MODEL REFERENCE ADAPTIVE CONTROL OF ROBOTS

NAGW-1333

by

Rodrigo Steinvorth

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

March 1991

CIRSSE REPORT #87

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to thank Professor Kaufman for his advice throughout my stay at RPI, which was necessary for the completion of this project. I also want to give special thanks to the people of the United States for having made my stay in their country a most pleasant one and without whose support nothing would have been possible. In addition, I want to thank CIRSSE for its support and acknowledge the financial assistance of the ECSE department and NASA Grant NAGW-1333.

Para Abuelo y Mamilú

ABSTRACT

This project presents the results of controlling two types of robots using new Command Generator Tracker (CGT) based Direct Model Reference Adaptive Control (MRAC) algorithms. Two mathematical models were used to represent a single-link, flexible joint arm and a Unimation PUMA 560 arm; and these were then controlled in simulation using different MRAC algorithms. Special attention was given to the performance of the algorithms in the presence of sudden changes in the robot load.

Previously used CGT based MRAC algorithms had several problems. The original algorithm that was developed guaranteed assymptotic stability only for almost strictly positive real (ASPR) plants. This condition is very restrictive, since most systems do not satisfy this assumption. Further developments to the algorithm led to an expansion of the number of plants that could be controlled, however, a steady state error was introduced in the response. These problems, led to the introduction of some modifications to the algorithms so that they would be able to control a wider class of plants and at the same time would asymptotically track the reference model.

This project presents the development of two algorithms that achieve the desired results and simulates the control of the two robots mentioned before. The results of the simulations are satisfactory and show that the problems stated above have been corrected in the new algorithms. In addition, the responses obtained show that the adaptively controlled processes are resistant to sudden changes in the load.

CHAPTER 1

Introduction

1.1 Introduction

This project presents some new modifications to a Direct Model Reference Adaptive Control (MRAC) algorithm proposed by BarKana and Kaufman [1], that were introduced to achieve asymptotic tracking and thus eliminate a steady state error that used to occur. In this project, we present the use of the new algorithms in simulations to control two different types of robot manipulators, and we compare their performance with algorithms that were used previously.

Why do we want to use adaptive control when we deal with robot manipulators? The reason is that there are always uncertainties that occur when we use robots to perform a given task, due to the changing environment in which they operate. If we use a non-adaptive controller, a set of gains which is adequate for a certain situation may not be adequate for another. The idea of adaptive control is to adjust to account for unexpected changes that occur in the system.

An example of a parameter that can suddenly change in a robot is the force or the torque exerted by the load that they carry. This alteration could be caused by different factors, such as unknown mass of the load, slippage at the end effector, or even drop of the load. Obviously, if no action is taken by the controller to account for these changes, there can be a negative effect on the performance of the robot. One solution to the problem presented by such unforeseen changes in the plant is to use an adaptive controller. As its name implies, an adaptive controller incorporates gains which adjust (adapt) with

time to account for changes that occur in the system.

Direct model reference adaptive control techniques are currently based on one of three different approaches [4]: First is the full state access method, which assumes that all state variables can be measured. This method has the limitation that the plant states are assumed to be directly measured which is not always possible. Second is the augmented error method which incorporates observers into the controller to be able to have access to the entire state vector. The disadvantage of this approach is that it becomes very complex when dealing with multi input-multi output systems such as robot manipulators where we have as input, several joint torques and as output, several joint angles. Finally, the algorithms presented in this project, are based upon command generator tracker theory as originally proposed by Sobel, Kaufman and Mabius [3].

Several advantages of the command generator tracker based approach over other methods include [4]:

- no need for direct estimates of the plant parameters
- 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 of full state feedback
- ease of implementation
- successful experimental validation

The major drawback with the original method proposed in [3] was the need for the system to satisfy a positive real condition. This greatly limited the number of plants which could be controlled using this algorithm. BarKana [1]

expanded the algorithm to include a larger class of plants by adding a feedforward term in parallel with the original plant; however, the difference between the augmented plant and the model's output was not the true difference between the model and plant outputs, and this situation introduced a steady state error. The modifications used in this project, eliminate this steady state error, while maintaining the larger number of plants that can be controlled. The following section describes the development of the algorithms given in [1] and [3] and explains their limitations in more detail.

1.2 Background

In this section we will show the development of the command generator tracker based Model Reference Adaptive Control algorithm derived by Sobel and Kaufman [2] and the extension provided by BarKana [1] which generalizes the approach to a wider class of plants.

1.2.1 Problem Description

We have a plant that is described by the following set of state equations:

$$\dot{x}_p(t) = A_p x_p(t) + B_p u_p(t) \tag{1.1}$$

$$y_p(t) = C_p x_p(t) \tag{1.2}$$

where $x_p(t)$ is the $(n_p x 1)$ plant state vector, $u_p(t)$ is the plant control vector, $y_p(t)$ is the plant output vector, and A_p, B_p, and C_p are matrices having the appropriate dimensions. Without knowing A_p, B_p, and C_p explicitly, we want to find the plant's control vector $u_p(t)$ such that its output vector $y_p(t)$ asymptotically tracks the output of a reference model given by the following state equations:

$$(1.3) \qquad \dot{x}_m(t) = A_m x_m(t) + B_m u_m(t)$$

$$(1.4) \qquad y_m(t) = C_m x_m(t)$$

where $x_m(t)$ is the $(n_m \times 1)$ model state vector with dimension n_m , $u_m(t)$ is the model control vector, $y_m(t)$ is the model output vector, and A_m, B_m, and C_m are matrices having the appropriate dimensions. It is important to note that the only restriction on the model is that it must have the same number of outputs as the plant; however, the dimension of the model state may be smaller than the dimension of the plant state. Therefore, it is possible to choose $n_m < n_p$ in order to simplify the problem. In addition, $u_m(t)$ can be any command signal that can be described as the solution of a differential equation forced by a step input as long as the time-varying portion of the command signal is augmented to the model state vector [5]. The basic strategy is to choose a model that will yield the desired output given a simple command input.

1.2.2 Development of the CGT Concept

The development of the adaptive algorithm is based on the command generator tracker (CGT) concept introduced by Broussard. Our description of this concept will closely follow the ones given in [2] and [5]. This approach assumes that there exist ideal trajectories of the plant $x^*_p(t)$ and $u^*_p(t)$ that satisfy the following equations:

$$(1.5) \qquad \dot{x}^*_p(t) = A_p x^*_p(t) + B_p u^*_p(t)$$

$$(1.6) \qquad y^*_p(t) = y_m = C_p x^*_p(t) = C_m x_m(t)$$

when perfect tracking occurs the real trajectories of the plant, $x_p(t)$ and $u_p(t)$, are the same as the ideal trajectories and therefore the real plant output

becomes the ideal plant output which is defined to be the output of the model.

It is assumed that the ideal trajectories $x^*_p(t)$ and $u^*_p(t)$ are linear functions of the model state and input $x_m(t)$ and $u_m(t)$, mathematically,

$$(1.7) \quad \begin{bmatrix} x_p^*(t) \\ u_p^*(t) \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} x_m \\ u_m \end{bmatrix}$$

In equation (1.7) u_m is assumed to be a constant input (otherwise we will need derivatives of the model input). We can rewrite equations (1.5) and (1.6) to obtain

$$(1.8) \quad \begin{bmatrix} \dot{x}_p^* \\ y_p^* \end{bmatrix} = \begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix} \begin{bmatrix} x_p^* \\ u_p^* \end{bmatrix}$$

Substituting this result into equation (1.7) yields

$$(1.9) \quad \begin{bmatrix} \dot{x}_p^* \\ y_p^* \end{bmatrix} = \begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix} \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} x_m \\ u_m \end{bmatrix}$$

Since u_m is a constant input we can differentiate the first equation in (1.7) to obtain the following (for simplicity from here on we will omit the reference to time, t.)

$$(1.10) \quad \dot{x}_p^* = S_{11}\dot{x}_m$$

Substituting equation (1.3) into (1.10) and concatenating with (1.6) results in

$$(1.11) \quad \begin{bmatrix} \dot{x}_p^* \\ y_p^* \end{bmatrix} = \begin{bmatrix} S_{11}A_m & S_{11}B_m \\ C_m & 0 \end{bmatrix} \begin{bmatrix} x_m \\ u_m \end{bmatrix}$$

We can now equate equations (1.9) and (1.11) to obtain

$$(1.12)\quad \begin{bmatrix} S_{11}A_m & S_{11}B_m \\ C_m & 0 \end{bmatrix}\begin{bmatrix} x_m \\ u_m \end{bmatrix}=\begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix}\begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix}\begin{bmatrix} x_m \\ u_m \end{bmatrix}$$

and since x_m and u_m are arbitrary this yields

$$(1.13)\quad \begin{bmatrix} S_{11}A_m & S_{11}B_m \\ C_m & 0 \end{bmatrix}=\begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix}\begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix}$$

A sufficient condition for Equation (1.13) to have a solution is that

$$(1.14)\quad \begin{bmatrix} \Omega_{11} & \Omega_{12} \\ \Omega_{21} & \Omega_{22} \end{bmatrix}=\begin{bmatrix} A_p & B_p \\ C_p & 0 \end{bmatrix}^{-1}$$

exists and no transmission zero of the plant is equal to any eigenvalue of A_m [5]. The resulting equations to be solved are

$$(1.15)\quad S_{11} = \Omega_{11}S_{11}A_m + \Omega_{12}C_m$$

$$(1.16)\quad S_{12} = \Omega_{11}S_{11}B_m$$

$$(1.17)\quad S_{21} = \Omega_{21}S_{11}A_m + \Omega_{22}C_m$$

$$(1.18)\quad S_{22} = \Omega_{21}S_{11}B_m$$

Even if (1.14) does not exist, a solution can almost always be found for S_{ij} [2]. For perfect output tracking, if $y_p = y_m$ at t=0, equation (1.7) shows that the control trajectory for this constant gain command generator tracker method is given by

$$(1.19)\quad u_p^*(t) = S_{21}x_m(t) + S_{22}u_m$$

The sufficient conditions to assure that perfect output tracking will occur using this control law are [5]:

A1) The matrices A_p, B_p, and C_p are known, linear, and time invariant.

A2) The inverse of equations (1.14) exists.

A3) No transmission zero of the plant is equal to any eigenvalue of A_m.

If $y_m \neq y_p$ at t=0, then we can achieve asymptotic output tracking if a stabilizing output feedback is included in the control law. The first step in obtaining this stabilizing feedback is to look at the error equation (i.e. the difference between the ideal and real states of the plant):

$$(1.20) \qquad e = x_p^* - x_p$$

We can differentiate equation (1.20) and substitute equations (1.1) and (1.5) to obtain

$$(1.21) \qquad \dot{e} = \dot{x}_p^* - \dot{x}_p = A_p x_p^* + B_p u_p^* - A_p x_p - B_p u_p$$

which is equivalent to

$$(1.22) \qquad \dot{e} = A_p e + B_p(u_p^* - u_p)$$

Choosing the following control law

$$(1.23) \qquad u_p = u_p^* + K(y_m - y_p) = u_p^* + K_e C_p e$$

and substituting into equation (1.22) yields the following error equation

$$(1.24) \qquad \dot{e} = (A_p - B_p K_e C_p)e$$

Obviously, the error will approach zero asymptotically provided that K_e is a stabilizing output feedback gain.

Therefore, we conclude that in order to achieve asymptotic output tracking when $y_m \neq y_p$ at t=0, we require the following condition in addition to those listed before (A1 - A3):

A4) A constant feedback gain K_e exists such that $(A_p - B_p K_e C_p)$ is asymptotically stable.

The resulting non-adaptive controller as given by equation (1.23) is represented in figure 1.1.

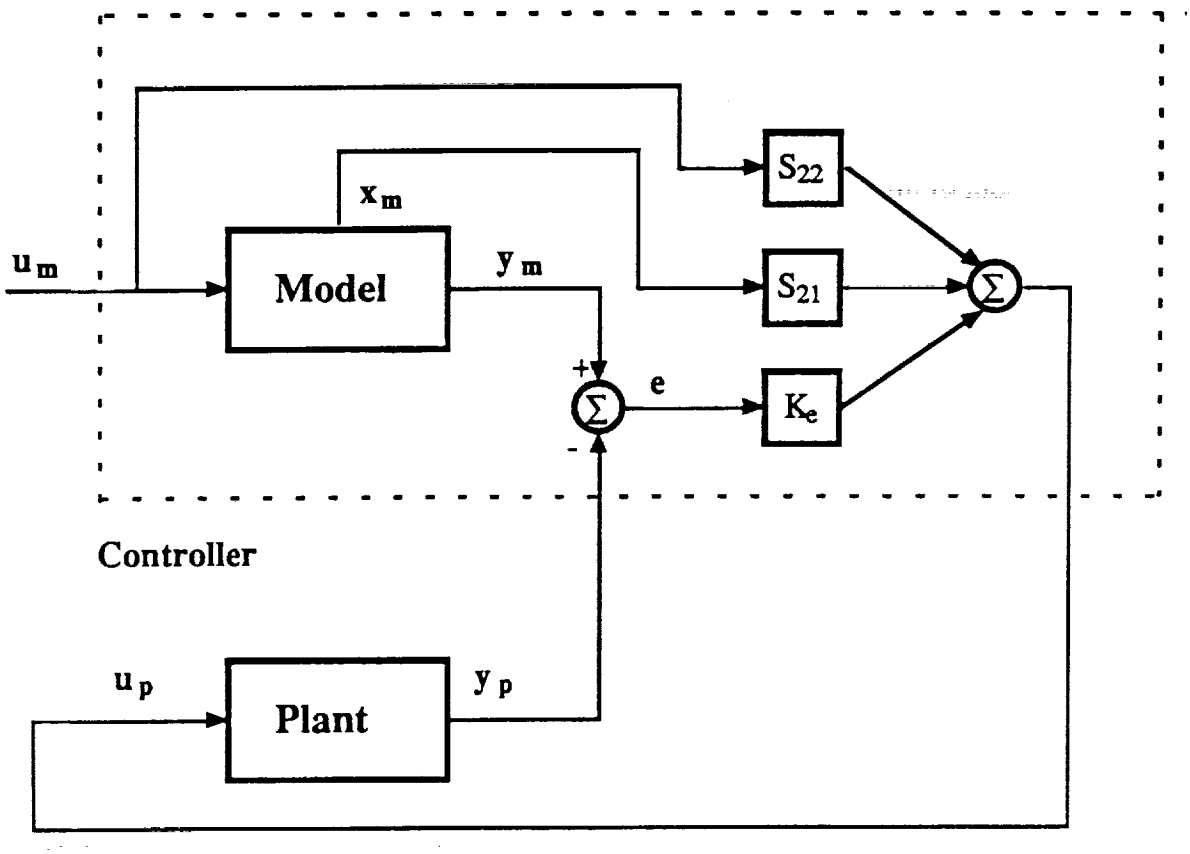


Figure 1.1: Non adaptive command generator tracker controller

1.2.3 Development of the Adaptive Control Law

As we mentioned previously, we are interested in the case when we do not have exact knowledge of the plant parameters, or in other words, condition A 1 is not satisfied. We want to determine a control law $u_p(t)$ which will cause the plant's output $y_p(t)$ to approximate "reasonably well" the model's output $y_m(t)$ without specific knowledge of A_p, B_p, and C_p. The adaptive control law chosen to achieve this is of the same form as the non-adaptive law given by equation (1.23) with the exception that the gains ($K_e(t)$, $K_u(t)$, and $K_x(t)$) are adaptive:

$$(1.25) \qquad u_p(t) = K_x(t)x_m(t) + K_u(t)u_m + K_e(t)(y_m(t) - y_p(t))$$

We are now faced with the task of finding adaptive laws for K_e, K_u, and K_x such that $e(t) \to 0$ as $t \to \infty$. In order to simplify the equations we will define the matrix $K_r(t)$ and the vector $r(t)$ as follows:

$$(1.26) \qquad K_r(t) = \begin{bmatrix} K_e(t) & K_x(t) & K_u(t) \end{bmatrix}$$

$$(1.27) \qquad r(t) = \begin{bmatrix} y_m(t) - y_p(t) \\ x_m(t) \\ u_m \end{bmatrix}$$

therefore

$$(1.28) \qquad u_p(t) = K_r(t)r(t)$$

The adaptive gains are obtained using the following equations which were proposed by Sobel, Kaufman, and Mabius [3]:

$$(1.29) \qquad K_p(t) = [y_m(t) - y_p(t)]\, r^T(t)\, \bar{T}$$

$$(1.30) \qquad \dot{K}_I(t) = [y_m(t) - y_p(t)]\, r^T(t)\, T$$

$$(1.31) \qquad K_r(t) = K_p(t) + K_I(t)$$

where $\bar{T}$ and T are time invariant square matrices. $K_p(t)$ and $K_I(t)$ are proportional and integral gains used only as an intermediate step in the calculation of $K_r(t)$. The following are sufficient conditions to achieve an asymptotically stable error:

A5) $\bar{T}$ and T are positive semidefinite and positive definite respectively.

A6) The plant is almost strictly positive real (ASPR).

Condition A6 means that there exist some feedback gain matrix $\tilde{K}_e$ such that the fictitious stabilized plant described by the triplet $(A_p - B_p\tilde{K}_eC_p, B_p, C_p)$ is strictly positive real. The proof of this stability result appears in [3]. Figure

1.2 shows the block diagram for the resulting adaptive algorithm. Comparison with Fig 1.1 shows that it is very similar to the non-adaptive case.

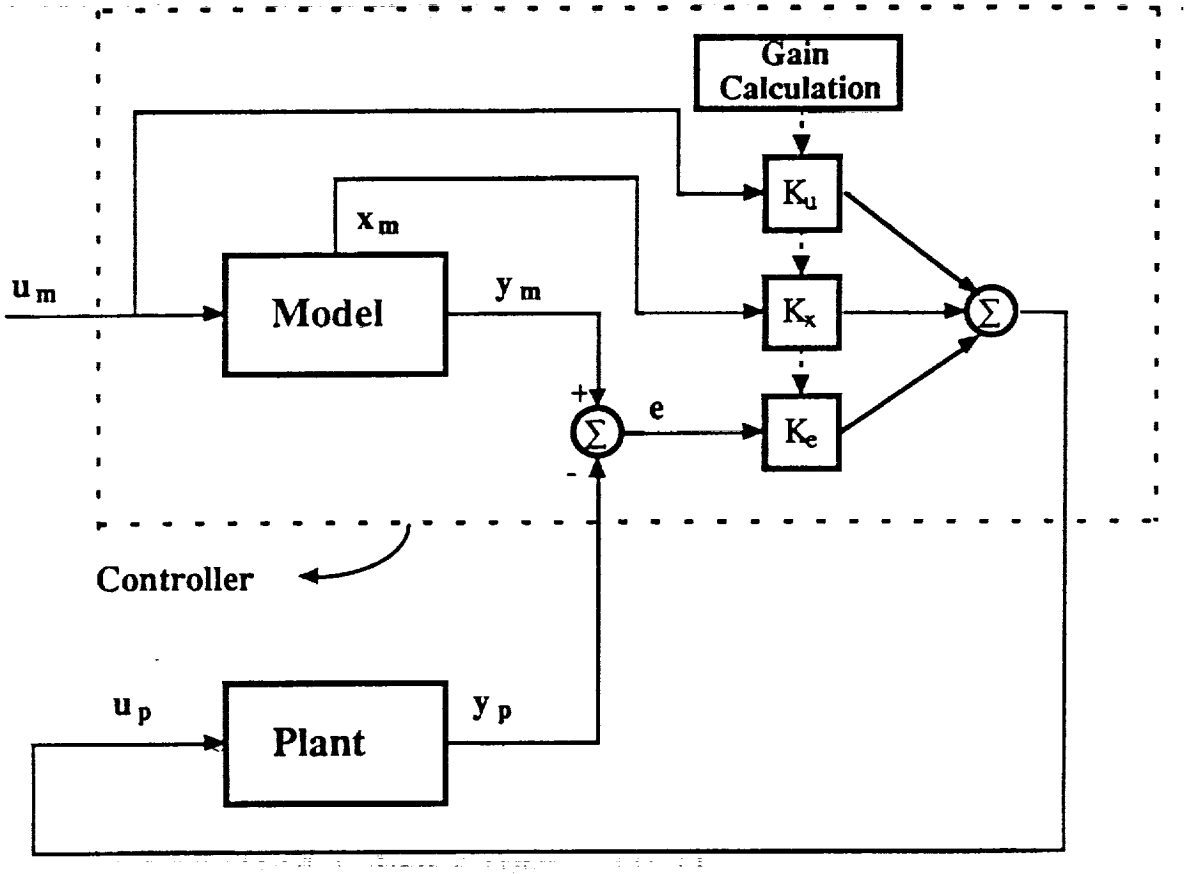


Figure 1.2: Model Reference Adaptive Controller

1.2.4 Extension of the Original Algorithm

As it turns out, **A 6** is a very restricting condition. Many plants do not satisfy the ASPR assumption and therefore the stability results from the previous section do not hold. To alleviate this problem, BarKana and Kaufman [6,7] suggested augmenting the plant with parallel dynamics such that the

augmented plant is ASPR so that the adaptive controller may be used.

Here we show the basic idea of this approach [8]. Let a non-ASPR plant be described by the following transfer matrix:

$$(1.32) \qquad G_p(s) = C_p(sI - A_p)^{-1}B_p$$

then, choose another transfer matrix H(s) in such a way that the augmented plant transfer matrix described by

$$(1.33) \qquad G_a(s) = G_p(s) + H^{-1}(s)$$

is ASPR. In [1] it is shown that the augmented plant $G_a(s)$ will be ASPR provided that both

- H(s) itself is ASPR
- H(s) stabilizes the closed loop output feedback system with transfer function $[I + G_p(s)H(s)]^{-1}G_p(s)$.

A choice for H(s), that is easy to implement and has been widely used, is

$$(1.34) \qquad H(s) = D_p^{-1}(1 + \tau s)$$

where D_p is a gain matrix and τ is a positive constant which can be chosen to satisfy the conditions stated previously. This results in the following augmented plant:

$$(1.35) \qquad G_a(s) = G_p(s) + \frac{D_p}{1 + \tau s}$$

The block diagram of the resulting system appears in Figure 1.3. In the the rest of this report we will refer to this algorithm as the "BarKana algorithm". As we can see in the figure, the error which is ensured to be stable (e_y) is not the true difference between the original plant's output and the model's output.

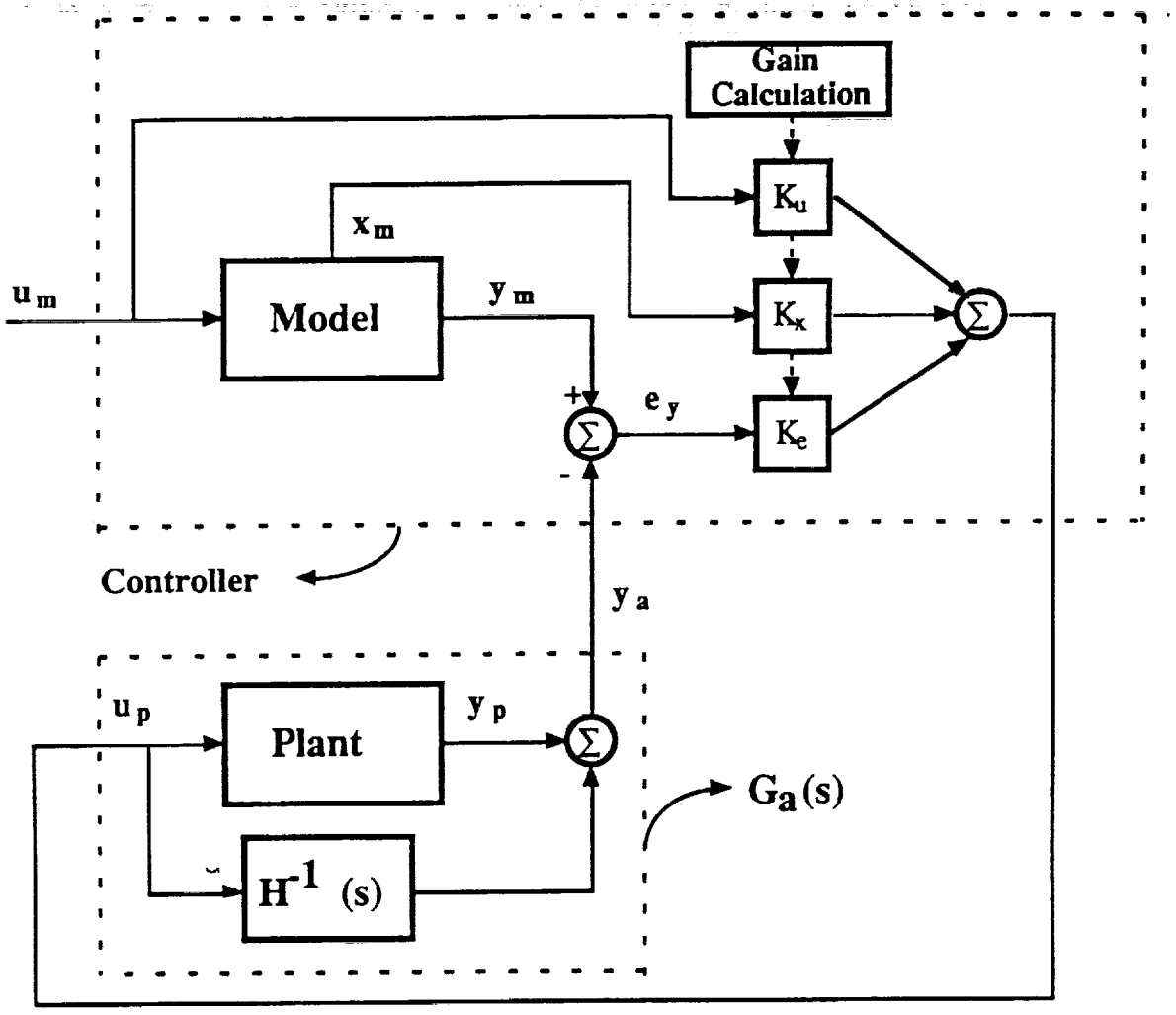


Figure 1.3: Extension to the CGT based MRAC system

In this case, the error is the difference between the augmented plant output and the model's output. This results in a steady state error. It is shown in [6] that if a plant is output stabilizable via high gain output feedback, then $\|D_p\|$ can be chosen to be small. In this case, the steady state error can be considered to be negligible and the original plant's output will be

approximately equal to the model's output.

