$

The λ_{pmax} and λ_{qmin} are the maximum and minimum eigenvalues of the $\mathcal{P}$ and $\mathcal{Q}$ matrices, respectively. This hypersphere implies that error constants exist such that

$$\|E\| < e_{max}$$

$$\|\dot{E}\| < \dot{e}_{max}.$$

In addition, the $\ddot{E}$ is also bounded as the disturbance and all other error terms are bounded (Craig 1988).

With respect to parameter convergence with external disturbances, the Φ term will no longer converge to zero. If the system is persistently exciting the Φ term will converge to a non-zero value (Craig 1988). The question of persistent excitation is complicated by the addition of the external disturbance. The disturbance will tend to reduce the excitation of the trajectory but the nonlinear nature makes it difficult to predict by how much (Craig 1988).

CHAPTER SIX

Simulation and Results

Simulation

A simulation was written to compare the non-adaptive and the adaptive computed-torque techniques. The explicit dynamic model developed by Armstrong et. al. (1986) was used as the basis for both the PUMA model and the model used in the controller. The explicit model was used to model the PUMA because it requires less calculations to solve the inverse dynamics problem than either the Newton-Euler or Lagrange-Euler methods. The explicit model was also used in the controller because it is easier to identify the parameters to be estimated in the explicit model than in the Newton-Euler recursion. Simulation showed that for the non-adaptive controller model, the explicit model and Newton-Euler model gave exactly the same results.

Figure 6.1 shows the block diagram outlining the adaptive computed-torque simulation. The non-adaptive controller uses the same structure with the exception of the parameter update mechanism. The adaptation can be removed by setting the Γ matrix equal to zero in the adaptation block. The specifics of each block used in the simulation are given below. Note that all of the theory used in the simulation is given in the first five chapters and that no new theory is required to understand the simulation.

For simplicity the path planning block was written using a joint space scheme. Initial and final positions, velocities and accelerations are chosen and a quintic polynomial is used to fit the points. The quintic polynomial coefficients, given in equation (3.1) and solved in equation (3.2), are solved six times. Six sets of coefficients are required to specify the desired motion of each joint of the PUMA. These coefficients are found before robot motion starts and are stored for use when motion takes place. Storing the coefficients before motion begins reduces the computations required during each iteration of the simulation.

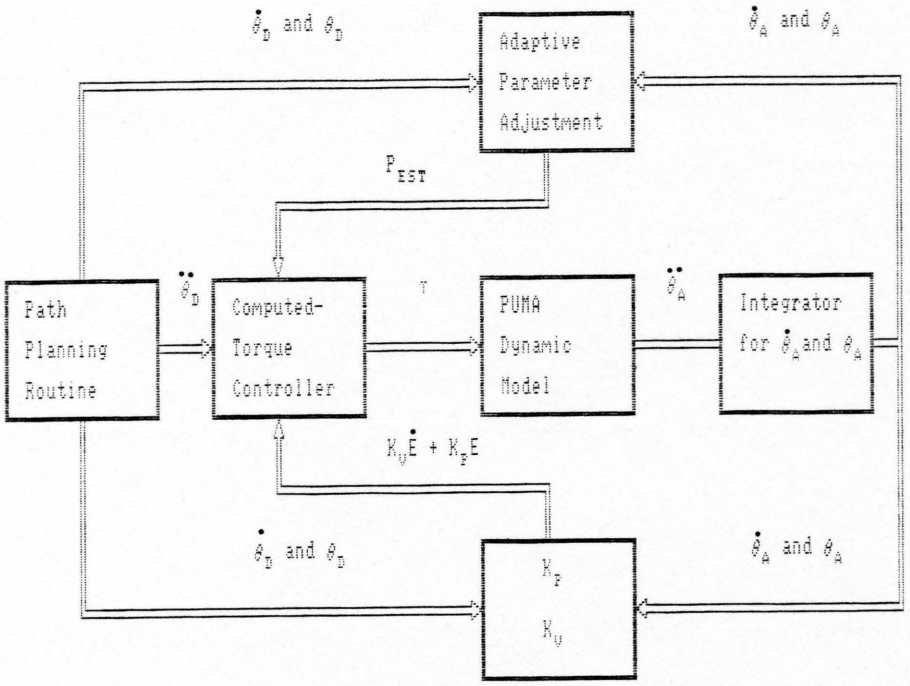

Figure 6.1 **Block diagram of PUMA Simulation**

The most important part of the computed-torque controller is the dynamic model used to find the desired joint torques. The desired positions, velocities, and accelerations are known, and therefore a forward dynamics program is required. The forward dynamics program uses the full explicit model but excludes a friction model. Since the link lengths and masses are easy to find for all robots, these terms were assumed to be exact. However, link and actuator inertial parameters are very difficult to measure. As a result, all inertial parameters that are not estimated are increased by ten percent over the values used in the PUMA model. This ten percent increase in the inertial values simulates the differences between a controller model and the PUMA dynamics.

The actual PUMA robot is modeled using an inverse dynamics program as a torque is applied and the resulting acceleration is desired. The inverse dynamics

program uses the full explicit model where all lengths, masses, and inertias are assumed to be the true values sought by the controller.

One drawback of the inverse dynamic algorithms is that they produce the new joint accelerations and not the new velocities or positions. As a result, a linear integrator is used to update the joint velocity and joint position values from the new joint accelerations. The resulting velocities and positions are given as

$$\dot{\theta}(n) = \dot{\theta}(n-1) + \ddot{\theta}(n)\Delta t$$

$$\theta(n) = \theta(n-1) + \dot{\theta}(n)\Delta t + \frac{1}{2}\ddot{\theta}(n)\Delta t^2$$

where Δt is the time interval between samples, n is the sample value which equals the actual time $n\Delta t$.

A discrete version of Craig's update equation (5.28) is used to update the desired parameters. The discrete $\hat{P}$ values are given by

$$\hat{P}(n) = \hat{P}(n-1) + \Delta t_c \Gamma W^T \hat{D}^{-1} E_1. \tag{6.1}$$

Note that the mass matrix $\hat{D}$ at sample n uses the $\hat{P}(n-1)$ values as $\hat{D}$ and $\hat{P}$ cannot be updated simultaneously. This inability to update at the same time is a result of the $\hat{P}$ dependence on $\hat{D}$. Also notice that the Δt_c in equation (6.1) is different from the Δt value used in the joint velocity and joint position integrators. To reduce the integrator error, the time increment of the overall system was set to 1 ms. However, 1 ms is a little fast for a computed-torque controller using full model dynamics. Therefore, the controller is only updated every 5 ms. Thus, the input torque sent to the PUMA model remains constant for four iterations and is updated on the fifth iteration. This difference in sample times more accurately represents the continuous nature of the robot and the discrete nature of the controller.

Finally, the control loop is closed by feeding the error signals back into the control model. These errors signals are formed by differencing the planned joint velocities and joint positions with the actual PUMA joint velocities and joint positions. The velocity feedback gains, k_{vi}, are all set to 20 and the position feedback gains, k_{pi}, are all set to 100. These position and velocity feedback gains place both

system error poles for each joint at $s = -10$. These feedback gains were chosen to match the pole placement used in other studies involving PUMA robots (Leahy et. al. 1989, 1986b, and Lee and Chung 1982).

Goals and Results

The goal of the simulation was to find under what conditions Craig's adaptive algorithm gave better tracking results than the non-adaptive method. In the case of the non-adaptive scheme, all inertial terms in the controller model were ten percent higher than those of the PUMA model. For the adaptive scheme the controller inertial parameters were initially ten percent larger than those of the PUMA model. The parameters chosen for adaptation were allowed to move within some specified bound. The parameters not chosen for adaptation remained ten percent in error.

Several questions need answering before analyzing the results of the simulations. First, which parameters should be chosen for the adaptation? Second, what values should be used for the ψ_i found in equation (5.23), and the Γ appearing in equation (6.1)? Third, how do path considerations affect the adaptation mechanism? Fourth, does the system degrade under unknown load conditions? Finally, does the adaptation scheme degrade after friction effects are added into the PUMA model but unaccounted for in the controller model?

The choice of parameters for adaptation depends heavily upon the accuracy of the model used in the controller. In this study the controller inertial parameters are assumed to be in error and all other controller parameters fall within acceptable limits. Knowing that the inertial parameters are in error better defines the problem of parameter selection. However, Armstrong et. al. (1986) still give 23 link inertia and 6 actuator inertia terms for the full dynamic PUMA model (Armstrong et. al. 1986). For this study the wrist link errors are ignored but their effects are included in the dynamics of the controller and PUMA models. The reason these links are ignored is that the wrist has no affect upon the Cartesian position of the PUMA arm. Their kinematic structure only effects the orientation of the hand. It was assumed that the controller produced acceptable gross motion control in the wrist

and that fine control could be used to achieve the final wrist orientation. Thus, only the first three actuator inertias and the major inertial terms in the arm joints are considered. Work by Leahy et. al. indicates that the most important model parameters in the PUMA mass matrix are the actuator inertia terms (Leahy et. al. 1989). The actuator inertia terms are a good choice for adaptation because in the PUMA they are large. In addition, the effects of the actuator inertias upon the mass matrix should remain constant over position and load.

After selecting the parameters for adaptation, bounds for these parameters must be set for the updating process. Simulation of the system using an open-loop computed-torque controller helps define mismatch in the controller model. If the actual positions always lag behind the desired positions at a particular joint, then the parameters modeling that joint are probably too small. Likewise, if the actual positions lead the desired positions, then the parameters are too large. System trajectories that lag behind the desired trajectories are more acceptable than those that lead. If a system trajectory always lags the desired trajectory, then the system is trying to catch up with the desired trajectory. The lagging case is better than the leading case, because if the robot leads the desired trajectory, then collisions near the trajectory end point might occur. As a result, it is better to underestimate a parameter than set its value too large. For the purposes of this study the actuator inertias were bounded by ten percent below their true values and at most fifteen percent greater than their true value.

The ψ_i and Γ dictate the rate of adaptation that occurs in equation (6.1). Recalling that ψ_i is picked to maintain a strictly positive real transfer function, ψ_i is bounded between zero and 20. Small ψ_i values indicate the inclusion of less position information and slower adaptation. Larger ψ_i terms indicate an emphasis on the position information and faster adaptation. The elements in Γ dictate the coarseness of the adaptation. Large Γ elements will give faster adaptation and coarse parameter convergence. Smaller Γ elements will give slower adaptation but finer parameter convergence. Also, the size of the parameter error should be considered when choosing a proper magnitude for the elements in Γ. Too large a value can cause the updated parameter to bang between its high and low limits.

Each joint was commanded to move from 0.1 radians to 1.1 radians following a quintic polynomial path for a prespecified amount of time. Recall that the actuator inertial parameters appear on the diagonals of the manipulator mass matrices. Therefore the W matrix is given as

$$W = \begin{pmatrix} \ddot{\theta}_1 & 0 & 0 \\ 0 & \ddot{\theta}_2 & 0 \\ 0 & 0 & \ddot{\theta}_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

where

$$\hat{P} = \begin{pmatrix} I_{a1} \\ I_{a2} \\ I_{a3} \end{pmatrix}$$

From the persistent excitation condition $W^T W$ is

$$W^T W = \begin{pmatrix} \ddot{\theta}_1^2 & 0 & 0 \\ 0 & \ddot{\theta}_2^2 & 0 \\ 0 & 0 & \ddot{\theta}_3^2 \end{pmatrix}$$

which indicates the relative degree of persistent excitation. The greater the accelerations in the first three joints, the greater the degree of persistent excitation. Thus, the time interval to move from 0.1 radians to 1.1 radians dictates the amount of excitation in the path. A shorter time interval results in higher accelerations and consequently greater excitation. Simulations showed that the longer interval times were less exciting than the shorter intervals. Also, the quintic polynomial gives an acceleration that is maximized at the quarter-point of the trajectory, and minimized at the three quarters point. However, the magnitudes of the maximum and minimum acceleration are equal which implies equal excitation about these two points. Figures 6.2, 6.3, and 6.4 show the desired joint path in terms of position, velocity and acceleration.

