

THE UNIVERSITY OF CALGARY

Direct Neural-Adaptive Control of Robotic Manipulators

Using a Forward Dynamics Approach

by

Arash Beirami

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

September, 2006

© Arash Beirami 2006

THE UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, “**Direct Neural-Adaptive Control of Robotic Manipulators Using a Forward Dynamics Approach**” submitted by Arash Beirami in partial fulfillment of the requirements for the degree of Master of Science.



Supervisor, Dr. C. Macnab,
Department of Electrical and Computer Engineering.



Dr. O.P. Malik,
Department of Electrical and Computer Engineering.



Dr. E. Nowicki,
Department of Electrical and Computer Engineering.



Dr. G. Goldsmith,
Department of Mechanical and Manufacturing Engineering

Sept. 14 2006
Date

Abstract

This thesis investigates neural-adaptive control of a two-link robotic manipulator, based on estimating the forward dynamics of the system. Modeling the forward dynamics is achieved on-line using the Cerebellar Model Articulation Controller (CMAC) associative memory neural network. The CMAC takes advantage of any previous knowledge of the dynamics of the system while compensating for external disturbances. This method contrasts previous results in the literature that use an approximation of the inverse dynamics, and to the best of the author's knowledge, is a new scheme that uses forward dynamic approach to compensate for both the structured and unstructured uncertainties on-line. The weight updates take advantage of the neural network property that a nonlinear function can be uniformly estimated using different sets of weights, and guide the weights toward their real values using previous knowledge of the system.

In this thesis, three different approaches of forward-dynamic CMAC control are compared: state error-based weight update, supervised learning, and combined state error and supervisory learning. All the approaches are shown to be Lyapunov-stable (Ultimately Uniformly Bounded). Simulation experiments are conducted with a two-link rigid planar manipulator arm that tries to follow a desired trajectory and adapt to payload changes. The conclusion is that training based on Lyapunov-stable weight updates combined with supervised learning using a model estimate of the inertia matrix is sufficient to achieve neural-adaptive control using a forward dynamics approach.

Acknowledgment

The author wishes to express his appreciation to Dr. Chris Macnab for his invaluable guidance during the course of this work and for his advice and help during the preparation of this manuscript.

Table of Contents

Approval Page	ii
Abstract	iii
Acknowledgment	iv
Table of Contents	v
List of Tables	vii
List of Figures	viii
Symbols	x
1 Introduction	1
1.1 Motivation	1
1.2 Literature Review	2
1.3 Thesis Statement and Objective / Methods of Investigation	5
1.4 Thesis Outline	5
2 Methodology	7
2.1 Forward versus Inverse Dynamics	7
2.2 Computed Torque Scheme	9
2.3 Computed Torque Controller with Adaptation Algorithm	10
2.4 Implementing Neural Networks into the Computed Torque Controller	12
2.5 Proposed Method	14
3 Background on CMAC Neural Network [1]	18
3.1 The $S \rightarrow M$ Mapping	19
3.2 The $M \rightarrow A$ Mapping	22
3.3 The $A \rightarrow P$ Mapping	23
3.4 Data Storage in CMAC	24
3.5 Memory Size Requirements in CMAC	24
3.6 Generalization in the CMAC Memory	25

4	Simulation	27
4.1	Dynamics	27
4.2	Objective	29
4.3	Simulation Parameters	30
4.3.1	Investigating the Weight Update Law	30
4.3.2	Proposed scheme versus Computed Torque	31
5	Examining Different Forward Dynamic Approaches	32
5.1	State-Error Based Weight Updates (Method 1)	33
5.1.1	Tuning the Weight Update Parameters	34
5.1.2	Performance	35
5.2	Supervised learning (Method 2)	36
5.2.1	Tuning the Weight Update Parameters	37
5.2.2	Performance	39
5.3	Supervisory inertia matrix (Method 3)	40
5.3.1	Tuning the Weight Update Parameters	41
5.3.2	Performance	42
5.4	Comparing Performances	43
6	Comparison / Robustness of the Proposed Method	46
6.1	Comparison between the Proposed Method and the Computed Torque Scheme	47
6.2	Robustness of proposed method	49
7	Stability Proof	51
7.1	Function Approximation Using CMAC Neural Networks	51
7.2	System Stability Proof	54
8	Conclusions	61
8.1	Results	61
8.2	Application and Limitations	63
	List of References	64

List of Tables

1.1	Classifying the reference papers	3
2.1	Three different approaches of the forward dynamic method	16
4.1	Investigating the proposed weight update	30
4.2	Physical parameters	31

List of Figures

2.1	Computed torque controller with adaptation algorithm	11
2.2	Computed torque controller with neural network estimation	12
2.3	Neural network off-line training	13
2.4	Proposed method	14
3.1	Model of Cerebellum	18
3.2	Schematic figure of CMAC	20
3.3	Typical Response of mossy fibers	21
3.4	A two-input CMAC with four quantization functions	22
3.5	Memory distribution for two adjacent points in the input space	26
4.1	Trajectory to be followed	27
4.2	Desired joint angle	29
4.3	Desired joint angular velocity	29
5.1	Comparing method 1 CMAC estimates with their actual values for the 1 st and 50 th iteration	34
5.2	RMS error of the angles and angular velocities for the Method 1	35
5.3	Comparing method 2 values of the state error term with the learning term in the weight update	38
5.4	Comparing method 2 CMAC estimates with their actual values for the 1 st and 50 th iteration	39
5.5	RMS error of the angles and angular velocities for the Method 2	40
5.6	Comparing CMAC method 3 estimates with their actual values for the 1 st and 50 th iterations	42
5.7	RMS error of the angles and angular velocities for the Method 3	43
5.8	Comparing the performance of the three methods	43
5.9	Comparing the amount of torque required for joint 1 to achieve the perfor- mance illustrated in 5.8	44
5.10	Comparing the amount of torque required for joint 2 to achieve the perfor- mance illustrated in 5.8	44
6.1	Comparison between the RMS error of the angles and angular velocities of the proposed method with a computed torque controller	46
6.2	Comparing the position error of the proposed method with a computed torque controller after convergence	47
6.3	Comparing the required RMS torque for the proposed and the computed torque scheme	48

6.4	Norm of the error between the terms in the model estimate and real model of the inertia matrix for different mass uncertainties / payloads	49
6.5	position error of the end-effector for different payloads/mass uncertainties	50

Symbols

\forall	for all
\in	belongs to
\subset	subset of
\rightarrow	tends to
\mathfrak{R}	the set of real numbers
\mathfrak{R}_+	the set of non-negative real numbers
\mathfrak{R}^n	the set of all n-dimensional real vectors
$\mathfrak{R}^{m \times n}$	the set of all $n \times m$ -dimensional real matrices
$ a $	the absolute value of a scalar a
a_{ij}	the ij -th element of \mathbf{A}
boldface	vectors, matrices or operators of multi-input and multi-output system
$\ \mathbf{x}\ $	the norm of the vector \mathbf{x}
\mathbf{A}^T	the transpose of \mathbf{A}
\mathbf{A}^{-1}	the inverse of \mathbf{A}
$\hat{\mathbf{A}}$	the estimate of \mathbf{A}
$\tilde{\mathbf{A}}$	$\mathbf{A} - \hat{\mathbf{A}}$
$\lambda_{min}(\mathbf{A})$	the minimum eigenvalue of \mathbf{A}
$\lambda_{max}(\mathbf{A})$	the maximum eigenvalue of \mathbf{A}
$\mathbf{I}_{m \times n}$	a m by n identity matrix
$\mathbf{0}_{m \times n}$	a m by n zero matrix
$\mathbf{P} > 0$	a positive definite matrix \mathbf{P}
$\mathbf{P} \geq 0$	a positive semidefinite matrix \mathbf{P}
$f : S_1 \rightarrow S_2$	a function of f mapping a set S_1 into a set S_2

Chapter 1

Introduction

Robotic manipulators are highly nonlinear systems that are commonly used in tasks such as material handling and accurate positioning. For a manipulator to perform these tasks, the manipulator's end-effector is required to move from one point in space to another, and follow a desired trajectory with a desired speed and acceleration. Therefore, trajectory tracking control of a robot is of a great importance.

1.1 Motivation

The computed torque is a model-based scheme applied heavily in the field of robotic control. The basis of this scheme is feedback linearization which is used on nonlinear systems. It is an effective scheme when there are no structured and unstructured uncertainties. Structured uncertainties are caused by imprecision in the manipulator link properties or unknown loads, whereas unstructured uncertainties refer to unmodeled dynamics such as nonlinear friction and disturbances. In practice, precise modeling of a system's dynamics cannot be easily achieved. In addition, the system's dynamics might vary due to a changing payload. Therefore, practical application of computed torque scheme is commonly seen with further modification of this scheme [2], [3], [4], [5], [6], [7].

Previous works have used adaptive control scheme or neural networks to improve the computed control scheme. In the adaptive control scheme [2], joint acceleration and bounds on the inverse of the inertia matrix are used to adaptively implement the computed

torque scheme. Although this scheme compensates for structured uncertainties, it is not robust to unstructured uncertainties. The method in [7] introduces a nonlinear compensator that incorporates the idea of computed torque. This method uses neural networks to compensate for both structured and unstructured uncertainties. The neural networks are trained off-line based on the dynamic model of the system. This means that using repetitive learning, the neural networks' weights are updated to produce the dynamic model of the system before they are used in the control scheme. After applying the neural networks to the system, their weights are updated based only on the state error.

To the best of author's knowledge, no work has previously been proposed that uses neural networks to cancel the nonlinearities based on on-line learning of the system's forward dynamics. The main characteristic of on-line learning is that the neural networks, have no previous knowledge of the system when they are applied to the system. Using repetitive learning, the neural networks learn the dynamics, and compensate for structured and unstructured uncertainties while controlling the system. The main advantage of on-line training is that if the system dynamics change, the controller is able to adapt to the changes. Such a controller is desired and proposed in this thesis.

1.2 Literature Review

The following works comprise the literature sources of this thesis. Each work presents alternatives to different sections of the proposed scheme. The relevance of the works to this thesis is illustrated in Table 1.1.

Ref.	Direct Adaptive	MLP	Fuzzy	CMAC	RBF
[1]				*	
[8]	*				
[9]	*	*			
[10]	*	*	*		
[11]		*	*	*	
[12]	*				*
[13]	*				
[14]	*			*	
[15]	*			*	
[16]				*	*
[17]				*	
[18]			*	*	
[19]				*	
[20]	*		*	*	
[21]	*				
[2]	*				
[22]	*				
[3]		*			
[4]					*
[5]	*				
[6]		*			
[7]		*			
[23]	*	*			
[24]	*		*		
[25]	*	*			
[26]	*	*			*
[27]	*	*			*
[28]	*			*	
[29]	*	*			
[30]	*	*			
[31]	*				

Table 1.1: Classifying the reference papers

In what follows, a brief description of the above terms are given:

- Direct Adaptive: The objective of this scheme is to find a control law that minimizes a predefined error that quantifies the performance of the closed loop system. The parameters of the control are adjusted according to the tracking or modeling error of the system. $\lim_{\sigma \rightarrow 0^+} \dot{\sigma} < 0$ and $\lim_{\sigma \rightarrow 0^-} \dot{\sigma} > 0$
- MLP (Multi-Layer Perceptron): It is a feed-forward neural network where the input

signals propagate through several layers before the output is calculated. It is a fully connected network meaning that all the nodes from one layer are connected to each node of the next layer. The nodes contain an activation function that calculates the node's output based on the weighted input.

- **RBF (Radial Basis Functions)** : It is a three layered feed-forward network consisting of input, hidden and output layers. Each hidden node contains a parameter called center. The Euclidean distance between the input vector and the center of the node is passed through a nonlinear function. The output of the network is the weighted sum of the outputs of the hidden layers.
- **CMAC (Cerebellar Model Articulation Controller)**: It is an associative memory neural network that consists of many layers. Each layer contains many cells, and there is a quantization function associated with each cell. For a binary CMAC, the value of the basis functions are either 1 or 0 depending on if the input falls in the cell. A detailed description of this network is presented in chapter 3.
- **Fuzzy Control**: This controller is based on modeling a system by implementing control laws in the form of IF-THEN statements. Fuzzy sets allow for partial membership where the degree of membership can vary from 0 to 1. Fuzzy control is mainly used when it is not feasible to obtain an accurate mathematical model of a system.

1.3 Thesis Statement and Objective / Methods of Investigation

Computed torque is a model-based scheme that requires a precise dynamic model of the system to perform. This limits the practical implementation of this scheme. To take advantage of previous knowledge of a system, a new scheme, to the best of author's knowledge, is proposed where a forward dynamics approach is used for on-line training of CMAC neural networks. This scheme results in a stable system that is robust to structured and unstructured uncertainties. The objectives of the work within this thesis are:

- Introducing a forward dynamics approach for on-line training of neural networks
- Proposing an on-line stable weight update that both learns the system dynamics and results in a robust system
- comparing the proposed scheme with the model-based computed torque to illustrate the advantages and disadvantages of the proposed scheme

1.4 Thesis Outline

The remaining chapters of this thesis are organized as follow:

- **Chapter 2:** A comparison between the forward and inverse dynamics approach is presented. Then, a description of a model-based computed torque controller is given. The constraints of the conventional computed torque controller are explained along with possible solutions. Two methods that propose different solutions, including an adaptation algorithm and neural networks are described in detail. Then, a description of the architecture of the proposed method and two other approaches that led to the novel method are presented.

- **Chapter 3:** This chapter presents a detailed description on a Cerebellar Model Articulation Controller (CMAC) neural network.
- **Chapter 4:** A description of a two-link planar robot which is used for comparison between the forward dynamic approaches is presented. Also, tracking objective and the desired trajectories are presented for the comparison.
- **Chapter 5:** Three on-line forward dynamics approaches are compared. The tuning of the weight update parameters for each case is explained and confirmed with the result of the simulations. Each approach is presented to eliminate the deficiencies of the previous one, and the last approach is the novel method that is used for further comparison with the conventional computed torque.
- **Chapter 6:** This chapter consists of the comparisons performed between the proposed and the computed torque controller. The comparisons can be categorized as:
 1. The ideal case where the dynamic model of the system is precisely known
 2. The case where the dynamic model is inaccurate or equivalently the robotic manipulator has to adapt to changing payload
- **Chapter 7:** This chapter covers concepts and theories relevant to function approximation of CMAC neural networks. A Lyapunov stability proof is then presented which shows that the system is Uniformly Ultimately Bounded. The weight update and control law are chosen according to the stability proof.
- **Chapter 8:** In this chapter, the advantages and disadvantages of the forward dynamics approaches, and the comparison between the novel approach and computed torque scheme are emphasized.

Chapter 2

Methodology

A comparison between the forward and inverse dynamics scheme is presented followed by a description of the fundamental properties of the system. Next, a description of the computed torque scheme is given along with its limitations. Two previous works that presented solution to these limitations are described. The chapter is concluded with a description of the method proposed in this thesis.

2.1 Forward versus Inverse Dynamics

The dynamics of a planar (horizontal) n -link robotic manipulator can be expressed as in [8]:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \boldsymbol{\tau} \quad (2.1.0.1)$$

where $\boldsymbol{\tau} \in \mathbb{R}^n$ contains the control torques, \mathbf{q} , $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}} \in \mathbb{R}^n$ are vectors of link angles, angular velocities, and angular accelerations respectively. $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n \times n}$ is the inertia matrix, and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n \times n}$ is the Coriolis and centripetal matrix.

Given the desired trajectories \mathbf{q}_d and $\dot{\mathbf{q}}_d$, the tracking errors can be written as:

$$\mathbf{e}_1 = \mathbf{q} - \mathbf{q}_d \quad (2.1.0.2)$$

$$\mathbf{e}_2 = \dot{\mathbf{q}} - \dot{\mathbf{q}}_d \quad (2.1.0.3)$$

Taking the derivative of the tracking errors and substituting $\ddot{\mathbf{q}}$ from Eq. (2.1.0.1), the error dynamics can be expressed as:

$$\dot{\mathbf{e}}_1 = \mathbf{e}_2 \quad (2.1.0.4)$$

$$\dot{\mathbf{e}}_2 = -\mathbf{M}^{-1}(\mathbf{q})\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{M}^{-1}(\mathbf{q})\boldsymbol{\tau} - \ddot{\mathbf{q}}_d \quad (2.1.0.5)$$

In inverse-dynamics approach, given the trajectories \mathbf{q} and $\dot{\mathbf{q}}$, the control torque is determined. Therefore, the control law is chosen as:

$$\boldsymbol{\tau} = \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_d - \mathbf{G}_1\mathbf{e}_1 - \mathbf{G}_2\mathbf{e}_2 \quad (2.1.0.6)$$

where $\mathbf{G}_1, \mathbf{G}_2 \in \mathfrak{R}^{n \times n}$ are positive definite constant gain matrices. When CMAC neural networks are used to estimate the nonlinear terms, the inverse-dynamics control law becomes:

$$\boldsymbol{\tau} = -\mathbf{C}_I(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_d) - \mathbf{G}_1\mathbf{e}_1 - \mathbf{G}_2\mathbf{e}_2 \quad (2.1.0.7)$$

where $\mathbf{C}_I \in \mathfrak{R}^n$ is the CMAC model of nonlinearities.