1.3 Summary

We have presented a CGT based MRAC algorithm. The algorithm has the disadvantage that it guarantees asymptotic tracking only for a very restricted group of plants (i.e. ASPR plants). This algorithm was extended by BarKana and Kaufman to comprise a wider range of plants. However, this extension has the complication that a steady state error develops between the model and plant outputs.

CHAPTER 2

Modifications to Insure Asymptotic Tracking

2.1 Introduction

It is apparent that there are some limitations with the CGT based MRAC algorithms discussed in Chapter 1. The original algorithm has the restriction that it requires the plant to be ASPR. An attempt to solve this problem by BarKana and Kaufman [6,7] has the limitation that it results in a bounded steady state error. What we want is an algorithm which expands the range of plants for which asymptotic stability is ensured, in other words we want to eliminate the steady state error. This chapter will cover two approaches that achieve the desired results.

2.2 Modification #1: Adding a Feedforward to the Model

One approach to eliminate the steady state error resulting from the addition of the feedforward term to the plant's output is to incorporate this term into the model as well. The following is the development of this idea [9]. Consider the system defined by equations (1.1) and (1.2) and the model given by equations (1.3) and (1.4). Define an augmented plant output

$$(2.1) \qquad z_p(s) = y_p(s) + H^{-1}(s)u_p(s)$$

where

$$(2.2) \qquad H^{-1}(s) = \frac{D_p}{1 + \tau s}$$

Substituting equation (1.28) into Equation (2.1) we obtain

$$(2.3) \qquad z_p(s) = y_p(s) + H^{-1}(s)[K_x x_m + K_u u_m + K_e e_y]$$

Up to now, nothing new has been added to the algorithm, the new concept is to define an augmented model output as we have done with the plant's output:

$$(2.4) \qquad z_m(s) = y_m(s) + H^{-1}(s)[K_x x_m + K_u u_m]$$

Now, in order to control the augmented plant we will consider the augmented error between augmented plant and model outputs:

$$(2.5) \qquad e_z = z_m - z_p$$

which is equivalent to

$$(2.6) \qquad e_z = y_m - y_p - H^{-1}K_e e_z$$

or

$$(2.7) \qquad e_z = (I + H^{-1}K_e)^{-1} e_y$$

where $e_y = y_m - y_p$.

Substituting equation (2.2) into (2.7) gives

$$(2.8) \qquad e_z = \left(I + \frac{D_p K_e}{1 + \tau s}\right)^{-1} e_y$$

which is equivalent to

$$(2.9) \qquad ((1 + \tau s)I + D_p K_e)e_z = (1 + \tau s)e_y$$

We can now take the inverse Laplace transform to obtain

$$(2.10) \qquad \tau \dot{e}_z(t) + (I + K_e D_p)e_z(t) = \tau \dot{e}_y(t) + e_y(t)$$

therefore if the MRAC is designed so that $z_p \rightarrow z_m$ asymptotically then e_z and $\dot{e}_z$ will both approach zero and equation (2.10) reduces to

$$\tau \dot{e}_y(t) + e_y(t) = 0 \qquad (2.11)$$

from which we can immediately tell that e_y will decay to zero asymptotically; this is the desired result. The stability proof for this approach is presented in [9]. Figure 2.1 shows the block diagram of the resulting system. In the rest of this report we will refer to this algorithm as the "Kaufman algorithm".

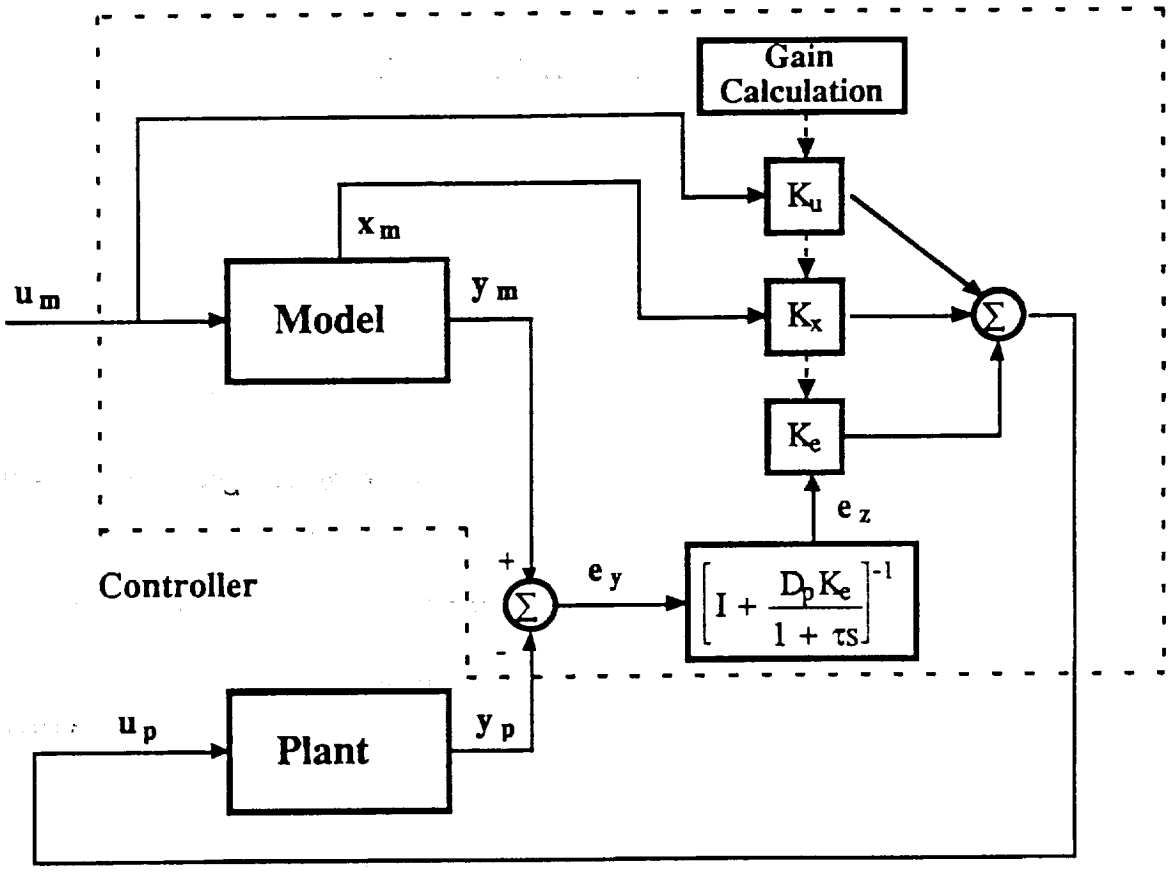


Figure 2.1: Modification #1 to the Model Reference Adaptive Controller to achieve assymptotic tracking

2.3 Modification #2: Adding a Zero to the Feedforward

Another way to achieve asymptotic tracking is by adding a zero at the origin to the feedforward term in parallel with the plant. The reason for this is that if the feedforward term has a zero at the origin it will asymptotically decay to zero and thus eliminate the steady state error. To implement this, we might make the feedforward term $H^{-1}(s)$ equal to one of the following two transfer matrices:

$$(2.12) \qquad H^{-1}(s) = \frac{D_p s}{\tau s + 1}$$

or

$$(2.13) \qquad H^{-1}(s) = \frac{D_p s}{as^2 + bs + 1}$$

where D_p is a gain matrix and τ, a, b are positive constants. The block diagram of the system is the same as the one previously given in Figure 1.3. In our simulations, which appear in the next chapter, we used equation (2.13) because it gave better results. In the rest of this project we will refer to this algorithm as the "Derivative algorithm".

2.4 Addition of a Derivative Term to the Plant Output

A modification to the algorithms presented in sections 2.2 and 2.3 which might make the system less sensitive to change is the augmentation of the plant's output with a derivative term as follows:

$$(2.14) \qquad y_a = y_p + \alpha \dot{y}_p$$

where α is a positive constant. The augmented plant's output would be used instead of the actual plant's output in each of the previous algorithms. We now

intuitively give some validation to this claim. We know that the plant output can be expressed as follows:

$$(2.15) \qquad y_p(s) = H(s)u_p(s)$$

and equation (2.14) is equivalent to

$$(2.16) \qquad y_a(s) = (\alpha s + 1)y_p(s)$$

or

$$(2.17) \qquad y_a(s) = (\alpha s + 1)H(s)u_p(s)$$

which means that we are adding a zero to the plant, therefore making the system "more strictly positive real", since we know that a system cannot be strictly positive real if its relative degree is larger than one. Even though we now have an augmented plant, its output at steady state will be the same as the output of the real plant since at that point the derivative term will become zero.

As we will see in later chapters, some of the algorithms presented above will in some cases have high frequency oscillations. The alpha term introduced in this section was observed to alleviate this problem. The larger the magnitude of α, the larger the reduction of the high frequency components of the response. However, increasing α also increases the error during the transient part of the response, because at these times the derivative term is not zero, and therefore the difference between the augmented plant's output and the model's output is not equal to the difference between the real plant's output and the model's output. As we reach steady state, the derivative terms decay to zero, and the augmented output is equal to the real output which results in asymptotic tracking.

To summarize, if there are high frequency components in the response we can eliminate them at the expense of a larger error during the transient. The amount of compromise will depend on the value that we choose for α. Figure 2.2 give a block diagram of our augmented plant.

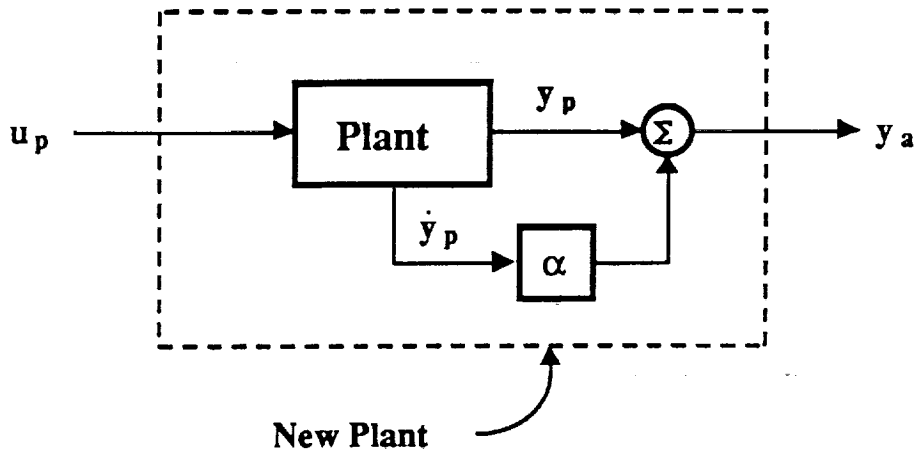


Figure 2.2: Plant augmented by weighted derivative term.

2.5 Summary

In this chapter we have shown two major modifications to the previous MRAC algorithms that accomplish asymptotic output tracking while at the same time maintaining the capability of controlling non-ASPR plants. The first modification involved augmenting both the model and the plant outputs, and the second included a zero at the origin in the feedforward. In addition, the idea of augmenting the original plant with a derivative term was considered in order to make the system less sensitive to change.

CHAPTER 3

Application to a Single-Link, Flexible-Joint Arm

3.1 Introduction

This chapter contains simulations to evaluate the use of the modified MRAC algorithms. We will control a single-link, flexible-joint robot arm that is described in [10], using the different variations of the MRAC algorithm described in the previous chapter. In addition, to show the usefulness of adaptive control, we will carry out simulations which demonstrate its performance during unforeseen circumstances (i.e. sudden load changes). All the simulations were carried out using Advanced Continuous Simulation Language (ACSL) in a VAX computer system. A listing of the ACSL programs used appears in the appendix.

3.2 Plant Description

Here we present the model of the single-link, flexible-joint arm (as given in [10]), that we will use to carry out our simulations. The joint is formed by two aluminum plates joined by extension springs with an actuator directly driving one plate. The dynamics of the system are given by the following equations:

$$(3.1) \qquad I\ddot{q}_1 + Mgl \sin(q_1) + k(q_1 - q_2) = 0$$

$$(3.2) \qquad J\ddot{q}_2 + B\dot{q}_2 - k(q_1 - q_2) = u_p$$

where: u_p = control torque which is calculated from the adaptive algorithms

q_1 = angle at the drive end of the link

q_2 = angle at the load end of the link

I = link inertia = 0.031 kg-m^2

J = rotor inertia = 0.004 kg-m^2

B = rotor friction = 0.007 N-m-sec/rad

Mgl = loading effect = 0.8 N-m

k = joint stiffness = 31.0 N-m/rad

Figure 3.1 shows a sketch of the link and the different parameters which describe the above equations:

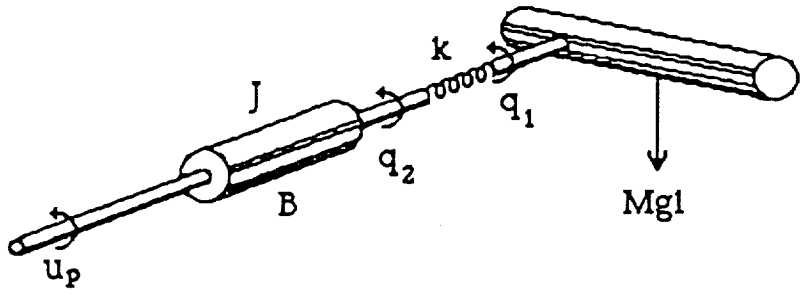


Figure 3.1: Model of single-link flexible -joint arm from [11]

It is very important to emphasize that the plant's model is used only to simulate the plant's behavior and it is not used in the control algorithm in any way. In other words, we use these equations to program the arm's behavior in ACSL to see how it will handle when we use our algorithms to control it.

In order to implement the MRAC algorithms we need to define a reference model. In our case we chose the following first order model:

$$(3.3) \qquad \frac{y_m}{u_m} = \frac{1}{s+1}$$

so as to yield an undamped response with a settling time of about 4 seconds.

We want the output of the system, which is the angle at the end of the link (i.e. $y_p = q_1$), to asymptotically track the output of the model (y_m). The command applied to the model (u_m) was arbitrarily chosen to be one radian for the first 30 seconds of the simulation, followed by a switch to a negative one radian command for the rest of the simulation as shown in Figure 3.2. Some of the simulations will involve a sudden change in the load the arm is carrying to test how the algorithm adapts to this "unforeseen" circumstance. In these occasions, the load change will occur at 15 seconds, and we will double the parameter Mgl from its nominal value of 0.8 N-m to a new value of 1.6 N-m instantaneously. Such a situation might occur in practice by an unwanted shift in the arm's load.

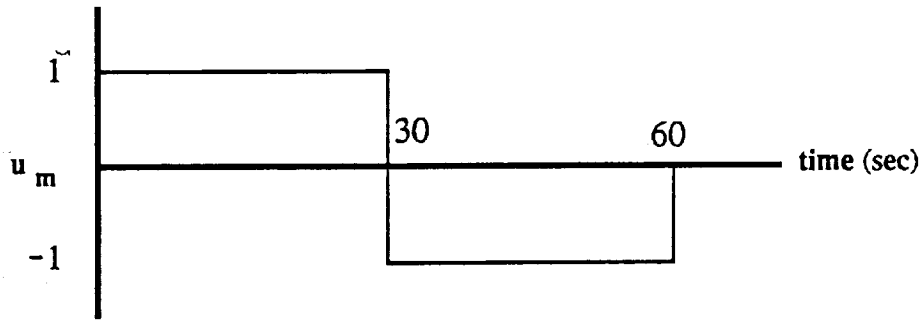


Figure 3.2: Command given to the model

3.3 Simulation Results

3.3.1 BarKana Algorithm

We will first show results of controlling the flexible arm using the extended CGT based MRAC algorithm with no modification to achieve asymptotic tracking (described in section 1.2.4), to be able to later compare its performance with the algorithms that include the new modifications. In these simulations we used the following values for the parameters for the algorithm:

$$D_p = 1.0 \tag{3.4}$$

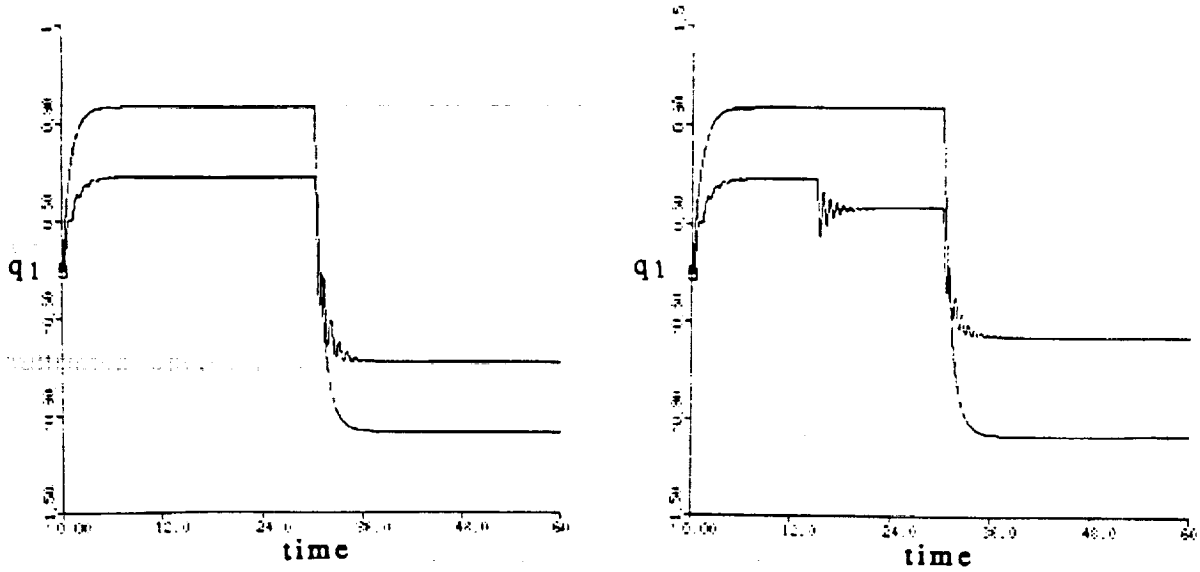
$$T = \bar{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\tau = 0.1$$

We can see in Figure 3.3 (a) a simulation of controlling the arm. The plot shows both, the output of the plant (i.e. the joint angle of the robot) and the output of the reference model. In this case the load the robot is carrying remains unchanged throughout the entire simulation. Figure 3.3 (b) depicts the same situation, however, in this case the load on the arm is doubled (i.e. Mgl = 1.6) at time = 15 seconds.

In both cases, a large steady state error is present in the response. Figure 3.3(b) shows that this steady state error becomes larger when the load increases. The steady state error is also directly dependent on the value of D_p, in other words, increasing D_p results in larger error and decreasing it results in smaller error. In addition, it was observed that decreasing D_p, while decreasing the steady state error, results in larger oscillations and decreased robustness. The results confirm the need for another algorithm which can

eliminate these problems.



(a) (b)

Figure 3.3: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "BarKana algorithm" for (a) no change in the arm's load, and (b) when the arm's load is doubled at 15 sec.

3.3.2 Kaufman Algorithm

We will now deal with simulations that use the algorithm described in section 2.2. This eliminates the steady state error that occurs in the previous simulations. The values used for the algorithm parameters in the following simulations were:

$$D_p = 100.0 \qquad (4.5)$$

$$T = \bar{T} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

$$\tau = 0.1$$

It important to mention that the parameters used in this simulation and all the others that come in this and the next chapters, were chosen to obtain a satisfactory response, however, they are not optimal and it is most likely that values which yield better responses exist. Also note, that we no longer have the restriction of using a small value for D_p as for the "BarKana algorithm".

Figure 3.4 shows the result of using our new controller to operate the robot arm. We can see that the difference between the plant and model outputs is barely noticeable, and that we achieve asymptotic tracking (i.e. the steady state error is eliminated). To test the robustness of the system, we doubled the load (i.e. Mgl = 1.6 N-m) on the manipulator at time=15 sec., and the results appear in figure 3.5

As we can see in figure 3.5(a), there is a small discontinuity when the load changes, however, the joint angle continues to asymptotically track the output of the reference model. The difference between the reference model and the arm's joint angle is very small after 20 seconds. Figure 3.5(b) shows the actual difference between the model output and the joint angle. The difference becomes large at three places: at the two transients and at the load change, and then it rapidly decays to zero, as expected. In addition, figures 3.5(c) and 3.5(d) show the plots of the torque applied to the arm (i.e. input command to the plant) and the value of one of the adaptive gains. Both of these values are bounded and achieve a steady state when the response of the system is also in steady state. Notice that the values for the control torque

remain below 1.5 N-m. It is apparent that the large changes in the input and the gains occur when there is something to "adapt" to, that is at the transients and the changes in the plant parameters (i.e. changing the load). In further simulations, both, the command input and the gains will be similar and therefore plots of their values will not be given from here on.

With respect to the algorithm parameters we noticed that in general, a larger value of D_p increases the rise time of the response. The opposite is generally true for τ and the weight matrices, the larger these values are, the faster the response of the system.

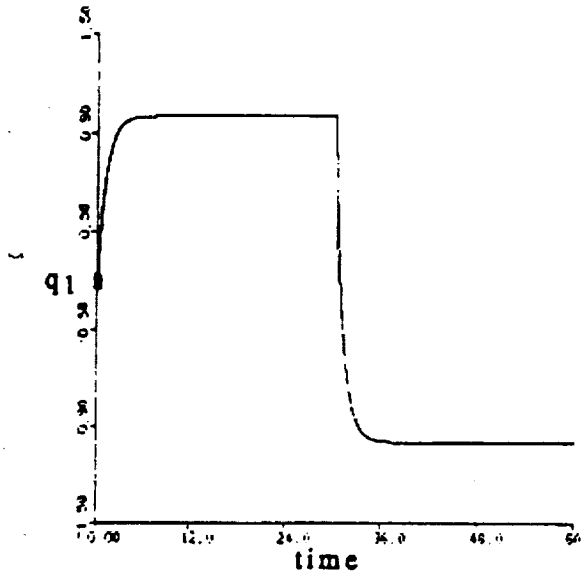


Figure 3.4: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" when there is no change in the arm's load.

