Center for Intelligent Robotic Systems for Space Exploration

Rensselaer Polytechnic Institute
Troy, New York 12180-3590

COMMAND GENERATOR TRACKER BASED DIRECT MODEL REFERENCE ADAPTIVE CONTROL OF A PUMA 560 MANIPULATOR

by

David C. Swift

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering Department
Troy, New York 12180-3590

August 1992

CIRSSE REPORT #124

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express many thanks to my thesis advisor, Dr. Howard Kaufman, for his patience and guidance, and also to the CIRSSE faculty and staff whose comments were appreciated. A special thanks to the CIRSSE/CTOS/MCS/8th Floor clan who made my stay at Rensselaer Polytechnic Institute very enjoyable. I would also like to thank my family, Mom, Dad, Todd, and Jason. Finally, I would like to thank my wife Linda and my dog Sebastian whose sacrifices have made my stay at RPI possible.

ABSTRACT

This project presents the results of controlling a PUMA 560 Robotic Manipulator using a Command Generator Tracker (CGT) Based Model Reference Adaptive Controller (DMRAC). The goal of the DMRAC algorithm is to asymptotically force the plant output to follow a known reference model output with dynamics chosen by the designer. The development of the DMRAC algorithm from its CGT roots is discussed. Initially, the DMRAC algorithm was run in simulation using a detailed dynamic model of the PUMA 560. The algorithm was tuned on the simulation and then used to control the manipulator using minimum jerk trajectories as the desired reference inputs. The ability to track a trajectory in the presence of load changes was also investigated in the simulation.

When satisfactory performance in simulation was achieved, the DMRAC algorithm was recoded to run on an actual PUMA 560 Manipulator in the Center for Intelligent Systems for Space Exploration (CIRSSE) Testbed using the newly developed CTOS/MCS software package. A discussion of the CIRSSE Testbed, CTOS, and MCS is also included. As with the simulation runs, the ability to track a trajectory in the presence of dynamic load changes was investigated using the PUMA 560.

Satisfactory performance was achieved in both simulation and on the actual robot. The obtained responses showed that the algorithm was robust in the presence of sudden load changes. These results indicate that the DMRAC algorithm can be successfully applied to the control of robotic manipulators.

CHAPTER 1
Introduction

This project dealt with the application of a Direct Model Reference Adaptive Control algorithm to the control of a PUMA 560 Robotic Manipulator. This chapter will present some motivation for using Direct Model Reference Adaptive Control, followed by a brief historical review, the project goals, and a summary of the subsequent chapters.

1.1 Motivation for Using Direct Model Reference Adaptive Control

For robotic control, a control engineer may be faced with joint and link flexibilities, unknown manipulator dynamic parameters, non-linear joint interactions, and changing dynamics due to unknown and varying loads. Traditional robotic control algorithms have relied on explicit knowledge of the robotic parameters and dynamic equations [1, 2, 3, 4]. When a designer has limited knowledge of these parameters and interactions, it may be desirable to utilize adaptive techniques to reduce the effects of these problems.

As more robots are used for space applications, there will be an increased need for adaptive control because of the need to keep space robots light weight. This weight constraint introduces joint and link flexibilities into the control problem which may necessitate obtaining extensive model information and the synthesis of observers. Robotic manipulation of objects in space will present a manipulator with sometimes unknown and possibly varying load inertias which are most suitably handled by adaptive control methods.

Adaptive control techniques can provide a uniform solution to control problems involving plant parameter uncertainties and/or environmental uncertainties.

Specifically, Direct Model Reference Adaptive Control (DMRAC) offers the following benefits [5]:

- Lack of dependence on plant parameter estimates,
- asymptotically zero output error with all states bounded,
- direct applicability to multiple input-multiple output plants,
- sufficiency conditions which are independent of plant dimension,
- control calculation which does not require adaptive observers or the need for full state feedback,
- ease of implementation.

Because of these advantages, Direct Model Reference Adaptive Algorithms are a step towards uniform control of robotic manipulators.

1.2 Literature Review

Adaptive controllers can be divided into two categories, Indirect methods and Direct methods. Indirect adaptive methods rely on estimates of the plant parameters which are then utilized to form the control to be applied to the plant. This two stage process of identification and control requires the implementation of explicit parameter identifiers, or observers. In contrast, the Direct methods do not explicitly try to identify the plant parameters. Rather, they directly adjust the plant control using only plant input and output signals. This project will deal with a Direct method of adaptive control.

A well known Direct adaptive control method is the direct version of the model reference adaptive controller or DMRAC. Model reference control deals with matching the response of a plant to that of some desired reference model [6]. The

reference model is designed such that it takes into account the desired plant design specifications. The desired reference inputs are fed to the reference model which responds in a known fashion according to the design specifications. In a properly designed DMRA controller, an adaptive mechanism drives the plant outputs to follow the reference model outputs.

Present DMRAC algorithms have evolved from one of three different approaches [5]:

- Full state access method [7] which assumes that all of the state variables can be measured,
- input-output methods which originated from augmented error signal concepts [8] which uses adaptive observers to reconstruct the state vector,
- Command Generator Tracker (CGT) based methods introduced by Sobel, Kaufman, and Mabius [9]

The later CGT based method [9] resulted in the benefits listed in Section 1.1 but had the drawback of requiring the plant under control to satisfy a positive real condition. Stability was guaranteed provided that there existed a feedback gain matrix which forced the plant to be almost strictly positive real (ASPR). That is, there exists a feedback gain matrix $\tilde{K}$ such that for a plant represented by the triple (A, B, C), $(C(sI - A + B\tilde{K}C)^{-1}B)$ is strictly positive real.

The major drawback to [9] was the necessity of satisfying the positive real condition. BarKana [10] proposed adding a feed-forward term in parallel with the original plant dynamics forming an augmented plant. This augmented plant then had to satisfy the original conditions of [9]. This approach was susceptible to steady-state tracking errors. By decreasing the contribution from the feed-forward filter, the true plant would more closely follow the augmented plant output. Asymptotic tracking was achievable by plants which were high gain feedback stabilizable.

For plants which are not high gain feedback stabalizable, Kaufman, Neat, and Steinvorth [5] proposed including the feed-forward into the reference model as well. This modification restored the desired asymptotic model following characteristics of [6]. This final version of the DMRAC algorithm was selected to control a PUMA 560 Manipulator.

1.3 Goal of This Project

The goal of this project was to test the ability of a DMRAC algorithm to control a PUMA 560 Manipulator. First, an accurate model of the PUMA 560 was formulated to test the DMRAC algorithm in simulation. Next, after verification in simulation, the algorithm was run on an actual PUMA 560 in the CIRSSE[1] Testbed using the newly developed CIRSSE Testbed Operating System and Motion Control System. For both the simulation runs and the actual hardware runs, the robot was commanded over typical minimum jerk trajectories and subjected to sudden payload variations.

1.4 Summary of Topics in Thesis

Below is a brief overview of the topics which will be covered in the subsequent chapters.

- *Chapter 2* will present the evolution of the DMRAC algorithm from the Basic DMRAC algorithm proposed by Sobel, Kaufman, and Mabius [9] to the final discretized version used to control the PUMA 560 Manipulator.

- *Chapter 3* will describe the simulation environment created with the Matlab[2] program along with some further details of the DMRAC algorithm.

[1]Center for Intelligent Systems for Space Exploration, Troy, NY

[2]Mathworks, Inc.

- *Chapter 4* will describe the process used to tune a DMRAC algorithm and present the results from some single joint evaluation simulations.
- *Chapter 5* will present the results of some six joint tracking simulations and show the effects of changing the tuning parameters.
- *Chapter 6* will present the results from simulation runs where the robot was subjected to static and dynamic load variations.
- *Chapter 7* will address the issue of reducing the trajectory tracking error.
- *Chapter 8* will describe the CIRSSE Robotic Testbed and the newly developed CIRSSE Testbed Operating System and Motion Control System. This chapter will also present some implementation issues.
- *Chapter 9* will present the results of actual runs on a PUMA 560 Manipulator in the CIRSSE Testbed. The robot was subjected to static and dynamic load variations and disturbances.
- *Chapter 10* will conclude the project with a summary and discussion of simulation and experimental results. Issues for future work will also be discussed.
- *Appendix A* lists the dynamic equations of motion used to simulate the PUMA 560.

CHAPTER 2
Development of the DMRAC Control Law

This chapter will present the development of the Direct Model Reference Adaptive Control Algorithm which was implemented on the CIRSSE Robotic Testbed. The motivating Command Generator Tracker theory will be discussed along with the basic DMRAC algorithm and its various extensions. The discretization of the control law for implementation on the CIRSSE Testbed will also be discussed. As the algorithm is expanded in the following sections, each new version will be labeled with some descriptive words separated by slashes and enclosed in angle brackets for later reference. For example, the final algorithm in this chapter is labeled – $\langle BASIC/FF^2/\alpha/bias/disc \rangle$.

2.1 Goal

The goal in the development of the continuous linear DMRAC algorithm is to control a plant such that the plant output follows the output of a desired reference model which is chosen by the designer. The plant is described by the following set of linear state space equations:

$$\dot{x}_p(t) = A_p x_p(t) + B_p u_p(t) \tag{2.1}$$

$$y_p(t) = C_p x_p(t) \tag{2.2}$$

where $x_p(t)$ is the $(n_p \times 1)$ plant state vector, $u_p(t)$ is the $(m_p \times 1)$ plant input vector, $y_p(t)$ is the $(q_p \times 1)$ plant output vector, and A_p, B_p, C_p are matrices of appropriate dimension.

Without explicit knowledge of A_p, B_p, and C_p, we wish to find a plant input, $u_p(t)$, such that the plant output, $y_p(t)$, asymptotically tracks the output of some

desired reference model, $y_m(t)$. The reference model is described by the following linear state space equations:

$$\dot{x}_m(t) = A_m x_m(t) + B_m u_m(t) \tag{2.3}$$

$$y_m(t) = C_m x_m(t) \tag{2.4}$$

where $x_m(t)$ is the $(n_m \times 1)$ reference model state vector, $u_m(t)$ is the $(m_m \times 1)$ reference model input vector, $y_m(t)$ is the $(q_m \times 1)$ reference model output vector, and A_m, B_m, C_m are matrices of appropriate dimension.

The reference model must have the same number of outputs as the plant ($q_m = q_p$) and is assumed to be bounded-input/bounded-output stable. The dimension of the reference model state vector, however, can be less than the dimension of the plant state vector. Thus, it is possible to simplify the on-line computation of the model by choosing $n_m < n_p$.

2.2 Command Generator Tracker Development

The DMRAC control law is based on the Command Generator Tracker (CGT) technique for non-adaptive controllers, proposed by Broussard and O'Brien [11], in which the plant parameters are assumed to be known. The following development will review the CGT concept and closely follows the development given in [6].

In this CGT method it is assumed that there exists an ideal plant with ideal state and input trajectories, $x_p^*(t)$ and $u_p^*(t)$, respectively, which occur when there is perfect output tracking (i.e., when $y_p(t) = y_m(t)$ for $t \geq 0$). By definition, this ideal plant satisfies the same dynamics as the actual plant, and the ideal plant output is identically equal to the model output. Thus,

$$\dot{x}_p^*(t) = A_p x_p^*(t) + B_p u_p^*(t) \tag{2.5}$$

$$y_p^*(t) = y_m(t) \Rightarrow C_p x_p^*(t) = C_m x_m(t) \tag{2.6}$$

We shall assume that the ideal plant state and input trajectories can be formed as linear functions of the model state and model input. Thus,

$$\begin{bmatrix} x_p^*(t) \\ u_p^*(t) \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} x_m(t) \\ u_m(t) \end{bmatrix} \tag{2.7}$$

Note that we will restrict $u_m(t)$ in (2.7) to be a constant input so derivatives of the model input will not be required. The ideal plant state equation (2.5) and the ideal output equation (2.6) can be combined, which yields,

$$\begin{bmatrix} \dot{x}_p^*(t) \\ y_p^*(t) \end{bmatrix} = \begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix} \begin{bmatrix} x_p^*(t) \\ u_p^*(t) \end{bmatrix} \tag{2.8}$$

Substituting equation (2.7) into equation (2.8) yields,

$$\begin{bmatrix} \dot{x}_p^*(t) \\ y_p^*(t) \end{bmatrix} = \begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix} \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} x_m(t) \\ u_m(t) \end{bmatrix} \tag{2.9}$$

If we differentiate the first equation in (2.7) and note that $u_m(t)$ is constant so $\dot{u}_m(t) = 0$, we have,

$$\dot{x}_p^*(t) = S_{11} \dot{x}_m(t) \tag{2.10}$$

Now we substitute the model dynamics (2.3) into (2.10) to obtain,

$$\dot{x}_p^*(t) = S_{11} A_m x_m(t) + S_{11} B_m u_m(t) \tag{2.11}$$

Combining equations (2.11) and (2.6) yields,

$$\begin{bmatrix} \dot{x}_p^*(t) \\ y_p^*(t) \end{bmatrix} = \begin{bmatrix} S_{11} A_m & S_{11} B_m \\ C_m & 0 \end{bmatrix} \begin{bmatrix} x_m(t) \\ u_m(t) \end{bmatrix} \tag{2.12}$$

Equating the right-hand sides of (2.9) and (2.12) yields,

$$\begin{bmatrix} S_{11}A_m & S_{11}B_m \\ C_m & 0 \end{bmatrix} \begin{bmatrix} x_m(t) \\ u_m(t) \end{bmatrix} = \begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix} \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} x_m(t) \\ u_m(t) \end{bmatrix} \tag{2.13}$$

Noting that $x_m(t)$ and $u_m(t)$ are arbitrary, we obtain,

$$\begin{bmatrix} S_{11}A_m & S_{11}B_m \\ C_m & 0 \end{bmatrix} = \begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix} \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \tag{2.14}$$

If we define

$$\begin{bmatrix} \Omega_{11} & \Omega_{12} \\ \Omega_{21} & \Omega_{22} \end{bmatrix} = \begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix}^{-1} \tag{2.15}$$

then the solution to (2.14) is,

$$S_{11} = \Omega_{11}S_{11}A_m + \Omega_{12}C_m \tag{2.16}$$

$$S_{12} = \Omega_{11}S_{11}B_m \tag{2.17}$$

$$S_{21} = \Omega_{21}S_{11}A_m + \Omega_{22}C_m \tag{2.18}$$

$$S_{22} = \Omega_{21}S_{11}B_m \tag{2.19}$$

For the inverse (2.15) to exist, the number of controls m_p must be equal to the number of outputs q_p. If $m_p > q_p$ then a pseudo-inverse will be required. Broussard and O'Brien [11] have shown that S_{ij} will exist if:

- u_m is a constant,
- the number of controls m_p is not less than the number of outputs q_p,
- the product of the i^{th} eigenvalue of Ω_{11} and the j^{th} eigenvalue of A_m does not equal unity for all i, j.

In summary, when perfect output tracking occurs, $y_p(t) = y_m(t)$ at $t = 0$, then the ideal control is given by (2.7) to be,

$$u_p^*(t) = S_{21}x_m(t) + S_{22}u_m \tag{2.20}$$

If perfect output tracking does not occur, $y_p(t) \neq y_m(t)$ at $t = 0$, we may still achieve asymptotic tracking if we include a stabilizing output feedback in the actual plant control law of the form,

$$u_p(t) = u_p^*(t) + K(y_m(t) - y_p(t)) \tag{2.21}$$

To see this, form the error between ideal and actual plant state as follows,

$$e_x(t) = x_p^*(t) - x_p(t) \tag{2.22}$$

Differentiating the error and substituting in (2.1) and (2.5) yields,

$$\begin{aligned} \dot{e}_x(t) &= \dot{x}_p^*(t) - \dot{x}_p(t) \\ &= A_p x_p^*(t) + B_p u_p^*(t) - A_p x_p(t) - B_p u_p(t) \\ &= A_p e_x(t) + B_p(u_p^*(t) - u_p(t)) \end{aligned} \tag{2.23}$$

Since $y_m(t) - y_p(t) = y_p^*(t) - y_p(t) = C_p(x_p^*(t) - x_p(t))$, (2.21) can be written as,

$$u_p(t) = u_p^*(t) + KC_p e_x(t) \tag{2.24}$$

Substituting (2.24) into the error equation (2.23) yields,

$$\dot{e}_x(t) = (A_p - B_p K C_p)e_x(t) \tag{2.25}$$

From linear control theory, (2.25) will approach zero if K is a stabilizing output feedback gain; therefore, we desire a controller for which $e_x(t) \to 0$ as $t \to \infty$.

Observe that when $e_x(t) = 0 \Rightarrow x_p(t) = x_p^*(t)$, then $C_p x_p(t) = C_p x_p^*(t)$. By definition (2.6), we have $C_p x_p^*(t) = C_m x_m(t)$. Therefore, $C_p x_p(t) = C_m x_m(t) \Rightarrow y_p(t) = y_m(t)$ which gives us asymptotic output tracking as $t \rightarrow \infty$.

In summary, when perfect output tracking does not occur, $y_p \neq y_m$ at $t = 0$, then the actual control to achieve asymptotic output tracking is found from (2.21) (substituting in (2.20) for $u_p^*(t)$) to be,

$$u_p(t) = S_{21} x_m(t) + S_{22} u_m(t) + K(y_m(t) - y_p(t)) \tag{2.26}$$

A block diagram of the non-adaptive controller, using (2.26), is shown in Figure 2.1. This algorithm will be referred to as – $\langle CGT \rangle$.

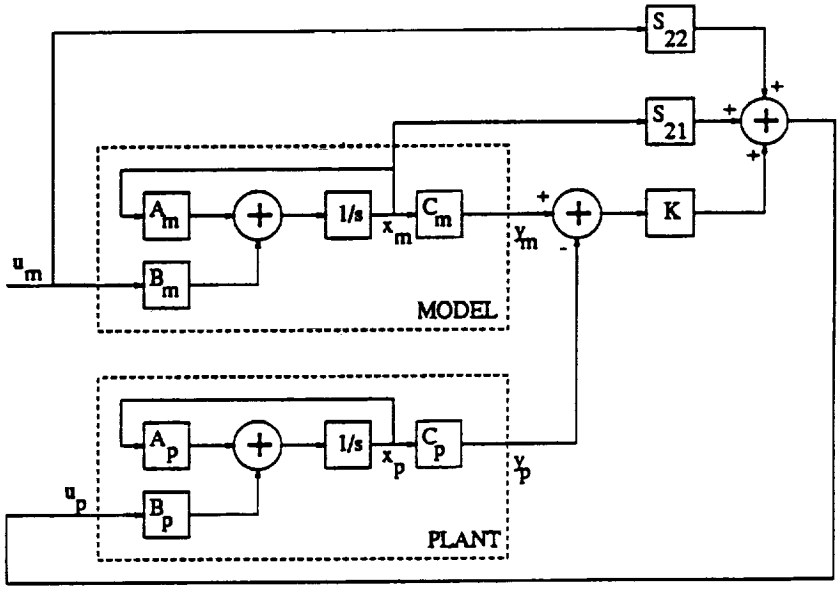

Figure 2.1: Non-Adaptive Command Generator Tracker Block Diagram

2.3 Basic Direct Model Reference Adaptive Algorithm

This section will discuss the basic DMRAC algorithm as proposed by Sobel, Kaufman, and Mabius [9].

In the previous section, the final control law (2.26) assumed that the plant parameters, A_p, B_p, and C_p, were known. If this is not the case then an adaptive version of the CGT control law is required. The adaptive control law has the same form as (2.26) and is given below,

$$u_p(t) = K_x(t)x_m(t) + K_u(t)u_m(t) + K_e(t)[y_m(t) - y_p(t)] \tag{2.27}$$

where $K_x(t)$, $K_u(t)$, and $K_e(t)$ are adaptive gains. We must now find adaptive laws for $K_x(t)$, $K_u(t)$, and $K_e(t)$ to drive the output tracking error $e_y(t) = y_m(t) - y_p(t) \rightarrow 0$ as $t \rightarrow \infty$. To create more compact equations, we will concatenate the adaptive gains into a matrix as follows:

$$K_r(t) = [K_e(t) \;\; K_x(t) \;\; K_u(t)] \tag{2.28}$$

and concatenate the output tracking error signal and the model state and input as follows:

$$r(t) = \begin{bmatrix} y_m(t) - y_p(t) \\ x_m(t) \\ u_m(t) \end{bmatrix} \tag{2.29}$$

Using the above notation simplifications, the adaptive control law (2.27) becomes,

$$u_p(t) = K_r(t)r(t) \tag{2.30}$$

From [9], the adaptive law for the gains $K_x(t)$, $K_u(t)$, and $K_e(t)$ is composed of a proportional and integral part as follows:

$$K_P(t) = e_y(t)[r(t)]^T T_{pro} \tag{2.31}$$

$$\dot{K}_I(t) = e_y(t)[r(t)]^T T_{int} \tag{2.32}$$

$$K_r(t) = K_P(t) + K_I(t) \tag{2.33}$$

where $e_y(t) = y_m(t) - y_p(t)$, T_{pro} is a constant proportional weighting matrix, T_{int} is a constant integral weighting matrix, $K_P(t)$ is the proportional part of $K_r(t)$, and $K_I(t)$ is the integral part of $K_r(t)$. Note: $K_I(t)$ is obtained by integrating $\dot{K}_I(t)$.

From [9], (2.31)-(2.33) will achieve asymptotic output tracking, $e_y \rightarrow 0$ as $t \rightarrow \infty$, if the following are true:

- T_{pro} is positive semi-definite,
- T_{int} is positive definite,
- The plant is Almost Strictly Positive Real.

The last condition, ASPR plant, means that there exists some feedback gain matrix, $\tilde{K}$, such that the fictitious stabilized plant, described by the triple $(A_p - B_p\tilde{K}C_p, B_p, C_p)$, is strictly positive real. The block diagram for the basic DMRAC algorithm is shown in Figure 2.2. This algorithm will be referred to as - $\langle BASIC \rangle$.

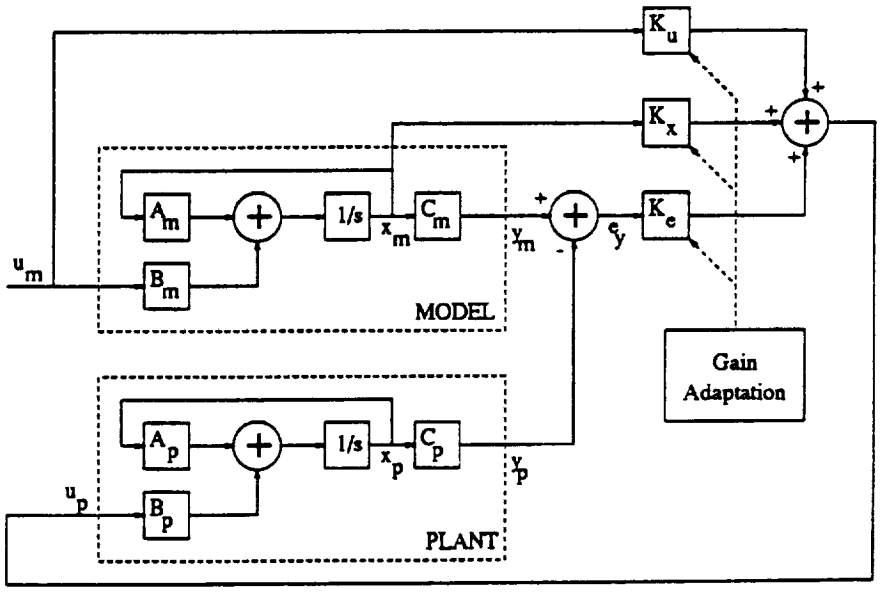

Figure 2.2: Basic Direct Model Reference Adaptive Controller Block Diagram

2.4 Modification of Basic DMRAC for Non-ASPR Plants

The development in the preceding section required the plant to satisfy an Almost Strictly Positive Real condition. For plants which are not ASPR, BarKana and Kaufman [12, 13] proposed augmenting the plant with parallel dynamics to make the augmented plant ASPR in which case the results from the previous section will hold.

The basic procedure, as discussed in [14], will now be presented. Let $G(s)$ be the transfer matrix of a continuous-time linear non-ASPR plant,

$$G(s) = C_p \left[sI - A_p\right]^{-1} B_p \tag{2.34}$$

which is not necessarily stable or minimum phase. Assume that there exists another transfer matrix, $H(s)$, such that the resulting closed-loop transfer matrix,

$$G_c(s) = \left[I + G(s)H(s)\right]^{-1} G(s) \tag{2.35}$$

is asymptotically stable and $H(s)$ is ASPR. In this case, there exists a feed-forward filter, $D(s)$, such that the augmented (open-loop) plant transfer matrix,

$$G_o(s) = G(s) + D(s) \tag{2.36}$$

is ASPR where $D(s) = H^{-1}(s)$. One widely used choice of $D(s)$ is,

$$D(s) = \frac{K_d}{1 + \tau s} \tag{2.37}$$

where τ is selected sufficiently small and K_d is a constant gain matrix. The augmented (open-loop) transfer matrix then becomes,

$$G_a(s) = D(s) + G(s) = \frac{K_d}{1 + \tau s} + G(s) \tag{2.38}$$

A block diagram of the DMRAC controller with the augmented plant is shown in Figure 2.3.

Notice that the error, e_y, in Figure 2.3 is the difference between the model output, y_m, and the augmented plant output, y_a. Thus, $y_m - y_a$ is guaranteed to go to zero, not $y_m - y_p$. Since we are interested in having the original plant output track the model, $\|K_d\|$ should be chosen to be very small. In this case, $G_a(s) \approx G(s)$ and the original plant output, y_p, will closely approximate the reference model output, y_m. This result holds if $G(s)$ is output stabilizable via high feedback gains, K. If the plant is not stabilizable by a high feedback gain, then an appreciable steady state error will occur. Although it is fairly easy to select supplemental dynamics, $D(s)$, in (2.38) to satisfy the ASPR condition, the resulting controller will in general result in a steady state error that is bounded but not equal to zero.

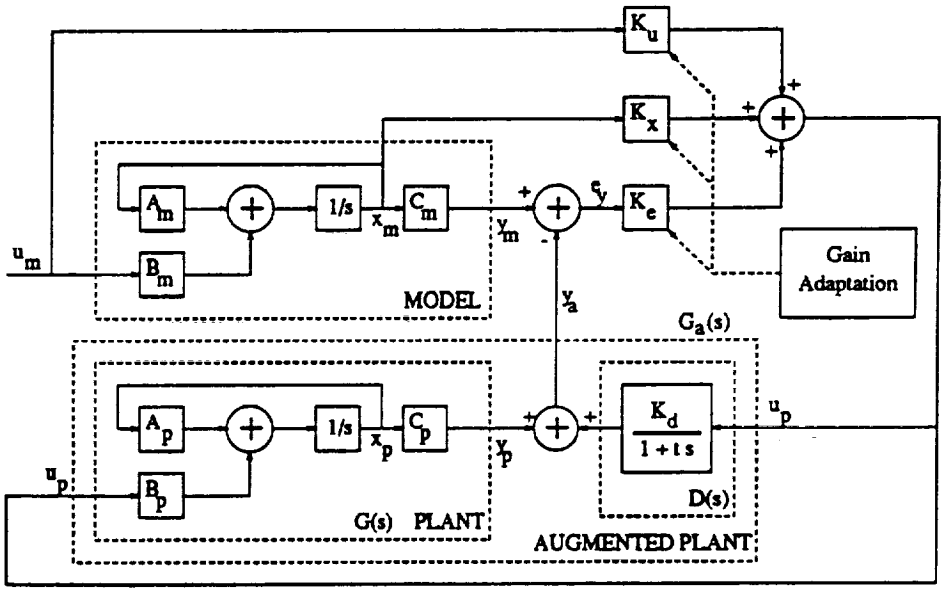

Figure 2.3: DMRAC with Augmented Plant Block Diagram

The gain adaptation is the same as in the previous section, (2.27)–(2.33). This algorithm will be referred to as – $\langle BASIC/FF \rangle$.

2.5 Modification to Insure Asymptotic Model Following

In the preceding section, we extended the Basic DMRAC algorithm to include Non-ASPR plants at the expense of an added steady state model following error. To compensate for this error, Kaufman, Neat, and Steinvorth [5] proposed incorporating the supplementary feed-forward dynamics of (2.38) into the reference model as well. This section will follow the development given in [14]. For a stability proof see [5].

Consider the original plant described by (2.1) and (2.2) and the reference model given by (2.3) and (2.4). As in the previous section, we define an augmented plant with an output of,

$$z_p(t) = y_p(t) + \bar{D}\,[u_p(t)] \tag{2.39}$$

where $\bar{D}$ denotes the operator defined by (2.37). As with the plant, we add the feed-forward dynamics to the reference model as well, by defining an augmented model output,

$$z_m(t) = y_m(t) + \bar{D}\,[u_p(t) - K_e(t)e_z(t)] \tag{2.40}$$

where $K_e(t)$ is the adaptive error gain matrix which is a function of $e_z(t)$ (to be defined next).

Now, consider the error between the augmented model output and the augmented plant output as follows,

$$e_z(t) = z_m(t) - z_p(t) \tag{2.41}$$

This error will become the new error term for the adaptive controller. Substituting (2.40) and (2.39) into (2.41) yields,

$$e_z(t) = y_m(t) - y_p(t) - \bar{D}\,[K_e(t)e_z(t)] \tag{2.42}$$

or

$$\bar{D}^{-1}\left[e_z(t)\right] + K_e(t)e_z(t) = \bar{D}^{-1}\left[e_y(t)\right] \tag{2.43}$$

Therefore, if the DMRAC controller is designed such that $e_z(t) \rightarrow 0$ as $t \rightarrow \infty$ and if $D(s)$ (2.37) is stable, then from (2.43), $e_y(t) \rightarrow 0$ as $t \rightarrow \infty$ which is the desired result.

Note that (2.42) can be written as,

$$e_z(t) = \left[I + \bar{D}K_e\right]^{-1} e_y(t) \tag{2.44}$$

which is equivalent to adding a time varying filter operating on $e_y(t)$ to form $e_z(t)$. Figure 2.4 shows the resulting block diagram using this modification (where $\bar{D}$ is given by (2.37)).

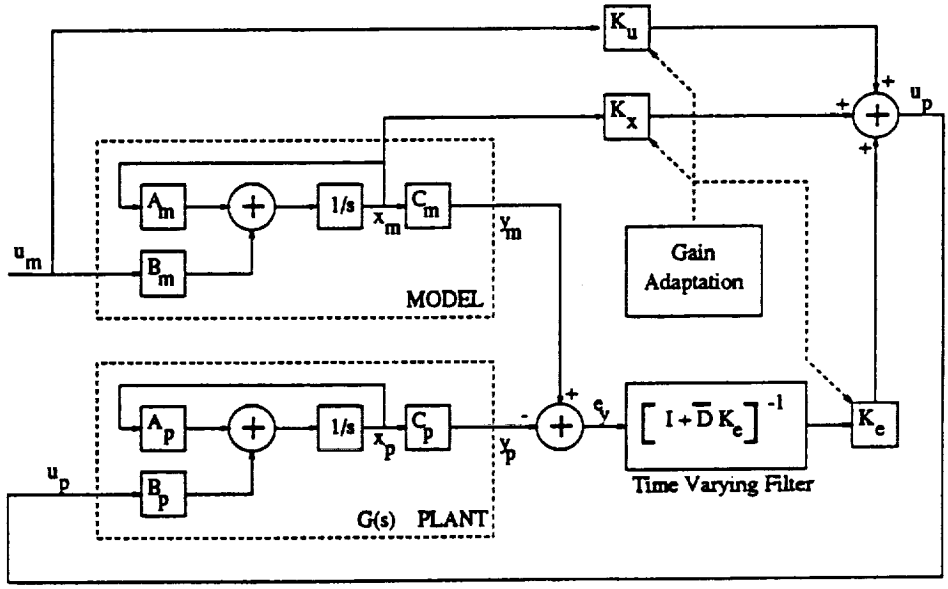

Figure 2.4: DMRAC with Supplementary Feed-Forward in Plant and Model Block Diagram

Asymptotic tracking is achieved [14] when the $e_y(t)$ terms in the gain update equations, (2.31)-(2.29), are changed to $e_z(t)$ as follows,

$$K_P(t) = e_z(t)[r(t)]^T T_{pro} \tag{2.45}$$

$$\dot{K}_I(t) = e_z(t)[r(t)]^T T_{int} \tag{2.46}$$

$$K_r(t) = K_P(t) + K_I(t) \tag{2.47}$$

where

$$r(t) = \begin{bmatrix} e_z(t) \\ x_m(t) \\ u_m(t) \end{bmatrix} \tag{2.48}$$

This algorithm will be referred to as - $\langle BASIC/FF^2 \rangle$.

2.6 Addition of Plant Output Derivative Term

One further modification to the algorithm, proposed by Steinvorth [15], was to inject a derivative term into the plant output, y_p, to form the augmented plant output,

$$y_d(t) = y_p(t) + \alpha \dot{y}_p(t) \tag{2.49}$$

or taking the Laplace Transform,

$$y_d(s) = [\alpha s + 1]\, y_p(s) \tag{2.50}$$

where α is a positive diagonal matrix of weighting constants. In this case, the above algorithms would need to be modified by replacing the original y_p with the augmented plant output y_d. This modification was added to help reduce the high frequency oscillations which generally occur in adaptive algorithms. Figure 2.5 shows $\langle BASIC/FF^2 \rangle$ with the derivative term addition. This modified algorithm will be referred to as - $\langle BASIC/FF^2/\alpha \rangle$.

From [15], the plant output can be expressed as,

$$y_p(s) = G(s)u_p(s) \tag{2.51}$$

where $G(s)$ is given by (2.34). Substituting (2.51) into (2.50) yields,

$$y_d(s) = [\alpha s + 1]\, G(s)u_p(s) \tag{2.52}$$

In steady state, the output of the augmented plant, y_d, will be the same as the original plant, y_p, since the derivative term will vanish. Note that a large term in α (2.49) will increase the model following error during transient periods.

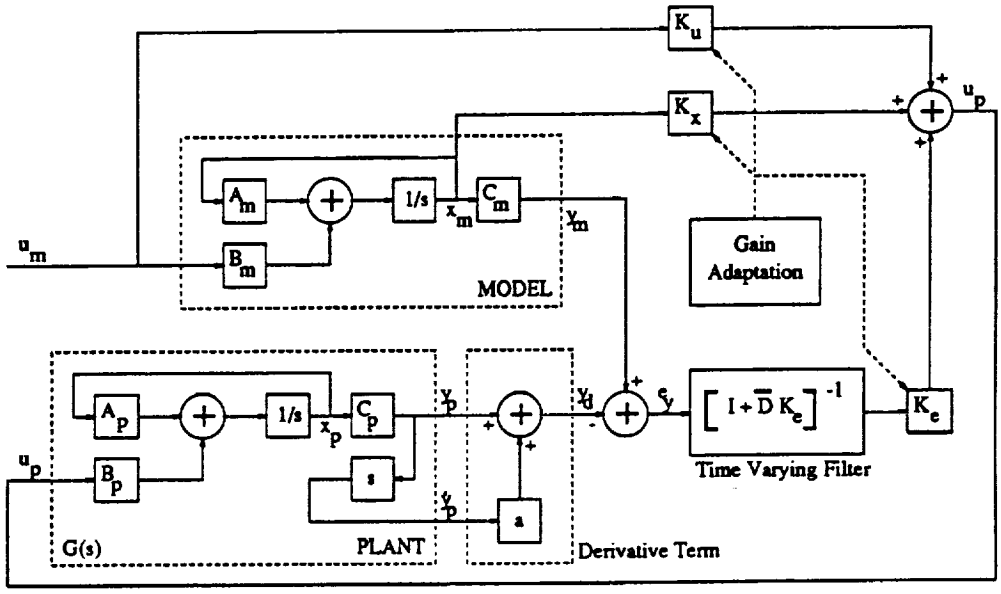

Figure 2.5: DMRAC with Added Plant Output Derivative Term Block Diagram

2.7 Addition of a Bias Term to Provide Adaptive Excitation Throughout Range of Interest

When applying the DMRAC algorithm to non-linear systems, such as the PUMA 560 Manipulator, the origin of the model coordinate system should be chosen

such that the adaptation gains have a non-zero excitation throughout the range of interest [16].

To illustrate, assume for some non-linear plant that in order to maintain an output of $y_p = [0 \ldots 0]^T$, a non-zero input, u_p, is required, and that a zero command to the reference model, $u_m = [0 \ldots 0]^T$, will result in a zero model output and state vector. Now, assume it is desired to drive the plant to this zero position. If u_m is set to zero, using $\langle BASIC \rangle$, the reference model state and output vectors will go to zero. Assuming that the plant was servoed to zero, then $e_y = y_m - y_p$ will also be zero. The vector, $r(t)$, defined by (2.29) will be zero which will result in a control, from (2.30), of $u_p = [0 \ldots 0]^T$. Since the plant requires a non-zero control to maintain a zero output, the DMRAC algorithm will require a small error signal in order to apply a non-zero control which will result in a steady-state error at the zero output position. This result holds for the augmented DMRAC algorithm $\langle BASIC/FF^2 \rangle$ as well.

If we shift the reference model coordinates by a constant bias term, then a zero command to the reference model, $u_m = [0 \ldots 0]^T$ will produce non-zero outputs for the model state and output vectors which, in turn, will produce a non-zero command to the plant by (2.30). The bias term is subtracted from the model command, u_m, and the plant output, y_p, as follows,

$$u_m(t) = \hat{u}_m(t) - q_{bias} \tag{2.53}$$

$$y_p(t) = \hat{y}_p(t) - q_{bias} \tag{2.54}$$

where $\hat{u}_m(t)$ is the original model command in the original coordinate system, $u_m(t)$ is the new biased model command to be applied to the model dynamics, $\hat{y}_p(t)$ is the actual plant output, $y_p(t)$ is the new biased plant output to be used to form the error signal, and q_{bias} is a constant bias term. For robotic manipulators, q_{bias} has units of radians and should be selected such that a new plant output of $y_p = [0 \ldots 0]$

corresponds to an equilibrium position (i.e. no gravity loading). Figure 2.6 shows the DMRAC algorithm with the bias terms added. This algorithm will be referred to as - $\langle BASIC/FF^2/\alpha/bias \rangle$.

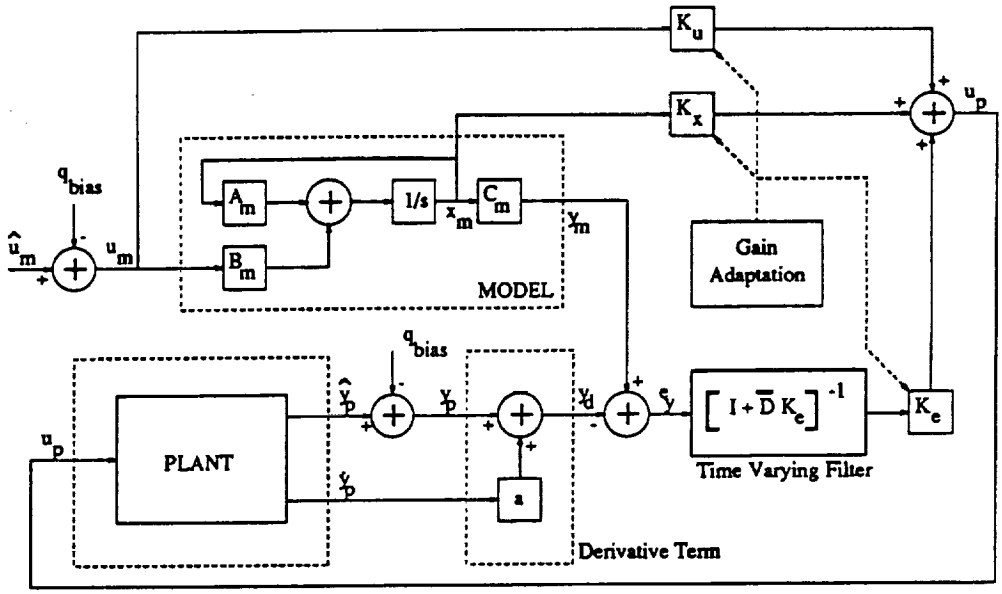

Figure 2.6: Addition of Bias Term to DMRAC Algorithm

2.8 Discretization of DMRAC Control Law for Implementation

In order to implement the DMRAC controller on the CIRSSE Testbed, the continuous time equations must be converted to discrete time so they can be coded into the CIRSSE Testbed Motion Control System which only allows for discrete control of the robotic manipulators.

To discretize the algorithm, the following continuous time dynamics were converted to discrete time:

- Reference model dynamics,
- feed-forward dynamics,
- integral adaptation dynamics, (2.32).

The discretization of the above dynamics will be discussed below. This discretized algorithm will be referred to as - $\langle BASIC/FF^2/\alpha/bias/disc \rangle$.

2.8.1 Reference Model Dynamics

The reference model, being a linear time invariant continuous system, is easily converted to discrete time using a Zero Order Hold [17] as described below.

Consider a continuous time system given by the following state space model,

$$\dot{q}_c(t) = A_c q_c(t) + B_c u(t) \tag{2.55}$$

$$y_c(t) = C_c q_c(t) + D_c u(t) \tag{2.56}$$

If we define the following constant matrices,

$$A_d = e^{A_c T} \tag{2.57}$$

$$B_d = \int_0^T e^{A_c \lambda} B_c d\lambda \tag{2.58}$$

$$C_d = C_c \tag{2.59}$$

$$D_d = D_c \tag{2.60}$$

where T is the desired sample time, then (2.55) and (2.56) can be expressed in discrete time as,

$$q_d^{(kT+T)} = A_d q_d^{(kT)} + B_d u^{(kT)} \tag{2.61}$$

$$y_d^{(kT)} = C_d q_d^{(kT)} + D_d u^{(kT)} \tag{2.62}$$

If $u(t)$ is held constant over the T-second intervals $kT \leq t < kT + T$; that is,

$$u(t) = u^{(kT)}, kT \leq t < kT + T \tag{2.63}$$

then the following will hold,

$$q_c(t) = q_d^{(kT)} \tag{2.64}$$

$$y_c(t) = y_d^{(kT)} \tag{2.65}$$

which is the desired result.

The function *c2d* in Matlab was used to perform this conversion once A_c, B_c, and T are known [18].

2.8.2 Feed-Forward Dynamics

The feed-forward dynamics, as given by (2.44), constitutes a time varying filter which does not have an easily derived closed form discrete counterpart. By rearranging the feed-forward dynamics, we can achieve an exact discretization much easier.

Substituting (2.47), (2.48), and (2.30) into (2.40) and (2.39) yields,

$$z_m(t) = y_m(t) + \bar{D}\left[K_x(t)x_m(t) + K_u(t)u_m(t)\right] \tag{2.66}$$

$$z_p(t) = y_p(t) + \bar{D}\left[K_x(t)x_m(t) + K_u(t)u_m(t) + K_e(t)e_z(t)\right] \tag{2.67}$$

Recall that the augmented error is defined as $e_z(t) = z_m(t) - z_p(t)$. Using (2.66) and (2.67), the DMRAC algorithm block diagram can be rearranged as shown in Figure 2.7 where $D(s)$ represents the $\bar{D}$ operator and is given by (2.37).

This modification results in splitting the single time-varying filter, (2.44), into two linear time invariant dynamic feed-forward blocks, $D(s)$. These two blocks can be represented in state space form and converted to discrete time using a Zero Order Hold as was done for the reference model in the preceding section.

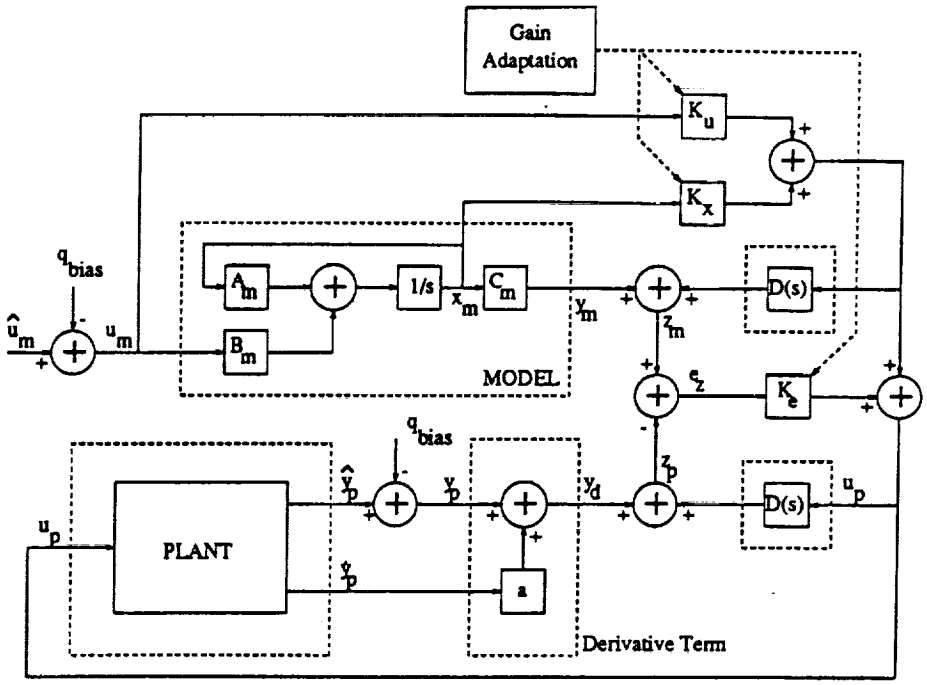

Figure 2.7: Rearranged DMRAC Algorithm Block Diagram

2.8.3 Integral Adaptation Dynamics

The adaptation laws require the integration of $\dot{K}_I(t)$, see (2.46). This integration was discretized using Backwards Rectangular Approximation[1] [19] which results in the following discrete *approximation* of the continuous adaptation equations,

$$K_P^{(kT)} = e_y^{(kT)}[r^{(kT)}]^T T_{pro} \tag{2.68}$$

$$K_I^{(kT+T)} = K_I^{(kT)} + T_s e_y^{(kT)}[r^{(kT)}]^T T_{int} \tag{2.69}$$

$$K_r^{(kT)} = K_P^{(kT)} + K_I^{(kT)} \tag{2.70}$$

where T_s is the sample time. The gains K_P, K_I, and K_r are updated every T_s seconds.

[1]The integration was also tried using Trapezoidal Approximation but there was no significant difference, thus the more efficient Backwards Rectangular Approximation was used.

2.9 Summary

In this chapter we introduced the end goal of the continuous DMRAC algorithm and discussed the Command Generator Tracker algorithm of Broussard and O'Brien, $\langle CGT \rangle$, which the DMRAC is based on. We then presented the basic DMRAC algorithm, $\langle BASIC \rangle$, as was proposed by Sobel, Kaufman, and Mabius. Next, we discussed two modifications to the basic algorithm. The first modification was the augmentation of the plant to support Non-ASPR plants as proposed by BarKana and Kaufman - $\langle BASIC/FF \rangle$. The second was the inclusion of the augmented dynamics in the model to achieve asymptotic tracking as proposed by Kaufman, Neat, and Steinvorth - $\langle BASIC/FF^2 \rangle$. We then discussed the addition of a plant output derivative term as proposed by Steinvorth and Kaufman, $\langle BASIC/FF^2/\alpha \rangle$, and the addition of a bias term as proposed by Cummings, Swift, and Kaufman, $\langle BASIC/FF^2/\alpha/bias \rangle$. Finally, we discussed the discretization of the algorithm for implementation on the CIRSSE Testbed, $\langle BASIC/FF^2/\alpha/bias/disc \rangle$.

CHAPTER 3
Simulation Environment

In order to test the performance of the DMRAC algorithm, a realistic simulation environment was needed. The Matlab program from *The Mathworks, Inc.* [20] was chosen as the "base" for the simulations. Matlab is a high-performance interactive software package for engineering numerical computation. Matlab integrates numerical analysis, matrix computations, and graphics in an easy-to-use environment. The DMRAC algorithm was written in "Matlab Code" as an "M" file [20] since modifications could be made easily without the need to compile any code. The computation intensive routines (integration and model dynamics simulation) were coded in C and linked in with the Matlab program using the Matlab supplied *CMEX* utility [20].

The simulation was composed of the following five modules of code:

- *Simulation Administrator* was responsible for coordinating the simulation and transferring data between the various modules.
- *Joint Control Algorithm Module* was used to compute the DMRAC control law to be applied to the robot.
- *PUMA 560 Dynamics Module* modeled the dynamics of the robot.
- *ODE Integration Routine Module* was used to integrate the state vector returned by the Dynamics Module.
- *Trajectory Generator Module* was called by the DMRAC Algorithm to compute the trajectory for the robot to follow.

3.1 Simulation Administrator

The simulation administrator coordinats the simulation and is responsible for transferring data between the other various modules. The flow diagram is shown in Figure 3.1 and will be described below.

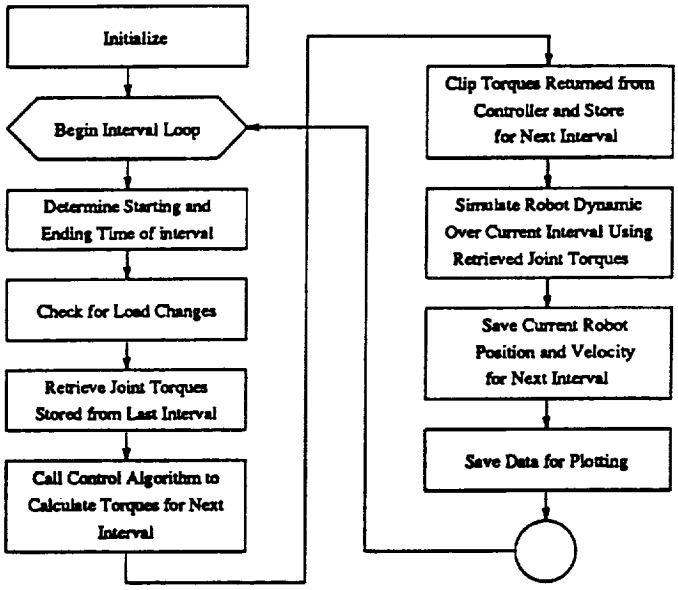

Figure 3.1: Simulation Administrator

The first task is to initialize the PUMA model, controller, and trajectory generator modules along with some initialization of local variables used by the administrator. Next, the control interval loop is begun. Each pass through this loop constitutes a new update of the control torques applied to the robot. The sample time used in the simulations, as well as on the actual hardware, is 4.5 ms.

At the beginning of each interval cycle, the administrator determines the starting and ending times, in seconds, of the interval loop, t_s and t_f respectively. Note: $t_f - t_s = 4.5\ ms$. Next, the administrator checks for any load changes. If a load change is desired, the model parameters are changed for Link 6 to reflect the addition or subtraction of the load. Note: A load change can only occur at the start of an interval. Next, the joint torques calculated in the last interval are retrieved

Table 3.1: Maximum Joint Torques for PUMA 560 Manipulator

Joint	Maximum Torque in (Nm)
1	97.6
2	186.4
3	89.4
4	24.2
5	20.1
6	21.3

for use during the integration of the model state equations. For the first interval, the retrieved torque values are set to $\{0,0,0,0,0,0\}$. The retrieved values will be referred to as Γ_{ret}.

The joint control algorithm is now called and is passed the position and velocity of the joints at the *start* of the interval. The control algorithm returns a vector of joint torques. These returned joint torques are then clipped at the maximum torque values for the joints and stored for use in the next interval. The torques used in the simulation are *joint* torques, not *motor* torques. From [21], the maximum link torque values for the PUMA 560 Manipulator are shown in Table 3.1. The clipping of the joint torques allows for an accurate simulation of amplifier saturation in the motor drivers which could happen on the hardware in the Testbed.

The robot dynamics are then simulated over the current interval by integrating the robot state equation from $t = t_s$ to $t = t_f$ using the initial conditions (joint position and velocity) saved in the previous interval and the retrieved torque values Γ_{ret}. Note: The torque values are held constant throughout the interval (Zero Order Hold) which is customary for discrete control. The position and velocity of the robot joints at the end of the interval are saved. The saved values are used by the control algorithm and also by the integration routine. Finally, the administrator collects any desired data to be plotted and stores it away in an array.

3.2 Joint Control Algorithm

The joint control algorithm is called at each interval to calculate a 6×1 vector of joint torques to be applied to the robot joints at the start of the *next* interval. The control algorithm is passed the position and velocity of the robot at the start of the interval only. This section will describe the implementation of the discretized DMRAC algorithm, $\langle BASIC/FF^2/\alpha/bias/disc \rangle$, used in the simulation.

3.2.1 Reference Model

The choice of the reference model order is a compromise between high gains and excessive response delays [16]. If the reference model order is too low, then excessively large gains may occur which may lead to control saturation in the command to the plant. On the other hand, if the reference model order is too high, then excessive response delays may be produced.

For the control of the PUMA 560 Manipulator, six decentralized linear models, each with an order of two, were chosen yielding a total reference model order of 12. The independent second order models were chosen [16] because in a PUMA 560, the mass matrix is approximately diagonal for all joint values making the system almost decoupled. Thus, the second order model should be a good approximation for each joint leaving the coriolis, centrifugal, and gravity terms to be adapted to by the DMRAC algorithm.

The selected reference model transfer function for each joint is given by,

$$G_{m_i}(s) = y_{m_i}(s)/u_{m_i}(s) = \frac{\omega_{n_i}^2}{s^2 + 2\zeta_i\omega_{n_i}s + \omega_{n_i}^2} \tag{3.1}$$

where i is the joint number $\{1, \ldots, 6\}$, ω_{n_i} is the natural undamped frequency, and ζ_i is the damping ratio. Equation (3.1) can be expressed in state space form as,

$$\dot{x}_{m_i}(t) = \begin{bmatrix} 0 & 1 \\ -\omega_{n_i}^2 & -2\zeta_i\omega_{n_i} \end{bmatrix} x_{m_i}(t) + \begin{bmatrix} 0 \\ \omega_{n_i}^2 \end{bmatrix} u_{m_i}(t) \tag{3.2}$$

$$y_{m_i}(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_{m_i}(t) \tag{3.3}$$

where $\dot{x}_{m_i}(t)$ is a 2×1 state vector. After selection of the ω_{n_i} and ζ_i values, (3.2) and (3.3) were discretized as discussed in Section 2.8.