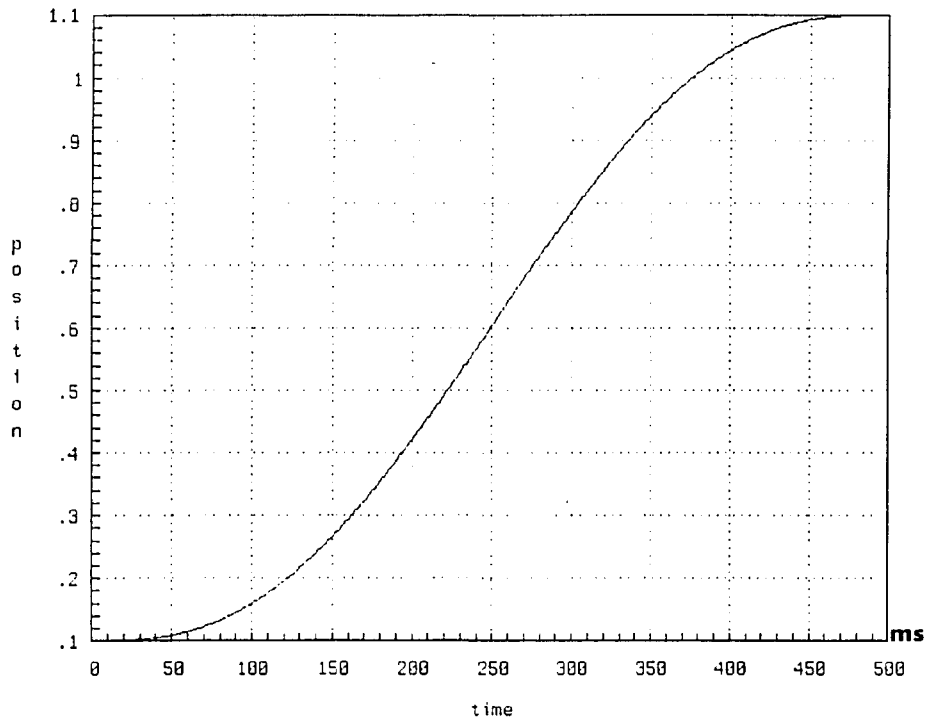

Figure 6.2 Desired joint position trajectory

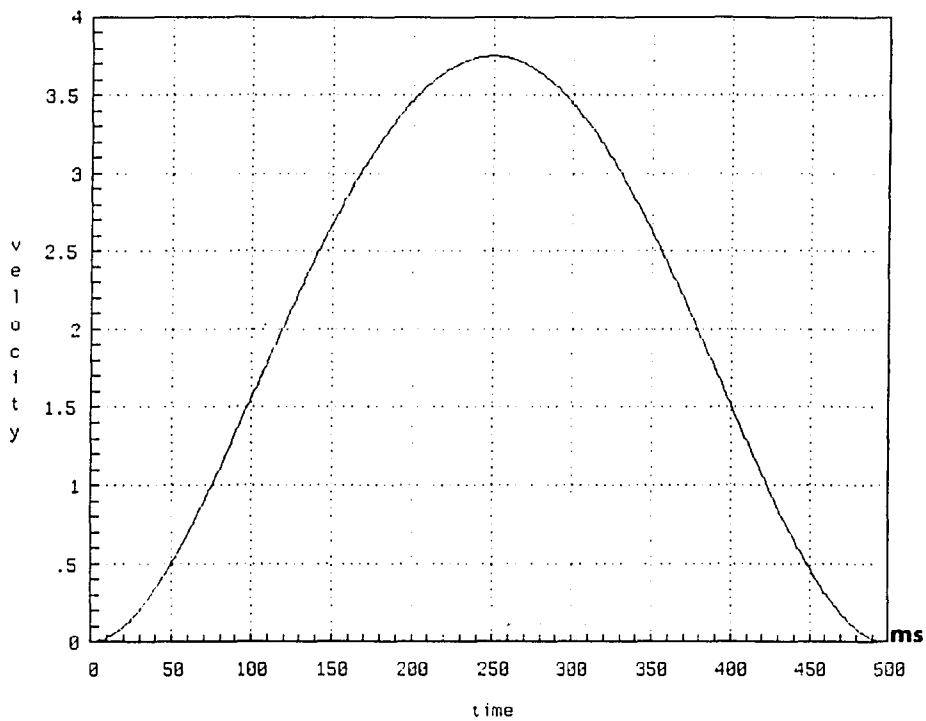

Figure 6.3 Desired joint velocity trajectory

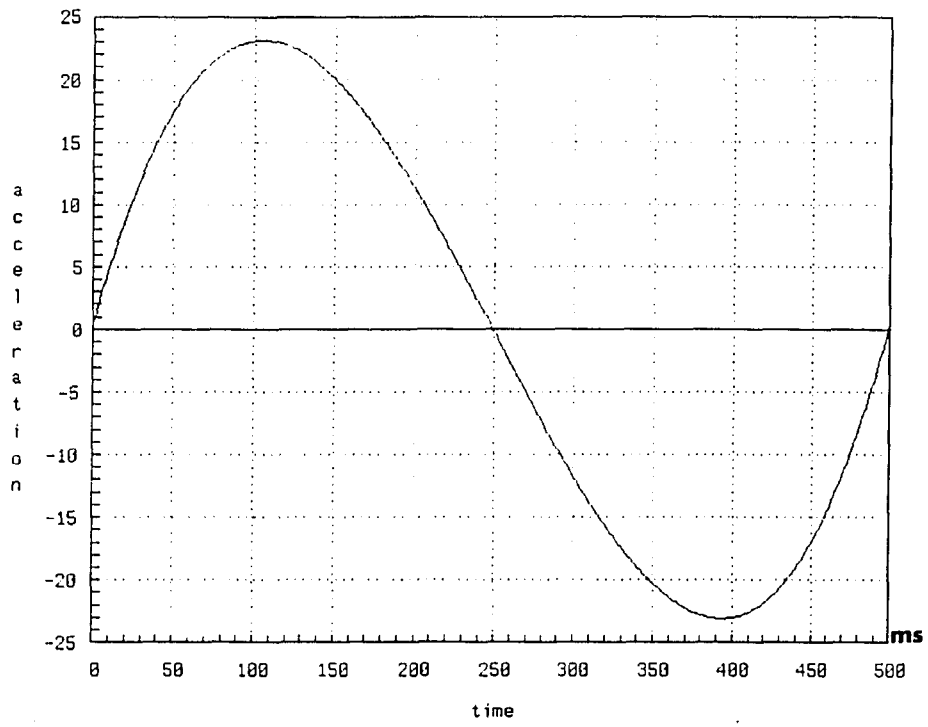

Figure 6.4 Desired joint acceleration trajectory

Using the above information concerning the selection of ϕ_i, Γ, and W, the first simulation represents a test of the parameter updating ability of equation (6.1) for the three actuator inertias. The actual actuator inertias in the PUMA model were 1.14, 4.71, and 0.827 for joints one, two, and three, respectively. The controller actuator inertias were initially set to 1.254, 5.181, and 0.90, respectively. The ψ_i values were all set to 8 which injects into the adaptation eight times more position error information than velocity error information. Γ is a diagonal matrix with elements of 2.0, 2.0, and 0.2. The third Γ element is smaller because the a priori actuator inertia in the third joint is smaller. The desired path time interval was set to 0.5 seconds to give relatively fast motion. The trajectory motion was repeated to feed the controller more data for learning the actual PUMA actuator inertias. Figures 6.5, 6.6 and 6.7 show the parameter adaptation after ten repeated trials.

Repeated trials means that the last estimated parameters for one trial are the initial values for the next trial. Notice that after ten trials the parameters have adapted to values very close to those in the PUMA model. After a certain point the trajectory errors become so small that the adaptation shows very little change. This explains why the parameters do not actually converge after ten trials.

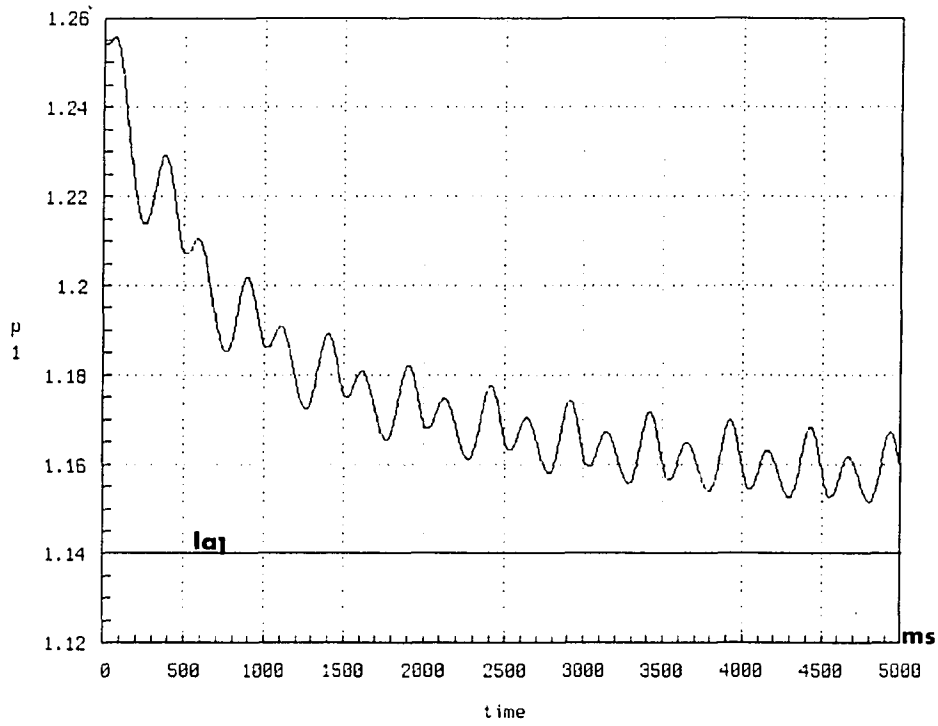

Figure 6.5 Convergence of P_1 after ten trials (5sec)

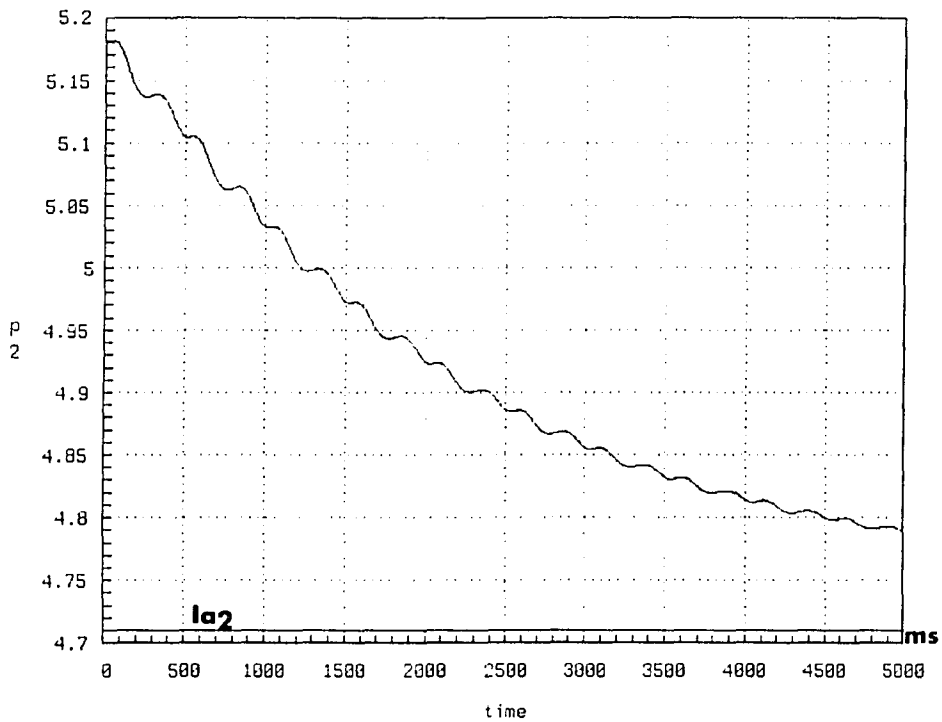

Figure 6.6 Convergence of P_2 after ten trials (5sec)

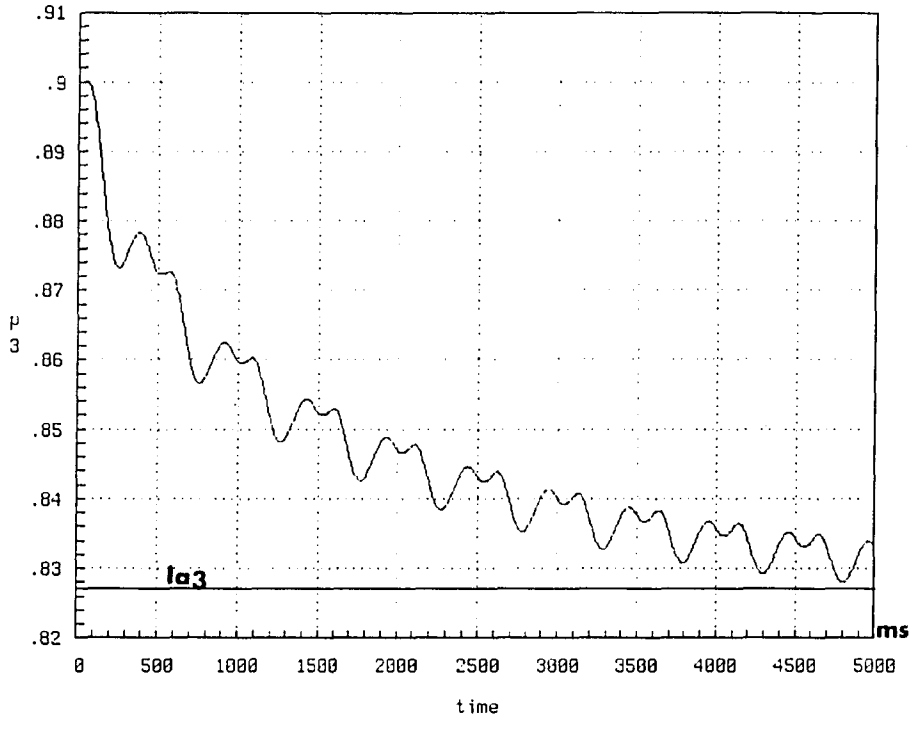

Figure 6.7 Convergence of P_3 after ten trials (5sec)