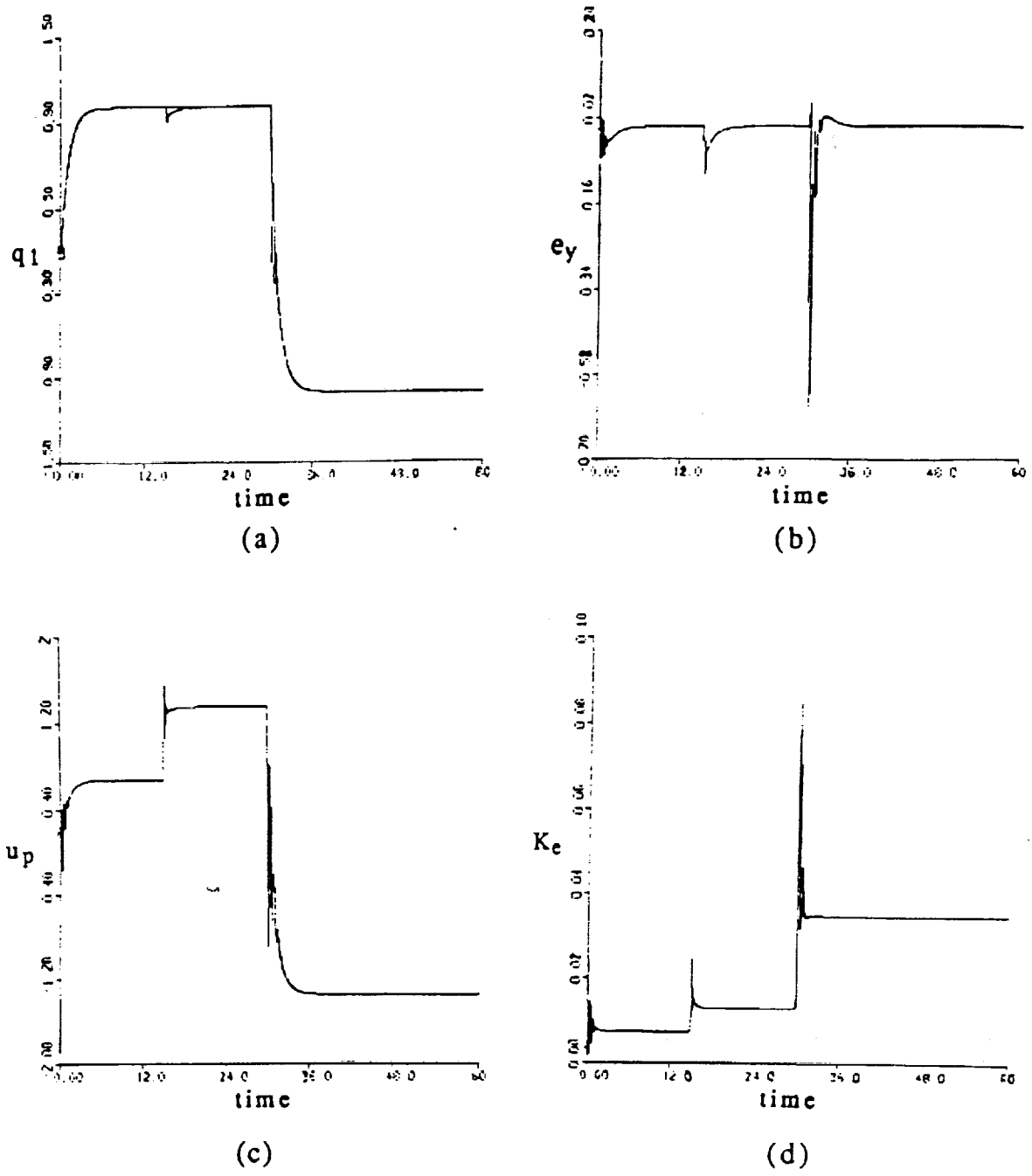


Figure 3.5: Simulation using the "Kaufman algorithm" with no change in the arm's load. (a) Plot of the plant and model outputs (rad.) vs. time (sec.), (b) Plot of the error between plant and model outputs vs. time, (c) Plot of the torque applied to the arm (N-m) vs. time, and (d) Plot of the gain K_e vs. time.

These rules are only approximate, however, they are useful in finding values for the parameters that will yield a good response.

It is apparent, that this modification to the previous algorithm achieves asymptotic tracking and can perform quite well in controlling the single-link flexible-joint arm. In addition, the system can successfully adapt to unexpected changes in the plant.

3.3.3 Derivative Algorithm

The other alternative to achieve asymptotic tracking was to use the algorithm presented in section 2.3. Simulations were made using this algorithm with the following nominal parameter values:

$$D_p = 0.2$$

$$T = \overline{T} = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

$$a = 0.4$$

$$b = 0.4$$

The result of the first simulation appears in Figure 3.6 and shows the responses of the model and the arm. It verifies that the steady state error is eliminated by using this algorithm. The actual error in the tracking is only noticeable in the transient part of the response. Figure 3.7 shows the results obtained when the torque at the load end of the robot is doubled to 1.6 N-m at time =15 sec. . The plot shows that there is a discontinuity at the instant of the change in the load, however, the system continues to be stable and asymptotically tracks the model. Figures 3.8(a) and (b) show plots of the

values of command torque and the gain acting on the error respectively. Both parameters are bounded and achieve a steady state value. As expected, at the transients and at the load change we can observe changes in their magnitudes. It is interesting to note that the steady state values achieved by the command torque in this simulation are the same as those achieved in the previous section.

The effects of changing the algorithm parameter values are the same as before. Generally increasing D_p results in a slower response, while increasing a, b, and the weight matrices results in a slower response.

Again, the modification introduced achieved asymptotic tracking and successfully controlled the robot arm. In addition, the system is resistant to sudden changes in the system.

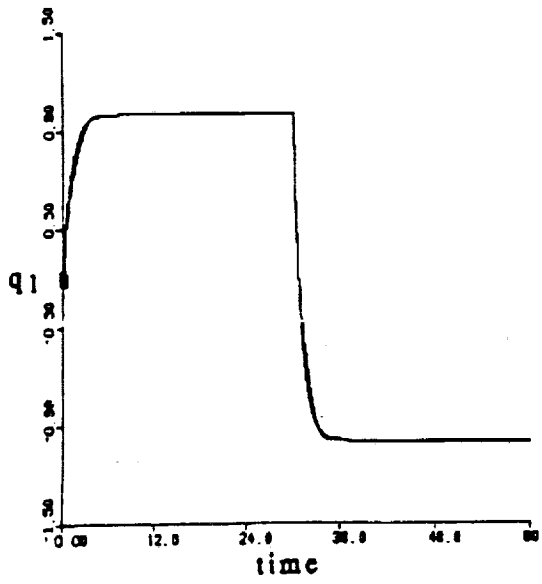


Figure 3.6: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Derivative algorithm" for no change in the arm's load.

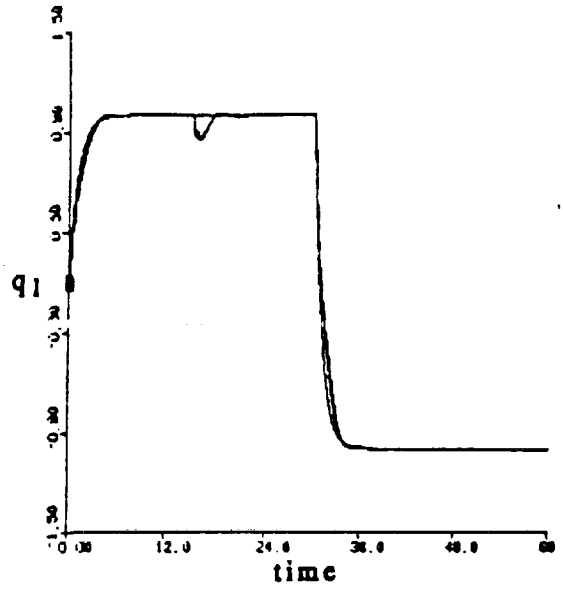
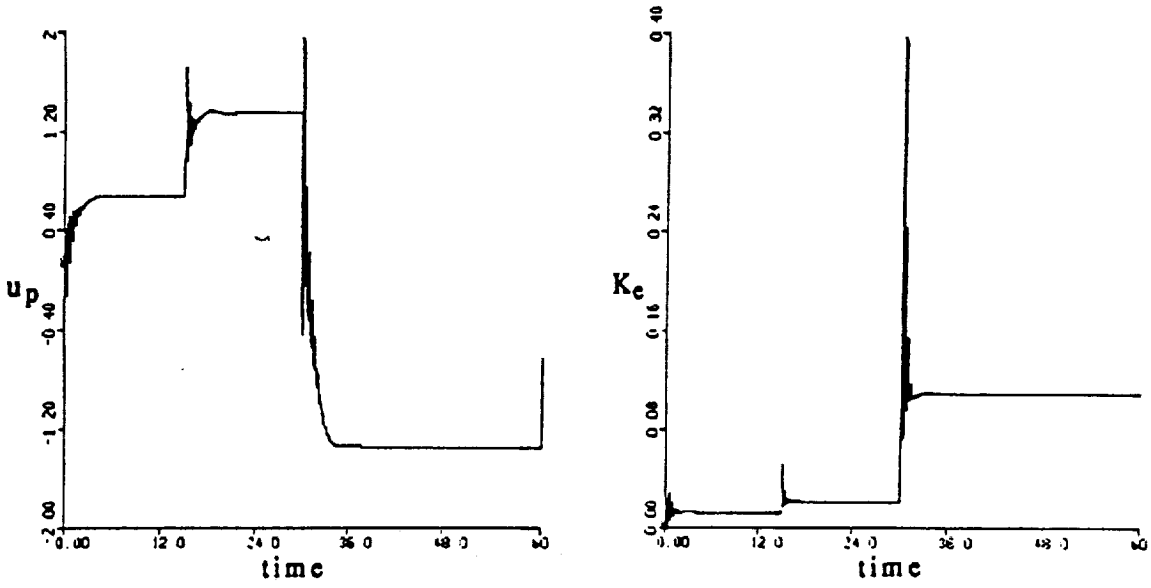


Figure 3.7: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Derivative algorithm" for a sudden change in the arm's load at 15 sec.



(a) (b)

Figure 3.8: Plots using the "Derivative algorithm" for a sudden change in the arm's load at 15 sec. of (a) torque command (N-m) vs time (sec.) and (b) gain K_e vs. time (sec.)

3.4 Summary

This chapter described a single-link flexible-joint arm and showed the results of simulations used to implement different MRAC algorithms to control it. The algorithms used were the "BarKana algorithm" which had a steady state error and the Kaufman and Derivative algorithms that contain the new modifications to achieve asymptotic tracking. The results of the simulations showed that the robot can be successfully controlled using the new algorithms and that the resulting systems are resistant to sudden changes in the payload.

A comparison of the two algorithms with the modifications showed that it is easier to find good parameter values for the "Kaufman algorithm" than for the "Derivative algorithm". In addition, the first consistently resulted in smaller error than the second. The range of values of D_p that can be used with the Kaufman algorithm seems to be larger than for the Derivative algorithm since it is difficult to achieve a good response with large values of D_p in the latter method.

CHAPTER 4

Application to a Model of the PUMA 560 Arm

4.1 Introduction

This chapter continues with simulations to evaluate the use of the modified MRAC algorithms. We will control the second and third joints of a Puma 560 robot arm using the model given in [11]. This is a multiple input-multiple output (MIMO) system; however, we will see that the complexity of programing the equations to implement the algorithms is not greatly increased. Again, we use the different variations of the MRAC algorithm described in Chapter 2. In addition, to show the usefulness of adaptive control, we will carry out simulations which demonstrate its performance during unforeseen circumstances (i.e. sudden load changes). We continue to carry out all the simulations using Advanced Continuous Simulation Language (ACSL) in a VAX computer system. The listings of the different programs used to perform the simulations which appear in this chapter are given in the appendix.

4.2 Plant Description

To carry out our simulations, we used a model of the second and third joint dynamics of the Puma 560 arm, which we will describe in this section following the development that appears in [11]. In contrast to the flexible arm considered in Chapter 3, this is a multi input-multi output system, where the inputs are the two torques applied at both joints of the manipulator, and the outputs are the two joint angles. The matrix equation that describes this

system is the following:

$$(4.1) \quad T = M(\theta)\ddot{\theta} + N(\theta,\dot{\theta}) + G(\theta) + H(\dot{\theta}) + mJ^T(\theta)\left[J(\theta)\ddot{\theta} + \dot{J}(\theta,\dot{\theta})\dot{\theta} + g\right]$$

where the different terms in the equation are:

$M(\theta)$ = Symmetric positive definite inertia matrix

$N(\theta,\theta')$ = Coriolis and centrifugal torque vector

$G(\theta)$ = Gravity loading vector

$H(\theta')$ = Frictional torque vector

T = Vector of applied joint torques (control input)

θ = Joint angle vector (plant output)

g = gravity vector

These terms are described by the following equations

$$(4.2) \quad M(\theta) = \begin{bmatrix} a_1 + a_2\cos\theta_2 & a_3 + \frac{a_2}{2}\cos\theta_2 \\ a_3 + \frac{a_2}{2}\cos\theta_2 & a_3 \end{bmatrix}$$

$$(4.3) \quad N(\theta,\dot{\theta}) = \begin{bmatrix} -(a_2\sin\theta_2)(\dot{\theta}_1\dot{\theta}_2 + \frac{\dot{\theta}_2^2}{2}) \\ (a_2\sin\theta_2)\frac{\dot{\theta}_1^2}{2} \end{bmatrix}$$

$$(4.4) \quad G(\theta) = \begin{bmatrix} a_4\cos\theta_1 + a_5\cos(\theta_1 + \theta_2) \\ a_5\cos(\theta_1 + \theta_2) \end{bmatrix}$$

$$(4.5) \quad H(\dot{\theta}) = \begin{bmatrix} V_1\dot{\theta}_1 + V_2\,\mathrm{sgn}(\dot{\theta}_1) \\ V_3\dot{\theta}_2 + V_4\,\mathrm{sgn}(\dot{\theta}_4) \end{bmatrix}$$

$$(4.6) \qquad J(\theta) = \begin{bmatrix} -L_1\sin\theta_1 - L_2\sin(\theta_1 + \theta_2) & -L_2\sin(\theta_1 + \theta_2) \\ L_1\cos\theta_1 + L_2\cos(\theta_1 + \theta_2) & L_2\cos(\theta_1 + \theta_2) \end{bmatrix}$$

$$(4.7) \qquad g = \begin{bmatrix} 0 \\ 9.81 \end{bmatrix}$$

The terms $(a_1 \ldots a_5)$ appearing in the previous expressions are constants that are obtained from the masses (m_1, m_2) and lengths (L_1, L_2) of both robot links. In the case of links 2 and 3 of the Unimation PUMA 560 arm, the masses are $m_1 = 15.91$ kg and $m_2 = 11.36$ kg respectively, and the lengths are $L_1 = L_2 = 0.432$ m. These result in the following numerical values for the model parameters:

$$(4.8) \qquad (a_1, a_2, a_3, a_4, a_5) = (3.82, 2.12, 0.71, 81.82, 24.06)$$

The terms (V_1, V_3) and (V_2, V_4) are coefficients of viscous and Coulumb friction, respectively. The following values were assigned to these coefficients: $V_1 = V_3 = 1.0$ Nt-m/rad-sec^{-1}, and $V_2 = V_4 = 0.5$ Nt-m. The payload mass of the arm was set to m = 10 kg. Figure 4.1 shows a 2-D view of the two links of the PUMA 560 arm that we want to control.

As for the case of the single-link, flexible joint arm, this description of the PUMA 560 arm is used only to create an ACSL simulation of the plant's behavior to different command inputs. We do not use any of the knowledge that we have from the equations describing the model in our control algorithms.

Again, we chose a first order reference model for the MRAC algorithms. This model is given by the following equations:

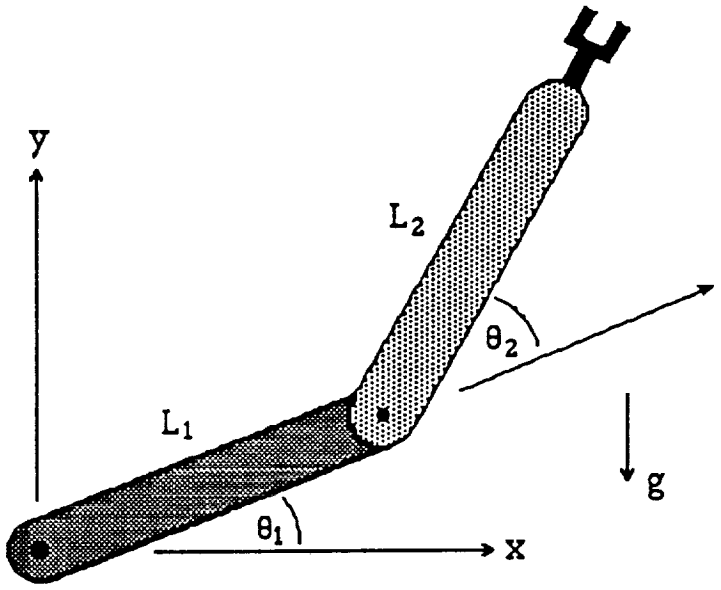


Figure 4.1: 2-D representation of the 2nd and 3rd links of Unimation PUMA 560 robot manipulator.

$$(4.9) \qquad y_{m1} = \frac{u_{m1}}{(0.1s + 1)}$$

$$(4.10) \qquad y_{m2} = \frac{u_{m2}}{(0.1s + 1)}$$

We can see that there are two command inputs and two outputs. This is necessary because the system to be controlled also has two inputs and two outputs. We want the the angle at joint 1 (θ_1) and the angle at joint 2 (θ_2), which are the outputs of the system, to asymptotically track the outputs of the model, y_{m1} and y_{m2} respectively. The model was chosen so that its dynamics are fast enough to have its output be approximately the same as the command

input. The two controls (u_{m1}, u_{m2}) are described by the following equations which change much slower than the model dynamics:

$$(4.11) \quad u_{m1} = \frac{-\pi}{2} + 0.25\left[\frac{2\pi t}{3} + \sin\left[\frac{2\pi t}{3}\right]\right] \text{ rad.} \qquad 0 \le t \le 3$$

$$= 0 \text{ rad} \qquad 3 < t \le 5$$

$$= -\ 0.25\left[\frac{2\pi(t-5)}{3} - \sin\left[\frac{2\pi(t-5)}{3}\right]\right] \text{ rad.} \qquad 5 < t \le 8$$

$$= -\frac{\pi}{2} \text{ rad} \qquad 8 < t$$

$$(4.12) \quad u_{m2} = u_{m1} - \frac{\pi}{2} \text{ rad} \qquad 0 < t$$

To test the robustness of our algorithms we introduced a sudden change in the payload that the arm carries in some of our simulation runs that appear later in this chapter. The change in the load occurs instantaneously 6.5 seconds into the simulation, and the value of m changes from 10 kg. to 20 kg.

4.3 Simulation Results

4.3.1 BarKana Algorithm

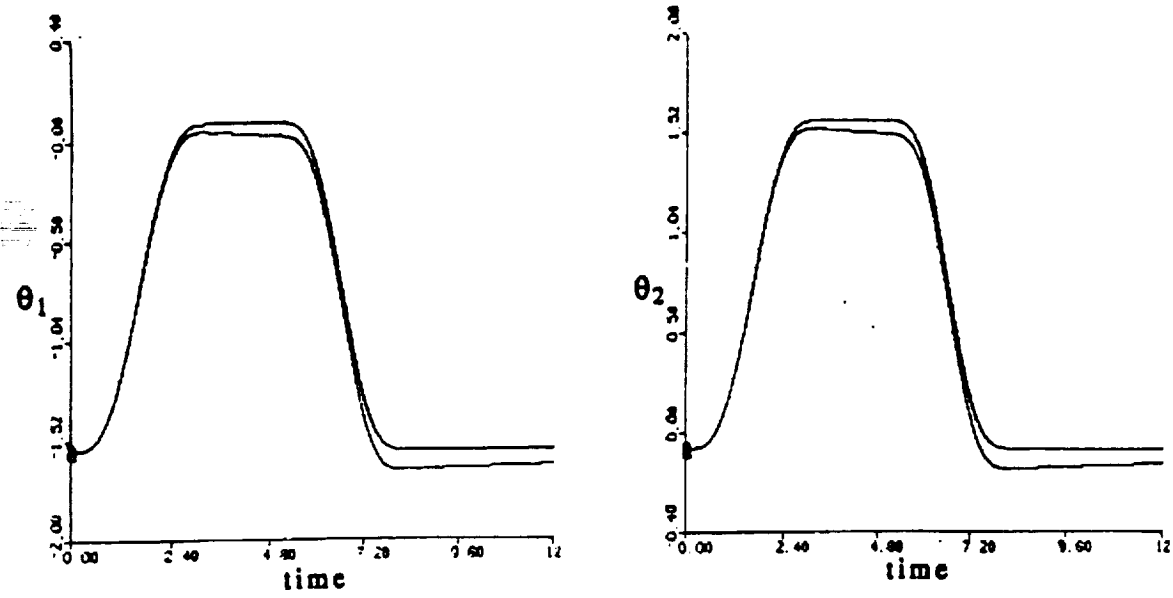
In this section we control the PUMA 560 arm using the algorithm described in section 1.2.4 which did not contain the new modifications to achieve asymptotic output tracking. This is done to compare the performance of the previous algorithm with the new ones, and to be able to point out the deficiencies that it has. The values used for the algorithm's parameters were the following:

$$(4.13) \qquad D_p = \begin{bmatrix} 0.005 & 0 \\ 0 & 0.005 \end{bmatrix}$$

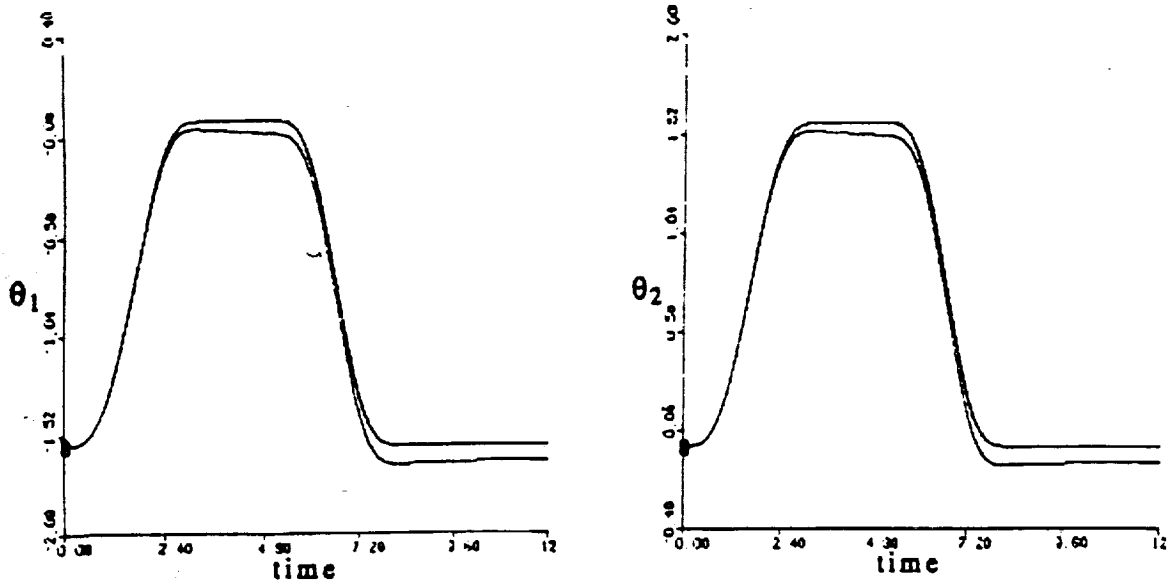
$$\tau = 50.0$$

$$T = \overline{T} = \begin{bmatrix} 3\ 0\ 0 & 0\ 0\ 0 \\ 0\ 3\ 0 & 0\ 0\ 0 \\ 0\ 0\ 3 & 0\ 0\ 0 \\ 0\ 0\ 0 & 3\ 0\ 0 \\ 0\ 0\ 0 & 0\ 3\ 0 \\ 0\ 0\ 0 & 0\ 0\ 3 \end{bmatrix} \times 10^4$$

The results of the first simulation appear in Figure 4.2, which shows the outputs of the model and the outputs of the plant (in radians) versus time (in seconds). Figure 4.2(a) shows the response of the joint angle between the first and second links, and 4.2(b) between the second and third links of the PUMA 560. It is clearly apparent that there is a steady state error in the response. This error can be decreased by making the value of D_p smaller, however, if we make D_p too small unstable oscillations can appear in the response. Figure 4.3 shows the results when $D_p = 0$ when a sudden change in the load is present during the simulation. It is clearly appreciated that the system is not asymptotically stable since we have some increasing oscillations in the response. These results give further reason for the modifications introduced to the algorithm which will be used to simulate the control of the robot in the subsequent sections.