3.2.2 Feed-Forward Filter

The feed-forward filter dynamics for each joint are given by (2.37) as,

$$D(s) = \frac{K_d}{1 + \tau s} \tag{3.4}$$

which has the following state space representation,

$$\dot{x}_{f_i}(t) = [-1/\tau]\, x_{f_i}(t) + [K_d/\tau]\, u_{f_i}(t) \tag{3.5}$$

$$y_{f_i}(t) = [1]\, x_{f_i}(t) \tag{3.6}$$

where K_d is the DC gain, τ is the time constant, $x_{f_i}(t)$ is the filter state variable, and i is the joint number $\{1, \ldots, 6\}$. As with the model equations above, (3.5) and (3.6) were converted to discrete form.

3.2.3 Bias Term

The bias term, as discussed in Section 2.7, was included to shift the reference model coordinates. By examining the zero position of the robot, Figure 3.3, it is clear that $y_p = \{0, 0, 0, 0, 0, 0\}$ is not an equilibrium. A bias of,

$$q_{bias} = \{0, \frac{\pi}{2}, \frac{\pi}{2}, 0, 0, 0\} \tag{3.7}$$

will shift the zero position to that shown in Figure 3.2.

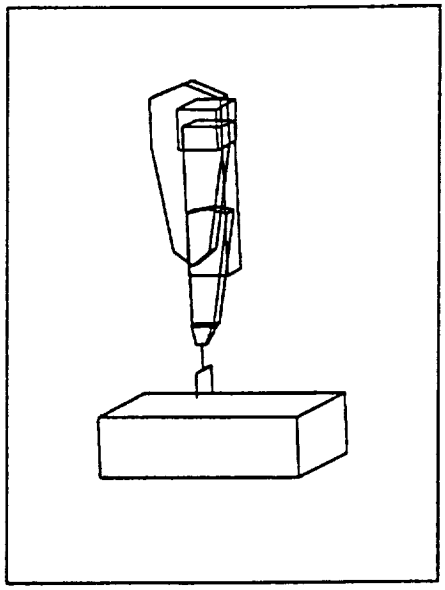

Figure 3.2: Stable Equalibrium for the PUMA 560

3.3 PUMA 560 Manipulator Dynamic Model

In order to test the performance of the DMRAC algorithm, an accurate non-linear coupled model of the manipulator was needed. A full explicit dynamic model of the PUMA 560 Manipulator, derived by Armstrong, Khatib, and Burdick [21] was selected. The formulation of the PUMA model was computationally efficient using 25% fewer calculations than a six degree of freedom recursive Newton-Euler method (RNE). The algebraic formulation of the model also allowed for the easy addition of a load by modifying the mass, center of mass, and inertia parameters for Link 6 as described in [22].

3.3.1 Coordinate Frame Assignments

The chosen coordinate system for the PUMA 560 Manipulator is identical to that used in [23] except for the labeling convention[1]. Figure 3.3 shows the six rotational joint axis, $\{z_1, \ldots, z_6\}$, for the PUMA 560. Only the rotational, z_i, axis

[1] [23] defines labels for all 18 Testbed joints. Since this project dealt with only the six joints of the PUMA, the coordinate labeling of [21] will be used.

are shown in the figure. Positive rotations follow the right hand rule – counter-clockwise looking down the z axis. The six joints of the PUMA 560 are as follows:

- *Joint 1.* A vertical rotation about the base, z_1.
- *Joint 2.* A horizontal rotation about the shoulder, z_2.
- *Joint 3.* A horizontal rotation about the elbow, z_3.
- *Joint 4.* A twist of the wrist, z_4.
- *Joint 5.* An inclination of the wrist, z_5.
- *Joint 6.* A twist of the mounting flange, z_6.

The position of the manipulator in Figure 3.3 illustrates the zero position. Note: When Joint 5 is at zero, axis z_4 and z_6 coincide.

3.3.2 Derivation of Dynamic Equations

From [24], the dynamic equations used to model the PUMA 560 Manipulator are:

$$A(\theta)\ddot{\theta} + B(\theta)[\dot{\theta}\dot{\theta}] + C(\theta)[\dot{\theta}^2] + g(\theta) + b(\dot{\theta}) = \Gamma \tag{3.8}$$

where

$A(\theta)$ is the 6×6 kinetic energy matrix,
$B(\theta)$ is the 6×15 matrix of coriolis torques,
$C(\theta)$ is the 6×6 matrix of centrifugal torques,
$g(\theta)$ is the 6 vector of gravity torques,
$\ddot{\theta}$ is the 6 vector of joint accelerations,
$[\dot{\theta}\dot{\theta}]$ is the 15 vector of velocity products,

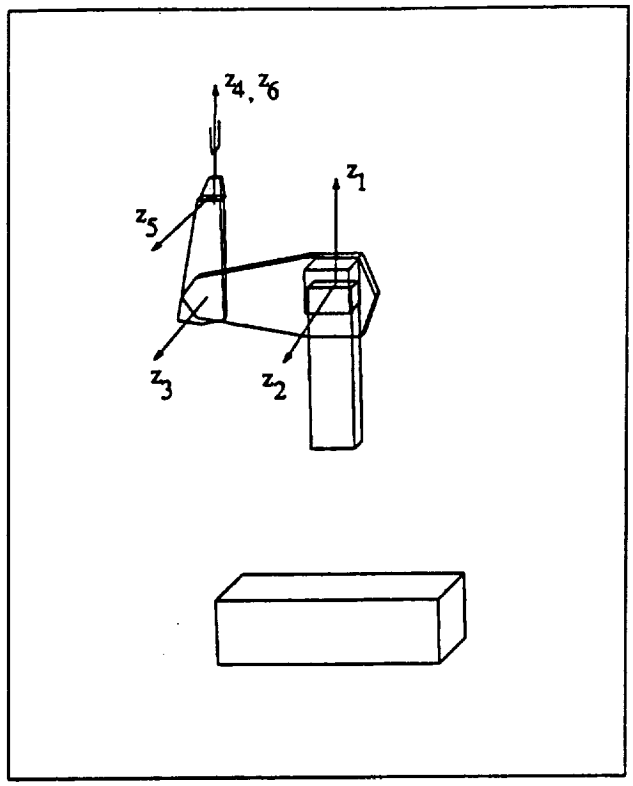

Figure 3.3: PUMA 560 Coordinate Frame Assignments

$[\dot{\theta}^2]$ is the 6 vector of squared velocities,

$b(\dot{\theta})$ is the 6 vector of friction torques,

and Γ is the 6 vector of joint torques.

Note:

$$[\dot{\theta}\dot{\theta}] = [\dot{\theta}_1\dot{\theta}_2, \dot{\theta}_1\dot{\theta}_3, \ldots, \dot{\theta}_1\dot{\theta}_6, \dot{\theta}_2\dot{\theta}_3, \ldots, \dot{\theta}_4\dot{\theta}_6, \dot{\theta}_5\dot{\theta}_6]^T$$
$$[\dot{\theta}^2] = [\dot{\theta}_1^2, \dot{\theta}_2^2, \ldots, \dot{\theta}_6^2]^T$$

The equations for $A(\theta)$, $B(\theta)$, $C(\theta)$, and $g(\theta)$ were compiled from [21] and are described in detail in [24]. They will not be presented here because of space limitations (see appendix?). The dynamic and kinematic parameters for the PUMA 560 Manipulator were compiled from [21] and [25] and are also described in [24]. Tables 3.2 and 3.3 show the manipulator parameters as listed in [24]. The motor inertias listed

in Table 3.3 have been reflected to the link side by multiplying them by the square of the gear ratio, n. The inertias in Table 3.3 are about the center of mass of the respective link except where noted.

Table 3.2: Masses and Centers of Gravity of Puma Arm Links

	mass (kg)	r_x (m)	r_y (m)	r_z (m)
Link 1	12.95	0.0	0.0389	-0.3088
Link 2	17.40	0.068	0.006	-0.016
Link 3	4.80	0.0	-0.070	0.014
Link 4	0.82	0.0	0.0	-0.019
Link 5	0.34	0.0	0.0	0.0
Link 6	0.09	0.0	0.0	0.032

The friction vector, $b(\dot{\theta})$ in (3.8), was arbitrarily set to,

$$b(\dot{\theta}) = K_b\dot{\theta} \tag{3.9}$$

where $K_b = diag(5,5,5,10,10,10)$, to provide some viscous[2] friction to the model. The units of K_b are $N\,m\,sec/rad$. No friction identification was performed on the PUMA 560 Manipulator.

[2]Initially a Viscous-Coulomb-Stiction friction model was used which resulted in a very stiff set of equations for the model. This slowed the model integration down and, as a result, the simulation was slowed down by a factor of about 10. The Viscous-Coulomb-Stiction model was abandoned due to this delay.

Table 3.3: Diagonal Inertia Terms and Reflected Motor Inertias

	I_{xx} (kg-m^2)	I_{yy} (kg-m^2)	I_{zz} (kg-m^2)	$n^2 I_{\text{motor}}$ (kg-m^2)	n Gear Ratio
Link 1	2.35*	2.34*	0.197*	1.14	62.61
Link 2	0.130	0.524	0.539	4.71	107.36
Link 3	0.066	0.0125	0.086	0.827	53.69
Link 4	$1.80\text{x}10^{-3}$	$1.80\text{x}10^{-3}$	$1.30\text{x}10^{-3}$	0.2	76.01
Link 5	$0.30\text{x}10^{-3}$	$0.30\text{x}10^{-3}$	$0.40\text{x}10^{-3}$	0.179	71.91
Link 6	$0.15\text{x}10^{-3}$	$0.15\text{x}10^{-3}$	$0.04\text{x}10^{-3}$	0.193	76.73

* = About the Coordinate Frame

The above model can be cast into state space form by solving (3.8) for $\ddot{\theta}$ as follows:

$$\ddot{\theta} = A^{-1}(\theta)\left[\Gamma - B(\theta)[\dot{\theta}\dot{\theta}] - C(\theta)[\dot{\theta}^2] - g(\theta) - b(\dot{\theta})\right] \tag{3.10}$$

The kinetic energy matrix, $A(\theta)$, is positive-definite [1] and therefore non-singular; thus, the inverse exists. Now, by choosing the following 12×1 state vector,

$$x = \begin{bmatrix} \theta \\ v \end{bmatrix} \tag{3.11}$$

where $\theta = [\theta_1, \ldots, \theta_6]^T$ and $v = [\dot{\theta}_1, \ldots, \dot{\theta}_6]^T$, (3.10) can be written,

$$\dot{\theta} = v \tag{3.12}$$

$$\dot{v} = A^{-1}(\theta)\left[\Gamma - B(\theta)[\dot{\theta}\dot{\theta}] - C(\theta)[\dot{\theta}^2] - g(\theta) - b(\dot{\theta})\right] \tag{3.13}$$

The robot dynamics can now be simulated by integrating (3.12) and (3.13) over the period of interest with appropriate initial conditions (joint position and velocity) and with Γ set to the constant torque values calculated by the control algorithm.

3.3.3 End-Effector Parameters

The PUMA 560 Manipulator in the CIRSSE Testbed includes a Force Torque Sensor (FTS) and a pneumatic gripper which are attached to the last link of the PUMA. The combined weight of the FTS and gripper is about 3.4 lbs which was significant enough to affect the accuracy of the model. The model developed in the preceding section did not include the dynamic parameters of this end-effector. In order to achieve accurate modeling of the actual Testbed arm, the dynamic parameters of the end-effector were measured, [22], and included in the model by modifying the mass, inertia, and center of mass parameters of Link 6 as in [22]. The gripper load parameters are given in Table 3.4.

Item	Units	Value
Mass	kg	1.548
Distance to Center of Mass along x_6	m	0.0
Distance to Center of Mass along y_6	m	0.0
Distance to Center of Mass along z_6	m	0.1357
Moment of Inertia about x_6	$kg\,m^2$	$33.2 * 10^{-3}$
Moment of Inertia about y_6	$kg\,m^2$	$33.0 * 10^{-3}$
Moment of Inertia about z_6	$kg\,m^2$	$1.18 * 10^{-3}$

Table 3.4: End-Effector Parameters

3.3.4 Verification of Model

The model was verified by comparing it to an existing recursive Newton-Euler (RNE) formulated model of the PUMA 560. The kinetic energy, coriolis, centrifugal, and gravity matrices were extracted from the RNE model for various joint positions with the use of a RNE *inverse* dynamics routine. By selecting the joint velocities, joint accelerations, robot base velocities and accelerations, and tip forces, it was possible to make individual components of the kinetic, coriolis, centrifugal, and gravity matrices show up in the joint torque vector returned from the RNE inverse dynamics routine. These torque vectors could then be used to reconstruct the dynamic matrices.

The extracted matrices calculated by the RNE model were then compared to the kinetic, coriolis, centrifugal, and gravity matrices generated by the explicit model of Armstrong. Both models agreed to within accountable numerical round off errors. The same kinematic, mass, and inertia parameters[3] were used for both models (Tables 3.2 and 3.3).

As another test of the mass and center of mass parameters, the gravity torque vector, $g(\theta)$ in (3.8), was used to compensate for the gravity loading on the actual PUMA 560 arm in the CIRSSE Testbed [26]. When the open-loop gravity control was applied to the arm, the links could be freely moved throughout their entire joint

[3]The gripper parameters were not used for the comparison

space and would hold any position when released without falling due to gravity. Thus, it can be assumed that the manipulator mass and center of mass parameters were quite accurate.

3.3.5 Robot Model Implementation

The model was coded in C and interfaced to Matlab using the CMEX utility supplied with Matlab which allows for linking of C code directly into the Matlab environment [20]. By implementing the model in C rather than as a standard Matlab "M" file, a reduction in the model computation time by a factor of 4 to 5 was achieved. Fast simulations greatly reduced the amount of time needed to tune the adaptive controller. Note: None of the PUMA 560 model information was used in the DMRAC algorithm. The dynamic model was created for simulation purposes only.

3.4 Integration Routine

The integration routine used to integrate the robot dynamic equations was obtained from Sandia National Labs [27] and was also interfaced into Matlab using the CMEX utility. The FORTRAN implementation of the integrator allowed for fast, accurate integration of ordinary differential equations. The algorithm is very robust and is described in detail in [28]. A brief description of the routine, referfed to as *newodeif*, follows[4].

Newodeif integrates a system of n first order ordinary differential equations of the form,

$$\frac{\delta y_i(t)}{\delta t} = f_i(t, y_1(t), y_2(t), \ldots, y_n(t)) \tag{3.14}$$

[4]This information was extracted from the programming comments in the code as no manual for the integrator existed.

where $i = 1, 2, \ldots, n$. The solution is returned at evenly spaced discrete moments in time, called mesh points, which can be selected by the user. In our case, *newodeif* is used to integrate from the start of the control interval to the end of the control interval. The solution is only returned for the end of the interval.

Newodeif is composed of the following three routines:

1. *DE* is a supervisor which directs the solution.
2. *STEP1* advances the solution one sample step.
3. *INTRP* interpolates at the output points.

The routine *DE* controls the integration and calls *STEP1* as needed to integrate between the mesh points. In our case, the step size is set to the sample period since we are not interested in values between the sample intervals. The routine *STEP1* performs the actual integration using a modified divided difference form of the Adams Pece Formulas [28]. To improve absolute stability and accuracy, *STEP1* uses local extrapolation. The order and step size of the integration is automatically adjusted to control the local error. Special devices are also included to control roundoff error. To improve accuracy near the mesh points, *STEP1* approximates the solution by a polynomial and calls *INTRP* to approximate the solution by evaluating the polynomial at the mesh points. To improve accuracy at the last endpoint, *STEP1* integrates past the point and interpolates the solution using *INTRP*.

3.5 Trajectory Generator

The trajectory generator (TGEN) module is responsible for planning the joint space trajectories for the manipulator. These trajectories are passed directly to the reference model. The TGEN is called every interval by the joint control module and must return the desired position, velocity, and acceleration setpoints which the controller tries to servo the arm to meet. The trajectories produced by the TGEN

are minimum jerk, meaning that the derivative of acceleration (jerk) is minimized producing very smooth motions. The equations used to create the minimum jerk trajectories are from [29] and will be outlined below.

The TGEN produces a minimum jerk trajectory based on a set of supplied knot points. Each knot point indicates a point in the joint space of the manipulator which should be "visited" by the robot. The TGEN will produce minimum jerk trajectory *segments* between each pair of adjacent knot points such that the manipulator stops at each knot point in the order in which they are specified. Associated with each knot point, except the first, is a time value which specifies the amount of transit time, in seconds, between the current and previous knot points. The first knot point is implicitly set to the initial "shut-down" position of the robot $\{0, -45, 180, 0, 45, 90\}$ degrees, see Figure 3.4) and can not be changed by the user. To wait at a knot point position, the knot point can simply be repeated in the list.

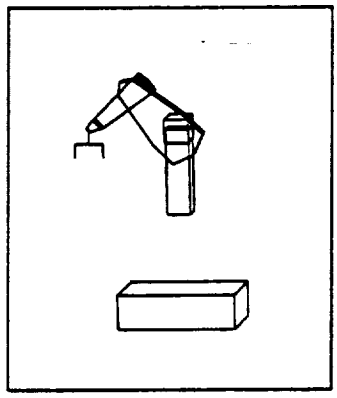

Figure 3.4: Shutdown Position, $\{0, -45, 180, 0, 45, 90\}$ degrees

Between each pair of adjacent knot points there is a trajectory segment. Each trajectory segment is described by the following joint position, velocity, and acceleration equations:

$$\theta_i(t) = \begin{cases} q_i^{[j-1]} & (t \le 0) \\ q_i^{[j-1]} + \frac{\alpha t^3}{6} & (0 < t < \frac{T}{4}) \\ q_i^{[j-1]} + \frac{\alpha t^3}{6} + \frac{\alpha T t^2}{4} - \frac{\alpha T^2 t}{16} + \frac{\alpha T^3}{192} & (\frac{T}{4} \le t < \frac{3T}{4}) \\ q_i^{[j-1]} + \frac{\alpha t^3}{6} + \frac{\alpha T t^2}{2} - \frac{\alpha T^2 t}{2} + \frac{13\alpha T^3}{96} & (\frac{3T}{4} \le t < T) \\ q_i^{[j]} & (t > T) \end{cases} \tag{3.15}$$

$$\dot{\theta}_i(t) = \begin{cases} 0 & (t \le 0) \\ \frac{\alpha t^2}{2} & (0 < t < \frac{T}{4}) \\ -\frac{\alpha t^2}{2} + \frac{\alpha T t}{2} - \frac{\alpha T^2}{16} & (\frac{T}{4} \le t < \frac{3T}{4}) \\ \frac{\alpha t^2}{2} - \alpha T t + \frac{\alpha T^2}{2} & (\frac{3T}{4} \le t < T) \\ 0 & (t > T) \end{cases} \tag{3.16}$$

$$\bar{\theta}_i(t) = \begin{cases} 0 & (t \le 0) \\ \alpha t & (0 < t < \frac{T}{4}) \\ -\alpha t + \frac{\alpha T}{2} & (\frac{T}{4} \le t < \frac{3T}{4}) \\ \alpha t - \alpha T & (\frac{3T}{4} \le t < T) \\ 0 & (t > T) \end{cases} \tag{3.17}$$

where

i is the joint number $\{1, \ldots, 6\}$,

j is the index of the current knot point $\{0, 1, 2, \ldots\}$ (0 indicates the implicit knot point),

$q_i^{[j]}$ is the current knot point which is being moved towards,

$q_i^{[j-1]}$ is the previous knot point,

$\alpha = 32(q_i^{[j]} - q_i^{[j-1]})/(T^3)$,

T is the desired time for completing the trajectory segment,

t is the relative time on the trajectory segment ($t = 0$ indicates position $q_i^{[j]}$ and $t = T$ indicates position $q_i^{[j-1]}$),

and $\theta_i(t)$, $\dot{\theta}_i(t)$, and $\ddot{\theta}_i(t)$ are the desired position, velocity, and acceleration of joint i at relative time t.

The joint position, velocity, and acceleration functions for an example minimum jerk path are shown in Figure 3.5. For this single joint example, the starting and ending knot point values were 0 degrees and 90 degrees respectively and the time value was 4 seconds.

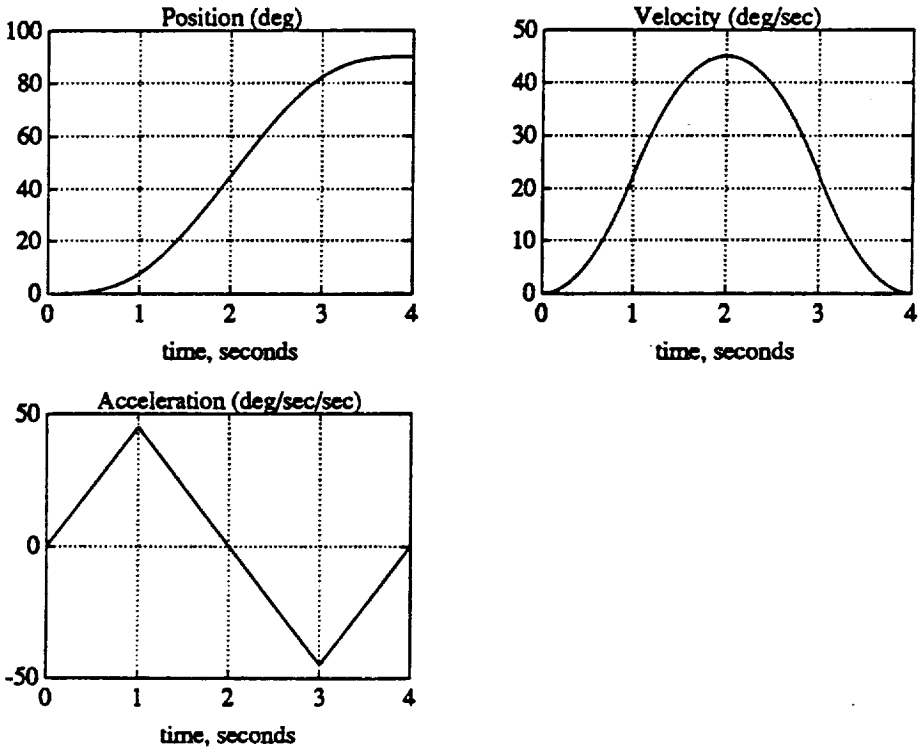

Figure 3.5: An Example Minimum Jerk Path

The minimum jerk joint position command (3.15) was passed to the reference model input u_m. The velocity and acceleration equations were not used.

3.6 Summary

In this chapter we discussed the five modules comprising the simulation environment. The first module, the *simulation administrator*, is responsible for coordinating the simulation and moving data around. The second module, the *joint control algorithm*, computes the control law to be applied to the robot manipulator at each interval. Details of the reference model, feed-forward filter, and bias term selection were discussed. Next, the *PUMA 560 dynamic model* module was developed. It is important to note that the robot model is used only for simulation. No model information is used in the DMRAC algorithm. The method used by the *ODE integration module* to integrate the robot model was then discussed. The final module, the *trajectory generator*, is used to generate the desired joint motions which are passed on to the *joint control algorithm*.

CHAPTER 4

Simulation Results (Tuning and Joint Evaluation Cases)

This chapter will present the results of the Matlab simulations of Direct Model Reference Adaptive Control of a six link PUMA 560 Manipulator, fully-centralized. First the issue and method of tuning a DMRAC algorithm will be discussed. Next, the tracking performance of the PUMA 560 under DMRA control will be tested on a joint by joint basis. All results will be displayed with the bias term, q_{bias} removed (Section 2.7).

4.1 Tuning

This section will describe the process used to tune DMRAC algorithms in general. Specific tuning for the control of the PUMA 560 Manipulator will be illustrated.

4.1.1 Tuning Parameters

For the fully centralized DMRAC algorithm with the plant derivative output term and the supplementary feed-forward in the reference model and plant, $\langle BASIC/FF^2/\alpha/bias/disc\rangle$, there are 1182 parameters to be selected, see Table 4.1. At first, this number seems very intimidating, but as we will show, the number of tuning parameters can be greatly reduced by some simplifications and by adjusting the parameters in groups rather than individually.

The most drastic reduction in the number of tuning parameters can be achieved by forcing the integral and proportional adaptation weighting matrices, T_{int} and T_{pro} from (2.68-2.69), to be diagonal. This reduces the number of tuning parameters from 1182 to 78.

The reference model dynamics have 12 tuning parameters, six w_{n_i}'s and six ζ_i's.

Table 4.1: Tunable Parameters for $\langle BASIC/FF^2/\alpha/bias/disc \rangle$

Parameter	Description	Values
T_{int}	24 × 24 integral weighting matrix	576
T_{pro}	24 × 24 proportional weighting matrix	576
w_{n_i}	Undamped natural frequency for Joint i model	6
ζ_i	Damping ratio for Joint i model	6
α	6 × 6 diagonal plant derivative weighing matrix	6
K_{d_i}	DC gain of Joint i supplementary feed-forward block	6
τ_i	Time constant of Joint i supplementary feed-forward block	6
	Total	1182

It is customary in robotic applications to tune controllers such that critical damping is achieved so there is no over shoot. Over shoot may cause a robot end effector to penetrate the surface of its work environment which is not desirable. Thus, all of the damping terms, ζ_i, can be set to 1.0 to achieve critical damping. The undamped natural frequency terms, w_{n_i}, are chosen such that the reference model will have some desired step response. Typically, the reference model dynamics are chosen such that they are "reasonable" for the plant to follow since the DMRAC algorithm will try to force the plant to follow the model output. For the case of a PUMA 560 Manipulator, all of the w_{n_i} were initially set to 5.0. The model's dynamic parameters can be changed as needed if the robot is having problems tracking the model. The model time constant should be greater than 5 times the sample frequency. The number of remaining parameters for tuning is now 66.

Initially, the plant output derivative weights, α, are set to zero leaving 60 parameters. The α weights are used to remove high frequency components from the plant control signal, u_m, and should only be used when needed as they will affect the transient response as discussed in Section 2.6.

The feed-forward filter has 12 tuning parameters, six gains K_{d_i} and six time constants τ_i. A good first choice for the τ_i is approximately one-tenth the model time constant. The τ_i should be kept greather than about 5–10 times the sample

period. For our case, the initial value of the τ_i was set to $20T_s = 20(4.5\ ms) \approx 0.1\ s$. The six DC filter gains can initially be set to 1.0. Increasing the filter gain will typically improve the tracking performance.

The remaining 48 parameters are the diagonal components of T_{pro} and T_{int}. Initially, T_{int} can be set equal to T_{pro} leaving 24 parameters unspecified. A reasonable initial guess for the remaining 24 parameters is $T_{int} = T_{pro} = diag(1, 1, \ldots, 1)$.

4.1.2 Tuning Process

A reasonable method of tuning a DMRA controller is to start the plant at an equilibrium position and apply small step inputs[1]. Set the tuning parameters to the initial values as discussed in the preceding section and check on the step response. With the information on the effects of the tuning parameters on the tracking response (which will be presented in Section 5.2), one can alternately run a simulation (or control the actual plant) and then update the tuning parameter values. This cycle is repeated until the desired performance is achieved. After a reasonable performance is achieved with the step inputs, the DMRAC should be fine tuned using typical plant trajectories.

If the closed loop system is very sensitive to initial conditions, start with small steps as described above, let the system reach steady-state, and then save all of the DMRA controller state information (integral adaptation matrix, K_I; reference model state vector, x_m; and the filter state vector) to be used as initial conditions for the next run. This will significantly cut down the adaptation time required for the gains to reach their steady-state values.

In order to compare the tuning results, some criterion must be established. For our case, the goal was to minimize the peak model following errors and keep the error trajectory as close to zero as possible. Small errors were tolerable during

[1] If step inputs drive the plant unstable, try holding the plant at an equilibrium.

motion. It was also desired to achieve zero error in steady-state.

It is important to note that the DMRAC was tuned to minimize the model following error, $(y_p - y_m)$, not the over all input/output error, $(y_p - u_m)$.

4.1.3 DMRAC Tuning for a PUMA 560 Manipulator

The DMRAC algorithm was tuned by followed the suggestions given above. A 10 degree step from the PUMA 560 stable equilibrium (arm down position[2] with joint angles of { 0, 90, 90, 0, 0, 0 } degrees, see Figure 4.1) was commanded. With the diagonal components of the weighting matrices set to 1.0, the reference model parameters set to $w_n = 5.0$ and $\zeta = 1.0$, the output derivative terms, α, set to 0.0, and the feed-forward terms set to $K_d = 1.0$ and $\tau = 0.1$, the step response is as shown in Figures 4.2 and 4.3 where the solid lines represent the plant output (joint positions) and the dashed lines represent the model outputs. Note the smoothing of the step input introduced by the reference model dynamics. As the plots show, the step response with the initial tuning values is sluggish for Joints 1, 4, 5, and 6 with overshoot and oscillations. Joints 2 and 3 settle into their steady-state values quickly but with a very large steady-state error. The process used to complete the tuning was as follows:

1. Refine the tuning for the 10 degree step from the equilibrium position.

2. Using the refined parameter values, move the robot to the shutdown position, see Figure 3.4, and save the DMRAC internal state values at that position for use as initial conditions.

3. Refine the tuning for a 10 degree step from the shutdown position using the initial conditions from Step 2.

[2]Note: A PUMA 560 in its stable equilibrium does not hang straight down, Joints 2, 3, and 5 are at very slight angles from vertical due to an offset in Joint 3. This slight difference was ignored when tuning without any problems.

4. Refine the tuning from typical min-jerk trajectories from the shutdown position.

The final tuning parameter values after Step 4 are shown in Table 4.2. The weighting matrix values for Joints 1, 2, and 3 differ from the weighting matrix values for the last three joints by a factor of about 100 which reflects the mass/inertia difference between the upper arm and the wrist. The weighting matrix values which are multiplied by the "x_{m2}" products are about a factor of 7 lower than the values multiplying the "x_{m1}" products since the second state variable of each decoupled reference model has a higher peak value in a transient (see Figure 4.6). The Joint 1, 2, and 3 reference models have an undamped natural frequency of 4.0 *rad/sec* where the wrist model used 7.0 *rad/sec* which again reflected the inertia difference between the upper arm and the wrist. The feed-forward filter values were set to $K_d = 6.0$ and $\tau = 0.1$ for all joints. The alpha values were increased from the initial values of zero to damp out some high frequency oscillations.

A typical response to a minimum jerk trajectory using the parameters in Table 4.2 is shown in Figures 4.4 and 4.5 where the solid lines are the plant outputs and the dashed lines are the model outputs. The model following error with the final tuning values was quite good. The peak errors for the plots in Figures 4.4 and 4.5 are shown in Table 4.3.

4.2 Individual Joint Evaluations

This section will investigate the DMRAC algorithm's ability to adapt to the non-linear arm dynamics by first evaluating each joint individually and then looking at the entire joint motion. For the individual joint evaluations, joint trajectories will be selected which check each joint near its minimum and maximum inertias and/or minimum and maximum gravity loading and at different speeds.

Note: The first couple of trajectory segments are normally used to move the

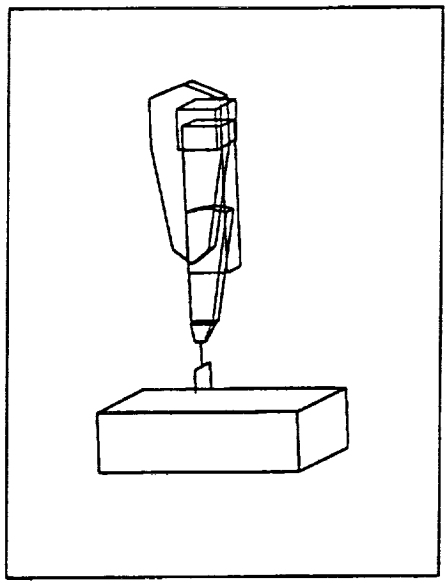

Figure 4.1: PUMA 560 in Stable Equilibrium

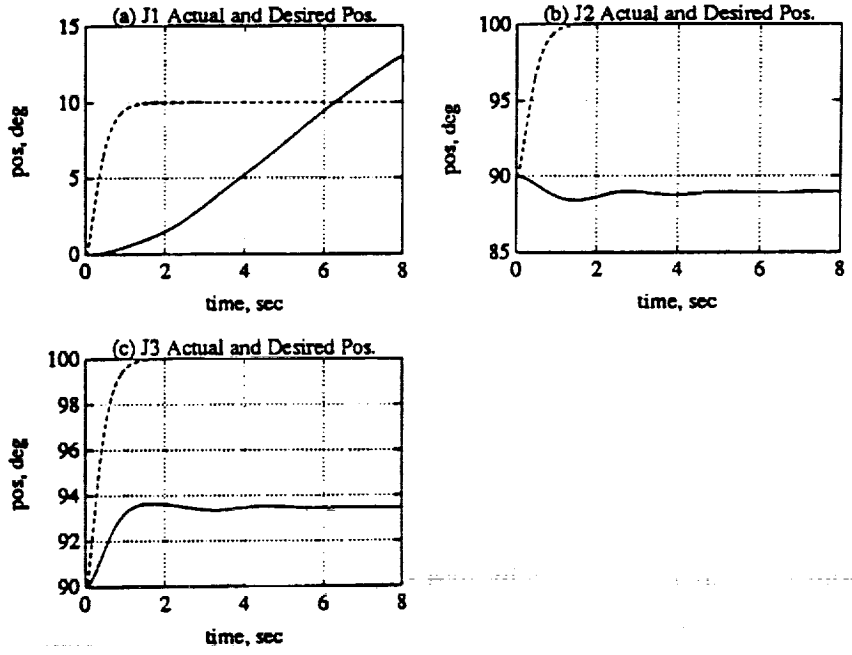

Figure 4.2: Step response using Initial Tuning Parameter Values. (a) Joint 1. (b) Joint 2. (c) Joint 3.

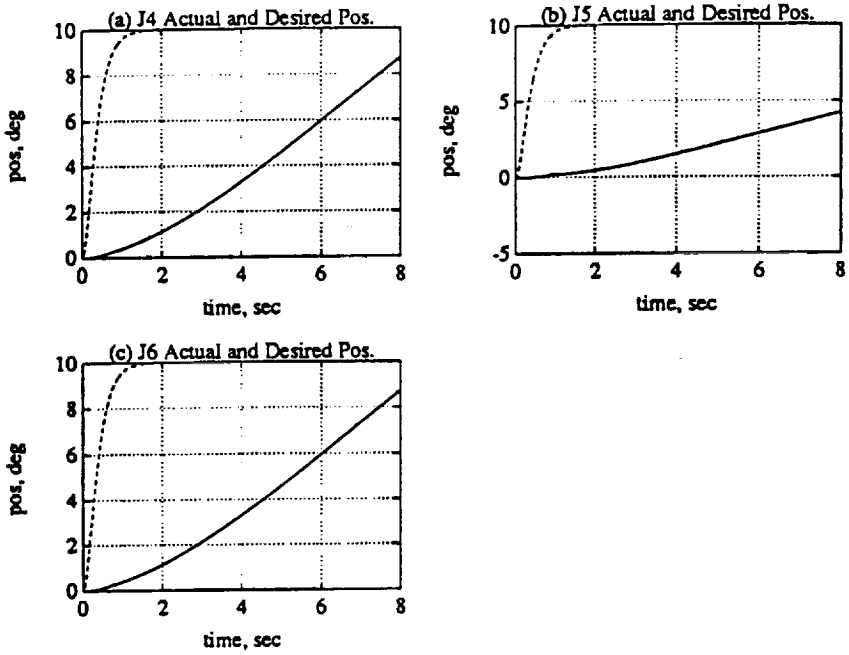

Figure 4.3: Step response using Initial Tuning Parameter Values. (a) Joint 4. (b) Joint 5. (c) Joint 6.

Table 4.2: Final Parameter Values

T_{pro} (diag component)	"e_z"	20	40	22	0.2	0.2	0.2
	"x_m"	140	20	140	35	100	22
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	110	1.4	1.4	1.4
T_{int} (diag component)	"e_z"	20	60	25	0.2	0.2	0.2
	"x_m"	140	20	150	35	140	25
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	130	1.4	1.4	1.4
Joint		1	2	3	4	5	6
Model	w_n	4	4	4	7	7	7
	ζ	1	1	1	1	1	1
Feed Forward	K_d	6	6	6	6	6	6
	τ	0.1	0.1	0.1	0.1	0.1	0.1
alpha	α	0.035	0.02	0.02	0.01	0.01	0.01

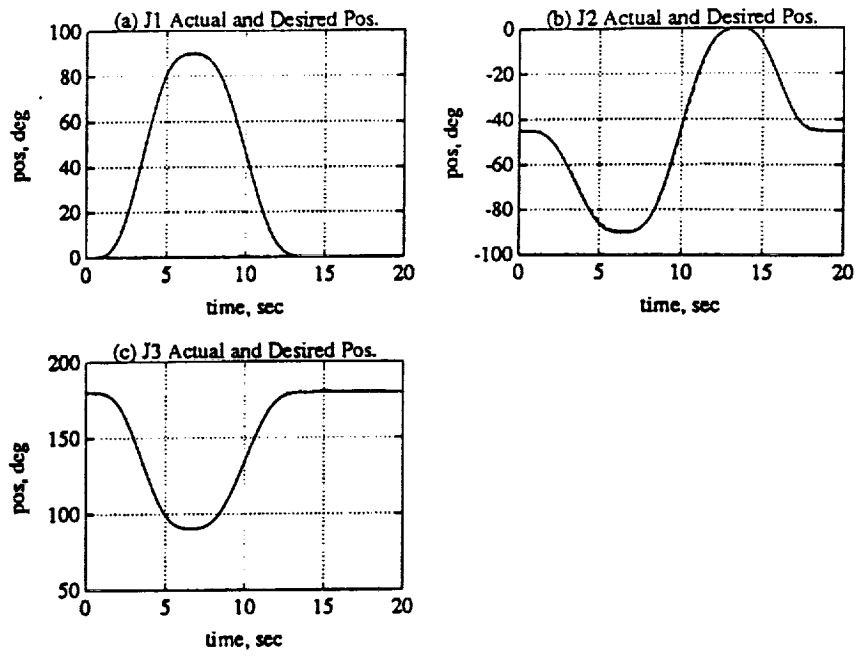

Figure 4.4: Response using Final Tuning Parameter Values. (a) Joint 1. (b) Joint 2. (c) Joint 3.

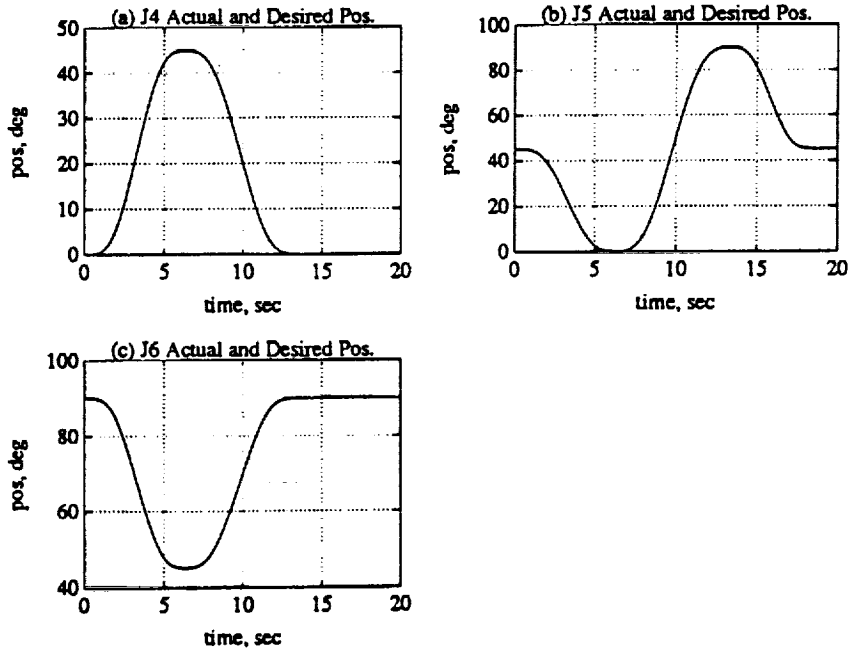

Figure 4.5: Response using Final Tuning Parameter Values. (a) Joint 4. (b) Joint 5. (c) Joint 6.

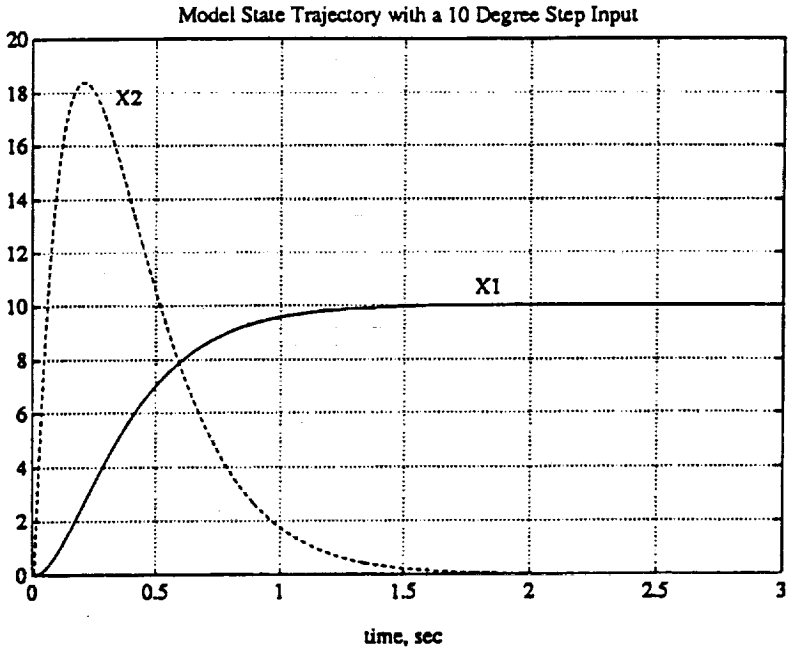

Figure 4.6: Step Response of Reference Model with $w_n = 5.0$ and $\zeta = 1.0$

Table 4.3: Peak Errors for Final Tuning Values

Joint	Peak Error (degrees)
1	-1.086
2	1.759
3	0.6481
4	0.1751
5	-0.373
6	-0.2529

Table 4.4: Parameter Values for Joint Evaluation Runs

T_{pro} (diag component)	"e_z"	20	40	22	0.2	0.2	0.2
	"x_m"	140	20	140	35	100	22
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	110	1.4	1.4	1.4
T_{int} (diag component)	"e_z"	20	60	25	0.2	0.2	0.2
	"x_m"	140	20	150	35	140	25
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	130	1.4	1.4	1.4
Joint		1	2	3	4	5	6
Model	w_n	4	4	4	7	7	7
	ζ	1	1	1	1	1	1
Feed Forward	K_d	6	6	6	6	6	6
	τ	0.1	0.1	0.1	0.1	0.1	0.1
alpha	α	0.02	0.02	0.02	0.02	0.02	0.02

robot from the shutdown position to the "range of interest" and may produce large joint tracking errors since the segment times are small. The trajectory segments which are starred in the Tables are the ones of interest.

The tuning parameter values used for the joint evaluations are given in Table 4.4.

4.2.1 Joint One Evaluation

The first joint was evaluated using two trajectories (see Tables 4.5 and 4.6). These trajectories present joint one with its maximum inertia, Table 4.5, and its minimum innertia, Table 4.6, at four different speeds. Figure 4.7 shows a top view of the first trajectory where the numbered positions refer to knot point positions in the Table. The second trajectory is simply a repeat of the first trajectory only with the arm straight up rather then straight out.

The response of Joint 1 to the first trajectory is shown in Figure 4.8. In Figure 4.8(a), the solid line shows the first joint actual position and the dashed line (not visible) shows the desired position (model output, y_m). Figure 4.8(c)

Table 4.5: Joint 1 Evaluation Trajectory (Maximum Innertia)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	90	0	0	0	2
2	0	0	90	0	0	0	1
3*	90	0	90	0	0	0	2
4*	0	0	90	0	0	0	3
5*	-90	0	90	0	0	0	4
6*	0	0	90	0	0	0	5
7	0	0	90	0	0	0	10

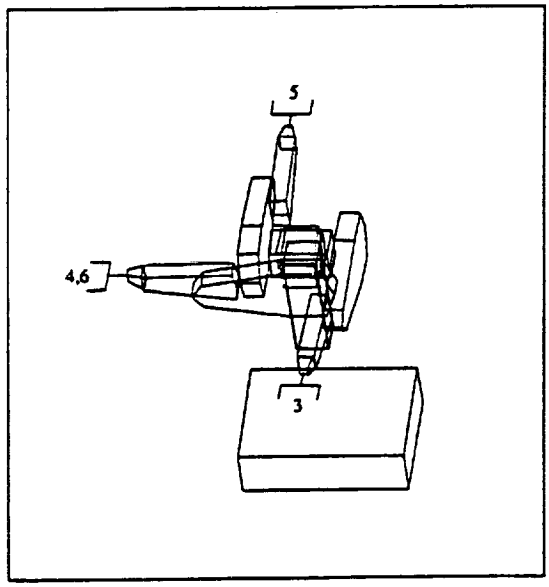

Figure 4.7: Trajectory Used to Evaluate Joint 1

shows the error between plant and model. As the figure shows, the maximum error was about -1.8 degrees for the 2 second segment, 1.25 degrees for the 3 second segment, 1.0 degrees for the 4 second segment, and about -0.8 degrees for the 5 second segment. This indicates that the DMRAC algorithm has a more difficult time tracking faster trajectories which is expected. The joint torque is shown in Figure 4.8(b). The overall performance for this high innertia trajectory is quite satisfactory. Figure 4.8(d) shows the lag introduced by the model. As was mentioned before, the lag is being ignored since it is predictable and can be compensated for by an appropriate predictive trajectory generator. We will instead concentrate on the error between model and plant as was stated in our original DMRAC goal.

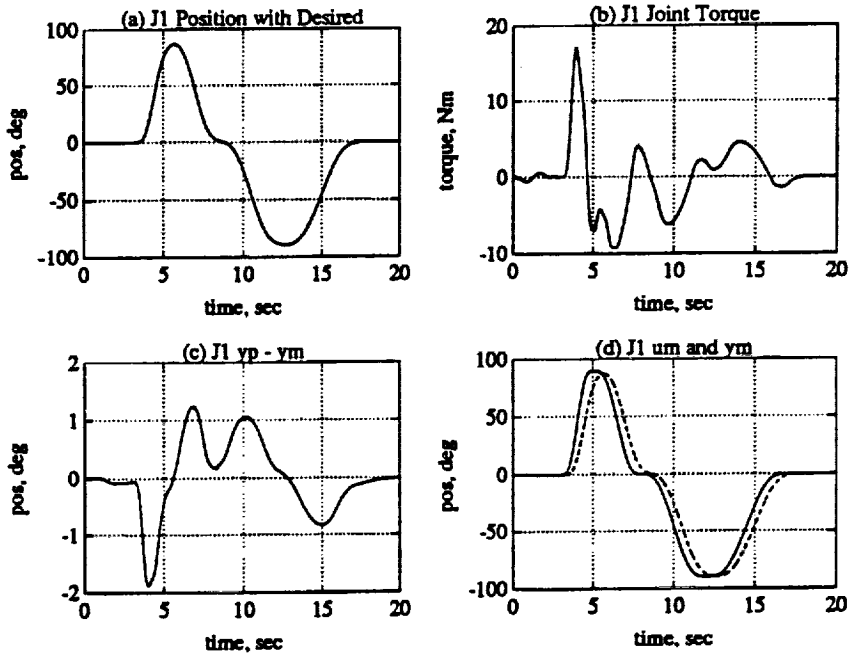

Figure 4.8: Joint 1 Evaluation, Maximum Inertia. (a) Position. (b) Torque. (c) Model following error. (d) Model input and Output.

The response of Joint 1 to the second trajectory is shown in Figure 4.9. This trajectory presents Joint 1 with its minimum inertia. Comparison of the error plot, Figure 4.9(c), to the previos case, Figure 4.8(c), shows that the response is about

Table 4.6: Joint 1 Evaluation Trajectory (Minimum Innertia)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	90	0	0	0	2
2	0	0	90	0	0	0	1
3*	90	0	90	0	0	0	2
4*	0	0	90	0	0	0	3
5*	-90	0	90	0	0	0	4
6*	0	0	90	0	0	0	5
7	0	0	90	0	0	0	10

the same where this case has slightly better tracking. The interesting result is that the joint torque signal, Figure 4.8b, contains a high frequency component with an amplitude of about $\pm 0.2 Nm$ which was not present in the previous case.

4.2.2 Joint Two Evaluation

The second joint was evaluated using three trajectories (see Tables 4.7, 4.8, and 4.9). These trajectories present Joint 2 with its maximum gravity loading, Table 4.7, its minimum gravity loading, Table 4.8, and a coupling effect Table 4.9, at four different speeds. The first two trajectories allow Joint 2 to see its maximum innertia. Figures 4.10 and 4.12 show a side view of the first and second trajectories respectively, where the numbered positions refer to knot points in the Tables. Figure 4.14 shows a view of the third trajectory.

The response of Joint 2 to the first trajectory is shown in Figure 4.11. As the error plot shows, Figure 4.11(c), the peak error for the 2 second trajectory segment was around 2.8 degrees. Joint 2 did not recover from this error until about 11 seconds into the trajectory. The 4 and 5 second trajectory segments both had peak errors less than 1.0 degree. The joint torque signal is shown in Figure 4.11(b) and indicates some ringing during the fast portions of the trajectory ($5 \leq t \leq 10$).

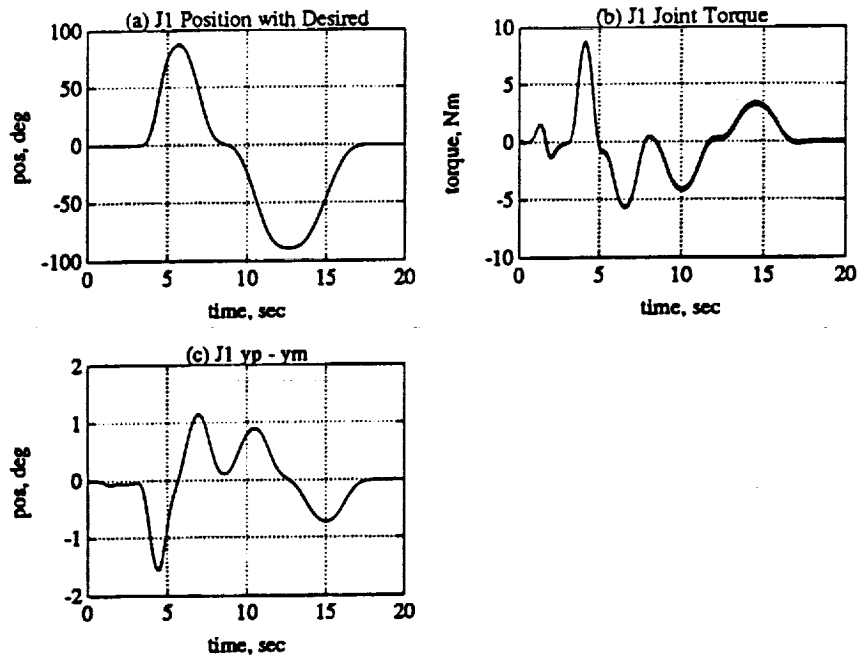

Figure 4.9: Joint 1 Evaluation, Minimum Inertia. (a) Position. (b) Torque. (c) Model following error.

Table 4.7: Joint 2 Evaluation Trajectory (Maximum Gravity Load)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	90	0	0	0	2
2	0	0	90	0	0	0	3
3*	0	20	90	0	0	0	2
4*	0	0	90	0	0	0	3
5*	0	-20	90	0	0	0	4
6*	0	0	90	0	0	0	5
7	0	0	90	0	0	0	10

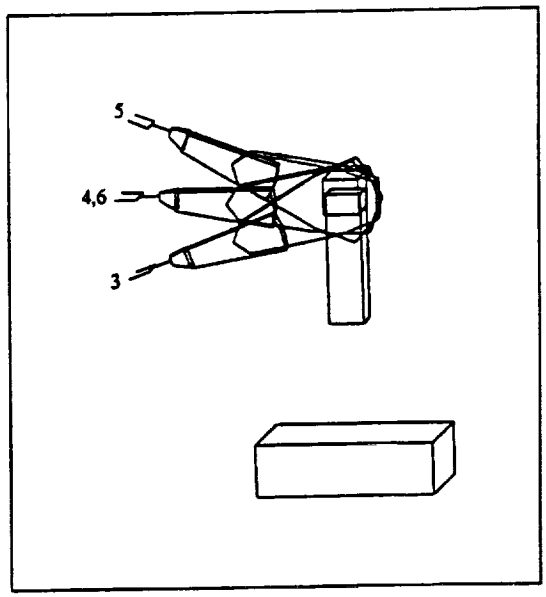

Figure 4.10: Trajectory Used to Evaluate Joint 2, Maximum Gravity Loading

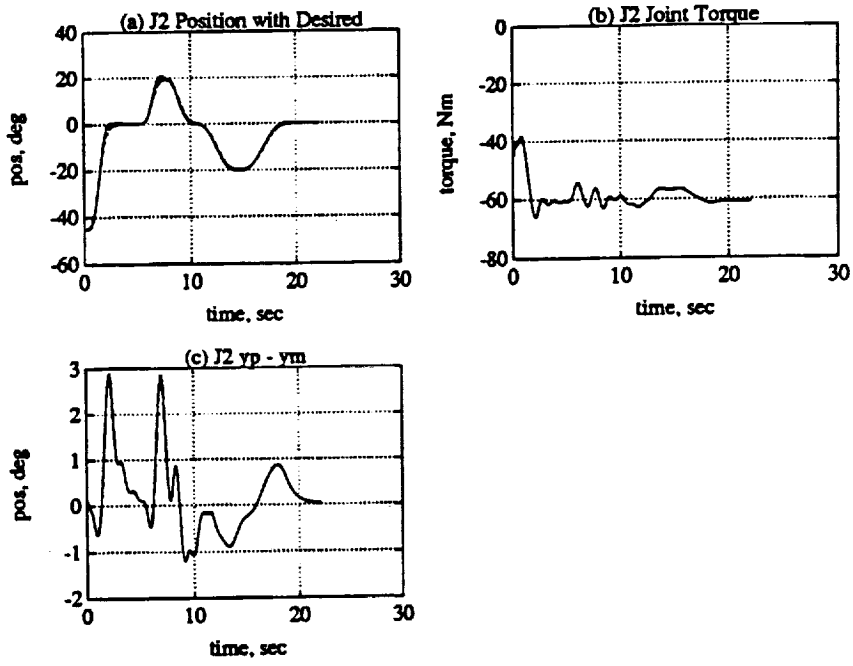

Figure 4.11: Joint 2 Evaluation, Maximum Gravity Loading. (a) Position. (b) Torque. (c) Model following error.