Contrary to inverse-dynamics, in a forward-dynamics approach the trajectories \mathbf{q} and $\dot{\mathbf{q}}$ are determined based on the given torques. Therefore, model estimates of $\mathbf{M}(\mathbf{q})$ and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ are required. Choosing the control law as:

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})(\mathbf{M}^{-1}(\mathbf{q})\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \ddot{\mathbf{q}}_d - \mathbf{G}_1\mathbf{e}_1 - \mathbf{G}_2\mathbf{e}_2) \quad (2.1.0.8)$$

allows for the estimation of $\mathbf{M}(\mathbf{q})$ and $\mathbf{M}^{-1}(\mathbf{q})\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$. Using two CMAC neural networks to estimate the nonlinear terms, the control law can be rewritten as:

$$\boldsymbol{\tau} = \mathbf{C}_M^{-1}(\mathbf{q})(-\mathbf{C}_F(\mathbf{q}, \dot{\mathbf{q}}) + \ddot{\mathbf{q}}_d - \mathbf{G}_1\mathbf{e}_1 - \mathbf{G}_2\mathbf{e}_2) \quad (2.1.0.9)$$

where $\mathbf{C}_M, \mathbf{C}_F \in \mathfrak{R}^{n \times n}$ are ideal CMAC models of \mathbf{M}^{-1} and $-\mathbf{M}^{-1}\mathbf{F}\dot{\mathbf{q}}$ respectively.

When designing a controller that uses forward dynamics approach, the fundamental properties associated with the dynamics of the robot need to be satisfied. Some of these properties are described below.

- **Property 1:** The inertia matrix $\mathbf{M}(\mathbf{q})$ is a symmetric positive definite matrix. This matrix and its inverse are uniformly bounded as a function of $\mathbf{q} \in \mathfrak{R}^n$
- **Property 2:** The dynamic equation 2.1.0.1 is linear in the unknown parameters. This property may be expressed as:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\Theta = \tau$$

where $\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \in \mathfrak{R}^{n \times r}$ is a matrix of known functions, known as regressor, and $\Theta \in \mathfrak{R}^r$ is a vector of parameters.

- **Property 3:** The matrix $\mathbf{N}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{M}}(\mathbf{q}) - 2\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ is skew-symmetric, i.e. $\mathbf{N}^T(\mathbf{q}, \dot{\mathbf{q}}) = -\mathbf{N}(\mathbf{q}, \dot{\mathbf{q}})$

2.2 Computed Torque Scheme

One of the well-known robotic control schemes is the model-based computed torque control [2], [3], [4], [5], [6]. This scheme does not exploit the skew-symmetry, but relies on exact cancellation of nonlinearities in the system. It uses the dynamic model of the robot to produce the torque required for a desired trajectory, and is proven to be effective if the structured and unstructured uncertainties can be ignored. In an ideal case, this scheme results in a linear closed loop system, and for a system defined with the dynamic equation 2.1.0.1, the control law is chosen as:

$$\tau = \hat{\mathbf{M}}(\mathbf{q})(\ddot{\mathbf{q}}_d - \mathbf{K}_v(\dot{\mathbf{q}} - \dot{\mathbf{q}}_d) - \mathbf{K}_p(\mathbf{q} - \mathbf{q}_d)) + \hat{\mathbf{F}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \quad (2.2.0.10)$$

where \mathbf{q}_d , $\dot{\mathbf{q}}_d$, and $\ddot{\mathbf{q}}_d$ are, respectively, the desired position, desired velocity, and desired acceleration, \mathbf{K}_v , $\mathbf{K}_p \in \mathcal{R}^{n \times n}$ are positive definite gain matrices, and $\hat{\mathbf{M}}(\mathbf{q})$ and $\hat{\mathbf{F}}(\mathbf{q}, \dot{\mathbf{q}})$ are estimates of $\mathbf{M}(\mathbf{q})$ and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$.

In an ideal case where there are no uncertainties, the control law leads to the closed loop system expressed as:

$$\ddot{\mathbf{e}} + \mathbf{K}_v \dot{\mathbf{e}} + \mathbf{K}_p \mathbf{e} = 0 \quad (2.2.0.11)$$

where \mathbf{e} is a vector of systems's tracking errors. However, uncertainties in the robot dynamic model are always present and inevitable in practice. The uncertainties will lead to degradation of robot performance and could cause instability. To ensure stability and improved performance, compensating controllers are required.

In [2], joint acceleration and bounds on the inverse of the inertia matrix are used to adaptively implement the computed torque scheme. This scheme compensates for structured uncertainties, but is not robust to the unstructured ones. In [7], a nonlinear compensator is presented that incorporates the idea of computed torque. This method uses neural networks to compensate for all the uncertainties. A brief description of the mentioned schemes are presented below.

2.3 Computed Torque Controller with Adaptation Algorithm

This method, presented in [2], proposes an adaptation algorithm for computed torque controllers to cancel the uncertainties in the model estimates. The idea is to take advantage of the previous knowledge of the system and produce an acceptable performance where the dynamic model is not precise. Figure 2.1 illustrates a block diagram of this globally stable method. In practice, the dynamic model of the manipulator is not exact, and Eq. 2.2.0.11

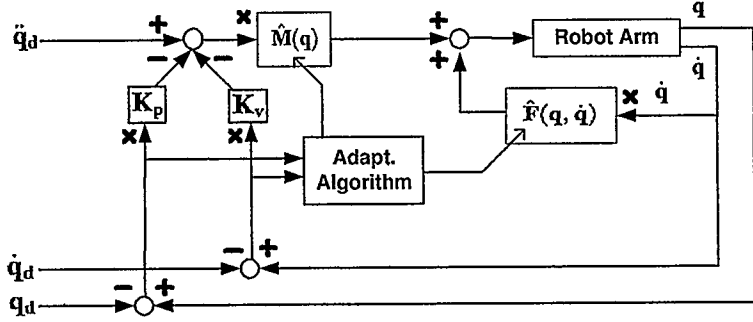


Figure 2.1: Computed torque controller with adaptation algorithm

is presented as:

$$\ddot{e} + \mathbf{K}_v \dot{e} + \mathbf{K}_p e = -\hat{\mathbf{M}}^{-1}(\mathbf{q})[\tilde{\mathbf{M}}(\mathbf{q})\ddot{\mathbf{q}} + \tilde{\mathbf{F}}(\mathbf{q}, \dot{\mathbf{q}})] \quad (2.3.0.12)$$

where $\tilde{\mathbf{M}}(\mathbf{q}) = \mathbf{M}(\mathbf{q}) - \hat{\mathbf{M}}(\mathbf{q})$ and $\tilde{\mathbf{F}}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) - \hat{\mathbf{F}}(\mathbf{q}, \dot{\mathbf{q}})$ represent the errors in the dynamic modeling. Since Eq. 2.1.0.1 is linear in parameters, the error equation 2.3.0.12 can be rewritten as:

$$\ddot{e} + \mathbf{K}_v \dot{e} + \mathbf{K}_p e = -\hat{\mathbf{M}}^{-1}(\mathbf{q})\mathbf{W}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\Theta \quad (2.3.0.13)$$

where $\Theta \in \mathcal{R}^r$ is a vector containing the parameter errors, and $\mathbf{W}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \in \mathcal{R}^{n \times r}$ is a matrix of known functions. For this method, it is assumed that $\ddot{\mathbf{q}}$ is measurable and the update law is modified to ensure that $\hat{\mathbf{M}}^{-1}$ is bounded. Under these assumptions, and using the adaptation law

$$\dot{\Theta} = \Gamma \mathbf{W}^T \hat{\mathbf{M}}^{-1} \mathbf{z}$$

where $\mathbf{z} = \mathbf{e} + \alpha \dot{e}$ and α is a positive constant, the system is proven in [2] to be Lyapunov globally stable.

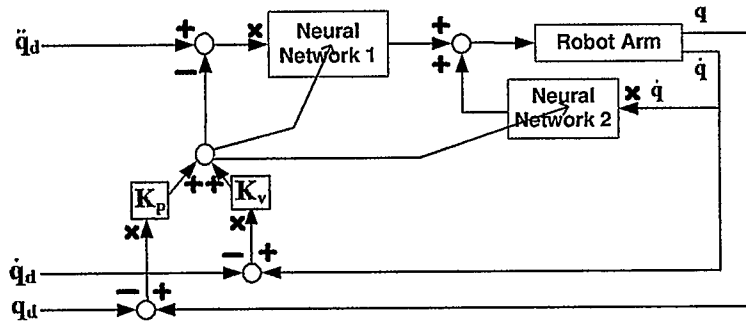


Figure 2.2: Computed torque controller with neural network estimation

2.4 Implementing Neural Networks into the Computed Torque Controller

This method, presented in [7], uses neural networks to both learn the dynamic model of the system, and cancel unstructured uncertainties. The neural networks are used on-line in a feed-forward loop and use state error feedback for training. They do not learn the inverse dynamics, but compensate for the nonlinearities of the robotic manipulator. This method implements the idea of computed torque as shown in Figure 2.2. The neural networks are used to produce estimates of $\mathbf{M}(\mathbf{q})$ and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ and are trained off-line before they are applied to the system. The off-line learning procedure is illustrated in the block diagram 2.3. The desired trajectories are used as inputs to the neural networks, $\mathbf{M}(\mathbf{q})$, and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$. The outputs of $\mathbf{M}(\mathbf{q})$ and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ are then compared with their corresponding neural network and the errors are fed back to train the neural network weights. This procedure is repeated until the error decreases to an acceptable level. After the networks have been trained, they are applied to the system as part of a hybrid control that implements the computed torque scheme. For the simulation experiments, three-layered neural networks

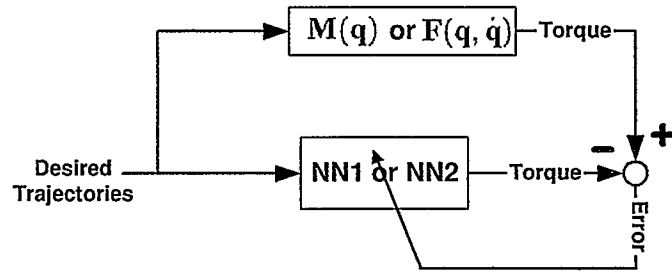


Figure 2.3: Neural network off-line training

are used to learn the dynamics of a two-link planar robot. The inertia matrix $\mathbf{M}(\mathbf{q})$ and the Coriolis/centripetal forces $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ of a two-link planar robot can be described as:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} p_1 + p_2 + 2p_3 \cos(q_2) & p_2 + p_3 \cos(q_2) \\ p_2 + p_3 \cos(q_2) & p_2 \end{bmatrix} \quad (2.4.0.14)$$

$$\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} h\dot{q}_2 & h\dot{q}_1 + h\dot{q}_2 \\ -h\dot{q}_1 & 0 \end{bmatrix} \quad (2.4.0.15)$$

$$\text{for:} \quad \begin{aligned} p_1 &= m_1 L_1^2 + m_2 L_1^2 & p_2 &= m_2 L_2^2 \\ p_3 &= m_2 L_1 L_2 & h &= -m_2 L_1 L_2 \sin(q_2) \end{aligned}$$

where m_i and L_i are the mass and length of link i respectively.

The terms representing the matrix $\mathbf{M}(\mathbf{q})$ change based on the value of $\cos(q_2)$. Therefore, the desired trajectory chosen for off-line training of the neural network 1 is $\cos(q_2)$. The values of the terms representing the matrix $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ depend on \dot{q}_1 , \dot{q}_2 , and q_2 , and the desired trajectory input in Figure 2.3 includes these terms for off-line training of neural network 2.

Through simulation, it is illustrated in [7], that without any off-line training, the neural networks take numerous iterations to learn and produce acceptable performance. Also, the resulting system does not adapt itself easily to changes in the desired trajectories.

The method presented in [7] uses neural network in a forward dynamics fashion. However not many, if any, papers have been published based on this approach since it was published in 1991. The objective of this thesis is to finally make this forward dynamics approach work with the advantage of having no off-line training, better performance, and guaranteed stability.

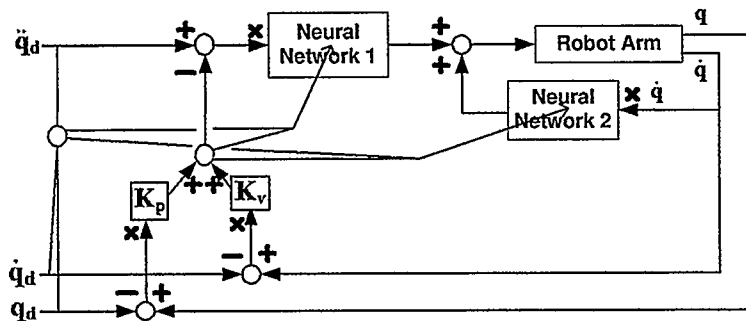


Figure 2.4: Proposed method

2.5 Proposed Method

The scheme proposed in this work uses two neural networks in a forward dynamics fashion to compensate for the structured and unstructured uncertainties of the system. The networks used in this scheme are CMAC neural networks, and are described in detail in chapter 3. The overall picture 2.4 is similar to that of the work presented in [7], with two major differences:

- There is no off-line training, meaning that the neural networks have zero knowledge of the system when they are applied to the system
- The desired trajectories \mathbf{q}_d , $\dot{\mathbf{q}}_d$, and $\ddot{\mathbf{q}}_d$ are also used to train the neural network weights on-line

The control law of the proposed scheme is similar to that of the computed torque. Recall the control equation for the computed torque controller:

$$\boldsymbol{\tau} = \hat{\mathbf{M}}(\mathbf{q})(\ddot{\mathbf{q}}_d - \mathbf{K}_v(\dot{\mathbf{q}} - \dot{\mathbf{q}}_d) - \mathbf{K}_p(\mathbf{q} - \mathbf{q}_d)) + \hat{\mathbf{F}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \quad (2.5.0.16)$$

In order to use CMAC neural network in a forward dynamics fashion, and be able to prove stability, the computed torque equation is rewritten as:

$$\boldsymbol{\tau} = \hat{\mathbf{M}}(\mathbf{q})(\ddot{\mathbf{q}}_d - \mathbf{K}_v(\dot{\mathbf{q}} - \dot{\mathbf{q}}_d) - \mathbf{K}_p(\mathbf{q} - \mathbf{q}_d) + \hat{\mathbf{M}}^{-1}(\mathbf{q})\hat{\mathbf{F}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{u}_{ND}) \quad (2.5.0.17)$$

where the only added term \mathbf{u}_{ND} (Nonlinear Damping control) is to compensate for CMAC's modeling error. Let \mathbf{C}_M and \mathbf{C}_F be CMACs' models of \mathbf{M}^{-1} and $-\mathbf{M}^{-1}\mathbf{F}\dot{\mathbf{q}}$ respectively. Also, let $(\tilde{\cdot})$ and $(\hat{\cdot})$ indicate the error and estimate matrix respectively, then Eq. 2.5.0.17 can be rewritten as:

$$\boldsymbol{\tau} = \hat{\mathbf{C}}_M^{-1}(\mathbf{q}) \left(\ddot{\mathbf{q}}_d - \mathbf{K}_v(\dot{\mathbf{q}} - \dot{\mathbf{q}}_d) - \mathbf{K}_p(\mathbf{q} - \mathbf{q}_d) + \mathbf{u}_{ND} - \hat{\mathbf{C}}_F(\mathbf{q}, \dot{\mathbf{q}}) \right) \quad (2.5.0.18)$$

Based on the Lyapunov stability proof (chapter 7), the CMAC weight update law is chosen as:

$$\dot{\hat{\mathbf{w}}}_i = \beta_i(\eta_i - v_i\hat{\mathbf{w}}_i + \gamma_i\phi(g_i - \phi^T\hat{\mathbf{w}}_i)) \quad (2.5.0.19)$$

$$\phi = \phi_M \quad i = 1, 2, 3 \quad \text{and} \quad \phi = \phi_F \quad i = 4, 5$$

where:

$$\eta_i : \text{state error dependent term}$$

$$v\hat{\mathbf{w}}_i : \text{leakage term}$$

$$\gamma_i\phi(g_i - \phi^T\hat{\mathbf{w}}_i) : \text{learning term}$$

The above weight update law results in a Uniformly Ultimately Bounded (UUB) system. The two deciding terms in this weight update are the state error dependent term (η_i) and the learning term ($\gamma_i\phi(g_i - \phi^T\hat{\mathbf{w}}_i)$). Depending on how these two terms are tuned, the weight updates could range from completely learning the dynamic model of the system to completely ignoring the real model and only focusing on canceling the unstructured uncertainties. It should be noted that three different models are used in this thesis:

1. Real Model: contains the actual dynamics of the system and is used only for the purpose of simulation. The proposed controller has no knowledge of this model.
2. Model Estimate: is the best guess based on previous knowledge of the system. It is used to train CMAC neural networks.
3. CMAC Model: is CMAC's estimate of the Real Model of the system.