(a)



(b)

Figure 4.2: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "BarKana algorithm" for (a) no change in the arm's load, and (b) when the arm's load is doubled at 6.5 sec.

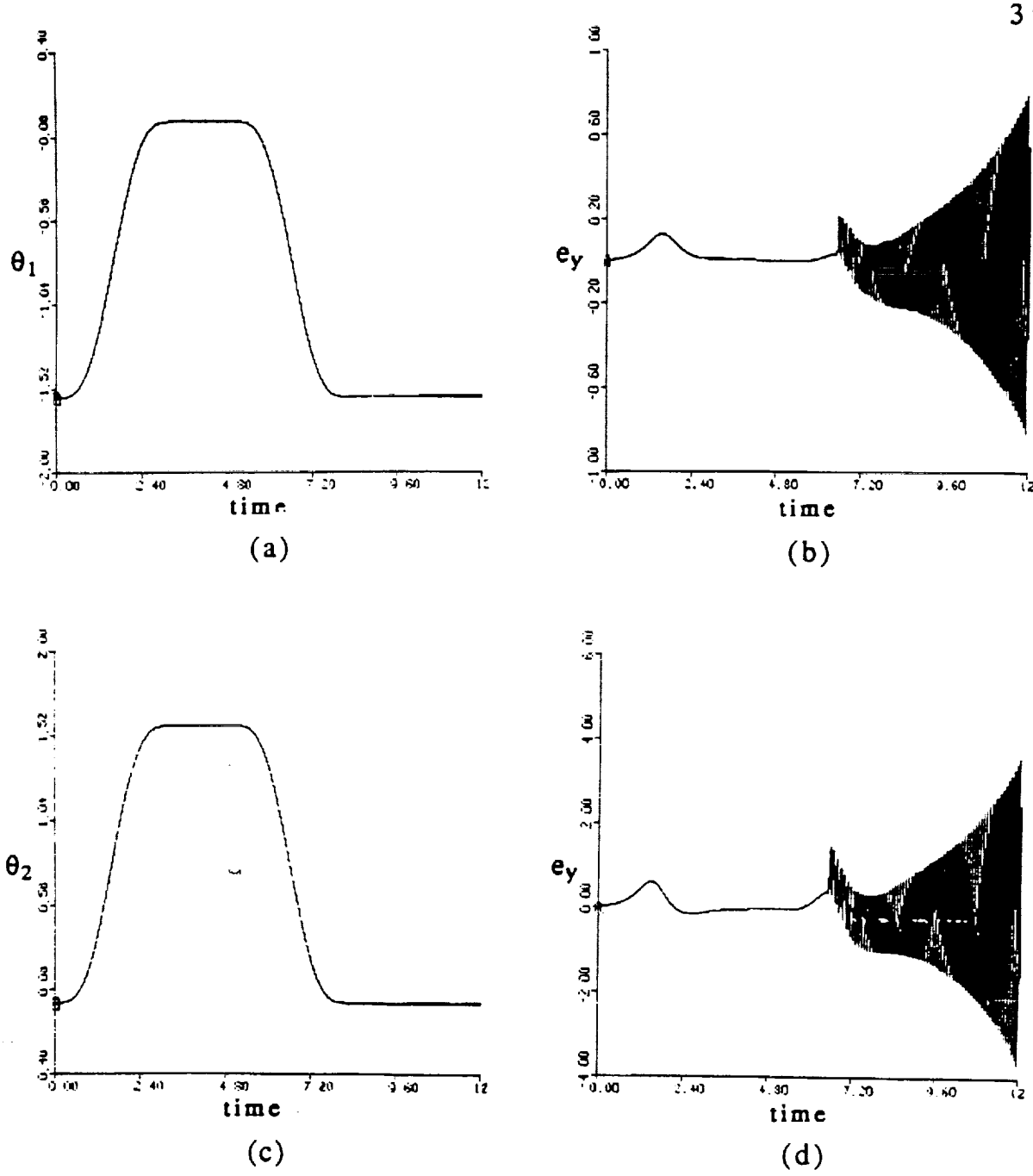


Figure 4.3: Plot using the "BarKana algorithm" when the arm's load is doubled at 6.5 sec. for (**a**) the plant and model's first output, (**b**) the error between plant and model's first output, (**c**) the plant and model's second output, and (**d**) the error between plant and model's second output

4.3.2 Kaufman Algorithm

The robot control is now simulated using the first of the modified algorithms introduced to achieve asymptotic stability. The nominal values used for the parameters of the algorithm are the following:

$$(4.14) \qquad D_p = \begin{bmatrix} 6 & 0 \\ 0 & 6 \end{bmatrix}$$

$$T = \overline{T} = \begin{bmatrix} 0.2 & 0 & 0 & & & \\ 0 & 0.2 & 0 & & 0 & \\ 0 & 0 & 1.4 & & & \\ & & & 1.4 & 0 & 0 \\ & 0 & & 0 & 1.4 & 0 \\ & & & 0 & 0 & 1.4 \end{bmatrix} \times 10^4$$

$$\tau = 0.01$$

Figure 4.4 shows the response of the plant and the model when no change in the load is present, and we can see that the error between both is so small that it cannot be observed. Therefore, the plots of the error are given in Figure 4.5. In the future we will only present the plots of the errors when no difference can be observed between model and plant outputs (as occurred in Figure 4.4).

The results when a sudden change in the load is introduced at 6.5 sec. appear in Figure 4.6, which shows the error between the model and the plant outputs. A discontinuity is noticeable at the time of the load change, however, the error decays to zero even though there are some high frequency oscillations present in the response.

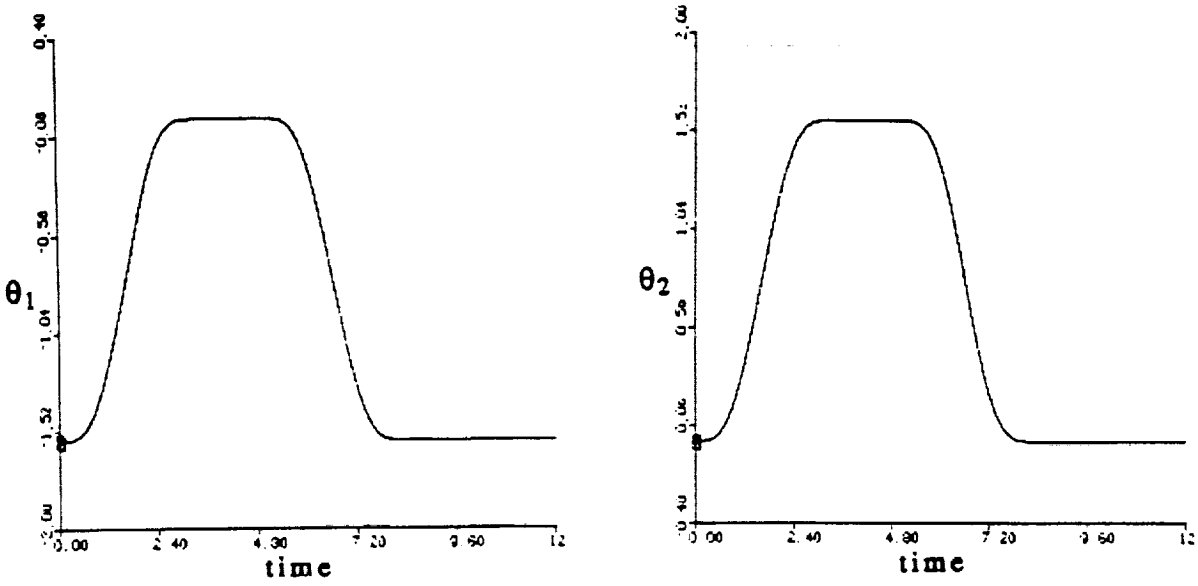


Figure 4.4: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" for no change in the arm's load.

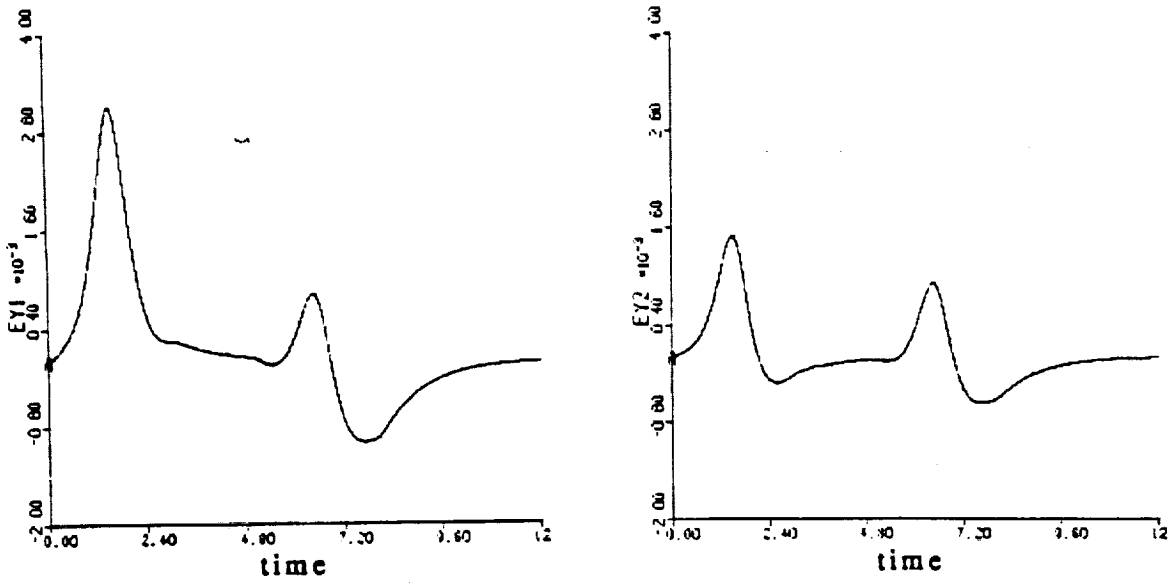


Figure 4.5: Plot of the error between the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" for no change in the arm's load.

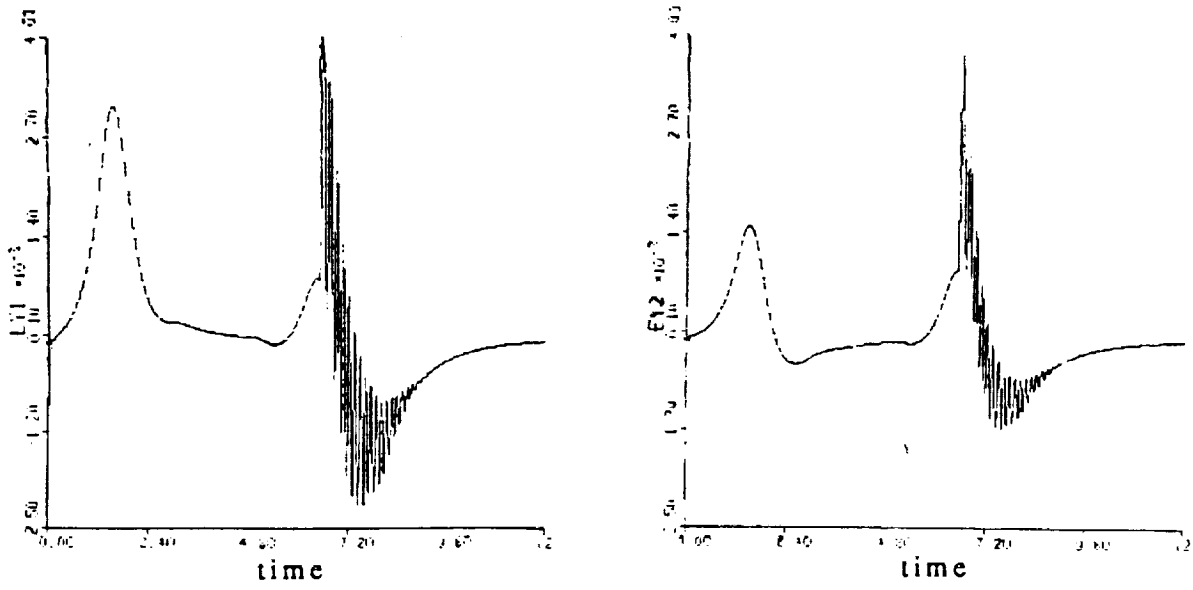
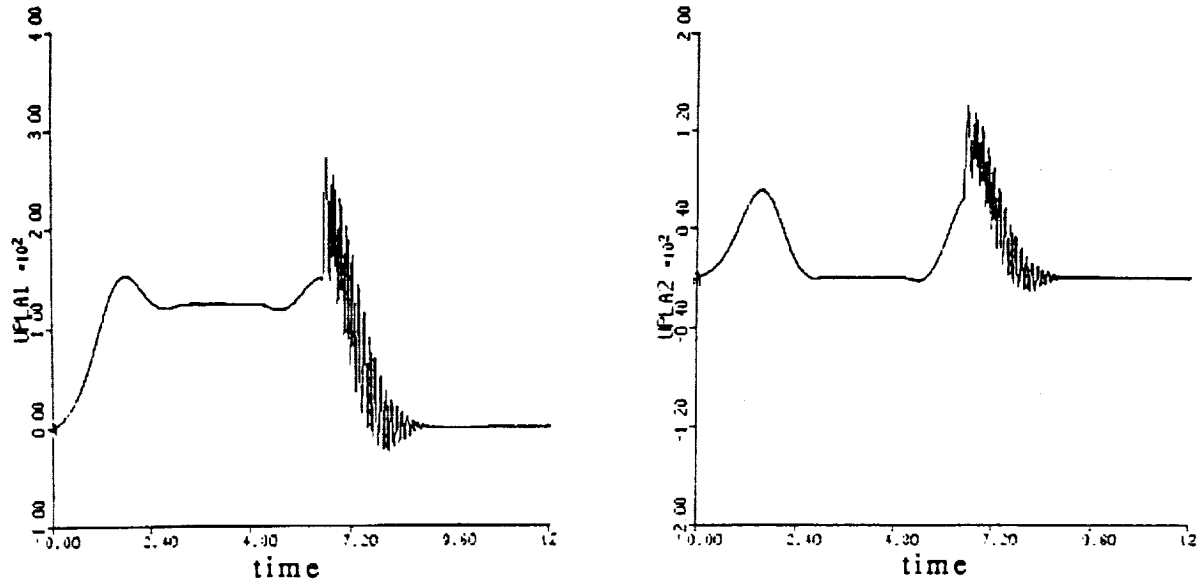


Figure 4.6: Plot of error between the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" when a sudden load change is present.

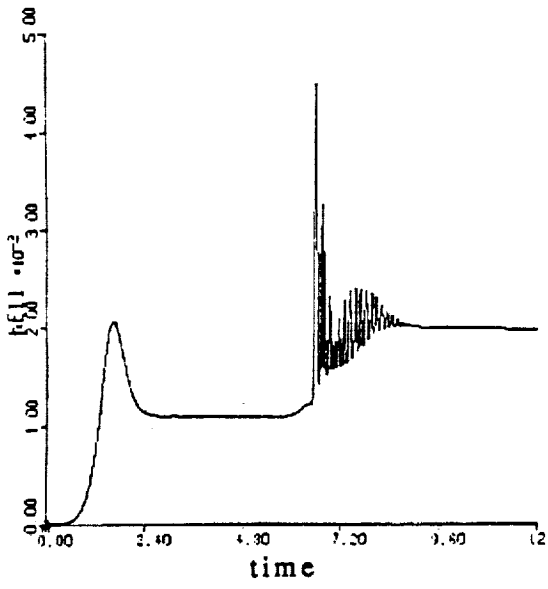
Figure 4.7(a) and (b) show the values of command torque and one of the gains respectively. Again, as expected, the discontinuity can be observed at 6.5 sec., but all the magnitudes remain bounded and achieve a steady state.

The error between the responses can be made as small as desired by increasing the ratio between D_p and τ (i.e. D_p/τ) and increasing the weights T and $\bar{T}$ accordingly to achieve the desired results. The larger the allowable values of T and $\bar{T}$, the smaller the error that can be achieved.

To try to reduce the high frequency terms present at the time of the load change the derivative term described in section 2.4 was incorporated into the algorithm for the next simulation. We set α = 0.0065 and left unchanged all the other parameters of the algorithm given in eq. (4.14).



(a)



(b)

Figure 4.7: Plots using the "Kaufman algorithm" when a sudden load change is present of (a) the command torques (N-m) to both joints and (b) one of the controller gains.

Figure 4.8 shows the resulting error between the plant and model's response, and it is clear that the high frequency oscillations have been notably reduced. Figure 4.9 shows the command torque and one of the gains of the controller, and again, the high frequency oscillations are greatly diminished. The disadvantage of this approach, however, is that as figure 4.8 shows, the error at the transients increases by a factor of three when compared to the results obtained when no derivative term was used (figure 4.6). A compromise must be reached between the reduction of the high frequency terms and the size of the error by choosing the proper value of α. A larger value of a will create a larger error but at the same time dampen out the oscillations.

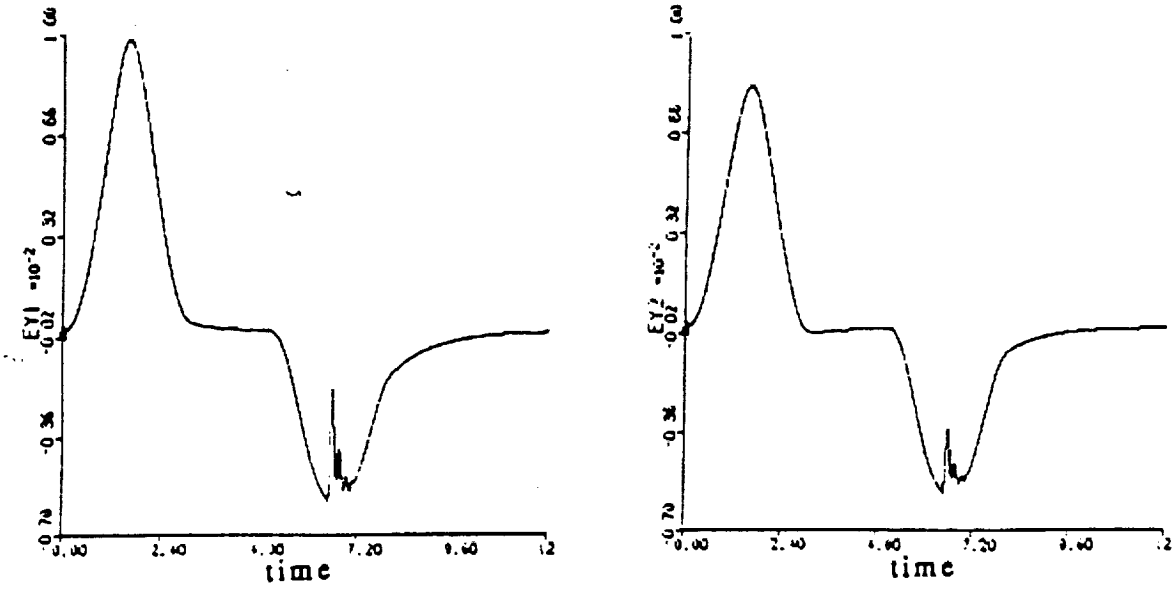
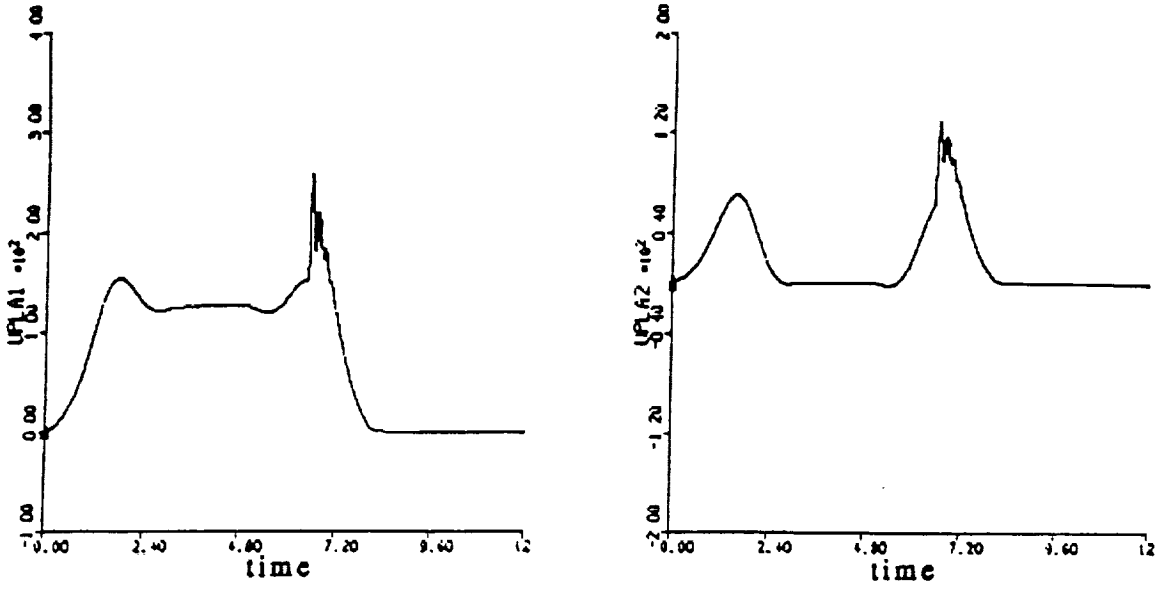
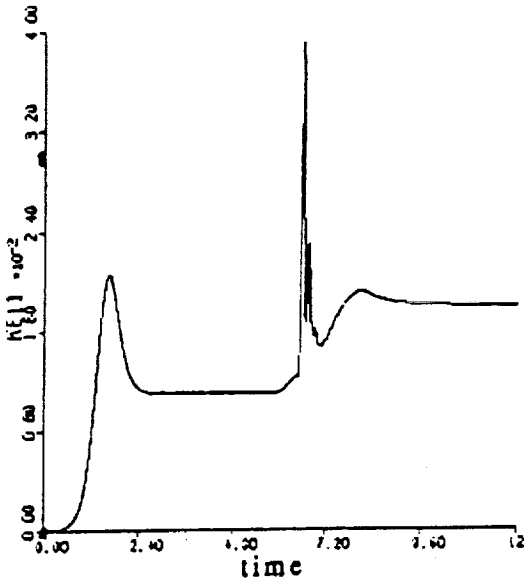


Figure 4.8: Plot of error between the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" with a derivative term (α = 0.0065) when a sudden load change is present.



(a)



(b)

Figure 4.9: Plots using the "Kaufman algorithm" with a derivative term (α = 0.0065) when a sudden load change is present of (a) the command torques (N-m) to both joints and (b) one of the controller gains.

The simulations presented show that this modification to the original algorithm is successful in controlling the given model of the PUMA 560 even in the presence of sudden changes in the arm's payload.

4.3.3 Derivative Algorithm

In this section we will show the results of some simulations when a zero at the origin of the feedforward was introduced to achieve asymptotic stability. This algorithm was previously described in section 2.3. The values used for the parameters of the algorithm were the following:

$$(4.15) \qquad D_p = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

$$T = \overline{T} = \begin{bmatrix} 1.0 & 0 & 0 & & & \\ 0 & 1.0 & 0 & & 0 & \\ 0 & 0 & 1.0 & & & \\ & & & 1.0 & 0 & 0 \\ & 0 & & 0 & 1.0 & 0 \\ & & & 0 & 0 & 1.0 \end{bmatrix} \times 10^4$$

$$a = b = 50.0$$

Figure 4.10 shows the difference between the plant and model outputs when no sudden change in the load is present. It shows that the system has zero steady state error, however, it does takes this algorithm twice as long to reach steady state than using the "Kaufman algorithm".