Table 4.8: Joint 2 Evaluation Trajectory (Minimum Gravity Load)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	-90	90	0	0	0	2
2	0	-90	90	0	0	0	3
3*	0	-70	90	0	0	0	2
4*	0	-90	90	0	0	0	3
5*	0	-110	90	0	0	0	4
6*	0	-90	90	0	0	0	5
7	0	-90	90	0	0	0	10

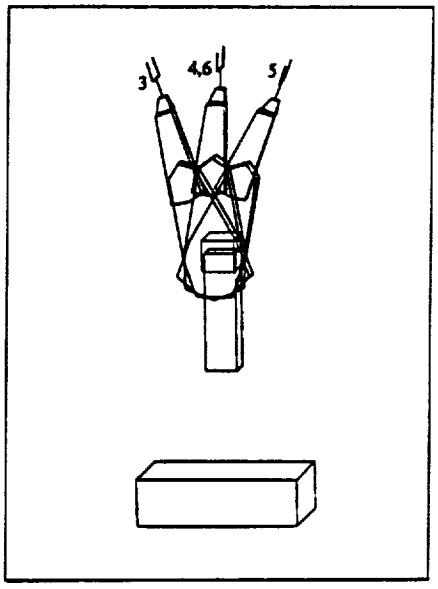

Figure 4.12: Trajectory Used to Evaluate Joint 2, Minimum Gravity Loading

The response of Joint 2 to the second trajectory is shown in Figure 4.13. The error plot in Figure 4.13(c) shows the improved tracking performance over the previous case. For the 2 second trajectory segment, the peak error was about 0.6 degrees. The error remains within ± 0.3 degrees for the 3, 4, and 5 second trajectory segments. The joint torque signal, Figure 4.13(b) is just beginning to show a small amount of high frequency oscillation.

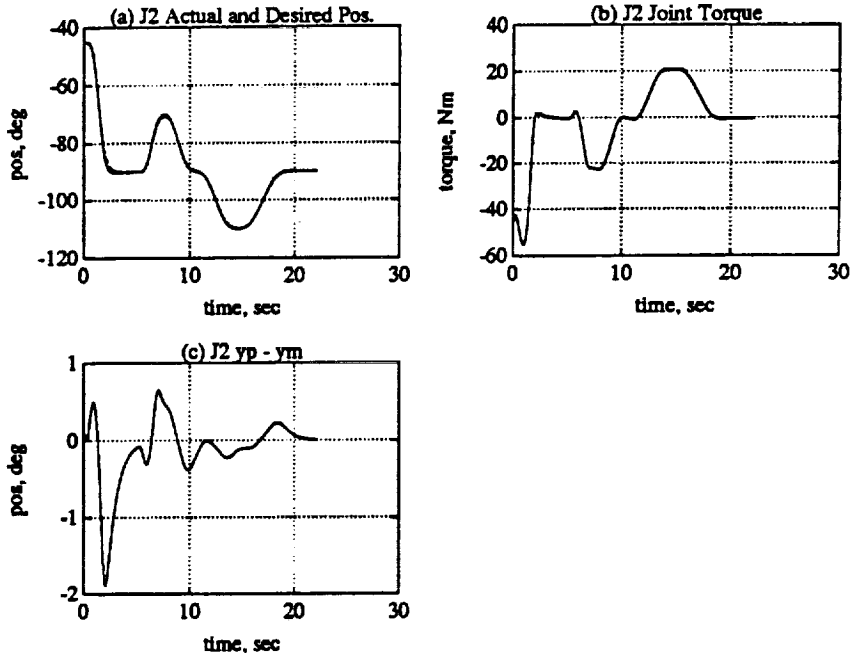

Figure 4.13: Joint 2 Evaluation, Minimum Gravity Loading. (a) Position. (b) Torque. (c) Model following error.

The final trajectory subjects Joint2 to a centrifugal force from Joint 1. The swinging motion of Joint1 is used to apply a centrifugal torque to Joint 2. Figure 4.14 shows the trajectory. The response of Joint 2 to the coupling trajectory is shown in Figure 4.15. The peak tracking errors for the 2, 3, 4, and 5 second trajectory segments all remain within ± 0.34 degrees. The Joint 2 link torque signal is well behaved, Figure 4.15(b). The DMRAC does not seem to have trouble adapting for the centrifugal torque.

Table 4.9: Joint 2 Evaluation Trajectory (Coupling Effect)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	-45	-45	0	0	0	2
2	0	-45	-45	0	0	0	2
3*	30	-45	-45	0	0	0	2
4*	0	-45	-45	0	0	0	3
5*	-30	-45	-45	0	0	0	4
6*	0	-45	-45	0	0	0	5
7	0	-45	-45	0	0	0	10

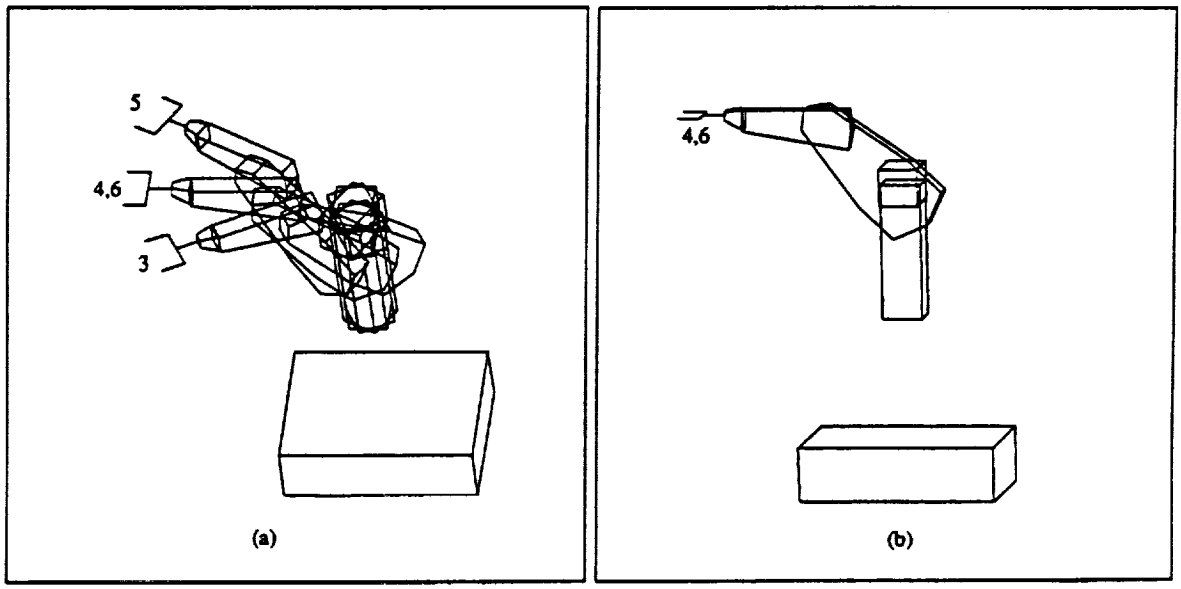

Figure 4.14: Trajectory Used to Evaluate Joint 2, Coupling Effect

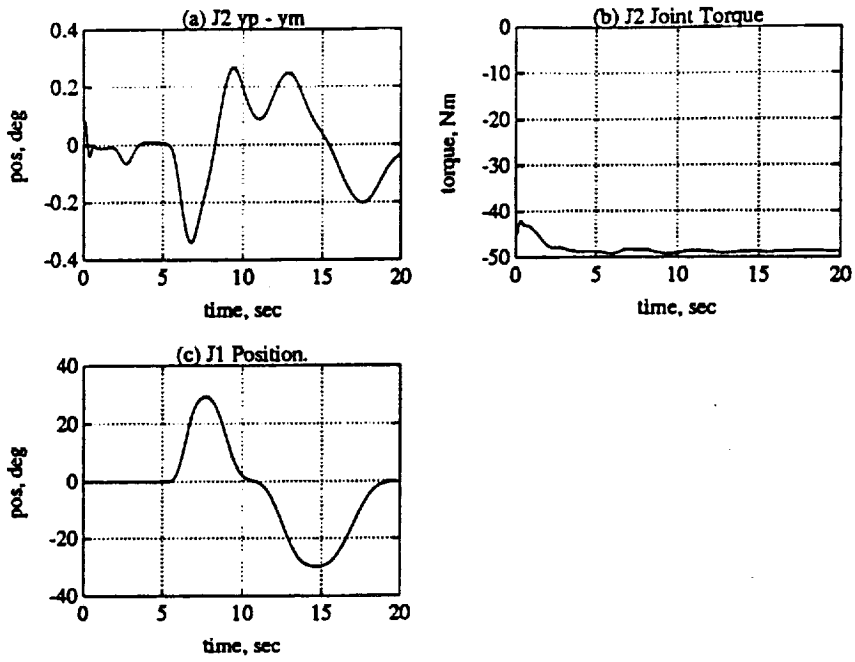

Figure 4.15: Joint 2 Evaluation, Coupling Effect. (a) Model following Error. (b) Torque. (c) Joint 1 Position.

4.2.3 Joint Three Evaluation

The third joint was evaluated using three trajectories similar to the ones used for Joint 2. (see Tables 4.10, 4.11, and 4.12). These trajectories present Joint 3 with its maximum gravity loading, Table 4.10, its minimum gravity loading, Table 4.11, and a coupling effect Table 4.12, at four different speeds. Figures 4.16 and 4.18 show a side view of the first and second trajectories respectively, where the numbered positions refer to knot points in the Tables. Figure 4.20 shows a view of the third trajectory.

The response of Joint 3 to the first trajectory is shown in Figure 4.17. As Figure 4.17(c) shows, the error is bounded by $\pm$ 0.4 for all four test trajectory segments.

The response of Joint 3 to the second trajectory is shown in Figure 4.19. The error, Figure 4.19(c) remains bounded by $\pm$ 0.3 over the 2, 3, 4, and 5 second

Table 4.10: Joint 3 Evaluation Trajectory (Maximum Gravity Load)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	90	0	0	0	3
2	0	0	90	0	0	0	2
3*	0	0	110	0	0	0	2
4*	0	0	90	0	0	0	3
5*	0	0	70	0	0	0	4
6*	0	0	90	0	0	0	5
7	0	0	90	0	0	0	10

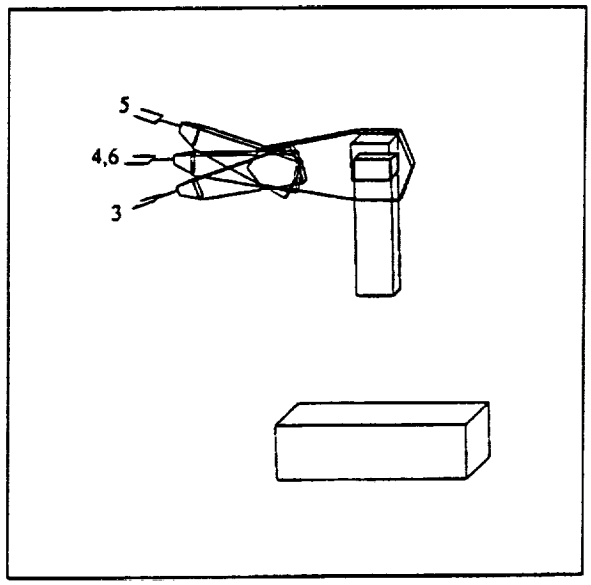

Figure 4.16: Trajectory Used to Evaluate Joint 3, Maximum Gravity Loading

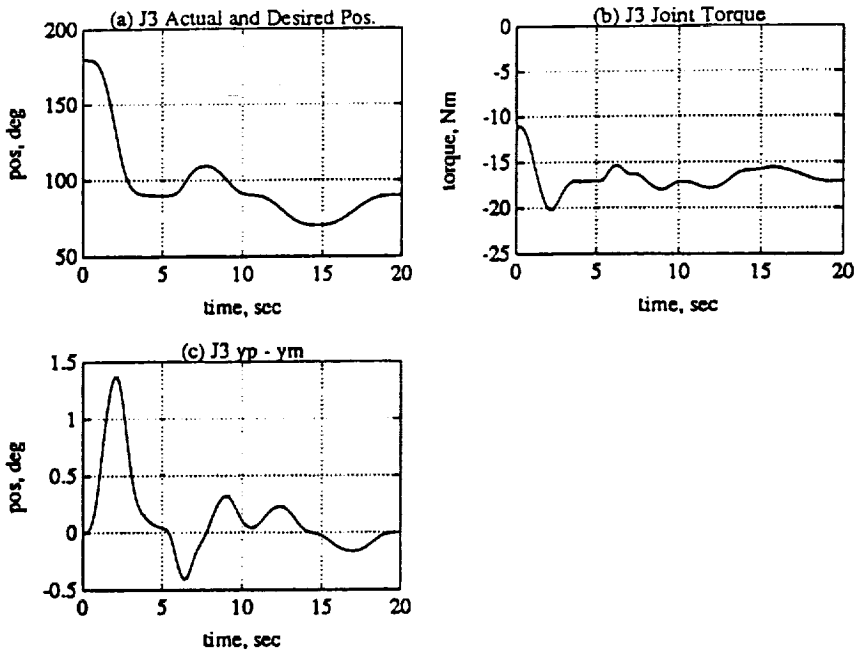

Figure 4.17: Joint 3 Evaluation, Maximum Gravity Loading. (a) Position. (b) Torque. (c) Model Following Error

Table 4.11: Joint 3 Evaluation Trajectory (Minimum Gravity Load)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	0	0	0	0	3
2	0	0	0	0	0	0	2
3*	0	0	20	0	0	0	2
4*	0	0	0	0	0	0	3
5*	0	0	-20	0	0	0	4
6*	0	0	0	0	0	0	5
7	0	0	0	0	0	0	10

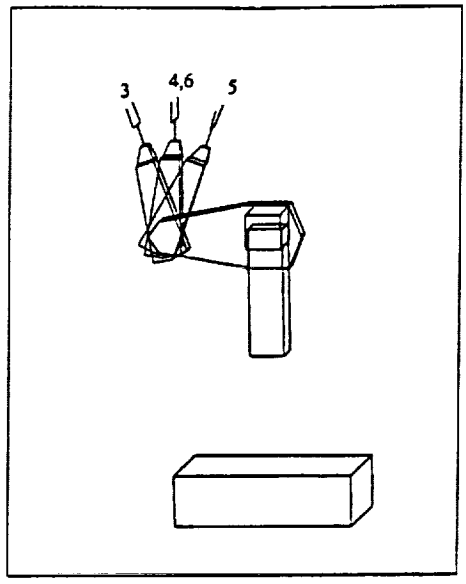

Figure 4.18: Trajectory Used to Evaluate Joint 3, Minimum Gravity Loading

trajectory segments. The large error signal for $0 \leq t \leq 5$ is caused by the fast trajectory of Joint 3 to position the arm for this test.

The response of Joint 3 to the coupling trajectory is shown in Figure 4.21. The error for the coupling evaluation, Figure 4.21(a) remains bounded by $\pm$ 0.05 degrees for the four trajectory segments of interest. The large error for $0 \leq t \leq 5$ is caused by the fast trajectory of Joint 3 used to position the robot for this evaluation

Table 4.12: Joint 3 Evaluation Trajectory (Coupling)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	0	0	0	0	3
2	0	0	0	0	0	0	2
3*	30	0	0	0	0	0	2
4*	0	0	0	0	0	0	3
5*	-30	0	0	0	0	0	4
6*	0	0	0	0	0	0	5
7	0	0	0	0	0	0	10

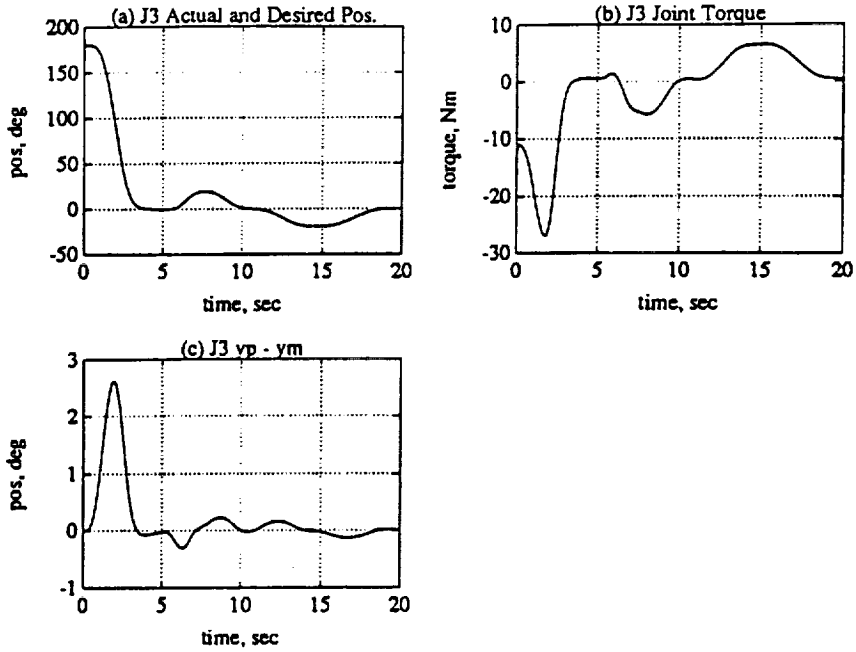

Figure 4.19: Joint 3 Evaluation, Minimum Gravity Loading. (a) Position. (b) Torque. (c) Model Following Error

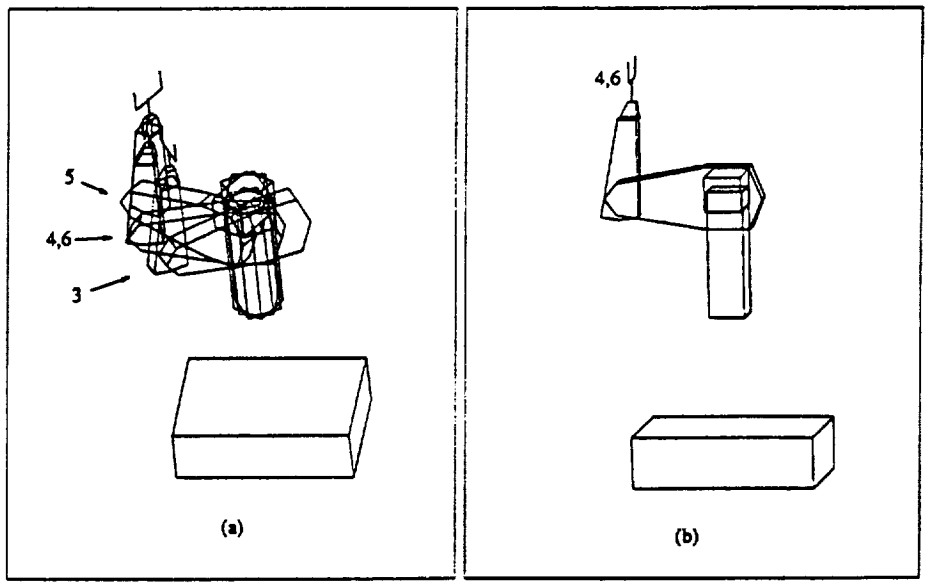

Figure 4.20: Trajectory Used to Evaluate Joint 3, Coupling Effect

run. Figure 4.21(d) shows the position of Joint1 for reference.

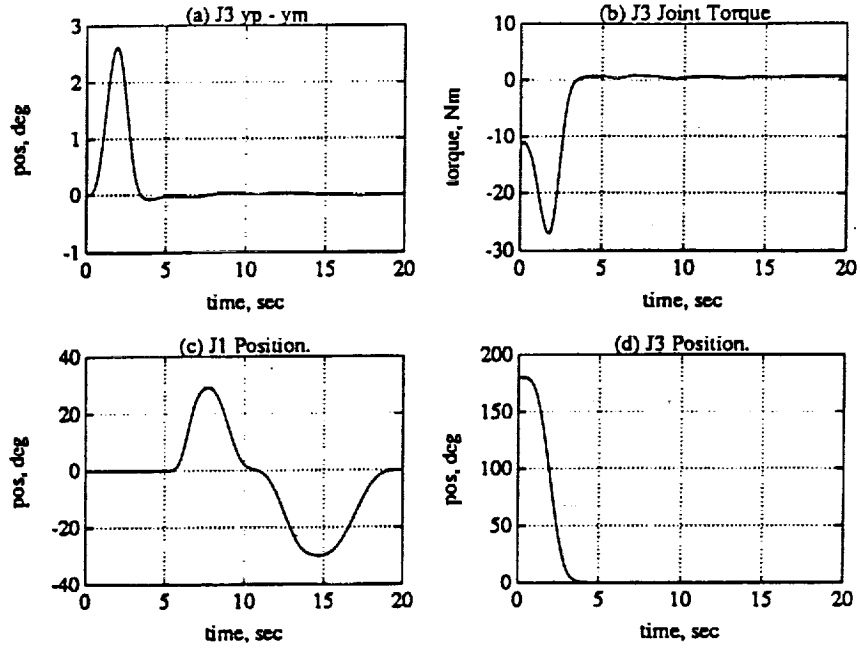

Figure 4.21: Joint 3 Evaluation, Coupling. (a) Model following error. (b) Torque. (c) Joint 1 Position. (d) Joint 3 Position

4.2.4 Joint Four Evaluation

The forth joint was evaluated using two trajectories (see Tables 4.13 and 4.14). These trajectories present Joint 4 with its maximum inertia, Table 4.13, and its minimum inertia, Table 4.14, at four different speeds. Figures 4.22 and 4.24 show a view of the first and second trajectories respectively, where the numbered positions refer to knot points in the Tables.

The response of Joint 4 to the first trajectory is shown in Figure 4.23. Joint 4 tracked the 2 second trajectory segment with a peak error of -1.0 degree, the 3 second segment with a peak error of 0.7 degrees, the 4 second segment with a peak error of 0.5 degrees, and the final 5 second trajectory segment with a peak error of -0.4 degrees.

Table 4.13: Joint 4 Evaluation Trajectory (Maximum Inertia)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	180	0	90	0	3
2	0	0	180	0	90	0	2
3*	0	0	180	45	90	0	2
4*	0	0	180	0	90	0	3
5*	0	0	180	-45	90	0	4
6*	0	0	180	0	90	0	5
7	0	0	180	0	90	0	10

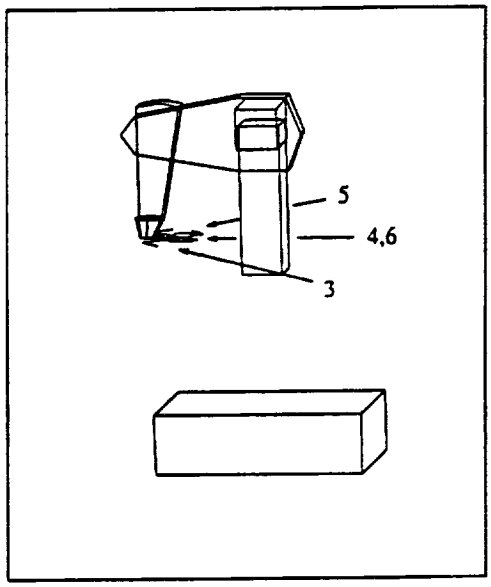

Figure 4.22: Trajectory Used to Evaluate Joint 4, Maximum Inertia

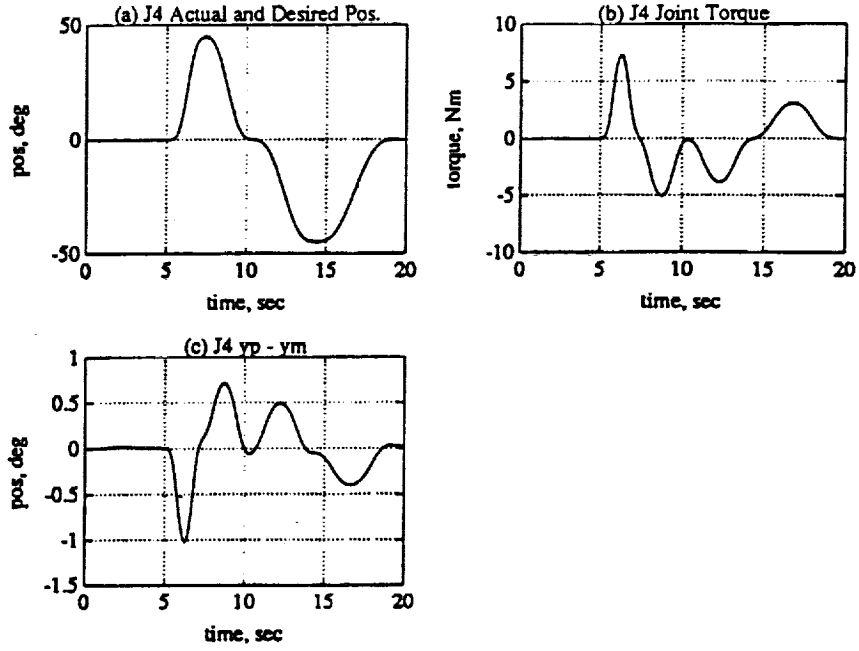

Figure 4.23: Joint 4 Evaluation, Maximum Inertia. (a) Position. (b) Torque. (c) Model following error.

Table 4.14: Joint 4 Evaluation Trajectory (Minimum Inertia)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	180	0	0	0	3
2	0	0	180	0	0	0	2
3*	0	0	180	45	0	0	2
4*	0	0	180	0	0	0	3
5*	0	0	180	-45	0	0	4
6*	0	0	180	0	0	0	5
7	0	0	180	0	0	0	10

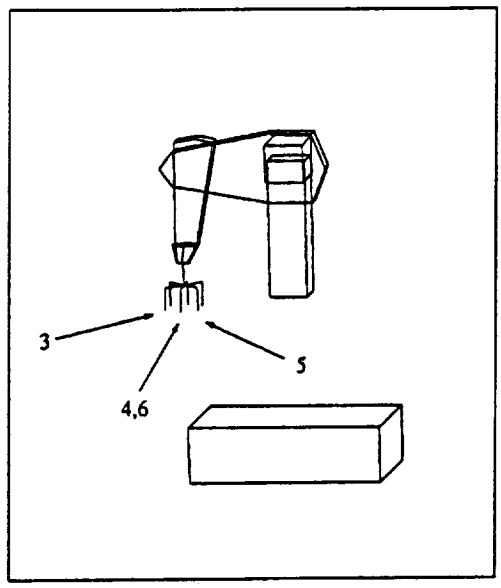

Figure 4.24: Trajectory Used to Evaluate Joint 4, Minimum Inertia

The response of Joint 4 to the second trajectory is shown in Figure 4.25. The tracking performance, Figure 4.25(c) for the second case is nearly identical to that of the previous case, Figure 4.23(c). The peak errors for the 2, 3, 4, and 5 second trajectory segments were approximately -1.0, 0.7, 0.5, and -0.4 degrees, respectively.

4.2.5 Joint Five Evaluation

The fifth joint was evaluated using two trajectories (see Tables 4.15 and 4.16). These trajectories presented Joint 5 with its minimum gravity loading, Table 4.13. and its maximum gravity loading, Table 4.14, at four different speeds. Figures 4.22 and 4.24 show a view of the first and second trajectories, respectively, where the numbered positions refer to knot points in the Tables.

The response of Joint 5 to the first trajectory is shown in Figure 4.27. As the error plot in Figure 4.27(c) shows, Joint 5 tracks the 2, 3, 4, and 5 second trajectory segments with a peak error of -1.1, 0.7, 0.5, and -0.4 degrees, respectively.

The response of Joint 5 to the second trajectory is shown in Figure 4.29. For

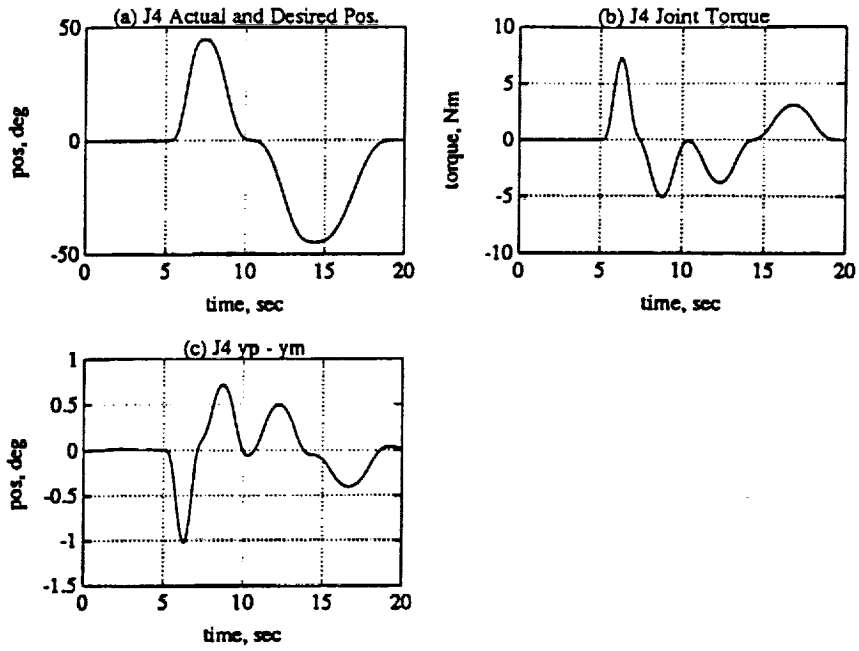

Figure 4.25: Joint 4 Evaluation, Minimum Inertia. (a) Position. (b) Torque. (c) Model following error.

Table 4.15: Joint 5 Evaluation Trajectory (Minimum Gravity Loading)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	180	0	0	0	3
2	0	0	180	0	0	0	2
3*	0	0	180	0	0	0	2
4*	0	0	180	0	45	0	3
5*	0	0	180	0	0	0	4
6*	0	0	180	0	-45	0	5
7	0	0	180	0	0	0	10

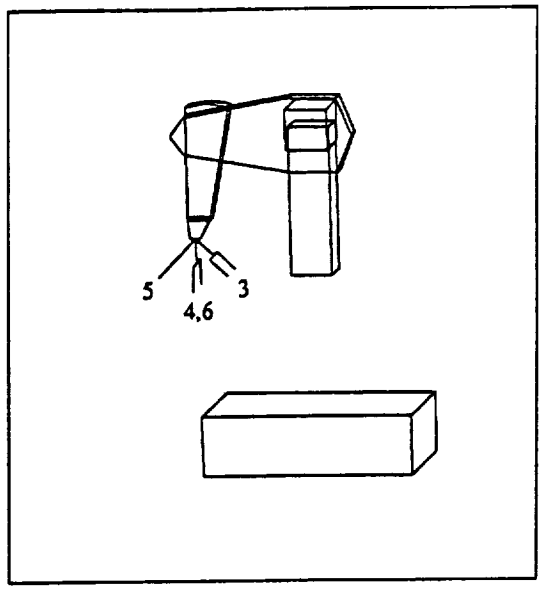

Figure 4.26: Trajectory Used to Evaluate Joint 5, Minimum Gravity Loading

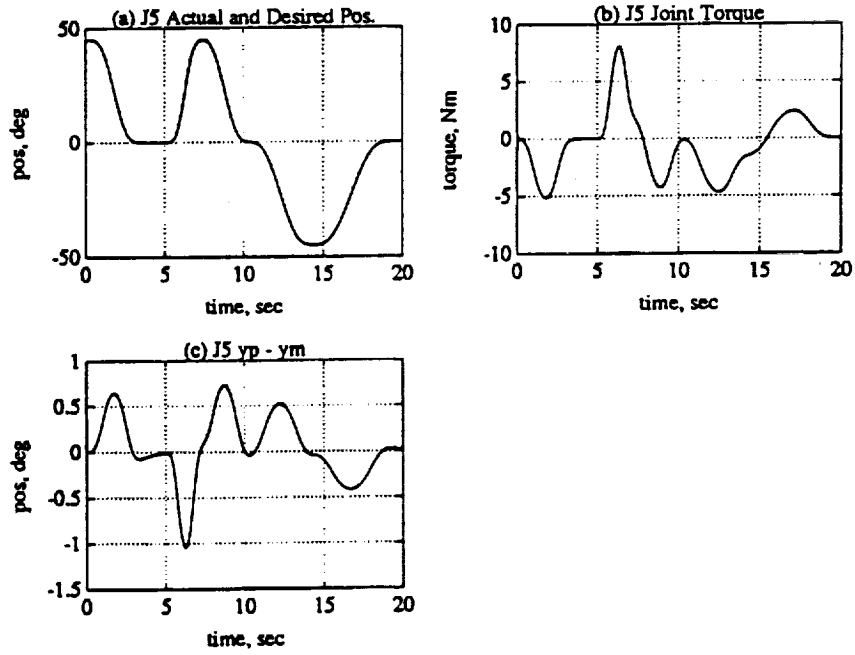

Figure 4.27: Joint 5 Evaluation, Minimum Gravity Loading. (a) Position. (b) Torque. (c) Model following error.

Table 4.16: Joint 5 Evaluation Trajectory (Maximum Gravity Loading)

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	90	0	45	0	-
1	0	0	90	0	0	0	3
2	0	0	90	0	0	0	2
3*	0	0	90	0	0	0	2
4*	0	0	90	0	45	0	3
5*	0	0	90	0	0	0	4
6*	0	0	90	0	-45	0	5
7	0	0	90	0	0	0	10

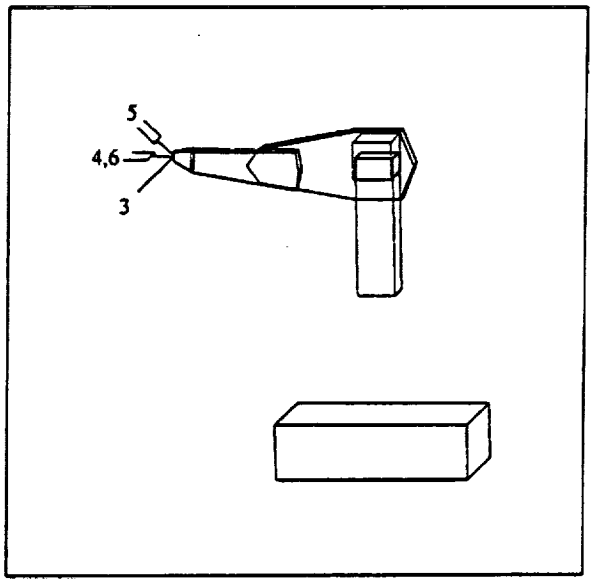

Figure 4.28: Trajectory Used to Evaluate Joint 5, Maximum Gravity Loading

the second trajectory, the peak error increased slightly over those of the first case. The peak errors for the four trajectory segments were -1.2, 0.8, 0.6, and -0.45 degrees for the 2, 3, 4, and 5 second segments, respectively.

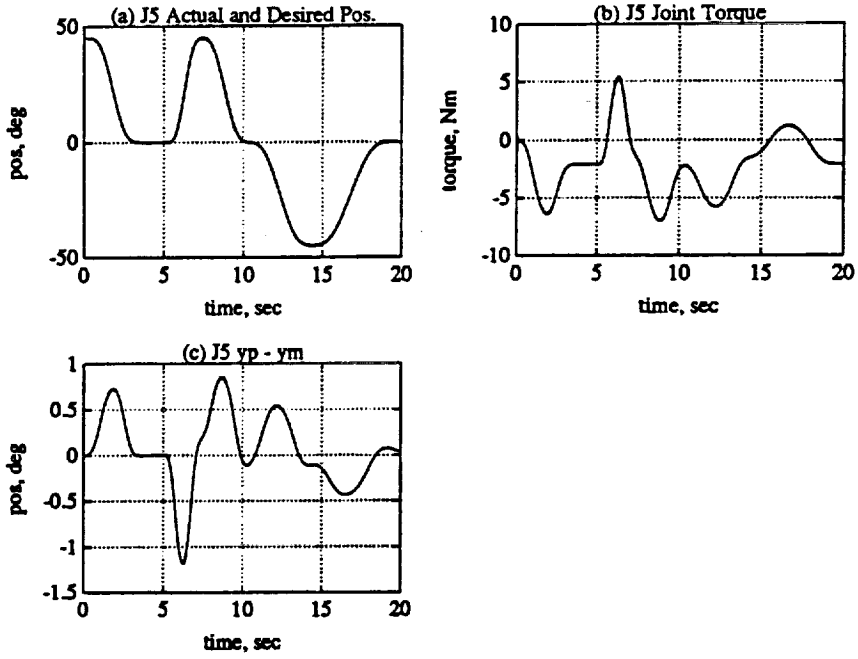

Figure 4.29: Joint 5 Evaluation, Maximum Gravity Loading. (a) Position. (b) Torque. (c) Model following error.

4.2.6 Joint Six Evaluation

The sixth joint was evaluated using the trajectory in Table 4.17. Figure 4.30 shows a view of the trajectory where the numbered positions refer to knot points in the Tables.

The response of Joint 6 to the trajectory is shown in Figure 4.31. The peak tracking errors for the 2, 3, 4, and 5 second segments were -1.0, 0.7, 0.5, and -0.4 degrees respectively.

Table 4.17: Joint 6 Evaluation Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	0	-
1	0	0	180	0	0	0	3
2	0	0	180	0	0	0	2
3*	0	0	180	0	0	0	2
4*	0	0	180	0	0	45	3
5*	0	0	180	0	0	0	4
6*	0	0	180	0	0	-45	5
7	0	0	180	0	0	0	10

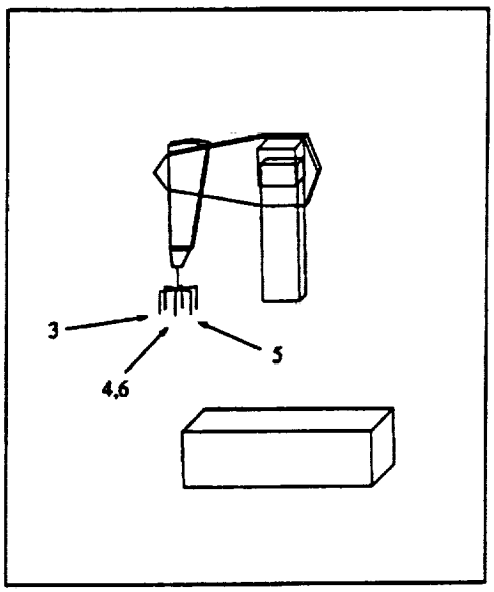

Figure 4.30: Trajectory Used to Evaluate Joint 6

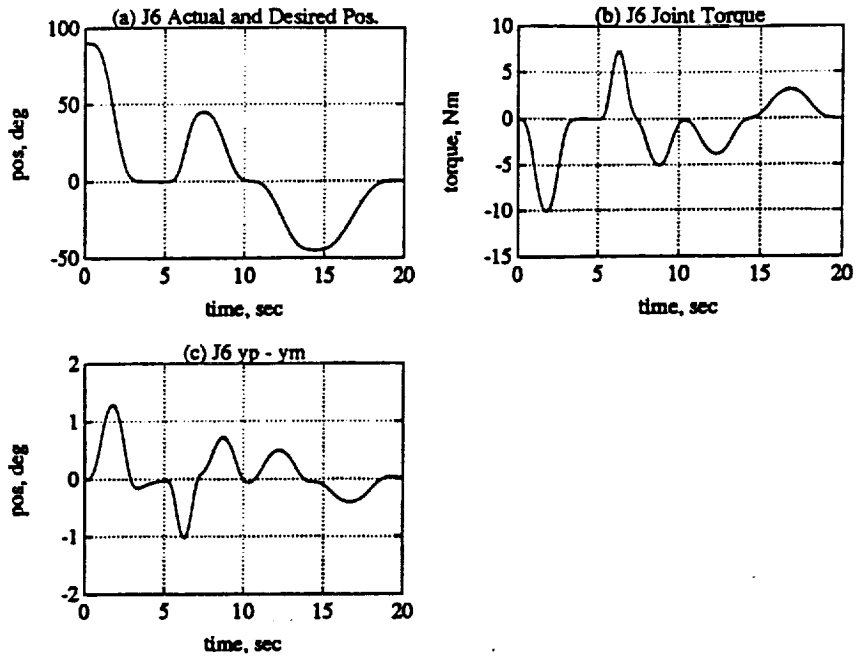

Figure 4.31: Joint 6 Evaluation. (a) Position. (b) Torque. (c) Model follwoing error

4.3 Summary

In this chapter we discussed and illustrated the tuning of a DMRAC algorithm and showed its application to a PUMA 560 Manipulator. Next, we ran some simulations to evaluate each of the six PUMA joints individually. The DMRAC controlled joints performed quite well.

Table 4.18 shows a summary of the peak tracking errors for all of the various joint evaluation runs. As the table shows, the tracking performance was quite good. The worst case tracking errors occurred when the algorithm was adapting to the changing gravity vector.

Table 4.18: Joint Evaluations Summary, Simulation

Joint	Peak Errors in Degrees				Trajectory (see Table)
	2–3	3–4	4–5	5–6	
	(2 sec)	(3 sec)	(4 sec)	(5 sec)	
1	-1.90	1.25	1.08	-0.83	4.5 (Maximum Inertia)
1	-1.55	1.14	0.91	-0.73	4.6 (Minimum Inertia)
2	2.83	-1.22	-0.89	0.39	4.7 (Maximum Gravity Load)
2	0.65	-0.38	-0.23	0.19	4.8 (Minimum Gravity Load)
2	-0.34	0.26	0.25	-0.20	4.8 (Coupling Effect)
3	-0.40	0.33	0.23	-0.17	4.10 (Maximum Gravity Load)
3	-0.32	0.23	0.16	-0.14	4.11 (Minimum Gravity Load)
3	-0.08	0.04	0.03	-0.02	4.12 (Coupling Effect)
4	-1.03	0.72	0.51	-0.40	4.13 (Maximum Inertia)
4	-1.04	0.72	0.51	-0.40	4.14 (Minimum Inertia)
5	-1.05	0.73	0.51	-0.42	4.15 (Minimum Gravity Load)
5	-1.18	0.87	0.53	-0.43	4.16 (Maximum Gravity Load)
6	-1.04	0.74	0.50	-0.42	4.17

CHAPTER 5

Simulation Results (Trajectory Tracking Cases and Parameter Effects)

This chapter will present the results of the Matlab simulations of Direct Model Reference Adaptive Control of a six link PUMA 560 Manipulator, fully-centralized. First the arm will be commanded to track some six joint trajectories. Then, after the general performace characteristics are determined, the effects of changing the tuning parameters will be investigated. All results will be displayed with the bias term, q_{bias} removed (Section 2.7).

5.1 Tracking of 6 Joint Trajectories

This section will investigate the ability of the DMRAC controlled PUMA 560 to track some six joint trajectories. The tuning parameter values used for these runs are shown in Table 5.1.

5.1.1 Tracking Trajectory #1

The first trajectory is shown in Table 5.2 and is illustrated by Figure 5.1 where the numbers correspond to the knot points in the table. The arm first moves to a straight up position, curls up, and then moves back to the safe position.

The model following error plots are shown in Figure 5.2 and the peak errors for each of the joints are summarized in Table 5.3. Joints 1 and 2 had the worst performance with peak errors of -1.10 degrees and 1.76 degrees. Joints 3 through 6 had error trajectories within $\pm$ 1.0 degree. The torque signals for the first four joints are shown in Figure 5.3.

Table 5.1: Parameter Values for 6 Joint Trajectory Tracking Runs

T_{pro} (diag component)	"e_z"	20	40	22	0.2	0.2	0.2
	"x_m"	140	20	140	35	100	22
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	110	1.4	1.4	1.4
T_{int} (diag component)	"e_z"	20	60	25	0.2	0.2	0.2
	"x_m"	140	20	150	35	140	25
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	130	1.4	1.4	1.4
Joint		1	2	3	4	5	6
Model	w_n	4	4	4	7	7	7
	ζ	1	1	1	1	1	1
Feed Forward	K_d	6	6	6	6	6	6
	τ	0.1	0.1	0.1	0.1	0.1	0.1
alpha	α	0.035	0.02	0.02	0.01	0.01	0.01

Table 5.2: First Six Joint Tracking Test Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	90	-
1*	90	-90	90	45	0	45	6
2*	0	0	180	0	90	90	7
3*	0	-45	180	0	45	90	5

Table 5.3: Peak Errors for First Trajectory

Joint	Peak Error (deg)
1	-1.0889
2	1.7596
3	0.6527
4	0.1728
5	-0.3613
6	0.2347

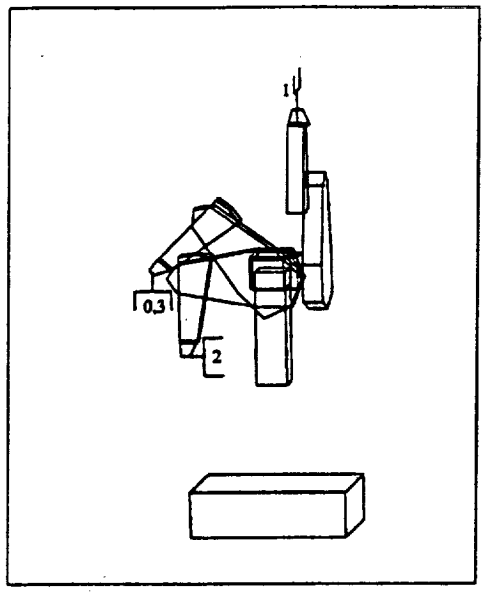

Figure 5.1: First Six Joint Tracking Test Trajectory

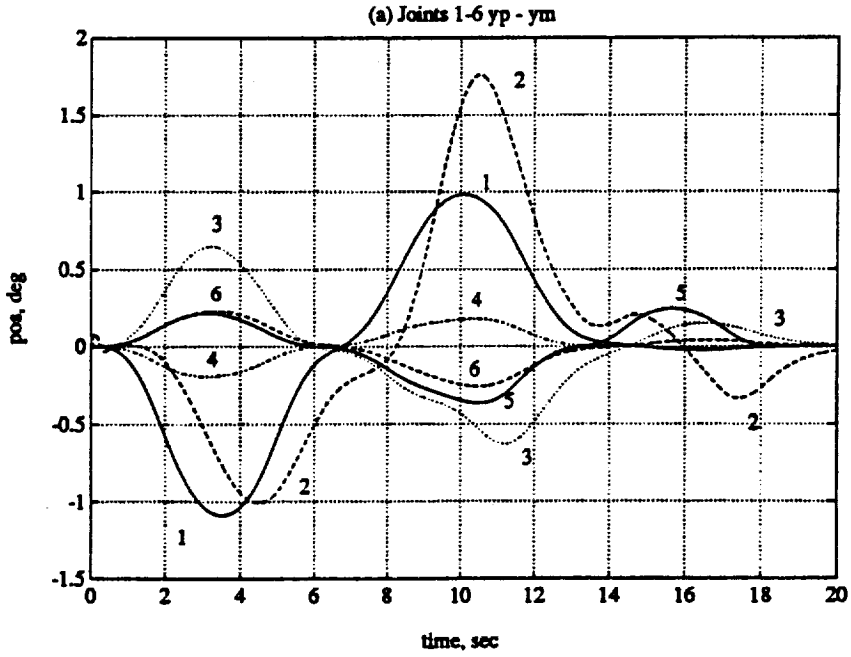

Figure 5.2: Model Following Errors for First Trajectory

C-2

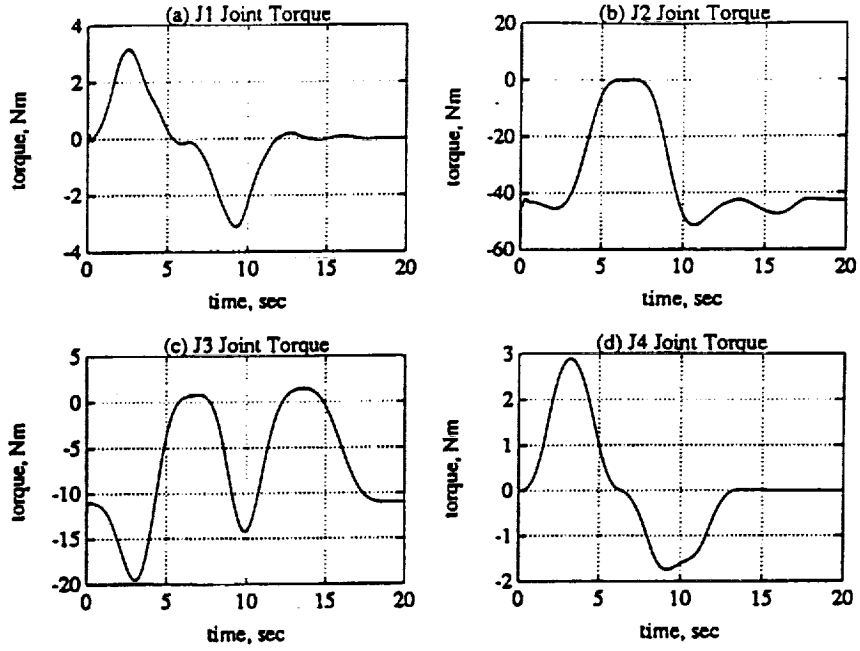

Figure 5.3: Torque Signals for Joints 1-4 for First Trajectory. (a) Joint 1. (b) Joint 2. (c) Joint 3. (d) Joint 4.

Table 5.4: Second Six Joint Tracking Test Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	90	-
1*	45	45	45	90	0	0	5
2*	0	-45	180	0	45	90	5

5.1.2 Tracking Trajectory #2

The second six joint tracking trajectory is shown in Table 5.4 and is illustrated in Figure 5.4. The model following error trajectories are shown in Figure 5.5. The peak errors are listed in Table 5.5. Joints 2 and 3 have the worst tracking performance with peak errors of 1.8 and 1.6 degrees. Joints 1, 4, 5, and 6 all have error trajectories within $\pm$ 1.0 degree.

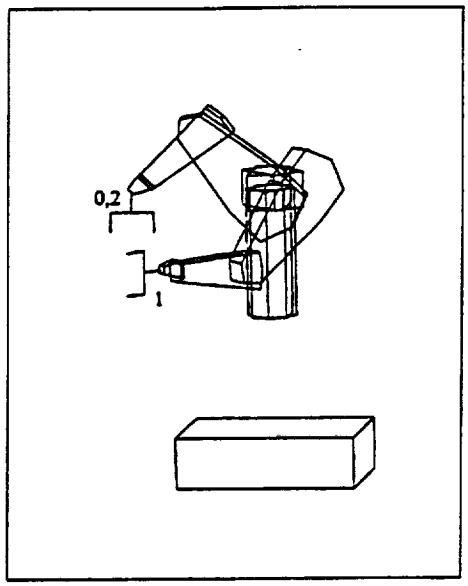

Figure 5.4: Second Six Joint Tracking Test Trajectory

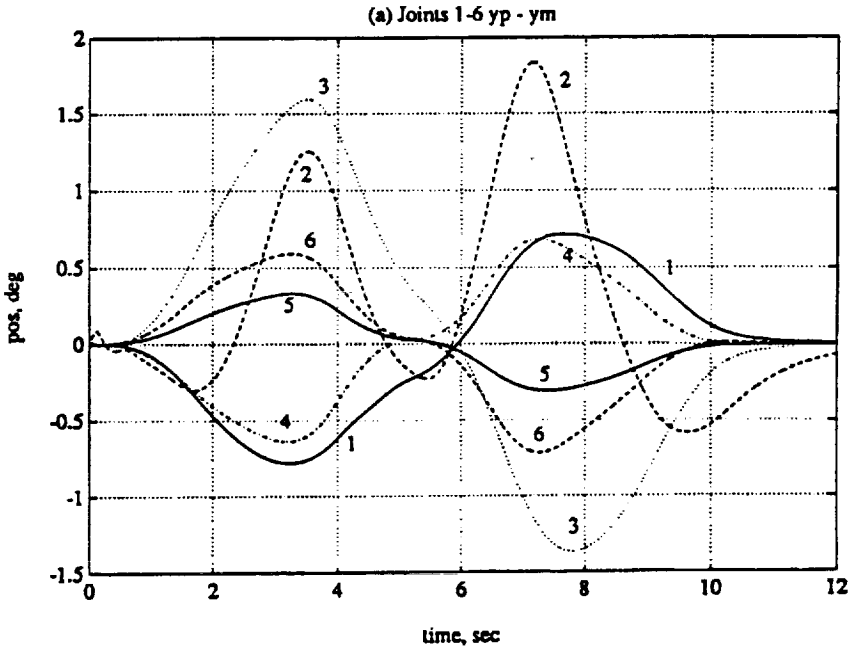

Figure 5.5: Model Following Errors for Second Trajectory

Table 5.5: Peak Errors for Second Trajectory

Joint	Peak Error (deg)
1	-0.7793
2	1.8370
3	1.5970
4	0.6837
5	0.327 6
6	-0.7096

Table 5.6: Third Six Joint Tracking Test Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	90	-
1	0	-90	180	90	90	0	2
2	90	-90	180	90	90	90	2
3*	90	0	90	0	90	90	10
4*	90	0	90	0	90	90	1
5*	90	-90	180	90	90	90	10
6	0	-90	180	90	90	0	2
7	0	-45	180	0	45	90	2

5.1.3 Tracking Trajectory #3

The third six joint tracking trajectory is shown in Table 5.6 and is illustrated in Figure 5.6. This trajectory was used to simulate gross and fine robot motion. Gross motion is fast motion through free space where speed is required but tight tracking errors are not needed. Fine motion is slow motion where tracking errors should be minimized. For the third trajectory, the arm moves up using a 2.0 second trajectory and twists 90 degrees at Joint 1 again using a 2.0 second trajectory. These fast trajectories were used to simulate gross motion. Once into position, the arm slowly straightens out with a 10 second trajectory, simulating fine motion, and holds the extended position for 1.0 second. The procedure is then repeated in reverse to get the arm back to the shutdown position.