To determine the most suitable CMAC models \mathbf{C}_F and \mathbf{C}_M , different sets of weight update parameters are investigated. These set of parameters force the CMAC neural network to be effectively trained based on the state error or the model estimates of the system. The

	\mathbf{C}_M	\mathbf{C}_F
Method 1	State Error	State Error
Method 2	Model Estimate	Model Estimate
Method 3	Model Estimate	State Error

Table 2.1: Three different approaches of the forward dynamic method

basis of training of each CMAC model is tabulated in Table 2.1. The approaches shown in Table 2.1 are investigated in order where each approach is based on an improvement over the previous one. The last approach, where a model estimate is used for training C_M , and the state error for training C_F , performs the best.

Chapter 3

Background on CMAC Neural Network [1]

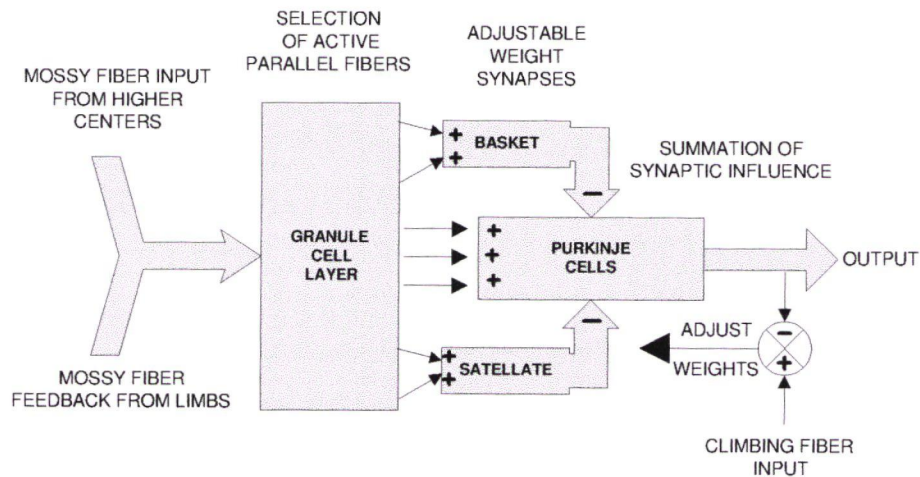


Figure 3.1: Model of Cerebellum

The part of the brain that is closely involved with the control of rapid, precise, and coordinated movement of hands, limbs and eyes is called cerebellum. The main input to the cerebellar cortex is a set of *mossy fibers*, which carry information from two main sources including commands from higher levels, and those carrying feedback information from the limbs. The mossy fibers make excitatory (+) contact with several granule cells. Golgi

cells sample the response of the granule cells and suppress all but the most highly excited granule cell using an inhibitory (-) connection. As a result, every input is transformed, by the granule layer, into a small subset of active parallel fibers. Purkinje cells, the output of the cerebellar cortex, sum the excitatory (+) effect of these parallel fibers through weighted connections. They also receive inhibitory (-) input from *basket*, and *stellate* cell inverters which use to sum along with the positive weights from parallel fibers.

A second set of fibers entering the cerebellar cortex are the *climbing fibers*. They are believed to adjust the strength of the weighted synaptic connections with the parallel fibers so as to train the cerebellum. The overall function is illustrated in Figure 3.1, which was developed independently in Great Britain by David Marr and in the United States by James S. Albus. Further work by James S. Albus has produced a mathematical formalism called the Cerebellar Model Arithmetic Computer (CMAC). The schematic of this model is presented in Figure 3.2. CMAC is defined by series of mapping

$$S \rightarrow M \rightarrow A \rightarrow P$$

where S is an input vector

M is the set of mossy fiber used to encode S

A is the set of granule cells contacted by M

and P is the output

In what follows the mappings are explained in details.

3.1 The $S \rightarrow M$ Mapping

This mapping transmits the vector component of S from the various points of origin to their destination in the cerebellar granular layer. Since mossy fibers are unreliable and are

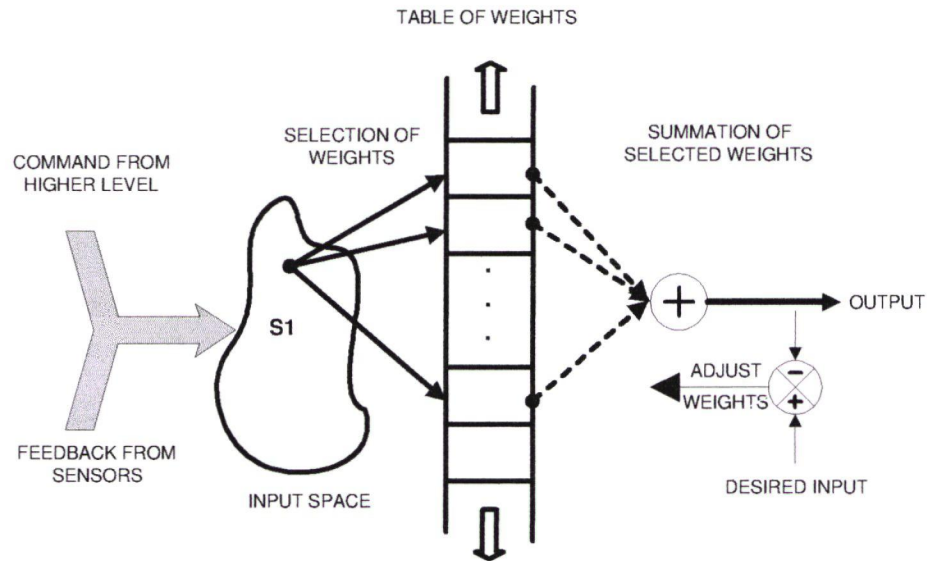


Figure 3.2: Schematic figure of CMAC

imprecise information channels with limited dynamic ranges, the brain encodes each high precision variable so that it can be carried on a large number of low precision channels. As a result, any particular mossy fiber is maximally active over some limited ranges of the variable. For example figure 3.3 illustrates the firing rate of 2 mossy fibers carrying information from an elbow joint. The mossy fiber labeled (a) is maximally active when the elbow position is between 80° to 100° .

CMAC models this encoding in a similar fashion. Define m_i^* to be a set of maximally active mossy fibers assigned to transmit the variable s_i . Mapping $s_i \rightarrow m_i^*$ is defined by a set of K quantized functions, each of which is offset by $\frac{1}{K}$ units of the quantizing interval.

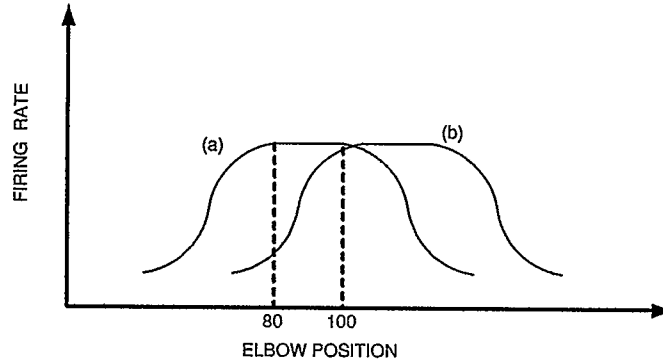


Figure 3.3: Typical Response of mossy fibers

The quantization function is denoted as ${}^n C_k$ where n is the input number. Figure 3.4 shows quantization of a 2-input space where $K = 4$. The quantization functions for input 1 and 2 are:

$${}^1 C_1 = \{A, B, C, D\}$$

$${}^1 C_2 = \{E, F, G, H\}$$

$${}^1 C_3 = \{I, J, K, L\}$$

$${}^1 C_4 = \{M, N, O, P\}$$

$${}^2 C_1 = \{a, b, c, d\}$$

$${}^2 C_2 = \{e, f, g, h\}$$

$${}^2 C_3 = \{i, j, k, l\}$$

$${}^2 C_4 = \{m, n, o, p\}$$

For every value of s_i , there exist a unique set m_i^* consisting of the set of values produced by Q quantizing functions. For instance, in Figure 3.4 the value of $s_1 = 7$ maps into the set $m_1^* = \{B, G, K, N\}$, and $s_2 = 10$ into the set $m_2^* = \{a, f, j, n\}$

One of the main advantages of the above encoding scheme is that a precise variable can be transmitted reliably over multiple imprecise information channels. The resolution of the transmitted variable depends on the number of channels. The more quantizing functions

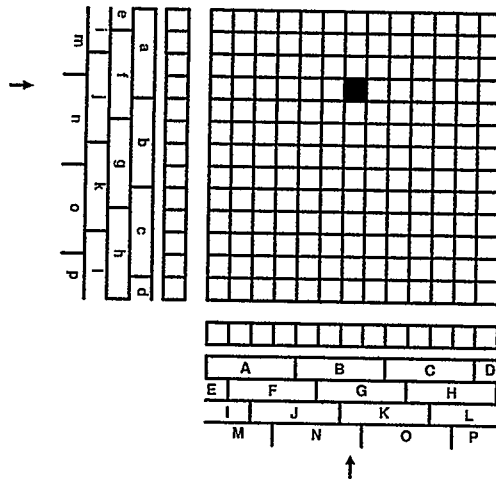


Figure 3.4: A two-input CMAC with four quantization functions

used to map a particular variable, the greater the precision with which it is represented. Another equally important advantage is that small changes to the input variable s_i have no effect on most of the elements in m_i^* . This property is called generalization. In CMAC there is a trade off between the generalization and resolution. The greater the resolution, the smaller the generalization and vice versa.

3.2 The $M \rightarrow A$ Mapping

The main purpose of this mapping is to identify active granule cells using the maximally active mossy fibers. Each granule cell receives information from specific combination of mossy fibers. This means that a unique address can be assigned to a granule cell by simply listing the mossy fibers that are maximally active.

In CMAC, each granule cell, represented by the mossy fibers, is given a unique address. The granule cell will be active only if all its inputs are active. Defining A to be the set of active granule cells, it can be uniquely identified using the address assigned to each granule cell. For example, let Cd be the address of the granule cell represented by C and d in Figure 3.4. Then, the set of active granule cell (A), that represents the vector $(s_1, s_2) = (7, 10)$, is identified as $A = \{Ba, Gf, Kj, Nn\}$. The level in which each granule cell is activated is determined by its quantization function. The closer the input is to the center of the activate cell, the larger the activation value, with one being the highest level.

3.3 The $A \rightarrow P$ Mapping

Let $W \in \mathfrak{R}^{m \times n}$ be a matrix of CMAC weights where m and n denote the number of active granule functions and number of outputs respectively. Also, let $\Gamma = [\gamma_1 \gamma_2 \dots \gamma_m]^T$ be the activation level of the activate set, then the output can be presented with the equation:

$$P = \Gamma^T W$$

For a binary CMAC, Γ is a vector of ones, and the output is simply the sum of the weights of the active granule cells. Using a binary CMAC, and assuming

$$W_{Ba} = 0.7$$

$$W_{Gf} = 1.5$$

$$W_{Kj} = 0.3$$

$$W_{Nn} = 0.5$$

the input vector $S = (7, 10)$ in Figure 3.4 produces the output $H(S) = 3.0$.

3.4 Data Storage in CMAC

Data stored in CMAC is based on its estimate. The following procedure explains how cerebellar learning is modeled in the CMAC:

1. Let \hat{H} be the CMAC's estimate of the function to be computed, and $\hat{P} = \hat{H}(S)$ the CMAC's attempt of estimating the desired value of each point in the input space.
2. Compute the desired value of the function for every input in S: $P = H(S)$
3. For every element in $\hat{P} = (\hat{p}_1, \hat{p}_2, \dots, \hat{p}_L)$, add Δ_i to every weight which was used to produce p_i where:

$$\Delta_i = g \frac{\hat{p}_i - p_i}{A}$$

A: the number of weights used to compute p_i

g: the gain factor controlling the amount of error correction

3.5 Memory Size Requirements in CMAC

In a conventional look-up table, each input address selects a unique location where the contents of that address are stored. In CMAC, each input address selects a unique set of memory locations in which the sum of their contents is the contents of the input address. This means that CMAC requires less memory than the conventional look-up table to store a specific function. A conventional look-up table requires R^N memory locations to store a function that has N variables, and R number of resolution elements on each variable. The number of memory locations required for the same function is QK^N , where Q is the number of quantizing functions and K is the number of resolution elements on each

quantizing functions. An example of CMAC versus conventional memory(look-up table) is illustrated in Figure 3.4 where $N = 2$ and $R = 13$. Using the conventional memory, 13^2 or 169 memory locations are required to cover the input space. This number reduces to 4×4^2 or 64 memory locations. This advantage of the CMAC becomes significant for large number of input variables (N). The number of memory locations required in CMAC can be further reduced using the hash coding techniques. Hash coding is a memory-addressing technique which maps a large address space into a smaller one. This technique maps many addresses in the larger space onto the same addresses in the smaller one. Therefore, the method works well when the original address space is sparsely populated and the probability of collision is low. CMAC's tolerance, unlike the conventional memory, is fairly high for the number of collisions occurred. This is because CMAC is distributed memory and a collision introduces only a small error into the output.

3.6 Generalization in the CMAC Memory

As previously explained, CMAC input vector selects a unique set of memory locations which implies that every memory location may be selected by more than one input. Also, this mapping ensures that input vectors that are close together in the input space activate many of the same granule cells. This is how CMAC memory generalizes. The extent of the neighborhood of generalization depends on the resolutions and the number of quantization functions used to quantize the input variables. It is possible to have different resolution for different input variables. Figure 3.5 shows the memory distribution for 2 adjacent points in the input space.

One of the advantages of generalization includes producing close estimate of the function

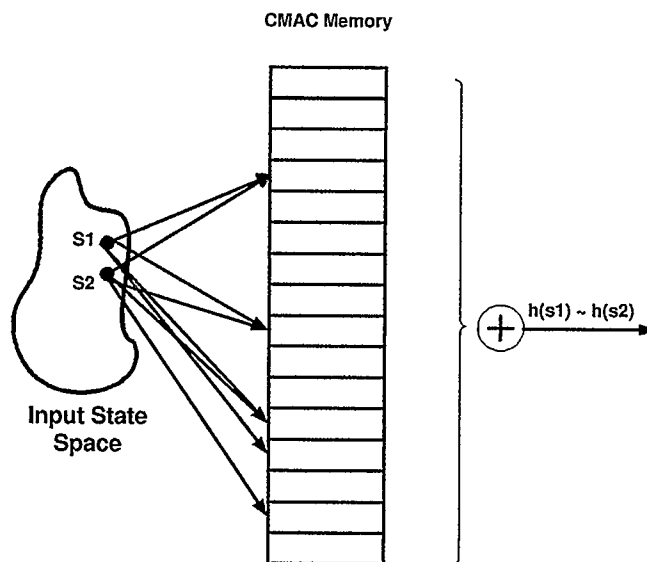


Figure 3.5: Memory distribution for two adjacent points in the input space

without having to train every point in the input space. Also, it allows CMAC to predict the appropriate response for a situation based on few similar learning experiences. This is highly practical in real world environment where identical trajectories hardly reoccur.

Chapter 4

Simulation

4.1 Dynamics

The simulation is performed on a two-link (horizontal) planar robotic manipulator (Figure 4.1). The dynamics of this system can be expressed as in [8]:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \boldsymbol{\tau} \quad (4.1.0.1)$$

where $\boldsymbol{\tau} \in \mathbb{R}^2$ contains the control torques, \mathbf{q} , $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}} \in \mathbb{R}^2$ are vectors of link angles, angular velocities, and angular accelerations respectively. $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{2 \times 2}$ is the inertia matrix, and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{2 \times 2}$ is the Coriolis and centripetal matrix. It should be noted that the gravity term does not appear in Eq. 4.1.0.1 because the planar robot is horizontal. Also, for simplicity, the friction force is ignored.