The next step in analyzing Craig's adaptation scheme is to add the ten percent inertial parameter error for all parameters in the controller model. Here the adaptive scheme is compared with the non-adaptive controller having a constant ten percent inertial error in its model. The non-adaptive computed torque method serves as a benchmark for analyzing the success of the adaptive scheme. The adaptive controller requires greater computational power and should not be used if the non-adaptive case proves acceptable. Figures 6.8, 6.9, and 6.10 show plots of position errors verses time. The position errors represent the results of the first five adaptive trials and the non-adaptive trial. The Γ matrix is a diagonal matrix with elements 2.0, 2.0, and 0.2 the same magnitudes as used in the test of parameter convergence. This simulation shows that the adaptive computed-torque controller

reduces the position error as the number of trials increases. Notice that the position errors are given in radians and the time axis represents the 500ms required to traverse the 0.5 second trajectory at the integrator sampling rate.

Note that for all plots of joint position error the time scale is given in milliseconds. In addition, the non-adaptive trial is denoted ——, and for the adaptive trials -o- denotes trial one, -|- trial two, -x- trial three, -Δ- trial four, and -□- trial five.

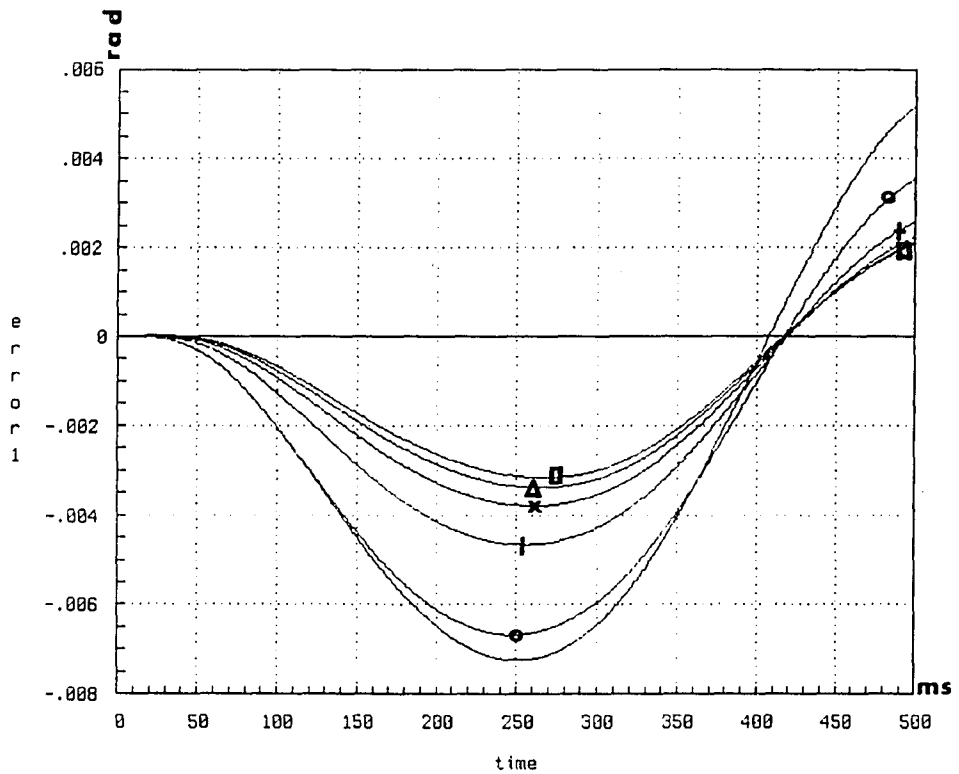

Figure 6.8 Joint 1 position error with no load

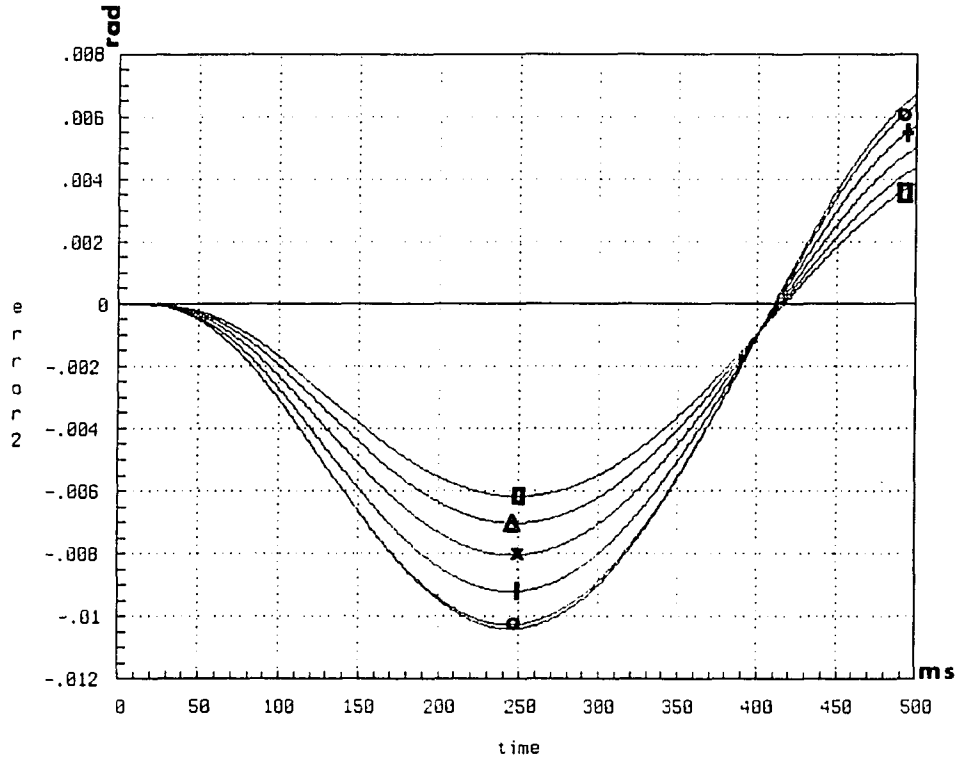

Figure 6.9 Joint 2 position error with no load

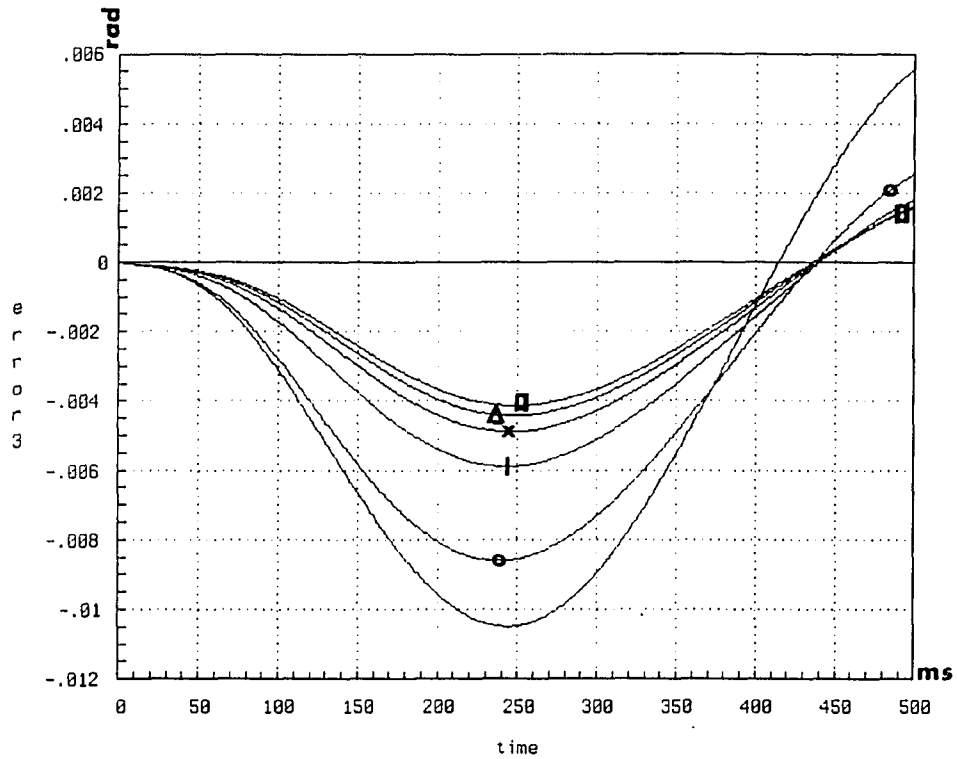

Figure 6.10 Joint 3 position error with no load

The following simulation analyzes the adaptive controllers performance under half and full load conditions. Here the PUMA model is given a 1.15 kg load and a 2.3 kg load, respectively. The controller model remains at no load and the tracking ability of the adaptive and non-adaptive computed-torque controllers are compared. The reason no load conditions are assumed in the controller model is that ideally the adaptive scheme should compensate for the loading difference. Figures 6.11, 6.12, and 6.13 show the half load trajectory errors; Figures 6.14, 6.15, and 6.16 show the full load errors. In the load simulations the Γ matrix had its diagonal elements reduced to 0.5, 0.5, and 0.05, respectively. This reduction is the result of poor parameter convergence. The load cases produce greater position errors which drive the parameters to their upper limits very quickly with the larger Γ values.