The actual joint positions and desired positions (reference model outputs) are

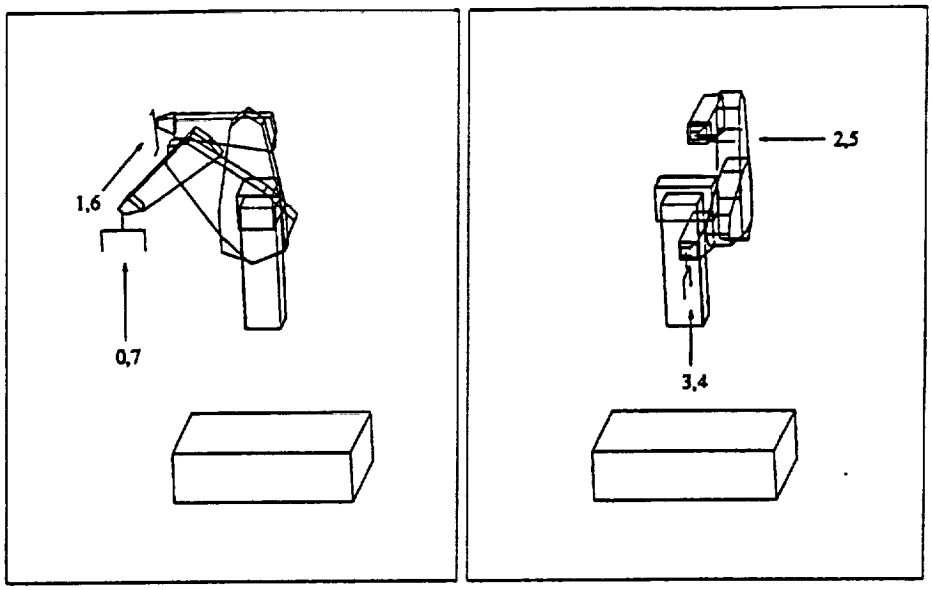

Figure 5.6: Third Six Joint Tracking Test Trajectory

shown in Figures 5.7 and 5.8. The model following errors are shown in Figures 5.9 and 5.10. For the fine motion trajectory segments, all of the model following errors were $< \pm 0.38$ degrees. The peak errors for the gross motion segments ranged from 0.4 degrees for Joint 3 to 2.5 degrees for Joint 1.

Notice that Joint 6 went unstable and saturated at its limits for a small time before and after the fine motion segments, Figure 5.10(d). This oscillation can be seen in the error plot for Joint 6, Figure 5.10(c). Even though Joint 6 went unstable for a time, the rest of the joints were not greatly affected.

5.2 Effects of DMRAC Parameter Changes

This section will examine the effects of changing the various tuning parameters of the DMRAC algorithm. In order to be able to compare the effects of the parameter changes, we will look at the the model following error, $(y_p - y_m)$. *The bias term, q_{bias}, will be subtracted out when displaying the plots so the error term will approach zero for easy comparison.* Also, we will only present results for Joint 2 or 3. These

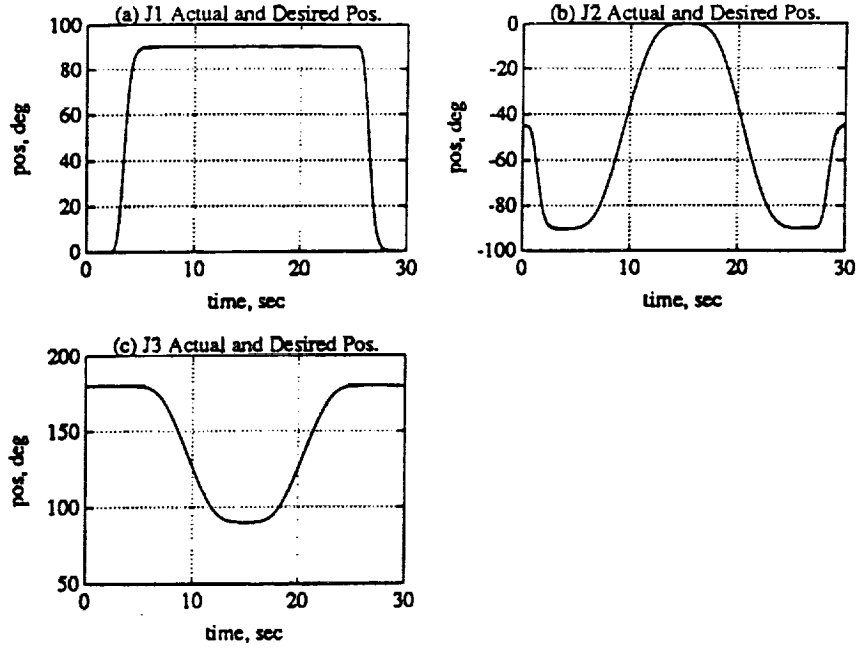

Figure 5.7: Actual and Desired (y_m) Joints 1-3 Positions for Third Case. (a) Joint 1. (b) Joint 2. (c) Joint 3.

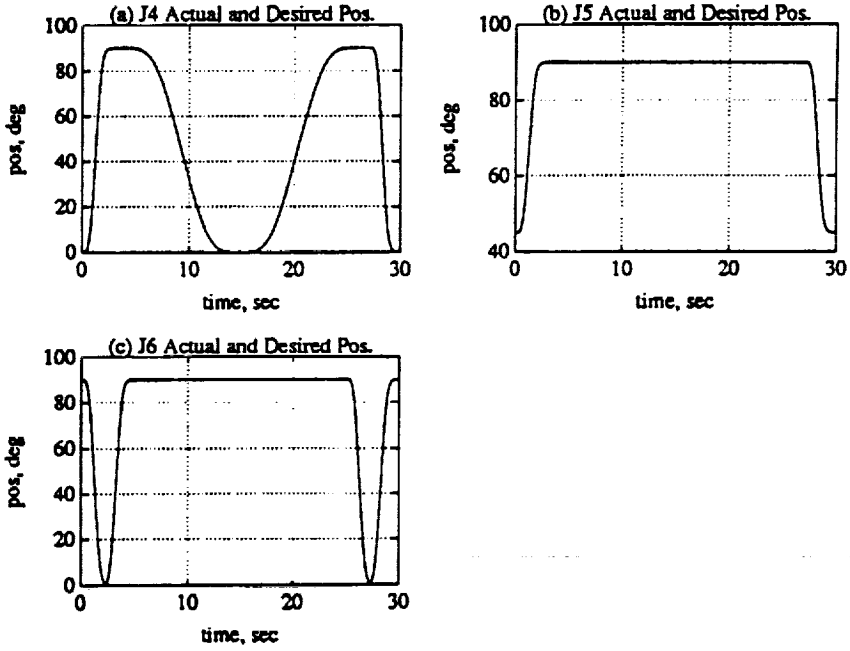

Figure 5.8: Actual and Desired (y_m) Joints 4-6 Positions for Third Case. (a) Joint 4. (b) Joint 5. (c) Joint 6.

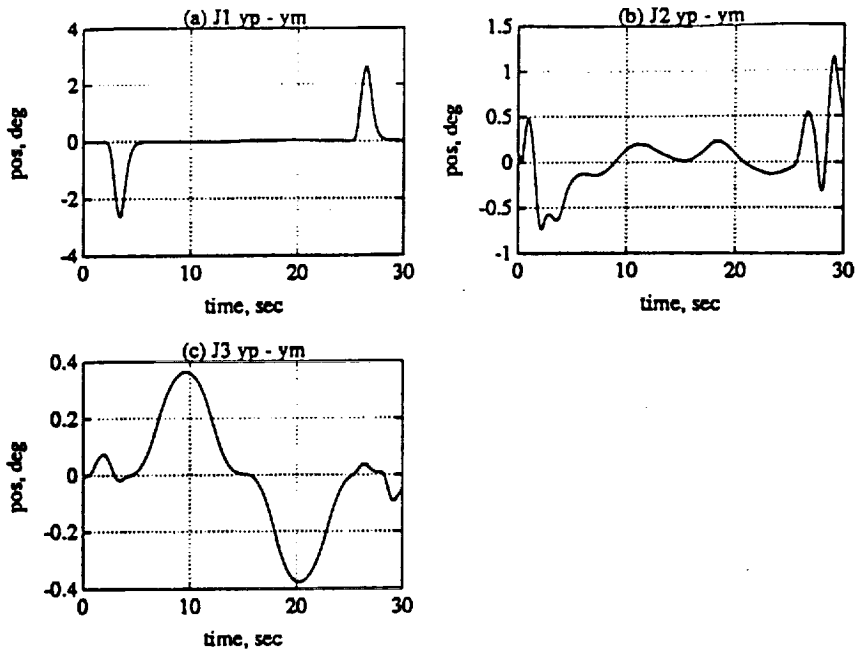

Figure 5.9: Model Following Error for Joints 1-3 for Third Case. (a) Joint 1. (b) Joint 2. (c) Joint 3.

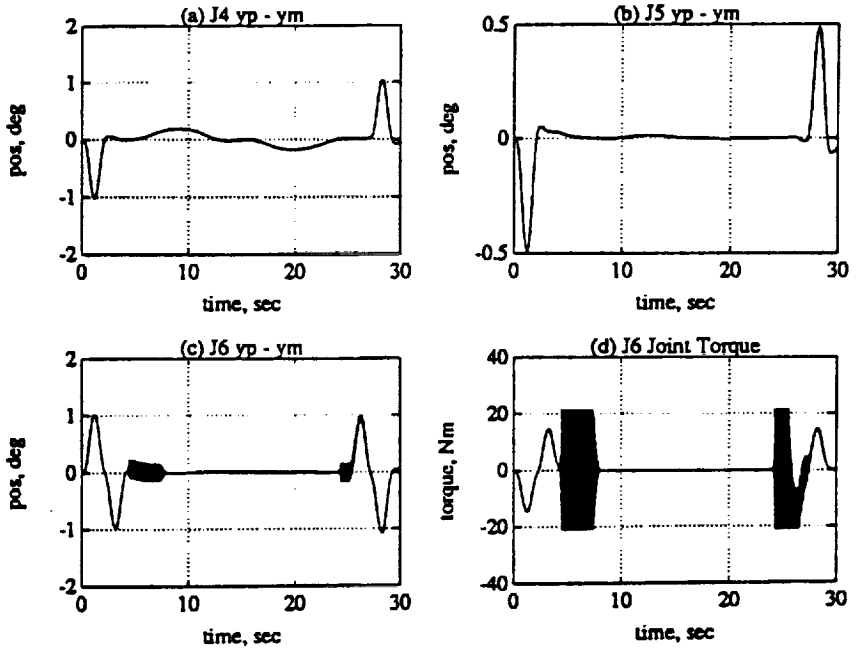

Figure 5.10: Model Following Error for Joints 4-6 for Third Case. (a) Joint 4. (b) Joint 5. (c) Joint 6. (d) Joint 6 Torque

Table 5.7: Base Parameter Values for Parameter Change Runs

T_{pro} (diag component)	"e_z"	20	40	22	0.2	0.2	0.2
	"x_m"	140	20	140	35	100	22
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	110	1.4	1.4	1.4
T_{int} (diag component)	"e_z"	20	60	25	0.2	0.2	0.2
	"x_m"	140	20	150	35	140	25
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	130	1.4	1.4	1.4
Joint		1	2	3	4	5	6
Model	w_n	4	4	4	7	7	7
	ζ	1	1	1	1	1	1
Feed Forward	K_d	6	6	6	6	6	6
	τ	0.1	0.1	0.1	0.1	0.1	0.1
alpha	α	0.035	0.02	0.02	0.01	0.01	0.01

two joints usually have the worst performance out of the six. The initial values for the tuning parameters, which will serve as a base for the comparisons, are shown in Table 5.7. The trajectory was a 10 degree step input from the shutdown position, Figure 3.4, at t=2.0 seconds. Thus the robot was commanded to move from position $\{0, -45, 180, 0, 45, 90\}$ to $\{10, -35, 190, 10.55, 100\}$ in zero time. This trajectory will show the importance of the reference model when dealing with set point control (step inputs). The reference model provides a controlled, predictable smoothing of the step input such that the robot can follow the reference model output.

5.2.1 Base Case for Comparison

The response to the base case for comparisons is shown in Figure 5.11. The tuning parameters are listed in Table 5.7 which include the feed-forward filter in the plant and model. The initial error for the first 0.5 seconds results from the fact that the start up values for the adaptation gains were not exact, and for the first

interval, there is no control[1] (the torques are zero). As the figure shows, after about 3 seconds, the error has settled back to zero.

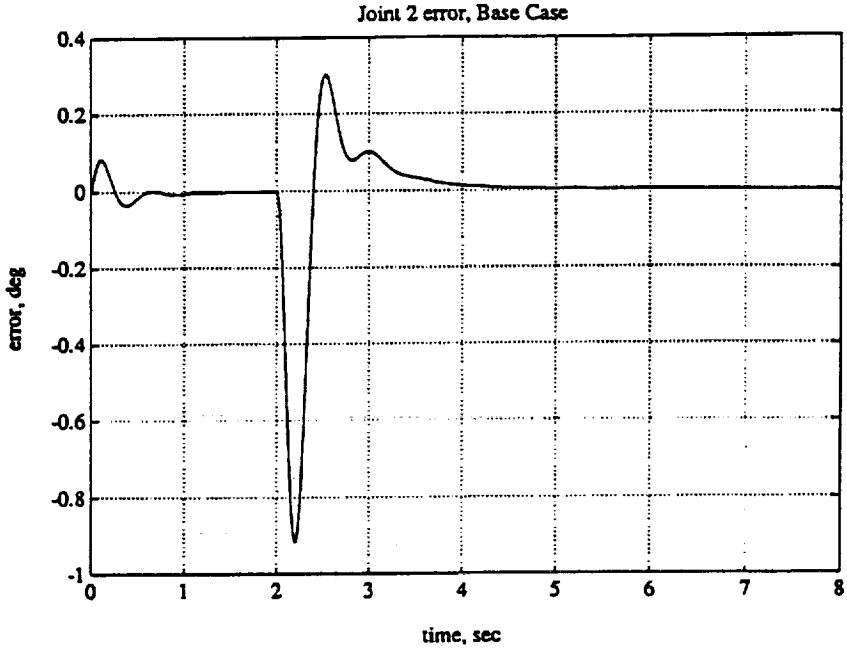

Figure 5.11: Base Case for Parameter Change Comparisons

5.2.2 Adaptive Weighting Matrices, T_{pro} and T_{int}

The weighting matrices T_{pro} and T_{int} affect the adaptive gains through (2.68) and (2.69). If the weighting matrices are taken to be diagonal, then by multiplying out (2.68) and (2.69), we find that the weighting matrices multiply six different products to form $\dot{K}_I$ and $\dot{K}_P$ as shown in Table 5.8. The effects of the weights on the six products will be investigated for Joint 2.

The e_z^2 product term for Joint 2 is weighted by the $T_{pro(2,2)}$ proportional weight and the $T_{int(2,2)}$ integral weight. The effects of changing $T_{pro(2,2)}$ are shown in Figure 5.12 where the dotted line is the base case with $T_{pro(2,2)} = 40$, the solid line

[1]This is a factor in the implementation of the controller on the CIRSSE Testbed and was duplicated in the simulation.

Table 5.8: Effects of Weighting Matrices on Adaptive Gains

Adaptive Gain	Product Term	Portion of Weighting Matrix
K_P	e_z^2	$diag_{1-6}(T_{pro})$
	$e_z x_m$	$diag_{7-18}(T_{pro})$
	$e_z u_m$	$diag_{19-24}(T_{pro})$
K_I	e_z^2	$diag_{1-6}(T_{int})$
	$e_z x_m$	$diag_{7-18}(T_{int})$
	$e_z u_m$	$diag_{19-24}(T_{int})$

*Where $diag_{i-j}(X)$ refers to the i^{th} through the j^{th} diagonal components of X.

is with $T_{pro(2,2)} = 4000$, and the dashed line (on top of the dotted line) is with $T_{pro(2,2)} = 0.4$. As the plot shows, decreasing the weight by a factor of 100 has little effect but increasing it by a factor of 100 increases the error. The effects of changing $T_{int(2,2)}$ are shown in Figure 5.13 where the dotted line is the base case with $T_{int(2,2)} = 60$, the solid line is with $T_{int(2.2)} = 6000$, and the dashed line (on top of the dotted line) is with $T_{int(2.2)} = 0.6$. As with the previous case, decreasing the weight has little effect, but increasing the weight increases the error signal. It was discovered that the $diag_{1-6}(T_{pro})$ and $diag_{1-6}(T_{int})$ components were not as effective as the other terms in the weighting matrices for fine tuning the performance of the DMRAC because the e_z^2 product which they multiply is small when the plant is tracking the model with a small error. The $diag_{1-6}(T_{pro})$ and $diag_{1-6}(T_{int})$ components have more effect when there are large errors.

The $e_z x_m$ product term for Joint 2 is multiplied by the $diag_{9-10}(T_{pro})$ proportional weighting terms and the $diag_{9-10}(T_{int})$ integral weighting terms. The effects of changing $diag_{9-10}(T_{pro})$ are shown in Figure 5.14 where the dotted line is the base case with $diag_{9-10}(T_{pro}) = \{140, 35\}$, the solid line is with $diag_{9-10}(T_{pro}) = \{1400, 350\}$, and the dashed line is with $diag_{9-10}(T_{pro}) = \{14, 3.5\}$. Decreasing $diag_{9-10}(T_{pro})$ resulted in a larger error signal with more oscillations while increasing $diag_{9-10}(T_{pro})$ resulted in a slightly larger error and a longer decay time. The

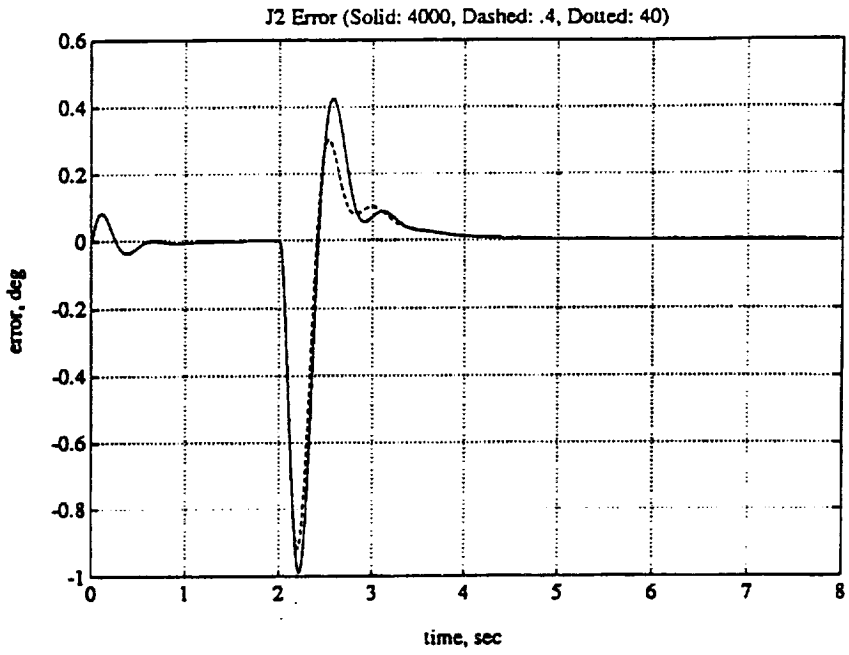

Figure 5.12: Effects of $T_{pro(2,2)}$ on Joint 2. (dotted, under dashed) $T_{pro(2,2)} = 40$, (solid) $T_{pro(2,2)} = 4000$, (dashed) $T_{pro(2,2)} = 0.4$

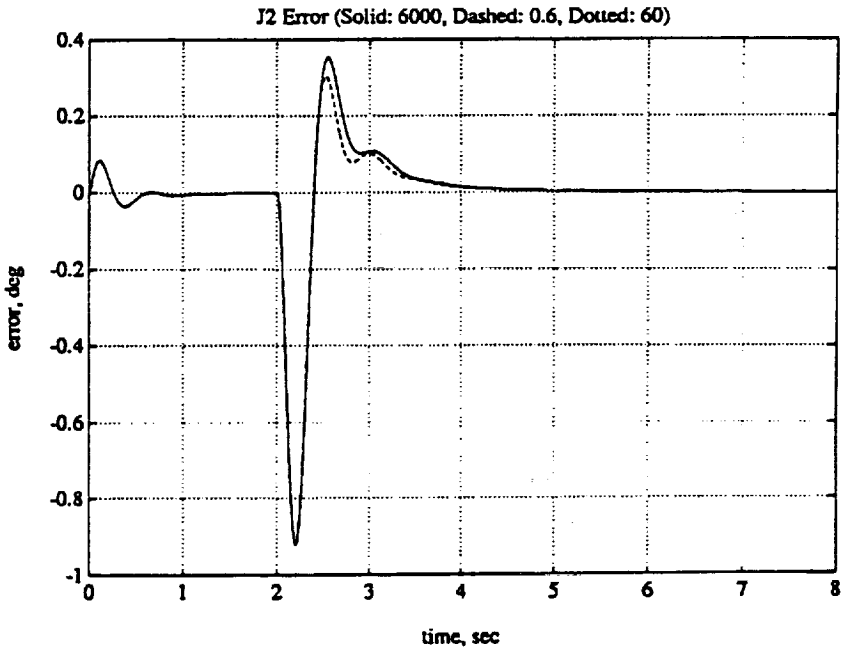

Figure 5.13: Effects of $T_{int(2,2)}$ on Joint 2. (dotted, under dashed) $T_{pro(2,2)} = 60$, (solid) $T_{pro(2,2)} = 6000$, (dashed) $T_{pro(2,2)} = 0.6$

effects of changing $diag_{9-10}(T_{int})$ are shown in Figure 5.15 where the dotted line is the base case with $diag_{9-10}(T_{int}) = \{150, 35\}$, the solid line is with $diag_{9-10}(T_{int}) = \{1500, 350\}$, and the dashed line is with $diag_{9-10}(T_{int}) = \{15, 3.5\}$. Increasing $diag_{9-10}(T_{int})$ resulted in a larger peak error with more oscillations but a faster decay rate. Decreasing $diag_{9-10}(T_{int})$ resulted in a slightly smaller peak error and a longer decay rate.

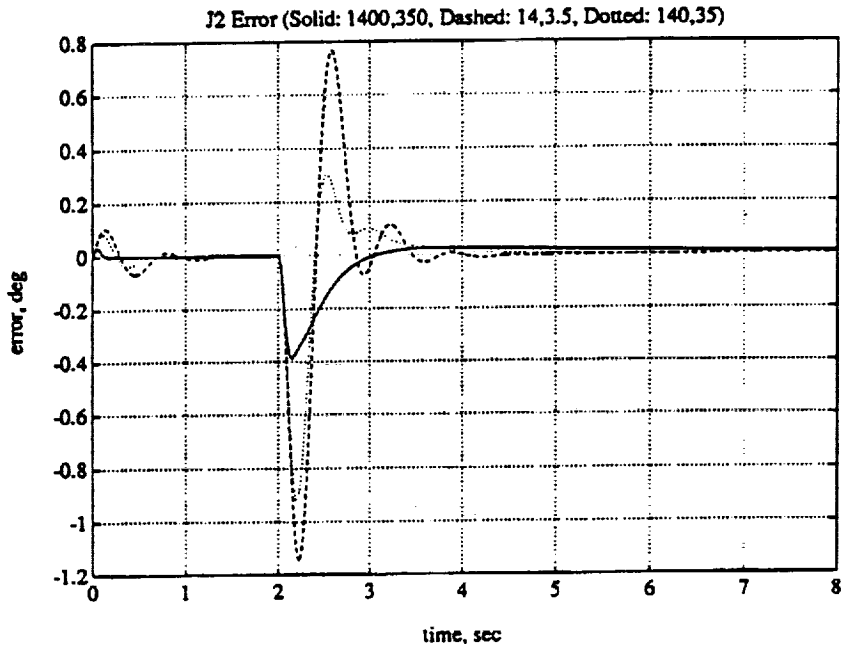

Figure 5.14: Effects of $T_{pro(9,9)}$ and $T_{pro(10,10)}$ on Joint 2. (dotted) $diag_{9-10}(T_{pro}) = \{140, 35\}$, (solid) $diag_{9-10}(T_{pro}) = \{1400, 350\}$, (dashed) $diag_{9-10}(T_{pro}) = \{14, 3.5\}$

The $e_z u_m$ product term for Joint 2 is multiplied by the $T_{pro(20,20)}$ and $T_{int(20,20)}$ weighting terms. The effects of changing $T_{pro(20,20)}$ are shown in Figure 5.16 where the dotted line is the base case with $T_{pro(20,20)} = 160$, the solid line is with $T_{pro(20,20)} = 1600$, and the dashed line is with $T_{pro(20,20)} = 16$. Decreasing $T_{pro(20,20)}$ resulted in a larger error signal with more oscillations while increasing $diag_{9-10}(T_{pro})$ resulted in a slightly larger error and a longer decay time. The effects of changing $T_{int(20,20)}$ are shown in Figure 5.17 where the dotted line is the base case with $T_{int(20,20)} = 160$,

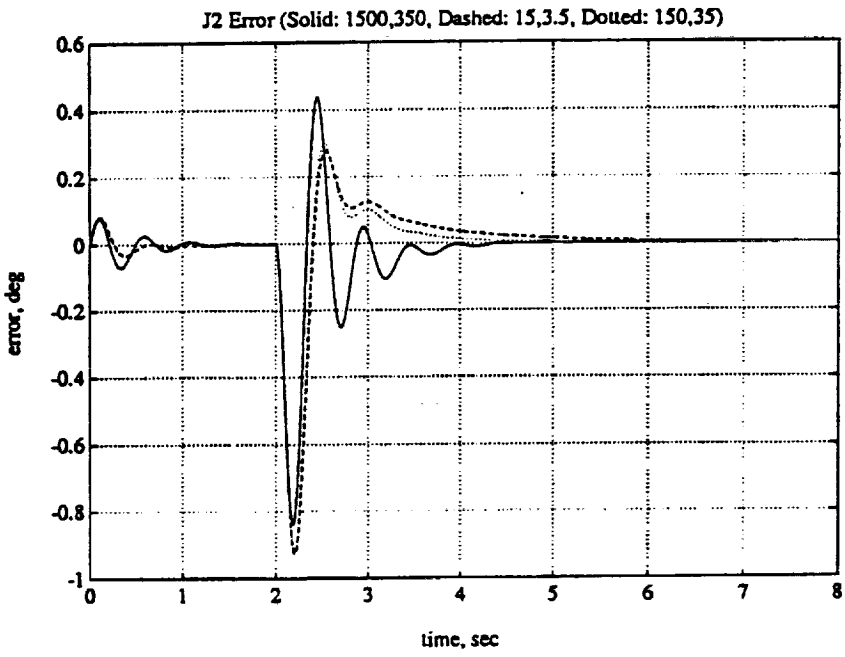

Figure 5.15: Effects of $T_{int(9,9)}$ and $T_{int(10,10)}$ on Joint 2. (dotted) $diag_{9-10}(T_{int}) = \{150, 35\}$, (solid) $diag_{9-10}(T_{int}) = \{1500, 350\}$, (dashed) $diag_{9-10}(T_{int}) = \{15, 3.5\}$

the solid line is with $T_{int(20,20)} = 1600$, and the dashed line is with $T_{int(20,20)} = 16$. Increasing $T_{int(20,20)}$ resulted in a larger peak error with more oscillations but a faster decay rate. Decreasing $T_{int(20,20)}$ resulted in a slightly smaller peak error and a longer decay rate.

It is interesting to note that both $diag_{9-10}(T_{pro})$, $diag_{9-10}(T_{int})$, $T_{pro(20,20)}$, and $T_{int(20,20)}$ had roughly the same effect on the performance.

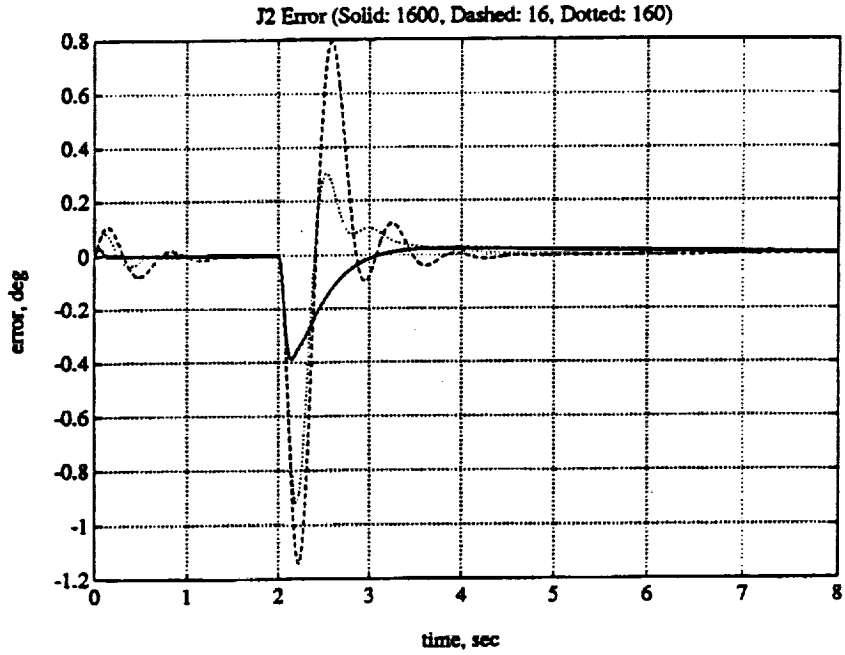

Figure 5.16: Effects of $T_{pro(20,20)}$ on Joint 2. (dotted) $T_{pro(20,20)} = 160$, (solid) $T_{pro(20,20)} = 1600$, (dashed) $T_{pro(20,20)} = 16$

5.2.3 Reference Model, w_n

The effects of changing the reference model dynamics on the response of Joint 3 will be investigated in this section. Joint 3 was selected rather than 2 since it more clearly illustrated the effects of changing the model dynamics. The model damping ratio, ζ, was not changed since we still desire a critically damped model response. The effects of changing w_{n_3} are shown in Figure 5.18 where the dotted

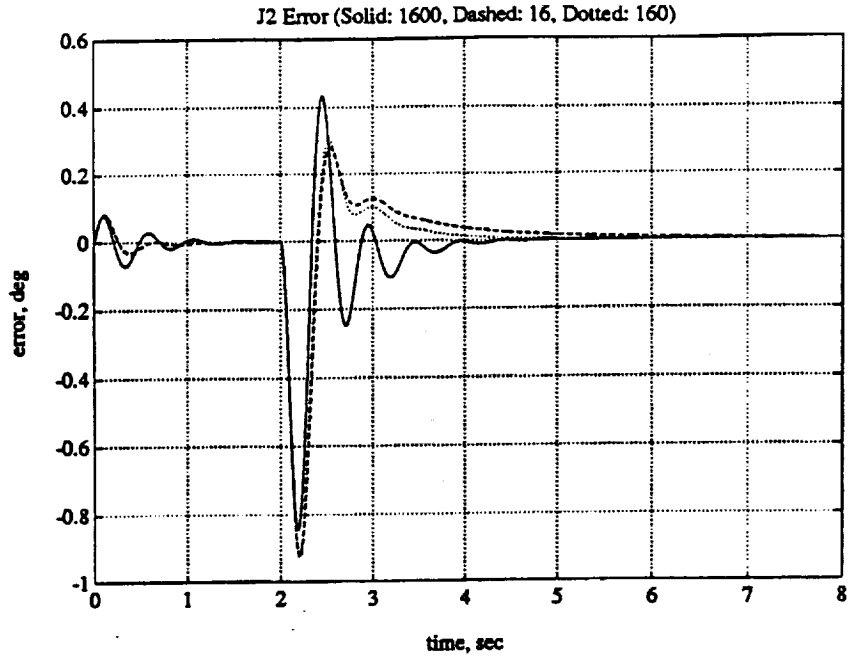

Figure 5.17: Effects of $T_{int(20,20)}$ on Joint 2. (dotted) $T_{int(20,20)} = 160$, (solid) $T_{int(20,20)} = 1600$, (dashed) $T_{int(20,20)} = 16$

line represents the base case with $w_{n_3} = 4$, the solid line is with $w_{n_3} = 8$, and the dashed line is with $w_{n_3} = 2$. Increasing w_{n_3} speeds up the model and produces a faster decay rate but increases the peak error. Decreasing w_{n_3} slows down the model and decreases the peak error but increases the decay rate.

5.2.4 Feed-Forward Filter, K_d and τ

The feed-forward filter added to the plant and reference model $\langle BASIC/FF^2 \rangle$ has two parameters associated with each joint; a gain, K_d, and a time constant, τ. This section will investigate the effects of changes in K_d and τ on the model following error $(y_p - y_m)$ for Joint 2.

The effects of changing K_d for Joint 2 are shown in Figure 5.19 where the dotted line is the base response with $K_d = 6$, the solid line is with $K_d = 12$, and the dashed line is with $K_d = 3$. Increasing K_d resulted in an increased peak error and

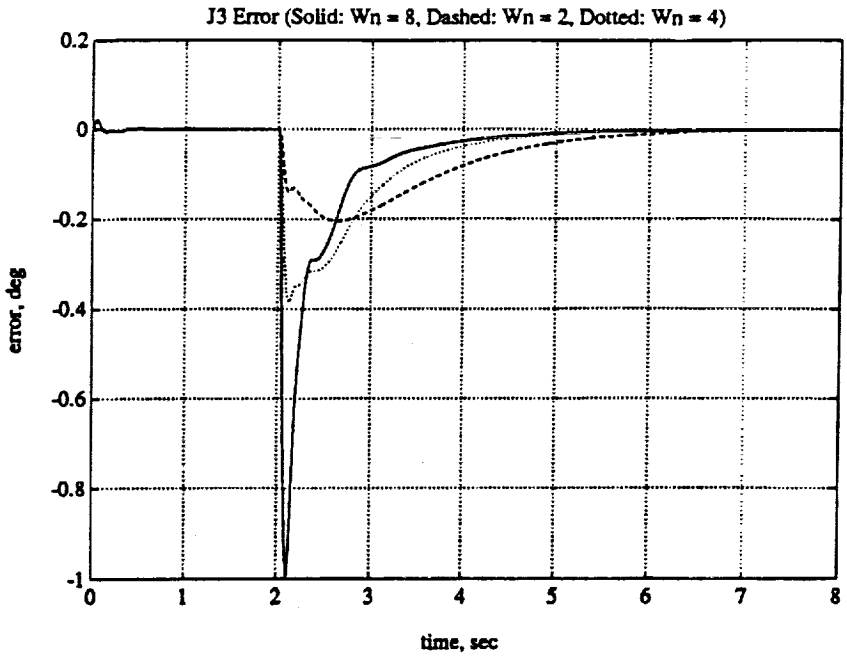

Figure 5.18: Effects of w_{n_3} on Joint 3. (dotted) $w_{n_3} = 4$, (solid) $w_{n_3} = 8$, (dashed) $w_{n_3} = 2$

a longer error decay rate. Decreasing K_d resulted in a lower peak error and a faster error decay rate.

The effects of changing τ for Joint 2 are shown in Figure 5.20 where the dotted line is the base response with $\tau = 0.1$, the solid line is with $\tau = 0.2$, and the dashed line is with $\tau = 0.05$. Increasing τ resulted in a lower peak error while decreasing τ resulted in a larger peak error with more oscillations. As a rule of thumb, τ should not be made smaller than about 3 to 5 times the sample period, T_s, or the discretization of the filter will not accurately reproduce the continuous filter.

5.2.5 Plant Output Derivative Weights, α

This section will investigate the effects of changing the output derivative weights, α, defined in $\langle BASIC/FF^2/\alpha \rangle$, on the performance of Joint 2. Figure 5.21 shows the effects of increasing and decreasing $\alpha_{(2,2)}$ on the Joint 2 error response.

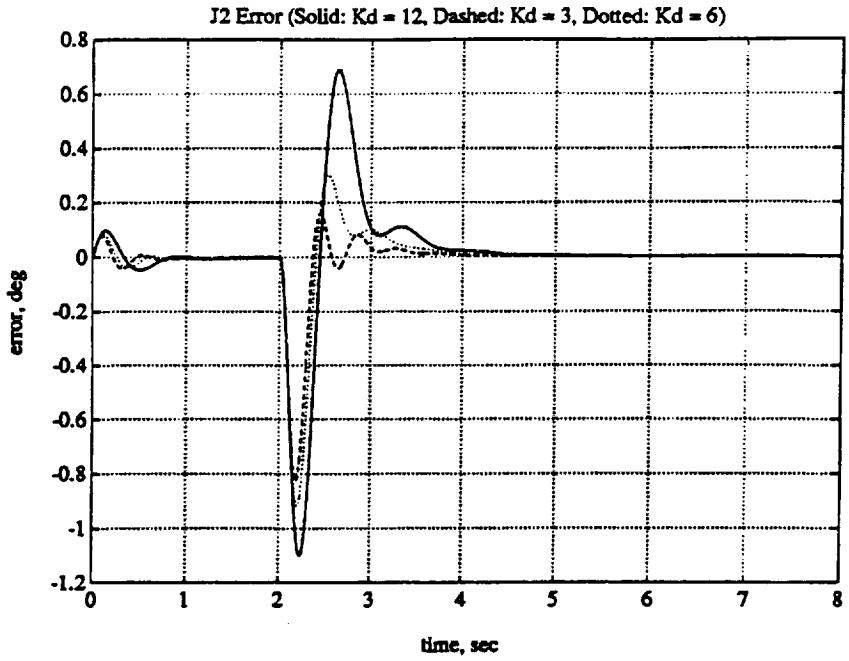

Figure 5.19: Effects of K_d in feed-forward on Joint 2. (dotted) $K_d = 6$, (solid) $K_d = 12$, (dashed) $K_d = 3$

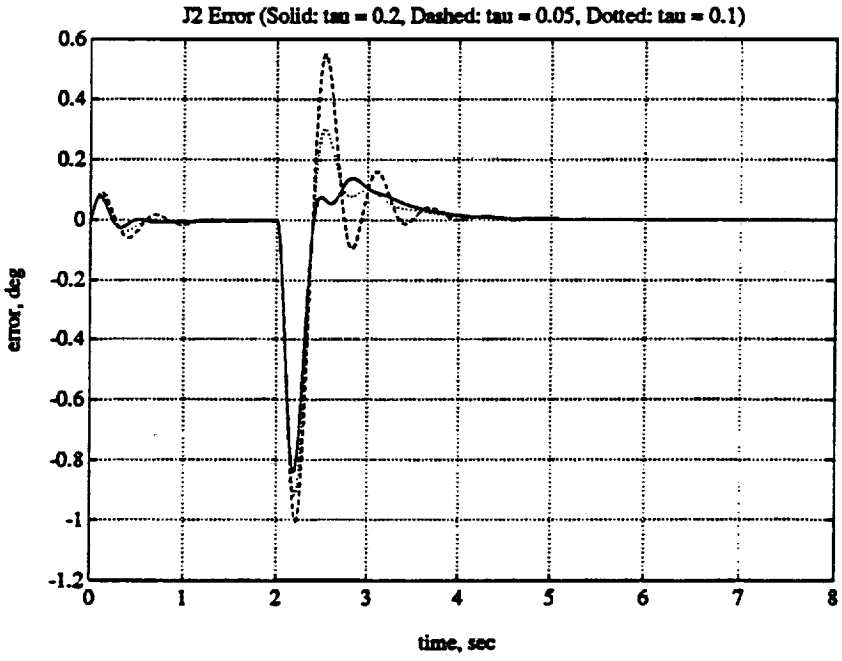

Figure 5.20: Effects of τ in feed-forward on Joint 2. (dotted) $\tau = 0.1$, (solid) $\tau = 0.2$, (dashed) $\tau = 0.05$

The dotted line in Figure 5.21 shows the base response with $\alpha_{(2,2)} = 0.02$, the solid line is with $\alpha_{(2,2)} = 0.04$, and the dashed line is with $\alpha_{(2,2)} = 0.01$. Decreasing α resulted in a slightly lower peak error but increased the oscillations. Increasing α resulted in a larger peak error but removed most of the oscillations.

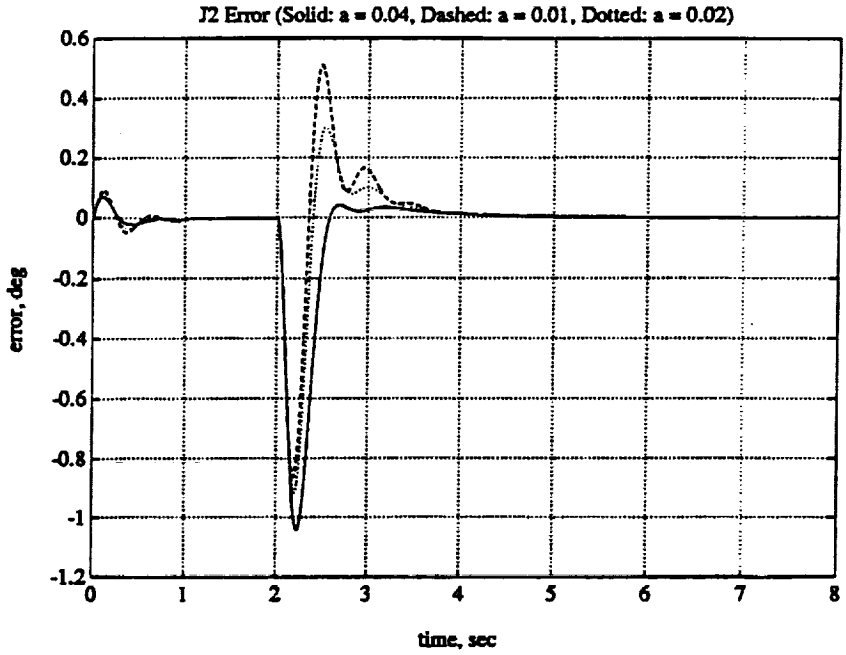

Figure 5.21: Effects of derivative weighting α on Joint 2. (dotted) $\alpha = 0.02$, (solid) $\alpha = 0.04$, (dashed) $\alpha = 0.01$

The effect of removing the output derivative weight for Joint 2 is shown in Figure 5.22. Setting $\alpha_{(2,2)}$ to zero increased the oscillations in the error signal but slightly reduced the peak error.

If was found that increasing the alpha weights beyond a certain point can actually produce unwanted oscillations in the control signals and instability of the system. Figure 5.23 shows the torque signals for Joint 4, 5, and 6 for a trajectory which moves the arm from the shutdown position to an upright position of $\{0, -90, 90, 45, 0, 45\}$ in 3 seconds. The parameter values in Table 5.7 were used with the diagonal component of the α matrix set to $\{0.2, 0.2, 0.2, 0.2, 0.2, 0.2\}$. Joint

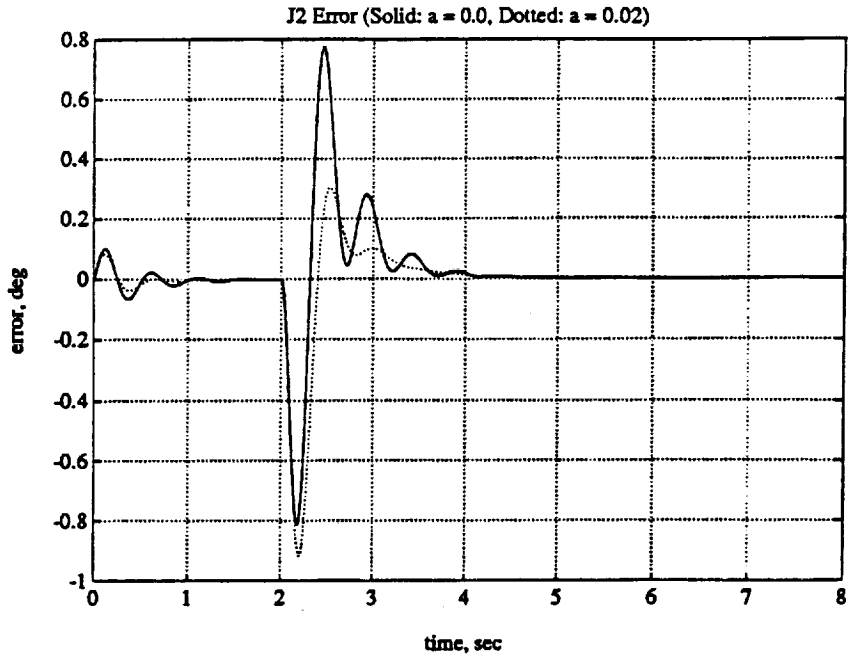

Figure 5.22: Effects of a zero α weight on Joint 2. (dotted) $\alpha = 0.2$, (solid) $\alpha = 0.0$

6 went into oscillation driving Joint 4 into oscillation which then drove Joint 5 into oscillation. If the wrist α terms are lowered to 0.1, then the oscillations can be removed as shown in Figure 5.24.

5.2.6 Removal of Model Feed-Forward Filter

The effects of removing the feed-forward filter from the reference model resulting in algorithm $\langle BASIC/FF/\alpha/bias/disc \rangle$ were investigated. It was found that removal of the feed-forward from the model resulted in an unstable controller. Tuning parameters could not be easily found which stabilized the robot, thus, algorithm $\langle BASIC/FF/\alpha/bias/disc \rangle$ was not investigated further.

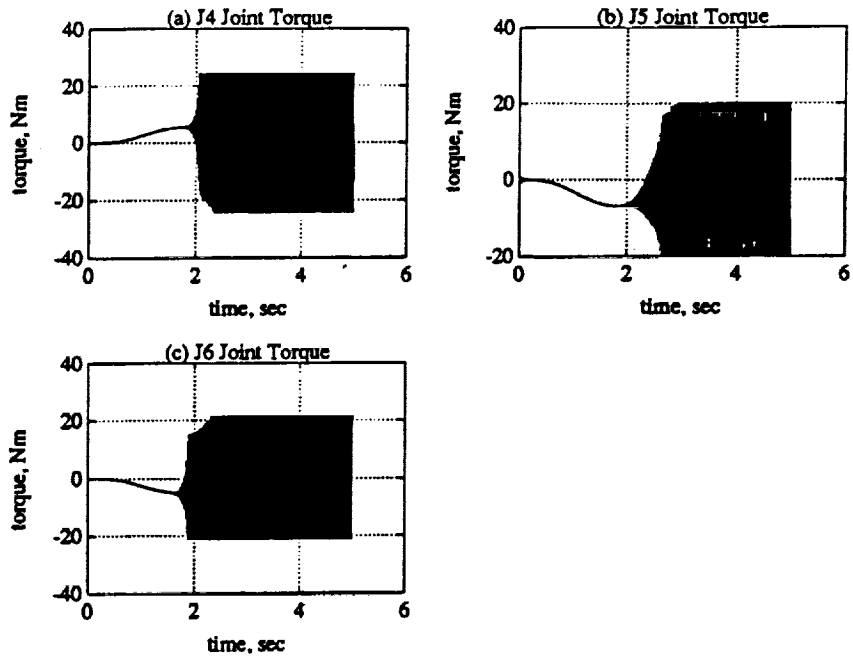

Figure 5.23: Wrist Joint Torques for Instability. (a) Joint 4. (b) Joint 5. (c) Joint 6.

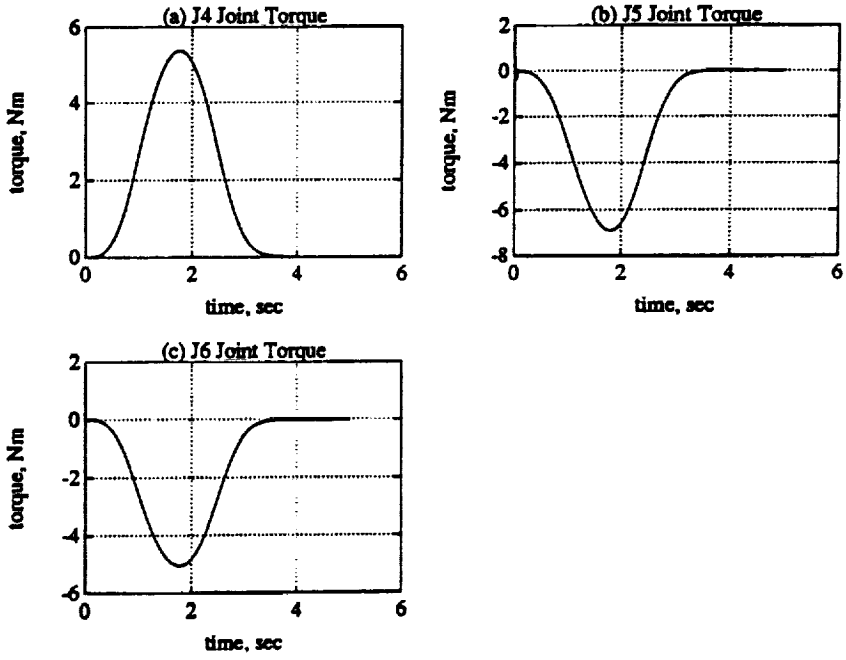

Figure 5.24: Removal of Wrist Instability by Lowering α Weights. (a) Joint 4 Torque. (b) Joint 5 Torque. (c) Joint 6 Torque.

5.2.7 Removal of Model and Plant Feed-Forward Filter

The effects of removing the feed-forward filter entirely from the plant and model resulting in algorithm $(BASIC/\alpha/bias/disc)$ were investigated. Figure 5.25 shows the model following error for Joint 2 with no feed-forward. The dotted line is the base response with feed-forward added to both model and plant, the solid line is with no feed-forward and $\alpha_{(2,2)} = 0.3$, and the dashed line is with no feed-forward and $\alpha_{(2,2)} = 0.2$. The removal of the feed-forward resulted in more oscillations which were lessened but not removed by increasing $\alpha_{(2,2)}$.

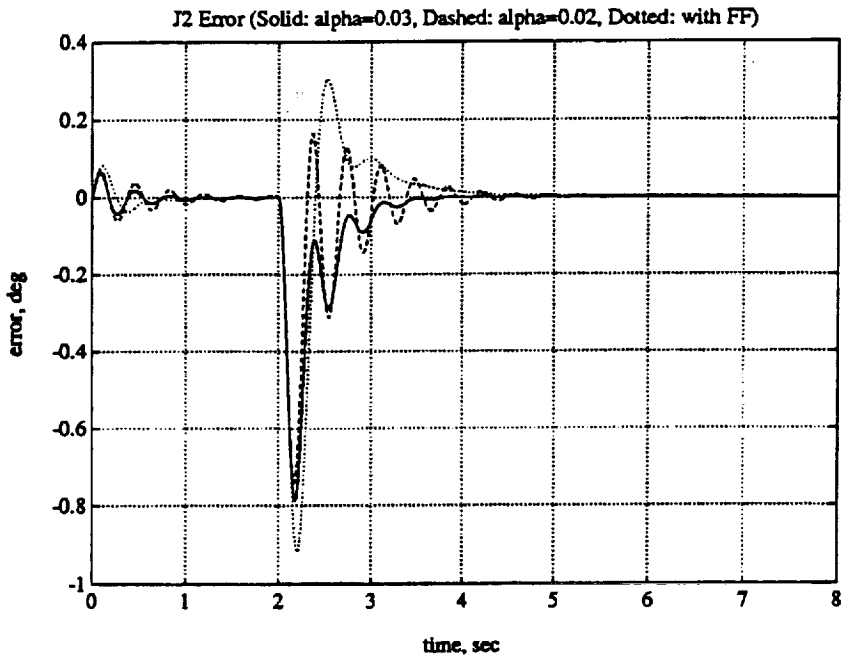

Figure 5.25: Joint 2 error with no Feed Forward. (dotted) Response with FF, (solid) No FF with $\alpha = 0.03$, (dashed) No FF with $\alpha = 0.02$

5.3 Summary

In this chapter we investigated the ability of the DMRAC algorithm to control a PUMA 560 Manipulator such that it followed some typical minimum jerk six joint

trajectories. It was found that the robot had no problems tracking the trajectores with an acceptable error. Next we investigated the effects of changing the tuning parameters on the overall performacnce.

CHAPTER 6
Simulation Results (Load Cases)

This chapter will present the results of the Matlab simulations of Direct Model Reference Adaptive Control of a six link PUMA 560 Manipulator, fully-centralized, in the presence of static and dynamic load changes. All results will be displayed with the bias term, q_{bias} removed (Section 2.7).

6.1 Adaptation to "Static" Payload Variation

In this section, we will investigate the performance of the DMARC algorithm in tracking a trajectory with different loads in the gripper. The algorithm will initially be given some time to adapt to the load and then will be command over a trajectory. Six runs will be performed for each trajectory. The first run will be with no load, and the next five will be with a load of $1kg$, $2kg$, $3kg$, $4kg$, and $5kg$. For reference, the masses of the arm links are listed in Table 6.2. Two different trajectories will be run. The tuning parameters used for the static load tests are shown in Table 6.1.

6.1.1 Trajectory One

The first trajectory for the static load tests is shown in Table 6.3 and is illustrated in Figure 6.1, where the numbers in the figure refer to the knot points in the table. The algorithm is allowed to adapt to the load for 4 seconds and then the arm is extended out to its full reach and swung 45 degrees. At this full extension, the payload mass exerts maximum gravity and inertial loading on the arm.