As mentioned in section 2.4, the inertia matrix $\mathbf{M}(\mathbf{q})$ and the Coriolis/centripetal forces

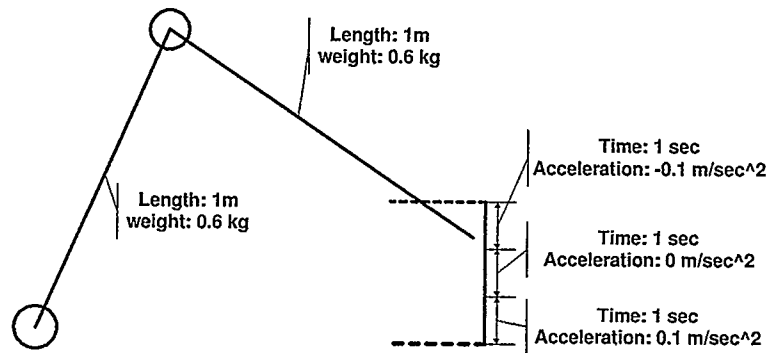


Figure 4.1: Trajectory to be followed

$\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ of a two-link planar robot can be described as:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} p_1 + p_2 + 2p_3 \cos(q_2) & p_2 + p_3 \cos(q_2) \\ p_2 + p_3 \cos(q_2) & p_2 \end{bmatrix} \quad (4.1.0.2)$$

$$\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} h\dot{q}_2 & h\dot{q}_1 + h\dot{q}_2 \\ -h\dot{q}_1 & 0 \end{bmatrix} \quad (4.1.0.3)$$

$$\begin{aligned} \text{for:} \quad p_1 &= m_1 L_1^2 + m_2 L_1^2 & p_2 &= m_2 L_2^2 \\ p_3 &= m_2 L_1 L_2 & h &= -m_2 L_1 L_2 \sin(q_2) \end{aligned}$$

where m_i and L_i are the mass and length of link i respectively.

For the simulations conducted in this thesis, two CMAC neural networks are used; one to estimate the positive definite and symmetric matrix $\mathbf{M}^{-1}(\mathbf{q})$, and one to model the vector $-\mathbf{M}^{-1}(\mathbf{q})\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$. CMAC 1 estimate (\mathbf{C}_F) contains three outputs predicting the terms representing the symmetric $\mathbf{M}^{-1}(\mathbf{q})$, and CMAC 2 estimate (\mathbf{C}_M) contains two outputs approximating the terms inside the vector $-\mathbf{M}^{-1}(\mathbf{q})\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$. The CMAC neural networks do not have any previous knowledge of the system. Therefore, for simulation, \mathbf{C}_M estimate is initially set to be an identity matrix ($\mathbf{I}_{2 \times 2}$), and \mathbf{C}_F to be a vector of zeros ($\mathbf{0}_2$). The CMAC weights representing \mathbf{C}_M are only updated if the result is not a singular matrix (i.e. invertible).

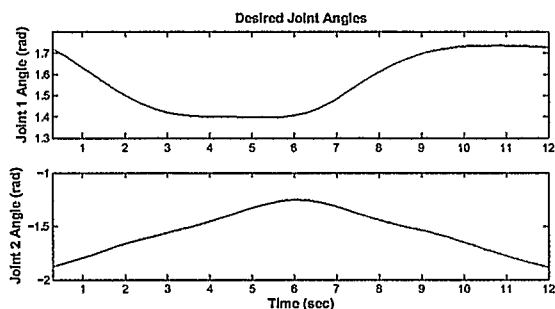


Figure 4.2: Desired joint angle

4.2 Objective

The end-effector of the robot is to track a $20\text{cm} \times 20\text{cm}$ square with specific desired acceleration (Figure 4.1). Each side of the square is divided into three sections. The end-effector moves along each section for 1 second and with an acceleration of $0.1 \frac{m}{\text{sec}^2}$, $0 \frac{m}{\text{sec}^2}$, and $-0.1 \frac{m}{\text{sec}^2}$ respectively. It takes the robot 12 seconds to finish one iteration. Using Inverse kinematics, the desired angle and angular velocities can be defined using the end effector position. Figures 4.2 and 4.3 illustrate the desired angles and angular velocities respectively. These trajectories can be used to determine the tracking error, which in turn

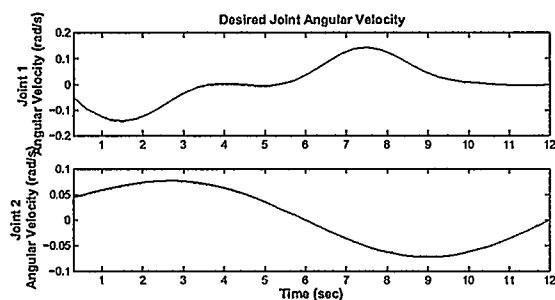


Figure 4.3: Desired joint angular velocity

is used to produce the desired torque.

4.3 Simulation Parameters

4.3.1 Investigating the Weight Update Law

Three different approaches to the proposed weight update law are investigated. The control torque and weight update laws for all three approaches are:

$$\tau = \mathbf{C}_M^{-1}(\mathbf{C}_F + \ddot{\mathbf{q}}_d - \mathbf{G}_1 \mathbf{e}_1 - \mathbf{G}_2 \mathbf{e}_2 - \alpha \|u_w\|^2 \mathbf{B}^T \mathbf{P}^T \mathbf{z}) \quad (4.3.1.1)$$

$$\dot{\hat{\mathbf{w}}}_i = \beta_i(\eta_i - v_i \hat{\mathbf{w}}_i + \gamma_i \phi(g_i - \phi^T \hat{\mathbf{w}}_i)) \quad (4.3.1.2)$$

$$\phi = \phi_M \quad i = 1, 2, 3 \quad \text{and} \quad \phi = \phi_F \quad i = 4, 5$$

The parameters of the torque equation will remain the same for all three simulations. The values of these parameters are:

$$\mathbf{G}_1 = \mathbf{G}_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \quad \text{and} \quad \alpha = 0.01$$

The parameters for the weight update, however, change according to Table 4.1. The values in this table are chosen based on the original objective shown in Table 2.1.

	$\beta_{1,\dots,5}$	$v_{1,\dots,5}$	$\gamma_{1,\dots,5}$
Method 1	0.5, 0.5, 0.5, 1, 1	0.5, ..., 0.5	0, ..., 0
Method 2	0.05, ..., 0.05	0, ..., 0	1000, ..., 1000
Method 3	0.05, 0.05, 0.05, 5, 5	0, 0, 0, 0.01, 0.01	1000, 1000, 1000, 0, 0

Table 4.1: Investigating the proposed weight update

4.3.2 Proposed scheme versus Computed Torque

The third method is the novel one, and is used for comparison with the computed torque scheme. The proposed method uses the model estimate and updates the weight update according to parameters shown in Table 4.1, whereas the computed torque scheme uses the real model of the system. The physical parameters of the manipulator are tabulated in Table 4.2. The comparison is also conducted with a changing payload. This is to illustrate how each method cope when the dynamic model of the system changes. The payload, or equally, uncertainties in m_2 used for the simulation, range from 0 to 2 Kg. The idea is to simulate a case where the robot arm picks up a load between 0 to 2 Kg, and drops it off at its destination.

Robot Parameter	Link 1	Link 2
mass (Kg)	0.6	0.6
length (m)	1	1

Table 4.2: Physical parameters

Chapter 5

Examining Different Forward Dynamic Approaches

This chapter compares three different approaches of forward-dynamic CMAC control, clarifying the need for a model estimate of inertia matrix.

1. State error-based weight updates
2. Supervised learning using the model estimates
3. Combined error-based updates and model estimate of inertia matrix

For all three approaches the control and the weight update equations can be presented as:

$$\tau = \mathbf{C}_M^{-1}(\mathbf{C}_F + \ddot{\mathbf{q}}_d - G_1 \mathbf{e}_1 - G_2 \mathbf{e}_2 - \alpha \|u_w\|^2 \mathbf{B}^T \mathbf{P}^T \mathbf{z}) \quad (5.0.2.1)$$

$$\dot{\hat{\mathbf{w}}}_i = \beta_i (\eta_i - v_i \hat{\mathbf{w}}_i + \gamma_i \phi (g_i - \phi^T \hat{\mathbf{w}}_i)) \quad (5.0.2.2)$$

$$\phi = \phi_M \quad i = 1, 2, 3 \quad \text{and} \quad \phi = \phi_F \quad i = 4, 5$$

A stability proof for the above control and weight update law is provided in chapter 7. The main difference between the approaches is the way the weight update parameters are tuned. These parameters can be explained quantitatively:

- η_i : is a state error dependent term that updates the weights according to the tracking error
- $v \hat{\mathbf{w}}_i$: is a leakage term that prevent the weights from drifting to infinity by forcing them to zero

- $\gamma_i \phi(g_i - \phi^T \hat{\mathbf{w}}_i)$: is a learning term which uses the error between the CMAC model ($\phi^T \hat{\mathbf{w}}_i$) and the model estimate (g_i) to train the CMAC neural networks

As will be explained later in this chapter, there is no need for both the learning and the leakage term, and they do not simultaneously appear in the weight update equations. After comparison, it can be concluded that the third approach results in the best performance.

5.1 State-Error Based Weight Updates (Method 1)

This method is a common way of estimating the inverse dynamics parameters using neural networks. This section investigates the performance of this method when used in forward dynamics fashion as conducted in [7]. It comprises of two CMAC neural networks that are trained solely based on the system's state error. The Uniformly Ultimately Boundedness of the stability proof in chapter 7 holds for this method and can be analyzed by letting γ_i equal zero. The forward-dynamics control law is the same as the one shown in (5.0.2.1). The weight update equations are formed by letting the learning factor (γ_i) equal to zero:

$$\dot{\hat{\mathbf{w}}}_1 = \beta_1(\phi_M(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{11} - \nu_1 \hat{\mathbf{w}}_1) \quad (5.1.0.3)$$

$$\dot{\hat{\mathbf{w}}}_2 = \beta_2(\phi_M[(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{12} + (\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{21}] - \nu_2 \hat{\mathbf{w}}_2) \quad (5.1.0.4)$$

$$\dot{\hat{\mathbf{w}}}_3 = \beta_3(\phi_M(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{22} - \nu_3 \hat{\mathbf{w}}_3) \quad (5.1.0.5)$$

$$\dot{\hat{\mathbf{w}}}_4 = \beta_4(\phi_F(\mathbf{z}^T \mathbf{P} \mathbf{B})_{11} - \nu_4 \hat{\mathbf{w}}_4) \quad (5.1.0.6)$$

$$\dot{\hat{\mathbf{w}}}_5 = \beta_5(\phi_F(\mathbf{z}^T \mathbf{P} \mathbf{B})_{21} - \nu_5 \hat{\mathbf{w}}_5) \quad (5.1.0.7)$$

where $\mathbf{z} = [\mathbf{e}_1 \quad \mathbf{e}_2]^T$, ν_i and β_i are positive constants, and $\mathbf{P} = \mathbf{P}^T > 0$ is the solution of the Lyapunov equation:

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

\mathbf{Q} is a positive definite matrix, and

$$\mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{G}_1 & \mathbf{G}_2 \end{bmatrix}$$

where $\mathbf{0} \in \mathbb{R}^{2 \times 2}$ is a matrix of zeros, $\mathbf{1} \in \mathbb{R}^{2 \times 2}$ is an identity matrix, and $\mathbf{G}_1, \mathbf{G}_2 \in \mathbb{R}^{2 \times 2}$ are positive definite gain matrices.

5.1.1 Tuning the Weight Update Parameters

As explained in the previous section, the weight updates for this method include the leakage term $\nu_i \hat{\mathbf{w}}_i$ in addition to the state error dependent term. The purpose of the leakage

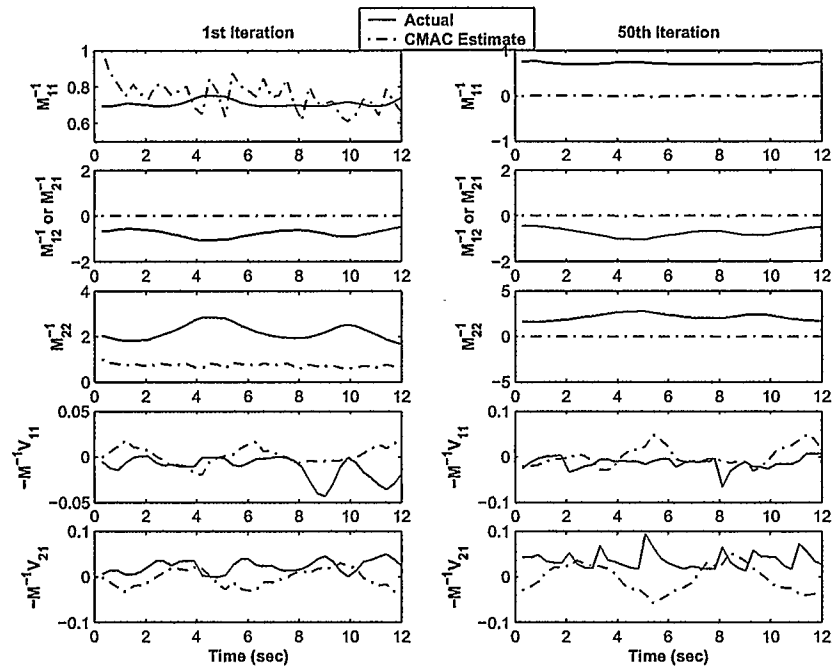


Figure 5.1: Comparing method 1 CMAC estimates with their actual values for the 1st and 50th iteration

term is to direct the weights toward zero and stop them from drifting to infinity. Therefore, parameter ν_i has to be chosen as small as possible so that it does not deteriorate the effect of the error term, but large enough to keep the system stable (stop the weights from drifting to infinity).

The other parameter that needs to be tuned is β_i . This parameter determines how fast the CMAC weights are converging. For fast convergence, large value of β_i is desired. However β_i has to be small enough to let the weights converge to their real values. The parameters used for the simulation of method 1 are $\beta_{1,2,3} = 0.5, \beta_{4,5} = 1, \nu_{1,2,\dots,5} = 0.5$. Figure 5.1 illustrates the 1st and 50th iteration simulated using Method 1. As expected, the CMAC estimates do not approach their actual values, which means that learning the real model is impossible using just the tracking error.

5.1.2 Performance

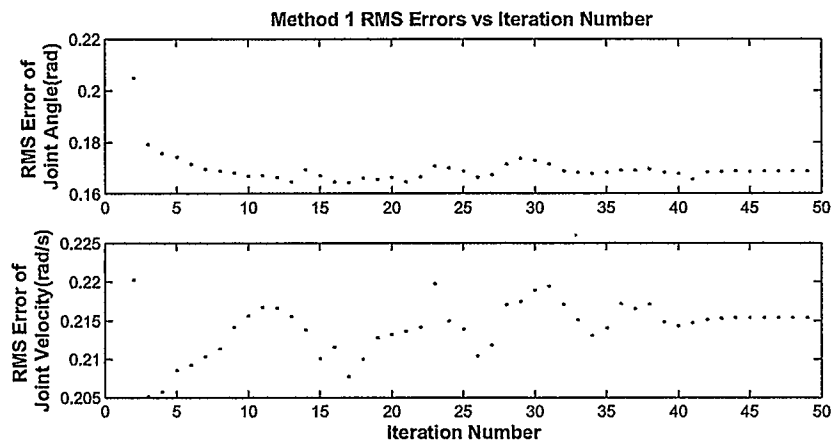


Figure 5.2: RMS error of the angles and angular velocities for the Method 1

Through simulation, it was concluded that state-error based weight updates are not

suitable for forward dynamics approach applications. The RMS error of the angles and angular velocities does not necessarily improve (Figure 5.2) with the number of iteration. In fact, it is even hard to keep the system stable. This behavior can be explained by examining the weight update and the control law. The final value of the control torque (5.0.2.1) is determined by multiplying the CMAC model of the inverse of the inertia matrix ($\hat{\mathbf{C}}_M^{-1}$) to a set of terms in a bracket. For that reason, failure to produce a somewhat accurate CMAC model of $\mathbf{M}(\mathbf{q})$ will radically deteriorate the performance.

As mentioned earlier, the weights are updated based on the state error which does not necessarily guide the weights toward their actual values. Therefore, the CMAC model of the inertia matrix is very unlikely to approach the real model (Figure 5.1). Since producing a somewhat accurate CMAC model of $\mathbf{M}(\mathbf{q})$ is essential for performance, the CMAC controller fails to follow the trajectories.