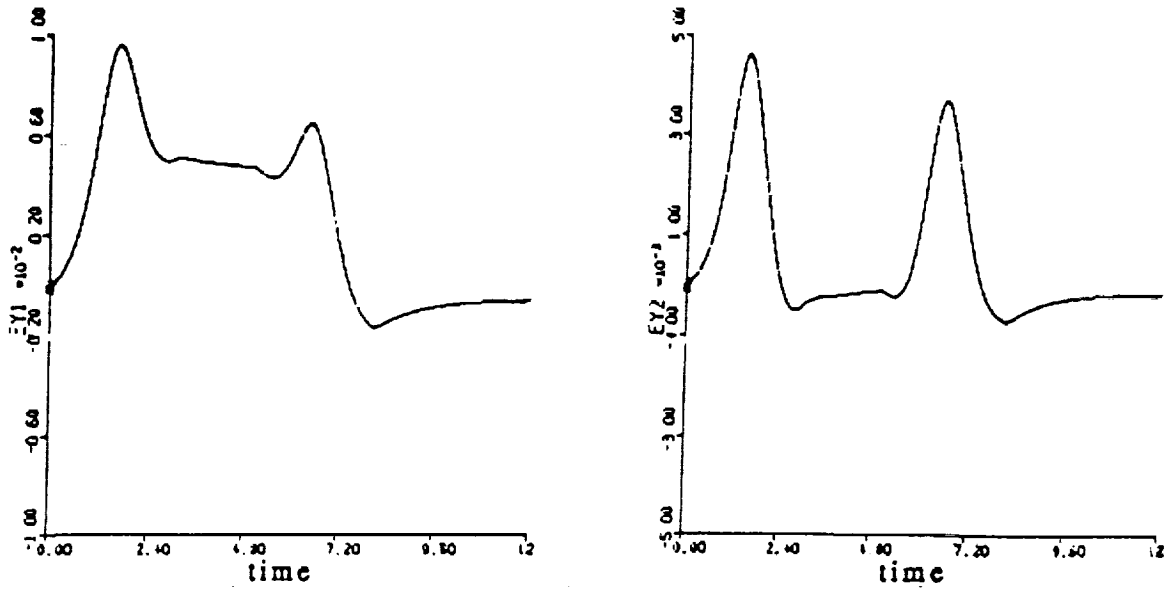


Figure 4.10: Plot of the error between the plant and model outputs (rad.) vs. time (sec.) using the "Derivative algorithm" for no change in the arm's load.

We also performed the simulation when a sudden change in the arm's load occurred at 6.5 sec. and the plots of the difference between plant an model outputs appear in Figure 4.11. A discontinuity appears at the time of the load change, however, the plant's output still tracks the model's output asymptotically. Notice, that in these simulations we did not have the high frequency oscillations which were present when the "Kaufman algorithm" was used, however, the error present in this case is twice as large than before. In addition, Figure 4.12 shows the command torques to the joints of the robot and one of the gains of the controller. These results contain the discontinuity, but they also achieve a steady state and in the case of the control torques they appear to be the same as the ones obtained using the "Kaufman algorithm".

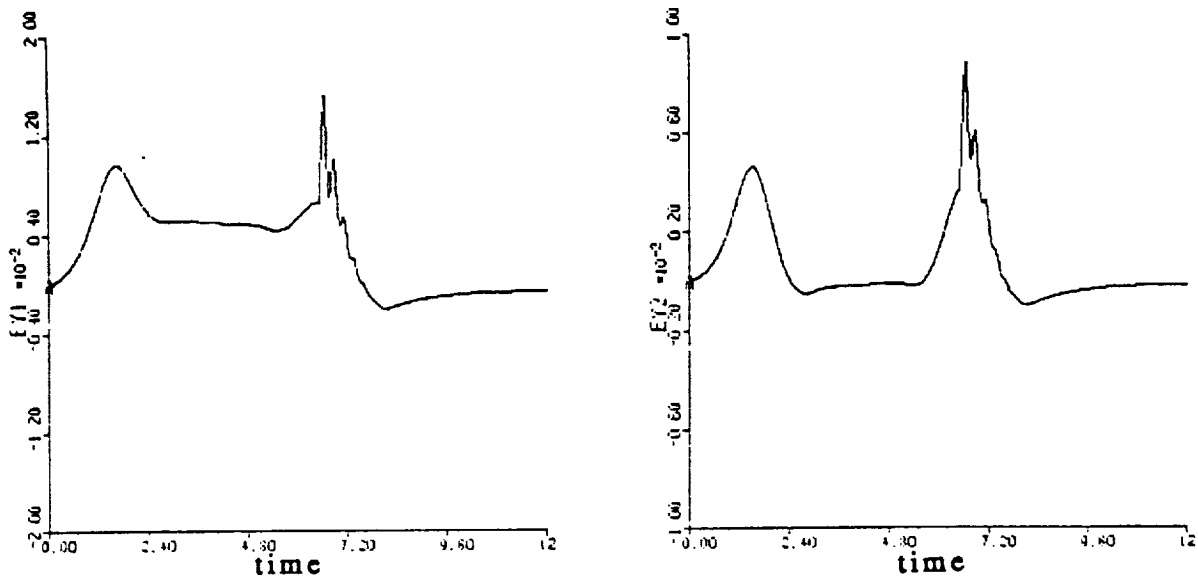
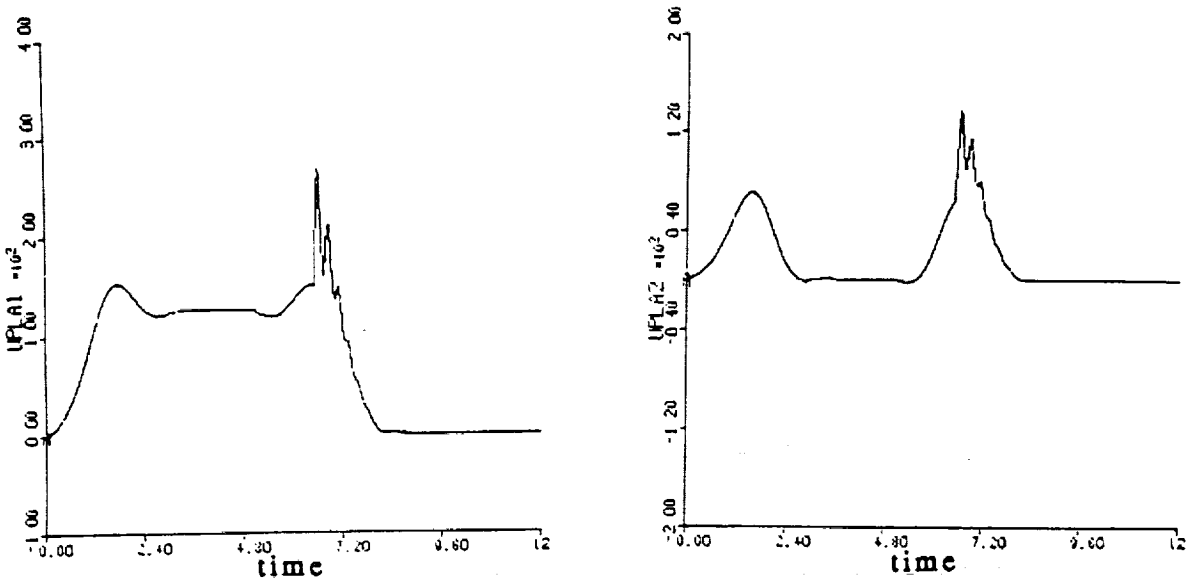


Figure 4.11: Plot of error between the plant and model outputs (rad.) vs. time (sec.) using the "Derivative algorithm" when a sudden load change is present.

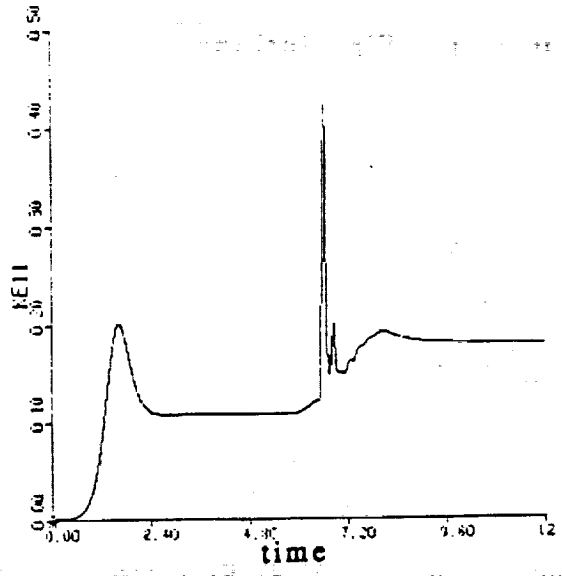
4.4 Discrete Simulations

The previous simulations were conducted as if the interaction between the plant and the controller was continuous. In this section we will change the program used to simulate the system in such a way that it takes into account the sampling period which is used by the controller to get the information it requires about the plant (i.e. joint encoder readings). The I/O program resident in the Unimation controller for the PUMA 560 arm allows the sampling period to be chosen as 7, 14, 28, or 56 ms. [11], and for these simulations we chose a value of 7 ms. To implement this, our program updated the joint angles from the robot to be used by the controller and the command torque calculated by the controller once every sample period.

In the simulations, the "Kaufman algorithm" was used with the following values for its parameters:



(a)



(b)

Figure 4.12: Plots using the "Derivative algorithm" when a sudden load change is present of (a) the command torques (N-m) to both joints and (b) one of the controller gains.

$$(4.16) \qquad D_p = \begin{bmatrix} 6 & 0 \\ 0 & 6 \end{bmatrix}$$

$$T = \overline{T} = \begin{bmatrix} 0.2 & 0 & 0 & & & \\ 0 & 0.2 & 0 & & 0 & \\ 0 & 0 & 1.4 & & & \\ & & & 1.4 & 0 & 0 \\ & 0 & & 0 & 1.4 & 0 \\ & & & 0 & 0 & 1.4 \end{bmatrix} \times 10^4$$

$$\tau = 0.01$$

which are the same as those used in section 4.3.2.

The resulting response of the system appears in Figure 4.13. We can see that the difference between the plant and model outputs is small, however, we can appreciate some high frequency oscillations of small magnitude. As the D_p to τ ratio and the weight matrices are increased, the magnitude of these oscillations decreases.

Another method of decreasing the magnitude of these oscillations is to include the a term as before. Figure 4.14 shows the results of the simulation using $\alpha = 0.1$. We can clearly appreciate that the magnitude of the oscillations has decreased and that they are barely perceptible in the plot of the response.

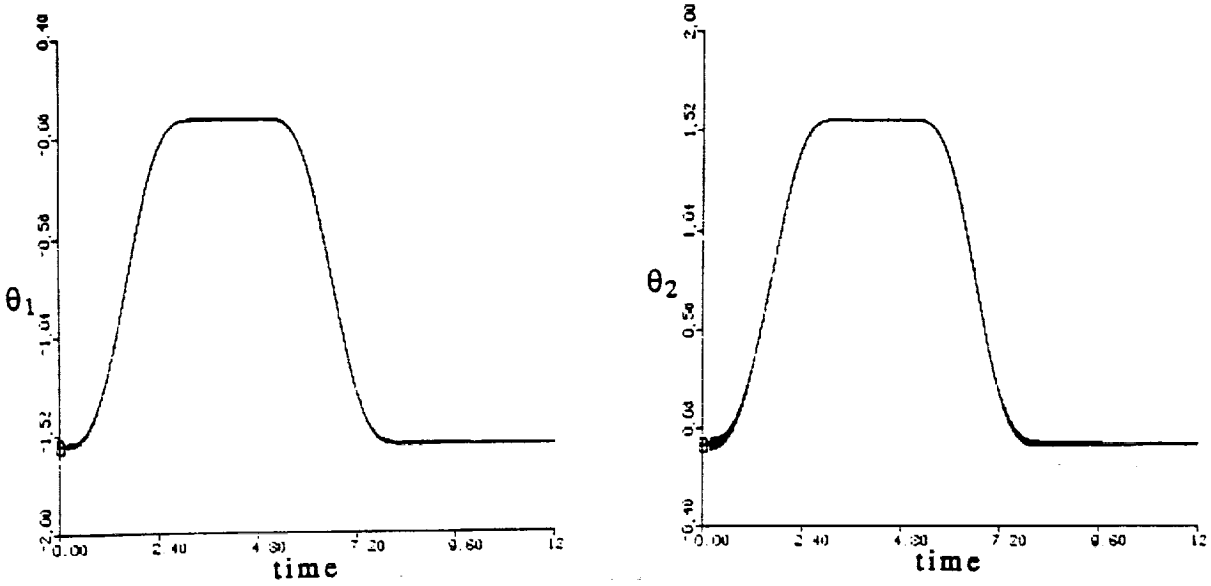


Figure 4.13:Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" when discrete control is simulated.

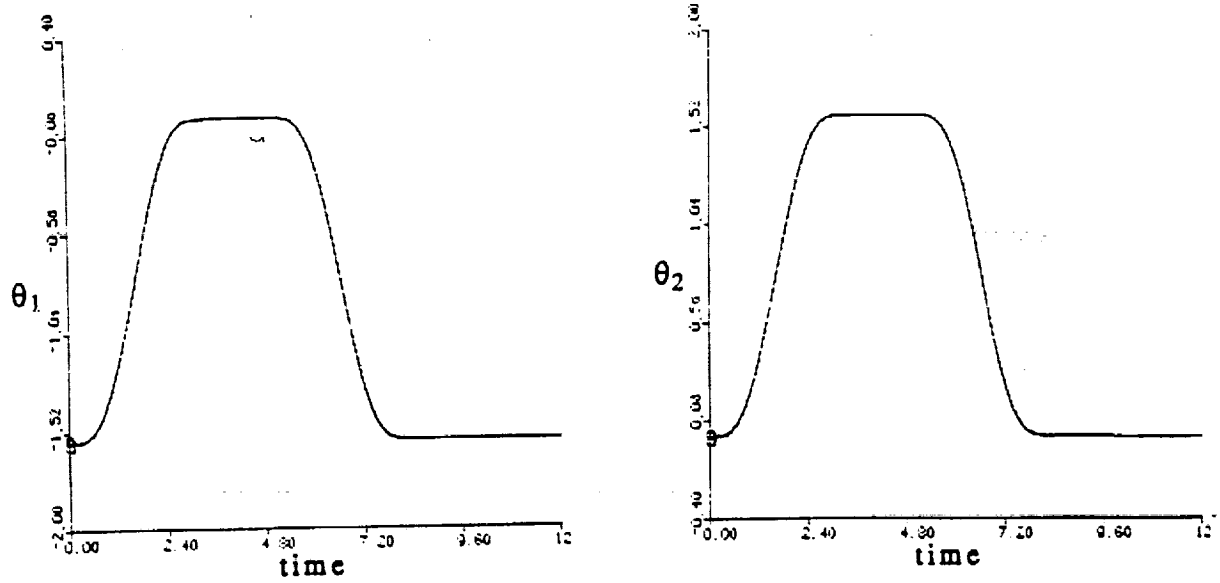


Figure 4.14:Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" when discrete control is simulated and a derivative term is used to augment the plant's output.

4.5 Decentralized Control

In this section we are concerned with the idea of treating each joint angle and its input torque as independent from each other. In other words, we want to use a first order controller to find the command torque of each one of the joints independently of each other. The advantage of such a system is that it is easier to implement and that it involves less calculations and is therefore faster. Both, the "Kaufman Algorithm" and the "Derivative algorithm" were considered in simulating the application of decentralized control to the PUMA 560 robot.

4.5.1 Kaufman Algorithm

To implement this algorithm we used the following parameters for the two first order controllers:

$$(4.17) \qquad D_p = 0.001$$

$$T = \overline{T} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} \times 10^4$$

$$\tau = 50.0$$

The results of the simulation are presented in Figures 4.15 and 4.16. Figure 4.15 shows the response of the model and the plant. We can see that the second joint angle tracks the model consistently, however, the first joint angle has a large error between 2 and 5 seconds. Therefore, as expected the results of using decentralized control are not as good as when the coupling between the joints is considered. Figure 4.16(a) and (b) shows the command control and

one of the gains of the controllers. We can see that there are oscillations present which are not desired, however, all the parameters are bounded.

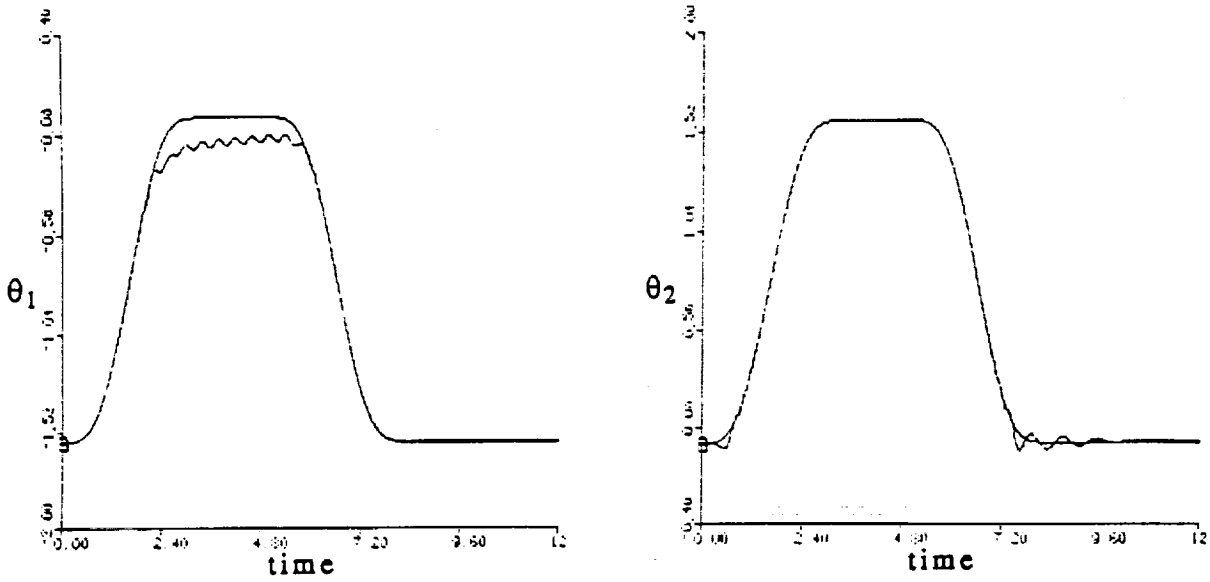
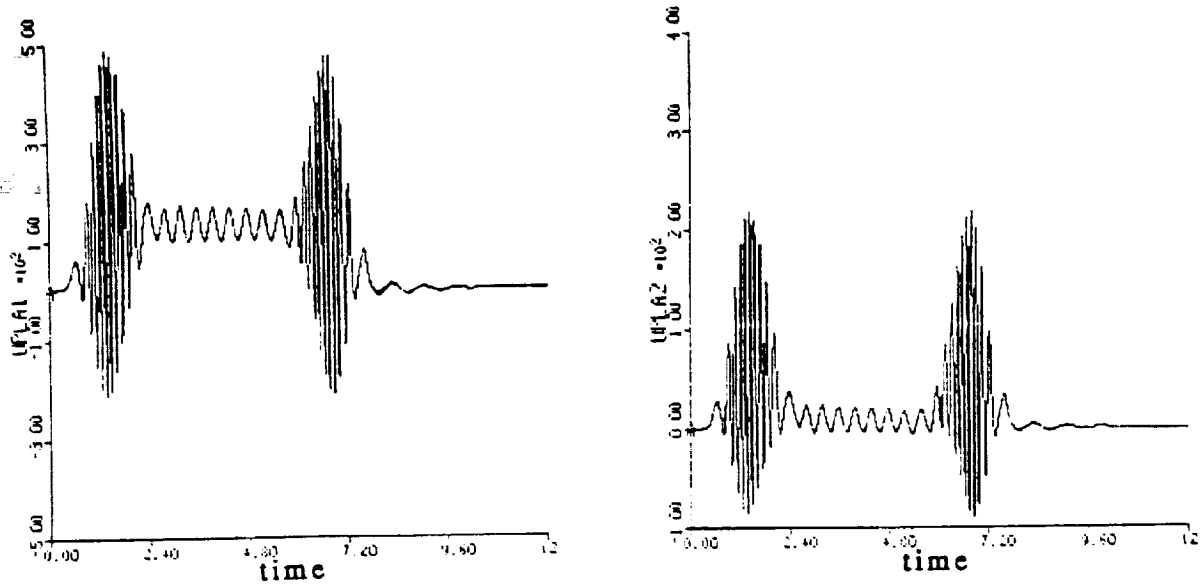
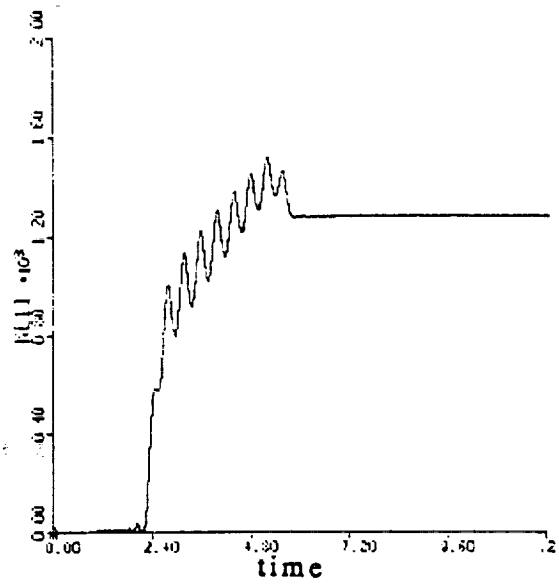


Figure 4.15: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" for decentralized control.



(a)



(b)

Figure 4.16: Plots using the "Kaufman algorithm" for decentralized control of (a) the command torque for each joint and (b) one of the gains.

To see if the problem of the oscillations can be reduced, the plant was augmented by a derivative term as explained before using a value of α = 0.05. Figures 4.17 and 4.18 show the results of this change, and we can observe that the oscillations are eliminated.

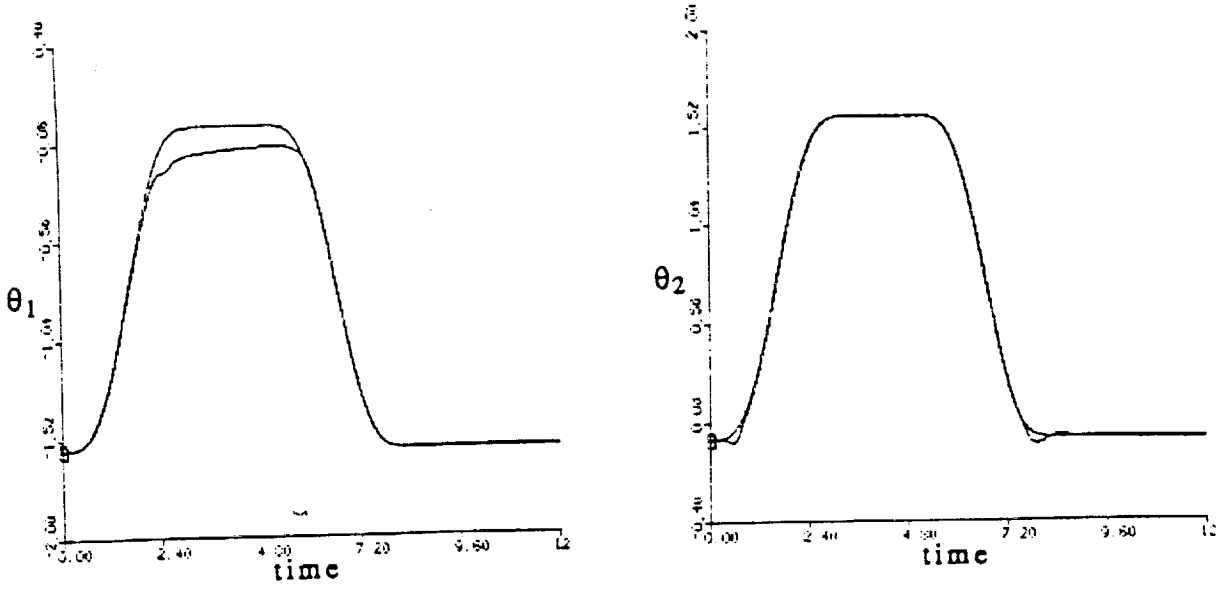
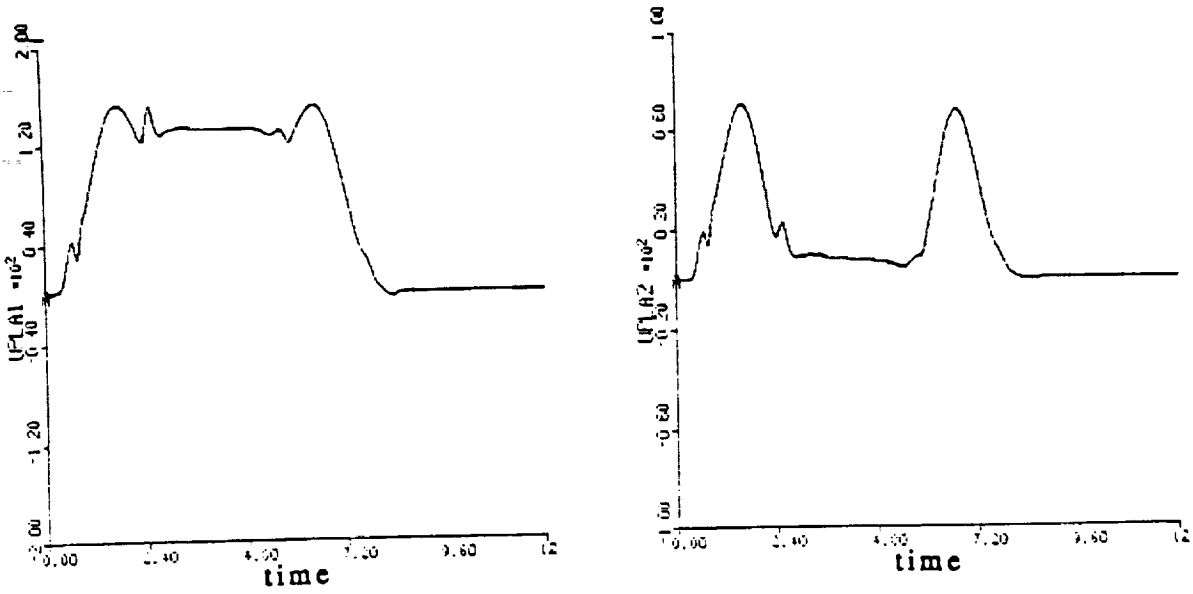
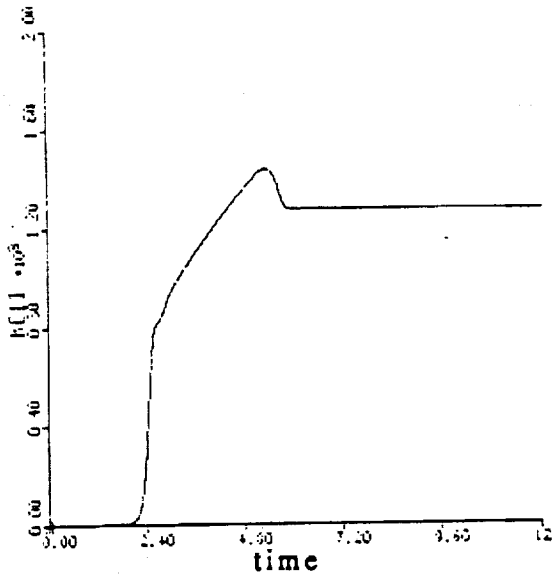


Figure 4.17: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Kaufman algorithm" for decentralized control with a derivative term augmenting the plant.



