Neural Network Based Modeling and Control of a Flexible-Link Manipulator

Aloke Chaudhuri

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

August 1994

ISBN 0-315-97635-7

ABSTRACT

Neural Network Based Modeling and Control
of a Flexible-Link Manipulator

Aloke Chaudhuri

Controlling the motion of a flexible-link manipulator has been an ongoing concern in recent years. This research work is aimed at developing a neural network based strategy to solve the problem of tip-position control for a single flexible-link manipulator. The proposed controller uses a partitioned strategy, wherein the inner loop stabilizes the plant, and the outer loop (servo portion) provides set-point tracking. A backpropagation network has been trained off-line to accurately identify the unmodeled and/or inaccurately modeled dynamics present in an actual manipulator, and then applied in the inner loop of the closed-loop system to compensate for these dynamics. A feed-through compensator has been designed following the method of transmission zero assignment, and used in the inner loop to ensure closed-loop stability of this nonminimum phase system. The servo loop employs a proportional plus integral control strategy to track a desired trajectory in two-dimensional space. In addition, a robust servo controller has been designed using the internal model principle, and its performance has been compared with that of the PI type controller mentioned above.

To my parents — whose smiles have always been a curve that set a lot of things straight in my life.

ACKNOWLEDGEMENTS

I wish to express my profound thanks to Dr. R.V. Patel for his invaluable guidance and help at every stage of this research. I am also grateful to my friend Mehrdad Moallem for his comments and suggestions.

"It's strange that how much you've got to know before you know how little you know".

– ***Anonymous***

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Preamble

Most of the industrial robots built to date are made of substantially stiff materials to minimize vibration, and need relatively simple control strategies for trajectory tracking. However, every real manipulator has some degree of both joint and link flexibilities which are neglected in most of these rigid-link robots. In certain cases, this assumpton of rigidity leads to inaccurate control and may even cause system instability.

In recent years, improvements in electric motor technology coupled with new designs, such as direct drive arms, have led to a rapid increase in the speed and load carrying capabilities of manipulators. Consequently, this has heightened the importance of considering the effects of flexibility of the nominally rigid links. Present generation manipulators are limited to a load carrying capability of typically 5-10% of their own weight by the requirement of rigidity. For example, the Cincinatti-Milicron T3R3 robot weighs more than 1800 kg., but has a maximum payload capacity of 23 kg. [12]. This level of performance, as indicated by the ratio of payload to arm weight, is completely unacceptable in many applications, such as space where economy and energy considerations strongly encourage light weight designs. In order to respond to this special demand, most of the robotic manipulators and tele-robotic motion systems in space are constructed of light-weight materials with inherent link-flexibility.

As indicated in [12], there are a number of potential advantages arising from the use of flexible links.

- **Faster Operation:** For a given load, a large increase in speed would be possible.

- **Lower Energy Consumption:** Lighter links require less power to produce the same accleration as compared to rigid links with the same payload lifting capacity. This reduced power requirement makes it possible to use smaller and cheaper actuators.

- **Safer Operation:** Should it collide with an obstacle, less damage would be caused due to reduced inertia.

- **Compliant Structure:** By introducing mechanical compliance into the manipulator structure, flexible links come in handy for delicate assembly operations where the links themselves can be used for force/torque sensing.

- **Possible Elimination of Gearing:** This is becoming increasingly relevant with the development of motors with high power/weight ratio, and indicates the possibility of reduced cost, reduced backlash and improved actuator linearity.

- **Less Bulky Design**

- **Lowered Mounting Strength and Rigidity Requirements:** This is relevant particularly to gantry and wall-mounted robots.

However, looking through the kaleidoscope of future directions in the realm of light-weight, state-of-the-art flexible-link manipulators, one also notices the inherent complexity involved. The problem of controlling the end-point of a flexible-link manipulator is considerably difficult for a number of reasons. The problem of **model truncation**, which arises due to a finite-dimensional representation of a distributed parameter system, causes some unmodeled dynamics to be present in the mathematical model of the manipulator. Using a reduced-order model for the controller design may also lead to the phenomena of **control and observation spillover**.[Control spillover is an excitation of the residual modes [1] by the control action, and observation spillover is contamination of sensor readings by the residual modes]. When both control and observation spillover are present, the closed-loop system may become unstable. Further complications arise because of the highly nonlinear nature of the system and the difficulty involved in accurate modeling of various friction and backlash terms. In addition, the system is non-minimum phase, and application of constant output feedback leads to closed-loop instability for a moderate gain. Further, the interaction between the gross dynamics and the deformational dynamics of the links may cause problems: The change of an arm configuration leads to a change in the spatial boundary conditions of the links, which, in effect, modifies their characteristic frequencies and modes. Picking up a load similarly leads to a change in the natural modes. This change has a serious degrading effect on the performance of controllers.

In face of all the difficulties outlined above, one has to determine a control objective prior to proceeding further to find any specific strategy. It is generally accepted in the literature that the control scheme of a flexible-link manipulator should satisfy all of the following criteria:

- It should enable the manipulator to track a reference trajectory with rea-

[1]Modes which are not present in the reduced order model

sonable accuracy.

- It should ensure closed-loop stability and reduce the elastic arm deflections as much as possible.

- It should be robust to changes in the manipulator behavior due to changing configuration or loading.

- The algorithm should be computationally efficient so that real-time implementation becomes feasible, and it should run fast enough to control rapid motions of the arm.

- It should be capable of rejecting disturbances.