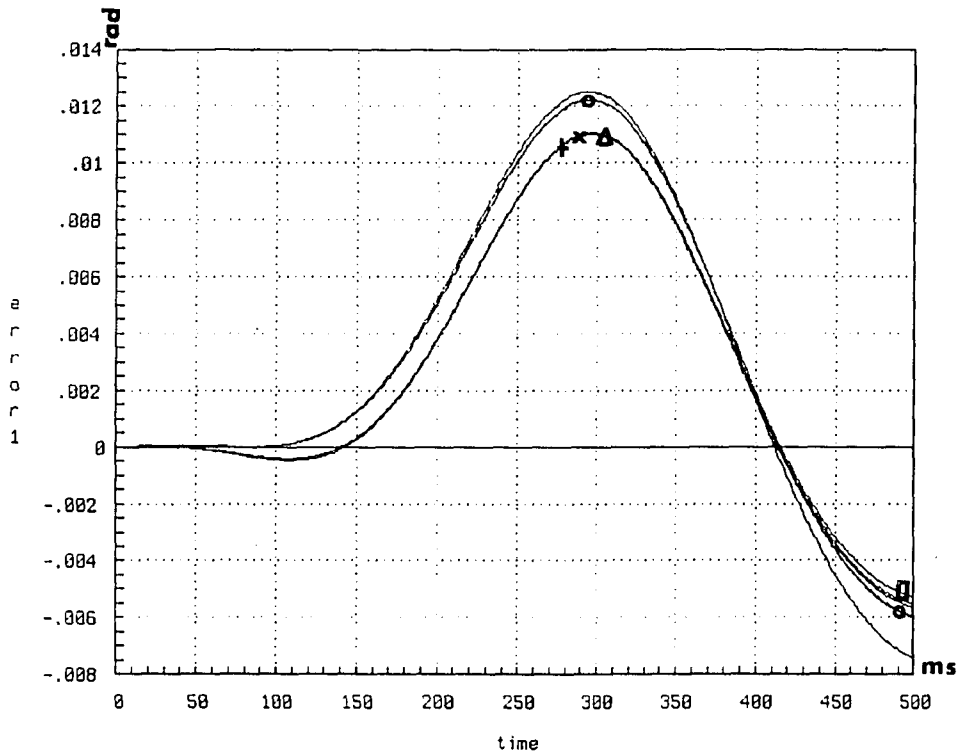

Figure 6.11 Joint 1 position error at half load

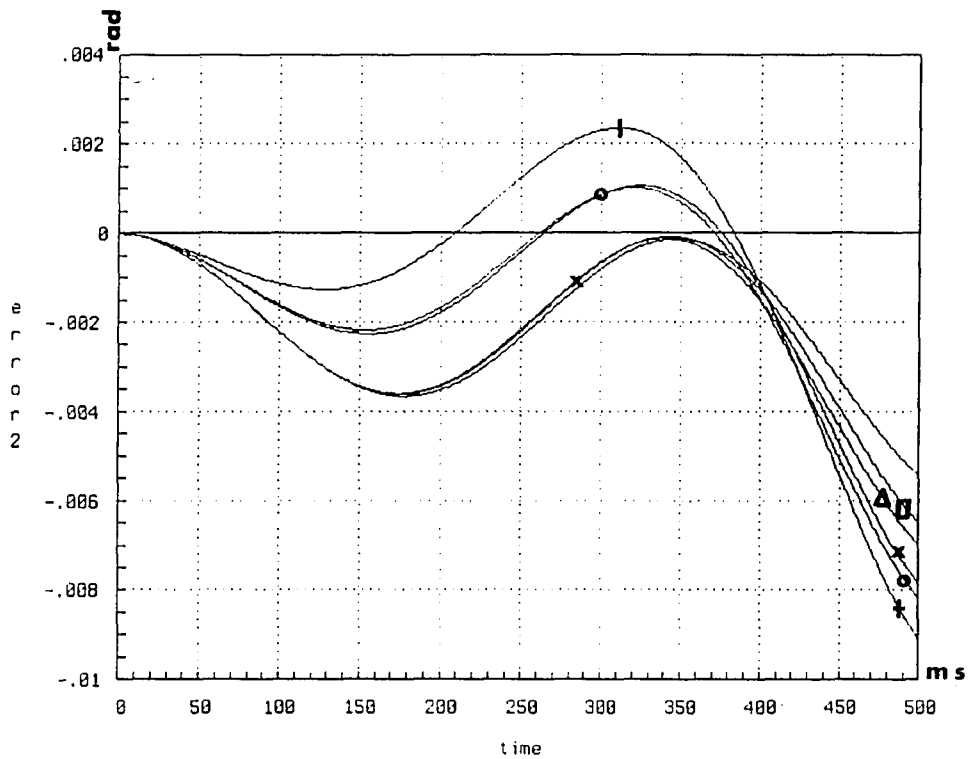

Figure 6.12 **Joint 2 position error at half load**

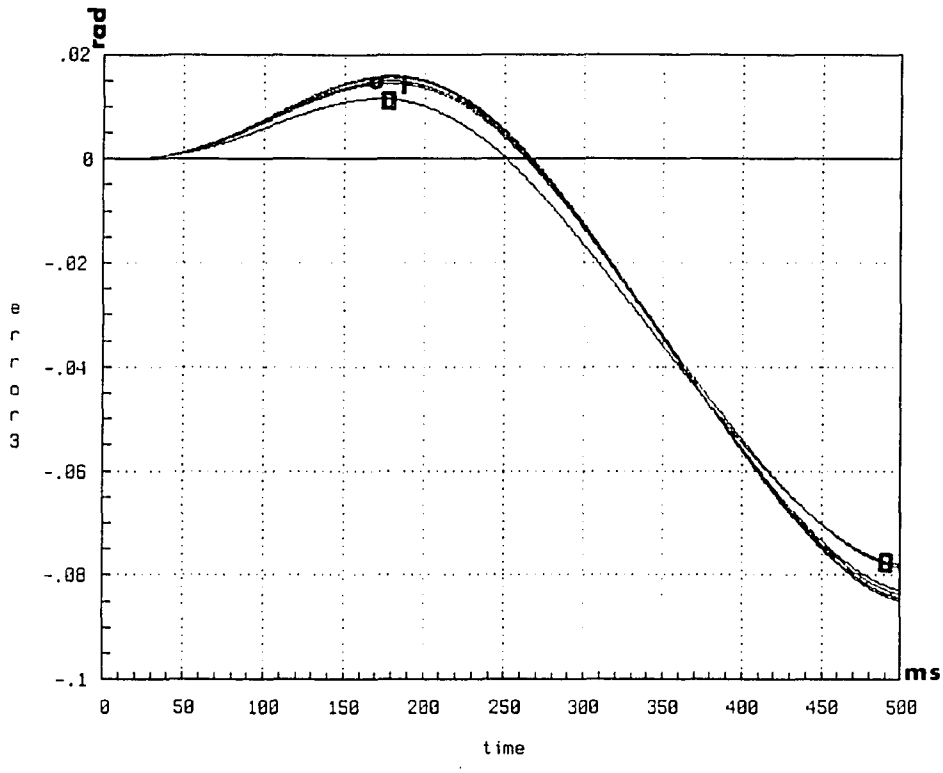

Figure 6.13 Joint 3 position error at half load

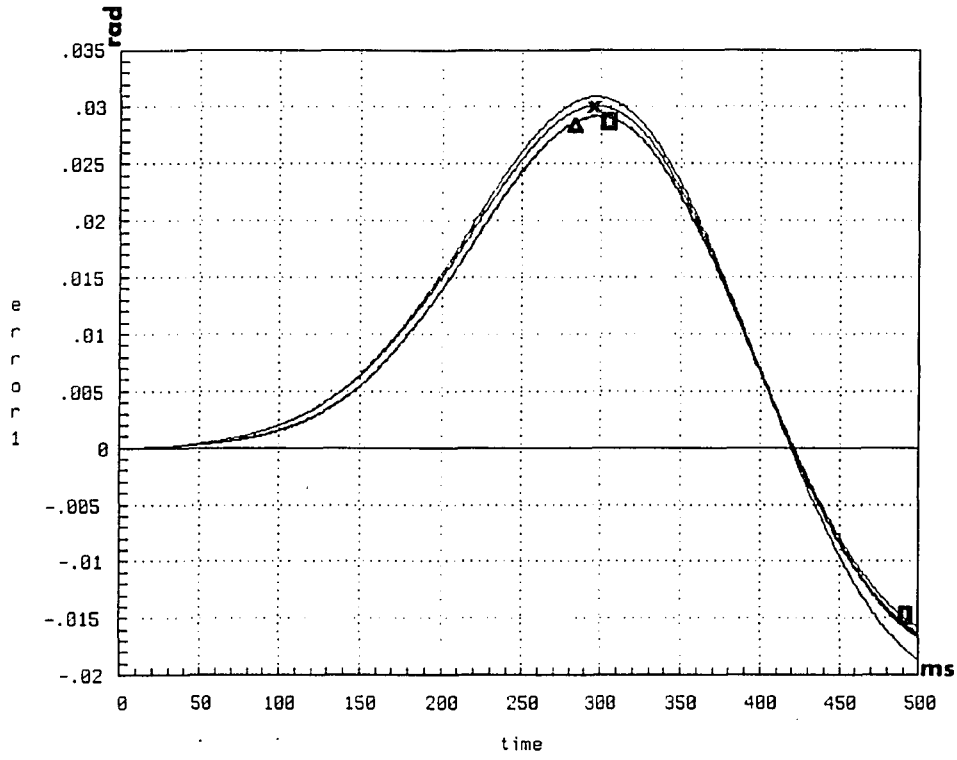

Figure 6.14 Joint 1 position error at full load

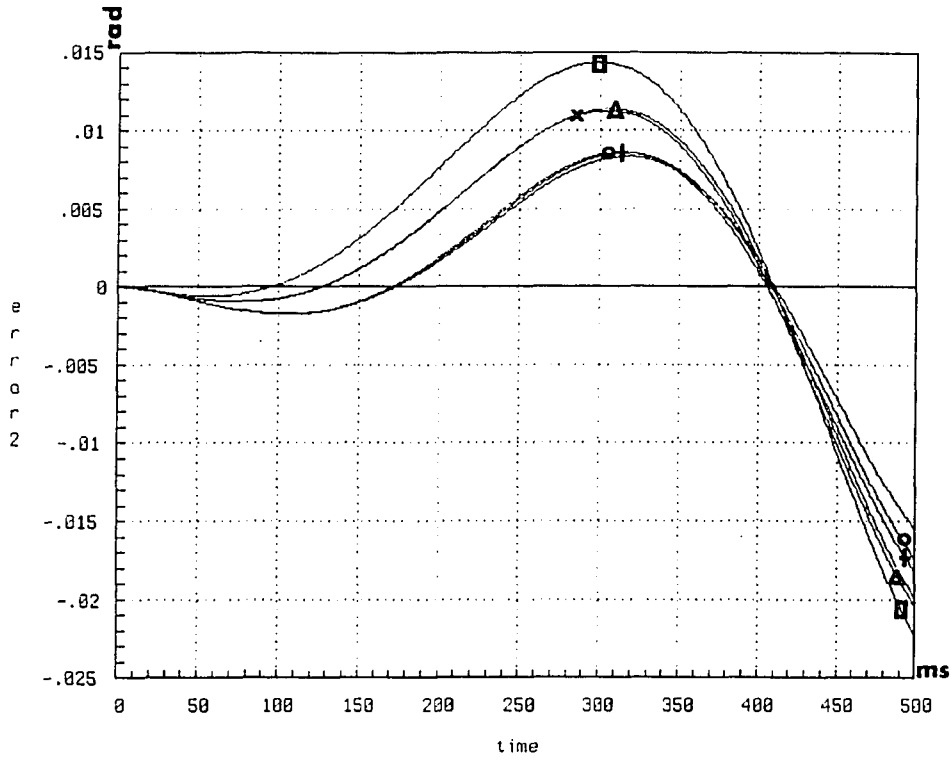

Figure 6.15 **Joint 2 position error at full load**

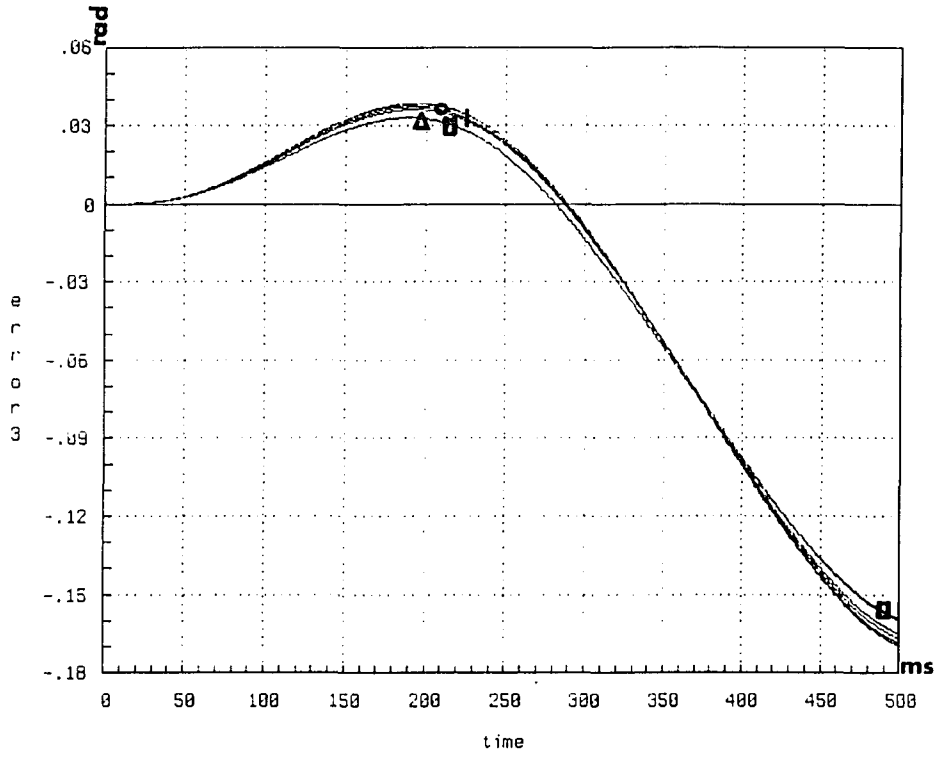

Figure 6.16 Joint 3 position error at full load

The addition of a friction model into the PUMA will further separate the controller and PUMA models. Figures 6.17, 6.18, and 6.19 show the position errors for the first three links under no-load and ten percent inertial error. The Γ values were set to 2.0, 2.0, and 0.2 along the diagonal, and five trials were run. These plots show how the position tracking improves after a few repeated trials. The half and full-load simulations were rerun giving results similar to those in Figures 6.11 through 6.16. Since the load results were so similar to the non-friction cases, they were not replotted here.