The errors between plant output and reference model output for Joints 1, 2, 3, 4, 5, and 6 are shown in Figures 6.2 to 6.7, respectively. The figures show the error plots for all six load cases ($0kg$, $1kg$, $2kg$, $3kg$, $4kg$, and $5kg$). A summary of the

Table 6.1: Parameter Values for Static Runs

T_{pro} (diag component)	"e_x"	20	40	22	0.2	0.2	0.2
	"x_m"	140	20	140	35	100	22
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	110	1.4	1.4	1.4
T_{int} (diag component)	"e_x"	20	60	25	0.2	0.2	0.2
	"x_m"	140	20	150	35	140	25
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	130	1.4	1.4	1.4
Joint		1	2	3	4	5	6
Model	w_n	4	4	4	7	7	7
	ζ	1	1	1	1	1	1
Feed Forward	K_d	6	6	6	6	6	6
	τ	0.1	0.1	0.1	0.1	0.1	0.1
alpha	α	0.02	0.02	0.02	0.02	0.02	0.02

Table 6.2: Link Masses

Link	Mass (kg)
1	(link 1 is fixed to the base)
2	17.4
3	4.8
4	0.82
5	0.34
6	0.09

Table 6.3: First Static Load Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	90	-
1	0	-45	180	0	45	90	4
2*	45	0	90	0	90	0	5

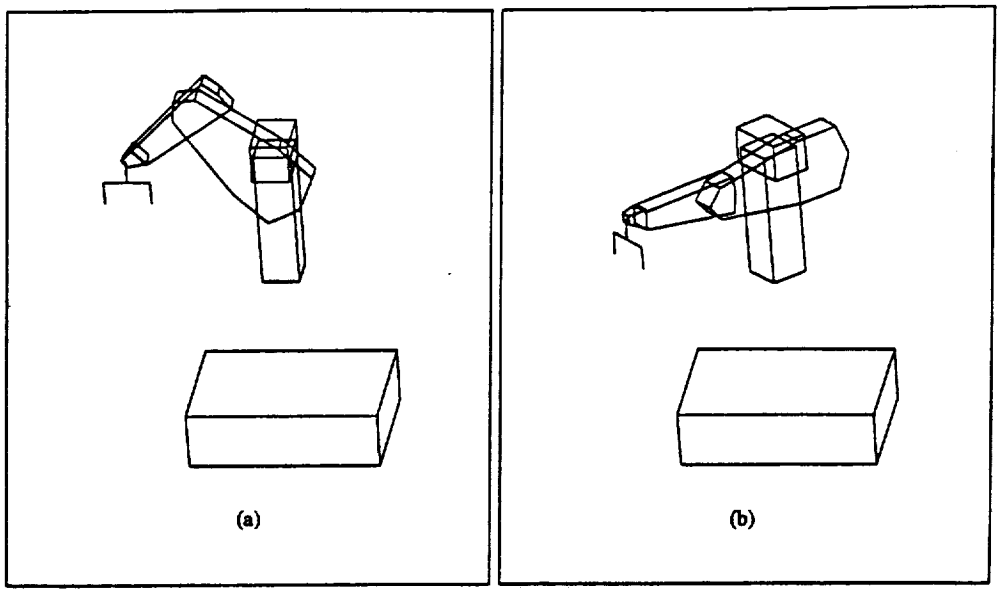

Figure 6.1: First Static Load Trajectory

Table 6.4: First Static Load Trajectory Error Summary

Mass (kg)	Peak Error (degrees)					
	1	2	3	4	5	6
0	-0.4279	0.2808	0.8027	-0.0623	-0.4008	0.8270
1	-0.4266	0.5258	0.8396	-0.0620	-0.4045	0.8244
2	0.4253	0.7709	0.8764	-0.0616	-0.4138	0.8217
3	0.4240	1.0477	0.9133	-0.0616	-0.4174	0.8191
4	0.4213	1.3165	0.9576	-0.0620	-0.4248	0.8164
5	0.4187	1.6169	1.0056	-0.0625	-0.4304	0.8138

peak errors over the trajectory segment 1–2 (see Table 6.3) is given in Table 6.4.

6.1.2 Trajectory Two

The second trajectory for the static load tests is shown in Table 6.5 and is illustrated in Figure 6.8, where the numbers in the figure refer to the knot points in the table. The algorithm is allowed to adapt to the load for 4 seconds and then the arm is extended upward.

The errors between plant output and reference model output for Joints 1, 2, 3,

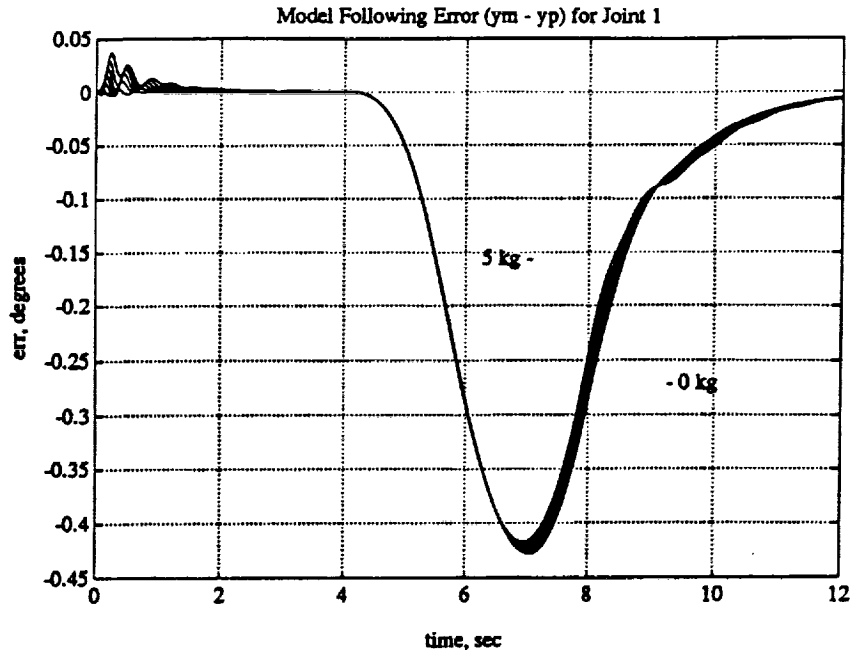

Figure 6.2: Joint 1 Error Plots for First Trajectory (All Loads)

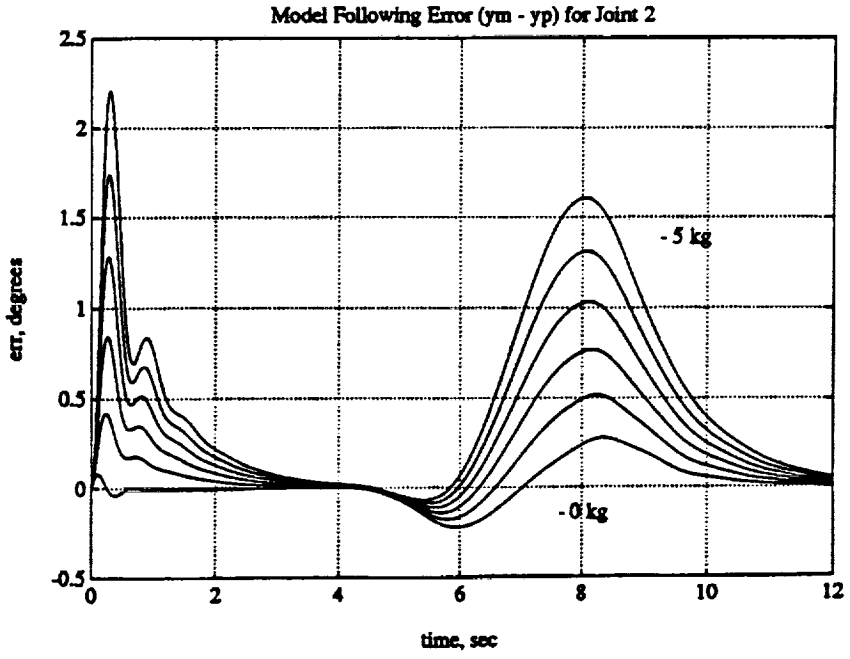

Figure 6.3: Joint 2 Error Plots for First Trajectory (All Loads)

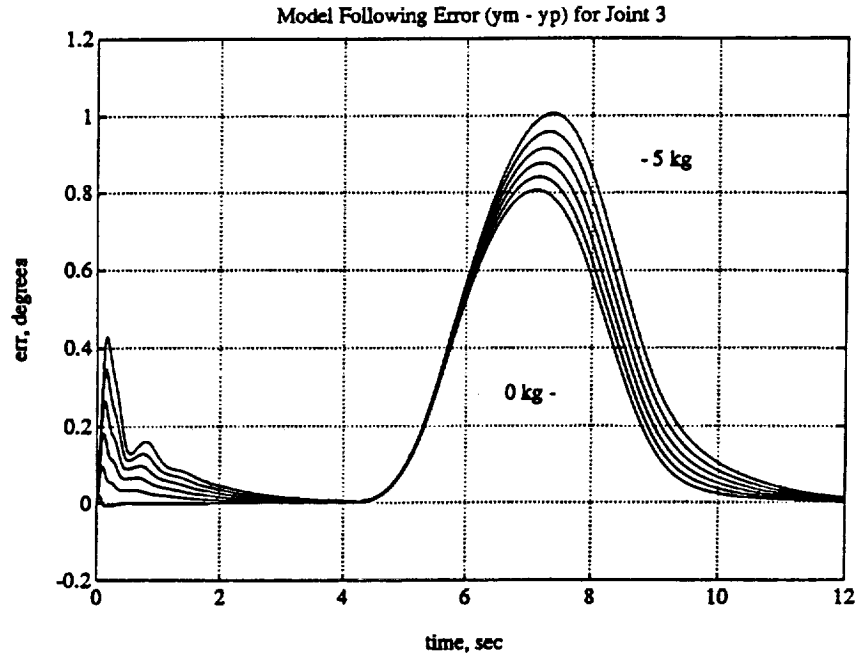

Figure 6.4: Joint 3 Error Plots for First Trajectory (All Loads)

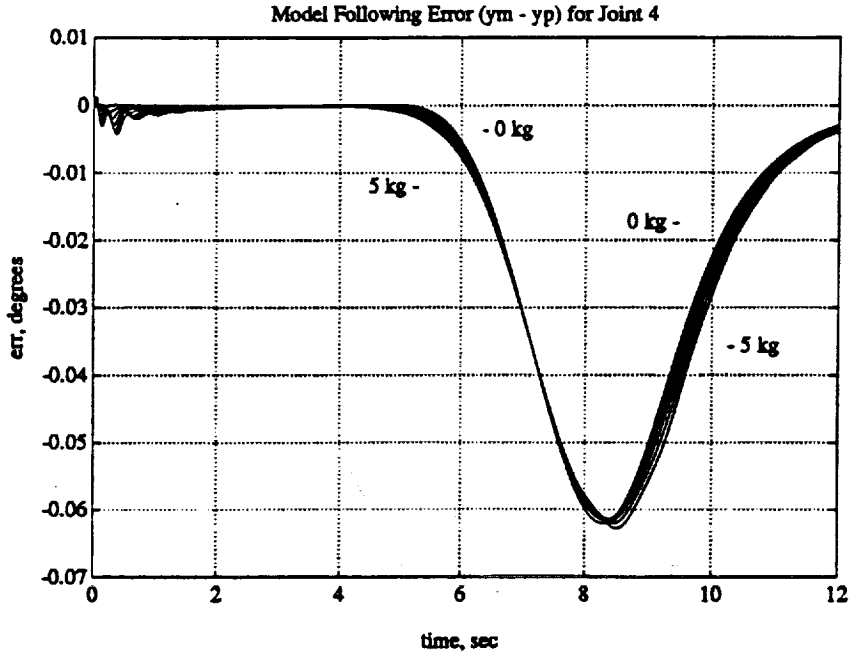

Figure 6.5: Joint 4 Error Plots for First Trajectory (All Loads)

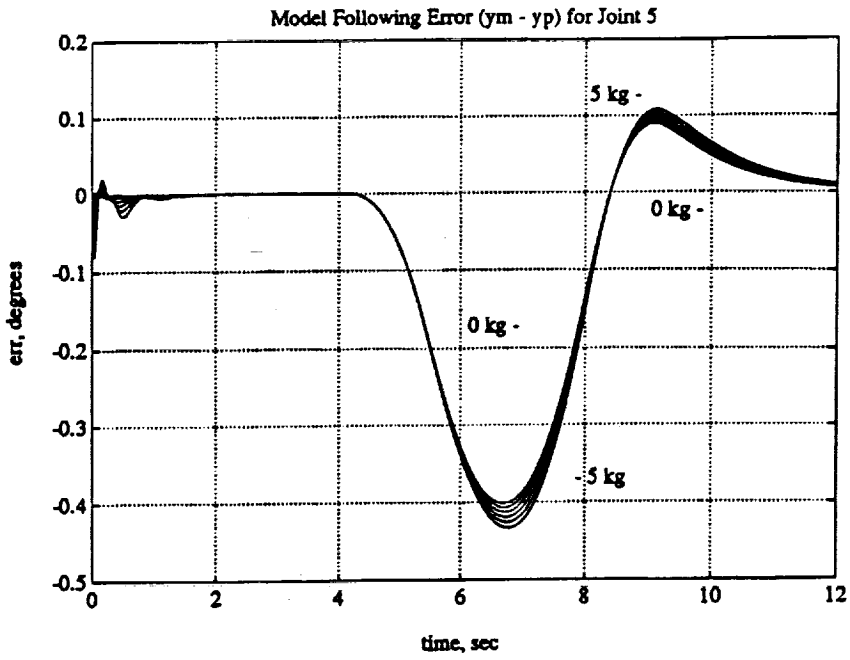

Figure 6.6: Joint 5 Error Plots for First Trajectory (All Loads)

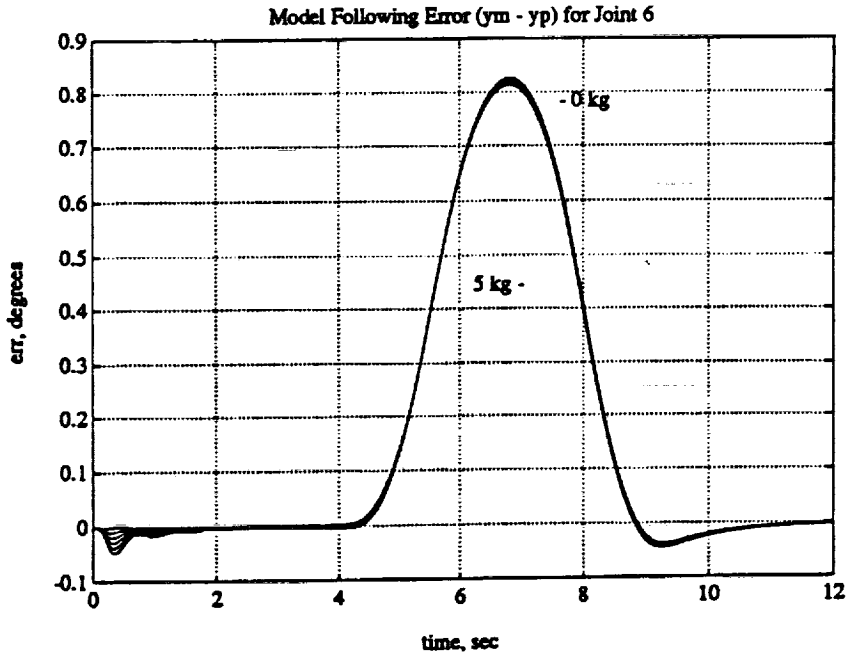

Figure 6.7: Joint 6 Error Plots for First Trajectory (All Loads)

Table 6.5: Second Static Load Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	90	-
1	0	-45	180	0	45	90	4
2*	45	-90	90	45	90	90	5

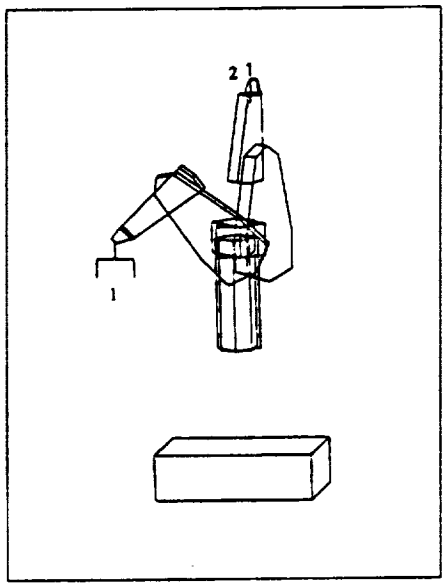

Figure 6.8: Second Static Load Trajectory

Table 6.6: First Static Load Trajectory Error Summary

Joint Number	$0kg$ Peak Error (deg)	$5kg$ Peak Error (deg)
1	-0.6685	-0.6583
2	-1.120	-1.789
3	0.7780	0.8946
4	-0.2325	-0.2809
5	-0.1659	-0.1191
6	0.00769	0.0879

4, 5, and 6 are shown in Figures 6.9 to 6.14, respectively. The figures show the error plots for all six load cases ($0kg$, $1kg$, $2kg$, $3kg$, $4kg$, and $5kg$). Table 6.6 summarizes the $0kg$ and $5kg$ peak errors over the range of interest, ($4 \leq t \leq 10$), for each of the joints.

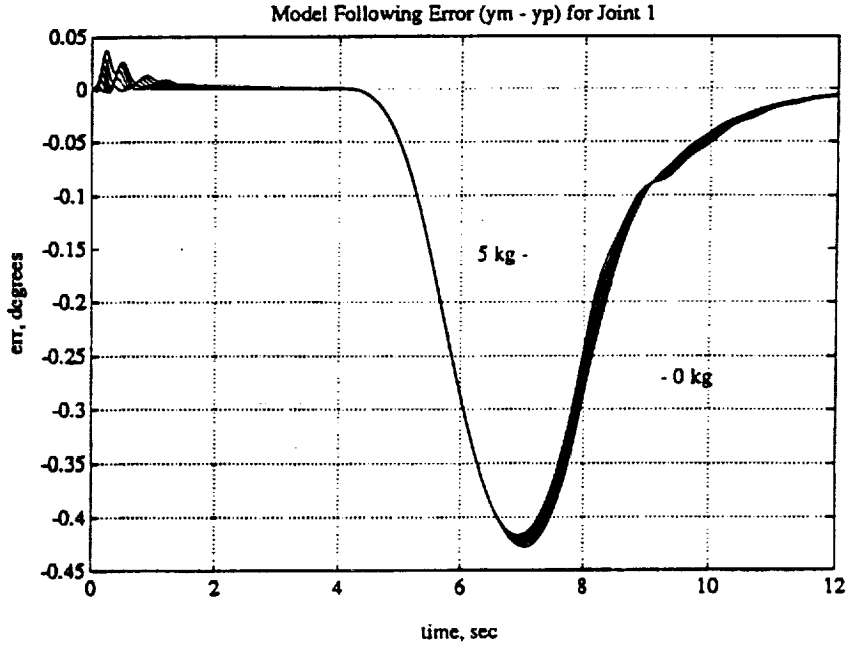

Figure 6.9: Joint 1 Error Plots for Second Trajectory (All Loads)

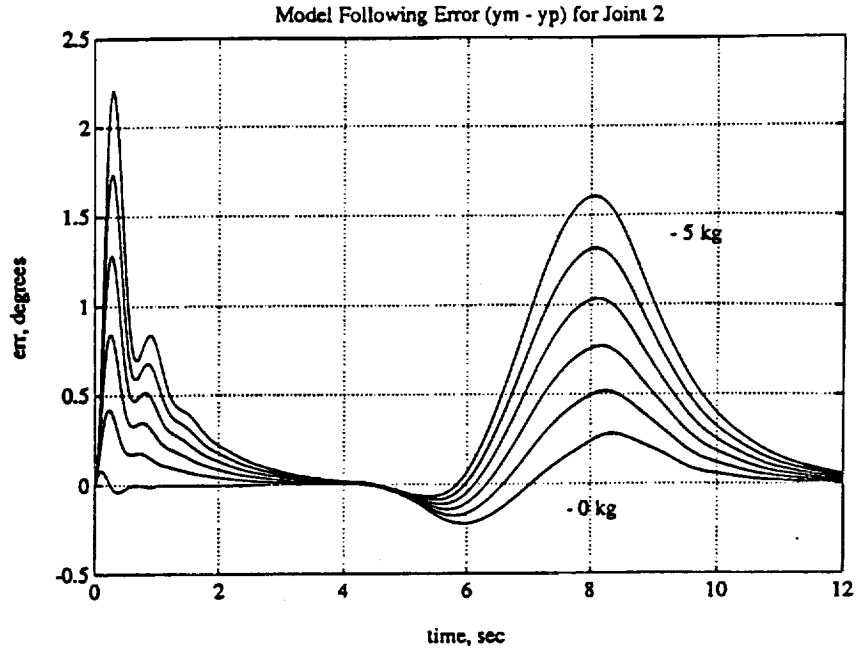

Figure 6.10: Joint 2 Error Plots for Second Trajectory (All Loads)

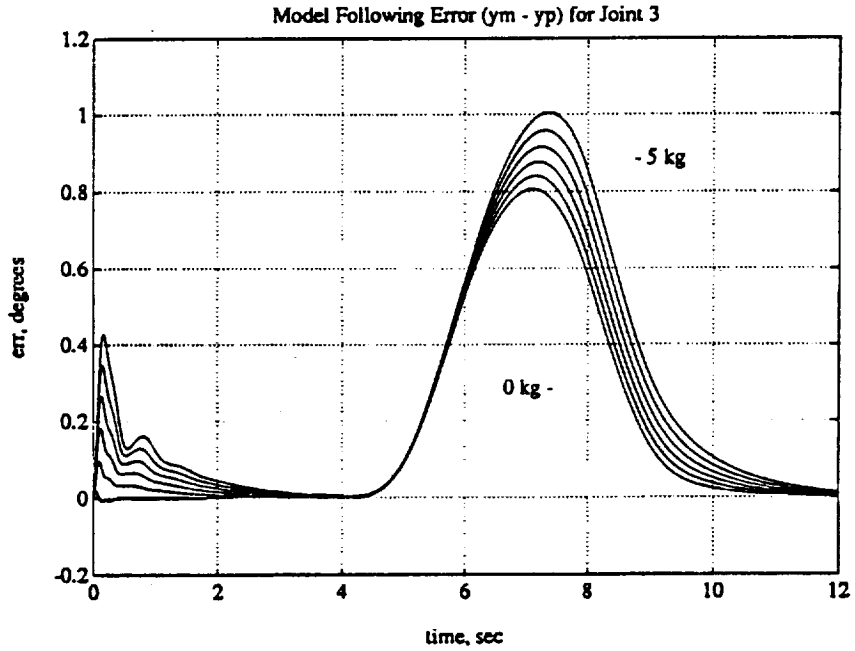

Figure 6.11: Joint 3 Error Plots for Second Trajectory (All Loads)

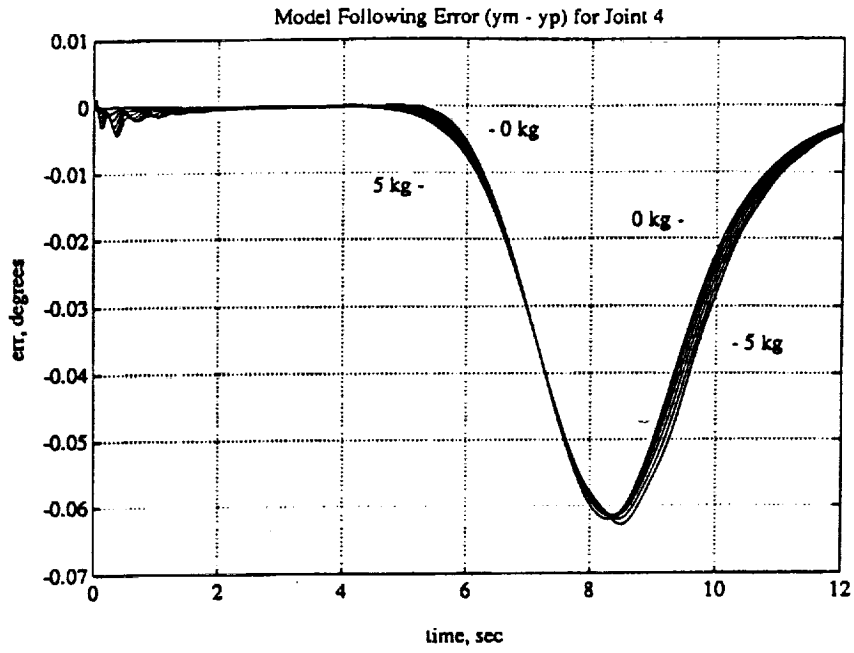

Figure 6.12: Joint 4 Error Plots for Second Trajectory (All Loads)

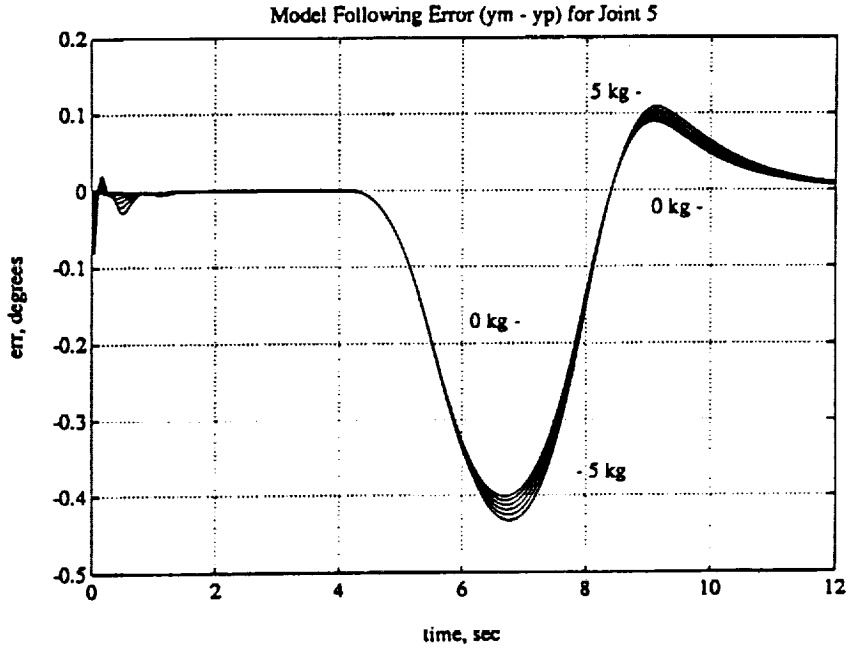

Figure 6.13: Joint 5 Error Plots for Second Trajectory (All Loads)

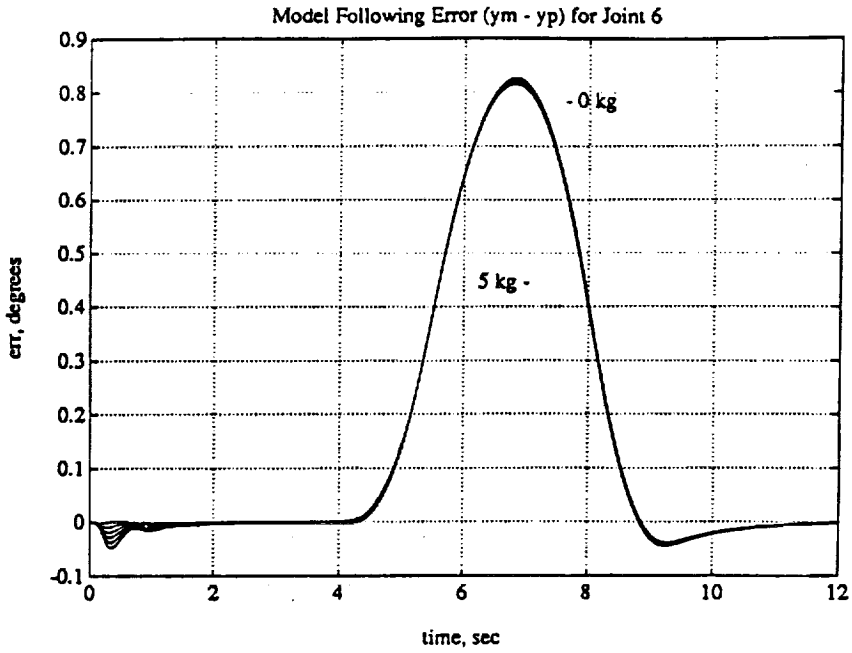

Figure 6.14: Joint 6 Error Plots for Second Trajectory (All Loads)

6.1.3 Summary

By comparison of the $0kg$ error plots to the $5kg$ error plots, it was found that for Joints 1, 3, 4, 5, and 6, a larger portion of the model following error was caused by the adaptation to the changing plant than was caused by the addition of the various loads. Thus the DMRAC algorithm was able to adjust for the different load inertias and gravity loading with acceptable peak errors as compared to the no load case. Joint 2 has the largest gravity loading and must do the most adapting to the changing load mass as Figures 6.3 and 6.10 show.

6.2 Adaptation to "Dynamic" Payload Variation

This section will investigate the effects of suddenly changing the load carried by the arm. The tuning parameters used for these runs are shown in Table 6.7.

Table 6.7: Parameter Values for Dynamic Load Change Runs

T_{pro} (diag component)	"e_z"	20	40	22	0.2	0.2	0.2
	"x_m"	140	20	140	35	100	22
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	110	1.4	1.4	1.4
T_{int} (diag component)	"e_z"	20	60	25	0.2	0.2	0.2
	"x_m"	140	20	150	35	140	25
		1.4	0.2	1.4	0.2	1.4	0.2
	"u_m"	140	160	130	1.4	1.4	1.4
Joint		1	2	3	4	5	6
Model	w_n	4	4	4	7	7	7
	ζ	1	1	1	1	1	1
Feed Forward	K_d	6	6	6	6	6	6
	τ	0.1	0.1	0.1	0.1	0.1	0.1
alpha	α	0.035	0.02	0.02	0.01	0.01	0.01

Table 6.8: Peak Errors for First Dynamic Load Change, $5kg$ Case

Joint	Peak Error (deg)
1	0.032
2	2.25
3	0.43
4	-4.15
5	-0.115
6	-0.0475

6.2.1 First Case

The first case investigated the addition of a load to the robot while it was trying to hold the robot at the shutdown position (see Figure 3.4). Loads of $0kg$, $1kg$, $2kg$, $3kg$, $4kg$, and $5kg$ were added at $t = 2s$. Figures 6.15 to 6.20 show the model input following error, $(y_p - u_m)$, for each of the joints subject to the six load cases. Joints 1, 2, 3, 4, and 6 recovered from the load addition, $(y_p - u_m) \approx 0$, in about 4 seconds. Joint 5 recovered in about $2.5s$. Table 6.8 shows the worst case peak errors for the $5kg$ load addition. Figure 6.21 shows the torque signals for Joints 1, 2, and 3 for the $5kg$ load case.

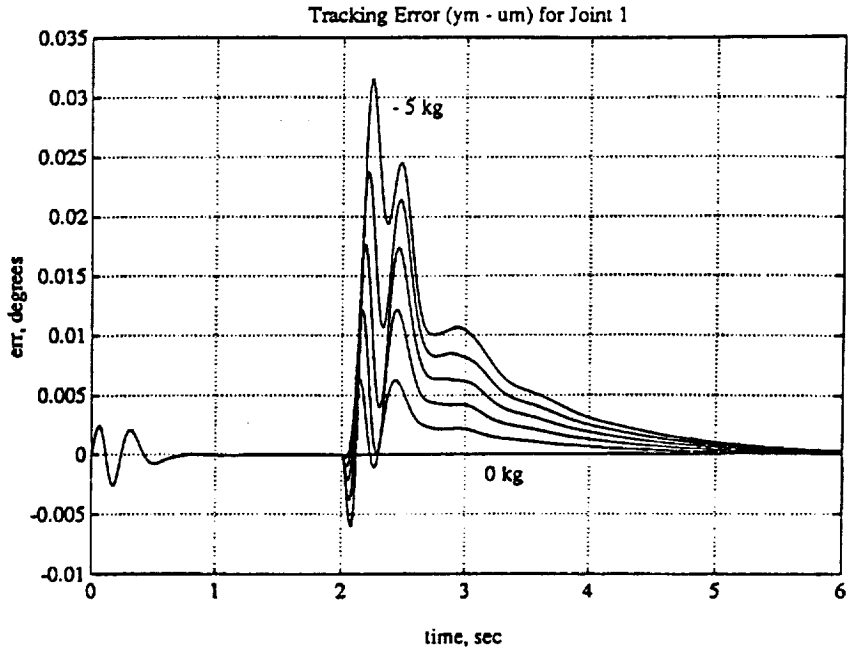

Figure 6.15: Joint 1 Error Plots for Addition of Load at Shutdown Position

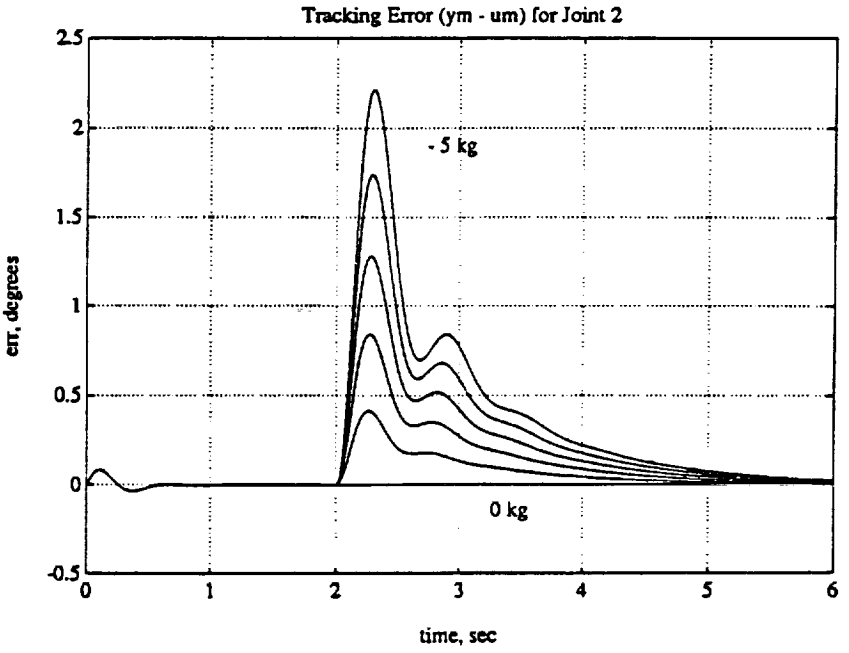

Figure 6.16: Joint 2 Error Plots for Addition of Load at Shutdown Position

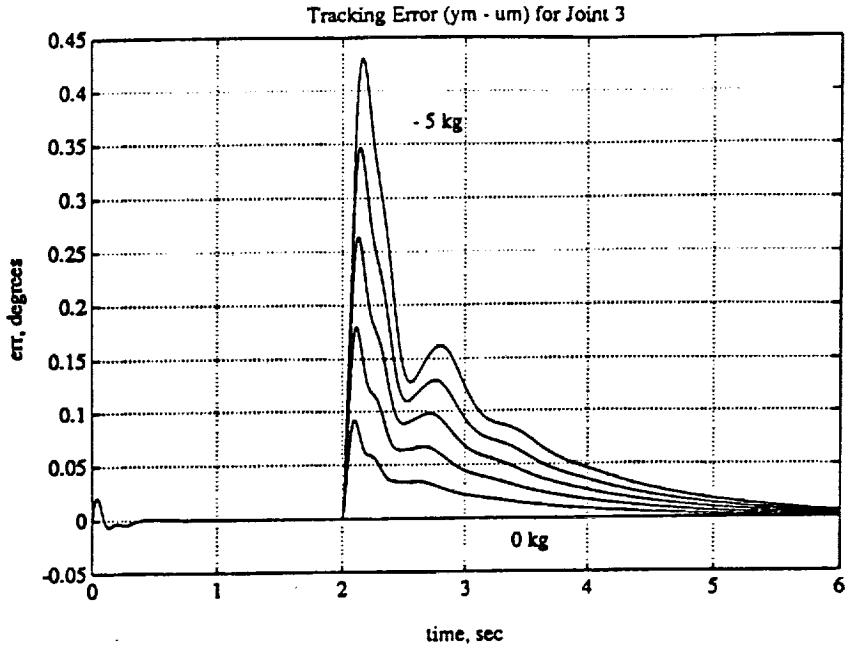

Figure 6.17: Joint 3 Error Plots for Addition of Load at Shutdown Position

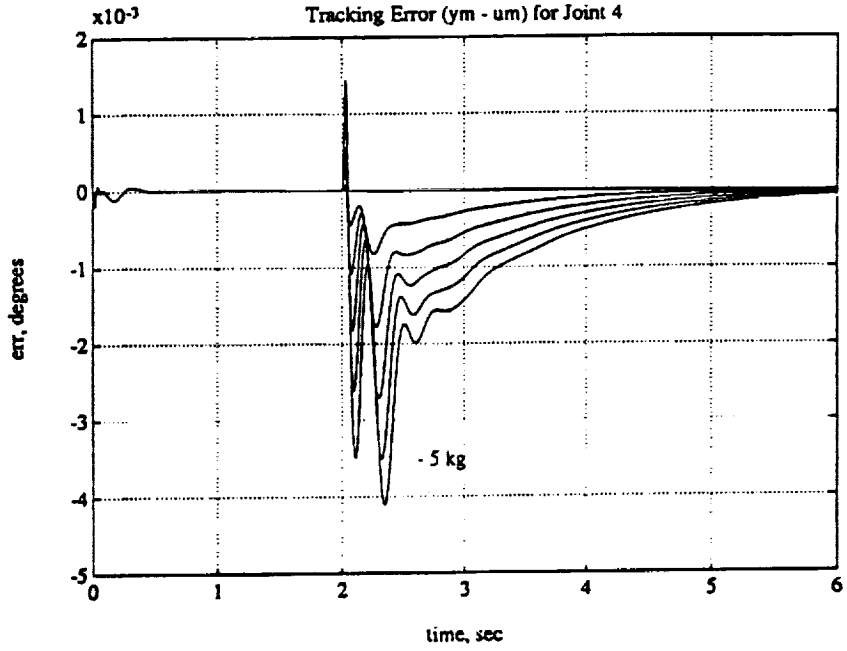

Figure 6.18: Joint 4 Error Plots for Addition of Load at Shutdown Position

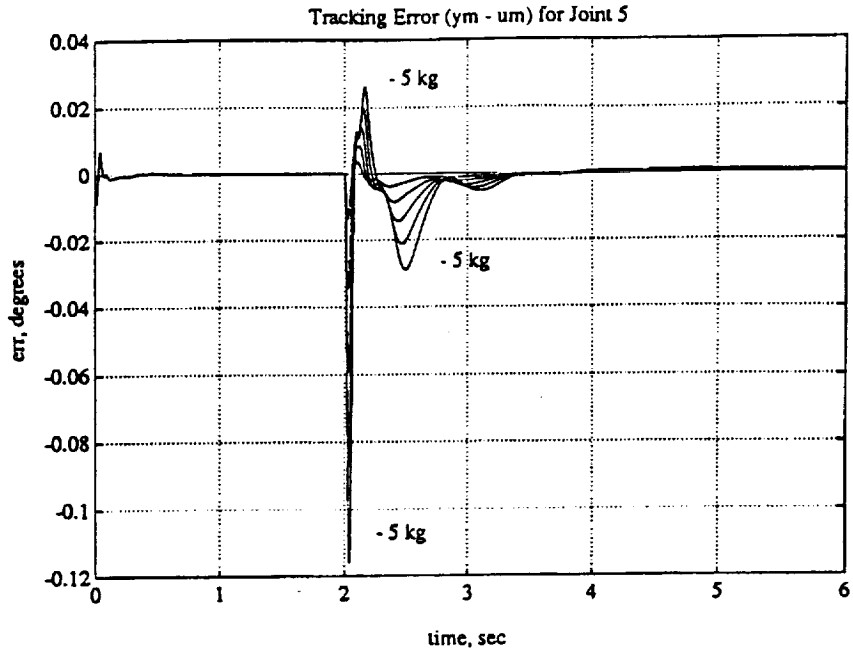

Figure 6.19: Joint 5 Error Plots for Addition of Load at Shutdown Position

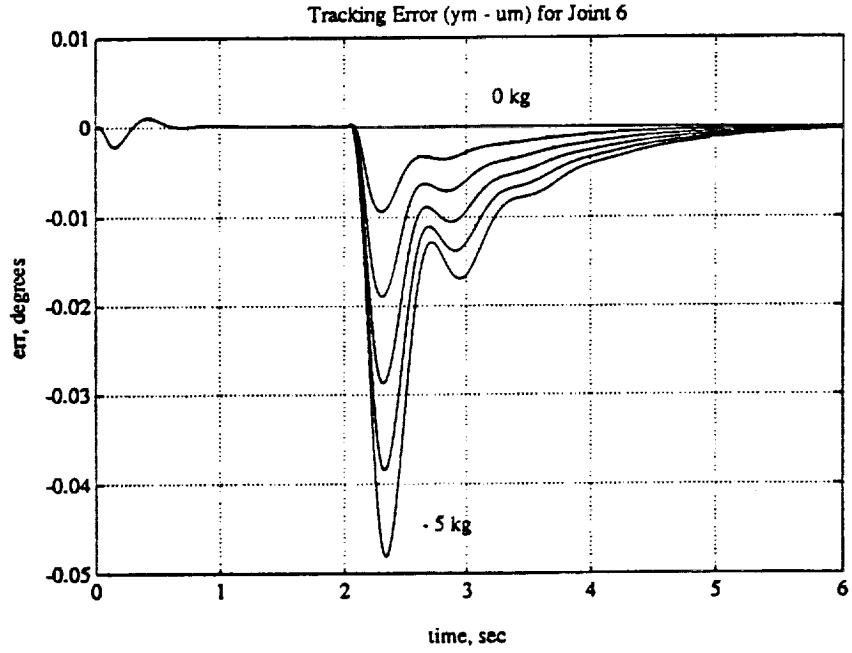

Figure 6.20: Joint 6 Error Plots for Addition of Load at Shutdown Position

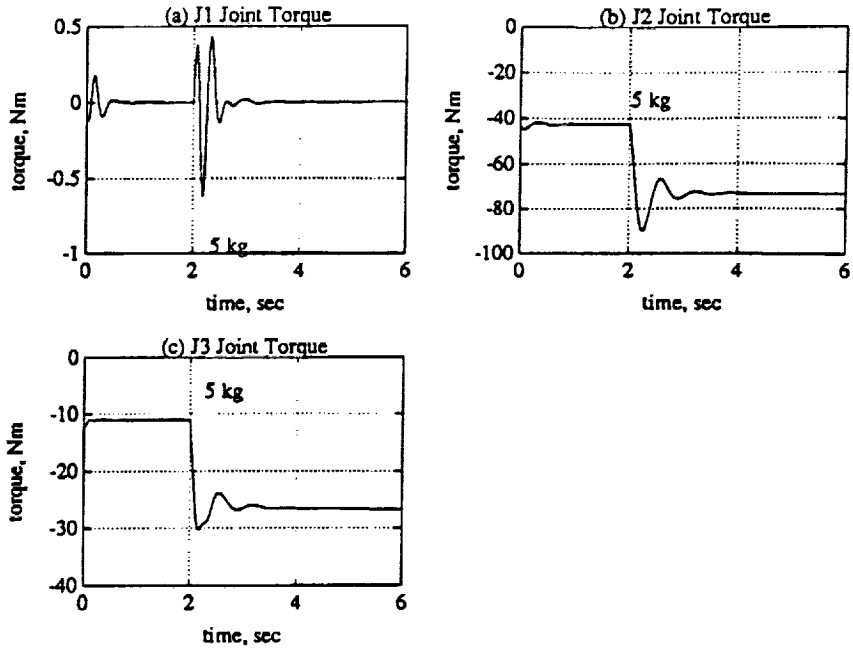

Figure 6.21: Joints 1, 2, and 3 Torque signals for Dynamic Case One, $5kg$. (a) Joint 1. (b) Joint 2. (c) Joint 3.

6.2.2 Second Case

The second case follows the trajectory shown in Table 6.9 and is illustrated in Figure 6.22. The arm moves from the shutdown position to a fully outstreatched position, waits there for 2 seconds, and then moves back to the shutdown position. The loads were added at t = 6 seconds when the robot was outstretched. Figures 6.23 to 6.28 show the reference model following error, $(y_p - y_m)$, for each of the six joints under the six load conditions. Joint 2 and 3 were affected the most by the load changes. Joint 2 had a peak error of about 9 degrees which decayed to zero in about 4 seconds. Joint 3 had a peak error of 1.8 degrees which decayed to the no load error signal in about 2.5 seconds. The other joints had load induced errors which were typically lower then the tracking errors for the no load case. The desired and actual joint positions for Joints 1, 2, and 3 are shown in Figure 6.29 for the $5kg$ case.

Table 6.9: Second Dynamic Load Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	90	-
1*	45	0	90	0	90	0	5
2*	45	0	90	0	90	0	2
3*	0	-45	180	0	45	90	5

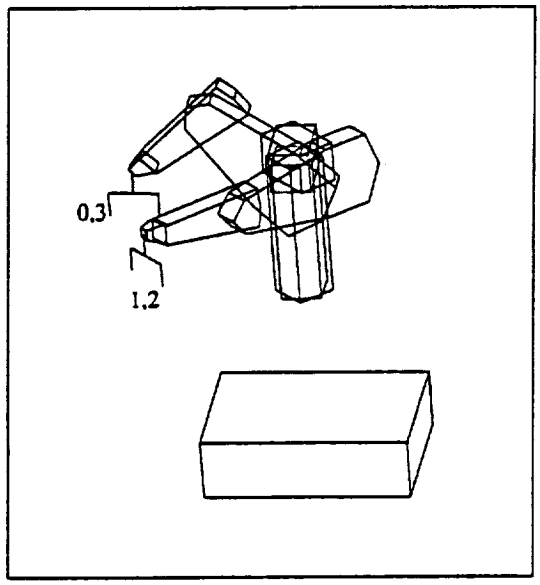

Figure 6.22: Trajectory Used for Second Dynamic Load Change

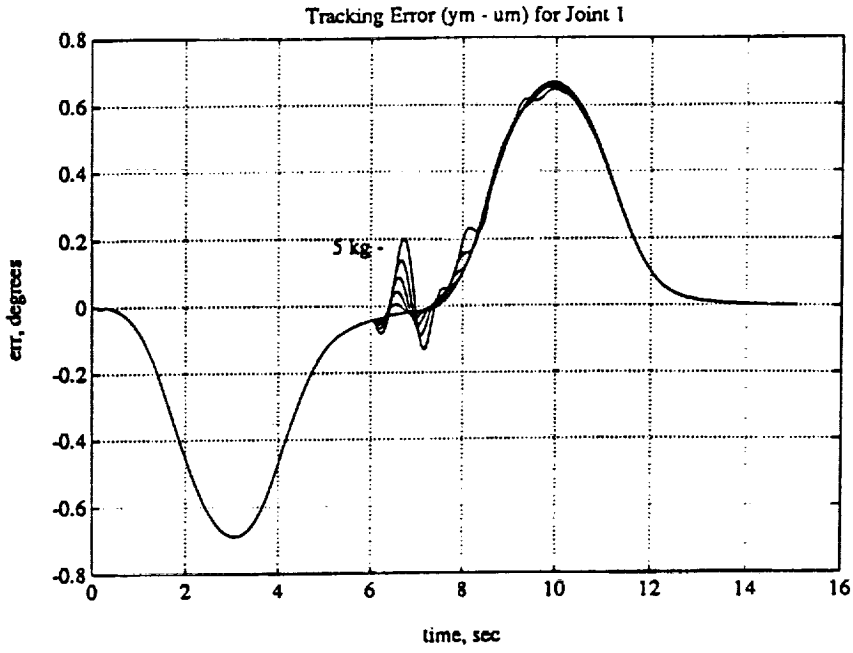

Figure 6.23: Joint 1 Error Plots for Second Dynamic Load Case

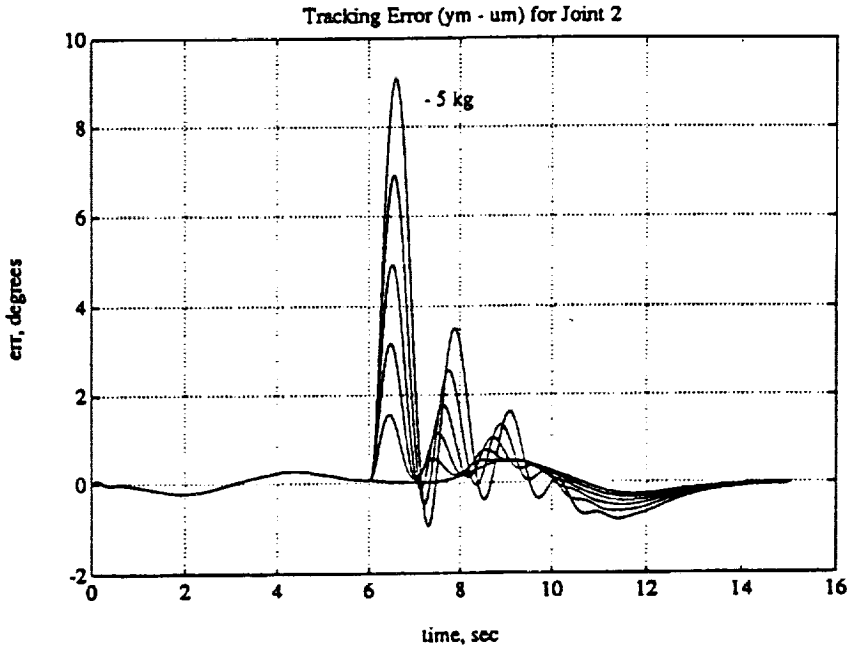

Figure 6.24: Joint 2 Error Plots for Second Dynamic Load Case

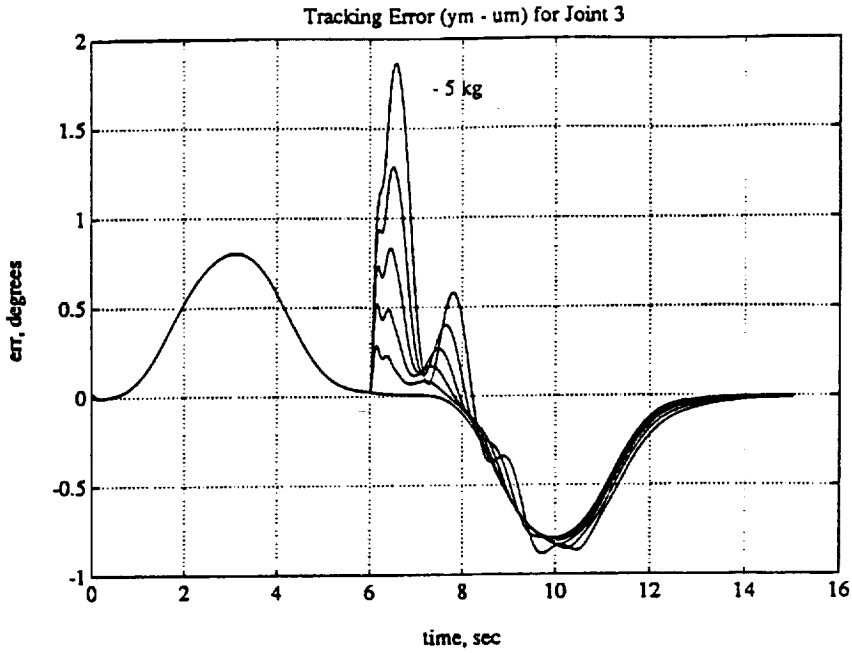

Figure 6.25: Joint 3 Error Plots for Second Dynamic Load Case

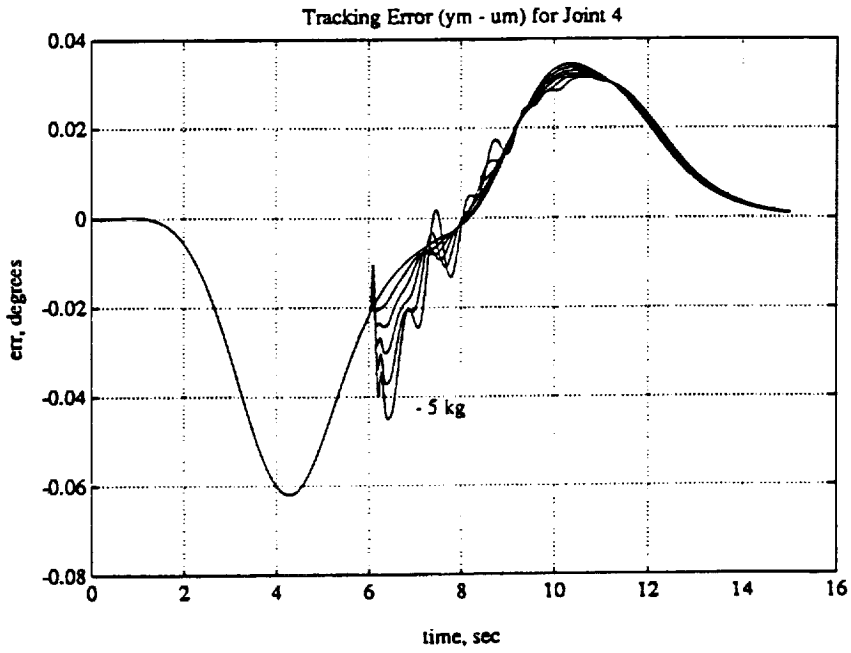

Figure 6.26: Joint 4 Error Plots for Second Dynamic Load Case

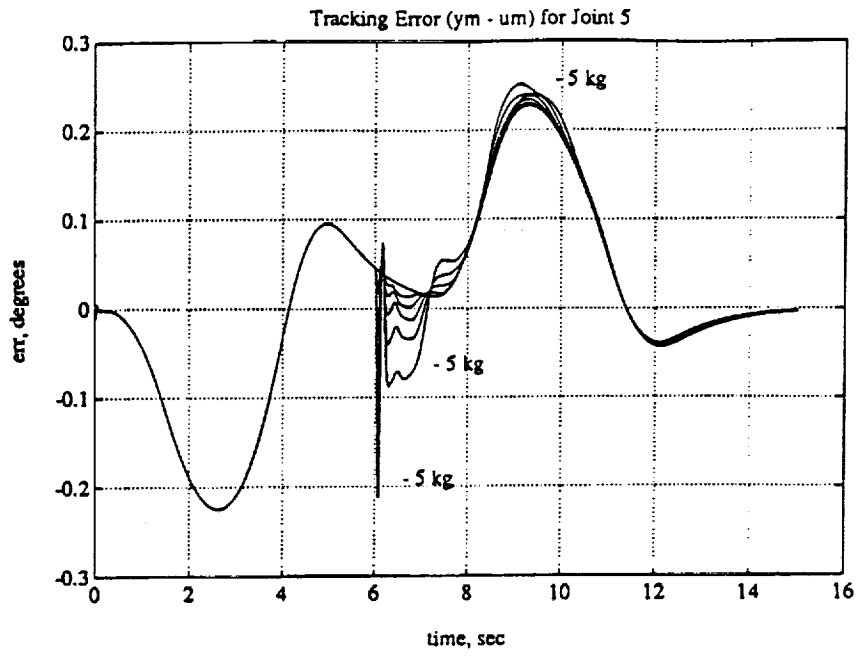

Figure 6.27: Joint 5 Error Plots for Second Dynamic Load Case

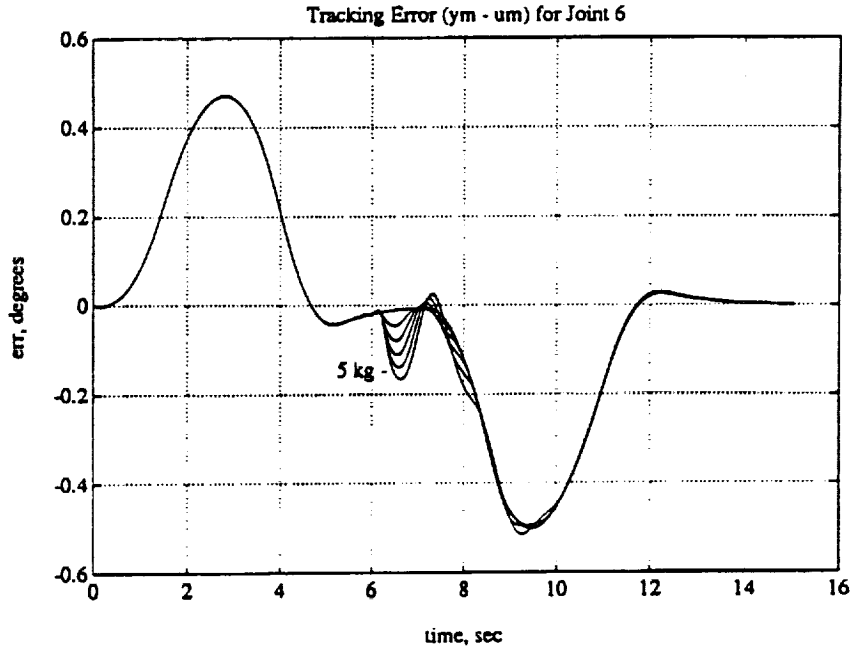

Figure 6.28: Joint 6 Error Plots for Second Dynamic Load Case

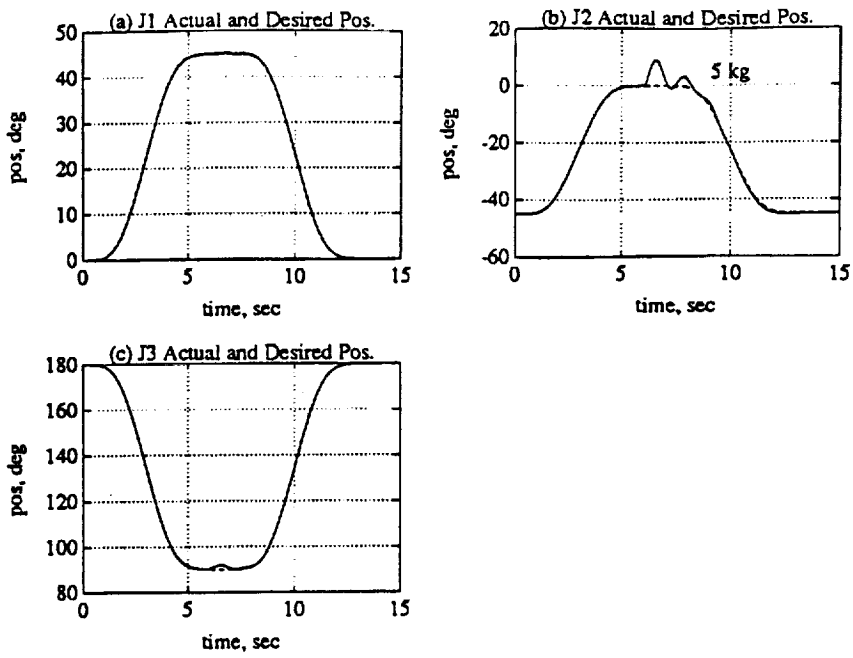

Figure 6.29: Joints 1, 2, and 3 Position for $5kg$ Dynamics Load Case Two. (a) Joint 1. (b) Joint 2. (c) Joint 3.