5.2 Supervised learning (Method 2)

In the previous section, it was concluded that one major factor that caused the method to fail was the lack of the ability to produce somewhat accurate CMAC model of the inertia matrix. In order to model \mathbf{M}^{-1} and $\mathbf{M}^{-1}\mathbf{V}$ more accurately, the weights have to be guided toward their real values. For that reason, a learning term is added to the weight update laws. This learning term is the error between the CMAC model and the model estimate. Like method 1, this method uses the control law (5.0.2.1). Incorporating the learning term,

the weight updates effectively become:

$$\dot{\hat{\mathbf{w}}}_1 = \beta_1(\phi_M(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{11} + \gamma_1 \phi_M(g_1 - \phi_M^T \hat{\mathbf{w}}_1)) \quad (5.2.0.1)$$

$$\dot{\hat{\mathbf{w}}}_2 = \beta_2(\phi_M[(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{12} + (\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{21}] + \gamma_2 \phi_M(g_2 - \phi_M^T \hat{\mathbf{w}}_2)) \quad (5.2.0.2)$$

$$\dot{\hat{\mathbf{w}}}_3 = \beta_3(\phi_M(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{22} + \gamma_3 \phi_M(g_3 - \phi_M^T \hat{\mathbf{w}}_3)) \quad (5.2.0.3)$$

$$\dot{\hat{\mathbf{w}}}_4 = \beta_4(\phi_F(\mathbf{z}^T \mathbf{P} \mathbf{B})_{11} - \gamma_4 \phi_F(g_4 - \phi_F^T \hat{\mathbf{w}}_4)) \quad (5.2.0.4)$$

$$\dot{\hat{\mathbf{w}}}_5 = \beta_5(\phi_F(\mathbf{z}^T \mathbf{P} \mathbf{B})_{21} - \gamma_5 \phi_F(g_5 - \phi_F^T \hat{\mathbf{w}}_5)) \quad (5.2.0.5)$$

where $\gamma_i > 0$, and g_i is the model estimate term used to supervise the learning

5.2.1 Tuning the Weight Update Parameters

Comparing the weight updates from method 2 to 1, not only a learning term is added, but also the leakage term is taken out. The weight updates used in method 2 do not include the commonly used stability terms such as, e-modification, leakage, dead zone and etc. Instead, the learning terms $\gamma_i \phi_M(g_i - \phi_M^T \hat{\mathbf{w}}_i)$ and $\gamma_i \phi_F(g_i - \phi_F^T \hat{\mathbf{w}}_i)$ are respectively added to equations calculating \mathbf{C}_M and \mathbf{C}_F to supervise the learning. The main purpose of using a stability term is to keep the weights from drifting to infinity. In the above weight updates, the learning terms guide the CMAC models to their real models, and prevent drifting of CMAC weights to large values. This is why the leakage term, which weakens the learning ability of the weight updates, is no longer required.

As previously mentioned, this method concentrates on producing accurate CMAC model of the forward dynamic parameters and so β and γ , which are the parameters determining the effect of the terms in the weight update, have to be tuned accordingly:

- Initially, the value of the learning term has to overcome the state error dependent term

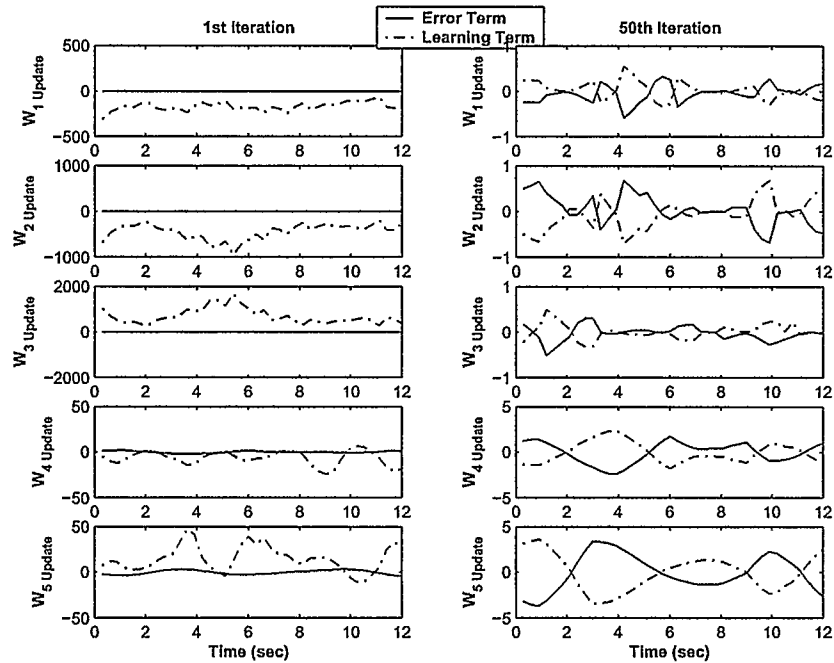


Figure 5.3: Comparing method 2 values of the state error term with the learning term in the weight update

- The value of the learning term and the error dependent term have to be comparable after convergence

The first rule of thumb ensures that the weights are guided to their real values, whereas having comparable terms in the weight update after convergence shows system stability.

Figure 5.3 illustrates comparison between the learning and the state error dependent terms in the weight update for the 1st and 50th iteration. As illustrated, the learning terms clearly dominate the error terms during the first iteration. However, these terms cancel each other out during the 50th iteration which suggests that the CMAC weights are no longer being updated and have converged to their real values. The values that these weights have converged to are shown in Figure 5.4. The comparison between the 1st and 50th

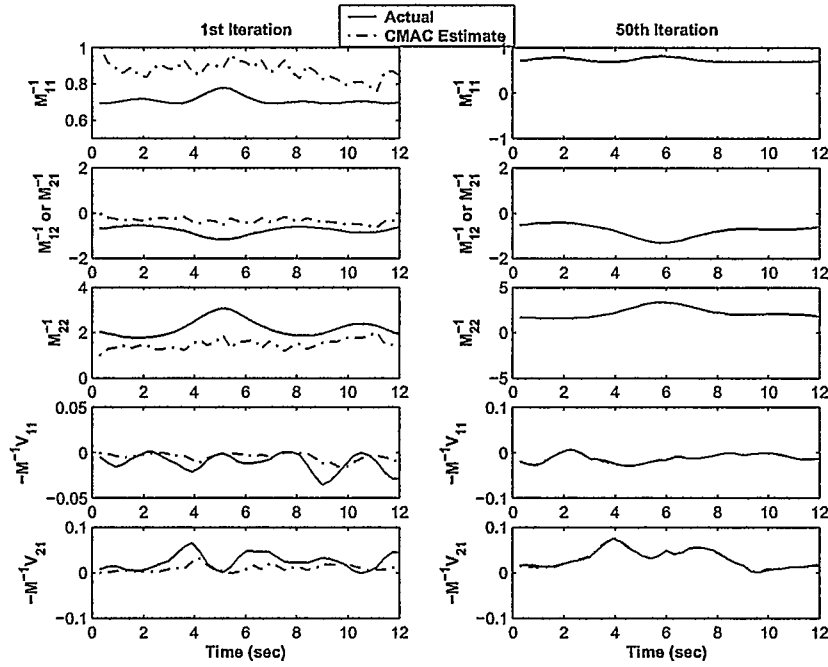


Figure 5.4: Comparing method 2 CMAC estimates with their actual values for the 1st and 50th iteration

iteration suggests that the chosen weight update factors guide the CMAC outputs to their real values, and the weights eventually converge to their real values.

5.2.2 Performance

Through simulation, it was illustrated that the RMS error for this method improves by the number of iterations and does not fluctuate randomly (Figure 5.5). It is also much easier to produce a stable system because one does not have to worry about radical changes of the control torque due to inaccurate C_M^{-1} . The parameters used for the above simulation were $\beta_{1,2,\dots,5} = 0.05$, $\gamma_{1,2,\dots,5} = 1000$. Although the RMS error is consistent, it does not improve much with the number of iterations. Also, the RMS error value after convergence suggests

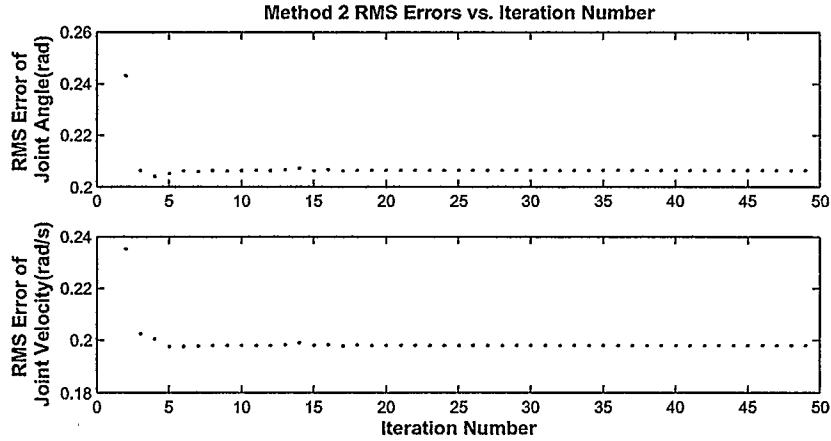


Figure 5.5: RMS error of the angles and angular velocities for the Method 2

that the robot's end-effector does not closely track the desired trajectories. Examining the control term (5.0.2.1), it can be concluded that producing accurate CMAC model of $\mathbf{M}^{-1}(\mathbf{q})$ is advantageous, but updating the terms in \mathbf{C}_F based on the learning term could deteriorate the performance. As shown in Eq. (5.0.2.1), the value of \mathbf{C}_F is added to the PD feedback error, and there is no guarantee that they even have the same sign. Therefore, \mathbf{C}_F could act as a large disturbance to the PD feedback error. More on this will be covered in the next section.

5.3 Supervisory inertia matrix (Method 3)

In this section, it will be illustrated that updating the terms in \mathbf{C}_F based on the learning term does not result in a better performance, and may even worsen the tracking error. To show this, the model estimate is only used for CMAC model of the inertia matrix (\mathbf{C}_M). The weights used in CMAC model of $\mathbf{M}^{-1}\mathbf{V}$ (\mathbf{C}_F) will be updated based on the state error (similar to method 1). After tuning the parameters, the weight updates will effectively

become:

$$\dot{\hat{\mathbf{w}}}_1 = \beta_1(\phi_M(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{11} + \gamma_1 \phi_M(g_1 - \phi_M^T \hat{\mathbf{w}}_1)) \quad (5.3.0.1)$$

$$\dot{\hat{\mathbf{w}}}_2 = \beta_2(\phi_M[(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{12} + (\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{21}] + \gamma_2 \phi_M(g_2 - \phi_M^T \hat{\mathbf{w}}_2)) \quad (5.3.0.2)$$

$$\dot{\hat{\mathbf{w}}}_3 = \beta_3(\phi_M(\tau \mathbf{z}^T \mathbf{P} \mathbf{B})_{22} + \gamma_3 \phi_M(g_3 - \phi_M^T \hat{\mathbf{w}}_3)) \quad (5.3.0.3)$$

$$\dot{\hat{\mathbf{w}}}_4 = \beta_4(\phi_F(\mathbf{z}^T \mathbf{P} \mathbf{B})_{11} - \nu_i \hat{\mathbf{w}}_4) \quad (5.3.0.4)$$

$$\dot{\hat{\mathbf{w}}}_5 = \beta_5(\phi_F(\mathbf{z}^T \mathbf{P} \mathbf{B})_{21} - \nu_i \hat{\mathbf{w}}_5) \quad (5.3.0.5)$$

The control is the same as the previous methods (5.0.2.1).

5.3.1 Tuning the Weight Update Parameters

The initial reason for updating the CMAC weights for \mathbf{C}_F based on the state error was to avoid its negative effects on the PD feedback error. However, it turns out that there are additional advantages. This way, the weight updates $\hat{\mathbf{w}}_3$ and $\hat{\mathbf{w}}_4$ are updated using the error in angles (\mathbf{e}) and angular velocities ($\dot{\mathbf{e}}$). Therefore, \mathbf{w}_4 and \mathbf{w}_5 and so \mathbf{C}_F will depend on $\int \mathbf{e}$ and \mathbf{e} . Since \mathbf{C}_F is used in the control equation (5.0.2.1), the control torque will now depend on $\int \mathbf{e}$, and effectively becomes a PID controller. The parameters used for the simulation of this method were: $\beta_{1,2,3} = 0.05$, $\beta_{4,5} = 5$, $\gamma_{1,2,3} = 1000$, and $\nu_{4,5} = 0.01$. To examine how close the CMAC models get to the real model using the chosen parameters, they are plotted and compared in Figure 5.6. During the 50th iteration, the CMAC model of \mathbf{M}^{-1} converges to its real model. However, as intended, the CMAC model of $-\mathbf{M}^{-1}\mathbf{V}$ does not converge to its real model. It is updated according to the state error, and helps to improve the performance.

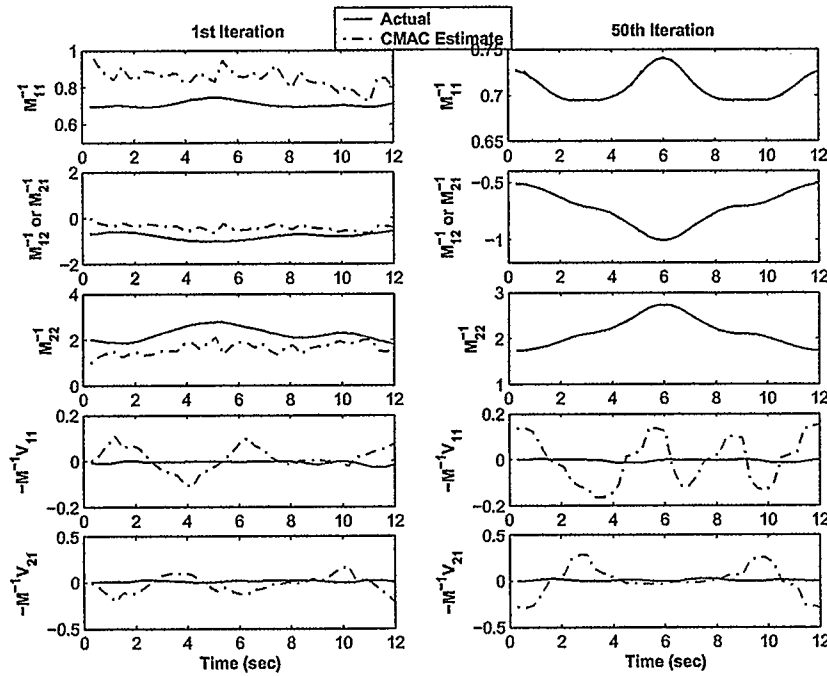


Figure 5.6: Comparing CMAC method 3 estimates with their actual values for the 1st and 50th iterations

5.3.2 Performance

Figure 5.7 illustrates the RMS error of the angles and angular velocities for method 3. Unlike the last two methods, the RMS error improves by the number of iterations and reaches an acceptable level. This means that the control torque is adequate for close tracking of the trajectories. The following section covers a more detailed comparison between the three methods.

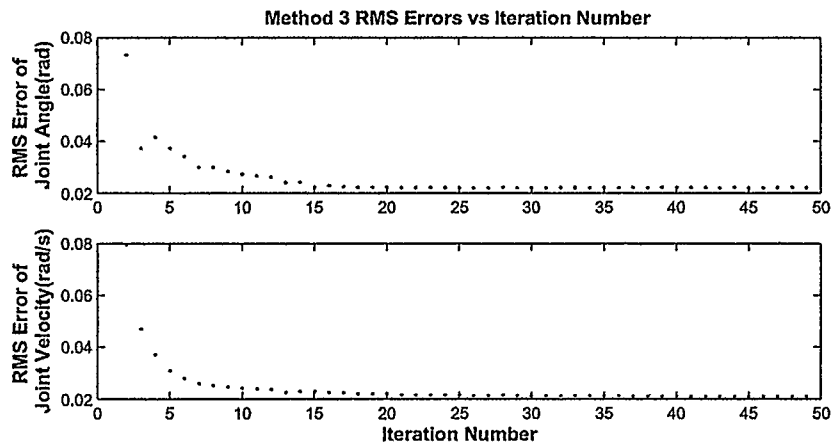


Figure 5.7: RMS error of the angles and angular velocities for the Method 3

5.4 Comparing Performances

Previous sections of this chapter evaluated the performances of each method by examining the error between the actual and desired angles and angular velocities. To get a feel of what state error means in terms of the robot's end effector position, the error between the end-effector position and the desired position after weight convergence is plotted and compared in Figure 5.8. For all three simulations, the robot has to follow a $20\text{ cm} \times 20\text{ cm}$ square.