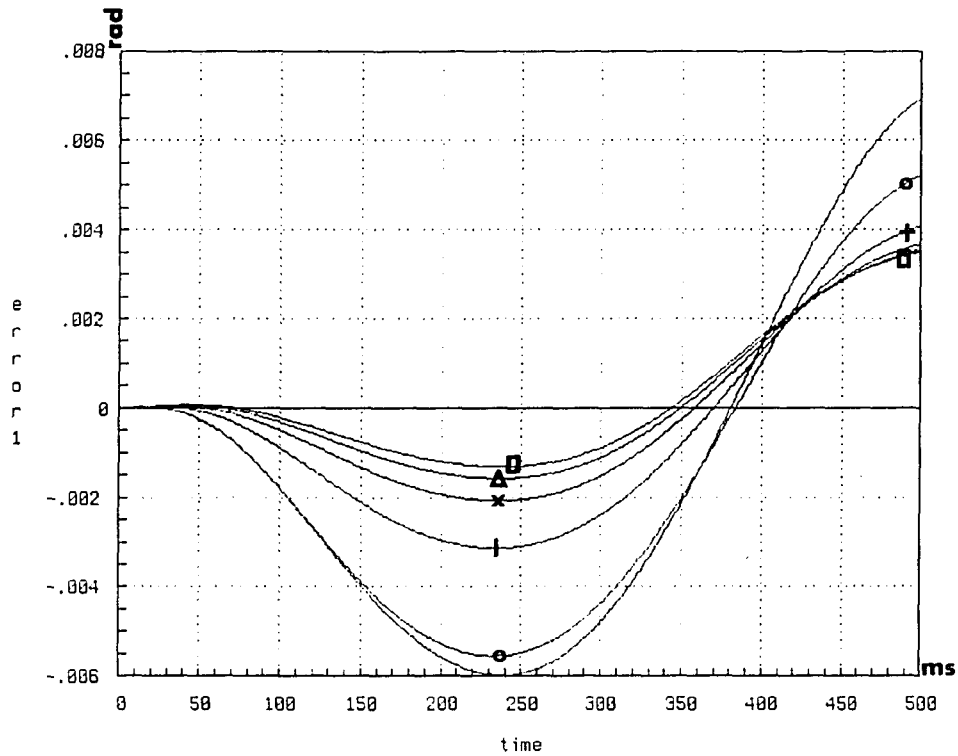

Figure 6.17 Joint 1 position error with friction

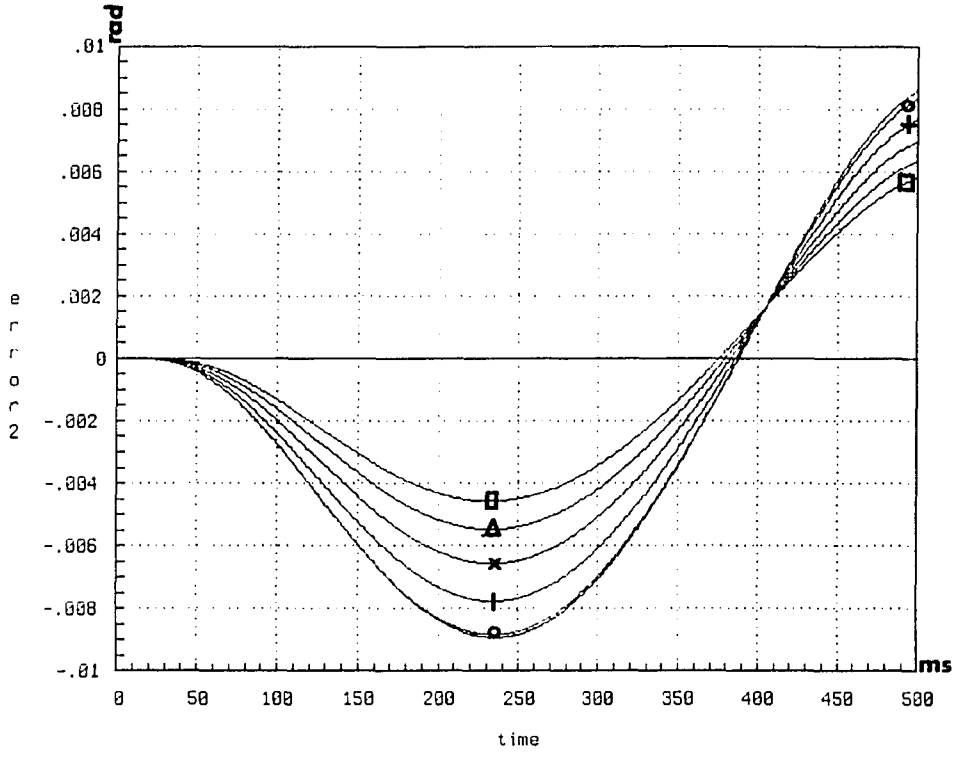

Figure 6.18 Joint 2 position error with friction

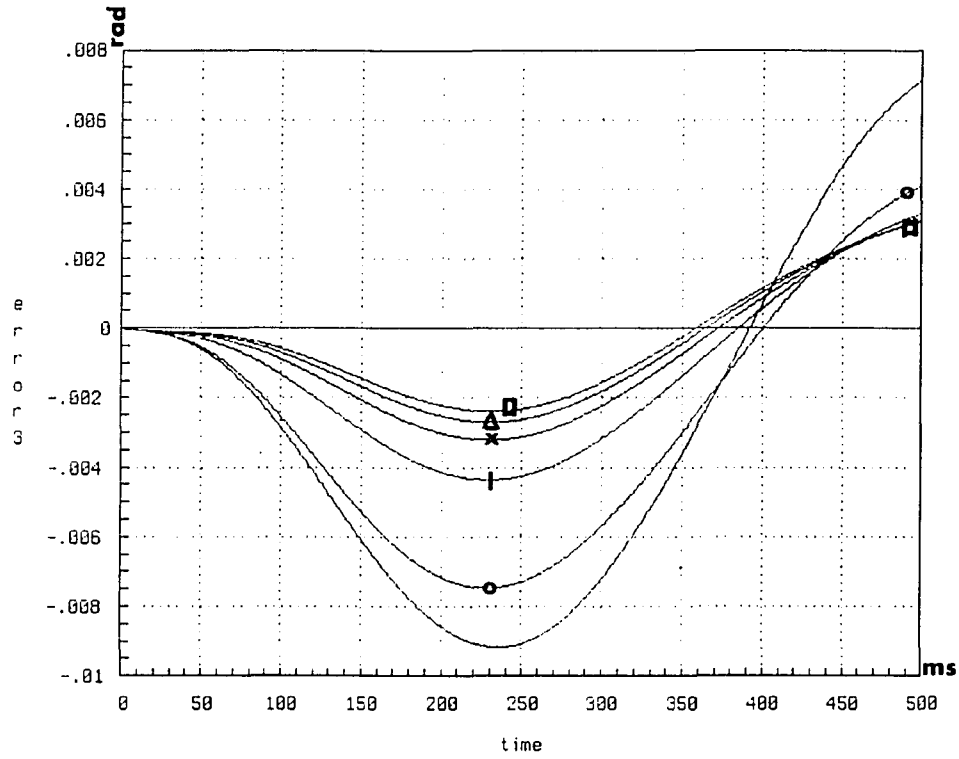

Figure 6.19 Joint 3 position error with friction

Finally, a simulation was run to test the adaptive schemes tracking ability when small differences exist between the actual and predicted load. The half and full load simulations show that the computed-torque scheme requires information of the load regardless of the actuator inertia values. A simulation was run with the controller model having a 1.0 kg load and the PUMA model having a 1.15 kg load. Figure 6.20, 6.21, and 6.22 show the results of the non-adaptive trial and five adaptive trials. Table 6.1 summarizes the results of the above simulations.