6.2.3 Third Case

For the third case, the trajectory in Table 6.10 was used. Figure 6.30 illustrates the trajectory given in the table. The robot was commanded to move from the shutdown position to a vertical position along a slow trajectory of 10 seconds. The various loads were added at t = 5 seconds when the robot was halfway to its destination. The model following error plots are shown in Figures 6.31 to 6.36.

For Joints 1 and 6 the error disturbances caused by the load additions were small compared to the no load error. Joints 2 and 3 were affected the most by the load changes having a 2.0 degree and 1.1 degree peak error for the $5kg$ load respectively. For the $5kg$ load, the wrist Joints 4 and 5 had their peak error doubled from the no load case to -0.41 degrees and 0.11 degrees respectively. In all cases, within about 2 seconds, the error trajectories decayed back to and approximately followed the no load error trajectories. The joint positions, model output, and

Table 6.10: Third Dynamic Load Trajectory

Knot Point	Joint Positions (deg)						Time (sec)
	1	2	3	4	5	6	
0	0	-45	180	0	45	90	-
1*	45	-90	90	90	90	0	10
2*	45	-90	90	90	90	0	10

torques for Joints 2 and 3 with the $5kg$ load are shown in Figure 6.37. The significant adaptation gains for Joint 2 (row 2 of K_P and row 2 of K_I) are shown in Figure 6.38 for the $5kg$ load addition.

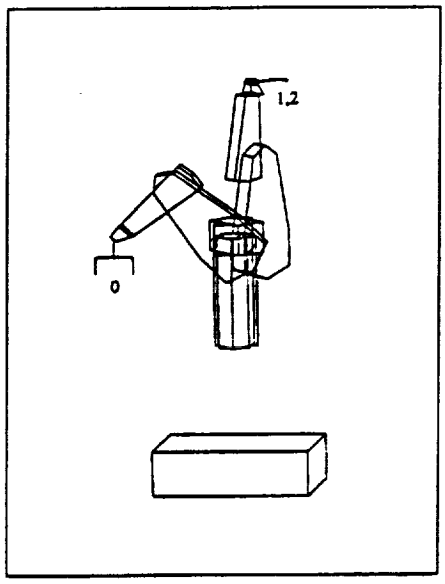

Figure 6.30: Trajectory Used for Third Dynamic Load Change

6.2.4 Summary

By examination of the above three cases, it is clear that the model following errors for Joints 2 and 3 had the worst performance of the six. The Joint 1 model following error was quite reasonable for the load additions. Joint 1 saw the largest increase in innertia from the load additions but saw no gravity loading change. Joints 2 and 3 had to compensate for most of the gravity loading due to the load

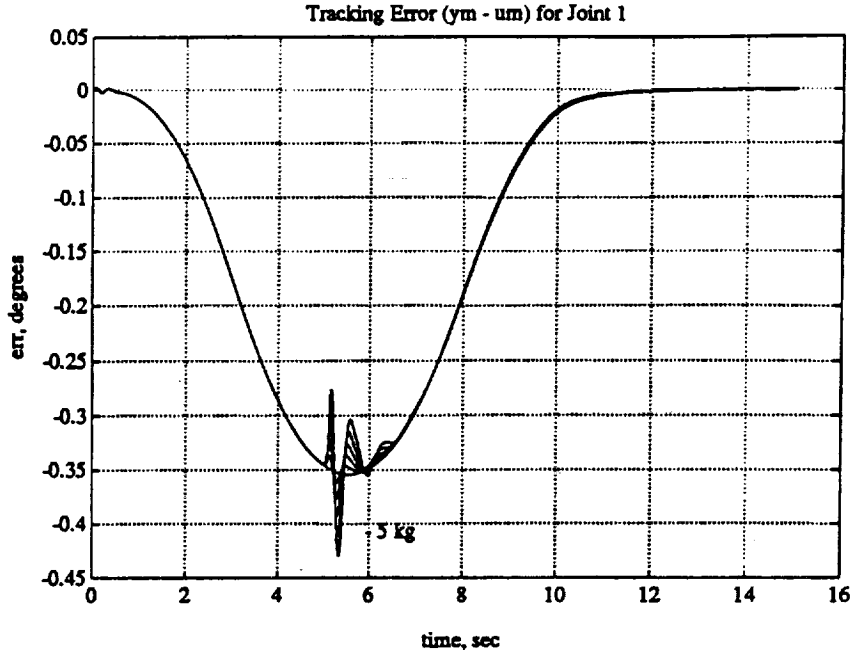

Figure 6.31: Joint 1 Error Plots for Third Dynamic Load Case

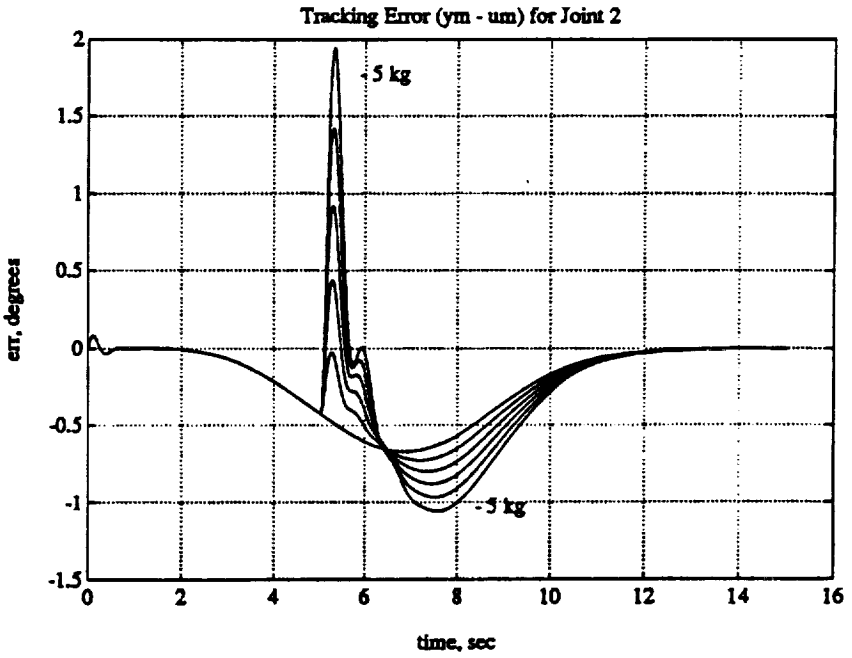

Figure 6.32: Joint 2 Error Plots for Third Dynamic Load Case

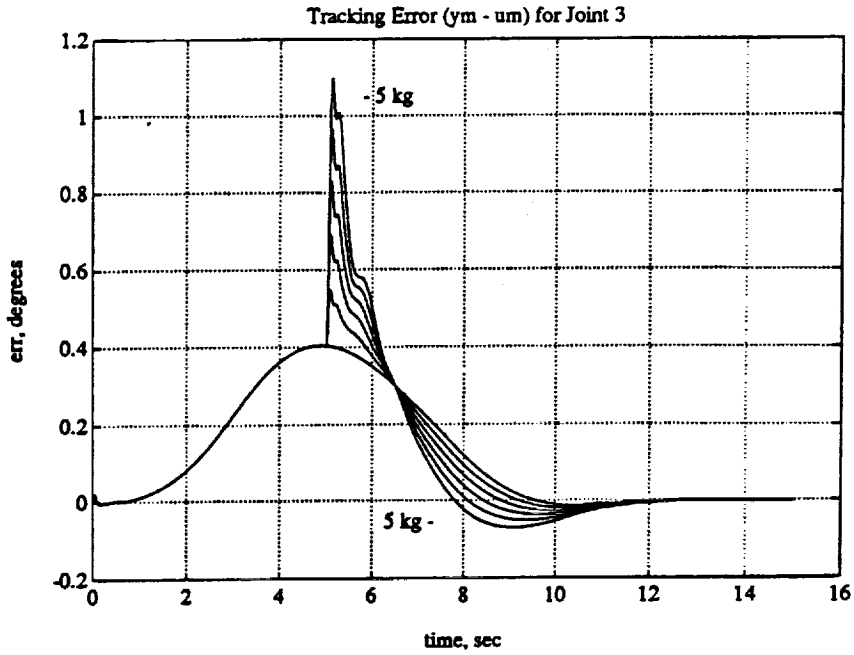

Figure 6.33: Joint 3 Error Plots for Third Dynamic Load Case

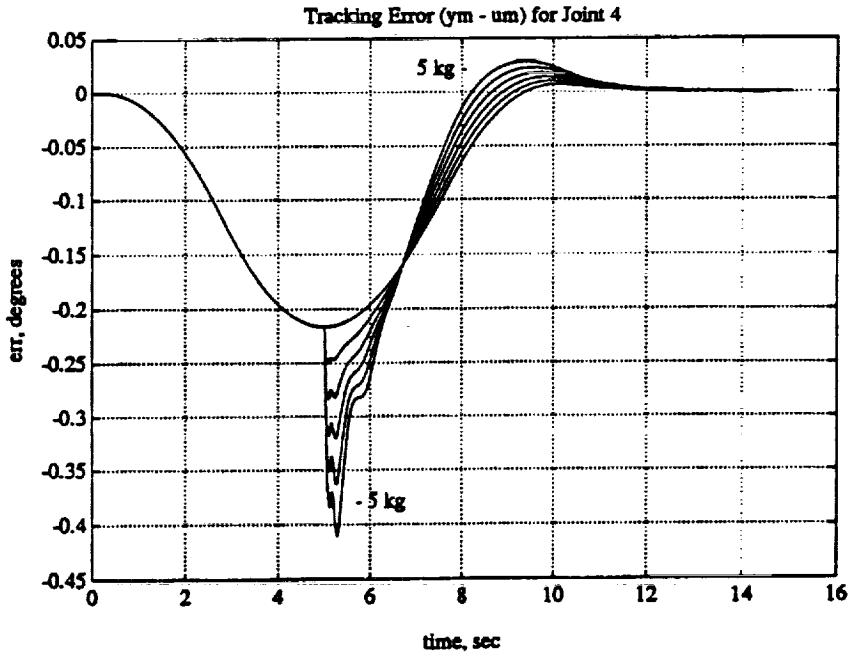

Figure 6.34: Joint 4 Error Plots for Third Dynamic Load Case

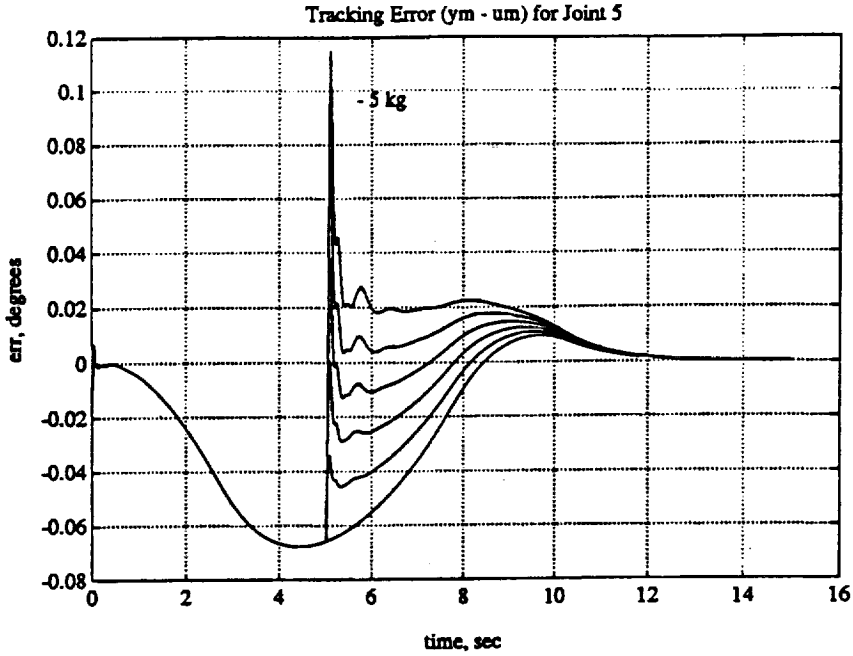

Figure 6.35: Joint 5 Error Plots for Third Dynamic Load Case

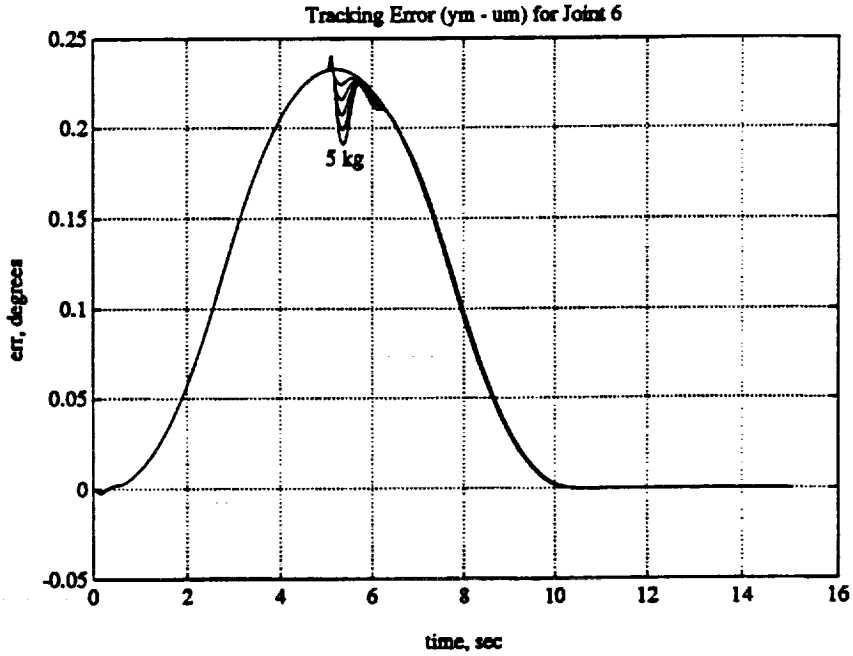

Figure 6.36: Joint 6 Error Plots for Third Dynamic Load Case

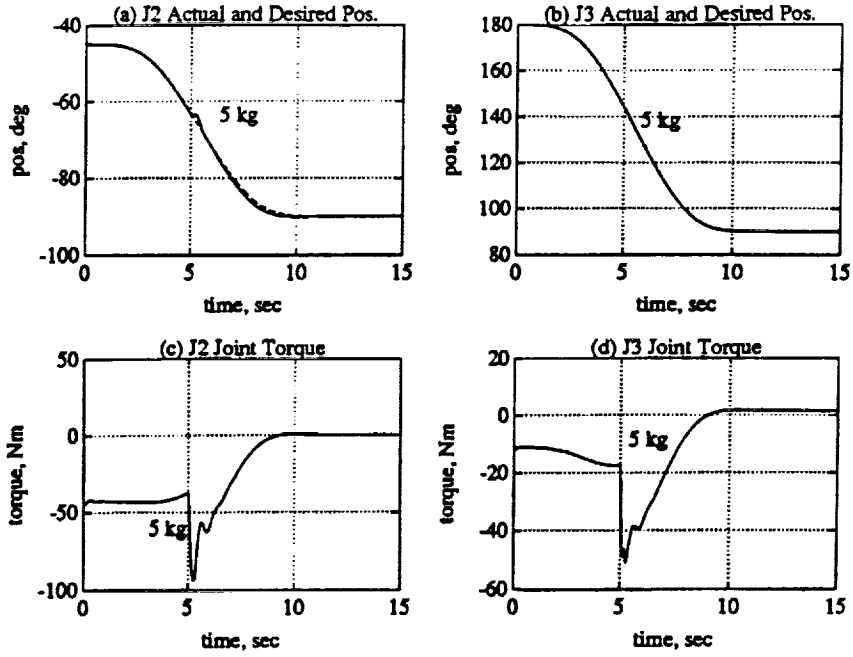

Figure 6.37: Joints 2 and 3 for Third Dynamic Load Case ($5kg$). (a) Joint 2 Position. (b) Joint 3 Position. (c) Joint 2 Torque. (d) Joint 3 Torque.

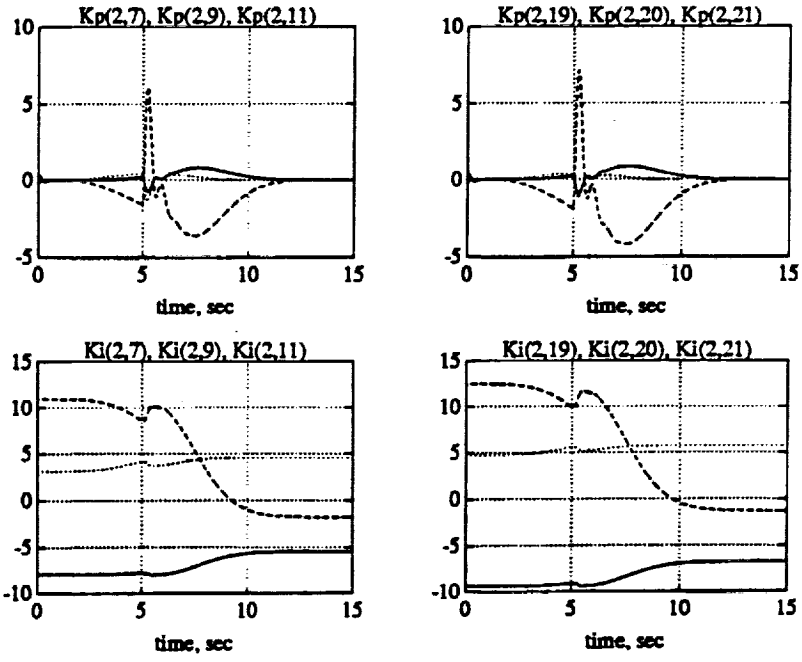

Figure 6.38: Significant Elements in K_I and K_P for Joint 2 ($5kg$)

additions but did not see as much of an innertia change as Joint 1. Thus, it would seem that the DMRAC algorithm had a more difficult time adjusting to the gravity loading than it did to the innertia changes.

CHAPTER 7

Simulation Results (Reducing Trajectory Tracking Error)

The results of the previous two chapters dealt with the error between the plant and the model output (the model following error) and ignored the trajectory tracking error ($y_p - u_m$) which is of interest when the time position of the robot joints is important as in multiple arm interactions or time dependent assembly tasks. As was seen in the preceding chapters, the reference model introduces a lag to the minimum jerk trajectories which produces a trajectory tracking error. This chapter will investigate two methods to reduce this error. The first method introduces an inverse reference model dynamics block between the trajectory generator and the reference model so y_m follows the desired trajectory. The second method investigates the effects of simply speeding up the reference model to reduce the lag. All results will be displayed with the bias term, q_{bias} removed (Section 2.7).

7.1 Tracking Errors

The results presented in the previous two chapters have delt primarily with the model following error, $y_p - y_m$, since the original goal of the DMRAC algorithm was to minimize the model following error. This is a valid goal for control design but is not sufficient for trajectory planning design. From the trajectory generation point of view it is desirable to have a control design which forces the robot manipulator to follow the commanded trajectory with as little trajectory following error as possible. Because of the lag introduced by the reference model dynamics, the results of the last two chapters do not meet the trajectory planning goals.

Figure 7.1 shows the response to a typical minimum jerk trajectory (Joint 2 was selected because it had the greatest error). Figure 7.1(a) shows the delay introduced by the model, the solid line is the desired trajectory (input to the model) and the

dashed line is the model output. Figure 7.1(b) shows the model following error. The peak error was about 1.75 degrees. The trajectory tracking error is shown in Figure 7.1(c). The peak trajectory tracking error was about -11.5 degrees where the model lag made up about 80% of the error.

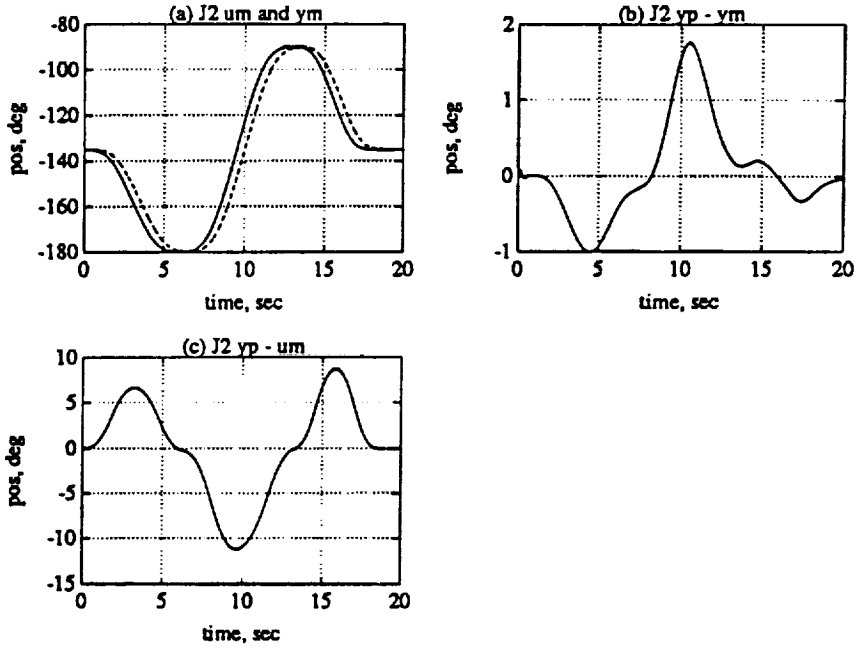

Figure 7.1: Reference Model Introduced Lag and Tracking Errors. (a) Model Introduced Lag, (b) Model Following Error, (c) Trajectory Following Error

7.2 Predictive Compensator

The first method used to reduce the trajectory following error is to use the information about the reference model dynamics to adjust the trajectory which is sent to the model. Figure 7.2 shows the current implementation of the trajectory generator and reference model interaction where q is the desired trajectory. To force y_m to follow q, we can insert some dynamics between the trajectory generator and the reference model as shown in Figure 7.3. The predictor in Figure 7.3 uses the fact that the trajectory and its higher derivatives (when using minimum jerk) are

known and the fact that the reference model dynamics are also known and constant to adjust the input to the model, u_m such that y_m follows q. If the reference model dynamics are given by $G(s)$ and the predictor dynamics are given by $H(s)$, then the following holds:

$$u_m(s) = H(s)q(s) \tag{7.1}$$

$$y_m(s) = G(s)u_m(s) \tag{7.2}$$

from which we get,

$$y_m(s) = G(s)H(s)q(s) \tag{7.3}$$

Now, if $H(s) = G(s)^{-1}$, then (7.3) reduces to,

$$y_m(s) = q(s) \tag{7.4}$$

which is the desired result. $G(s)$ is given by,

$$G(s) = y_m(s)/u_m(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{7.5}$$

Thus, $H(s)$ is then,

$$H(s) = u_m(s)/q(s) = \frac{s^2 + 2\zeta\omega_n s + \omega_n^2}{\omega_n^2} \tag{7.6}$$

Converting (7.6) into the time domain yields,

$$u_m(t) = \frac{\ddot{q}(t)}{w_n^2} + \frac{2\zeta\dot{q}(t)}{w_n} + q(t) \tag{7.7}$$

The realization of (7.6) requires two differentiations of the desired trajectory $q(t)$. Since the input, q, is typically formed analytically, there is no noise to be amplified by the differentiation and the higher order derivatives can also be solved

for analytically. For the minimum jerk trajectories, the higher order derivatives are given by (3.16)-(3.17). Using a minimum jerk trajectory gives us continuous functions for q, $\dot{q}$, and $\ddot{q}$, thus the model input (7.7) is continuous. Also, the minimum jerk equations yield $q = 0$, $\dot{q} = 0$, and $\ddot{q} = 0$ at knot points which will result in a $u_m = 0$ from (7.7). This means that u_m is causal (i.e. undefined at times before the beginning of a trajectory segment) which means that it can be realized in real time without the need for computing all of the trajectory segment paths off-line.

Figure 7.4 shows an example u_m generated for a minimum jerk trajectory q. The solid line is the desired trajectory and the dashed line is the computed model input which, when applied to the reference model, will yield a model output that follows q. Notice that u_m is not only continuous but is also smooth. When the u_m generated by this predictor is applied to the reference model, the model output will follow q and the DMRAC algorithm will force the plant to follow y_m and thus q.

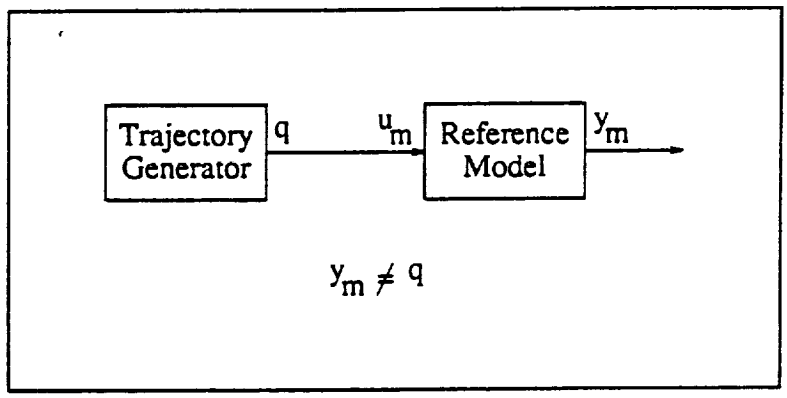

Figure 7.2: Current Implementation of Trajectory Generation

7.3 Predictive Compensation Simulation Results

The same trajectory illustrated in Figure 7.1 was used with the predictor setup illustrated in Figure 7.3. The results of the run are shown in Figure 7.5. Figure 7.5(a) shows the trajectory tracking error $y_p - q$. Figure 7.5(b) illustrates the

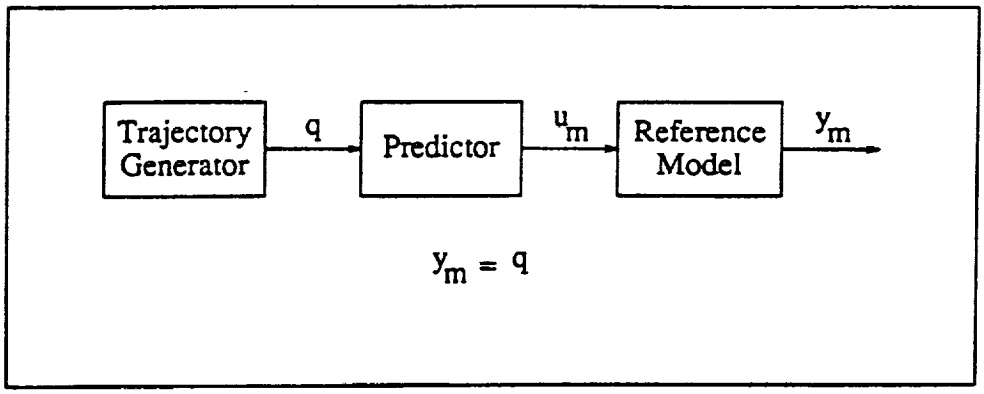

Figure 7.3: Predictive Implementation of Trajectory Generation

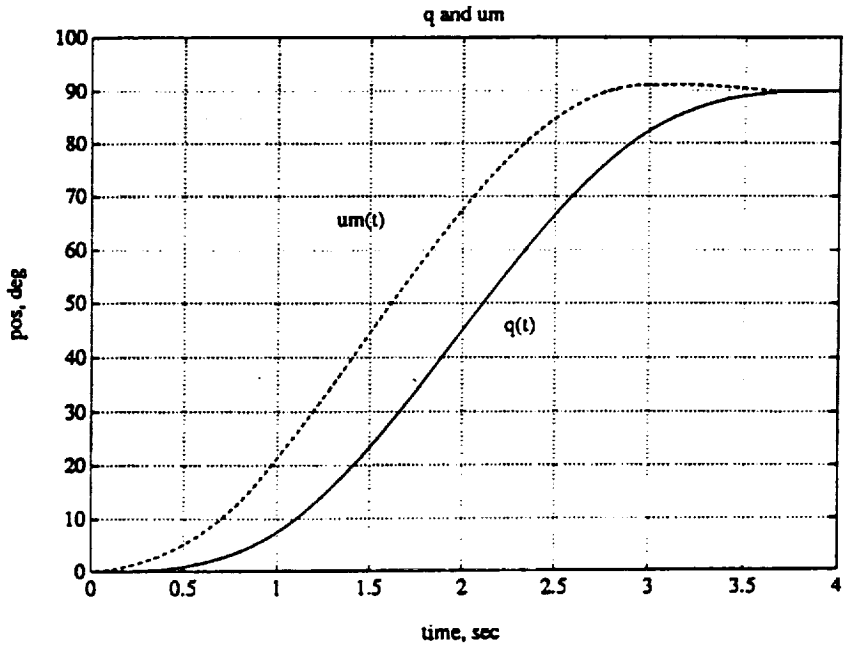

Figure 7.4: Example Output from Predictor

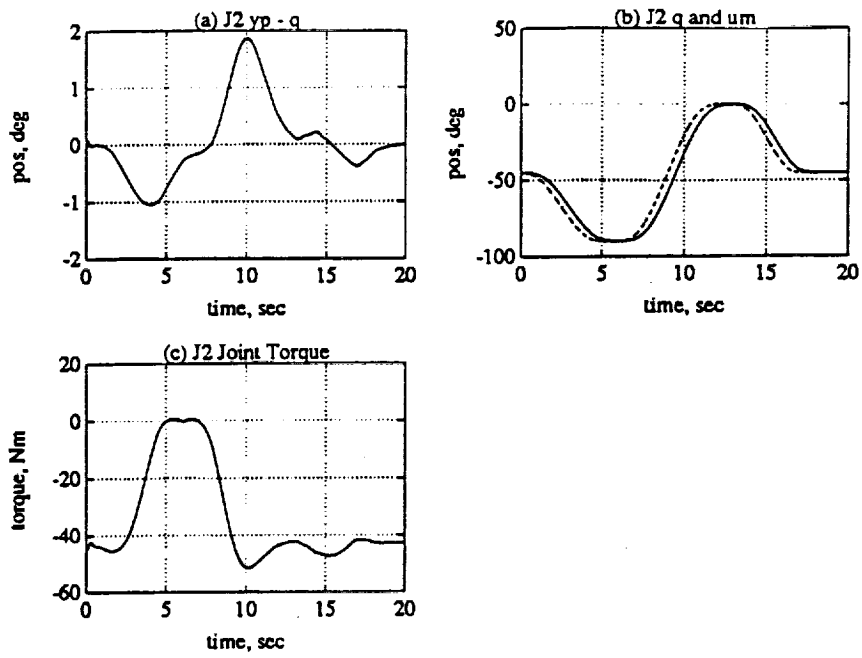

Figure 7.5: Joint 2 Response using Predictor. (a) Trajectory Tracking Error. (b) Trajectory and Synthesized Model Input. (c) Joint Torque.

desired trajectory q (solid) and the modified model input u_m (dashed). Figure 7.5(c) shows the torque for Joint 2. Comparing Figure 7.5(a) to Figure 7.1(c) shows the improvement gained by using the predictor. The peak trajectory tracking error was reduced from 11.5 degrees to 1.86 degrees. It is interesting to note that the trajectory tracking error was not improved beyond the original model following error (without the predictor).

7.4 Increasing Reference Model Speed

A second method to reduce the trajectory tracking error is to simply increase the undamped natural frequency of the reference models. Figure 7.6 shows the trajectory tracking error when w_n for the first three Joints is increased from 4 to 20 *rad/sec*. Comparing Figure 7.6 to Figure 7.1(c) shows the improvement gained by increasing w_n. The peak error was reduced from 11.5 degrees to 1.81 degrees. Note:

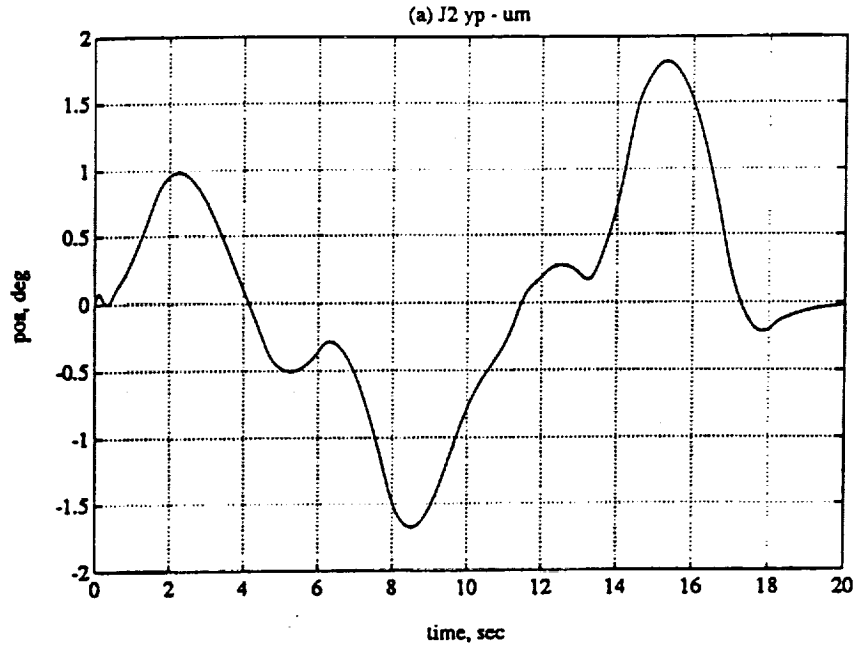

Figure 7.6: Joint 2 Response using Increased w_n

Increasing w_n will decrease the lag introduced by the model but will not remove it entirely.

7.5 Summary

In this chapter we introduced two methods to improve the trajectory tracking error. The first consisted of adding a predictor block between the trajectory generator and the reference model to "advance" the trajectory to counteract the lag introduced by the model. For minimum jerk trajectories, this predictor block resulted in a very simple realization. The result was a greatly improved tracking error.

The second method was to simply increase the reference model speed. This also resulted in a great improvement on the trajectory tracking error.

CHAPTER 8
CIRSSE Testbed Environment

This chapter will briefly discuss the portions of the CIRSSE[1] Testbed which were used to control an actual PUMA 560 Manipulator with the DMRAC algorithm. The CIRSSE Testbed is a two arm, 18 degree of freedom (DOF), redundant robotic manipulator equipped with an extensive sensory array. Each arm is composed of a six DOF PUMA 560/600 Robotic Manipulator mounted on a three DOF platform. The many sensors include 18 joint encoders for accurately determining manipulator joint positions, two force/torque sensors at the end of each arm, various joint limit switches, force controllable grippers with infra-red cross fire sensors, five black and white video cameras, and a laser range finder. The Testbed hardware is controlled by a collection of networked Sun4 workstations, a VME bus cage containing five 68000 series CPUS and various peripherals which is used to control aspects of manipulator motion, and a second VME bus cage containing two 68000 series CPUS and a Datacube high speed vision processor which is used for the vision aspects of the Testbed. The Testbed was developed at the Center for Intelligent Systems for Space Exploration at Rensselaer Polytechnic Institute to support development of cooperative robotic systems.

This chapter will briefly discuss the hardware and software comprising the CIRSSE Testbed. Only the portions of the Testbed which were used to control the PUMA 560 will be discussed. Although, at the time of this writing there did not exist a complete description of the CIRSSE Testbed, a good working knowledge can be pieced together from [30] and the sources cited therein. This chapter will also discuss some implementation issues.

[1]Center for Intelligent Systems for Space Exploration, Troy, NY

8.1 CIRSSE Testbed Hardware

The CIRSSE Testbed contains a vast array of robotic hardware for performing experiments in robotic control, manipulation, and planning; vision processing and visual servoing; and distributed computing and control. This section will describe the hardware portions of the testbed which were used in the experiments.

8.1.1 Puma Manipulators

The CIRSSE Testbed manipulators consists of a PUMA 560 and a PUMA 600 each mounted on three DOF transporter platforms. Only the PUMA 560 (to be referred to as the PUMA in subsequent sections) was used in these experiments. Figure 8.1 shows a picture of the PUMA mounted on the platform.

The PUMA is a six DOF manipulator where each joint is actuated through a speed reducing gear train by a permanent magnet direct current (PM DC) motor. Each motor is equipped with a position encoder to accurately determine the motor position and thus the joint position. Note: The motors are not equipped with tachometers so the velocity information must be derived from the position data as discussed in Section 8.3.1 The coordinate frames used for the PUMA are detailed in [23] and are the same as those used in the previous simulation sections. Table 8.1 lists the joint ranges for the PUMA. The PUMA end-effector is equipped with a pneumatic gripper which was used to grasp a hook onto which various weights could be attached.

8.1.2 Unimate Controller

The original Unimate controllers are used to power the joint motors and read the joint encoders in the PUMA arm. The Unimate controller is set up in a hierarchical fashion where each joint is individually powered by a separate power amp and controlled by a separate joint micro-processor. Each joint micro-processor takes

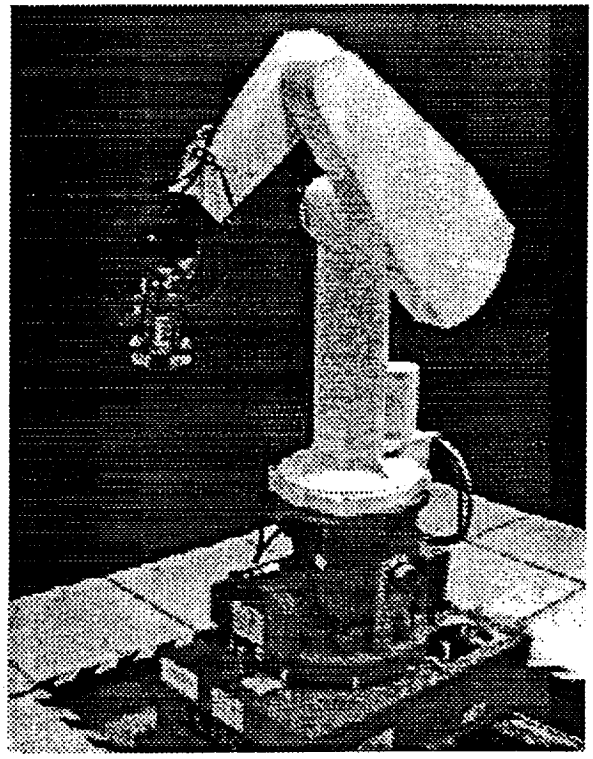

Figure 8.1: PUMA 560 Manipulator

Table 8.1: PUMA 560 Joint Ranges

Joint	Minimum Position (degrees)	Maximum Position (degrees)
1	-250	70
2	-223	43
3	-52	232
4	-134	150
5	-100	100
6	-262	250

care of reading the position encoders, servoing the arm with a built in control loop, sending desired torques to the joint, and some other miscellaneous low level tasks. Each of these joint processors communicates to an Arm Interface Board (AIB). The AIB is accessed through a DR-11C interface board. All of these boards are mounted in a card cage with a Qbus backplane inside the Unimate controller. The CIRSSE Motion Control Systems hardware (described in Section 8.1.4) interfaces to the DR-11C through a Qbus to VME mapper. For more detailed information on the interface to the Unimate Controller see [31]. Note: The original VAL II control language which was shipped with the Unimation controller was bypassed and the joint micro-processors were accessed directly through AIB card.

The joint micro-processor cards support two modes of joint motion. The first mode is POS MODE (position) where an internal servo loop is used to position the joints to some desired angular position. This positioning mode is initially used to position the robot in the shutdown position (as defined in the previous simulation sections) before the DMRAC algorithm was enabled. The second mode is CUR MODE (torque) where the micro-processor allows the joint torques to be controlled at each sample interval. The second mode is used by the DMRAC algorithm to control the robot joint positions.

8.1.3 CIRSSE Computing Network

The CIRSSE computing network is a collection of various processor platforms all connected via an EtherNet. The various platforms (or chassis) consist of Sun4 workstations and two VME Backplane Cages. One of the VME Cages is used to control the robots and is called the Motion Control System (MCS) and the other VME Cage is used to control the vision systems and is called the Vision Services System (VSS). Both of the VME Cages use one of the Sun4's (labeled Venus) as a gateway to the network which helps to isolate some of the Sun network traffic from

the VME Cages.

For the DMRAC experiments, only one of the Sun4's (Venus) and one of the VME Cages (MCS) were used. The majority of the software ran on the MCS Cage and the Sun4 was used primarily to display data and service user requests.

8.1.4 Motion Control System Cage

The Motion Control System Cage (MCS Cage) is a VME Bus card cage which contains the computers and peripherals used to control all aspects of robot motion on the CIRSSE Testbed. The portions of the MCS Cage used in the DMRAC experiments consists of the following boards:

- Three Motorola MVME147SA-2's (68030 series cpus, 32 MHz, 8 Meg RAM),
- Two Motorola MVME135's (68020 series cpus, 16 MHz, 1 Meg RAM),
- One Motorola MVME-224-1 (Shared Memory Module),
- One VMEbus to QBus Mapper (for Communication with Unimate Controller)

There are eight other VME cards in the cage which are used for other aspects of the CIRSSE Testbed (platform control, Force/Torque Sensor Control, etc.) which were not used for the DMRAC experiments and will not be described. One of the MVME147SA-2's has an EtherNet port which is connected to the CIRSSE computer network (described in Section 8.1.3).

8.2 Software

This section will describe the flexible multi-layer software system which runs on the CIRSSE Testbed processors. All of the code described below was written in *C* [32].

8.2.1 Overview

The CIRSSE Testbed is comprised of five major software components as shown in Figure 8.2. Unix and VxWorks [33] are the foundation of the CIRSSE software development environment. Unix is the multitasking operating system used on the Sun4 workstations and VxWorks is the real time multitasking operating system used on the 68000 series VME Cage processors. The CIRSSE Testbed Operating System (CTOS) was developed to overcome limitations in UNIX and VxWorks when dealing with interprocessor communication, synchronization, and distribution. The Testbed Components consist of the Motion Control System (MCS) which is used to control the motion of the robotic manipulators and the Vision Services System[2] (VSS) which is used to control the various Testbed vision systems. The final component, Applications and Experiments, consists of the code which is used to control a particular experiment on the CIRSSE Testbed.

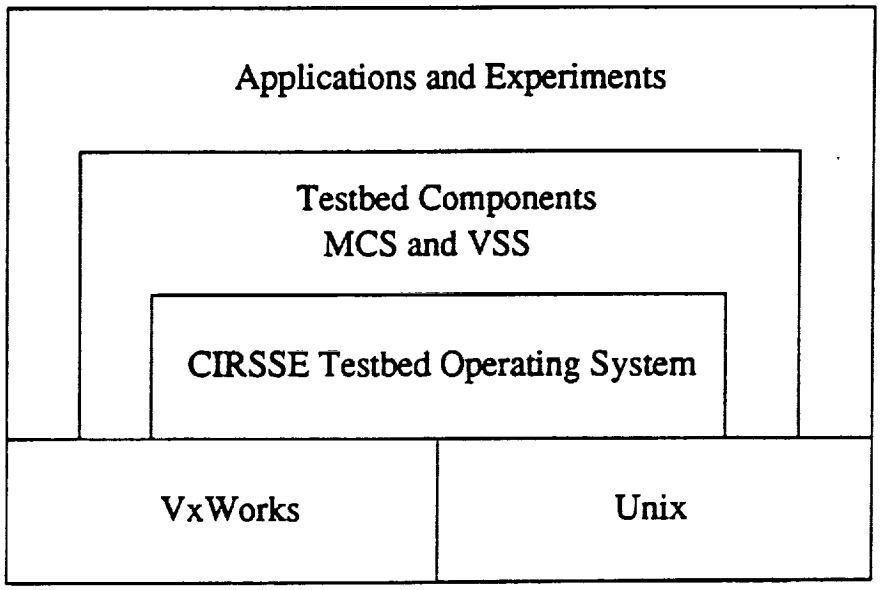

Figure 8.2: CIRSSE Testbed Software

Figure 8.3 shows a detailed block diagram of the software used to run the

[2]Not used in the DMRAC experiments

DMRAC experiments on an actual PUMA 560 Robotic Manipulator. The various components shown in the figure will be described in the subsequent sections.

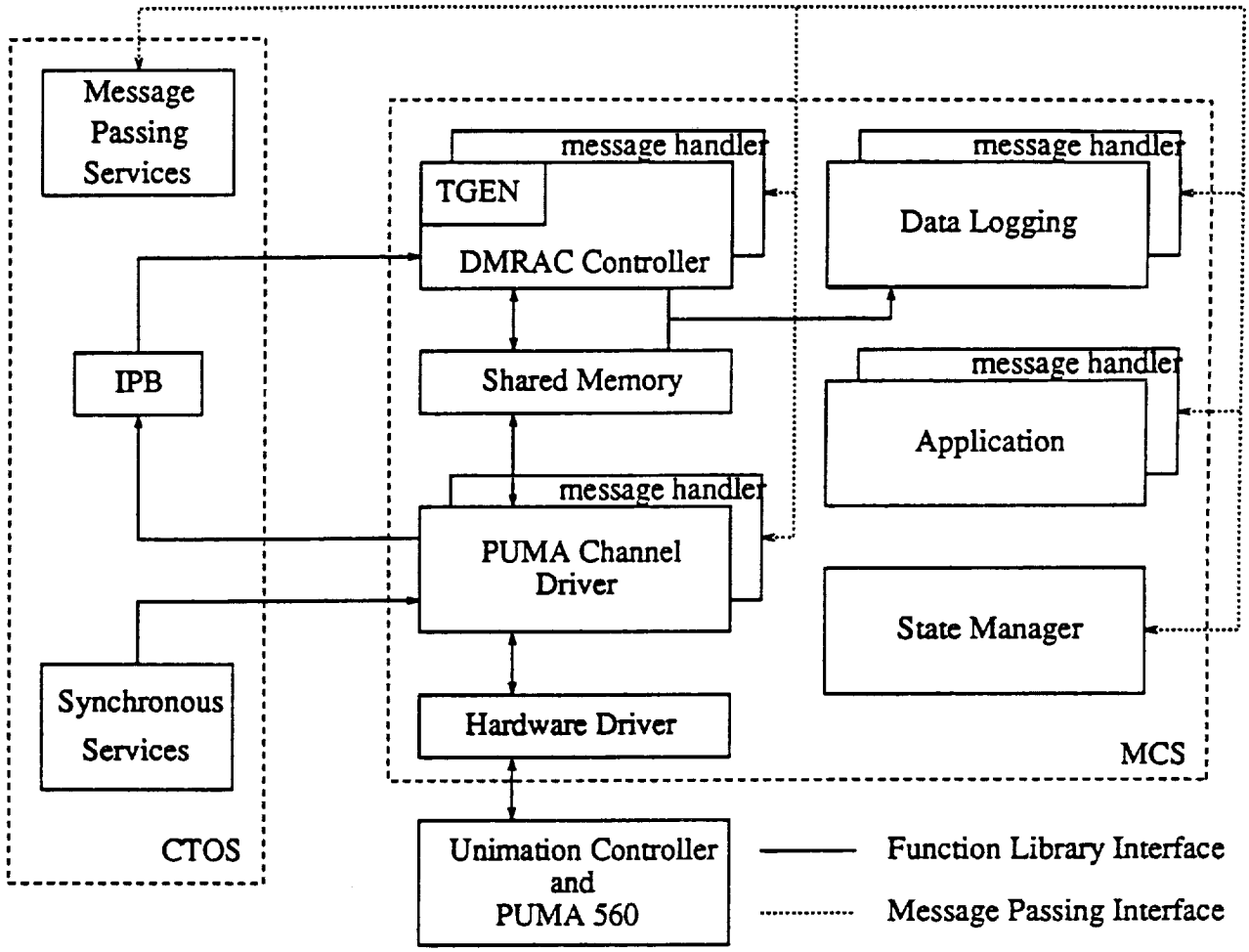

Figure 8.3: Block Diagram of Software Used in DMRAC Experiments

8.2.2 Multi-Tasking Unix and VxWorks

The basic building block of the software system is the task. A task is a single thread process, or program, which runs on a single processor. A task in the CTOS system can be classified as a message handler, a synchronous task, or a data driven task. These classifications will be defined in the following sections.

Tasks generally have two states, running and blocked. When a task is running

it is allowed to use processor resources in a time sliced manor. When a task is blocked, it is idle and does not run at all. A task is typically blocked when it is waiting for data or for some hardware peripheral to become available.

The entire software collection consists of many tasks all running at the same time. On a single processor, only one task can use the processors resources (memory, CPU, floating point processor, timers, etc.) at a time, thus all tasks are given a small window of time to use the resources and then this window is passed on to another task. Unix and VxWorks take care of this window passing or multi-tasking. What Unix and VxWorks do not support is a uniform and easy method for communication between tasks, synchronization of tasks, and distribution of tasks. This missing functionality is provided by CTOS.

8.2.3 CIRSSE Testbed Operating System

The CIRSSE Testbed Operating System is built on top of Unix and VxWorks to provide for interprocess communication, synchronization, and distribution. These three paradigm are provided by the three components of CTOS:

- *Interprocess Communication* – message passing services,
- *Interprocess Synchronization* – synchronous services,
- *Interprocess Distribution* – bootstrap services.

The message passing services are built on UNIX sockets (which are also supported by VxWorks) and allow tasks to communicate by passing messages amongst themselves. These messages may contain optional data. Tasks which communicate using messages are called message handlers since they respond to incoming messages or events. Message handler tasks are not considered real time since the message latency is on the order of $2-4$ *ms* on the VME Cages. Message passing is supported on the Sun4 Processors and both of the VME Cages.

Synchronous services provides a method for synchronizing tasks. Synchronization refers to the process of changing the state of one or more tasks from blocked to running such that all of the tasks in question are unblocked at roughly the same time. Synchronous Services provides two methods to unblock or release tasks, time synchronous and data synchronous. Time synchronization refers to releasing one or more tasks on a periodic basis and is the primary function of Synchronous Services. Data synchronization refers to releasing a task when data or a peripheral becomes available. This functionality is provided by a companion service called IPB which stands for Inter Processor Blocks. Tasks which are time synchronized are called time synchronous tasks and tasks which are data synchronized are called data synchronous tasks. Synchronous Services and IPB's are only provided for the VME Cages since they contain the tasks which deal with real time synchronous events.