(a)



(b)

Figure 4.18: Plots using the "Kaufman algorithm" for decentralized control with a derivative term augmenting the plant of (a) the command torque for each joint and (b) one of the gains.

4.5.2 Derivative Algorithm

Finally we implement the decentralized control using the "Derivative algorithm". The parameters used for the two first order controllers used in each of the joints were the following:

$$D_p = 0.1 \tag{4.18}$$

$$T = \overline{T} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} \times 10^4$$

$$a = b = 50.0$$

The results of the simulation appear in Figures 4.19 and 4.20. Figure 4.19 shows the outputs of the plant and model, and again, as expected we can see that the tracking is not as good as what was obtained in section 4.3. The command input to both joints and one of the gains are plotted in figure 4.20. It is apparent that there are some oscillations; however, they are not as extreme as those obtained using the "Kaufman algorithm" with no derivative term augmenting the plant.

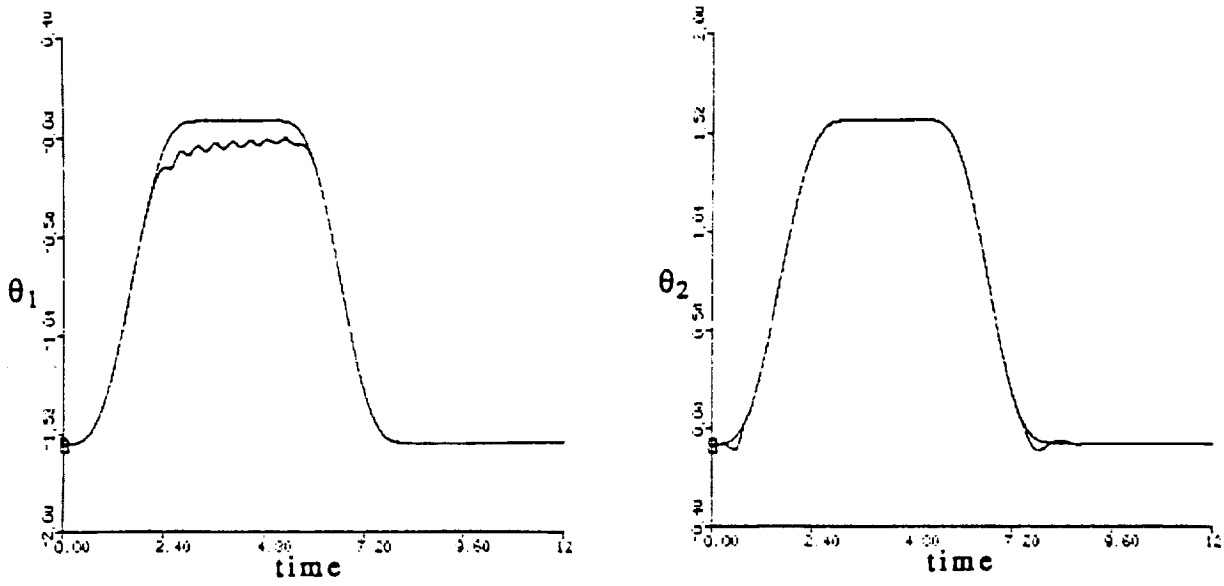
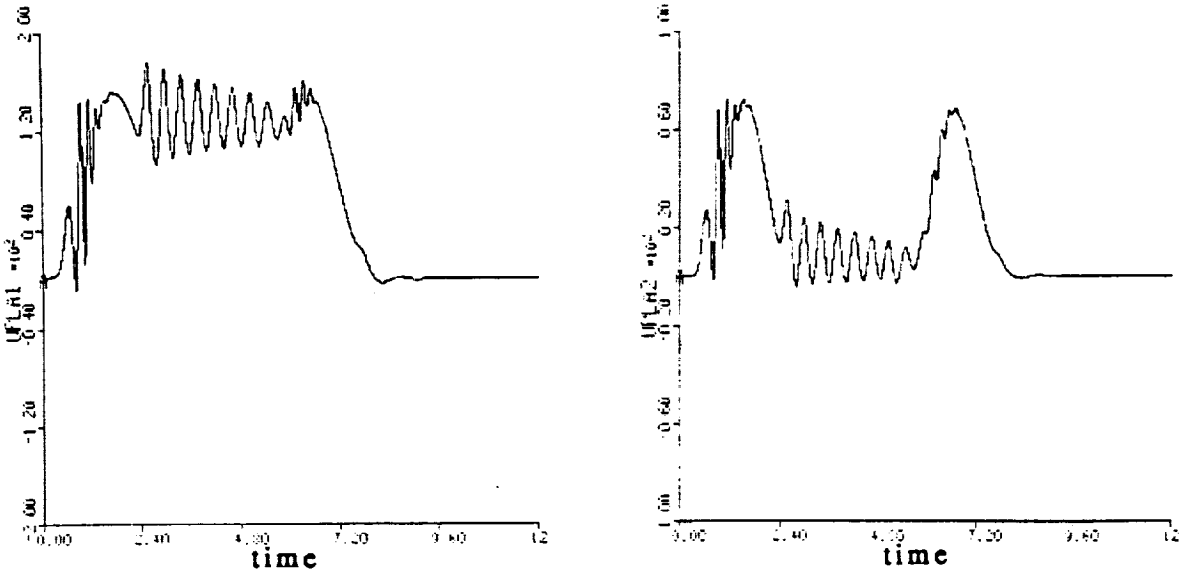
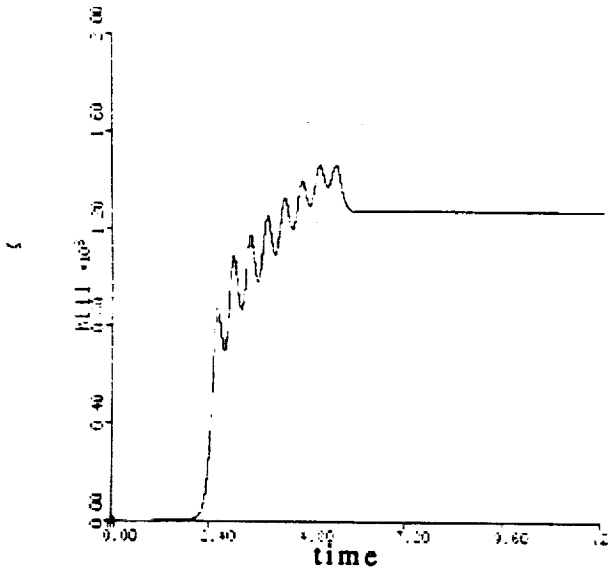


Figure 4.19: Plot of the plant and model outputs (rad.) vs. time (sec.) using the "Derivative algorithm" for decentralized control.



(a)



(b)

Figure 4.20: Plots using the "Derivative algorithm" for decentralized control of (a) the command torque for each joint and (b) one of the gains.

4.6 Summary

This chapter was devoted to the control of the Unimation PUMA 560 robot. It gave a complete description of the model used to simulate the robot arm and it showed the results of controlling the arm using the different algorithms presented in Chapters 1 and 2. The results of the simulations show that it possible to use model reference adaptive control to operate this type of robot. The modifications introduced to the previous MRAC algorithms achieve the desired result of eliminating the steady state error present in the response.

In addition, simulations showing the results of discrete implementation of the MRAC and decentralized control of the robot were carried out. The results show that these cases can achieve good results; however, the responses are not as good as those obtained in the normal simulations.

CHAPTER 5
Overview

5.1 Discussion

In this project we simulated the implementation of several MRAC algorithms to control two types of robots: a single-link flexible jointed arm and a model of 2 links of the Unimation PUMA 560 manipulator. It was clear that the existing MRAC algorithms used had major problems. The original algorithm explained in Section 1.2.3 had the serious limitation that it restricted its application to a very limited range of plants (almost strictly positive real (ASPR) systems). This introduced the need to find modifications so to make the algorithm applicable to a wider class of plants. An adjustment was proposed by BarKana (see Section 1.2.4) which expanded the types of plants that could be controlled with the algorithm, however, it did not achieve asymptotic tracking because it led to a steady state error. In our simulations, it was observed that this steady state error could, in some instances, be quite large and that it would change depending on the size of the load that the robot was carrying.

These results motivated the development of further modifications to the existing MRAC algorithms. These modifications had the goal of achieving asymptotic tracking, while at the same time expanding the class of controlled plants beyond those which are ASPR. This project displays two of these new algorithms which we called the "Kaufman algorithm" (Section 2.2) and the "Derivative algorithm" (Section 2.3) respectively. The simulations of the control of both robots using these algorithms were successful and showed that the problems described above were solved.

Certain simulations were carried out to observe the performance of the algorithms for a decentralized case of the PUMA robot, that is, each one of the to joint angles was controlled as a separate system. The results obtained, as expected, were not as good as for the normal operation, however, in cases were very fast computation times are required and accuracy can be sacrificed, this can yield acceptable results. Even so this should not usually be a necessity since the normal algorithm involves few computations.

Simulations of discrete control of the PUMA robot were also performed. These showed that we can obtain good results for the discrete case. However, there were high frequency terms present in the response which required the introduction of a derivative term to the output in order to weaken them. The only side effect of this is that the error during the transients is slightly increased depending on the weight given to the derivative term.

Comparing the results between the Kaufman and Derivative algorithms we could make several observations. First of all, the error in tracking (during the transients and changes in the arm's load) tended to be smaller for the "Kaufman algorithm". In addition, it was easier to adjust this algorithm to obtain a satisfactory response of the system, and it was generally less affected by changes in the plant's parameters. However, in using the "Derivative algorithm" the presence of high frequency oscillations was less frequent. Therefore, our recommendation for anyone using these algorithms is that they first try to solve their control problem using one of the two, and if it does not yield satisfactory results then the other should be tried instead.

In all the cases we looked at the control torque that was applied to the joint angles, and at some of the adaptive gains. It was observed that the torque

and the gain's magnitude remained bounded throughout the simulations. We also observed, as expected, that these parameters adapted when a change occurred in the plant or in the model command input .

In summary, these algorithms can be successfully used in control simulations of different types of robots. In addition they have the advantage that they are easy to implement because no there is no need to have any knowledge of the plant's parameters and because they can be readily applicable to MIMO systems without a great increase in the complexity of the calculations.

5.2 Future Work

This project dealt only with computer simulations of the systems, therefore, the logical continuation is to actually implement the algorithms to control a real robot. This step is very important in validating the value of using command generator tracker based model reference adaptive control.

Another area in which some additional work is possible is in implementing some type of theoretical rules about the choice of the parameters used in the implementation of the algorithms (i.e. Dp, τ, T, $\bar{T}$). This might require some knowledge of the system to be worked with, however, in most cases we have some knowledge available about the plant that will be controlled.

Finally, in the discrete simulations other sample times should be considered. All the work done in the discrete simulations performed for this project involved using a sample time of 7 ms., but sample times of 14, 28, and 56 ms. are also possible with the PUMA 560 manipulator.

APPENDIX A
ACSL Programs

This appendix contains a listing of all the ACSL programs used in the simulation of the MRAC algorithms and the different types of robots. The following are the names and a brief description of the programs listed:

BKFLEX "BarKana algorithm" used on single-link flexible-joint arm.

HKFLEX "Kaufman algorithm" used on single-link flexible-joint arm.

JDFLEX "Derivative algorithm" used on single-link flexible-joint arm.

PUMABK "BarKana algorithm" used on PUMA 560 model.

PUMAHK "Kaufman algorithm" used on PUMA 560 model.

PUMADHK "Kaufman algorithm" used on PUMA 560 model (discrete case).

PUMAJD "Derivative algorithm" used on PUMA 560 model.

Now we give a description of the variables with which the user must be concerned in order to properly operate these programs, this, by no means, is an exhaustive listing of all the variables used in the programs.

The following types of variables appear in all programs: DP, TAU, TN, and TB, which correspond to the algorithm parameters D_p, τ, T, and $\bar{T}$ respectively (see Chapter 1 for a description of these parameters). In the programs involving the flexible arm, DP is a constant and TN and TB are (3x3) matrices because the plant is SISO. For the programs simulating the PUMA 560, since we have two inputs and two outputs and two first order models, DP will be a (2x2) square matrix, and TN and TB are (6x6) matrices. In all the programs the weighting matrices TN and TB are broken up into the terms that act on the

error, the command input, and model states respectively and are assumed to be diagonal. In summary, the following variables compose the variable types described above for the different type of robot:

Flexible arm:

$$DP = DP$$

$$TN = \begin{bmatrix} TEN & 0 & 0 \\ 0 & TXN & 0 \\ 0 & 0 & TUN \end{bmatrix}$$

$$TB = \begin{bmatrix} TEB & 0 & 0 \\ 0 & TXB & 0 \\ 0 & 0 & TUB \end{bmatrix}$$

PUMA 560:

$$DP = \begin{bmatrix} DP1 & 0 \\ 0 & DP2 \end{bmatrix}$$

$$TN = \begin{bmatrix} TEN & 0 & 0 & & & \\ 0 & TEN & 0 & & 0 & \\ 0 & 0 & TUN & & & \\ & & & TUN & 0 & 0 \\ & 0 & & 0 & TXN & 0 \\ & & & 0 & 0 & TXN \end{bmatrix}$$

$$TB = \begin{bmatrix} TEB & 0 & 0 & & & \\ 0 & TEB & 0 & & 0 & \\ 0 & 0 & TUB & & & \\ & & & TUB & 0 & 0 \\ & 0 & & 0 & TXB & 0 \\ & & & 0 & 0 & TXB \end{bmatrix}$$

Therefore, for example, if the user is controlling the flexible arm and he wants $D_p = 6$ and weighting matrices

$$TB = TN = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

then all he has to do is to let DP = 6, TEB = TEN = 5, TXB = TXN = 6, and TUB = TUN = 7. Notice that we assumed that all the weighting matrices and DP matrices are diagonal, and that the weights acting on all the errors, command inputs, and states are the same. Therefore, there are many combinations which are not achievable due to these assumptions made in the program. However, making all the combinations available would cluster the programs with variables.

The other variable which appears in all the programs is TAU, and corresponds to the parameter τ (see Section 1.2.4, eq (1.34)). This is always a constant except when using the "Derivative algorithm", where there are two TAY's (see Section 2.3, eq (2.13)). Therefore in all the programs that use the "Derivative algorithm" (PUMAJD and JDFLEX) the user will have to specify two constants: TAU1 and TAU2 which correspond to a and b respectively.

Now we come to variables which are used only in some of the programs. These variables include DEC and ALPHA. The term ALPHA appears only in the programs PUMAHK and PUMADHK, and it implements the parameter that is

described in Section 2.4. This is a constant, and the user sets it to the desired value. If it is left equal to zero the program operates without adding a derivative term (i.e. as if it doesn't exist). The term DEC appears only in the programs PUMAHK, PUMADHK, and PUMAJD. It should only have one of two possible values, either 0 or 1. If it is set to 0, the program implements decentralized control on the system, if it is set to 1 normal control is implemented (see Section 4.5).

Finally, there are several control variables which are important to mention. These include FIN, IALG, and CINT. FIN just sets the time (in seconds) at which the simulation stops. Therefore if we want the simulation to end at 60 sec just set FIN = 60. IALG determines the algorithm that ACSL uses to calculate the integrals, in the simulations this was set to 9. For more information on this variable see [12]. CINT sets the communication interval in ACSL and is usually set in our simulations to 0.001. For more information on this variable see [12].