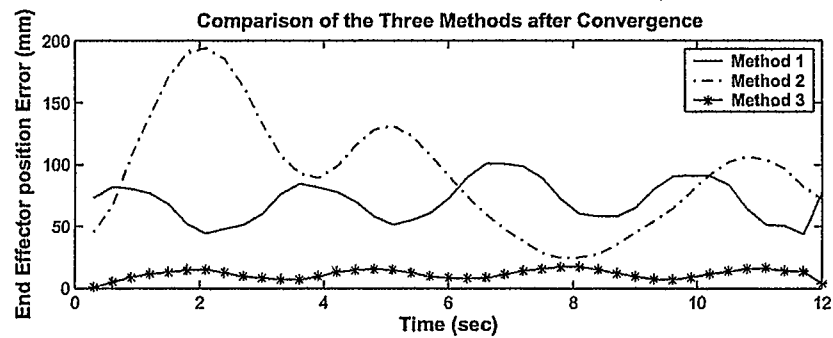


Figure 5.8: Comparing the performance of the three methods

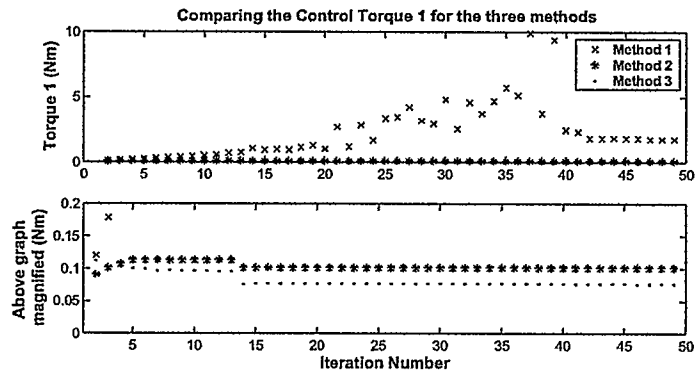


Figure 5.9: Comparing the amount of torque required for joint 1 to achieve the performance illustrated in 5.8

For the first two methods, the position errors are as high as 10 *cm* and 20 *cm* respectively. Method 3 shows a clear advantage over the other methods. The other factor that needs to be investigated is the output torque for each method. In general increasing the feedback gain will cause the system to produce more torque initially, which in turn would result in a closer tracking of the error. Therefore, the real advantage is seen when the improvement of performance results without any increase in the amount of torque required.

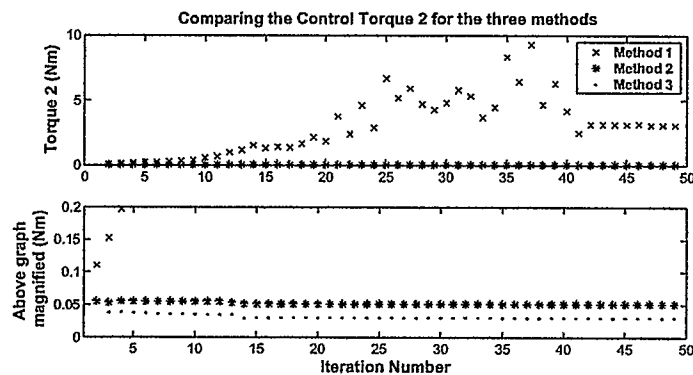


Figure 5.10: Comparing the amount of torque required for joint 2 to achieve the performance illustrated in 5.8

Figures 5.9 and 5.10 compare the amount of torque required for joint 1 and joint 2 to produce the performance shown in Figure 5.8. The amount of torque applied at the beginning is almost the same for all three methods. However, method 3 achieves closer trajectory tracking and requires less torque at the end.

Chapter 6

Comparison / Robustness of the Proposed Method

In what follows, the proposed method is first compared to the computed torque controller. Then the practicality of the proposed method is investigated by examining the performance under the model uncertainty caused by changing the mass of link 2 (m_2) in the simulation. This could equivalently be thought of as a changing payload. The key to the applicability of the proposed method is to ensure that the performance is not significantly affected under the model uncertainties.

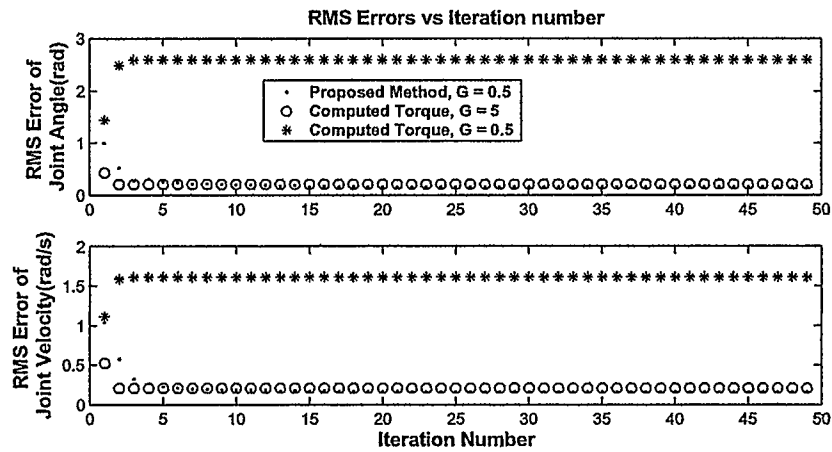


Figure 6.1: Comparison between the RMS error of the angles and angular velocities of the proposed method with a computed torque controller

6.1 Comparison between the Proposed Method and the Computed Torque Scheme

The reason for choosing computed torque scheme for comparison is that it too uses the forward dynamics model of the system to produce the required torque. The first comparison is conducted under the assumption that exact model of the system is known. For this simulation, two different gains are used to emphasize the advantages of the proposed method. Figure 6.1 illustrates the RMS error between the actual and desired angles and angular velocities for both schemes. When the feedback gain is (0.5) for both schemes, the proposed method shows a clear advantage over the computed torque. Unlike the computed torque, the performance of the proposed method improves with the number of iterations, and the error converges in less than 10 iterations. Same figure shows the improvement of computed torque performance when the feedback gain is increased to 5. It is illustrated that the performance of the computed torque reaches the same level as the proposed method after increasing the gain. To illustrate how closely the robot is tracking the desired path, the

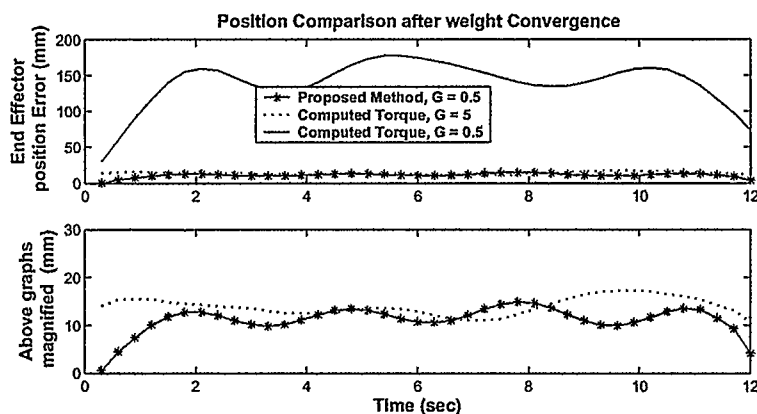


Figure 6.2: Comparing the position error of the proposed method with a computed torque controller after convergence

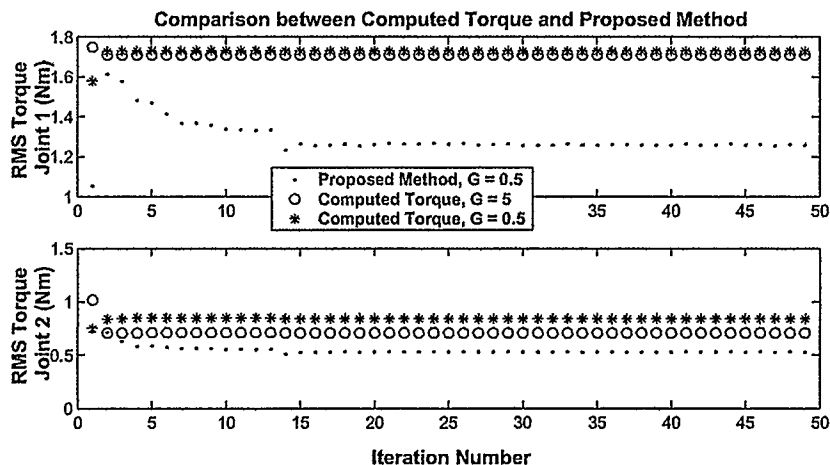


Figure 6.3: Comparing the required RMS torque for the proposed and the computed torque scheme

error between the end-effector position and the desired position, after weight convergence, is plotted and compared in Figure 6.2. The performance of the computed torque with the feedback gain of 0.5 is unacceptable. In this case, the position error reaches 17.5 cm for a $20\text{ cm} \times 20\text{ cm}$ square. When the feedback gain is increased to 5, the level of performance matches that of the proposed method (position error $\cong 1.5\text{ cm}$). It should be noted that the proposed method uses a gain of 0.5 in all cases.

As explained in the previous chapter, a fair comparison between different schemes includes comparing the performances as well as the torque required to achieve those performances. Figure 6.3 illustrates the torque required for each method to produce the performances shown in Figure 6.1. Notice that the initial applied torque increased when the feedback gain was increased from 0.5 to 5 for the computed torque method. Also, the torque applied at all time, including the initial torque and the torque after the system has converged, for the proposed method is lower than the other two cases. Therefore, the pro-

posed method results in a closer tracking of the trajectories and requires less torque to do so.

6.2 Robustness of proposed method

In the previous section, a comparison was made assuming exact knowledge of the system's dynamic model. In this section, through simulation, it is illustrated that the model estimate used for training of the inverse of the inertia matrix need not be accurate for the proposed method to perform well. The simulations are conducted in the same way with the exception that the actual value of m_2 is altered from $2Kg$ to $4Kg$ whereas its value used for training is kept constant at $2Kg$. The added uncertainties in m_2 corresponds to a changing payload or to uncertainties in the model estimates. To illustrate accuracy of the model estimate of inertia matrix (g_i) for different mass uncertainties, the error between g_i and the terms in the real model is recorded and plotted after the weight convergence. Figure 6.4 illustrates the norm of this error as the uncertainty in m_2 alters from 0 to 2 Kg for feedback gains of 0.5, 1, and 5. The error in the model estimate of the inertia matrix

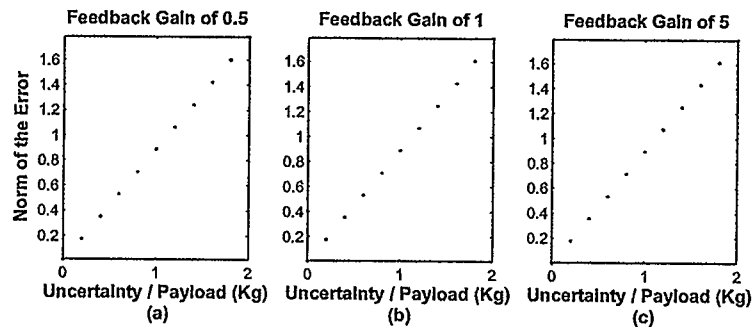


Figure 6.4: Norm of the error between the terms in the model estimate and real model of the inertia matrix for different mass uncertainties / payloads

increases with uncertainty in m_2 , and is similar for the three feedback gain.

To illustrate the effect of the above uncertainty in the model estimate or changing payload on CMAC's performance, the error in the end-effector position is inspected. The same test is repeated for a computed torque controller for comparison. Figure 6.5 shows the norm of the position error after weight convergence for feedback gains of 0.5, 1, and 5.

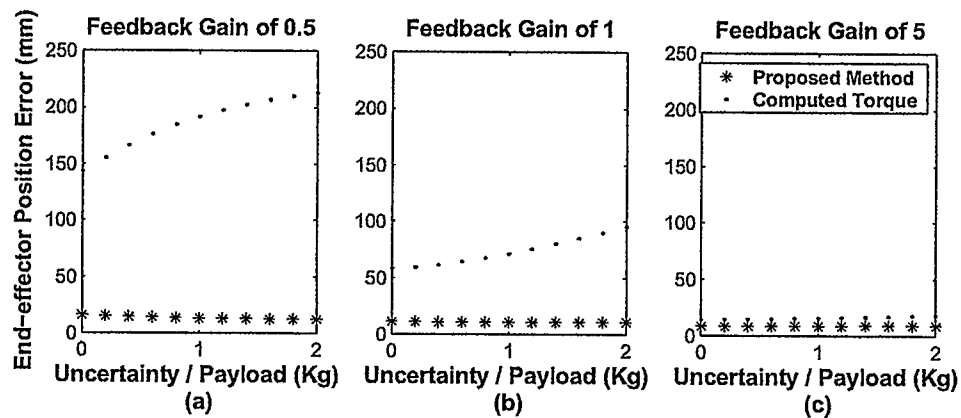


Figure 6.5: position error of the end-effector for different payloads/mass uncertainties

As illustrated in Figure 6.5, for the proposed method, the error in end-effector position stays small when m_2 changes from 0 to 2 Kg. This means acceptable performance can be achieved with limited error in the model estimate. It also illustrates that the proposed method can adapt itself with payload changes. Comparison between the proposed method and a computed torque controller shows that the proposed method performs drastically better, when the feedback gain is small. This could be advantageous when the required torque exceeds the maximum limit of the motors controlling the joints.

Chapter 7

Stability Proof

This chapter begins with presenting concepts and theories required to understand function approximation of CMAC neural networks. The theories are then used along with Lyapunov stability to prove that the system is Uniformly Ultimately Bounded.

7.1 Function Approximation Using CMAC Neural Networks

In order to verify approximation properties of CMAC Neural Network, its quantization functions $Q(x)$ must satisfy the interpolation property. For that reason, the quantization functions are chosen so that they span the entire input space and satisfy the normalization property.

For the simulations conducted in this thesis, the CMAC Neural Network obtains 10 inputs including the desired and actual trajectories. The desired trajectories are the desired angles (\mathbf{q}_d), desired angular velocities ($\dot{\mathbf{q}}_d$), and the desired accelerations ($\ddot{\mathbf{q}}_d$), and the actual trajectories include angles (\mathbf{q}) and angular velocities ($\dot{\mathbf{q}}$). For the CMAC to span this input space, the domain is chosen as:

$$\begin{aligned} \mathbf{q}_1 \ \& \ \mathbf{q}_{1d} &= [0 \ \pi] \ (\text{rad}) & \dot{\mathbf{q}}_1 \ \& \ \dot{\mathbf{q}}_{1d} &= [-1 \ 1] \ \left(\frac{\text{rad}}{\text{sec}} \right) \\ \mathbf{q}_2 \ \& \ \mathbf{q}_{2d} &= [-\pi \ 0] \ (\text{rad}) & \dot{\mathbf{q}}_2 \ \& \ \dot{\mathbf{q}}_{2d} &= [-1 \ 1] \ \left(\frac{\text{rad}}{\text{sec}} \right) \\ \ddot{\mathbf{q}}_{1d} \ \& \ \ddot{\mathbf{q}}_{2d} &= [-1 \ 1] \ \left(\frac{\text{rad}}{\text{sec}^2} \right) \end{aligned}$$

As explained in chapter 3, the components of the input vectors must be mapped to their corresponding CMAC granule cell. The CMAC used in this thesis utilizes triangular functions ($tr_{i,g_i}(x_i)$) for the mapping. Here x_i and g_i denote the input and the granule cell number in i^{th} input dimension respectively. $tr_{i,g_i}(x_i)$ is defined as:

$$tr_{i,g_i}(x_i) = \begin{cases} \frac{x_i-a}{b-a} & a \leq x_i \leq b \\ \frac{c-x_i}{c-b} & b \leq x_i \leq c \\ 0 & \text{otherwise} \end{cases}$$

where a and c are the boundaries, b is the midpoint, and y is the input position for each individual granule cell.

Using the triangular functions and satisfying the normalization property, the quantization function for any given $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n] \in \mathfrak{R}$ is defined as:

$$Q_{g_1, g_2, \dots, g_n}(\mathbf{x}) = \frac{\prod_{i=1}^n tr_{i, g_i}(x_i)}{\sum_{j_n=1}^{N_n} \dots \sum_{j_1=1}^{N_1} \prod_{i=1}^n tr_{i, g_i}(x_i)} \quad (7.1.0.1)$$

Therefore, the quantization functions of the CMAC used in this thesis is shown to span its entire input space and satisfies the normalization property.

Lemma 7.1.0.1. *Given $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n] \in \mathfrak{R}$, the quantization function defined in (7.1.0.1) form a cardinal bases that satisfy the interpolation property:*

$$Q_{g_1, g_2, \dots, g_n}(\mathbf{x}) = \delta_{1, g_1} \delta_{2, g_2} \dots \delta_{n, g_n}$$

Proof: See [32]

From lemma 7.1.0.1 it can be concluded that CMAC can be used to approximate functions.

Furthermore, CMAC approximation of any continuously differentiable function $f(\mathbf{x}) :$

$\mathfrak{R}^n \rightarrow \mathfrak{R}^m$ can be given by $\hat{f}(\mathbf{x}) = [\hat{f}_1 \ \hat{f}_2 \ \dots \ \hat{f}_m]$, with:

$$\hat{f}_k(\mathbf{x}) = \sum_{j_n=1}^{N_n} \dots \sum_{j_1=1}^{N_1} w_{k, g_1, \dots, g_n} Q_{g_1, \dots, g_n}$$

for some CMAC weight \mathbf{w} . Here the weights are the sample points of the function to be approximated.