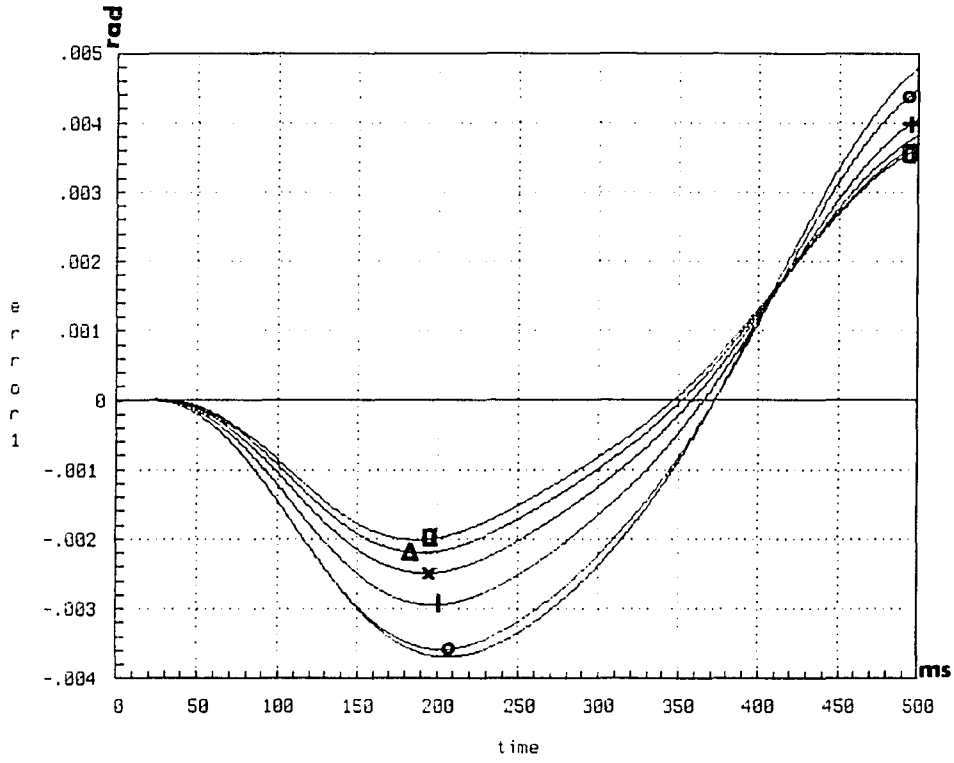

Figure 6.20 Joint 1 position error with inaccurate load

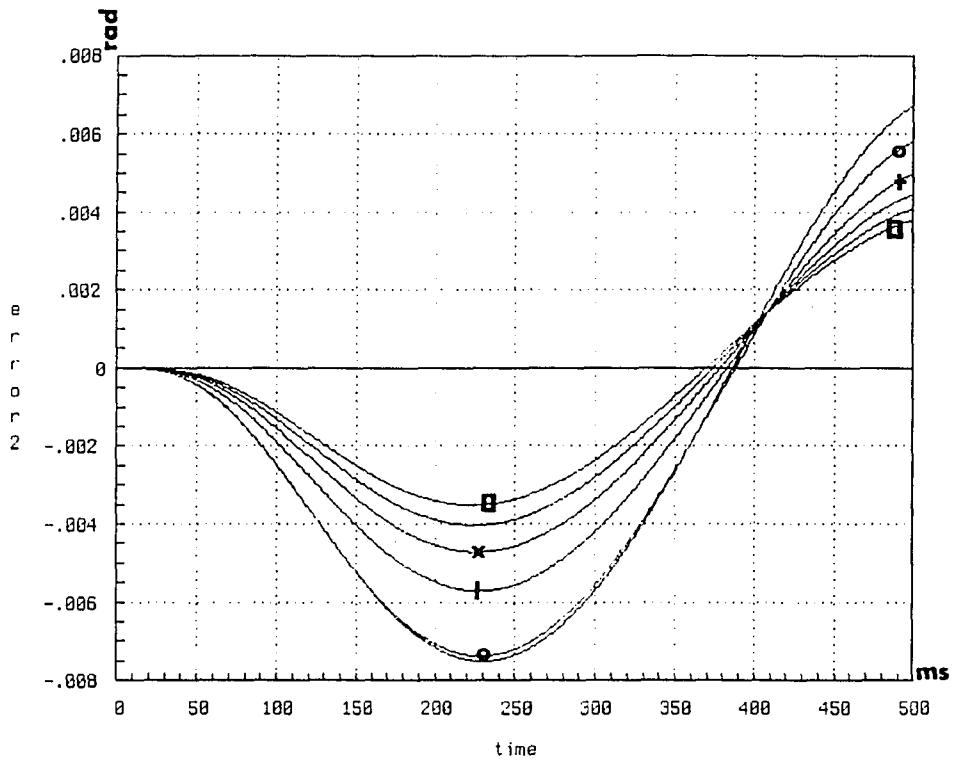

Figure 6.21 Joint 2 position error with inaccurate load

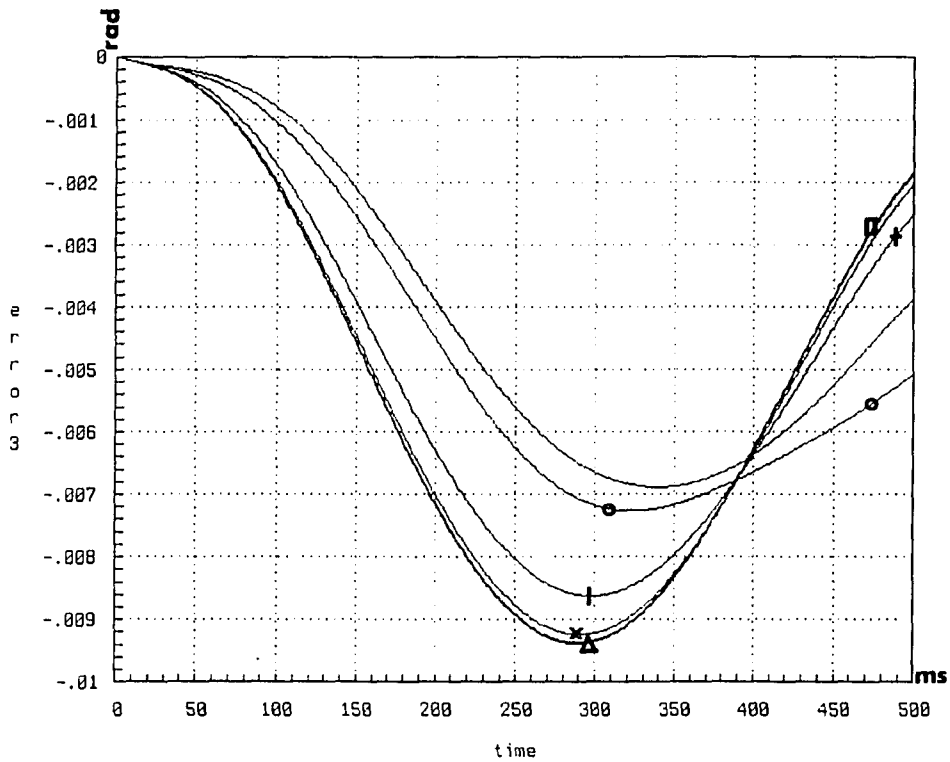

Figure 6.22 Joint 3 position error with innacurate load

Case	joint 1 max final (degree)	joint 2 max final (degree)	joint 3 max final (degree)	
No Load	-0.19 0.120 -0.41 0.298	-0.36 0.223 -0.69 0.384	-0.23 0.091 -0.69 0.321	adaptive non-adaptive
Half Load	0.630 -0.32 0.688 -0.42	-0.47 -0.47 -0.31 -0.31	-4.47 -4.47 -4.86 -4.86	adaptive non-adaptive
Full Load	1.71 -0.95 1.77 -1.07	-1.17 -1.17 -0.88 -0.88	-9.15 -9.15 -9.72 -9.72	adaptive non-adaptive
Friction	0.200 0.200 0.395 0.395	0.332 0.332 -0.52 0.493	0.178 0.178 -0.52 0.407	adaptive non-adaptive
Inaccurate Load	0.206 0.206 0.275 0.275	0.217 0.217 -0.43 0.383	-0.54 -0.10 -0.39 -0.29	adaptive non-adaptive

Table 6.1 **Summary of results.**

Discussion of Results

The simulations were run to test the adaptive controllers ability to converge to the actual parameters, tracking under inertial parameter error, friction loading, and load mismatch. Simulations showed that Craig's adaptive controller could outperform the non-adaptive controller under all no-load conditions. Neither the adaptive nor the non-adaptive controller performed well under loading conditions when no load information was included in the controller model. When the controller had an a priori load value, the adaptive scheme performed better than the non-adaptive scheme. In terms of a priori load information, the load must be known within several tenths of a kilogram. In the case of the third joint the maximum error grew slightly, but the final error was much better than the non-adaptive error

results. These results indicate that the adaptive scheme performs well when small mismatches exist between the actual and predicted load.

The motor inertial parameters were chosen for the simulations because they have the greatest effect upon the PUMA mass matrix. Adaptive simulations run with link inertias instead of actuator inertias did not perform as well. Two reasons the adaptation of the link parameters did not perform as well as the actuator inertia adaptations are parameter size, and position relationships. In terms of size, the actuator inertias at joint's two through six are greater than any individual link inertia at that same joint. At the first link two inertias exist that are as large as the actuator inertia. In terms of position, the actuator inertias remain constant over the course of the trajectory. Conversely, the influence of most of the link inertias depends upon the position of the PUMA arm. For example, the I_3 inertia is multiplied by $\cos^2 \theta_2$ and then added into the D_{11} term of the mass matrix. Thus, depending on the value of θ_2 the I_3 term may have very little effect on the dynamics.

The inability of the adaptive scheme to give good results under loading conditions is not surprising. First, the actuator inertia terms should remain constant under load conditions. However, the addition of a load will increase the terms of the manipulator mass matrix. The D_{11} term increases less than the D_{22} and D_{33} terms for the trajectory chosen. As a result, the errors for the first joint are considerably less than those of the second and third joints. On the other hand, the D_{22} and D_{33} terms grow large enough under half and full-load that the estimated I_{m2} and I_{m3} terms both saturated to their upper limits. Increasing the upper limits for these two parameters did not improve position tracking. The adaptation of the second joint under half and full load actually degrades below the non-adaptive results as the trials increase. The results for the third joint under full-load are unacceptable as the final position errors exceed -8.5 degrees. For the second and third joints the adaptation scheme is trying to recognize a value from a PUMA model whose inertia is increasing with load.

The results of the no-load simulations tend to support Leahy et. al.'s statement that more accurate actuator inertias and not link inertias improve the tracking accuracy of the computed-torque scheme implemented on the PUMA robot (Leahy

et. al. 1989). Repeated trials showed that the actuator inertias adapted to values that reduced the position errors compared to the non-adaptive cases. The no-load experiments with friction showed that Craig's algorithm not only reduced the maximum error but also the final position error. Thus, for the PUMA manipulator, Craig's adaptive computed-torque scheme produces valid results when load conditions are known and trajectories can be repeated for learning purposes.

Finally, since Craig proposed his adaptive scheme in 1987 (Craig et. al. 1987), several additions to the algorithm have been published. First, in actual implementation it is very difficult to actually measure the acceleration of a joint. Middleton and Goodwin showed that proper filtering of the system error given in equation (5.22) effectively removes the dependence of the adaptation on acceleration (Middleton and Goodwin 1988). Recall that in this simulation the joint acceleration appears in the W matrix used in the parameter adaptation rule. Spong and Ortega showed that it is not necessary to bound the parameters chosen for the adaptation (Spong and Ortega 1988). However, caution should be used in relaxing Craig's bound used in parameter updating. First, without the bound it becomes difficult to select a Γ value for that parameter. The bound helps show when a Γ element is too large, which might cause the parameter to oscillate between its bounding values. Second, the bound helps define cases where the adaptation is probably not useful. For example, the load cases shown above indicate situations where parameter adaptation is not useful. Finally, if a priori information is known about a parameter, then the inclusion of this information into the adaptation rule should improve the rule. A summary of these additions to Craig's computed-torque scheme can be found in Ortega and Spong (1988).

CHAPTER SEVEN

Conclusion

This thesis has outlined the algorithms needed to create a simulation of the PUMA 560 robot arm. The first four Chapters describe the kinematics, path planning, and dynamics routines. The Fifth Chapter outlines many of the trajectory control algorithms developed over the years. The Sixth Chapter describes the PUMA simulation and gives plots of the results. The simulation compares the tracking abilities of Craig's adaptive computed-torque controller and a non-adaptive computed-torque controller.

A rigid body robot simulation requires kinematic, path planning, and dynamic routines. The kinematic programs describe the spatial relationships between the various robot joints, base, and tip. For the PUMA the forward kinematic routine calculates the individual link transformation matrices from the six desired joint angles. These link transformations are calculated using the Denavit-Hartenberg representation describing the PUMA in its zero position. The inverse kinematic algorithm solves for a set of PUMA joint angles from the matrix transformation converting the six link coordinates into the base coordinates. The inverse kinematic routine is used to convert desired hand positions given in the Cartesian space into the joint space.

The path planning routines generate the trajectories the PUMA must follow when moving from an initial to a final point. Path planning can take place in either the joint or Cartesian space. Most dynamics and control algorithms require robot positions, velocities, and accelerations specified in the joint space as input. As a result, path planning in the joint space give outputs that directly apply to the robot dynamics and control algorithms. The difficulty with the joint space schemes is that they give trajectories that are difficult to describe in Cartesian space. Thus, Cartesian space schemes give results that describe trajectories useful for obstacle avoidance and easier visualization. However, a Jacobian transformation is required to convert the Cartesian velocities and accelerations into the joint space.

Depending upon the robot arm configuration, the Jacobian might be singular in which case no conversion to the joint space exists. Thus, joint space schemes give easily implementable results, while the Cartesian space schemes give results that are easy to visualize.

The study of dynamics merges the motion of the robot with the forces and torques required to achieve that motion. Two popular algorithms used to develop the dynamic equations are the Lagrange-Euler method and the Newton-Euler method. The Lagrange-Euler method makes use of the Lagrangian operator to relate the robot motion with the necessary torque. The set of equations calculated from the Lagrange-Euler method exhibit the dynamic structure of the robot arm. This dynamic structure allows the engineer to identify the parts of the dynamic model requiring greater control emphasis. The drawback of the Lagrange-Euler method is the large number of calculations needed to find a single set of output torques. A far more efficient dynamic algorithm is the Newton-Euler method. The Newton-Euler algorithm uses a recursive structure to reduce the calculations needed to solve the dynamics problem. However, the Newton-Euler's recursive nature hides the dynamic structure of the robot's links. An effective compromise between numerical efficiency and dynamic structure is given by the explicit dynamic model. In the explicit dynamic PUMA model the efficiency of the Newton-Euler algorithm is achieved, while the structure of the Lagrange-Euler method is maintained. Any of these dynamic models can be used to solve the forward or inverse dynamics problems. The forward dynamic routine calculates the joint torques from the input joint position, velocity, and acceleration. The forward solution is used commonly as the basis for the model-based trajectory controllers. The inverse dynamics routine calculates the joint acceleration achieved after applying a set of joint torques at a specific position. Inverse dynamics is commonly used in robot simulation to model the actual dynamic plant.

The field of robot control includes the areas of trajectory control, force control, and hybrid control. Trajectory control applies to both force and hybrid control, and is easy to simulate on a computer. Thus, it makes sense to investigate

schemes that improve the trajectory tracking abilities of a robot controller. Trajectory control algorithms have been derived from optimal control, nonlinear control, classical control, modern control, and adaptive control. Many industrial controllers are based on PD and PID controllers. Other popular schemes are model based in nature and attempt to match a computer simulated controller model with the actual robot dynamics. The computed-torque technique is a model based scheme that uses nonlinear feedback to decouple the robot dynamics and a PD controller to smooth errors. Craig developed an adaptive computed-torque controller that tries to match the desired model parameters with the actual dynamic models. The advantage of Craig's scheme over previous adaptive trajectory controllers is its derivation based on stability theory. Lyapunov stability theory is used to derive a controller that minimizes the system error. In addition, the parameters will converge to their desired values if the specified trajectory is persistently exciting. This persistent excitation is a function of the regression matrix defined by the parameters chosen for adaptation.

Finally, a series of simulations were run to compare the performance of Craig's adaptive computed-torque controller with the non-adaptive computed-torque controller. These simulations analyzed the controllers under no-load and load conditions, mismatched load conditions, and friction considerations. The following comments summarize the main results of the simulations.

1. In the case of the PUMA 560, the first three actuator inertias represent the best parameters for adaptation. Since the link inertias are smaller than the actuator inertias, they have much less effect upon the PUMA's position error.

2. The ability of the algorithm to adapt depends on the persistent excitation of the chosen path. The adaptive scheme is repeated several times to give the controller enough information to learn the parameters. Under the proper load conditions, the adaptive scheme requires only five repeated trials to greatly reduce the joint position error.

3. The adaptive algorithm performs well when a priori knowledge of the load is known. The adaptive controller outperformed the non-adaptive controller under no-load and small load mismatch conditions. Under unknown load conditions the

error profiles for both the adaptive and non-adaptive case were poor. The second joint is the most sensitive to loading as the adaptive controller actually produced greater errors at this joint than the non-adaptive controller.

4. Simulations run with friction included in the PUMA model did not drastically change the tracking abilities of either scheme.

5. Craig's adaptive scheme reduces the joint position errors under known, but not exact, load conditions. Therefore, Craig's adaptive controller should be considered for actual testing on the PUMA 560.