We will not go into describing the command input variables, the model variables, or the plant variables, in any more detail. If the user needs to change either the model, the command input, or the plant description, he can refer to the program listings which appear next.

```
*************************************************************************
*                                                                       *
*                                BKFLEX                                 *
*                                                                       *
*************************************************************************

PROGRAM BKFLEX

INITIAL
"Gives initial conditions and values for all constants"

        "Model Constants"
        "Second order model of the form: (N0)/((-1/D0)s+1)"

        CONSTANT    N0 = 1.0
        CONSTANT    D0 = -1.0
        CONSTANT    MIC1 = 0.0

        "Plant Constants: (Kexp(-sT0)/(s + A))"

        CONSTANT    I = 0.031,    J = 0.004,    B = 0.007
        CONSTANT    K = 31.0,     MGL = 0.8
        CONSTANT    PIC1 = 0.0,   PIC2 = 0.0,   PIC3 = 0.0
        CONSTANT    PIC4 = 0.0

        "Adaptive Gain Initial Conditions:"

        CONSTANT    KEIC = 0.0,    KUIC = 0.0
        CONSTANT    KX1IC = 0.0

        "Scaling Coefficients (used in gain calculation):"

        CONSTANT    TEN = 1.0,    TUN = 1.0,    TXN = 1.0
        CONSTANT    TEB = 1.0,    TUB = 1.0,    TXB = 1.0

        "Feedforward Constants:"

        CONSTANT    DP = .1,   TAU = 0.1,    DIC = 0.0

        "Used to stabilize flexible system (Ghorbel):"

        CONSTANT    KV = 0.0

        "These constants tell the system when to drop load"

        CONSTANT          DROP = 15.
        CONSTANT          NEWMGL=0

        "Square wave constants (to create input):"

        CONSTANT    START1 = 0.0
        CONSTANT    PERIOD = 14.0,   WIDTH  = 7.0

        "Program Control Constants:"
```

```
        CONSTANT   F1N = 28.0
        CINTERVAL  CINT = 0.01

        "Set all variables to zero:"

        CONSTANT   INPUT1 = 0.0,   UPLANT = 0.0,   KXOUT = 0.0
        CONSTANT   KEOUT  = 0.0,   KUOUT  = 0.0,   ERROR = 0.0
        CONSTANT   KE     = 0.0,   KX1    = 0.0,   U0    = 0.0
        CONSTANT   KU     = 0.0,   XMOD   = 0.0
        CONSTANT   DPPLA  = 0.0,   YPLANT = 0.0

END $ "of INITIAL"

DYNAMIC
DERIVATIVE

        "Input to the System (square wave):"

        U0 = 2*PULSE(START1,PERIOD,WIDTH)-1
"       U02 = PULSE(START2,PERIOD,WIDTH)    "
"       U0  = U01 - U02 "

        "Model Description:"

        XMOD   = INTEG(D0*XMOD - D0*U0, MIC1)
        YMODEL = N0*XMOD

        "Plant Description:"

        X1PLA  = INTEG(X2PLA, PIC1)
        X2PLA  = INTEG(-(MGL/I)*SIN(X1PLA)-(K/I)*(X1PLA-X3PLA), PIC
        X3PLA  = INTEG(X4PLA, PIC3)
        X4PLA  = INTEG(-(B/J)*X4PLA+(K/J)*(X1PLA-X3PLA)+UPLANT/J, PI
        YPLANT = X1PLA

        "Feedforward Gain (Dp(s)):"

        DUMM1  = 1/TAU
        XDP    = INTEG(-DUMM1*XDP + DUMM1*DP*UPLANT, DIC)
        DPPLA  = XDP

        "Adaptive Gains:"

        IE     = INTEG((ERROR**2)*TEN, KEIC)
        IX     = INTEG((ERROR*XMOD)*TXN, KX1IC)
        IU     = INTEG((ERROR*U0)*TUN, KUIC)
        KE     = ERROR**2*TEB + IE
        KX1    = ERROR*XMOD*TXB + IX
        KU     = ERROR*U0*TUB + IU

        "Output of the Adaptive Gains:"

        KEOUT = (ERROR*KE)
        KXOUT = (KX1*XMOD)
```

```
        KUOUT = (KU*U0)

        "Plant Input:"

        UPLANT = KXOUT+KUOUT+KEOUT+KV*(X2PLA-X4PLA)

        "Change the load"

        PROCEDURAL

                IF(DROP.GE.T)MGL=0.8
                IF(T.GT.DROP)MGL=NEWMGL
                END

END $ "of DERIVATIVE"

        "Error Calculation:"

        ERROR = (YMODEL)-(YPLANT+DPPLA)

        "Actual Error:"

        ACERR = YPLANT - YMODEL

        "Specify Termination Condition:"

        TERMT(T.GE.FIN)

END $ "of DYNAMIC"
END $ "of PROGRAM"
```

```
*****************************************************************
*                                                               *
*                            HKFLEX                             *
*                                                               *
*****************************************************************

PROGRAM HKFLEX

INITIAL
"Gives initial conditions and values for all constants"

        "Model Constants"
        "Second order model of the form: N0/(D0s+1)"

        CONSTANT    N0 = 1.0
        CONSTANT    D0 = -1.0
        CONSTANT    MIC1 = 0.0

        "Plant Constants: (K*exp(-sT0))/(s + A)"

        CONSTANT    I = 0.031,    J = 0.004,    B = 0.007
        CONSTANT    K = 31.0,     MGL = 0.8
        CONSTANT    PIC1 = 0.0,   PIC2 = 0.0,   PIC3 = 0.0
        CONSTANT    PIC4 = 0.0

        "Adaptive Gain Initial Conditions:"

        CONSTANT    KEIC = 0.0,    KUIC = 0.0
        CONSTANT    KX1IC = 0.0

        "Scaling Coefficients (used in gain calculation):"

        CONSTANT    TEN = 1.0,     TUN = 1.0,    TXN = 1.0
        CONSTANT    TEB = 1.0,     TUB = 1.0,    TXB = 1.0

        "Feedforward Constants:"

        CONSTANT    DP = .1,   TAU = 0.1,    DIC = 0.0

        "Square Wave Constants (to create input):"

        CONSTANT    START1 = 0.0
        CONSTANT    PERIOD = 60.0,   WIDTH  = 30

        "These constant tells the system when to drop the load"

        CONSTANT    DROP = 15.0
        CONSTANT    NEWMGL = 0.0, INIMGL = 0.8

        "Plant input constant (see paper by Ghorbel et al)"

        CONSTANT    KV = 0.0

        "Program Control Constants:"
```

```
        CONSTANT    FIN = 28.0
        CONSTANT    CINT = 0.01

        "Initialize all variables used in program to zero:"

        CONSTANT    INPUT1 = 0.0,   KEOUT = 0.0,    XDP = 0.0
        CONSTANT    KXOUT  = 0.0,   KUOUT = 0.0,    KE = 0.0
        CONSTANT    KX1    = 0.0,   KU    = 0.0
        CONSTANT    UPLANT = 0.0,   ERROR = 0.0,    X1PLA = 0.0
        CONSTANT    X2PLA  = 0.0,   X3PLA = 0.0,    X4PLA = 0.0

END $ "of INITIAL"

DYNAMIC
DERIVATIVE

        "Input to the System:"

        U0 = 2*PULSE(START1,PERIOD,WIDTH)-1

        "Model Description:"

        YMODEL = N0*XMOD
        XMOD   = INTEG(D0*XMOD - D0*U0, MIC1)

        "Plant Description:"

        X1PLA  = INTEG(X2PLA, PIC1)
        X2PLA  = INTEG(-(MGL/I)*SIN(X1PLA)-(K/I)*(X1PLA-X3PLA), PIC2
        X3PLA  = INTEG(X4PLA, PIC3)
        X4PLA  = INTEG(-(B/J)*X4PLA+(K/J)*(X1PLA-X3PLA)+UPLANT/J, PI
        YPLANT = X1PLA

        "Feedforward Gains (Dp(s)):"

        DUMM1  = 1/TAU
        XDP    = INTEG(-DUMM1*XDP - DUMM1*DP*KE*ERROR, DIC)
        DPPLA  = XDP

        "Adaptive Gains:"

        INTE   = INTEG((ERROR**2)*TEN, KEIC)
        INTX   = INTEG((ERROR*XMOD)*TXN, KX1IC)
        INTU   = INTEG((ERROR*U0)*TUN, KUIC)
        KE     = (ERROR**2)*TEB + INTE
        KX1    = ERROR*XMOD*TXB + INTX
        KU     = ERROR*U0*TUB + INTU

        "Output of the Adaptive Gains:"

        KEOUT = (ERROR*KE)
        KXOUT = (KX1*XMOD)
        KUOUT = (KU*U0)
```

```
            "Plant Input:"

            UPLANT =  KXOUT + KUOUT + KEOUT + KV*(X2PLA - X4PLA)

            "Drop the load"

            PROCEDURAL (MGL = NEWMGL,INIMGL)
            IF(T.GE.DROP)MGL = NEWMGL
            IF(T.LE.DROP)MGL = INIMGL
            END

END $ "of DERIVATIVE"

            "Error Calculation:"

            ERROR = YMODEL - YPLANT + DPPLA

            "Actual Error:"

            ACERR = YPLANT - YMODEL

            "Specify Termination Condition:"

            TERMT(T.GE.FIN)

END $ "of DYNAMIC"
END $ "of PROGRAM"
```

```
*****************************************************************
*                                                               *
*                                                               *
*                            JDFLEX                             *
*                                                               *
*****************************************************************

PROGRAM JDFLEX

INITIAL
"Gives initial conditions and values for all constants"

        "Model Constants"
        "Second order model of the form: (N0)/((-1/D0)s+1)"

        CONSTANT    N0 = 1.0
        CONSTANT    D0 = -1.0
        CONSTANT    MIC1 = 0.0

        "Plant Constants: (Kexp(-sT0)/(s + A))"

        CONSTANT    I    = 0.031,   J = 0.004,    B = 0.007
        CONSTANT    K    = 31.0,    MGL = 0.8
        CONSTANT    PIC1 = 0.0,     PIC2 = 0.0,   PIC3 = 0.0
        CONSTANT    PIC4 = 0.0

        "Adaptive Gain Initial Conditions:"

        CONSTANT    KEIC = 0.0,    KUIC = 0.0
        CONSTANT    KX1IC = 0.0

        "Scaling Coefficients (used in gain calculation):"

        CONSTANT    TEN = 1.0,    TUN = 1.0,    TXN = 1.0
        CONSTANT    TEB = 1.0,    TUB = 1.0,    TXB = 1.0

        "Feedforward Constants:"

        CONSTANT    DP = .10,     TAU1 = 0.1,    TAU2 = 0.1
        CONSTANT    DIC1 = 0.0,   DIC2 = 0.0

        "Square wave constants (to create input):"

        CONSTANT    START1 = 0.0
        CONSTANT    PERIOD = 14.0,  WIDTH  = 7.0

        "Program Control Constants:"

        CONSTANT    FIN = 28.0      $ "Execution stops in 28 seconds"
        CINTERVAL   CINT = 0.001     $ "Communication Interval"

        "The following variables are used to change the load:"

        CONSTANT    DROP = 15.0,    NEWMGL = 0.0
```

```
        "Set all variables to zero:"

        CONSTANT   INPUT  = 0.0,   UPLANT = 0.0,   KXOUT = 0.0
        CONSTANT   KEOUT  = 0.0,   KUOUT  = 0.0,   ERROR = 0.0
        CONSTANT   KE     = 0.0,   KX1    = 0.0,   U0    = 0.0
        CONSTANT   KU     = 0.0,   XMOD   = 0.0,   XDP1  = 0.0
        CONSTANT   DPPLA  = 0.0,   YPLANT = 0.0,   ACERR = 0.0
        CONSTANT   XDP2   = 0.0

END $ "of INITIAL"

DYNAMIC
DERIVATIVE

        "Input to the System (square wave):"

        U0 = 2*PULSE(START1,PERIOD,WIDTH)-1

        "Model Description:"

        XMOD   = INTEG(D0*XMOD - D0*U0, MIC1)
        YMODEL = N0*XMOD

        "Plant Description:"

        X1PLA  = INTEG(X2PLA, PIC1)
        X2PLA  = INTEG(-(MGL/I)*SIN(X1PLA)-(K/I)*(X1PLA-X3PLA), PIC2)
        X3PLA  = INTEG(X4PLA, PIC3)
        X4PLA  = INTEG(-(B/J)*X4PLA+(K/J)*(X1PLA-X3PLA)+UPLANT/J, PIC
        YPLANT = X1PLA

        "Feedforward Gain (Dp(s)):"

        INPUT  = DP*UPLANT/TAU1
        XDP1   = INTEG(INPUT-(TAU2/TAU1)*XDP1+XDP2, DIC1)
        XDP2   = INTEG(-(1/TAU1)*XDP1, DIC2)
        DPPLA  = XDP1

        "Adaptive Gains:"

        IE     = INTEG((ERROR**2)TEN, KEIC)
        IX     = INTEG((ERROR*XMOD)*TXN, KX1IC)
        IU     = INTEG((ERROR*U0)*TUN, KUIC)
        KE     = ERROR**2*TEB + IE
        KX1    = ERROR*XMOD*TXB + IX
        KU     = ERROR*U0*TUB + IU

        "Output of the Adaptive Gains:"

        KEOUT = (ERROR*KE)
        KXOUT = (KX1*XMOD)
        KUOUT = (KU*U0)

        "Plant Input:"
```

```
        UPLANT = KXOUT + KUOUT + KEOUT

        "Change the load:"

        PROCEDURAL
          IF(DROP.GE.T)MGL=0.8
          IF(T.GT.DROP)MGL=NEWMGL
        END

END $ "of DERIVATIVE"

        "Error Calculation:"

        ERROR = YMODEL - YPLANT - DPPLA

        "Actual Error:"

        ACERR = YPLANT - YMODEL

        "Specify Termination Condition:"

        TERMT(T.GE.FIN)

END $ "of DYNAMIC"
```

```
*****************************************************************
*                                                               *
*                            PUMABK                             *
*                                                               *
*****************************************************************

PROGRAM PUMABK

"This program implements the BK algorithm for 2 links of PUMA 560"
"robot arm (for decentralized control let variable DEC = 0.0)"

INITIAL

        "Model Constants"
        "Two first order models of the form: N/(TAUMs + 1)"

        CONSTANT    NUM1 = 1.0, NUM2 = 1.0, TAUM1 = 0.1, TAUM2 = 0.1
        CONSTANT    MIC1 = -1.570795, MIC2 = 0.0

        "Plant Constants: (two link robot)"

        CONSTANT    M  = 10.0, L = 0.432, A1 = 3.82
        CONSTANT    A2 = 2.12, A3 = 0.71, A4 = 81.82, A5 = 24.06
        CONSTANT    V1 = 1.0, V2 = 0.5, V3 = 1.0, V4 = 0.5
        CONSTANT    PIC1 = -1.570795, PIC2 = 0.0, PIC3 = 0.0
        CONSTANT    PIC4 = 0.0

        "Adaptive Gain Initial Conditions:"

        CONSTANT    KE11IC=0.0, KE12IC=0.0, KE21IC=0.0, KE22IC=0.0
        CONSTANT    KX11IC=0.0, KX12IC=0.0, KX21IC=0.0, KX22IC=0.0
        CONSTANT    KU11IC=0.0, KU12IC=0.0, KU21IC=0.0, KU22IC=0.0

        "Constants used for the adaptive gains"

        CONSTANT    TEN = 1.0, TEB = 1.0
        CONSTANT    TXN = 1.0, TXB = 1.0
        CONSTANT    TUN = 1.0, TUB = 1.0

        "Feedforward Constants:"

        CONSTANT    DP11  = 3.0, DP12 = 3.0, DP21 = 3.0, DP22 = 3.0
        CONSTANT    DIC1 = 0.0, DIC2 = 0.0
        CONSTANT    TAU = 0.1

        "Constants for the cycloidal reference trajectories:"

        CONSTANT    PI = 3.14159
        CONSTANT    UR1F = 0.0, UR2F = 1.570795

        "Constants to change parameters (drop the load):"

        CONSTANT    MINI = 10.0, MNEW = 0.0, TDROP = 10.0
```

```
        "Program control constants:"

        CONSTANT    FIN = 10.0, CINT=0.001

        "Set certain variables initially to zero:"

        CONSTANT    EY1 = 0.0, EY2 = 0.0
        CONSTANT    ERROR1 = 0.0, ERROR2 = 0.0

END $ "of INITIAL"

DYNAMIC
DERIVATIVE

        "System input (cycloid) (There are 2 reference inputs)"

        UR1 = FCNSW(T-3,(-PI/2.+.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),0,
        UR2 = FCNSW(T-3,(.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),(PI/2.),F
        F1 = FCNSW(T-5.,0,GG1,GG1)
        F2 = FCNSW(T-5.,(PI/2.),GG2,GG2)
        GG1 = FCNSW(T-8.,FH1,(-PI/2.),(-PI/2.))
        GG2 = FCNSW(T-8,FH2,0.,0.)
        FH1 = -.25*(2.*PI*(T-5.)/3-SIN(2.*PI*(T-5.)/3.))
        FH2 = PI/2.+FH1

        "Model Description: (Two first order models)"

        XMOD1 = INTEG((-XMOD1+NUM1*UR1)/TAUM1,MIC1)
        YMOD1 = XMOD1
        XMOD2 = INTEG((-XMOD2+NUM2*UR2)/TAUM2,MIC2)
        YMOD2 = XMOD2

        "Plant Description:"
        "Y1 = Theta1, Y2 = Theta2, Y3 = Theta1Dot, Y4 = Theta2Dot"

        M11  = A1 + A2*COS(Y2)
        M12  = A3 + (A2/2)*COS(Y2)
        M21  = M12
        M22  = A3

        N1   = -(A2*SIN(Y2))*(Y3*Y4+(Y4**2)/2)
        N2   = A2*SIN(Y2)*(Y3**2)/2

        G1   = A4*COS(Y1) + A5*COS(Y1+Y2)
        G2   = A5*COS(Y1+Y2)

        H1   = V1*Y3 + V2*SIGN(1.0,Y3)
        H2   = V3*Y4 + V4*SIGN(1.0,Y4)

        J11  = -L*(SIN(Y1)+SIN(Y1+Y2))
        J12  = -L*SIN(Y1+Y2)
        J21  = L*(COS(Y1)+COS(Y1+Y2))
        J22  = L*COS(Y1+Y2)
```

```
JD11 = -L*Y3*COS(Y1) + JD12
JD12 = -L*(Y3+Y4)*COS(Y1+Y2)
JD21 = -L*Y3*SIN(Y1) + JD22
JD22 = -L*(Y3+Y4)*SIN(Y1+Y2)

G    = 9.81

"The following matrix is multiplied times the vector"
"of derivatives in the equation of the robot, therefore"
"it will have to be inverted"

MTI11 = -M11 - M*(J11**2 + J21**2)
MTI12 = -M12 - M*(J11*J12 + J21*J22)
MTI21 = -M21 - M*(J11*J12 + J21*J22)
MTI22 = -M22 - M*(J12**2 + J22**2)

DETMT = MTI11*MTI22 - MTI12*MTI21

INV11 = MTI22/DETMT
INV12 = -MTI12/DETMT
INV21 = -MTI21/DETMT
INV22 = MTI11/DETMT

"Now we calculate the right hand side of the differential"
"equation for the last two state variables (y3 and y4):"

DUM1  = (JD11*Y3+JD12*Y4)
DUM2  = (JD21*Y3+JD22*Y4)
ARHS1 = N1+G1+H1-UPLA1+M*(G*J21+J11*DUM1+J21*DUM2)
ARHS2 = N2+G2+H2-UPLA2+M*(G*J22+J12*DUM1+J22*DUM2)
RHS1  = INV11*ARHS1 + INV12*ARHS2
RHS2  = INV21*ARHS1 + INV22*ARHS2

"Now we can calculate the state variables:"

Y1 = INTEG(Y3, PIC1)
Y2 = INTEG(Y4, PIC2)
Y3 = INTEG(RHS1, PIC3)
Y4 = INTEG(RHS2, PIC4)

"Feedforward gain (DP(s)):"

DUM = 1/TAU
XDP1 = INTEG(-DUM*XDP1+DUM*(DP11*UPLA1+DP12*UPLA2),DIC1)
XDP2 = INTEG(-DUM*XDP2+DUM*(DP21*UPLA1+DP22*UPLA2),DIC2)
DPPLA1 = XDP1
DPPLA2 = XDP2

"Adaptive Gains:"

IE11 = INTEG((ERROR1**2)*TEN, KE11IC)
IE12 = INTEG((ERROR1*ERROR2)*TEN, KE12IC)
IE21 = INTEG((ERROR2*ERROR1)*TEN, KE21IC)
IE22 = INTEG((ERROR2**2)*TEN, KE22IC)
```

```
            IX11 = INTEG((ERROR1*XMOD1)*TXN, KX11IC)
            IX12 = INTEG((ERROR1*XMOD2)*TXN, KX12IC)
            IX21 = INTEG((ERROR2*XMOD1)*TXN, KX21IC)
            IX22 = INTEG((ERROR2*XMOD2)*TXN, KX22IC)
            IU11 = INTEG((ERROR1*UR1)*TUN, KU11IC)
            IU12 = INTEG((ERROR1*UR2)*TUN, KU12IC)
            IU21 = INTEG((ERROR2*UR1)*TUN, KU21IC)
            IU22 = INTEG((ERROR2*UR2)*TUN, KU22IC)

            KE11 = ERROR1**2*TEB + IE11
            KE12 = ERROR1*ERROR2*TEB + IE12
            KE21 = ERROR2*ERROR1*TEB + IE21
            KE22 = ERROR2**2*TEB + IE22
            KX11 = ERROR1*XMOD1*TXB + IX11
            KX12 = ERROR1*XMOD2*TXB + IX12
            KX21 = ERROR2*XMOD1*TXB + IX21
            KX22 = ERROR2*XMOD2*TXB + IX22
            KU11 = ERROR1*UR1*TUB + IU11
            KU12 = ERROR1*UR2*TUB + IU12
            KU21 = ERROR2*UR1*TUB + IU21
            KU22 = ERROR2*UR2*TUB + IU22

            "Output of the Adaptive Gains:"

            KEOUT1 = ERROR1*KE11 + ERROR2*KE12
            KEOUT2 = ERROR1*KE21 + ERROR2*KE22
            KXOUT1 = XMOD1*KX11 + XMOD2*KX12
            KXOUT2 = XMOD1*KX21 + XMOD2*KX22
            KUOUT1 = UR1*KU11 + UR2*KU12
            KUOUT2 = UR1*KU21 + UR2*KU22

            "Now we can obtain the Input to the Plant:"

            UPLA1 = KEOUT1 + KXOUT1 + KUOUT1
            UPLA2 = KEOUT2 + KXOUT2 + KUOUT2

            "The following lines change the load on the arm:"

            PROCEDURAL
            IF(T.LT.TDROP)M=MINI
            IF(T.GE.TDROP)M=MNEW
            END

            "Calculation of the actual and augmented errors:"

            EY1    = YMOD1 - Y1
            EY2    = YMOD2 - Y2
            ERROR1 = EY1 - DPPLA1
            ERROR2 = EY2 - DPPLA2
end
            "Specify termination condition:"

            TERMT(T.GE.FIN)
END $ "of DYNAMIC"
END $ "of PROGRAM"
```

```
*************************************************************************
*                                                                       *
*                                  PUMAHK                               *
*                                                                       *
*************************************************************************

PROGRAM PUMAHK

"This program implements HK MRAC algorithm for 2 links"
"of the PUMA 560 robot (DEC = 0.0 for decentralized control)"
"(Set ALPHA = positive constant to add derivative term)"

INITIAL

        "Model Constants"
        "Two first order models of the form: N/(TAUMs + 1)"

        CONSTANT    NUM1 = 1.0, NUM2 = 1.0, TAUM1 = 0.1, TAUM2 = 0.
        CONSTANT    MIC1 = -1.570795, MIC2 = 0.0

        "Plant Constants: (two link robot)"

        CONSTANT    M  = 10.0, L = 0.432, A1 = 3.82
        CONSTANT    A2 = 2.12, A3 = 0.71, A4 = 81.82, A5 = 24.06
        CONSTANT    V1 = 1.0, V2 = 0.5, V3 = 1.0, V4 = 0.5
        CONSTANT    PIC1 = -1.570795, PIC2 = 0.0, PIC3 = 0.0
        CONSTANT    PIC4 = 0.0

        "Adaptive Gain Initial Conditions:"

        CONSTANT    KE11IC=0.0, KE12IC=0.0, KE21IC=0.0, KE22IC=0.0
        CONSTANT    KX11IC=0.0, KX12IC=0.0, KX21IC=0.0, KX22IC=0.0
        CONSTANT    KU11IC=0.0, KU12IC=0.0, KU21IC=0.0, KU22IC=0.0

        "Constants used for the adaptive gains"

        CONSTANT    TEN = 1.0, TEB = 1.0
        CONSTANT    TXN = 1.0, TXB = 1.0
        CONSTANT    TUN = 1.0, TUB = 1.0

        "Feedforward Constants:"

        CONSTANT    DP1  = 3.0, DP2 = 3.0
        CONSTANT    DIC1 = 0.0, DIC2 = 0.0
        CONSTANT    TAU = 0.1

        "Constants for the cycloidal reference trajectories:"

        CONSTANT    PI = 3.14159
        CONSTANT    UR1F = 0.0, UR2F = 1.570795

        "Constants to change parameters (drop the load):"

        CONSTANT    MINI = 10.0, MNEW = 0.0, TDROP = 10.0
```

```
        "Program control constants:"

        CONSTANT   FIN = 5.0, CINT=0.001

        "Set certain variables initially to zero:"

        CONSTANT   EY1 = 0.0, EY2 = 0.0, ERROR1 = 0.0, ERROR2 = 0.0

        "ThIS variable is set to 1.0 for normal control and to"
        "0 if we want to use decentralized control"

        CONSTANT DEC = 1.0

        "This constant is used to add derivative term"

        CONSTANT ALPHA = 0.0

END $ "of INITIAL"

DYNAMIC
DERIVATIVE

        "System input (cycloid) (There are 2 reference inputs)"

        UR1 = FCNSW(T-3,(-PI/2.+.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),0,E
        UR2 = FCNSW(T-3,(.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),(PI/2.),F
        F1 = FCNSW(T-5.,0,GG1,GG1)
        F2 = FCNSW(T-5.,(PI/2.),GG2,GG2)
        GG1 = FCNSW(T-8.,FH1,(-PI/2.),(-PI/2.))
        GG2 = FCNSW(T-8,FH2,0.,0.)
        FH1 = -.25*(2.*PI*(T-5.)/3-SIN(2.*PI*(T-5.)/3.))
        FH2 = PI/2.+FH1

        "Model Description: (Two first order models)"

        XMOD1 = INTEG((-XMOD1+NUM1*UR1)/TAUM1,MIC1)
        YMOD1 = XMOD1
        XMOD2 = INTEG((-XMOD2+NUM2*UR2)/TAUM2,MIC2)
        YMOD2 = XMOD2

        "Plant Description:"
        "Y1 = Theta1, Y2 = Theta2, Y3 = Theta1Dot, Y4 = Theta2Dot"

        M11  = A1 + A2*COS(Y2)
        M12  = A3 + (A2/2)*COS(Y2)
        M21  = M12
        M22  = A3

        N1   = -(A2*SIN(Y2))*(Y3*Y4+(Y4**2)/2)
        N2   = A2*SIN(Y2)*(Y3**2)/2

        G1   = A4*COS(Y1) + A5*COS(Y1+Y2)
        G2   = A5*COS(Y1+Y2)
```

```
H1    = V1*Y3 + V2*SIGN(1.0,Y3)
H2    = V3*Y4 + V4*SIGN(1.0,Y4)

J11   = -L*(SIN(Y1)+SIN(Y1+Y2))
J12   = -L*SIN(Y1+Y2)
J21   = L*(COS(Y1)+COS(Y1+Y2))
J22   = L*COS(Y1+Y2)

JD11 = -L*Y3*COS(Y1) + JD12
JD12 = -L*(Y3+Y4)*COS(Y1+Y2)
JD21 = -L*Y3*SIN(Y1) + JD22
JD22 = -L*(Y3+Y4)*SIN(Y1+Y2)

G     = 9.81

"The following matrix is multiplied times the vector"
"of derivatives in the equation of the robot, therefore"
"it will have to be inverted"

MTI11 = -M11 - M*(J11**2 + J21**2)
MTI12 = -M12 - M*(J11*J12 + J21*J22)
MTI21 = -M21 - M*(J11*J12 + J21*J22)
MTI22 = -M22 - M*(J12**2 + J22**2)

DETMT = MTI11*MTI22 - MTI12*MTI21

INV11 = MTI22/DETMT
INV12 = -MTI12/DETMT
INV21 = -MTI21/DETMT
INV22 = MTI11/DETMT

"Now we calculate the right hand side of the differential"
"equation for the last two state variables (y3 and y4):"

DUM1  = (JD11*Y3+JD12*Y4)
DUM2  = (JD21*Y3+JD22*Y4)
ARHS1 = N1+G1+H1-UPLA1+M*(G*J21+J11*DUM1+J21*DUM2)
ARHS2 = N2+G2+H2-UPLA2+M*(G*J22+J12*DUM1+J22*DUM2)
RHS1  = INV11*ARHS1 + INV12*ARHS2
RHS2  = INV21*ARHS1 + INV22*ARHS2

"Now we can calculate the state variables:"

Y1 = INTEG(Y3, PIC1)
Y2 = INTEG(Y4, PIC2)
Y3 = INTEG(RHS1, PIC3)
Y4 = INTEG(RHS2, PIC4)

"Feedforward gain (DP(s)):"

XDP1 = INTEG((-XDP1+DP1*KE11*ERROR1+DEC*DP1*KE12*ERROR2)/TAU,
XDP2 = INTEG((-XDP2+DEC*DP2*KE21*ERROR1+DP2*KE22*ERROR2)/TAU,
DPPLA1 = XDP1
DPPLA2 = XDP2
```

```
"Adaptive Gains:"

IE11 = INTEG((ERROR1**2)*TEN, KE11IC)
IE12 = INTEG((ERROR1*ERROR2)*TEN, KE12IC)
IE21 = INTEG((ERROR2*ERROR1)*TEN, KE21IC)
IE22 = INTEG((ERROR2**2)*TEN, KE22IC)
IX11 = INTEG((ERROR1*XMOD1)*TXN, KX11IC)
IX12 = INTEG((ERROR1*XMOD2)*TXN, KX12IC)
IX21 = INTEG((ERROR2*XMOD1)*TXN, KX21IC)
IX22 = INTEG((ERROR2*XMOD2)*TXN, KX22IC)
IU11 = INTEG((ERROR1*UR1)*TUN, KU11IC)
IU12 = INTEG((ERROR1*UR2)*TUN, KU12IC)
IU21 = INTEG((ERROR2*UR1)*TUN, KU21IC)
IU22 = INTEG((ERROR2*UR2)*TUN, KU22IC)

KE11 = ERROR1**2*TEB + IE11
KE12 = ERROR1*ERROR2*TEB + IE12
KE21 = ERROR2*ERROR1*TEB + IE21
KE22 = ERROR2**2*TEB + IE22
KX11 = ERROR1*XMOD1*TXB + IX11
KX12 = ERROR1*XMOD2*TXB + IX12
KX21 = ERROR2*XMOD1*TXB + IX21
KX22 = ERROR2*XMOD2*TXB + IX22
KU11 = ERROR1*UR1*TUB + IU11
KU12 = ERROR1*UR2*TUB + IU12
KU21 = ERROR2*UR1*TUB + IU21
KU22 = ERROR2*UR2*TUB + IU22

"Output of the Adaptive Gains:"

KEOUT1 = ERROR1*KE11 + DEC*ERROR2*KE12
KEOUT2 = DEC*ERROR1*KE21 + ERROR2*KE22
KXOUT1 = XMOD1*KX11 + DEC*XMOD2*KX12
KXOUT2 = DEC*XMOD1*KX21 + XMOD2*KX22
KUOUT1 = UR1*KU11 + DEC*UR2*KU12
KUOUT2 = DEC*UR1*KU21 + UR2*KU22

"Now we can obtain the Input to the Plant:"

UPLA1 = KEOUT1 + KXOUT1 + KUOUT1
UPLA2 = KEOUT2 + KXOUT2 + KUOUT2

"The following lines change the load on the arm:"

PROCEDURAL
IF(T.LT.TDROP)M=MINI
IF(T.GE.TDROP)M=MNEW
END

"Calculation of the actual and augmented errors:"

EY1    = YMOD1 - Y1
EY2    = YMOD2 - Y2
ERROR1 = EY1 - DPPLA1 - ALPHA*Y3
```

```
        ERROR2 = EY2 - DPPLA2 - ALPHA*Y4

end
        "Specify termination condition:"

        TERMT(T.GE.FIN)

END $ "of DYNAMIC"
END $ "of PROGRAM"
```

```
*****************************************************************
*                                                               *
*                          PUMADHK                              *
*                                                               *
*****************************************************************

PROGRAM PUMADHK

"This program implements the HK MRAC algorithm for a 2 link"
"PUMA 560 robot. (For decentralized control let DEC = 0.0)"
"(For derivative term set ALPHA to a small positive constant)"
"In addition discrete implementation is simulated"

INITIAL

        "Model Constants"
        "Two first order models of the form: N/(TAUMs + 1)"

        CONSTANT    NUM1 = 1.0, NUM2 = 1.0, TAUM1 = 0.1
        CONSTANT    TAUM2 = 0.1
        CONSTANT    MIC1 = -1.570795, MIC2 = 0.0

        "Plant Constants: (two link robot)"

        CONSTANT    M  = 10.0, L = 0.432, A1 = 3.82
        CONSTANT    A2 = 2.12, A3 = 0.71, A4 = 81.82, A5 = 24.06
        CONSTANT    V1 = 1.0, V2 = 0.5, V3 = 1.0, V4 = 0.5
        CONSTANT    PIC1 = -1.570795, PIC2 = 0.0, PIC3 = 0.0
        CONSTANT    PIC4 = 0.0

        "Adaptive Gain Initial Conditions:"

        CONSTANT    KE11IC=0.0, KE12IC=0.0, KE21IC=0.0, KE22IC=0.0
        CONSTANT    KX11IC=0.0, KX12IC=0.0, KX21IC=0.0, KX22IC=0.0
        CONSTANT    KU11IC=0.0, KU12IC=0.0, KU21IC=0.0, KU22IC=0.0

        "Constants used for the adaptive gains"

        CONSTANT    TEN = 1.0, TEB = 1.0
        CONSTANT    TXN = 1.0, TXB = 1.0
        CONSTANT    TUN = 1.0, TUB = 1.0

        "Feedforward Constants:"

        CONSTANT    DP1  = 3.0, DP2 = 3.0
        CONSTANT    DIC1 = 0.0, DIC2 = 0.0
        CONSTANT    TAU = 0.1

        "Constants for the cycloidal reference trajectories:"

        CONSTANT    PI = 3.14159
        CONSTANT    UR1F = 0.0, UR2F = 1.570795

        "Constants to change parameters (drop the load):"
```

```
        CONSTANT    MINI = 10.0, MNEW = 0.0, TDROP = 10.0

        "Program control constants:"

        CONSTANT    FIN = 5.0, CINT=0.001

        "Set certain variables initially to zero:"

        CONSTANT    EY1 = 0.0, EY2 = 0.0, ERROR1 = 0.0, ERROR2 = 0.0
        CONSTANT    UPD1 = 0.0, UPD2=0.0, Y1D = -1.570795, Y2D = 0.
        CONSTANT    UPLA1 = 0.0, UPLA2 = 0.0, Y1 = -1.570795
        CONSTANT    Y2 = 0.0

        "This variable equalS 1.0 if we want normal control, if we
        "want decentralized control set DEC = 0.0"

        CONSTANT    DEC = 1.0

        "This constant adds a derivative term to the output"

        CONSTANT    ALPHA = 0.0

END $ "of INITIAL"

DYNAMIC
DISCRETE

        "Set the interval of communication between computer and rob

        INTERVAL PERIOD = 0.007

        "The only things sampled are the input to the robot and the
        "current angles of its joints"

        UPD1 = UPLA1
        UPD2 = UPLA2
        Y1D = Y1
        Y2D = Y2
        Y3D = Y3
        Y4D = Y4

END $   "of DISCRETE"

DERIVATIVE

        "System input (cycloid) (There are 2 reference inputs)"

        UR1 = FCNSW(T-3,(-PI/2.+.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),0.
        UR2 = FCNSW(T-3,(.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),(PI/2.),
        F1 = FCNSW(T-5.,0,GG1,GG1)
        F2 = FCNSW(T-5.,(PI/2.),GG2,GG2)
        GG1 = FCNSW(T-8.,FH1,(-PI/2.),(-PI/2.))
        GG2 = FCNSW(T-8,FH2,0.,0.)
        FH1 = -.25*(2.*PI*(T-5.)/3-SIN(2.*PI*(T-5.)/3.))
```

```
FH2 = PI/2.+FH1

"Model Description: (Two first order models)"

XMOD1 = INTEG((-XMOD1+NUM1*UR1)/TAUM1,MIC1)
YMOD1 = XMOD1
XMOD2 = INTEG((-XMOD2+NUM2*UR2)/TAUM2,MIC2)
YMOD2 = XMOD2

"Plant Description:"
"Y1 = Theta1, Y2 = Theta2, Y3 = Theta1Dot, Y4 = Theta2Dot"

M11  = A1 + A2*COS(Y2)
M12  = A3 + (A2/2)*COS(Y2)
M21  = M12
M22  = A3

N1   = -(A2*SIN(Y2))*(Y3*Y4+(Y4**2)/2)
N2   = A2*SIN(Y2)*(Y3**2)/2

G1   = A4*COS(Y1) + A5*COS(Y1+Y2)
G2   = A5*COS(Y1+Y2)

H1   = V1*Y3 + V2*SIGN(1.0,Y3)
H2   = V3*Y4 + V4*SIGN(1.0,Y4)

J11  = -L*(SIN(Y1)+SIN(Y1+Y2))
J12  = -L*SIN(Y1+Y2)
J21  = L*(COS(Y1)+COS(Y1+Y2))
J22  = L*COS(Y1+Y2)

JD11 = -L*Y3*COS(Y1) + JD12
JD12 = -L*(Y3+Y4)*COS(Y1+Y2)
JD21 = -L*Y3*SIN(Y1) + JD22
JD22 = -L*(Y3+Y4)*SIN(Y1+Y2)

G    = 9.81

"The following matrix is multiplied times the vector"
"of derivatives in the equation of the robot, therefore"
"it will have to be inverted"

MTI11 = -M11 - M*(J11**2 + J21**2)
MTI12 = -M12 - M*(J11*J12 + J21*J22)
MTI21 = -M21 - M*(J11*J12 + J21*J22)
MTI22 = -M22 - M*(J12**2 + J22**2)

DETMT = MTI11*MTI22 - MTI12*MTI21

INV11 = MTI22/DETMT
INV12 = -MTI12/DETMT
INV21 = -MTI21/DETMT
INV22 = MTI11/DETMT
```

```
"Now we calculate the right hand side of the differential"
"equation for the last two state variables (y3 and y4):"

DUM1  = (JD11*Y3+JD12*Y4)
DUM2  = (JD21*Y3+JD22*Y4)
ARHS1 = N1+G1+H1-UPD1+M*(G*J21+J11*DUM1+J21*DUM2)
ARHS2 = N2+G2+H2-UPD2+M*(G*J22+J12*DUM1+J22*DUM2)
RHS1  = INV11*ARHS1 + INV12*ARHS2
RHS2  = INV21*ARHS1 + INV22*ARHS2

"Now we can calculate the state variables:"

Y1 = INTEG(Y3, PIC1)
Y2 = INTEG(Y4, PIC2)
Y3 = INTEG(RHS1, PIC3)
Y4 = INTEG(RHS2, PIC4)

"Feedforward gain (DP(s)):"

XDP1 = INTEG((-XDP1+DP1*KE11*ERROR1+DEC*DP1*KE12*ERROR2)/TAU
XDP2 = INTEG((-XDP2+DEC*DP2*KE21*ERROR1+DP2*KE22*ERROR2)/TAU
DPPLA1 = XDP1
DPPLA2 = XDP2

"Adaptive Gains:"

IE11 = INTEG((ERROR1**2)*TEN, KE11IC)
IE12 = INTEG((ERROR1*ERROR2)*TEN, KE12IC)
IE21 = INTEG((ERROR2*ERROR1)*TEN, KE21IC)
IE22 = INTEG((ERROR2**2)*TEN, KE22IC)
IX11 = INTEG((ERROR1*XMOD1)*TXN, KX11IC)
IX12 = INTEG((ERROR1*XMOD2)*TXN, KX12IC)
IX21 = INTEG((ERROR2*XMOD1)*TXN, KX21IC)
IX22 = INTEG((ERROR2*XMOD2)*TXN, KX22IC)
IU11 = INTEG((ERROR1*UR1)*TUN, KU11IC)
IU12 = INTEG((ERROR1*UR2)*TUN, KU12IC)
IU21 = INTEG((ERROR2*UR1)*TUN, KU21IC)
IU22 = INTEG((ERROR2*UR2)*TUN, KU22IC)

KE11 = ERROR1**2*TEB + IE11
KE12 = ERROR1*ERROR2*TEB + IE12
KE21 = ERROR2*ERROR1*TEB + IE21
KE22 = ERROR2**2*TEB + IE22
KX11 = ERROR1*XMOD1*TXB + IX11
KX12 = ERROR1*XMOD2*TXB + IX12
KX21 = ERROR2*XMOD1*TXB + IX21
KX22 = ERROR2*XMOD2*TXB + IX22
KU11 = ERROR1*UR1*TUB + IU11
KU12 = ERROR1*UR2*TUB + IU12
KU21 = ERROR2*UR1*TUB + IU21
KU22 = ERROR2*UR2*TUB + IU22

"Output of the Adaptive Gains:"
```

```
            KEOUT1 = ERROR1*KE11 + DEC*ERROR2*KE12
            KEOUT2 = DEC*ERROR1*KE21 + ERROR2*KE22
            KXOUT1 = XMOD1*KX11 + DEC*XMOD2*KX12
            KXOUT2 = DEC*XMOD1*KX21 + XMOD2*KX22
            KUOUT1 = UR1*KU11 + DEC*UR2*KU12
            KUOUT2 = DEC*UR1*KU21 + UR2*KU22

            "Now we can obtain the Input to the Plant:"

            UPLA1 = KEOUT1 + KXOUT1 + KUOUT1
            UPLA2 = KEOUT2 + KXOUT2 + KUOUT2

            "The following lines change the load on the arm:"

            PROCEDURAL
            IF(T.LT.TDROP)M=MINI
            IF(T.GE.TDROP)M=MNEW
            END

            "Calculation of the actual and augmented errors:"

            EY1    = YMOD1 - Y1D - ALPHA*Y3D
            EY2    = YMOD2 - Y2D - ALPHA*Y4D
            ERROR1 = EY1 - DPPLA1
            ERROR2 = EY2 - DPPLA2

end
            "Specify termination condition:"

            TERMT(T.GE.FIN)

END $ "of DYNAMIC"
END $ "of PROGRAM"
```

```
*****************************************************************
*                                                               *
*                            PUMAJD                             *
*                                                               *
*****************************************************************

PROGRAM PUMAJD

"This program implements JD algorithm for 2 links of the PUMA 560"
"robot arm (For decentralized control set variable DEC = 0)"

INITIAL

        "Model Constants"
        "Two first order models of the form: N/(TAUMs + 1)"

        CONSTANT    NUM1 = 1.0, NUM2 = 1.0, TAUM1 = 0.1
        CONSTANT    TAUM2 = 0.1
        CONSTANT    MIC1 = -1.570795, MIC2 = 0.0

        "Plant Constants: (two link robot)"

        CONSTANT    M  = 10.0, L = 0.432, A1 = 3.82
        CONSTANT    A2 = 2.12, A3 = 0.71, A4 = 81.82, A5 = 24.06
        CONSTANT    V1 = 1.0, V2 = 0.5, V3 = 1.0, V4 = 0.5
        CONSTANT    PIC1 = -1.570795, PIC2 = 0.0, PIC3 = 0.0
        CONSTANT    PIC4 = 0.0

        "Adaptive Gain Initial Conditions:"

        CONSTANT    KE11IC=0.0, KE12IC=0.0, KE21IC=0.0, KE22IC=0.0
        CONSTANT    KX11IC=0.0, KX12IC=0.0, KX21IC=0.0, KX22IC=0.0
        CONSTANT    KU11IC=0.0, KU12IC=0.0, KU21IC=0.0, KU22IC=0.0

        "Constants used for the adaptive gains"

        CONSTANT    TEN = 10000.0, TEB = 10000.0
        CONSTANT    TXN = 10000.0, TXB = 10000.0
        CONSTANT    TUN = 10000.0, TUB = 10000.0

        "Feedforward Constants:"

        CONSTANT    DP1 = 0.1, DP2 = 0.1
        CONSTANT    DIC1 = 0.0, DIC2 = 0.0, DIC3 = 0.0, DIC4 = 0.0
        CONSTANT    TAU1 = 50.0, TAU2 = 50.0

        "Constants for the cycloidal reference trajectories:"

        CONSTANT    PI = 3.14159
        CONSTANT    UR1F = 0.0, UR2F = 1.570795

        "Constants to change parameters (drop the load):"

        CONSTANT    MINI = 10.0, MNEW = 0.0, TDROP = 10.0
```

```
        "Program control constants:"

        CONSTANT   FIN = 12.0, CINT=0.001

        "Set certain variables initially to zero:"

        CONSTANT   EY1 = 0.0, EY2 = 0.0, ERROR1 = 0.0, ERROR2 = 0.0

        "The following variable is set to 1 for normal control"
        "and set to 0.0 if we want decentralized control:"

        CONSTANT   DEC = 1.0

END $ "of INITIAL"

DYNAMIC
DERIVATIVE

        "System input (cycloid) (There are 2 reference inputs)"

        UR1 = FCNSW(T-3,(-PI/2.+.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),0
        UR2 = FCNSW(T-3,(.25*(2.*PI*T/3.-SIN(2.*PI*T/3.))),(PI/2.),
        F1 = FCNSW(T-5.,0,GG1,GG1)
        F2 = FCNSW(T-5.,(PI/2.),GG2,GG2)
        GG1 = FCNSW(T-8.,FH1,(-PI/2.),(-PI/2.))
        GG2 = FCNSW(T-8,FH2,0.,0.)
        FH1 = -.25*(2.*PI*(T-5.)/3-SIN(2.*PI*(T-5.)/3.))
        FH2 = PI/2.+FH1

        "Model Description: (Two first order models)"

        XMOD1 = INTEG((-XMOD1+NUM1*UR1)/TAUM1,MIC1)
        YMOD1 = XMOD1
        XMOD2 = INTEG((-XMOD2+NUM2*UR2)/TAUM2,MIC2)
        YMOD2 = XMOD2

        "Plant Description:"
        "Y1 = Theta1, Y2 = Theta2, Y3 = Theta1Dot, Y4 = Theta2Dot"

        M11  = A1 + A2*COS(Y2)
        M12  = A3 + (A2/2)*COS(Y2)
        M21  = M12
        M22  = A3

        N1   = -(A2*SIN(Y2))*(Y3*Y4+(Y4**2)/2)
        N2   = A2*SIN(Y2)*(Y3**2)/2

        G1   = A4*COS(Y1) + A5*COS(Y1+Y2)
        G2   = A5*COS(Y1+Y2)

        H1   = V1*Y3 + V2*SIGN(1.0,Y3)
        H2   = V3*Y4 + V4*SIGN(1.0,Y4)

        J11  = -L*(SIN(Y1)+SIN(Y1+Y2))
```

```
J12  = -L*SIN(Y1+Y2)
J21  = L*(COS(Y1)+COS(Y1+Y2))
J22  = L*COS(Y1+Y2)

JD11 = -L*Y3*COS(Y1) + JD12
JD12 = -L*(Y3+Y4)*COS(Y1+Y2)
JD21 = -L*Y3*SIN(Y1) + JD22
JD22 = -L*(Y3+Y4)*SIN(Y1+Y2)

G    = 9.81

"The following matrix is multiplied times the vector"
"of derivatives in the equation of the robot, therefore"
"it will have to be inverted"

MTI11 = -M11 - M*(J11**2 + J21**2)
MTI12 = -M12 - M*(J11*J12 + J21*J22)
MTI21 = -M21 - M*(J11*J12 + J21*J22)
MTI22 = -M22 - M*(J12**2 + J22**2)

DETMT = MTI11*MTI22 - MTI12*MTI21

INV11 = MTI22/DETMT
INV12 = -MTI12/DETMT
INV21 = -MTI21/DETMT
INV22 = MTI11/DETMT

"Now we calculate the right hand side of the differential"
"equation for the last two state variables (y3 and y4):"

DUM1  = (JD11*Y3+JD12*Y4)
DUM2  = (JD21*Y3+JD22*Y4)
ARHS1 = N1+G1+H1-UPLA1+M*(G*J21+J11*DUM1+J21*DUM2)
ARHS2 = N2+G2+H2-UPLA2+M*(G*J22+J12*DUM1+J22*DUM2)
RHS1  = INV11*ARHS1 + INV12*ARHS2
RHS2  = INV21*ARHS1 + INV22*ARHS2

"Now we can calculate the state variables:"

Y1 = INTEG(Y3, PIC1)
Y2 = INTEG(Y4, PIC2)
Y3 = INTEG(RHS1, PIC3)
Y4 = INTEG(RHS2, PIC4)

"Feedforward gain (DP(s)):"

XDP1 = INTEG(-TAU2*XDP1+XDP2+DP1*UPLA1,DIC1)
XDP2 = INTEG(-XDP1,DIC2)
XDP3 = INTEG(-TAU2*XDP3+XDP4+DP2*UPLA2,DIC3)
XDP4 = INTEG(-XDP3,DIC4)
DPPLA1 = XDP1/TAU1
DPPLA2 = XDP3/TAU1

"Adaptive Gains:"
```

```
IE11 = INTEG((ERROR1**2)*TEN, KE11IC)
IE12 = INTEG((ERROR1*ERROR2)*TEN, KE12IC)
IE21 = INTEG((ERROR2*ERROR1)*TEN, KE21IC)
IE22 = INTEG((ERROR2**2)*TEN, KE22IC)
IX11 = INTEG((ERROR1*XMOD1)*TXN, KX11IC)
IX12 = INTEG((ERROR1*XMOD2)*TXN, KX12IC)
IX21 = INTEG((ERROR2*XMOD1)*TXN, KX21IC)
IX22 = INTEG((ERROR2*XMOD2)*TXN, KX22IC)
IU11 = INTEG((ERROR1*UR1)*TUN, KU11IC)
IU12 = INTEG((ERROR1*UR2)*TUN, KU12IC)
IU21 = INTEG((ERROR2*UR1)*TUN, KU21IC)
IU22 = INTEG((ERROR2*UR2)*TUN, KU22IC)

KE11 = ERROR1**2*TEB + IE11
KE12 = ERROR1*ERROR2*TEB + IE12
KE21 = ERROR2*ERROR1*TEB + IE21
KE22 = ERROR2**2*TEB + IE22
KX11 = ERROR1*XMOD1*TXB + IX11
KX12 = ERROR1*XMOD2*TXB + IX12
KX21 = ERROR2*XMOD1*TXB + IX21
KX22 = ERROR2*XMOD2*TXB + IX22
KU11 = ERROR1*UR1*TUB + IU11
KU12 = ERROR1*UR2*TUB + IU12
KU21 = ERROR2*UR1*TUB + IU21
KU22 = ERROR2*UR2*TUB + IU22

"Output of the Adaptive Gains:"

KEOUT1 = ERROR1*KE11 + ERROR2*KE12*DEC
KEOUT2 = ERROR1*KE21*DEC + ERROR2*KE22
KXOUT1 = XMOD1*KX11 + XMOD2*KX12*DEC
KXOUT2 = XMOD1*KX21*DEC + XMOD2*KX22
KUOUT1 = UR1*KU11 + UR2*KU12*DEC
KUOUT2 = UR1*KU21*DEC + UR2*KU22

"Now we can obtain the Input to the Plant:"

UPLA1 = KEOUT1 + KXOUT1 + KUOUT1
UPLA2 = KEOUT2 + KXOUT2 + KUOUT2

"The following lines change the load on the arm:"

PROCEDURAL
IF(T.LT.TDROP)M=MINI
IF(T.GE.TDROP)M=MNEW
END

"Calculation of the actual and augmented errors:"

EY1    = YMOD1 - Y1
EY2    = YMOD2 - Y2
ERROR1 = EY1 - DPPLA1
ERROR2 = EY2 - DPPLA2
```

```
end
        "Specify termination condition:"

        TERMT(T.GE.FIN)

END $ "of DYNAMIC"
END $ "of PROGRAM"
```