Lemma 7.1.0.2. *For any given $d > 0$ and L , the Lipschitz constant of a continuously differentiable function $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$, the maximum width of the CMAC's granule cell $\delta = \max(\|x_{i,j_{i-1}}, x_{i,j_i}\|)$ can be chosen such that:*

$$\|f(\mathbf{x}) - \hat{f}(\mathbf{x})\| \leq d \quad \text{where } \delta = \frac{d}{mL}$$

Proof: See [33]

The result from lemma 7.1.0.2 is of major importance, since it provides a design criteria for CMAC neural networks. It shows that CMAC estimates can be made arbitrary accurate.

Corollary 7.1.0.3. *Any continuously differentiable function $f(\mathbf{x})$ can be expressed as:*

$$f(\mathbf{x}) = \phi^T \mathbf{w} + d \quad \forall \mathbf{x} \in \mathbb{R}^n \quad (7.1.0.2)$$

where d is the function estimation error and $\|d\| \leq d_N$, with d_N a known bound

According to Corollary 7.1.0.3 any nonlinear function can be approximated to any required degree of accuracy using CMAC neural network. The CMAC output is:

$$\hat{f}(\mathbf{x}) = \phi^T \mathbf{w} \quad (7.1.0.3)$$

where \mathbf{w} is a matrix of ideal weights representing the function sample points. Since the function to be estimated is unknown in control applications, \mathbf{w} is also unknown and has to be leaned on-line. The difference between the ideal weights \mathbf{w} and CMAC weight estimates $\hat{\mathbf{w}}$ is expressed as:

$$\tilde{\mathbf{w}} = \mathbf{w} - \hat{\mathbf{w}}$$

7.2 System Stability Proof

In this section it is proven that the system described by equation (7.2.0.5) is Uniformly Ultimately Bounded when the weight update (5.0.2.2) and the torque (5.0.2.1) are used. The dynamics of a planar (horizontal) two-link robot manipulator can be expressed as in [8]:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \boldsymbol{\tau} \quad (7.2.0.4)$$

or equivalently:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau} \quad (7.2.0.5)$$

where $\boldsymbol{\tau} \in \mathfrak{R}^2$ contains the control torques, \mathbf{q} , $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}} \in \mathfrak{R}^2$ are vectors of joint angles, angular velocities, and angular acceleration respectively. $\mathbf{M}(\mathbf{q}) \in \mathfrak{R}^{2 \times 2}$ is the inertia matrix, and $\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathfrak{R}^2$ is the Coriolis and centripetal vector.

Given desired trajectory $\mathbf{q}_d, \dot{\mathbf{q}}_d$, the tracking errors are:

$$\mathbf{e}_1 = \mathbf{q} - \mathbf{q}_d \quad (7.2.0.6)$$

$$\mathbf{e}_2 = \dot{\mathbf{q}} - \dot{\mathbf{q}}_d \quad (7.2.0.7)$$

Taking the derivative of the tracking errors and substituting $\ddot{\mathbf{q}}$ from Eq. (7.2.0.5), the error dynamics can be expressed as:

$$\begin{aligned} \dot{\mathbf{e}}_1 &= \mathbf{e}_2 \\ \dot{\mathbf{e}}_2 &= -\mathbf{M}^{-1}(\mathbf{q})\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{M}^{-1}(\mathbf{q})\boldsymbol{\tau} - \ddot{\mathbf{q}}_d \end{aligned} \quad (7.2.0.8)$$

For the simulations conducted in this thesis, two CMAC neural networks are used; one to model the positive definite and symmetric matrix $\mathbf{M}^{-1}(\mathbf{q}) \in \mathfrak{R}^{2 \times 2}$, and one to model the vector $-\mathbf{M}^{-1}(\mathbf{q})\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})$. CMAC 1 produces three outputs estimating the terms representing the symmetric $\mathbf{M}^{-1}(\mathbf{q})$, and CMAC 2 produces two outputs approximating the terms

inside the vector $-\mathbf{M}^{-1}(\mathbf{q})\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})$. Using Corollary 7.1.0.3, the terms to be estimated can be expressed as:

$$\begin{aligned}\mathbf{M}^{-1} &= \begin{bmatrix} \phi_M^T \mathbf{w}_1 + d_1 & \phi_M^T \mathbf{w}_2 + d_2 \\ \phi_M^T \mathbf{w}_2 + d_2 & \phi_M^T \mathbf{w}_3 + d_3 \end{bmatrix} \\ &= \begin{bmatrix} \phi_M^T \mathbf{w}_1 & \phi_M^T \mathbf{w}_2 \\ \phi_M^T \mathbf{w}_2 & \phi_M^T \mathbf{w}_3 \end{bmatrix} + \begin{bmatrix} d_1 & d_2 \\ d_2 & d_3 \end{bmatrix} = \mathbf{C}_M + \mathbf{D}\end{aligned}\quad (7.2.0.9)$$

$$-\mathbf{M}^{-1}\mathbf{V} = \begin{bmatrix} \phi_F^T \mathbf{w}_4 + d_4 \\ \phi_F^T \mathbf{w}_5 + d_5 \end{bmatrix} = \begin{bmatrix} \phi_F^T \mathbf{w}_4 \\ \phi_F^T \mathbf{w}_5 \end{bmatrix} + \begin{bmatrix} d_4 \\ d_5 \end{bmatrix} = \mathbf{C}_F + \mathbf{d}\quad (7.2.0.10)$$

where:

ϕ_M^T and ϕ_F^T are the quantization functions of CMAC 1 and CMAC 2 respectively,

$\mathbf{C}_M \in \mathfrak{R}^{2 \times 2}$ is CMAC 1 model, $\mathbf{C}_F \in \mathfrak{R}^{2 \times 2}$ is CMAC 2 model, and

\mathbf{D} and \mathbf{d} contain the modeling errors for CMAC 1 and CMAC 2 respectively.

Substituting Eq. (7.2.0.9) and (7.2.0.10) into (7.2.0.8) results:

$$\dot{\mathbf{e}}_2 = \mathbf{C}_F + \mathbf{d} + \mathbf{C}_M \tau + \mathbf{D} \tau - \ddot{\mathbf{q}}_d\quad (7.2.0.11)$$

As explained previously, \mathbf{w} is a matrix of ideal weights representing the function sample points, and so \mathbf{C}_M and \mathbf{C}_F are ideal CMAC models. Since the terms to be estimated are unknown and learned on-line, the ideal weights are unknown. Let $\tilde{(\cdot)}$ and $\hat{(\cdot)}$ indicate the error and estimate respectively, then $\mathbf{w} = \hat{\mathbf{w}} + \tilde{\mathbf{w}}$ and so:

$$\mathbf{C}_M = \begin{bmatrix} \phi_M^T \mathbf{w}_1 & \phi_M^T \mathbf{w}_2 \\ \phi_M^T \mathbf{w}_2 & \phi_M^T \mathbf{w}_3 \end{bmatrix} = \begin{bmatrix} \phi_M^T \hat{\mathbf{w}}_1 & \phi_M^T \hat{\mathbf{w}}_2 \\ \phi_M^T \hat{\mathbf{w}}_2 & \phi_M^T \hat{\mathbf{w}}_3 \end{bmatrix} + \begin{bmatrix} \phi_M^T \tilde{\mathbf{w}}_1 & \phi_M^T \tilde{\mathbf{w}}_2 \\ \phi_M^T \tilde{\mathbf{w}}_2 & \phi_M^T \tilde{\mathbf{w}}_3 \end{bmatrix} = \hat{\mathbf{C}}_M + \tilde{\mathbf{C}}_M \quad (7.2.0.12)$$

$$\mathbf{C}_F = \begin{bmatrix} \phi_F^T \mathbf{w}_4 \\ \phi_F^T \mathbf{w}_5 \end{bmatrix} = \begin{bmatrix} \phi_F^T \hat{\mathbf{w}}_4 \\ \phi_F^T \hat{\mathbf{w}}_5 \end{bmatrix} + \begin{bmatrix} \phi_F^T \tilde{\mathbf{w}}_4 \\ \phi_F^T \tilde{\mathbf{w}}_5 \end{bmatrix} = \hat{\mathbf{C}}_F + \tilde{\mathbf{C}}_F \quad (7.2.0.13)$$

Substituting Eq. (7.2.0.12), and Eq. (7.2.0.13) into Eq. (7.2.0.11) gives:

$$\dot{\mathbf{e}}_2 = \hat{\mathbf{C}}_F + \tilde{\mathbf{C}}_F + \mathbf{d} + \hat{\mathbf{C}}_M \tau + \tilde{\mathbf{C}}_M \tau + \mathbf{D} \tau - \ddot{\mathbf{q}}_d \quad (7.2.0.14)$$

The control term is chosen to be:

$$\begin{aligned} \tau &= \hat{\mathbf{C}}_M^{-1} (-\hat{\mathbf{C}}_F + \ddot{\mathbf{q}}_d - G_1 \mathbf{e}_1 - G_2 \mathbf{e}_2 + \mathbf{u}_{ND}) \\ &= \hat{\mathbf{C}}_M^{-1} (\mathbf{u}_w + \mathbf{u}_{ND}) \end{aligned} \quad (7.2.0.15)$$

where \mathbf{G}_1 and \mathbf{G}_2 are positive-definite control gain matrices, and \mathbf{u}_{ND} is a robusting term that will be evaluated using nonlinear damping.

Substituting Eq. (7.2.0.15) into (7.2.0.14) gives:

$$\dot{\mathbf{e}}_2 = -G_1 \mathbf{e}_1 - G_2 \mathbf{e}_2 + \tilde{\mathbf{C}}_F + \tilde{\mathbf{C}}_M \tau + \mathbf{u}_{ND} + \mathbf{d} + \mathbf{D} \tau \quad (7.2.0.16)$$

Using an error vector \mathbf{z} :

$$\mathbf{z} = \begin{bmatrix} \mathbf{e}_1(t) \\ \mathbf{e}_2(t) \end{bmatrix} \quad (7.2.0.17)$$

the tracking errors \mathbf{e}_1 and \mathbf{e}_2 can be written as:

$$\dot{\mathbf{z}} = \mathbf{A} \mathbf{z} + \mathbf{B} [\tilde{\mathbf{C}}_F + \tilde{\mathbf{C}}_M \tau + \mathbf{u}_{ND} + \mathbf{d} + \mathbf{D} \tau] \quad (7.2.0.18)$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{G}_1 & \mathbf{G}_2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$

$\mathbf{0} \in \mathbb{R}^{2 \times 2}$ is a matrix of zeros and $\mathbf{1} \in \mathbb{R}^{2 \times 2}$ is an identity matrix

Now consider the Lyapunov candidate:

$$V = \mathbf{z}^T \mathbf{P} \mathbf{z} + \sum_{i=1}^5 \frac{1}{2\beta_i} \tilde{\mathbf{w}}_i^T \tilde{\mathbf{w}}_i \quad (7.2.0.19)$$

where $\tilde{\mathbf{w}} = \mathbf{w} - \hat{\mathbf{w}}$, β_i is a positive constant, and $\mathbf{P} = \mathbf{P}^T > 0$ is the solution of the Lyapunov equation:

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

\mathbf{Q} is a positive definite matrix, typically chosen as identity matrix.

Taking the derivative of the Lyapunov candidate Eq. (7.2.0.19) and substituting $\dot{\mathbf{z}}$ from Eq. (7.2.0.18) gives:

$$\begin{aligned} \dot{V} &= \mathbf{z}^T (\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A}) \mathbf{z} + \mathbf{z}^T \mathbf{P} \mathbf{B} [\tilde{\mathbf{C}}_F + \tilde{\mathbf{C}}_{M^T} + \mathbf{u}_{ND} + \mathbf{D}^T + \mathbf{d}] + \sum_{i=1}^5 \frac{1}{\beta_i} \tilde{\mathbf{w}}_i^T \dot{\tilde{\mathbf{w}}}_i \\ \dot{V} &= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + \mathbf{z}^T \mathbf{P} \mathbf{B} [\tilde{\mathbf{C}}_F + \tilde{\mathbf{C}}_{M^T}] + \sum_{i=1}^5 \frac{1}{\beta_i} \tilde{\mathbf{w}}_i^T \dot{\tilde{\mathbf{w}}}_i + \zeta \end{aligned} \quad (7.2.0.20)$$

$$\text{where } \zeta = \mathbf{z}^T \mathbf{P} \mathbf{B} [\mathbf{D}^T + \mathbf{d} + \mathbf{u}_{ND}]$$

Substituting the matrices of $\tilde{\mathbf{C}}_F$, $\tilde{\mathbf{C}}_M$ into Eq. (7.2.0.20) and letting $\mathbf{z}^T \mathbf{P} \mathbf{B} = [\epsilon_4 \quad \epsilon_5]$

gives:

$$\begin{aligned}
\dot{V} &= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + [\epsilon_4 \quad \epsilon_5] \left[\begin{array}{c} \left[\begin{array}{c} \phi_F^T \tilde{\mathbf{w}}_4 \\ \phi_F^T \tilde{\mathbf{w}}_5 \end{array} \right] \\ \left[\begin{array}{c} \phi_M^T \tilde{\mathbf{w}}_1 \tau_1 + \phi_M^T \tilde{\mathbf{w}}_2 \tau_2 \\ \phi_M^T \tilde{\mathbf{w}}_2 \tau_1 + \phi_M^T \tilde{\mathbf{w}}_3 \tau_2 \end{array} \right] \end{array} \right] + \sum_{i=1}^5 \frac{1}{\beta_i} \tilde{\mathbf{w}}_i^T \dot{\tilde{\mathbf{w}}}_i + \zeta \\
&= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + \tilde{\mathbf{w}}_4^T \phi_F \epsilon_4 + \tilde{\mathbf{w}}_5^T \phi_F \epsilon_5 + \tilde{\mathbf{w}}_1^T \phi_M \epsilon_4 \tau_1 + \tilde{\mathbf{w}}_2^T \phi_M \epsilon_4 \tau_2 + \tilde{\mathbf{w}}_2^T \phi_M \epsilon_5 \tau_1 + \tilde{\mathbf{w}}_3^T \phi_M \epsilon_5 \tau_2 \\
&+ \sum_{i=1}^5 \frac{1}{\beta_i} \tilde{\mathbf{w}}_i^T \dot{\tilde{\mathbf{w}}}_i + \zeta \tag{7.2.0.21}
\end{aligned}$$

Letting $\tau \mathbf{z}^T \mathbf{P} \mathbf{B} = \begin{bmatrix} \epsilon_1 & \epsilon_{12} \\ \epsilon_{21} & \epsilon_3 \end{bmatrix}$, and $\epsilon_2 = \epsilon_{12} + \epsilon_{21}$, the Eq. (7.2.0.21) can be rewritten as:

$$\begin{aligned}
\dot{V} &= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + \tilde{\mathbf{w}}_4^T \phi_F \epsilon_4 + \tilde{\mathbf{w}}_5^T \phi_F \epsilon_5 + \tilde{\mathbf{w}}_1^T \phi_M \epsilon_1 + \tilde{\mathbf{w}}_2^T \phi_M \epsilon_2 + \tilde{\mathbf{w}}_3^T \phi_M \epsilon_3 - \sum_{i=1}^5 \frac{1}{\beta_i} \tilde{\mathbf{w}}_i^T \dot{\tilde{\mathbf{w}}}_i + \zeta \\
&= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + \sum_{i=1}^5 \tilde{\mathbf{w}}_i^T (\eta_i - \beta_i^{-1} \dot{\tilde{\mathbf{w}}}_i) + \zeta \tag{7.2.0.22}
\end{aligned}$$

where $\eta_{1,2,3} = \phi_M \epsilon_{1,2,3}$, and $\eta_{4,5} = \phi_F \epsilon_{4,5}$

Choose the weight update to be

$$\dot{\hat{\mathbf{w}}}_i = \beta_i (\eta_i - v_i \hat{\mathbf{w}}_i + \gamma_i \phi(g_i - \phi^T \hat{\mathbf{w}}_i))$$

$$\phi = \phi_M \quad i = 1, 2, 3 \quad \text{and} \quad \phi = \phi_F \quad i = 4, 5$$

where $g_{1,2,3}$ and $g_{4,5}$ are model estimate of \mathbf{M}^{-1} and $\mathbf{M}^{-1} \mathbf{V}$ respectively.