The information contained within this thesis provides the basis for future research in adaptive robot control. First, the load calculations within the current PUMA model need improving. Improved load modeling would allow better analysis of Craig's adaptive controller under load conditions. Second, improved load modeling also allows for load estimation within the adaptive controller. Third, the speed of the adaptive algorithm can be improved by using a reduced dynamic model in the controller. Finally, joint acceleration measurements are required to calculate the W regression matrix. Do the techniques, eliminating the need for joint acceleration, developed by Middleton and Goodwin (1988) give valid results?

In conclusion, this thesis was written as a guide to formulating a robot simulation and using it to test control algorithms. The PUMA 560 was chosen for the simulation because it is a very popular industrial robot. Although slow, the PUMA simulation does highlight many of the difficulties incurred when trying to control a robot. It is hoped that this work will help others improve their understanding of robotics before attempting to develop a rigid body robot simulation.

APPENDIX A

Forward and Inverse Kinematic Solutions

The first section of this Appendix gives the forward kinematic transformations relating individual links. The second section gives the inverse kinematic solutions for the PUMA 560.

Forward Kinematic Solution

The forward kinematic matrices relating individual links are found by substituting the Denavit-Hartenburg parameters into the matrix

$$T_i^{i-1} = \begin{pmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i c\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (A.1)$$

Substituting the values of Table 2.1 into equation (A.1) gives the following six link transformations.

$$T_1^0 = \begin{pmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_2^1 = \begin{pmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ -s\theta_2 & -c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_3^2 = \begin{pmatrix} c\theta_3 & -s\theta_3 & 0 & a_2 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_4^3 = \begin{pmatrix} c\theta_4 & -s\theta_4 & 0 & a_3 \\ 0 & 0 & -1 & -d_4 \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_5^4 = \begin{pmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s\theta_5 & -c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_6^5 = \begin{pmatrix} c\theta_6 & -s\theta_6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_6 & c\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The overall transformation relating link 6 coordinates to the base coordinates is given by

$$T_6^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5.$$

$$T_6^0 = \begin{pmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The coefficients of the T_6^0 terms are given as

$$r_{11} = c_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] - s_1(s_4c_5c_6 + c_4s_6)$$

$$r_{12} = c_1[c_{23}(-c_4c_5s_6 - s_4c_6) + s_{23}s_5s_6] - s_1(-s_4c_4s_6 + c_4c_6)$$

$$r_{13} = c_1(c_{23}c_4s_5 + s_{23}c_5) - s_1s_4s_5$$

$$p_x = c_1(a_3c_{23} + d_4s_{23} + a_2c_2) - s_1(d_2 + d_3)$$

$$r_{21} = s_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] + c_1(s_4c_5c_6 + c_4s_6)$$

$$r_{22} = s_1[c_{23}(-c_4c_5s_6 - s_4c_6) + s_{23}s_5s_6] + c1(-s_4c_4s_6 + c_4c_6)$$

$$r_{23} = s_1(c_{23}c_4s_5 + s_{23}c_5) + c_1s_4s_5$$

$$p_y = s_1(a_3c_{23} + d_4s_{23} + a_2c_2) + c_1(d_2 + d_3)$$

$$r_{31} = -s_{23}(c_4c_5c_6 - s_4s_6) - c_{23}s_5c_6$$

$$r_{32} = -s_{23}(-c_4c_5s_6 - s_4c_6) + c_{23}s_5s_6$$

$$r_{33} = -s_{23}c_4s_5 + c_{23}c_5$$

$$p_z = -a_3s_{23} + d_4c_{23} - a_2s_2$$

Inverse Kinematic Solution

The inverse kinematic equations are found by multiplying matrices and equating like terms. In the case of the inverse kinematics the transformation T_6^0 is known. The problem is finding the θ values that achieve the T_6^0 transformation. Let $\hat{T}_6^0$ be the symbolic representation for the T_6^0 transformation. The θ_1 term is found by equating the (2,4) terms of the equation

$$[\hat{T}_1^0]^{-1}\, T_6^0 = \hat{T}_6^1. \tag{A.2}$$

The relation gives

$$-s_1 p_x + c_1 p_y = d_2 + d_3.$$

Defining k as

$$k = \sqrt{p_x^2 + p_y^2 - (d_2 + d_3)^2}$$

then the θ_1 is given as

$$\theta_1 = \arctan \frac{p_y}{p_x} - \arctan \frac{d_2 + d_3}{k}.$$

With θ_1 known, T_1^0 is also known. Equating the (1,4) terms and the (3,4) terms of (A.2) and defining k and ρ as

$$k = \frac{p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2 - d_4^2 - (d_2 + d_3)^2}{2a_2}$$

and

$$\rho = \sqrt{a_3^2 + d_4^2}$$

then θ_3 is given as

$$\theta_3 = \arctan \frac{k}{\sqrt{\rho^2 - k^2}} - \arctan \frac{a_3}{d_4}.$$

Equating the (1,4) and (2,4) terms of the equation

$$[\hat{T}_3^0]^{-1} T_6^0 = \hat{T}_6^3 \tag{A.3}$$

gives the $\theta_2 + \theta_3$ term from

$$c_{23} = \frac{(c_1 p_x + s_1 p_y)(a_2 c_3 + a_3) + p_z(a_2 s_3 + a_3)}{p_z^2 + (c_1 p_x + s_1 p_y)(c_1 p_x + s_1 p_y)}$$

$$s_{23} = \frac{(c_1 p_x + s_1 p_y)(a_2 s_3 + d_4) - p_z(a_2 c_3 + a_3)}{p_z^2 + (c_1 p_x + s_1 p_y)(c_1 p_x + s_1 p_y)}.$$

Since θ_3 is already known, θ_2 is found as

$$\theta_2 = \arctan \frac{s_{23}}{c_{23}} - \theta_3.$$

Equating the (1,3) and (3,3) terms of (A.3) and assuming s_5 is not equal to zero, leads to

$$s_4 = -s_1 r_{13} + c_1 r_{23}$$

and

$$c_4 = c_1 c_{23} r_{13} + s_1 c_{23} r_{23} - s_{23} r_{33}.$$

The θ_4 term is

$$\theta_4 = \arctan \frac{s_4}{c_4}.$$

Note that the r_{ij} terms are known from the overall T_6^0 transformation. Now equating the (1,3) terms of the equation

$$[\hat{T}_4^0]^{-1} T_6^0 = \hat{T}_6^4 \tag{A.4}$$

gives

$$s_5 = r_{13}(c_1 c_4 c_{23} - s_1 s_4) + r_{23}(s_1 c_4 c_{23} + c_1 s_4) - r_{33} c_4 s_{23}$$

and

$$c_5 = r_{13} c_1 s_{23} + r_{23} s_1 s_{23} + r_{33} c_{23}.$$

The resulting θ_5 term is

$$\theta_5 = \arctan \frac{s_5}{c_5}.$$

Finally, equating the (1,1) and (3,1) terms of the following equation

$$[\hat{T}_5^0]^{-1} T_6^0 = \hat{T}_6^5 \tag{A.5}$$

gives

$$s_6 = r_{11}(-c_1 s_4 c_{23} - s_1 c_4) + r_{21}(-s_1 s_4 c_{23} + c_1 c_4) + r_{31} s_{23} s_4$$

and

$$c_6 = r_{11}(c_5(c_1 c_4 c_{23} - s_1 s_4) - s_5 c_1 s_{23}) - r_{31}(s_{23} c_4 c_5 + s_5 c_{23}) +$$

$$r_{21}(c_5(s_1 c_4 c_{23} + c_1 s_4) - s_5 s_1 s_{23}).$$

Thus, the final angle θ_6 is

$$\theta_6 = \arctan \frac{s_6}{c_6}.$$

APPENDIX B

Summary of Routines

The following list defines the input and output variables used in the simulation routines.

List of variables

q: column vector of joint positions θ.
qd: column vector of joint velocities $\dot{\theta}$.
qdd: column vector of joint accelerations $\ddot{\theta}$.
inertia: column of 23 link inertias.
imotor: column of six actuator or motor inertias.
gravity: column of six gravity constants.
m6ld: load mass added to the sixth link.
bias: constant used to bias the link and motor inertias.
b1-b6: Coriolis matrices for each link B.
centr: centrifugal matrix for all links C.
gvect: column vector of gravity torques at each joint G.
m: manipulator mass matrix D.
friction: column vector of joint frictions.
torque: column vector of the joint torques τ.
px: desired x position for kinematics.
py: desired y position for kinematics.
pz: desired z position for kinematics.
alpha: required for inverse kinematics final rotation.
beta: required for inverse kinematics final rotation.
gamma: required for inverse kinematics final rotation.
trans: individual link transformations concatenated together.
fowkin: transformation T_6^0.

rotate: rotation matrices R_{i+1}^{i} concatenated together.
pnew: column vector of new adapted parameters.
pold: column vector of old adapted parameters.
e: column vector of joint position errors E.
ed: column vector of joint velocity errors $\dot{E}$.
inc: time increment.
a0-a5: column vectors of coefficients of path planning polynomial.
timf: total time for path execution.

The following list defines the program statement and gives a brief description of the program. These programs can only be run in the Matrix_x environment. Before running, each program has to be DEFINEd for use in the Matrix_x environment. For help in using the DEFINE function in Matrix_x, see the Matrix_x User's Guide, Version 6.

The programs given below represent those basic to the PUMA 560 simulation. However, not all of the programs used in the simulation are described below. All programs required by the simulation are DEFINEd in the program PUMA. To DEFINE these programs, first type DEFINE 'PUMA' cr (cr means carriage return), and then execute 'PUMA' by typing
[output]=PUMA(1) cr.

List of Routines

[inertia,gravity,imotor]=CONST(m6ld,bias)

Program CONST calculates the constant terms used in the explicit dynamic model routines of the PUMA 560 robot. Variables m6ld and bias are used to simulate loads and inertia errors.

[m]=MAPUMA(q,inertia,imotor)

Program MAPUMA calculates the manipulator mass matrix (kinetic energy matrix), D, for the PUMA 560 robot arm. This routine uses the explicit model dynamics to arrive at the solution.

[b1,b2,b3,b4,b5,b6]=COPUMA(q,inertia)

Program COPUMA calculates the six Coriolis matrices that are needed in the explicit dynamic solution of the PUMA 560 arm.

[centr]=CEPUMA(q,b1,b2,b4,b5,inertia)

Program CEPUMA calculates the centrifugal torque matrix for the explicit PUMA 560 robot model. Notice that CEPUMA require four of the outputs from COPUMA. Therefore, COPUMA should always be run before CEPUMA.

[gvect]=GRPUMA(q,gravity)

Program GRPUMA calculates the vector of torques due to gravitation. These gravitational torques are calculated using the explicit PUMA model.

[friction]=FRIC(qd)

Program FRIC calculates the friction contribution to the joints of the PUMA 560 robot.

[qdd,m]=DYPUMA(q,qd,torque,m6ld)

Program DYPUMA uses the Newton-Euler solution to solve for the forward dynamic equations for the PUMA 560 arm. The program solves for the joint accelerations when a set of joint torques is applied. In addition, the mass matrix is also output.

[torque]=IDPUMA(q,qd,qdd,bias,m6ld)

Program IDPUMA solves the inverse dynamic equations for the PUMA 560 robot arm. The Newton-Euler equations are used to find the desired torques at each joint necessary to achieve a certain joint acceleration.

[q]=IKPUMA(alpha,beta,gamma,px,py,pz)

Program IKPUMA calculates the inverse kinematic solutions for the PUMA 560 arm. The Z-Y-X Euler angle system is used in deciding the rotation of the wrist for the actual T_6^0 transformation. Another system such as the roll, pitch and yaw transformation can be substituted for the Z-Y-X Euler transformation. The alpha, beta and gamma angles are used to find the Z-Y-X Euler transformation.

[trans,fowkin]=KPUMA(q)

Program KPUMA calculates the forward kinematic transformations for the PUMA 560 arm. KPUMA arrives at these equations from manipulations of the Denavit-Hartenberg notation. To find the desired rotations, KPUMA calls program RPUMA.

[rotate]=RPUMA(q)

Program RPUMA calculates the link rotation matrices for the PUMA 560 arm. These link rotation matrices are concatenated together into the single output 'rotate'.

[pnew]=ADPT(qdd,pold,m,ed,e,inc)

Program ADPT performs the adaptive updating of the desired parameters. Craig's routine is used to update the old parameter estimates, pold, with the new parameter estimates pnew.

[ao,a1,a2,a3,a4,a5]=JPLAN(qo,qdo,qddo,qn,qdn,qddn,timf)

Program JPLAN calculates the coefficients needed for a fifth order polynomial used for path planning in the joint space of the PUMA 560 robot. Note that the qo and qn terms indicate old or initial, and new or final joint positions respectively.