LITERATURE CITED

[1] BarKana, I., "Adaptive Control - A Simplified Approach." Advances in Control and Dynamic Systems, C.T. Leondes, Ed., Vol. 25, Academic Press, 1987.

[2] Sobel, K. M. and Kaufman, H., "Direct Model Reference Adaptive Control of a Class of MIMO Systems." Advances in Control and Dynamic Systems, C.T. Leondes, Ed., Vol. 24, Academic Press, 1986.

[3] Sobel, K. M., Kaufman, H., and Mabius, L., "Implicit Adaptive Control for a Class of MIMO Systems." IEEE TAES, Vol. AES-18, No. 5, 1982.

[4] Kaufman, H., and Neat, G. W., "Asymptotically Stable Direct Model Reference Adaptive Controllers for Processes Not Necessarily Satisfying a Positive Real Constraint." CIRSSE Report #66, Rensselaer Polytechnic Institute, October 1990.

[5] Neat, G. W., "Expert Adaptive Control: Method and Medical Application." PhD Thesis, Rensselaer Polytechnic Institute, 1990.

[6] BarKana, I. and Kaufman, H., "Robust Simplified Adaptive Control for a Class of Multivariable Continuous Systems." 24th IEEE Conference on Decision and Control, Ft. Lauderdale, Florida, 1985.

[7] BarKana, I. and Kaufman, H., "Global Stability and Performance of a Simplified Adaptive Control Algorithm." International Journal of Control, Vol. 42, No. 6, 1985.

[8] Steinvorth, R., Kaufman, H., and Neat, G. W., "Direct Model Reference Adaptive Control with Application to Flexible Robots." SPIE International Symposia, Boston, Mass., 1990.

[9] Kaufman, H., Neat, G., and Steinvorth, R., "Asymptotically Stable MIMO Direct Model Reference Adaptive Controller for Processes Not Necessarily Satisfying a Positive Real Constraint." European Control Conference, Grenoble, July 1991.

[10] Ghorbel, F., Hung, J., and Spong, M., "Adaptive Control of Flexible Joint Manipulators." IEEE Control Systems Magazine, pp 9-13, Dec 1989.

[11] Seraji, H., "Decentralized Adaptive Control of Manipulators: Theory, Simulation, and Experimentation." IEEE Transactions on Robotics and Automation, Vol. 5, No. 2, April 1989.

[12] Advanced Continuous Simulation Language Reference Manual. 4th ed. Mitchell and Gauthier Associates. Concord, Mass., 1986.