Each time or data synchronous task is typically paired with some message handler task. There is no additional component of CTOS to support communication between time/data synchronous tasks and message handler tasks. This communication is typically carried out using shared memory or simply by using common global variables on the same processor. This shared data can be polled by synchronous tasks and simply read by data synchronous tasks when released.

An additional form of data synchronization is provided by the hardware interrupts on the VME Cage processors. Most of the lower level code of CTOS is built on interrupts.

The third paradigm, process distribution, refers to the problem of assigning the many tasks to many different processors. A task must be compiled to run on a Sun4 workstation (compiled for Unix) *or* a VME Cage Processor (compiled for VxWorks) but not both. Once a task is compiled it can be run on any of the VME processors *or* any of the Sun4 workstations. The decision as to which processor the task will run on is made at boot time when an experiment is started. A boot strap

mechanism reads a configuration file which specifies where all of the tasks are to run and then loads the tasks on the target processors. The boot strap code is also responsible for providing an orderly means for all tasks to initialize in sequence. This is accomplished by broadcasting initialization messages to all message handler tasks at start up.

8.2.4 Motion Control System

CTOS provides a flexible operating for distributed real time control but know nothing of the CIRSSE Testbed Robotic Manipulators. The Motion Control System (MCS) fills this gap by providing the following:

- Uniform interface to the Testbed manipulators,
- Standard components for joint control, trajectory planning, etc.,
- Well defined layered structure which allows for the replacement of a standard component for research.

A functioning MCS system is setup by specifying several MCS components in the boot strap configuration file along with an application manager. There is a large library of MCS components to choose from such as various joint control algorithms, trajectory generators, hardware drivers, force controllers, etc. An application manager can either be a user supplied task which will control the experiment or it can be a standard client interface program which provides a uniform interface to all aspects of MCS for researchers who wish to pursue research in task planning and intelligent robotics. The client interface hides some of the details of configuration and parameter selection.

MCS deals with slots. A slot can be a robot joint, a 6 vector of force/torque data, or anything which requires data input and output. MCS provides components

for manipulating slot data and controlling devices hooked up to slots. MCS consists of the following components:

- *State Manager* – The State Manager monitors and maintains the state of the motion control system (startup, shutdown, and various other state transitions). For more information on the various state transitions see [30]. It also provides a uniform interface between the various MCS configuration components and the client interface or application. The state manager is simply a message handler task.

- *Hardware Drivers* – Hardware drivers are software libraries which simplify the interface to some hardware device. Typically, a hardware driver provides functions to write to, read from, and initialize a device.

- *Channel Drivers* – Channel Drivers use the Hardware Drivers to synchronously access a device, like the robot manipulators. Channel Drivers are a combination of a message handler task used to communicate with the state manager and a time synchronous task used to communicate with the hardware. There is also communication between the time synchronous task and the message handler. Tasks requiring synchronous access to the hardware do so using slots which are read from and written to by the Channel Drivers.

- *Controllers* – Controllers are tasks which provide the computations required to control a device through a slot. For example, the DMRAC algorithm used to control one of the PUMA Manipulators classifies as a Controller. Controllers consist of a message handler shell which communicates with the state manager and a data synchronous task which is synchronized to the channel driver time synchronous task.

- *Trajectory Generators* – The Trajectory Generators provide smooth trajectory paths for slots which require it (generally only the robot joint slots).

- *Other Components* – Other components include collision detection processes and any task which a researcher wishes to add to MCS.

All of the data exchange between various components is either by message passing or by access to shared memory. All slot data is stored in shared memory and is accessed by standardized shared memory libraries. The above components may be freely distributed amongst the processors in the MCS VME Cage, thus much parallelism can be achieved. This flexibility of distributing tasks is provided by CTOS and easily allows processor loading to be evened out as components are added and subtracted from MCS.

MCS and CTOS also provide many safety features such as overrun tasks which are activated if a time synchronous task does not complete its computations within its allotted period. This and many other features will not be discussed here, see [30] and the references listed therein.

8.2.5 Synchronization and Data Exchange for Joint Control

Figure 8.4 shows a time diagram of the data exchange and synchronization between a joint channel driver and a joint controller. The diagram is for a single joint or slot for illustration purposes. Each interval is started when the synchronous services releases or unblocks the time synchronous task in the channel driver. The task is unblocked in this synchronous fashion on a periodic basis (every 4.5 ms for the DMRAC experiments). This task then writes a torque[3] value which was stored in shared memory to the motor controlling the slot in question. The torque value written was calculated by the controller in the previous interval thus there is a one interval delay added to the closed loop system.

After writing the torque, the channel driver task then reads the joint position, stores it in shared memory (in a slot), and releases the data synchronous task in the

[3]MCS deals with motor torques, not link torques, so scaling by the gear ratio is required.

controller associated with this slot (using the IPB services described above). Recall that the controller and the channel driver can be on separate processors.

After being released, the controller task grabs the joint position from shared memory, calculates the desired motor torque to be applied next period, stores the torque in shared memory, and becomes blocked. If the controller wrote the torque value to shared memory before the channel driver needed it in the next interval, then the cycle starts over. If the controller delayed too long, then the channel driver will shutdown that joint.

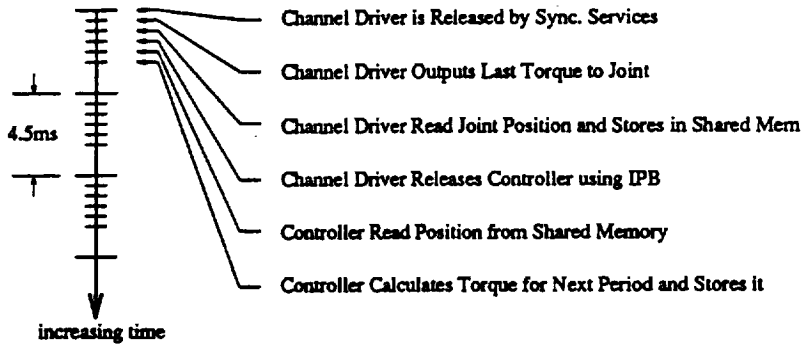

Figure 8.4: Synchronization and Data Exchange for Joint Control

8.2.6 Additional Software

This section will describe the additional software which was added to the CTOS/MCS collection in order to carry out the DMRAC experiments. The following components were added:

- *ctrlShell* – The control shell is an MCS tool which allows one to test a control algorithm on the CIRSSE Testbed with minimal interfacing work. The required data synchronous task and message handler task are provided already. To complete the controller, one simply writes the code which calculates the control law.

- *traj* – Traj is minimum jerk trajectory generator using the same algorithm as described in Section 3.5. This trajectory generator runs in the control shell and thus does not require a message handler and synchronous task as would a full blown MCS trajectory generator.

- *app* – App is the application which controls the DMRAC experiments. By using CTOS, app allows the experiments to be easily controlled from a Sun4 Workstation.

- *datalogLib* – datalogLib is a data logging library which interfaces with MCS to allow for high speed data collection within time/data synchronous tasks. This library also has the capability to upload the collected data from the MCS Cage to a Sun4 Workstation for graphical display[4].

- *matrixLib* – matrixLib is a generic linear algebra package designed to run in real time. It operates on arbitrarily sized matrices and is used extensively in the DMRAC code.

Because of the flexible and modular nature of MCS and CTOS, the creation and integration of the above additional components with the existing MCS/CTOS code was straightforward and easily accomplished.

8.2.7 Task Distribution

Table 8.2 shows the tasks and libraries as they were distributed amongst the MCS VME processors where the first grouping in the table is the CTOS tasks and the second grouping is the MCS tasks. Table 8.3 shows the tasks which were running on the Sun4 Chassis where the first grouping is the CTOS tasks and the second grouping is the custom tasks added for the data logging. The below distribution was specified in a CTOS config file. Note: At the time of the experimental runs, the

[4]The data is converted into a format which Matlab can read and display.

Table 8.2: Distribution of Libraries and Tasks Amongst the MCS Processors

cpu 0 (68030)	cpu 1 (68020)	cpu 2 (68020)	cpu 4 (68030)	cpu 5 (68030)
btsErrorSvr				
btsMsgSvr	btsMsgSvr	btsMsgSvr	btsMsgSvr	btsMsgSvr
btsCtosSvr	btsCtosSvr	btsCtosSvr	btsCtosSvr	btsCtosSvr
syncP0	syncLsph	syncLsph	syncLsph	syncLsph
btsMsgRelay				
btsBCSvr				
tidServer	tidServer	tidServer	tidServer	tidServer
msgDispatcher*	msgDispatcher*	msgDispatcher*	msgDispatcher*	msgDispatcher*
socketServer*	socketServer*	socketServer*	socketServer*	socketServer*
chasSocketSrv*				
mcsLib	mcsLib	mcsLib	mcsLib	mcsLib
chanLib	chanLib	chanLib	chanLib	chanLib
interpLib	interpLib	interpLib	interpLib	interpLib
configLib	configLib	configLib	configLib	configLib
matrixLib	matrixLib	matrixLib	matrixLib	matrixLib
datalogLib	datalogLib	datalogLib	datalogLib	datalogLib
	application	gripLib		gripLib
stateManager	gripUser	channelDriver		ctrlShell

* = Non CTOS Task.

underline = Function Library, not a Task.

MCS processor number 3 was being repaired, thus the omission in Table 8.2. Some of the names listed in Tables 8.2 and 8.3 were abbreviated to fit in the columns. Also, some of the tasks shown in the tables were not described in the preceding sections but were included for completeness.

8.3 Hardware Implementation Issues

This section will discuss some of the issued related to running the DMRAC algorithm on the actual testbed hardware. These issues include deriving the joint velocity data and addressing the computational complexity of the fully centralized DMRAC algorithm.

Table 8.3: Distribution of Tasks on Sun4 Chassis

bootstrap*
msgDispatcher*
tidServer
msgServer
btsSequencer
recServer
msgBroadcast
datalogServer

* = Non CTOS Task.

8.3.1 Deriving Velocity Information from Position Data

Recall that the DMRAC algorithm with the plant output derivatives weights, α, requires a joint velocity signal. As was mentioned earlier, the PUMA 560 robot used in the CIRSSE Testbed is not instrumented with joint tachometers, thus the velocity information must be derived from the position encoder data. This is achieved by forming a backwards difference from the position data as follows:

$$v_i(kT) = \frac{\theta_i(kT) - \theta_i(kT - T)}{T} \tag{8.1}$$

where T is the sampling period, i is the joint number, θ_i is the i^{th} joint position, and v_i is the derived velocity signal for Joint i. One problem with this velocity derivation method is that it magnifies the position noise by a factor of $(1/T)$ which yields a value of 222.2 with the DMRAC sample period of $T = 4.5\ ms$. In order to remove some of this noise, a simple first order filter of the form,

$$v_{i_{filtered}} = \frac{\omega_c T v_i(kT) + v_i(kT - T)}{\omega_c T + 1.0} \tag{8.2}$$

was used where $v_{i_{filtered}}$ is the filtered velocity for Joint i, ω_c is the cut off frequency of the filter, T is the sample time, and v_i is the unfiltered velocity signal. The cut off frequency was set to $w_c = 125\ rad/sec$, which was approximately one tenth the sampling frequency, which gave good results.

One other problem introduced by this backwards differencing is spike noise. The encoder values in the Unimation Controller are updated every 0.9 ms[5] and read every 4.5 ms by the channel drivers. It turns out that the channel driver sampling period is only 4.5 ms on the average and can deviate $\pm 10\%$ at times. This deviation will cause spikes to appear in the backwards differenced velocity signal because the difference is always divided by a constant 4.5 ms even though the period varies around 4.5 ms.

8.3.2 DMRAC Computation Complexity

Because of the centralized nature of the DMRAC algorithm, a full six joint centralized DMRAC control of a PUMA was too numerically intensive to be run on a single processor on the MCS system. The version of MCS used for the DMRAC experiments did not support the ability to easily parallelize a controller (i.e. spread it out over many processors). There were two solutions to this problem. One was to run at a slower sampling rate to allow the DMRAC algorithm more time and the other was to control a smaller subset of the joints. To run all six joints, a sampling rate of $10 - 15$ ms was required. At this slow rate, the DMRAC algorithm was unable to effectively control the arm. With this in mind, the results presented in this paper will be for the first three joints (1-3) of the PUMA at a sampling interval of 4.5 ms (same as used in the simulation runs).

8.4 Summary

In this chapter we introduced the hardware and software components comprising the CIRSSE Testbed. The hardware used for the DMRAC exerpiments included the PUMA 560 Manipulator, the Unimate controller, the MCS Cage, and the CIRSSE Computing Network. The software used included the multi-tasking Unix

[5]Because of this fact, the sampling period was chosen to be a multiple of 0.9 ms, i.e. 4.5 ms.

an VxWorks operating systems, the CTOS operating system, the Motion Control System, and some additional software needed for the DMRAC experiments. Also discussed in this chapter were some hardware implementation issues regarding the DMRAC computational complexity and the deriving of joint velocity information.

CHAPTER 9
Experimental Results

An actual PUMA 560 Manipulator was controlled using the DMRAC algorithm on the CIRSSE Testbed. This chapter will present the results of these experiments. The tuning parameter values used for all of the experimental runs are listed in Table 9.1 and are very similar to the gains used in the simulation runs. Details of the intermediate results during the tuning process will not be discussed. The tuning process is the same as that described in Section 4.1. All results will be displayed with the bias term, q_{bias} removed (see Section 2.7).

9.1 Three Joint Trajectory Tracking

This section will investigate the ability of a DMRAC controlled PUMA 560 to track two different three joint trajectories. In each case, the robot will start at the shutdown position and follow a trajectory which finished back at the shutdown position.

9.1.1 First Trajectory

The first trajectory is listed in Table 9.2 and is illustrated by Figure 9.1 where the numbered positions refer to the knot points in the table. This trajectory is very similar to the one used in the simulation (Section 5.1.1). The arm first moves to a straight up position, curls up, and then moves back to the safe position. The wrist joints remain locked in their shutdown positions of $\{0.0, 45.0, 90.0\}$ degrees.

The response to the first trajectory is shown in Figure 9.2. The response is quite good. The effects of stiction can be seen on Joint 2 at $t = 15 seconds$ in Figure 9.2(b). Figures 9.3–9.5 show the model following error and the link torques for Joints 1, 2, and 3 respectively. Figure 9.3(b) shows that the Joint 1 torque signal

Table 9.1: Parameter Values for 3 Joint Trajectory Tracking Runs

T_{pro} (diag component)	"e_z"	20	40	40
	"x_m"	140	20	200
		30	200	30
	"u_m"	140	200	200
T_{int} (diag component)	"e_z"	30	60	40
	"x_m"	200	30	400
		60	400	60
	"u_m"	200	400	400
joint		1	2	3
Model	w_n	10	10	10
	ζ	1	1	1
Feed Forward	K_d	6	6	6
	τ	0.05	0.05	0.05
alpha	α	0.02	0.02	0.02

Table 9.2: First Three Joint Tracking Test Trajectory

Knot Point	Joint Positions (deg)			Time (sec)
	1	2	3	
0	0	-45	180	-
1	-90	-90	90	6
2	0	0	180	8
3	0	-45	180	6

was quite noisy. This noise did not have a physically detectable effect on the actual arm motion. Typically one can feel or hear a noisy torque signal on the actual arm. The peak tracking errors for the three joints are listed in Table 9.3.

The stiction effect mentioned above for Joint 2 can also be seen in Figure 9.4(a) at $t = 15sec$ near the 'X' at the peak error location. When stiction grabs a joint, the error ramps up as does the torque (Figure 9.4(b)).

9.1.2 Second Trajectory

The second trajectory is listed in Table 9.4 and is illustrated by Figure 9.6. The arm first moves to an upright L position, stretches out horizontally, and then

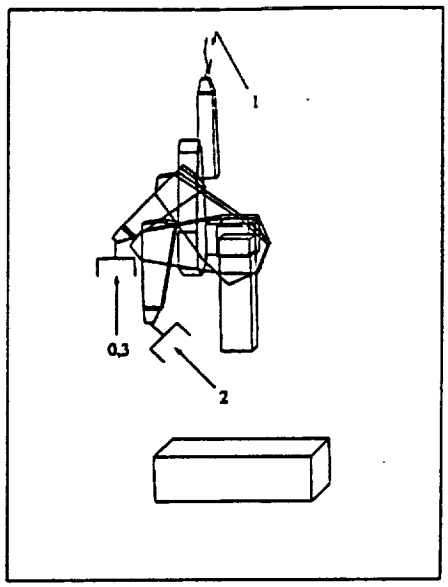

Figure 9.1: First Three Joint Tracking Test Trajectory

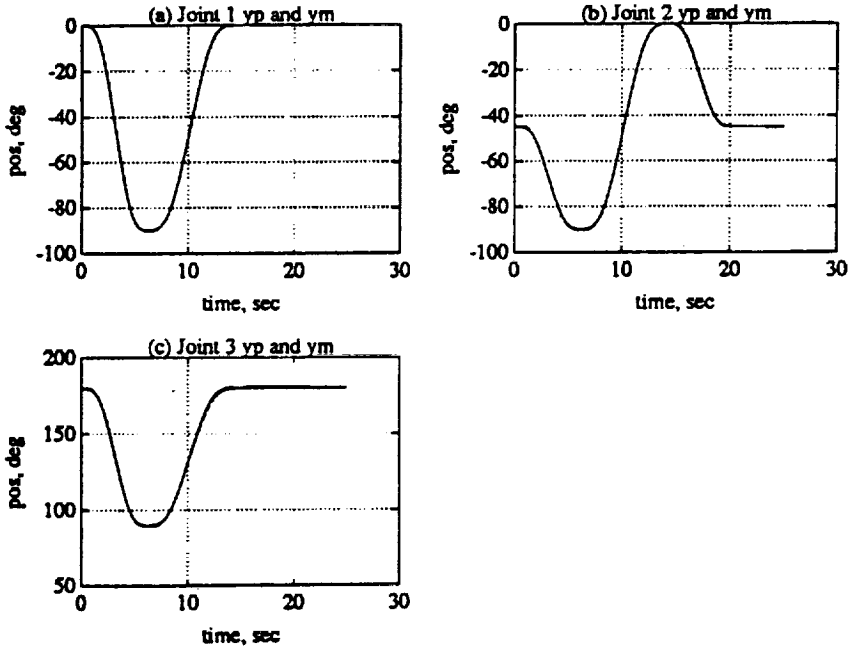

Figure 9.2: Plant and Model Output for First Trajectory. (a) Joint 1. (b) Joint 2. (c) Joint 3.

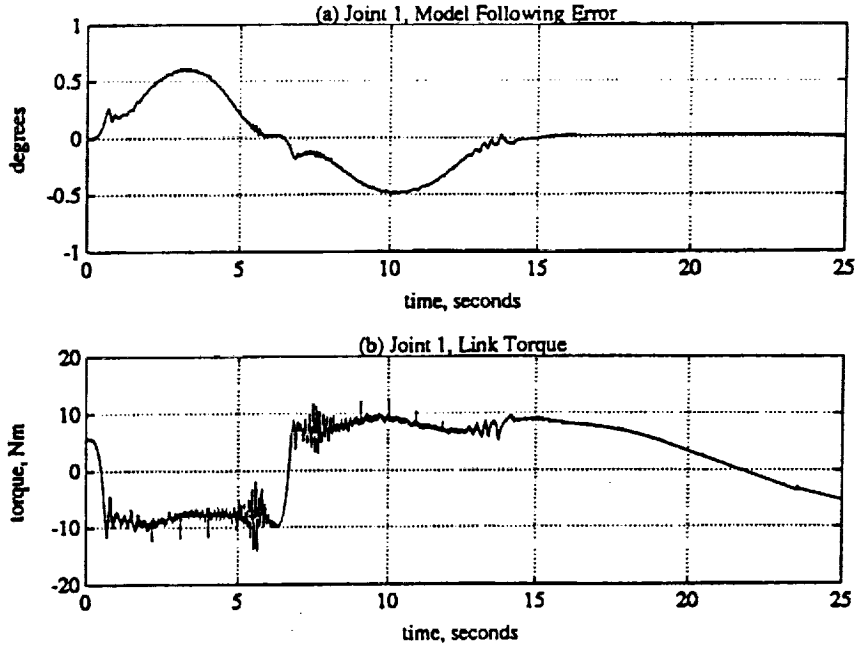

Figure 9.3: Joint 1 Data for First Trajectory. (a) Model following error. (b) Joint Torque.

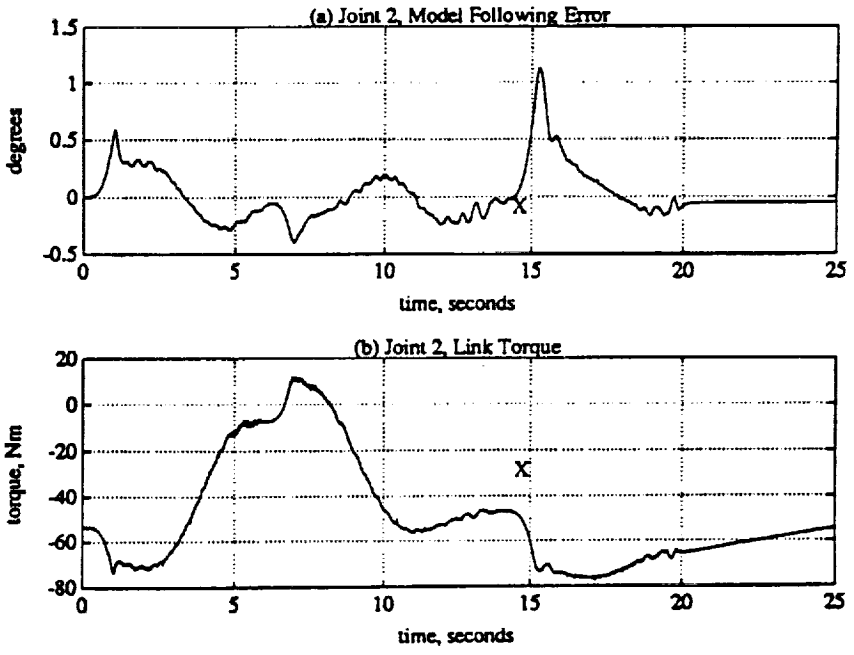

Figure 9.4: Joint 2 Data for First Trajectory. (a) Model following error. (b) Joint Torque.

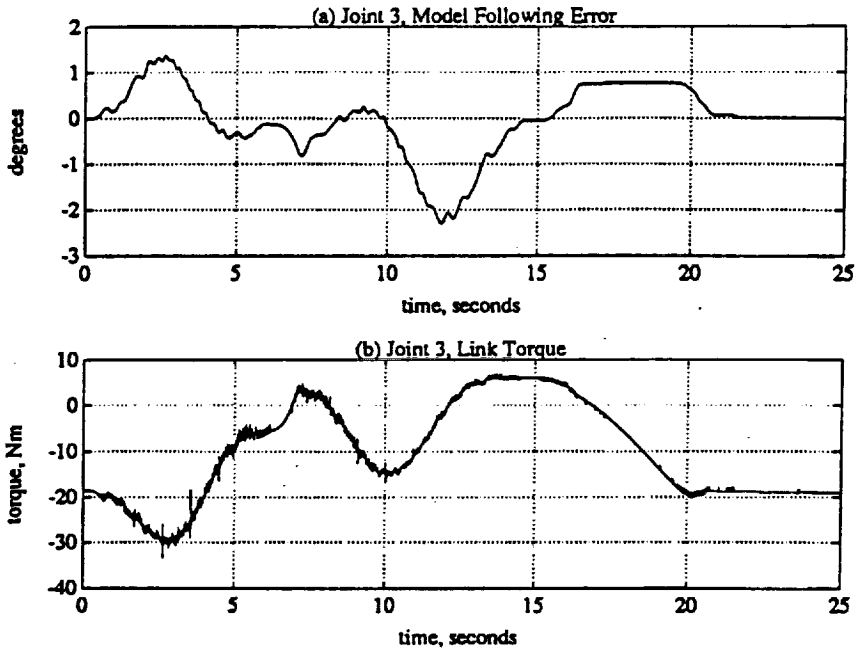

Figure 9.5: Joint 3 Data for First Trajectory. (a) Model following error. (b) Joint Torque.

Table 9.3: First Trajectory Peak Tracking Errors

Joint	Peak Error (degrees)
1	0.6038
2	1.1233
3	-2.2892

Table 9.4: Second Three Joint Tracking Test Trajectory

Knot Point	Joint Positions (deg) 1	2	3	Time (sec)
0	0	-45	180	-
1	45	-90	180	5
2	45	0	90	8
3	0	-45	180	7

Table 9.5: Second Trajectory Peak Tracking Errors

Joint	Peak Error (degrees)
1	0.4437
2	1.8734
3	-3.6948

moves back to the safe position. The wrist joints remain locked in their shutdown positions of $\{0.0, 45.0, 90.0\}$ degrees.

The response to the second trajectory is shown in Figure 9.7. The response is acceptable. Again, there are stiction effects visible in Joints 2 and 3 near $t = 15sec.$ Figures 9.8–9.10 show the model following error and the link torques for Joints 1, 2, and 3 respectively. The 'X's in Figures 9.9 and 9.10 show where the stiction force is exceeded. Also visible in the two figures is a steady state error beginning at $t = 20seconds.$ Notice how the torque slowly winds up from $20 \leq t \leq 25$. If the experiment was continued, eventually the torque would windup to a point where the stiction would break and then "re-stick" causing a slow limit cycle. The peak tracking errors for the three joints are listed in Table 9.5. As with the previous case, the peak errors were caused by torque windup due to stiction in the joints.

9.2 Static Load Changes

This section will test the ability of the DMRAC algorithm to adjust to static load variations. The same trajectory will be run with different loads in the gripper. The algorithm will first be allowed to adjust to the load and then the trajectory will

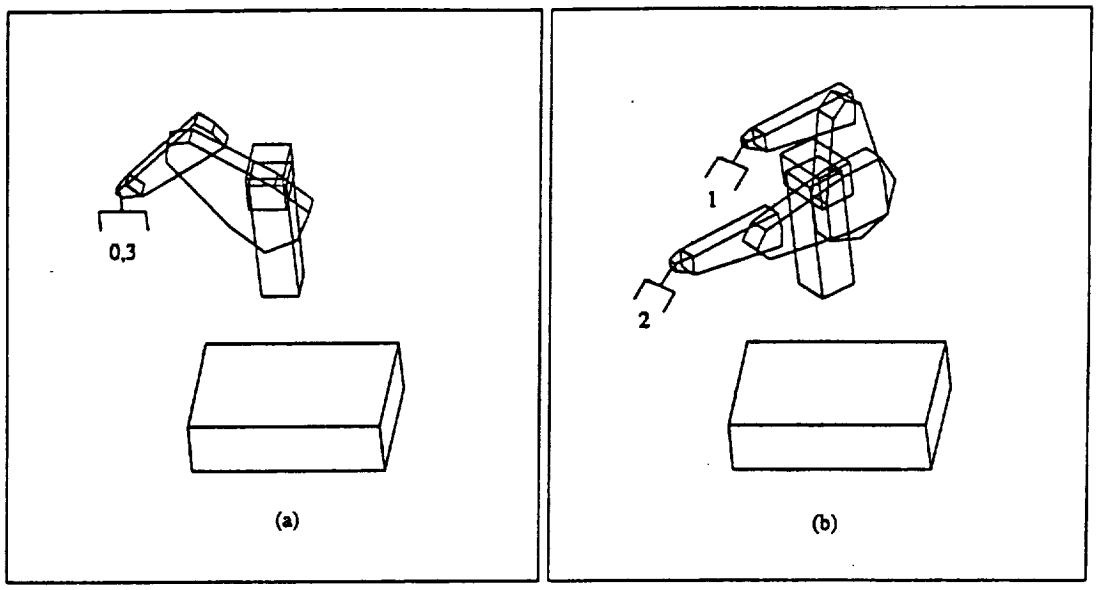

Figure 9.6: Second Three Joint Tracking Test Trajectory

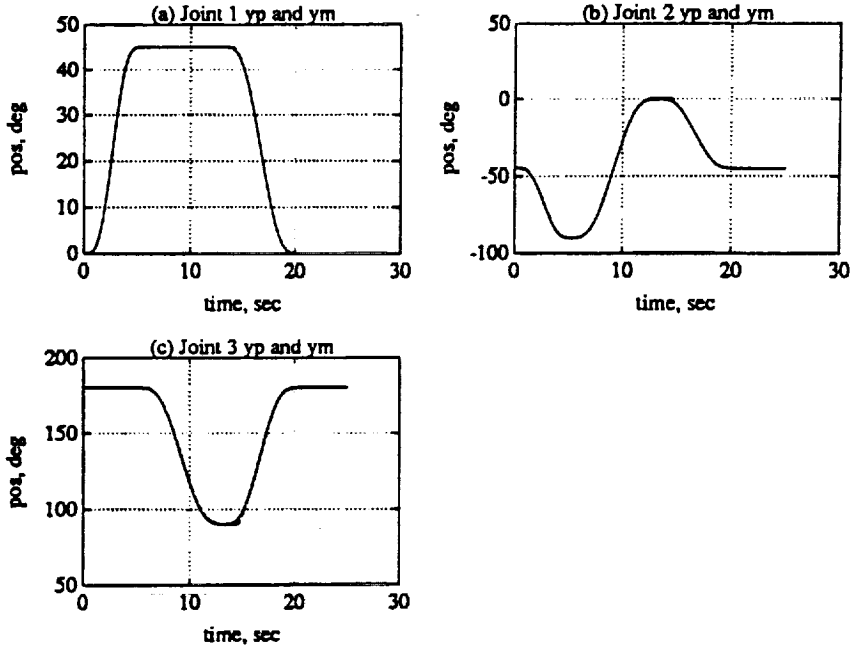

Figure 9.7: Plant and Model Output for Second Trajectory. (a) Joint 1. (b) Joint 2. (c) Joint 3.

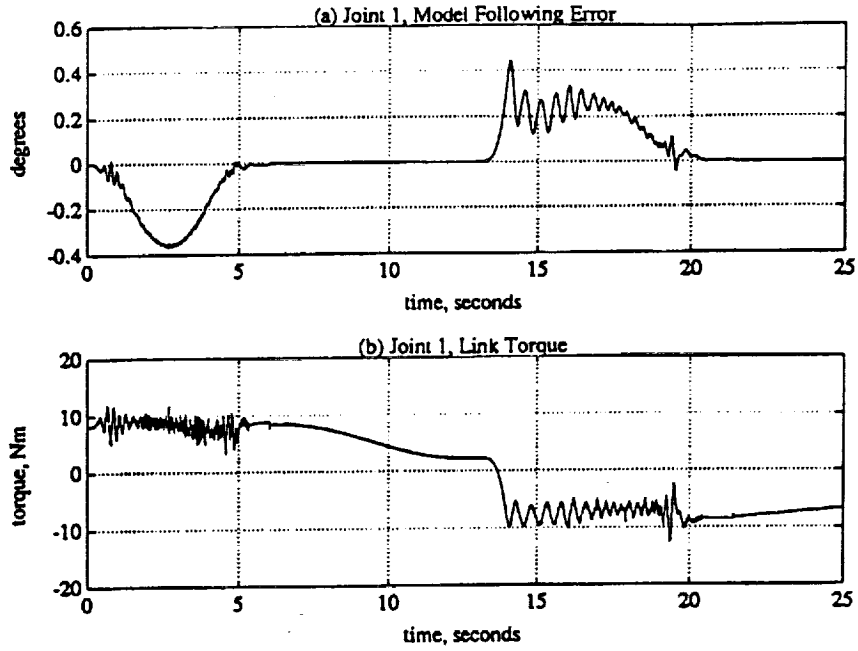

Figure 9.8: Joint 1 Data for Second Trajectory. (a) Model following error. (b) Joint Torque.

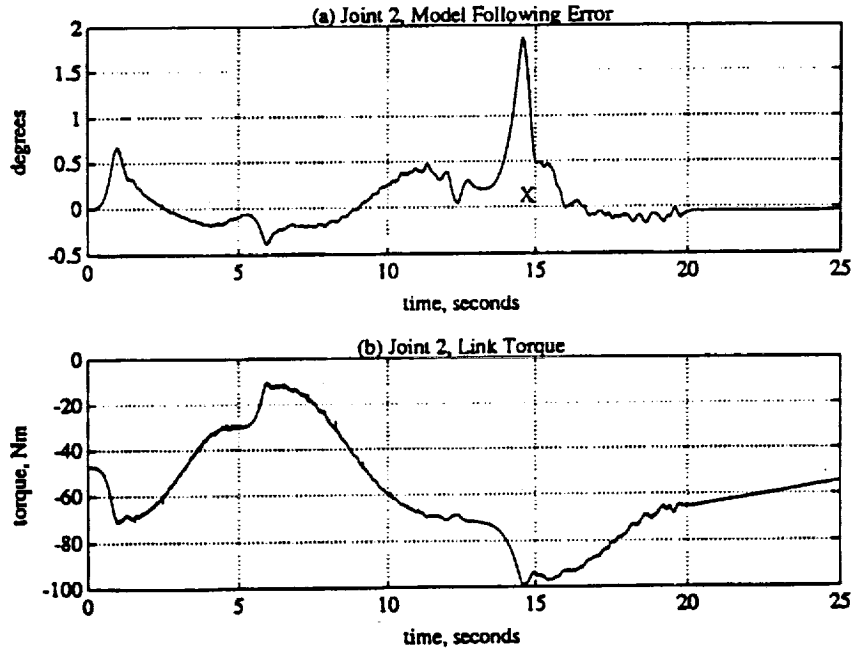

Figure 9.9: Joint 2 Data for Second Trajectory. (a) Model following error. (b) Joint Torque.

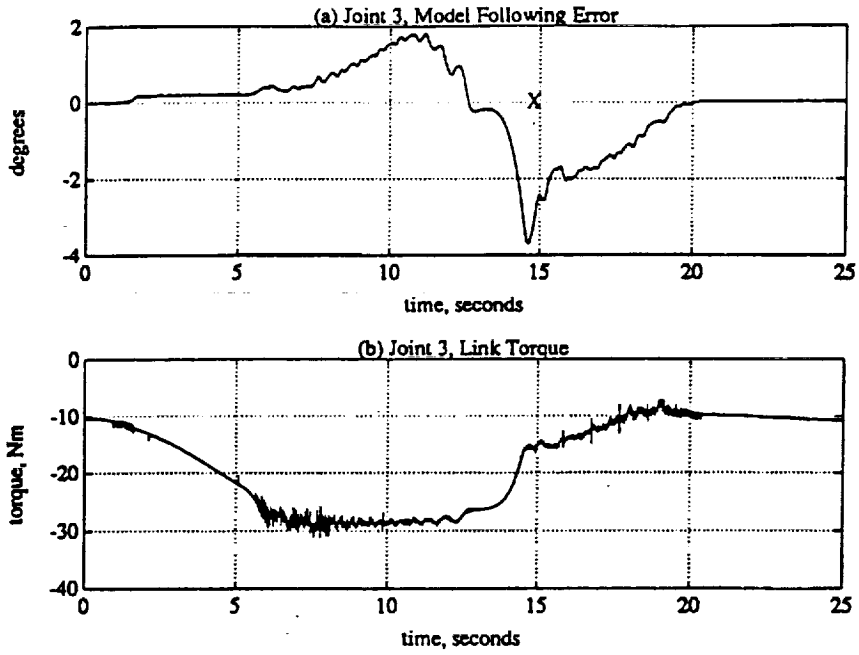

Figure 9.10: Joint 3 Data for Second Trajectory. (a) Model following error. (b) Joint Torque.

be started. The trajectory used is listed in Table 9.6 and is illustrated in Figure 9.11. The wrist joints remained locked in their shutdown positions of $\{0.0, 45.0, 90.0\}$ degrees. Five different loads were run for the trajectory - $0kg$, $1kg$, $2kg$, $3kg$, and $4kg$.

Figures 9.12 and 9.13 show the response for Joints 2 and 3 respectively. The numbers on the plots are to help identify which curve represents which payload. For Joint 3, the peak errors vary from 2.4390 degrees for the no load case to 3.9972 degrees for the $4kg$ load case. The load changes make up only about 50% of the error. The other 50% is due to the adaptation to the changing arm dynamics. For Joint 2, the peak errors are around 0.8 - 1.0 degrees. As with Joint 3, the portion of the error due to the load change for Joint 2 is small compared to the no load case. Note: It is not so important to distinguish each individual error trace as it is the see the trend as the load is changed.

Table 9.6: Static Load Change Trajectory

Knot Point	Joint Positions (deg)			Time (sec)
	1	2	3	
0	0	-45	180	-
1	0	-45	180	3
2	45	0	0	10
3	0	-45	180	10

For Joint 1, the error signals did not vary by more than 0.1 degrees between the five different load cases. Figure 9.14 shows the model following error for the $4kg$ load case.

The joint torque signals for Joint 2 are shown in Figure 9.15. Figure 9.16(b) shows the torque signal for the $4kg$ load for Joint 3. Notice the spikes in the torque signal which are caused by the backwards differencing process used to derive the velocity information (Section 8.3.1).

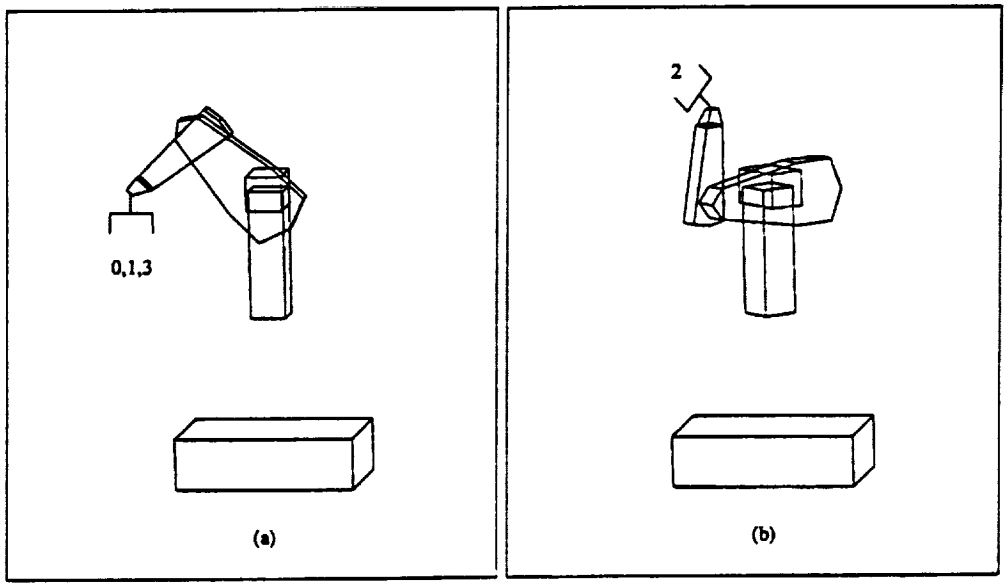

Figure 9.11: Static Load Change Trajectory

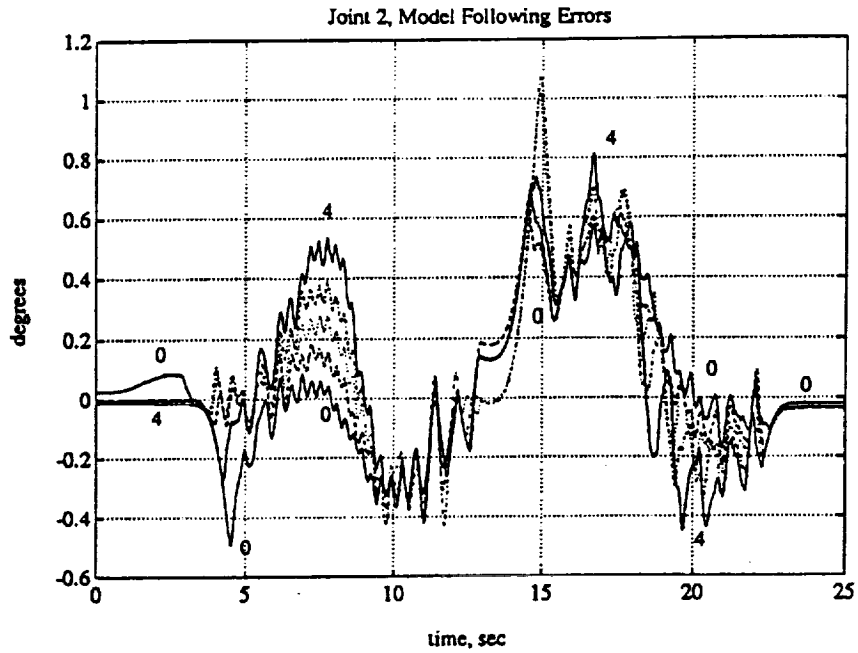

Figure 9.12: Joint 2 Static Load Model Following Error

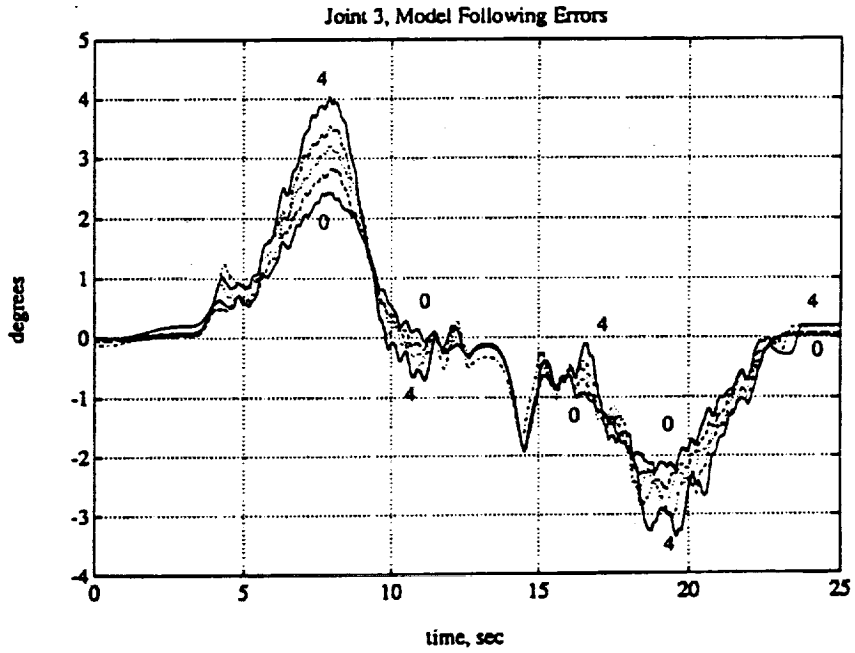

Figure 9.13: Joint 3 Static Load Model Following Error

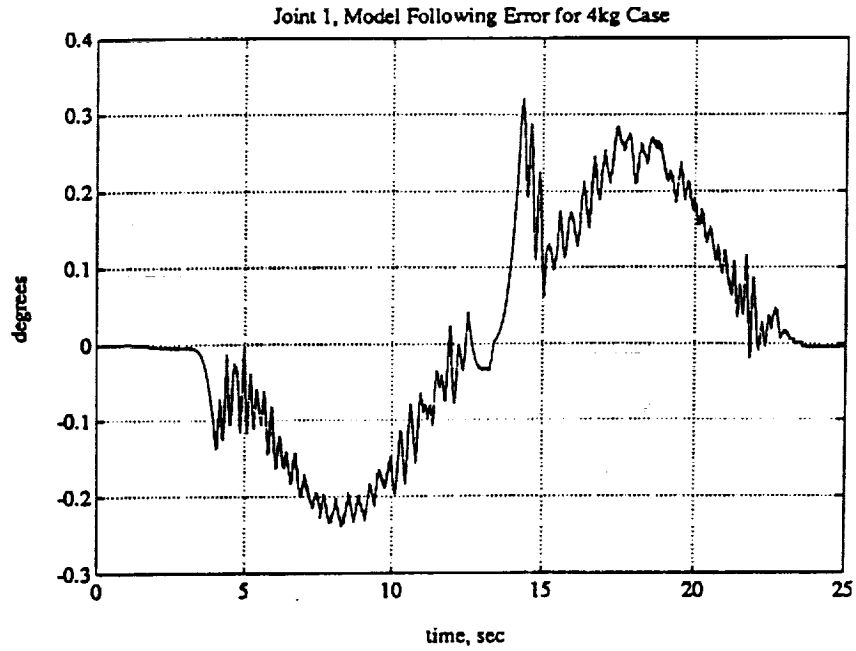

Figure 9.14: Joint 1 Static Load Model Following Error for $4kg$ Case Only

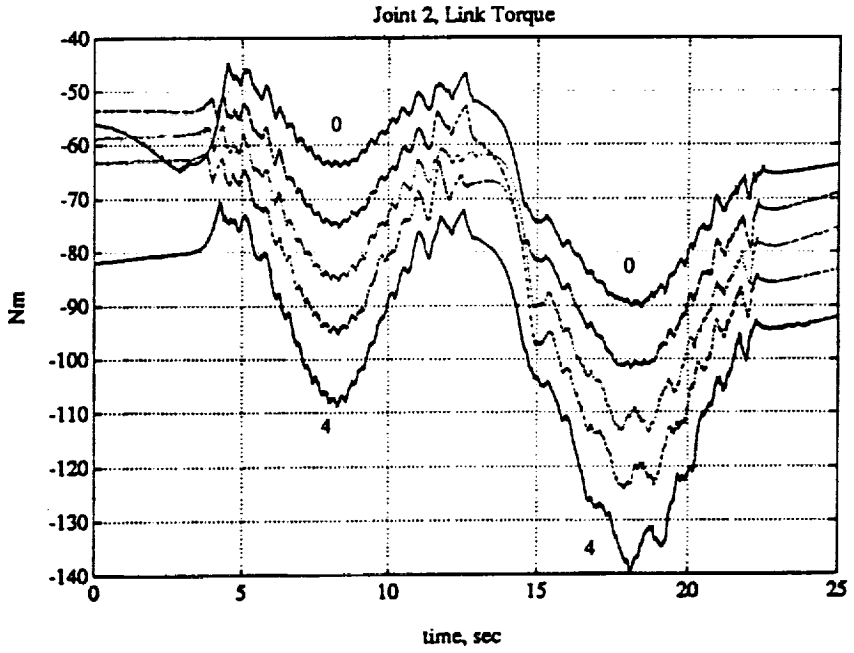

Figure 9.15: Joint 2 Static Load Torque Signal

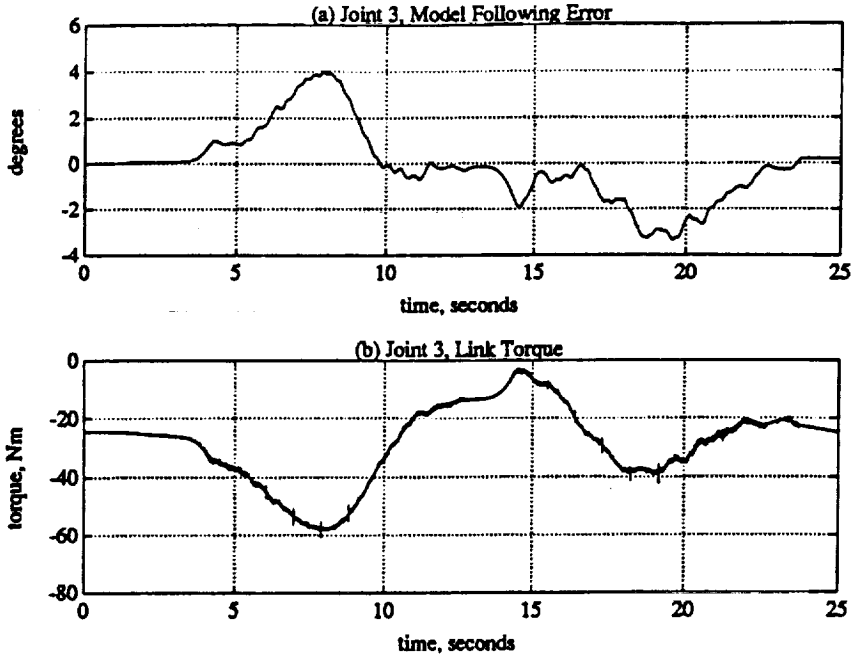

Figure 9.16: Joint 3 Static Load Torque Signal for $4kg$ Load. (a) Model following error. (b) Joint Torque.

9.3 Dynamic Load Changes

This section will test the ability of the DMRAC algorithm to adjust to dynamic load variations. While running the same trajectory, various loads will be added to the gripper while the robot is in motion. The same loads used in the previous section were employed. The trajectory used is listed in Table 9.7 and is illustrated in Figure 9.17. The wrist joints remained locked in their shutdown positions of $\{0.0, 45.0, 90.0\}$ degrees. Note: The $1kg$ and $4kg$ loads were added at about $t = 6.76$ seconds and the $2kg$ and $3kg$ loads were added at about $t = 7.34$ *sec.*

Figure 9.18 shows the model following error for Joint 2 for all of the loads. The numbers on the graph indicate which peaks in the error plots match up with the various loads. This figure shows that the DMRAC algorithm has a good load disturbance rejection. The transient period only lasts about 2 seconds. The peak errors at the time of the load addition are listed in Table 9.8.

Table 9.7: Dynamic Load Change Trajectory

Knot Point	Joint Positions (deg)			Time (sec)
	1	2	3	
0	0	-45	180	-
1	0	-45	180	3
2	45	-90	90	10
3	0	-45	180	10

Table 9.8: Joint 2 Peak Errors for Dynamic Load Case

Load (kg)	Peak Error at Time of Load Addition (degrees)
0	-0.0018
1	0.3651
2	0.5712
3	0.7690
4	1.2843

Figure 9.19 shows the error for Joint 3 for the various loads. Joint 3 suffers more with a load disturbance having a peak error of almost 5 degrees when the $4kg$ load is added. Again, the transient period is roughly 2 seconds. After the transient, good tracking performance is achieved with the additional loads. Table 9.9 lists the peak errors at the time of the load additions.

As with the static load case, the model following errors for Joint 1 did not vary by more than 0.1 degrees. Figure 9.20 shows the worst Joint 1 tracking error which occurred with the $4kg$ load. The peak tracking errors for Joint 1 are all within a quarter of a degree.

The worst case errors resulted from the $4kg$ weight. Figure 9.21 shows the plant (solid line) and model (dashed line) outputs for this $4kg$ load case.