REFERENCES

An, C.H., C.G. Atkenson, J.S. Griffiths, and J.M. Hollerbach. 1989. Experimental Evaluation of Feedforward and Computed Torque Control. *IEEE Transactions on Robotics and Automation.* Vol. 5, No. 3, 368-373.

An, C.H., C.G. Atkenson, and J.M. Hollerbach. 1986. Experimental Determination of the Effects of Feedforward Control on Trajectory Tracking Errors. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco CA, 55-60.

Armstrong, B. 1987. On Finding Exciting Trajectories for Identification Experiments Involving Systems with Non-Linear Dynamics. *Proceedings of the IEEE International Conference on Robotics and Automation.* Raleigh, NC, 1131-1139.

Armstrong, B., O. Khatib, and J. Burdick. 1986. The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 Arm. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, 510-518.

Asada, H. and K. Hara. 1986. Load Sensitivity Analysis and Adaptive Control of a Direct-Drive Arm. *Proceedings of the American Control Conference*, Seattle WA, 799-804.

Asare, H.R., and D.G. Wilson. 1987. Evaluation of Three Model Reference Adaptive Control Algorithms for Robotic Manipulators. *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1531-1542.

Åström, K.J. and B. Wittenmark. 1988. *Adaptive Control.* Reading Mass.: Addison-Wesley Publishing Company.

Bejczy, A.K., T.J. Tarn, X. Yun, and S. Han. 1985a. Nonlinear Feedback Control of PUMA 560 Robot Arm By Computer. *Proceedings of the 24th IEEE Conference on Decision and Control*, Fort Lauderdale, FL, 1680-1688.

Bejczy, A.K., T.J. Tarn, and Y.L. Chen. 1985b. Robot Arm Dynamic Control by Computer. *Proceedings of the IEEE Conference on Robotics and Automation*, St. Louis, MO, 960-970.

Canudas, C., K.J. Åström, and K. Braun. 1986. Adaptive Friction Compensation in DC Motor Drives. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, 1556-1561.

Chen, Y. 1989. Replacing a PID Controller by a Lag-Lead Compensator for a Robot-A Frequency Response Approach. *IEEE Transactions on Robotics and Automation.* Vol. 5, No. 2, 174-182.

Craig, J.J. 1986. *Introduction to Robotics: Mechanics and Control.* Reading, Mass.: Addison-Wesley Publishing Company.

Craig, J.J. 1988. *Adaptive Control of Mechanical Manipulators.* Reading, Mass.: Addison-Wesley Publishing Company.

Craig, J.J., P. Hsu, and S.S. Sastry. 1987. Adaptive Control of Mechanical Manipulators. *The International Journal of Robotics Research.* Vol. 6, No. 2, 16-28.

Denavit, J., and R.S. Hartenberg. 1955. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. *Journal of Applied Mechanics*, 215-221.

Dubowsky, S., and D.T. DesForges. 1979. The Application of Model Referenced Adaptive Control to Robotic Manipulators. *Transactions of the ASME Journal of Dynamic Systems, Measurement and Control.* Vol. 101, 193-200.

Egeland, O. 1986. On the Robustness of the Computed Torque Technique in Manipulator Control. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, 1203-1208.

Elgazzar, S. 1985. Efficient Kinematic Transformations for the PUMA 560 Robot. *IEEE Journal of Robotics and Automation*, Vol. RA-1, No. 3, 142-151.

Featherstone, R. 1983a. The Calculation of Robot Dynamics Using Articulated-Body Inertias. *The International Journal of Robotics Research*, Vol. 2, No. 1, 13-30.

Featherstone, R. 1983b. Position and Velocity Transformations Between Robot End Effector Coordinates and Joint Angles. *The International Journal of Robotics Research*, Vol. 2, No. 2, 35-45.

Featherstone, R. 1987. *Robot Dynamics Algorithms.* Boston, Mass.: Kluwer Academic Publisher.

Guo, L., and J. Angeles. 1989. Controller Estimation for the Adaptive Control of Robotic Manipulators. *IEEE Transactions on Robotics and Automation.* Vol 5, No. 3, 315-323.

Haykin, S. 1986. *Adaptive Filter Theory.* Englewood Cliffs N.J.: Prentice Hall.

Hemami, H., and P.C. Camana. 1986. Nonlinear Feedback in Simple Locomotion Systems. *IEEE Transactions on Automatic Control.* December, 855-860.

Hollerbach, J.M. 1980. A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-10, No. 11, 730-736.

Hsia, T.C. 1986. Adaptive Control of Robot Manipulators - A Review. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco CA, 183-189.

Integrated Systems Inc. 1986. *$Matrix_x$ User's Guide.* Version 6.0, Santa Clara, CA.

Izaguirre, A., and R.P. Paul. 1985. Computation of the Inertial and Gravitational Coefficients of the Dynamics Equations for a Robot Manipulator with a Load. *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis MO, 1024-1032.

Kahn, M.E., and B. Roth. 1971. The Near Time Control of Open-Loop Articulated Kinematic Chains. *ASME Journal of Dynamic Systems, Measurement and Control.* September, 164-172.

Khosla, P.K. 1989. Categorization of Parameters in the Dynamic Robot Model. *IEEE Transactions on Robotics and Automation.* Vol. 5, No. 3, 261-168.

Khosla, P.K., and T. Kanade. 1986. Experimental Evaluation of the Feedforward Compensation and Computed-Torque Control Schemes. *Proceedings of the American Control Conference*, Seattle WA, 790-798.

Khosla, P.K., and T. Kanade. 1989. Real-Time Implementation and Evaluation of Computed-Torque Scheme. *IEEE Transactions On Robotics and Automation.* Vol. 5, No. 2, 245-253.

Koivo, A.J., and T.H. Guo. 1983. Adaptive Linear Controller for Robotic Manipulators. *IEEE Transactions on Automatic Control.* Vol. AC-28, No. 2, 162-171.

Kreutz, K., and A. Lokshin. 1988. Load Balancing and Closed Chain Multiple Arm Control. *JPL Engineering Memorandum 347-88-233.*

Leahy, M.B., L.M. Nugent, K.P. Valavanis, and G.N. Saridis. 1986a. Efficient Dynamics for a PUMA 600. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco CA, 519-524.

Leahy, M.B., and G.N. Saridis. 1986. The Effects of Unmodeled Forces on Robot Control. *Proceedings of the 25th Conference on Decision and Control*, Athens, Greece, 403-408.

Leahy, M.B., K.P. Valavanis, and G.N. Saridis. 1986b. The Effects of Dynamic Models on Robot Control. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, 49-54.

Leahy, M.B., K.P. Valavanis, and G,N, Saridis. 1989. Evaluation of Dynamic Models for PUMA Robot Control. *IEEE Transactions on Robotics and Automation*, Vol. 5, No. 2, 242-245.

Lee, B.H. 1989. Constraints Identification in Time-Varying Obstacle Avoidance for Mechanical Manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 1, 140-143.

Lee, C.S.G. 1982. Robot Arm Kinematics, Dynamics and Control. *Computer*, Vol. 18, No. 3, 62-80.

Lee, C.S.G., and M.J. Chung. 1982. An Adaptive Control Strategy for Computer-Based Manipulators. *Proceedings of the 21st Conference on Decision and Control*, Orlando, Fl., 95-100.

Lee, C.S.G., T.N. Mudge, and J.L. Turney. 1982. Hierarchical Control Structure Using Special Purpose Processors for the Control of Robot Arms. *Proceedings of the Pattern Recognition and Image Processing Conference.* 634-640.

Lee, C.S.G., B.H. Lee, and R. Nigam. 1983. Development of the Generalized d'Alembert Equations of Motion for Mechanical Manipulators. *Proceedings of the 22nd Conference on Decision and Control.* San Antonio, TX, 1205-1210.

Lefschetz, S. 1965. *Stability of Nonlinear Control Systems*, New York: Academic Press, 114-118.

Lin, K.Y., and M. Eslami. 1986. Robust Adaptive Controller Designs for Robot Manipulator Systems. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco CA, 1209-1215.

Lozano-Perez, T., and M.A. Wesley. 1979. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Communications of the ACM*, Vol 22, Number 10, 560-570.

Luh, J.Y.S., M.W. Walker, and R.P.C. Paul. 1980a. On-Line Computation for Mechanical Manipulators. *Journal of Dynamic Systems, Measurement, and Control*, Vol. 102, 69-76.

Luh, J.Y.S., M.W. Walker, and R.P.C. Paul. 1980b. Resolved-Acceleration Control of Mechanical Manipulators. *IEEE Transactions on Automatic Control.* Vol AC-25, No. 3, 468-474.

Lyapunov, A.M. 1892. On the General Problem of Stability in Motion. *Soviet Union: Karkov Mathematical Society.*

McInnis, B.C., and C.K.F. Liu. 1986. Kinematics and Dynamics in Robotics: A Tutorial Based Upon Classical Concepts Of Vector Mechanics. *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 4, 181-187.

Middleton, R.H., and G.C. Goodwin. 1988. Adaptive Computed Torque Control for Rigid Link Manipulators. *Systems and Control Letters*, Vol. 10, 9-16.

Neuman, C.P., and J.J. Murray. 1987. The Complete Dynamic Model and Customized Algorithm of the PUMA Robot. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-17, No. 4, 635-644.

Orin, D.E., and W.W. Schrader. 1984. Efficient Computation of the Jacobian for Robot Manipulators. *The International Journal of Robotics Research*, Vol 3, No. 4, 66-75.

Ortega, R., and M.W. Spong. 1988. Adaptive Motion Control of Rigid Body Robots: A Tutorial. *Proceedings of the 27th Conference on Decision and Control*, Austin Texas, 1575-1584.

Paul, R.P. 1972. Modeling, Trajectory Calculation, and Servoing of a Computer Controlled Manipulator. *Stanford Artificial Intelligence Laboratory Memo 177*, Stanford CA, Stanford University.

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

Paul, R.P. 1981b. Kinematic Control Equations For Simple Manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-11, No. 6, 449-455.

Paul R.P., and C.N. Stevenson. 1983. Kinematics of Robot Wrists. *The International Journal of Robotics Research*, Vol. 2, No. 1, 31-38.

Paul, R.P., M. Rong and H. Zhang. 1983. The Dynamics of the PUMA Manipulator. *Proceedings of the American Control Conference*, San Francisco, CA, 491-495.

Rodriguez, G., and K. Kreutz. 1988. Recursive Mass Matrix Factorization and Inversion. *JPL Publication 88-11.*

Silver, W.M. 1982. On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators. *The International Journal of Robotics Research*, Vol. 1, No. 2. 60-70.

Slotine, J.J.E. 1985. The Robust Control of Robot Manipulators. *The International Journal of Robotics Research.* Vol. 4, No. 2, 49-64.

Slotine, J.J.E., and W. Li. 1987. On the Adaptive Control of Robot Manipulators. *The International Journal of Robotics Research.* Vol. 6, No. 3, 49-59.

Spong, M.W., and R. Ortega. 1988. On Adaptive Inverse Dynamics Control of Rigid Robots. *IEEE Journal of Robotics and Automation*, submitted.

Spong, M.W., and M. Vidyasagar. 1985a. Robust Nonlinear Control of Robot Manipulators. *Proceedings of the 24th IEEE Conference on Decision and Control.* Ft. Lauderdale, Fl., 1767-1772.

Spong, M.W., and M. Vidyasagar. 1985b. Robust Linear Compensator Design for Nonlinear Robotic Control. *Proceedings of the IEEE Conference on Robotics and Automation.* St. Louis, MO, 954-959.

Taylor, R.H. 1979. Planning and Execution of Straight Line Manipulator Trajectories. *IBM Journal of Research and Development*, Vol 23, No. 4, 253-264.

Tondu, B., and H.I. El-Zorkany. 1986. Experimental Identification and Validation of a Model for a Robot's Trajectory Generator. *Theory of Robots. Selected Papers from the IFAC/IFAP/IMACA.* Oxford, U.K.: Pergamon Press, 331-335.

Tourassis, V.D., and C.P. Neuman. 1985. The Inertial Characteristics of Dynamic Robot Models. *Mechanism and Machine Theory*, Vol. 20, No. 1, 41-52.

Walker, M.W., and D.E. Orin. 1982. Efficient Dynamic Computer Simulation of Robotic Mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, Vol. 104, 205-211.

Walters, R.G., and M.M. Bayoumi. 1982. Application of a Self-Tuning Pole-Placement Regulator to an Industrial Manipulator. *Proceedings of 21st IEEE Conference on Decision and Control*, Orlando, FL, 323-329.

Whitney, D.E. 1969. Resolved Motion Rate Control of Manipulators and Human Prostheses. *IEEE Transactions on Man-Machine System*, Vol. MMS-10, No. 2, 47-53.

Wolovich, W.A. 1986. *Robotics: Basic Analysis and Design.* New York: Holt, Rinehart and Winston.