Let:

$$L = \begin{bmatrix} v_1 & & & \mathbf{0} \\ & v_2 & & \\ & & \ddots & \\ \mathbf{0} & & & v_5 \end{bmatrix}, \quad S = \begin{bmatrix} C_1 & & & \mathbf{0} \\ & C_2 & & \\ & & \ddots & \\ \mathbf{0} & & & C_5 \end{bmatrix} \quad C_i = \gamma_i \phi \phi^T I_{n \times n}$$

$$w = [w_1^T \cdots w_5^T]^T \quad k = [k_1^T \cdots k_5^T]^T, \quad \text{where } k_i = \gamma_i \phi g_i \quad i = 1, 2, \dots, 5$$

Substitute the weight update into (7.2.0.22) gives:

$$\begin{aligned} \dot{V} &= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + \tilde{w}^T L \hat{w} - \tilde{w}^T k + \tilde{w}^T S \hat{w} + \mathbf{z}^T \mathbf{P} \mathbf{B} [\mathbf{D} \tau + \mathbf{d} + \mathbf{u}_{ND}] \\ \dot{V} &= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + \tilde{w}^T L w - \tilde{w}^T L \tilde{w} - \tilde{w}^T k + \tilde{w}^T S w - \tilde{w}^T S \tilde{w} \\ &\quad + \mathbf{z}^T \mathbf{P} \mathbf{B} \mathbf{D} \hat{\mathbf{C}}_M^{-1} \mathbf{u}_w + \mathbf{z}^T \mathbf{P} \mathbf{B} \mathbf{D} \hat{\mathbf{C}}_M^{-1} \mathbf{u}_{ND} + \mathbf{z}^T \mathbf{P} \mathbf{B} \mathbf{d} + \mathbf{z}^T \mathbf{P} \mathbf{B} \mathbf{u}_{ND} \end{aligned} \quad (7.2.0.23)$$

let

$$\mathbf{u}_{ND} = -\alpha \|\mathbf{u}_w\|^2 \mathbf{B}^T \mathbf{P}^T \mathbf{z}$$

Substituting \mathbf{u}_{ND} into (7.2.0.23) gives:

$$\begin{aligned} \dot{V} &= -\mathbf{z}^T \mathbf{Q} \mathbf{z} + \mathbf{z}^T \mathbf{P} \mathbf{B} \mathbf{d} - \alpha \|\mathbf{u}_w\|^2 \mathbf{z}^T \mathbf{P} \mathbf{B} (\mathbf{D} \hat{\mathbf{C}}_M^{-1} + I) \mathbf{B}^T \mathbf{P}^T \mathbf{z} + \mathbf{z}^T \mathbf{P} \mathbf{B} \mathbf{D} \hat{\mathbf{C}}_M^{-1} \mathbf{u}_w \\ &\quad + \tilde{w}^T L w - \tilde{w}^T L \tilde{w} - \tilde{w}^T k + \tilde{w}^T S w - \tilde{w}^T S \tilde{w} \end{aligned} \quad (7.2.0.24)$$

As previously mentioned, \mathbf{D} and \mathbf{d} contain the approximation errors of CMAC 1 and CMAC 2 respectively. According to lemma (7.1.0.2) CMAC model can be made arbitrary accurate and the approximation errors are bounded and known. Let $\lambda_{max}(D) \leq d_{1max}$, and $\|\mathbf{d}\| \leq d_{2max}$, then the upper bound of the derivative of the Lyapunov function can be

expressed as:

$$\begin{aligned}\dot{V} &\leq -\lambda_{\min}(Q)\|z\|^2 + \|\mathbf{z}^T \mathbf{PB}\|d_{2max} \\ &\quad - \alpha\|u_w\|^2\lambda_{\min}(D\hat{\mathbf{C}}_M^{-1} + I)\|\mathbf{z}^T \mathbf{PB}\|^2 + \|\mathbf{z}^T \mathbf{PB}\|\|u_w\|\lambda_{\max}(D\hat{\mathbf{C}}_M^{-1}) \\ &\quad - \|\tilde{w}\|^2(\lambda_{\min}(L) + \lambda_{\min}(S)) + \|\tilde{w}\|(\|w\|(\lambda_{\min}(L) + \lambda_{\max}(S)) - \|k\|)\end{aligned}$$

The region of convergence can then be found by completing the square:

$$\begin{aligned}\dot{V} &\leq -\lambda_{\min}(Q)\|z\|^2 \\ &\quad - \alpha\|u_w\|^2\lambda_{\min}(D\hat{\mathbf{C}}_M^{-1} + I) \left(\|\mathbf{z}^T \mathbf{PB}\| - \frac{\lambda_{\max}(D\hat{\mathbf{C}}_M^{-1})\|u_w\| + d_{2max}}{2\alpha\|u_w\|^2\lambda_{\min}(D\hat{\mathbf{C}}_M^{-1} + I)} \right)^2 \\ &\quad + \frac{(\lambda_{\max}(D\hat{\mathbf{C}}_M^{-1})\|u_w\| + d_{2max})^2}{4\alpha\|u_w\|^2\lambda_{\min}(D\hat{\mathbf{C}}_M^{-1} + I)} \\ &\quad - (\lambda_{\min}(L) + \lambda_{\min}(S)) \left(\|\tilde{w}\| - \frac{\|w\|(\lambda_{\max}(L) + \lambda_{\max}(S)) - \|k\|}{2(\lambda_{\min}(L) + \lambda_{\min}(S))} \right)^2 \\ &\quad + \frac{(\|w\|(\lambda_{\max}(L) + \lambda_{\max}(S)) - \|k\|)^2}{4(\lambda_{\min}(L) + \lambda_{\min}(S))}\end{aligned}$$

Matrices Q, S, L are positive definite. Thus, the only condition required for \dot{V} to be UUB is that the matrix $(D\hat{\mathbf{C}}_M^{-1} + I)$ be positive definite. The following projection rule is used to make sure this condition is satisfied:

$$\dot{\hat{\mathbf{w}}}_i = \begin{cases} \beta_i(\eta_i - v_i\hat{\mathbf{w}}_i + \gamma_i\Gamma(g_i - \Gamma^T\hat{\mathbf{w}}_i)) & \text{if } \lambda_{\min}(D\hat{\mathbf{C}}_M^{-1} + I) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Chapter 8

Conclusions

In this chapter, a summary of simulation results for each feedforward approach along with its advantages and disadvantages is emphasized. Then, the comparison result between the proposed and computed torque scheme is presented.

8.1 Results

In this thesis, a new scheme, to the best of author's knowledge, was proposed to improve the performance of the conventional computed torque scheme. Two CMAC neural networks were used on-line in a feedforward loop to model \mathbf{M}^{-1} and $\mathbf{M}^{-1}\mathbf{V}$ and compensate for structured and unstructured uncertainties. Through simulation, it was shown that CMAC neural networks are capable of estimating the forward dynamics of a two-link robotic manipulator that implements a computed torque controller. The novel scheme of this thesis was proposed by investigating and comparing three different forward dynamic approaches. The difficulty to follow the desired trajectories is demonstrated through simulation for two of the approaches.

The first approach uses state error to update the CMAC weights. This approach failed to learn the real model of the system. The performance was unpredictable, and the RMS error of the angles and angular velocities did not improve or converge with the number of iterations. In fact, the system's stability was only possible with very slow updates of the CMAC weights.

In the second approach, the weights of both CMACs are updated based on the model estimates of the system. Through simulation, it was illustrated that the CMAC weights used for this approach converge to their actual values and CMAC is capable of learning the real model of the system. It was also shown that the leakage term is not required for the stability when the learning term is present. Instead of forcing the weights to zero as conducted using the leakage term, they are guided to their actual values using the learning term. Since both CMACs are updated based only on the modeling error, the tracking error and uncertainties are ignored. Although no significant improvement is seen in the tracking error, it converges to a specific value and the system is clearly stable.

The last and the novel forward dynamic approach presented in this work utilizes one neural network to learn the real model of the inertia matrix, and one to focus on the tracking error and unstructured uncertainties of the system. Through simulation, it was illustrated that a close estimate of $\mathbf{M}(\mathbf{q})$ eliminates unpredictable changes of the control torque seen in approach one. Also, updating the weights of a second CMAC using the tracking error will significantly improve the controller's performance. The RMS error, resulting from simulation of this approach, decreases with the number of iterations and converges to a specific value. The convergence happens in 20 iterations and is to a small value which shows close tracking of the trajectories. The comparison of this approach with previous ones shows that this approach follows the desired trajectories more closely and requires less torque to do so.

To illustrate the advantages of the proposed scheme, it is compared with the conventional computed torque controller. The simulation results show that both methods perform well when real model of the system is used to cancel the nonlinearities and enough feedback gain is applied. However in less than ideal cases, the proposed method outperforms

the conventional computed torque. It is shown that when the feedback gain is small, even with application of the real model of the system, computed torque cannot closely follow the desired trajectories. In the same situation, the proposed method significantly improves the performance and requires less initial torque to do so. Also, the proposed scheme was shown to be more receptive to payload changes. Simulations were conducted with added payloads up to 2 Kg, where the proposed scheme showed robustness to the added payloads.

8.2 Application and Limitations

The added complexity of modeling the system's dynamics may be highly desirable in certain situations. In particular, the advantages include having an acceleration estimate provided by the CMAC, ability to use on-line system identification techniques to help train the CMAC, suitability for use in adaptive flexible-joint robot control using backstepping with tuning functions [15], and ability to quantify performance using a known Lyapunov function. Furthermore, an effective technique for estimating an inertia matrix is a prerequisite to the solution of adaptive trajectory tracking for flexible-link manipulator arms.

Although the proposed method results in improved performance over conventional computed torque method and compensates for the structured and unstructured uncertainties, its application is limited to cases where some knowledge of the system is given. Updating the CMAC neural networks based only on the tracking error, as illustrated in approach one, results in an unpredictable performance where the tracking RMS error does not converge.

List of References

- [1] J. S. Albus, "A new approach to manipulator control: the cerebellar model articulation controller (CMAC)," *J. Dyn. Syst. Meas., Contr.*, vol. 97, no. 3, pp. 220–227, Sept. 1975.
- [2] J. Craig, P. Hsu, and S. Sastry, "Adaptive control of mechanical manipulators," *Proc. IEEE Int. Conf., Robotics and Automation*, vol. 3, pp. 190 – 195, Apr 1986.
- [3] M. Teshnehlab and K. Watanabe, "Self tuning of computed torque gains by using neural networks with flexible structures," *Proc. IEE, Control Theory and Applications*, vol. 141, no. 4, pp. 235–242, Jul 1994.
- [4] G. Feng, "A new stable tracking control scheme for robotic manipulators," *IEEE Trans., Systems, Man and Cybernetics, Part B*, vol. 27, no. 3, pp. 510–516, Jun 1997.
- [5] Q. H. M. Meng, "Comparison study of model-based and non-model-based robot controllers," *IEEE Int. conf., Systems, Man and Cybernetics*, vol. 1, pp. 61 – 66, Oct. 1995.
- [6] S. Jung and T. Hsia, "A new neural network control technique for robot manipulators," *proc. conf., American Control*, vol. 1, pp. 878 – 882, June 1995.
- [7] T. Ozaki, T. Suzuki, T. Furuhashi, S. Okuma, and Y. Uchikawa, "Trajectory control of robotic manipulators using neural networks," *IEEE Trans., Industrial Electronics*, vol. 38, no. 3, pp. 195 – 202, June 1991.

- [8] M. Spong and M. Vidyasagar, *Robot Dynamics and Control*. New York: John Wiley and Sons, 1989.
- [9] J. Yegerlehner and P. Meckl, "Experimental implementation of neural network controller for robot undergoing large payload changes," *Proc. of IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 744 – 749, 1993.
- [10] P. Meckl and N. Nho, "Intelligent feedforward control and payload estimation for a two-link robotic manipulator," *IEEE / ASME Trans. Mechatronics*, vol. 8, pp. 277–282, June 2003.
- [11] L. Peng and W. Peng-Yung, "Neural-fuzzy control system for robotic manipulators," *IEEE Control Systems Magazine*, vol. 22, pp. 53 – 63, Feb. 2002.
- [12] T. Lee, S. Kumarawadu, and J. Perng, "Direct-adaptive neurocontrol of robots with unknown nonlinearities and velocity feedback," *IEEE Int. Conf. on Systems, Man and Cybernetics*, vol. 3, pp. 2073 – 2077, Oct. 2005.
- [13] M. Meng and X. Yang, "Real-time tracking control of robot manipulators with on-line learning based approach," *IEEE Canadian Conf. on Electrical and Computer Engineering*, vol. 2, pp. 1035 – 1040, May 1999.
- [14] C. Macnab, "Getting weights to behave themselves: achieving stability and performance in neural-adaptive control when inputs oscillate," *Proc. American Control Conference*, vol. 5, pp. 3192 – 3197, June 2005.
- [15] C. Macnab, G. D'Eleuterio, and M. Meng, "CMAC adaptive control of flexible-joint robots using backstepping with tuning functions," *Proc. ICRA, Robotics and*

- Automation*, vol. 3, pp. 2679 – 2686, Apr 26-May 1, 2004.
- [16] C. Macnab and G. D’Eleuterio, “Stable, online learning using cmacs for neuroadaptive tracking control of flexible-joint manipulators,” *Proc. IEEE Int. Conf., Robotics and Automation*, vol. 1, pp. 511 – 517, May 1998.
- [17] T. Tao, S. Su, and T. Hung, “Credit assigned cmac and its application to online learning robust controllers,” *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 33, no. 2, pp. 202 – 213, April 2003.
- [18] H. R. Lai and C. C. Wong, “A fuzzy CMAC structure and learning method for function approximation,” *10th IEEE Int. Fuzzy Systems*, vol. 1, pp. 436 – 439, Dec. 2001.
- [19] I. W.T. Miller, R. Hewes, F. Glanz, and I. L.G. Kraft, “Real-time dynamic control of an industrial manipulator using a neural network-based learning controller,” *IEEE Trans., Robotics and Automation*, vol. 6, no. 1, pp. 1 – 9, Feb. 1990.
- [20] W. Y. Wang, C. Y. Cheng, and Y. G. Leu, “An online GA-based output-feedback direct adaptive fuzzy-neural controller for uncertain nonlinear systems,” *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 34, no. 1, pp. 334 – 345, Feb. 2004.
- [21] R. Ortega and M. Spong, “Adaptive motion control of rigid robots,” *Proc. IEEE Conf., Decision and Control*, vol. 2, pp. 1575–1584, Dec 1988.
- [22] C. Y. Su and T. P. Leung, “A sliding mode controller with bound estimation for robot manipulators,” *IEEE Trans., Robotics and Automation*, vol. 9, no. 2, pp. 208–214, Apr 1993.

- [23] M. Leahy, M. Johnson, and S. Rogers, "Neural network payload estimation for adaptive robot control," *IEEE Int. Neural Networks*, vol. 2, no. 1, pp. 93 – 100, Jan. 1991.
- [24] E. Kim, "Output feedback tracking control of robot manipulators with model uncertainty via adaptive fuzzy logic," *IEEE Int., Fuzzy Systems*, vol. 12, no. 3, pp. 368 – 378, June 2004.
- [25] M. Johnson and M. Leahy, "Adaptive model-based neural network control: validation and analysis," *Proc. IEEE Int. Symposium, Intelligent Control*, vol. 1, pp. 486 – 491, Sept. 1990.
- [26] L. Behera, S. Chaudhury, and M. Gopal, "Neuro-adaptive hybrid controller for robot-manipulator tracking control," *Proc. IEE, Control Theory and Applications*, vol. 143, no. 3, pp. 270 – 275, May 1996.
- [27] C. Kwan and F. Lewis, "Robust backstepping control of nonlinear systems using neural networks," *IEEE Trans., Systems, Man and Cybernetics*, vol. 30, no. 6, pp. 753 – 766, Nov. 2000.
- [28] S. Commuri and F. Lewis, "Control of unknown nonlinear dynamical systems using cmac neural networks: structure, stability, and passivity," *Proc. IEEE Int. Symposium, Intelligent Control*, vol. 30, no. 6, pp. 123–129, Aug 1995.
- [29] F. C. Chen and C. C. Liu, "Adaptively controlling nonlinear continuous-time systems using multilayer neural networks," *IEEE Trans., Automatic Control*, vol. 39, no. 6, pp. 1306 – 1310, June 1994.

- [30] E. Khan and T. Ogunfunmi, "A multilayered neural net controller for servo systems," *IEEE Int. Conf., Neural Networks*, vol. 2, pp. 1577 – 1581, Nov. 1991.
- [31] S. Liuzzo and P. Tomei, "A global adaptive learning control for robotic manipulators," *IEEE Conf., Decision and Control*, pp. 3596 – 3601, Dec. 2005.
- [32] P. M. Prenter, *Splines and variational Methods*. New York: John Wiley and Sons, 1975.
- [33] S. Commuri and F. L. Lewis, "CMAC neural networks for control of nonlinear dynamical systems: Structure, stability and passivity," *Automatica*, vol. 33, no. 4, pp. 635 – 641, 1997.