9.4 Other Testbed Runs

This section will investigate two more runs on the hardware. The first run was to show the effects of stiction on steady state error. The second run was to

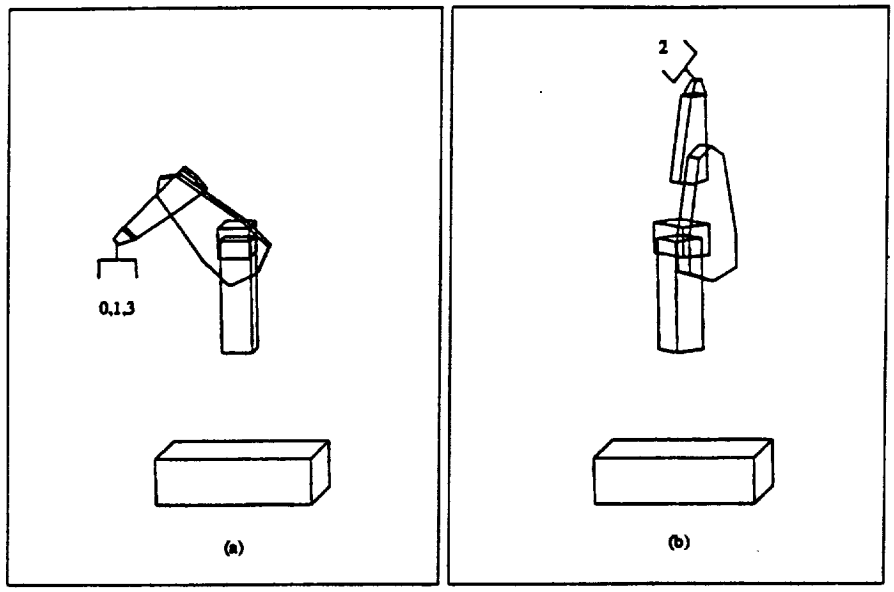

Figure 9.17: Dynamic Load Change Trajectory

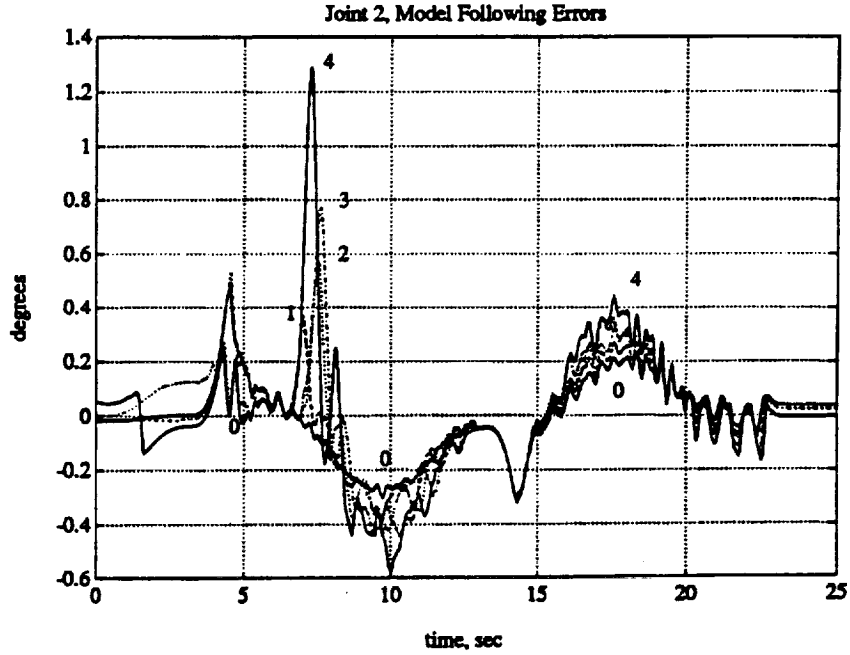

Figure 9.18: Joint 2 Dynamic Load Model Following Errors

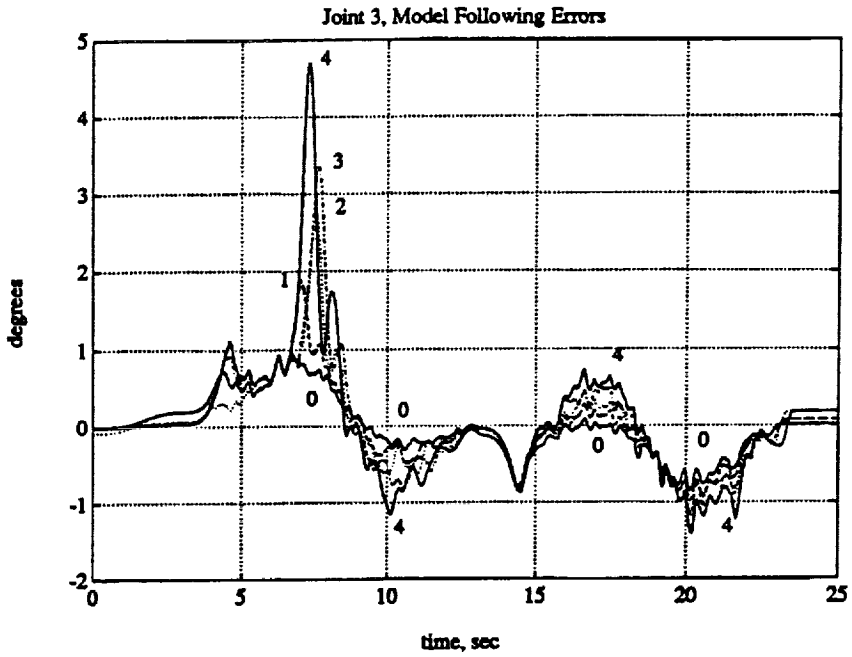

Figure 9.19: Joint 3 Dynamic Load Model Following Errors

Table 9.9: Joint 3 Peak Errors for Dynamic Load Case

Load (kg)	Peak Error at Time of Load Addition (degrees)
0	0.8583
1	1.8683
2	2.7772
3	3.3544
4	4.6962

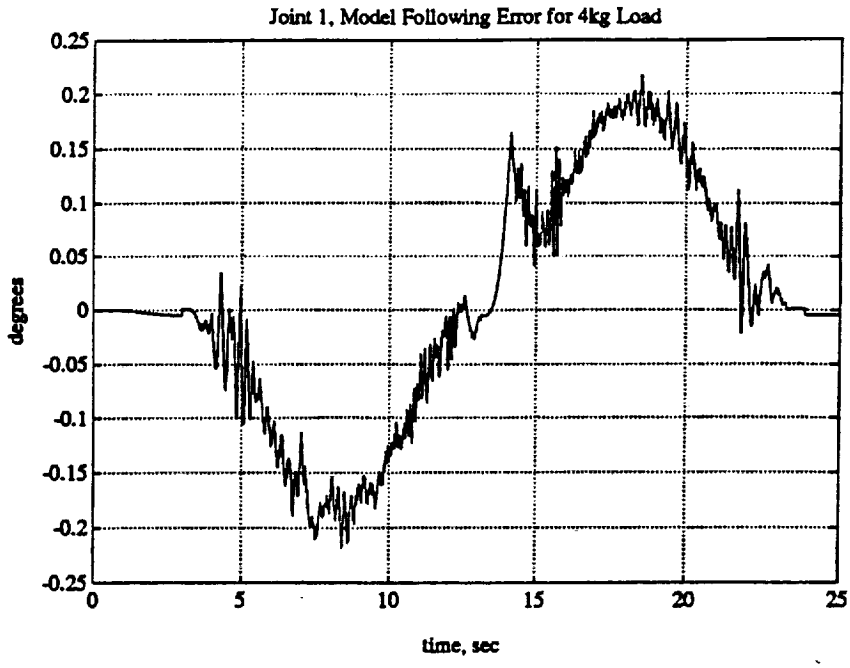

Figure 9.20: Joint 1 Dynamic Load Model Following Error for $4kg$ Load

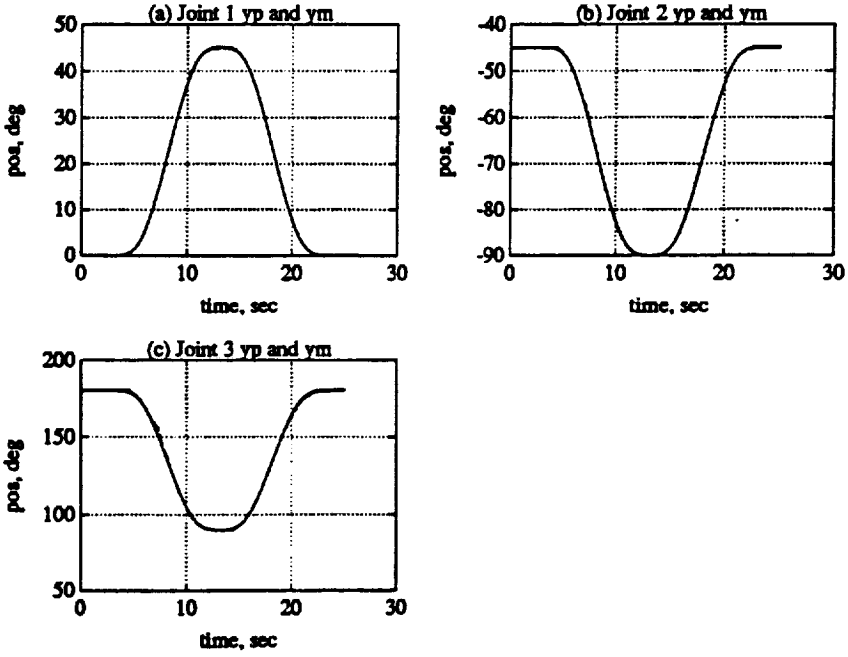

Figure 9.21: Plant and Model Outputs for $4kg$ Dynamic Load Change. (a) Joint 1. (b) Joint 2. (c) Joint 3.

investigate the disturbance rejection properties of the DMRAC algorithm.

9.4.1 Stiction Effects on Steady State Model Following Error

For this run, the arm was started in the shutdown position and commanded to stay in that position. Figures 9.22–9.24 show the model following error and the joint torque signal for Joints 1, 2, and 3 respectively. The error peak present on all joints for the first 5 seconds of the run were caused by the joints not being exactly at the shutdown position. The trajectory generator moved the robot from its starting position to the shutdown position along a 3 second minimum jerk trajectory segment causing the error peak.

Figure 9.22(a) shows the stiction breaking at $t = 25.9$ *sec.* Prior to the break in stiction, the torque signal ramps up due to the negative steady-state error, Figure 9.22(b), and then ramps down after the break due to a positive error caused by the joint sticking again. Figure 9.23(a) shows a stair-step release-grab sequence starting at $t = 15$ *sec.* It is obvious that stiction causes long limit cycles in this implementation. To determine the cycle period, much longer runs would need to be logged.

When a steady-state error exists, the integral portion of the K_e adaptive gain will ramp up because it is formed by the weighted product of $e_z e_z$. Thus, because of stiction there is a persistent steady-state error which will cause K_e to slowly build up. This was not a problem due to the short duration of the experiments. For longer experiments, the integral terms in K_e should be periodically reset.

9.4.2 Disturbance Rejection

This section will investigate the disturbance rejection abilities of the DMRAC algorithm. As in the previous section, the arm was commanded to stay at the shutdown position. After the run was started, Joints 1, 2, and 3 were disturbed in

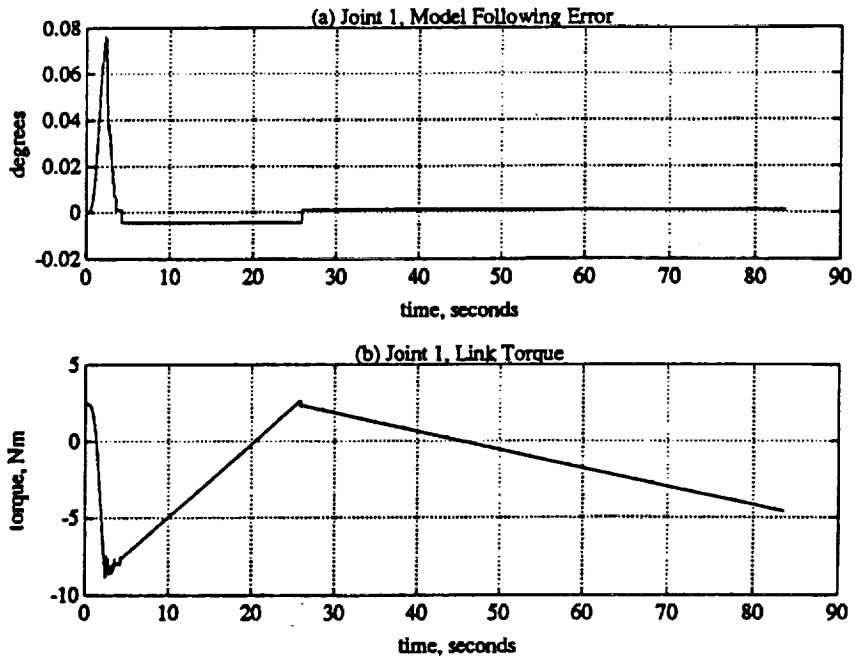

Figure 9.22: Joint 1 Stiction Effects. (a) Model following error. (b) Joint torque.

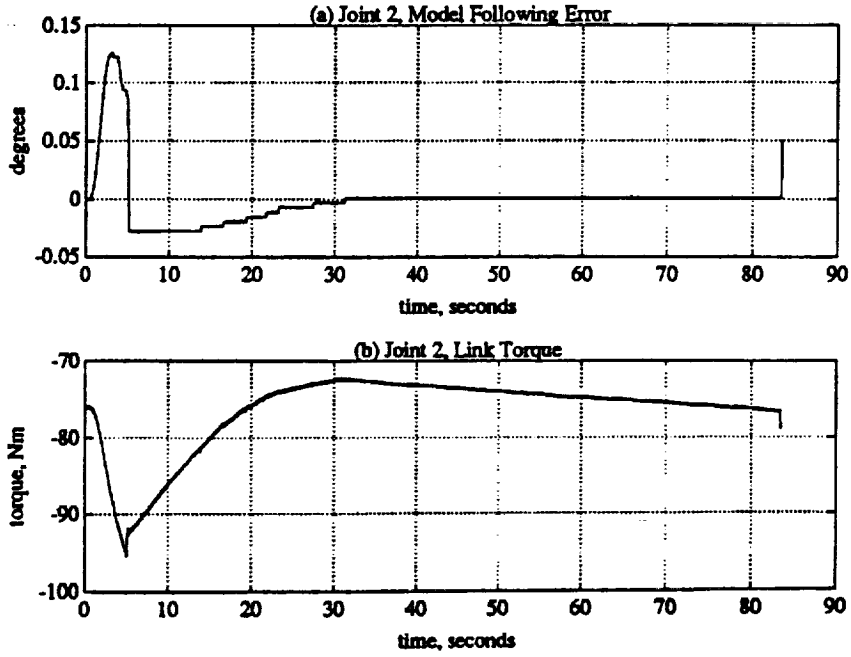

Figure 9.23: Joint 2 Stiction Effects. (a) Model following error. (b) Joint torque.

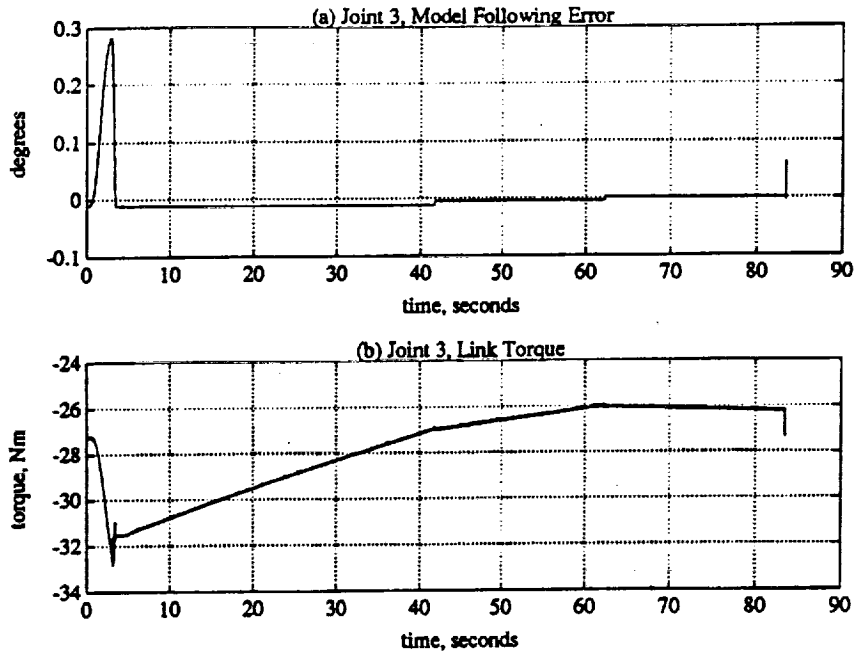

Figure 9.24: Joint 3 Stiction Effects. (a) Model following error. (b) Joint torque.

Table 9.10: Times of Disturbance Application

Joint	Time of Application of Disturbance
1	4.6566
2	8.5904
3	11.7196

succession by pushing hard on the manipulator. Table 9.10 shows the times that the disturbances were applied to the joints.

Figure 9.25 shows the tracking performance with the disturbances. For all joints, there was a fast recovery with a fast over shoot followed by a decay back to steady-state. The algorithm also does a good job of isolating the disturbances from the other joints. Figures 9.26–9.28 show the model following error and torque signals for the individual joints.

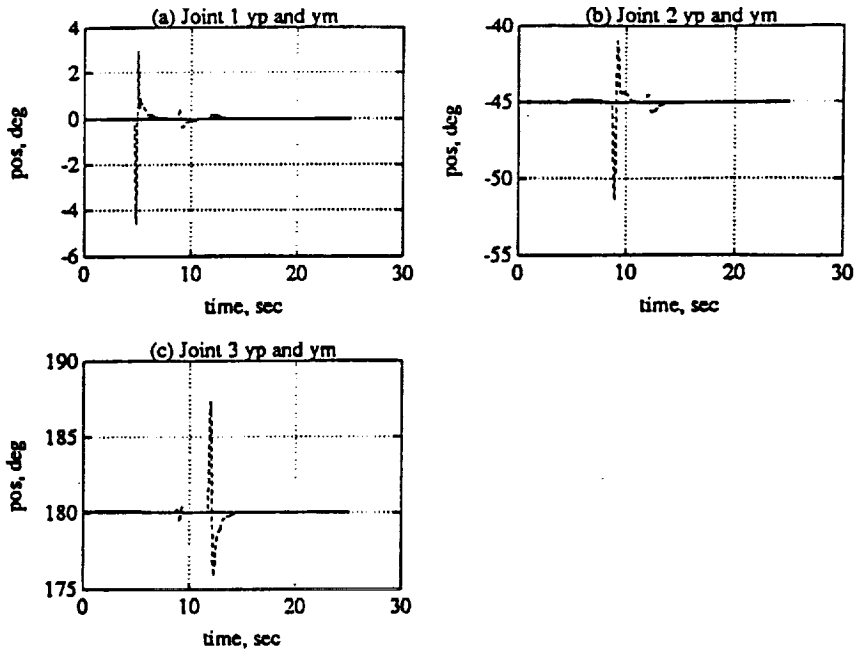

Figure 9.25: Disturbance Rejection Run. (a) Joint 1 position. (b) Joint 2 position. (c) Joint 3 position.

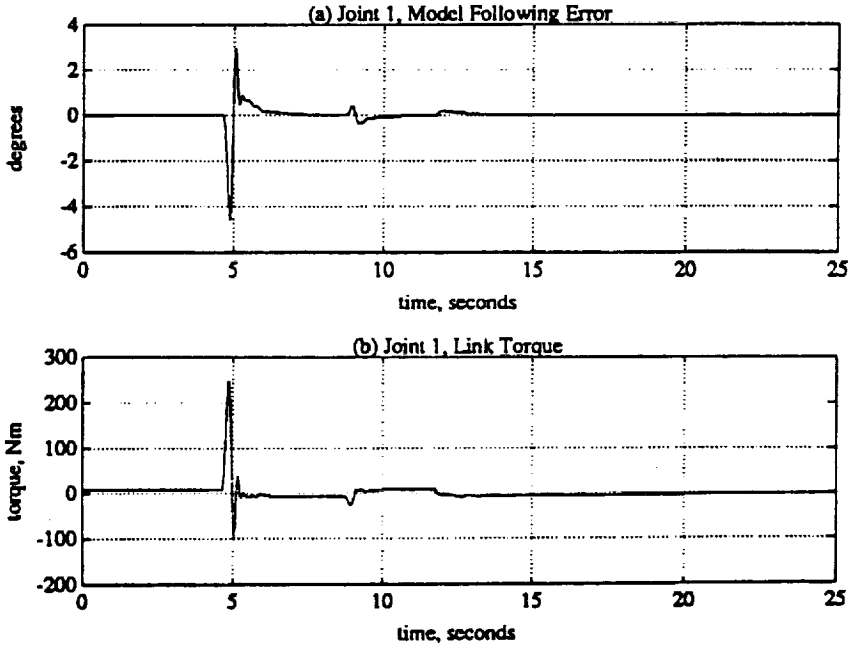

Figure 9.26: Joint 1 Response to Disturbance Rejection Run. (a) Model following error. (b) Joint torque.

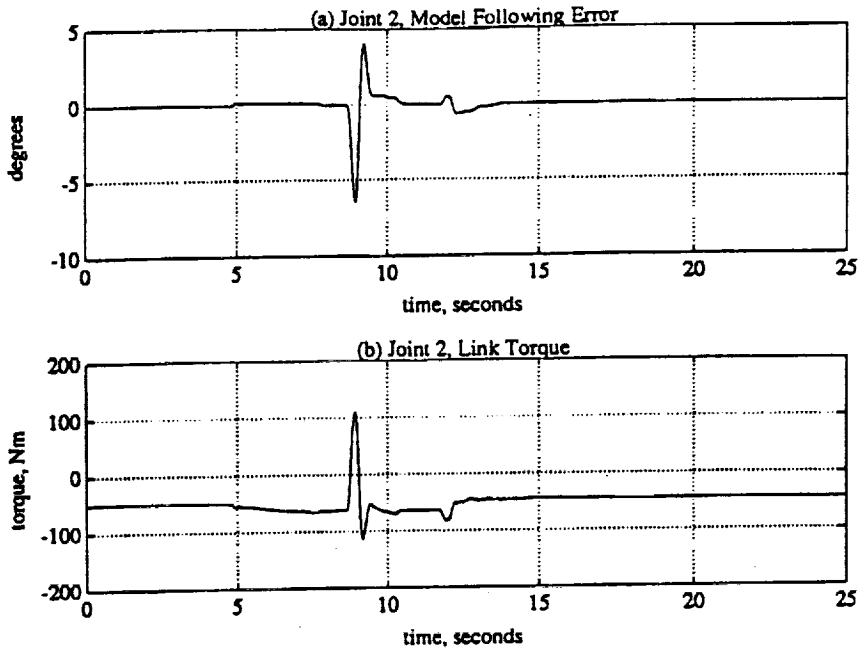

Figure 9.27: Joint 2 Response to Disturbance Rejection Run. (a) Model following error. (b) Joint torque.

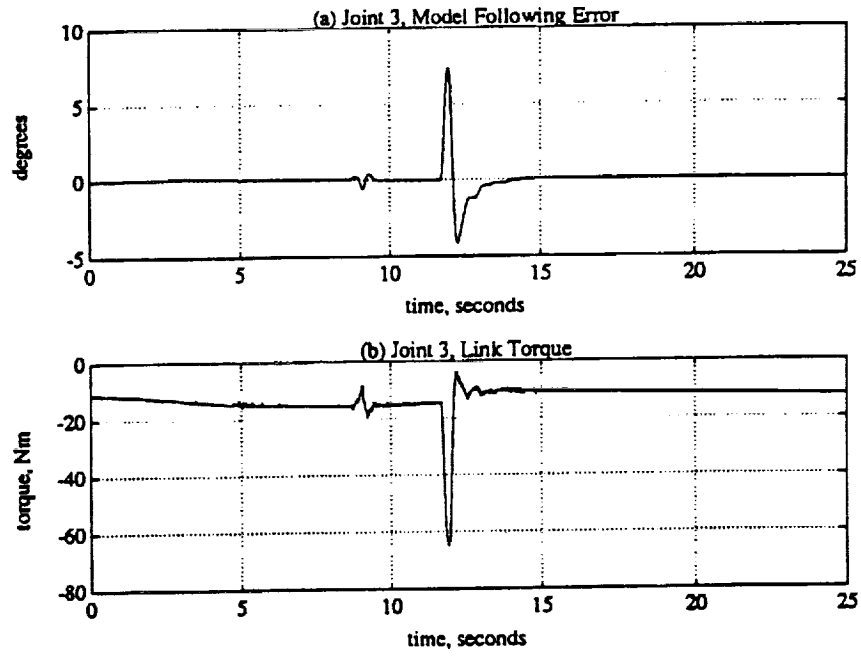

Figure 9.28: Joint 3 Response to Disturbance Rejection Run. (a) Model following error. (b) Joint torque.

9.5 Summary

This chapter presented the experimental results from the Direct Model Reference Adaptive Control of a PUMA 560 in the CIRSSE Testbed. First, the robot was controlled along two three-joint trajectories with acceptable tracking errors. Next, the robot was controlled in the presence of static and dynamic load changes. The DMRAC algorithm adapted quite well to these load changes. It was found that the effects of stiction had the most dramatic effect on the model following error typically causing over 50 % of the tracking error due to integral wind-up. Next, the stiction effects on the steady state error were investigated. Finally, the DMRA controlled PUMA was subjected to some disturbances which were handled nicely.

CHAPTER 10
Conclusions and Future Research

10.1 Summary and Conclusions

This project dealt with the control of a PUMA 560 Robotic Manipulator using a Direct Model Reference Adaptive Controller scheme. We first discussed the benefits of using a DMRAC algorithm some of which were an asymptotically zero output error, bounded states, multiple input-multiple output plant support, and the fact that adaptive observers and full state feedback are not required. Then the history of the DMRAC algorithm was briefly presented beginning with the basic algorithm of Sobel, Kaufman, and Mabius [9] and proceeding to the Kaufman, Neat, and Steinvorth algorithm [5]. The goal of the project was then stated which was to test the ability of a DMRAC algorithm to control a PUMA 560 robot with an interest in the ability to adapt to sudden load changes.

We then proceeded to present to development of the DMRAC algorithm. First, the motivating CGT concept was introduced which assumes that the plant parameters are known. These CGT concepts assume that there exists an ideal plant with ideal state and input trajectories which occur when there is perfect reference model output tracking. It was then shown that the control for the perfect output following case is a linear combination of the model state and input, $u_p = S_{21}x_m + S_{22}u_m$. When perfect output following does not occur, a stabilizing output feedback was added. The control was then seen to be, $u_p = S_{21}x_m + S_{22}u_m + K(y_m - y_p)$. This control law was then used to motivate the basic DMRAC algorithm as follows: $u_p = K_x x_m + K_u u_m + K_e(y_m - y_p)$, where K_x, K_u, and K_e are adapted by (2.31)–(2.33). This basic DMRAC law will produce asymptotic output tracking if

C-3

the proportional and integral weighting matrices in (2.31)–(2.33) are positive semi-definite and positive definite, respectively, and the plant under control is Almost Strictly Positive Real.

The ASPR condition on the plant can be restrictive, so BarKana and Kaufman proposed augmenting non-ASPR plants with supplemental feed-forward dynamics such that the augmented plant becomes ASPR and the above results hold for the augmented output. Unfortunately, a steady-state error will be present for plants which are not high gain feedback stabilizable. To remedy this, Kaufman, Neat, and Steinvorth proposed augmenting the model dynamics with the same feed-forward filter, thus eliminating the steady-state error.

Some further modifications to the Kaufman, Neat, and Steinvorth algorithm were to inject a weighted plant output derivative term into the plant output signal. This had the effect of damping out some high frequency oscillations at the expense of a slightly increased tracking error during transients. Also, a bias term was subtracted from the plant output and the model input to effectively shift the coordinate system and thus provide excitation for the adaptation process throughout the range of interest. The bias addition is necessary when the state space origin is not an equilibrium, as with the control of some non-linear plants. The algorithm was then discretized for simulation and implementation on the CIRSSE Testbed. The feed-forward and reference model dynamics were discretized exactly while the adaptation mechanism was discretized using backwards rectangular integration.

We then described the simulation environment which was used to test the DMRAC algorithm before it was implemented on the actual robot. We detailed the sequence of execution for the simulation and discussed the creation of an accurate simulation model of the PUMA 560. A minimum jerk trajectory generator was also discussed.

Next, a tuning process was described and carried out on the PUMA 560 in

simulation. The process consisted of beginning with a default set of tuning parameters and simulating small step inputs with the plant initially at an equilibrium. The tuning parameters were changed until a satisfactory step response was obtained at which time the parameters were fine-tuned using typical reference inputs.

Once tuned, the algorithm was used to control the simulated PUMA 560. The simulations were run with trajectories which subjected each joint to its extreme operating conditions such as maximum/minimum inertia seen at the joint and maximum/minimum gravity loading. The response of each joint to these trajectories was quite satisfactory. The peak errors for the first three joints were typically within $\pm$ 2.0 degrees with the average error within about $\pm$ 1.0 degree. For the wrist, the peak error were typically within $\pm$ 1.0 degree.

We next controlled the simulated robot over three typical minimum jerk six-joint trajectories. The peak model following errors were typically between 1.8 and 0.3 degrees. The torque signals were smooth and bounded. For the third trajectory, the Joint 6 torque signal saturated for a small time interval. Even with the Joint 6 command saturated, the rest of the joint signals remained bounded.

The effects of changing the tuning parameters were illustrated by stepping through the various parameters and changing them above and below their nominal values and comparing the simulation results. The effects of changing the adaptive weighting matrices was shown and it was noted that the weights associated with the reference model state vector and input had the greatest affect on the tracking performance. The reference model undamped natural frequency was adjusted showing the trade off between error signal overshoot and settling time. The effects of changing the feed-forward filter parameters was illustrated. It was also shown how the plant output derivative term weights are used to damp out any high frequency oscillations which may be present in control signals. It was noted that adjusting these output derivative weights too high will actually produce oscillations and cause

instability. Finally, the effects of removing the model and/or plant feed-forward dynamics was investigated. Removal of the model feed-forward term resulted in a process which was very difficult to tune. Removal of both feed-forward terms from the plant and model added oscillations into the response but still produced an acceptable performance.

The ability of the DMRAC algorithm to adjust for load variations was then investigated. Two types of load variations were considered, static and dynamic. The static load variation simulations allowed the algorithm to adapt to the load and then applied a typical trajectory to the arm which tested the ability of the DMRAC algorithm to control a loaded down manipulator without the transient effects. The response to the static load cases was quite encouraging. It was found that the added load mass had a small affect on the tracking performance for all joints except Joint 2 which sees the highest inertia and gravity change. It was found that the change in the gravity loading caused by the load had a larger affect on the error then the change in the inertia loading. The dynamic load variation simulations investigated the ability of the DMRAC algorithm to compensate for a sudden load change occurring while the robot was in motion. It was found that the algorithm had well behaved asymptotic tracking capabilities in the presence of load changes. The torque signals all remained bounded.

One problem with the DMRAC algorithm is that the reference model typically introduces a lag between the model input and the model output, thus, the trajectory tracking error ($y_p - u_m$) may be vary large. Two methods to over come this problem were investigated. The first involved adding dynamics between the trajectory generator and the reference model which forced the model output to match the desired trajectory. The second method involved increasing the speed of the reference model such that the lag was reduced to some acceptable value. Both methods produced acceptable results. Typical trajectory tracking errors were reduced by about 80

From the above simulation results, we gained confidence that the DMRAC algorithm could be an effective controller for the PUMA 560 Manipulator. We then proceeded to describe the CIRSSE Testbed environment which would be used to test the DMRAC algorithm on an actual PUMA Manipulator. The hardware and software components of the testbed were discussed along with some implementation issues. Due to the computational complexity of the algorithm and the existing hardware setup, only three joints of the PUMA could be controlled in real time. The first three joints of the PUMA were selected since they see the largest changes in inertia and gravity loading from a mass held in the manipulator gripper.

The DMRAC controlled PUMA was commanded over some typical three-joint minimum jerk trajectories with much success. The peak tracking error remained within about $\pm$ 1.5 degrees except where stiction effects caused the integral term to windup producing 2-4 degree peak errors.

The effects of disturbances were also investigated. The PUMA manipulator was physically disturbed from a setpoint to test the disturbance rejection capabilities of the DMRAC algorithm. In all cases, the algorithm recovered from the disturbance within about 1.5 seconds with a sharp decay back to steady-state. All torque signals remained bounded. The effects of stiction in the joints was also investigated. It was observed that the interaction between the joint stiction and the integral terms in the adaptation law produced slow limit cycles of small amplitude.

The DMRAC controlled PUMA was subjected to static load variations with great success. The effects of the load changes were quite small on the model following errors. The system was also subjected to dynamics load variations with equal success. The worst peak errors for the dynamics load runs were around 5 degrees and decayed to the nominal no load values within about 2 seconds. It was shown that the DMRAC algorithm was very robust in the presence of load changes.

In summary, the DMRAC algorithm was found to be an effective robotic control algorithm in both simulation and on the actual robotic manipulator being robust to static and dynamic load variations and also disturbances.

10.2 Future Research

A logical extension of this work would be to control all six joints of the actual PUMA 560 Manipulator. The existing version of the Testbed Motion Control System did not easily support a distributed controller. Recently proposed enhancements to MCS call for the support of distributed controllers, thus it will be possible to easily control all six joints of the PUMA with two DMRAC algorithms running, one for the wrist joints and one for the first three arm joints. The ability to control six joints in a fully centralized fashion will require increased computing power in the MCS VME Cage. One way to get this increased power might be to incorporate transputers into the MCS cage. Another method for achieving a centralized six joint controller would be to calculate the gain adaptation updates at a lower frequency than the control servo rate. With the addition of distributed controllers to MCS, this method will be easy to investigate.

One other area of future work involves the tuning and selection of tuning parameters. The algorithms used in this project were tuned by a very time consuming method of repeated runs with parameter adjustment between each run. If we assume we know nothing about the plant parameters then an automated tuning procedure could be designed. If we have some information regarding the plant parameters then a set of tuning rules could be developed to reduce the time needed to tune a DMRAC algorithm.

LITERATURE CITED

[1] R. J. Schilling, *Fundamentals of Robotics, Analysis and Control.* New Jersey: Prentice Hall, 1990.

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

[3] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence.* McGraw-Hill Book Company, 1987.

[4] J. J. Craig, *Introduction to Robotics, Mechanics & Control.* Reading, Mass.: Addison-Wesley, 1986.

[5] H. Kaufman, G. W. Neat, and R. Steinvorth, "Asymptotically stable multiple input multiple output direct model reference adaptive controller for processes not necessarily satisfying a positive real constraint," in *Proc. of European Control Conference*, (Grenoble, France), July 1991.

[6] K. M. Sobel and H. Kaufman, "Direct model reference adaptive control of a class of MIMO systems," in *Advances in Control and Dynamic Systems* (C. T. Leondes, ed.), vol. 24, pp. 245–314, Academic Press, 1986.

[7] I. D. Landau, "A survey of model reference adaptive techniques: Theory and applications," *Automatica*, vol. 10, 1974.

[8] R. V. Monopoli, "Model reference adaptive control with an augmenteeed error signal," *IEEE Transaction on Automatic Control*, vol. AC-18, 1974.

[9] K. M. Sobel, H. Kaufman, and L. Mabius, "Implicit adaptive control for a class of MIMO systems," *IEEE Transaction on Aerospace and Electronic Systems*, vol. AES-18, no. 5, pp. 576–590, 1982.

[10] I. BarKana, "Adaptive control - a simplified approach," in *Advances in Control and Dynamic Systems* (C. T. Leondes, ed.), vol. 25, Academic Press, 1987.

[11] J. R. Broussard and M. J. O'Brien, "Feed-forward control to track the output of a forced model," in *Proc. 17th IEEE Conf. Decision and Control*, pp. 1149–1155, Jan. 1979.

[12] I. BarKana and H. Kaufman, "Robust simplified adaptive control for a class of multivariable continuous systems," in *Proc. 24th IEEE Conf. Decision and Control*, (Ft. Lauderdale, Florida), pp. 141–146, 1985.

[13] I. BarKana and H. Kaufman, "Global stability and performance of a simplified adaptive control algorithm," *International Journal of Control*, vol. 42, no. 6, pp. 1491–1505, 1985.

[14] G. W. Neat, H. Kaufman, and S. R., "Comparison and extension of a direct model reference adaptive control procedure." Under Review by International Journal of Control, 1992.

[15] R. Steinvorth, "Model reference adaptive control of robots," Master's thesis, Rensselaer Polytechnic Institute, Troy, NY, 1991.

[16] S. T. Cummings, D. C. Swift, and K. H., "Direct model reference adaptive control of a six link puma arm." Under Review for Conference on Decision and Control, Dec. 1992.

[17] E. Kamen, *Introduction to Signals and Systems.* Macmillan Publishing Company, 1987.

[18] The MathWorks, Inc., Natick, MA, *Control Systems Toolbox for use with MATLAB*, Oct. 1990.

[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C.* Cambridge University Press, 1988.

[20] The MathWorks, Inc., Natick, MA, *PRO-MATLAB User's Guide*, 1990.

[21] B. Armstrong, O. Khatib, and J. Burdick, "The explicit dynamic model and inertial parameters of the PUMA 560 arm," in *Proc. 1986 IEEE Robotics and Automation Conference*, (San Francisco, CA), pp. 510–518, Mar. 1986.

[22] D. Swift, "Kinematic and dynamic parameters for the testbed grippers and loads," CIRSSE Technical Memorandum 14 (v. 1), Rensselaer Polytechnic Institute, Troy, NY, January 1992.

[23] J. F. Watson, III, "Testbed kinematic frames and routines," CIRSSE Technical Memorandum 1 (v. 2), Rensselaer Polytechnic Institute, Troy, NY, August 1991.

[24] S. Murphy and D. Swift, "Dynamic parameters and inverse dynamics for the PUMA 560," CIRSSE Technical Memorandum 13 (v. 1), Rensselaer Polytechnic Institute, Troy, NY, January 1992.

[25] T. J. Tarn, A. K. Bejczy, H. Shuotiao, and X. Yun, "Inertia parameters of PUMA 560 robot arm," Robotics Laboratory Report SSM-RL-85-01, Department of Systems Science and Mathematics, Washington University, September 1985.

[26] L. S. Wilfinger, "7 DOF gravity compensation for the testbed arms," CIRSSE Technical Memorandum 15 (v. 1), Rensselaer Polytechnic Institute, Troy, NY, February 1992.

[27] L. F. Shampine and M. K. Gordon, "FORTRAN double-precision ordinary differential equation integrator," 1978. ODE Integrator Program Down Loaded from Sandia National Labs.

[28] L. F. Shampine, M. K. Gordon, and W. H. Freeman, *Computer Solution of Ordinary Differential Equations, The Initial Value Problem.* San Francisco, CA: W. H. Freeman, 1975.

[29] K. Kyriakopoulos and G. Saridis, "Minimum jerk path generation," in *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, April 1988.

[30] K. Fieldhouse, "Lecture materials for the ctos/mcs introductory course," CIRSSE Report 97, Rensselaer Polytechnic Institute, Troy, NY, 1991.

[31] J. Tsai and Unimation, "The updated "breaking away from val"," tech. rep., Rensselaer Polytechnic Institute, Troy, NY, Mar. 1991.

[32] B. W. Kernighan and D. M. Ritchie, *The C Programming Language.* New Jersey: Prentice-Hall, Inc., 1978.

[33] Wind River Systems, Inc, 1351 Ocean Ave, Emeryville, CA 94608, *VxWorks Real-Time Operating System.*

APPENDIX A

Dynamic Equations of a PUMA 560 Manipulator

This appendix will list the equations used in the modeling the PUMA 560 Manipulator dynamics. Equations for the gravity loading, centrifugal matrix, coriolis matrix, and the kinetic energy matrix will be given. The details of the use of these equations to simulate the manipulator is described in Section 3.3.

```
Link and load masses in kilograms (links 2-6):

  m2 = 17.4;
  m3 = 4.8;
  m4 = 0.82;
  m5 = 0.34;
  m6 = 0.09;

Centers of gravity in meters:

  rx2 = 0.068;
  ry2 = 0.006;
  ry3 = -0.07;
  rz2 = -0.016;
  rz3 = 0.014;
  rz4 = -0.019;
  rz6 = 0.032;

Diagonal terms of the Inertia Dyadics:

  Ixx2 = 0.13;
  Ixx3 = 0.066;
  Ixx4 = 1.8e-3;
  Ixx5 = 0.3e-3;
  Ixx6 = 0.15e-3;
  Iyy2 = 0.524;
  Iyy3 = 0.0125;
  Iyy4 = 1.8e-3;
  Iyy5 = 0.3e-3;
  Iyy6 = 0.15e-3;
```

```
  Izz1 = 0.197; /* total Izz1 - Im1 = not about cm */
  Izz2 = 0.539;
  Izz3 = 0.086;
  Izz4 = 1.3e-3;
  Izz5 = 0.4e-3;
  Izz6 = 0.04e-3;

Motor Inertias:

  Im1 = 1.14;
  Im2 = 4.71;
  Im3 = 0.827;
  Im4 = 0.2;
  Im5 = 0.179;
  Im6 = 0.193;

Modified DH parameters:

  a2 = 0.43182;
  a3 = -0.02031;
  d2 = 0.243;
  d3 = -0.09391;
  d4 = 0.433;

Inertial Constants:

  I1, I2, I3, I4, I5, I6, I7, I8, I9, I10,
  I11, I12, I13, I14, I15, I16, I17, I18,
  I19, I20, I21, I22, I23

Gravitational Constants:

  g1, g2, g3, g4, g5

Kinetic energy (or mass) matrix (symetric).

  a11, a12, a13, a14, a15, a16,
       a22, a23, a24, a25, a26,
            a33, a34, a35, a36,
                 a44, a45, a46,
                      a55, a56,
                           a66
```

Coriolis matrix:

```
  b112, b113, b114, b115, b116,
        b123, b124, b125, b126,
              b134, b135, b136,
                    b145, b146,
                          b156,
  b212, b213, b214, b215, b216,
        b223, b224, b225, b226,
              b234, b235, b236,
                    b245, b246,
                          b256,
  b312, b313, b314, b315, b316,
        b323, b324, b325, b326,
              b334, b335, b336,
                    b345, b346,
                          b356,
  b412, b413, b414, b415, b416,
        b423, b424, b425, b426,
              b434, b435, b436,
                    b445, b446,
                          b456,
  b512, b513, b514, b515, b516,
        b523, b524, b525, b526,
              b534, b535, b536,
                    b545, b546,
                          b556,
  b612, b613, b614, b615, b616,
        b623, b624, b625, b626,
              b634, b635, b636,
                    b645, b646,
                          b656
```

Centrifugal matrix:

```
  c11, c12, c13, c14, c15, c16,
  c21, c22, c23, c24, c25, c26,
  c31, c32, c33, c34, c35, c36,
  c41, c42, c43, c44, c45, c46,
  c51, c52, c53, c54, c55, c56,
  c61, c62, c63, c64, c65, c66
```

Gravity Terms:

```
  gg1, gg2, gg3, gg4, gg5, gg6

Gravity constant:

  g

Sin / Cos terms:

  CC2, SS23, SC2, SS5, CC4, SC23, C4,
  SC5, C2, S23, C23, C5, S5, S4, CC23,
  S2, SC4, CC5, SS4, S3, C3, S223,
  C223, SS2

Inertial Constants:

  I1 = Izz1 + m2*d2*d2 + (m4 + m5 + m6)*a3*a3 +
       m2*rz2*rz2 + (m3 + m4 + m5 + m6)*(d2 + d3)*(d2 + d3) +
       Ixx2 + Iyy3 + 2*m2*d2*rz2 + m2*ry2*ry2 + m3*rz3*rz3 +
       2.0*m3*(d2 + d3)*rz3 + Izz4 + Iyy5 + Izz6;
  I2 = Izz2 + m2*(rx2*rx2 + ry2*ry2) + (m3 + m4 + m5 + m6)*a2*a2;
  I3 = -Ixx2 + Iyy2 + (m3 + m4 + m5 + m6)*a2*a2 + m2*rx2*rx2 -
       m2*ry2*ry2;
  I4 = m2*rx2*(d2 + rz2) + m3*a2*rz3 +
       (m3 + m4 + m5 + m6)*a2*(d2 + d3);
  I5 = -m3*a2*ry3 + (m4 + m5 + m6)*a2*d4 + m4*a2*rz4;
  I6 = Izz3 + m3*ry3*ry3 + m4*a3*a3 + m4*(d4 + rz4)*(d4 + rz4) +
       Iyy4 + m5*a3*a3 + m5*d4*d4 + Izz5 + m6*a3*a3 + m6*d4*d4 +
       m6*rz6*rz6 + Ixx6;
  I7 = m3*ry3*ry3 + Ixx3 - Iyy3 + m4*rz4*rz4 + 2.0*m4*d4*rz4 +
       (m4 + m5 + m6)*(d4*d4 - a3*a3) + Iyy4 - Izz4 + Izz5 -
       Iyy5 + m6*rz6*rz6 - Izz6 + Ixx6;
  I8 = -m4*(d2 + d3)*(d4 + rz4) - (m5 + m6)*(d2 + d3)*d4 +
       m3*ry3*rz3 + m3*(d2 + d3)*ry3;
  I9 = m2*ry2*(d2 + rz2);
  I10 = 2.0*m4*a3*rz4 + 2.0*(m4 + m5 + m6)*a3*d4;
  I11 = -2.0*m2*rx2*ry2;
  I12 = (m4 + m5 + m6)*a2*a3;
  I13 = (m4 + m5 + m6)*a3*(d2 + d3);
  I14 = Izz4 + Iyy5 + Izz6;
  I15 = m6*d4*rz6;
  I16 = m6*a2*rz6;
  I17 = Izz5 + Ixx6 + m6*rz6*rz6;
```

```
I18 = m6*(d2 + d3)*rz6;
I19 = Iyy4 - Ixx4 + Izz5 - Iyy5 + m6*rz6*rz6 + Ixx6 - Izz6;
I20 = Iyy5 - Ixx5 - m6*rz6*rz6 + Izz6 - Ixx6;
I21 = Ixx4 - Iyy4 + Ixx5 - Izz5;
I22 = m6*a3*rz6;
I23 = Izz6;
```

Gravitational Constants:

```
g1 = -g*((m3 + m4 + m5 + m6)*a2 + m2*rx2);
g2 = g*(m3*ry3 - (m4 + m5 + m6)*d4 - m4*rz4);
g3 = g*m2*ry2;
g4 = -g*(m4 + m5 + m6)*a3;
g5 = -g*m6*rz6;
```

Kinetic Energy Matrix:

```
a11 = Im1 + I1 + I3*CC2 + I7*SS23 + I10*SC23 + I11*SC2 +
      I20*(SS5*(SS23*(1.0 + CC4) - 1.0) - 2.0*SC23*C4*SC5) +
      I21*SS23*CC4 + 2.0*(I5*C2*S23 + I12*C2*C23 + I15*(SS23*C5 +
      SC23*C4*S5) + I16*C2*(S23*C5 + C23*C4*S5) + I18*S4*S5 +
      I22*(SC23*C5 + CC23*C4*S5));
a12 = I4*S2 + I8*C23 + I9*C2 + I13*S23 - I15*C23*S4*S5 +
      I16*S2*S4*S5 + I18*(S23*C4*S5 - C23*C5) + I19*S23*SC4 +
      I20*S4*(S23*C4*CC5 + C23*SC5) + I22*S23*S4*S5;
a13 = I8*C23 + I13*S23 - I15*C23*S4*S5 + I19*S23*SC4 +
      I18*(S2 C23*C5) + I22*S23*S4*S5 +
      I20*S4*(S23*C4*CC5 + C23*SC5);
a14 = I14*C23 + I15*S23*C4*S5 + I16*C2*C4*S5 + I18*C23*S4*S5 -
      I20*(S23*C4*SC5 + C23*SS5) + I22*C23*C4*S5;
a15 = I15*S23*S4*C5 + I16*C2*S4*C5 + I17*S23*S4 + I18*(S23*S5 -
      C23*C4*C5) + I22*C23*S4*C5;
a16 = I23*(C23*C5 - S23*C4*S5);
a22 = Im2 + I2 + I6 + I20*SS4*SS5 + I21*SS4 + 2.0*(I5*S3 + I12*C3 +
      I15*C5 + I16*(S3*C5 + C3*C4*S5) + I22*C4*S5);
a23 = I5*S3 + I6 + I12*C3 +I16*(S3*C5 + C3*C4*S5) + I20*SS4*SS5 +
      I21*SS4 + 2.0*(I15*C5 + I22*C4*S5);
a24 = -I15*S4*S5 - I16*S3*S4*S5 + I20*S4*SC5;
a25 = I15*C4*C5 + I16*(C3*S5 + S3*C4*C5) + I17*C4 + I22*S5;
a26 = I23*S4*S5;
a33 = Im3 + I6 + I20*SS4*SS5 + I21*SS4 + 2.0*(I15*C5 + I22*C4*S5);
a34 = -I15*S4*S5 + I20*S4*SC5;
a35 = I15*C4*C5 + I17*C4 + I22*S5;
```

```
a36 = I23*S4*S5;
a44 = Im4 + I14 - I20*SS5;
a45 = 0.0;
a46 = I23*C5;
a55 = Im5 + I17;
a56 = 0.0;
a66 = Im6 + I23;
```

Coriolis Matrix:

```
b112 = 2.0*(-I3*SC2 + I5*C223 + I7*SC23 - I12*S223 +
       I15*(2.0*SC23*C5 + (1.0 - 2.0*SS23)*C4*S5) + I16*(C223*C5 -
       S223*C4*S5) + I21*SC23*CC4 + I20*((1.0 + CC4)*SC23*SS5 -
       (1.0 - 2.0*SS23)*C4*SC5) + I22*((1.0 - 2.0*SS23)*C5 -
       2.0*SC23*C4*S5)) + I10*(1.0 - 2.0*SS23) + I11*(1.0 -
       2.0*SS2);
b113 = 2.0*(I5*C2*C23 + I7*SC23 - I12*C2*S23 + I15*(2.0*SC23*C5 +
       (1.0 - 2.0*SS23)*C4*S5) + I16*C2*(C23*C5 - S23*C4*S5) +
       I21*SC23*CC4 + I20*((1.0 + CC4)*SC23*SS5 - (1.0 -
       2.0*SS23)*C4*SC5) + I22*((1.0 - 2.0*SS23)*C5 -
       2.0*SC23*C4*S5)) + I10*(1.0 - 2.0*SS23);
b114 = 2.0*(-I15*SC23*S4*S5 - I16*C2*C23*S4*S5 + I18*C4*S5 -
       I20*(SS23*SS5*SC4 - SC23*S4*SC5) - I22*CC23*S4*S5 -
       I21*SS23*SC4);
b115 = 2.0*(I20*(SC5*(CC4*(1.0 - CC23) - CC23) - SC23*C4*(1.0 -
       2.0*SS5)) - I15*(SS23*S5 - SC23*C4*C5) - I16*C2*(S23*S5 -
       C23*C4*C5) + I18*S4*C5 + I22*(CC23*C4*C5 - SC23*S5));
b116 = 0.0;
b123 = 2.0*(-I8*S23 + I13*C23 + I15*S23*S4*S5 + I18*(C23*C4*S5 +
       S23*C5) + I19*C23*SC4 + I20*S4*(C23*C4*CC5 - S23*SC5) +
       I22*C23*S4*S5);
b124 = -I18*2.0*S23*S4*S5 + I19*S23*(1.0 - (2.0*SS4)) +
       I20*S23*(1.0 -2.0*SS4*CC5) - I14*S23;
b125 = I17*C23*S4 + I18*2.0*(S23*C4*C5 + C23*S5) +
       I20*S4*(C23*(1.0 - 2.0*SS5) - S23*C4*2.0*SC5);
b126 = -I23*(S23*C5 + C23*C4*S5);
b134 = b124;
b135 = b125;
b136 = b126;
b145 = 2.0*(I15*S23*C4*C5 + I16*C2*C4*C5 + I18*C23*S4*C5 +
       I22*C23*C4*C5) + I17*S23*C4 - I20*(S23*C4*(1.0 - 2.0*SS5) +
       2.0*C23*SC5);
b146 = I23*S23*S4*S5;
```

```
b156 = -I23*(C23*S5 + S23*C4*C5);
b212 = 0.0;
b213 = 0.0;
b214 = I14*S23 + I19*S23*(1.0 - (2.0*SS4)) + 2.0*(-I15*C23*C4*S5 +
       I16*S2*C4*S5 + I20*(S23*(CC5*CC4 - 0.5) + C23*C4*SC5) +
       I22*S23*C4*S5);
b215 = 2.0*(-I15*C23*S4*C5 + I22*S23*S4*C5 + I16*S2*S4*C5) -
       I17*C23*S4 + I20*(C23*S4*(1.0 - 2.0*SS5) - 2.0*S23*SC4*SC5);
b216 = -b126;
b223 = 2.0*(-I12*S3 + I5*C3 + I16*(C3*C5 - S3*C4*S5));
b224 = 2.0*(-I16*C3*S4*S5 + I20*SC4*SS5 + I21*SC4 - I22*S4*S5);
b225 = 2.0*(-I15*S5 + I16*(C3*C4*C5 - S3*S5) + I20*SS4*SC5 +
       I22*C4*C5);
b226 = 0.0;
b234 = b224;
b235 = b225;
b236 = 0.0;
b245 = 2.0*(-I15*S4*C5 - I16*S3*S4*C5) - I17*S4 + I20*S4*(1.0 -
       2.0*SS5);
b246 = I23*C4*S5;
b256 = I23*S4*C5;
b312 = 0.0;
b313 = 0.0;
b314 = 2.0*(-I15*C23*C4*S5 + I22*S23*C4*S5 + I20*(S23*(CC5*CC4 -
       0.5) + C23*C4*SC5)) + I14*S23 + I19*S23*(1.0 - (2.0*SS4));
b315 = 2.0*(-I15*C23*S4*C5 + I22*S23*S4*C5) - I17*C23*S4 +
       I20*S4*(C23*(1.0 - 2.0*SS5) - 2.0*S23*C4*SC5);
b316 = -b136;
b323 = 0.0;
b324 = 2.0*(I20*SC4*SS5 + I21*SC4 - I22*S4*S5);
b325 = 2.0*(-I15*S5 + I20*SS4*SC5 + I22*C4*C5);
b326 = 0.0;
b334 = b324;
b335 = b325;
b336 = 0.0;
b345 = -I15*2.0*S4*C5 - I17*S4 + I20*S4*(1.0 - 2.0*SS5);
b346 = b246;
b356 = b256;
b412 = -b214;
b413 = -b314;
b414 = 0.0;
b415 = -I20*(S23*C4*(1.0 - 2.0*SS5) + 2.0*C23*SC5) - I17*S23*C4;
b416 = -b146;
```

```
b423 = -b324;
b424 = 0.0;
b425 = I17*S4 + I20*S4*(1.0 - 2.0*SS5);
b426 = -b246;
b434 = 0.0;
b435 = b425;
b436 = -b346;
b445 = -I20*2.0*SC5;
b446 = 0.0;
b456 = -I23*S5;
b512 = -b215;
b513 = -b315;
b514 = -b415;
b515 = 0.0;
b516 = -b156;
b523 = -b325;
b524 = -b425;
b525 = 0.0;
b526 = -b256;
b534 = b524;
b535 = 0.0;
b536 = -b356;
b545 = 0.0;
b546 = -b456;
b556 = 0.0;
b612 = b126;
b613 = b136;
b614 = b146;
b615 = b156;
b616 = 0.0;
b623 = 0.0;
b624 = b246;
b625 = b256;
b626 = 0.0;
b634 = b624;
b635 = b625;
b636 = 0.0;
b645 = b456;
b646 = 0.0;
b656 = 0.0;
```

Centrifugal Matrix:

```
c11 = 0.0;
c12 = I4*C2 - I8*S23 - I9*S2 + I13*C23 + I15*S23*S4*S5 +
      I16*C2*S4*S5 + I18*(C23*C4*S5 + S23*C5) + I19*C23*SC4 +
      I20*S4*(C23*C4*CC5 - S23*SC5) + I22*C23*S4*S5;
c13 = 0.5*b123;
c14 = -I15*S23*S4*S5 - I16*C2*S4*S5 + I18*C23*C4*S5 +
      I20*S23*S4*SC5 - I22*C23*S4*S5;
c15 = -I15*S23*S4*S5 - I16*C2*S4*S5 + I18*(S23*C5 + C23*C4*S5) -
      I22*C23*S4*S5;
c16 = 0.0;
c21 = -0.5*b112;
c22 = 0.0;
c23 = 0.5*b223;
c24 = -I15*C4*S5 - I16*S3*C4*S5 + I20*C4*SC5;
c25 = -I15*C4*S5 + I16*(C3*C5 - S3*C4*S5) + I22*C5;
c26 = 0.0;
c31 = -0.5*b113;
c32 = -c23;
c33 = 0.0;
c34 = -I15*C4*S5 + I20*C4*SC5;
c35 = -I15*C4*S5 + I22*C5;
c36 = 0.0;
c41 = -0.5*b114;
c42 = -0.5*b224;
c43 = 0.5*b423;
c44 = 0.0;
c45 = 0.0;
c46 = 0.0;
c51 = -0.5*b115;
c52 = -0.5*b225;
c53 = 0.5*b523;
c54 = -0.5*b445;
c55 = 0.0;
c56 = 0.0;
c61 = 0.0;
c62 = 0.0;
c63 = 0.0;
c64 = 0.0;
c65 = 0.0;
c66 = 0.0;
```

Gravity Terms:

```
gg1 = 0.0;
gg2 = g1*C2 + g2*S23 + g3*S2 + g4*C23 + g5*(S23*C5 + C23*C4*S5);
gg3 = g2*S23 + g4*C23 + g5*(S23*C5 + C23*C4*S5);
gg4 = -g5*S23*S4*S5;
gg5 = g5*(C23*S5 + S23*C4*C5);
gg6 = 0.0;
```