Many researchers have tried to explore different control strategies to satisfy all or some of the above mentioned criteria. Before getting into those schemes, we will have a brief review of current strategies for modeling the dynamic behavior of flexible-link manipulators.

1.2 Dynamics of Flexible Manipulators

The investigation of methodologies to model the dynamics of flexible-link manipulators ranges from single-link arms rotating about a fixed axis [7, 15] to three-dimensional multi-link arms [3, 29]. But we will primarily focus on single-link arms, because with the increase of the number of links, the mathematics gets considerably involved.

The various methods of modeling can be broadly categorized as follows [12].

1.2.1 Lagrange's Equation and Modal Expansion

This is one of the most frequently used methods found in literature [2, 4, 15, 19, 44]. In this approach the deflection of the arm is represented as a sum of the product of two terms, one a function of the distance along the beam, and the other a function of time.

$$w(x,t) = \sum_{i=1}^{n} q_i(t)\phi_i(x)$$

As indicated in [26], various mode shapes $\phi_i(x)$ can be chosen to represent the dynamic behavior of the manipulator, provided they all satisfy certain boundary and orthogonality conditions. Hastings [15] used this method to derive a state-space model of a single flexible link, and observed the relative merits of two sets of assumed modes— pinned-mass and clamped-mass. Comparing the frequency response of an experimental set-up with his theoretical model, he noticed good agreement for the zeros but less accuracy for the poles. This was attributed to unmodeled frictional effects.

In general, it has been noticed that this method is computationally less efficient compared to Newton-Euler approaches. Another major drawback of this scheme is the inaccuracy of simple modal modeling at high frequency. When the wavelengths of vibration are comparable to the cross-sectional dimensions of the beam, Euler-Bernoulli beam theory fails to represent the arm dynamics appropriately and results in error. However, the advantage of this method lies in the ease with which one can reformulate the dynamic model to a state-space form, which is used almost universally as the starting point for control system design.

1.2.2 Lagrange's Equation and Finite Elements

The Lagrangian finite element method is similar to the assumed modes approach. The elastic deflection of any point along the arm is expressed as a set of generalized coordinates which are subsequently used to derive the expression of total kinetic and potential energy of the beam. These are then substituted in Lagrange's equation and solved to find the time dependency of the generalized coordinates and hence the link. Usoro et. al. [46] have used this method to model a single flexible link. By formulating a Generalized Inertia Matrix (G.I.M.), they have been able to express concisely both the rigid and elastic inertial effects. Sunada and Dubowsky [43] have used the same method to model a Cincinatti T3R3 industrial robot. They used the finite element approach for individual links and later combined the results with transformation matrices to form an overall dynamic model. Cyril et. al. [11] have used a similar approach for dynamic modeling.

1.2.3 Newton-Euler Equation and Modal Expansion

In this method, Newton's second law is applied to balance the rate of change of angular momenta with the applied forces. Rakhsha and Goldenberg [36] adapted this method for modeling a single-link arm with tip mass. The obvious demerits of this method are the algebraical complexity of the Newton-Euler formulation and the approximate nature of the constrained modes. The latter becomes significant when truncated models are used with a smaller number of modes, because the actual modes are assumed to be summations of an infinite number of constrained modes. This problem is commmon to both Newton-Euler and Lagrangian methods because they do not use the natural modes of the system directly.

1.2.4 Newton-Euler Equation and Finite Elements

The essence of this method is to divide the arm into a number of small segments and solve the Newton-Euler equations for each one of them. Nagathan and Soni [29] have recently used this method for dynamic modeling. The apparent shortcoming of this method is the algebraic complexity which increases with the number of elements.

1.2.5 Singular Perturbation Analysis

Lately, this method has been examined by a number of researchers [10, 39, 44] as a suitable tool to reformulate an existing dynamic model for facilitating controller design. The underlying assumption in this theory is that the modes of the flexible manipulator can be divided into low frequency rigid-body modes or *slow modes* and high frequency flexible modes or *fast modes*. After this partitioning is done and the resulting subsystems are expressed in state-space form, the control input is assumed to be a summation of two terms— the *slow control signal* responsible for the gross motion of the arm, and the *fast control signal* to stabilize the vibrations. This makes the design considerably simplified. However, the assumption of partitioning is believed to be unjustified in many cases, specially for high speed motions of flexible-link manipulators.

1.2.6 Frequency Domain Methods

Instead of modeling the link behavior in time domain, some researchers have tried to develop a model in frequency domain [1, 5]. A model worth mentioning in this context is due to Book [5], where starting from the Euler-Bernoulli beam equation, a transformation has been made to express the resulting equations in an elegant form in frequency domain. This model, however, does not take into account the effect of interaction between the gross motion of the arm and the flexible dynamics.

Further, an inverse Laplace or Fourier transform is needed every time one wishes to observe time domain behavior, and this is computationally expensive for real-time applications.

1.3 Control of Flexible-link Manipulators

The various control strategies for flexible-link manipulators can be broadly classified into a number of categories in the following manner.

1.3.1 Non-adaptive Control

There are two broad areas of control that fall in this category.

Classical Methods

Book et. al. [4] compared the control schemes for a two-link arm in three different ways, viz. **Independent Joint Control** (assuming rigid links), **General Rigid Control** (assuming rigid links with dynamic interaction between them), and **Flexible Feedback Control** (employing a feedback scheme for flexible states and joint variables). Essentially, they all used PD feedback to achieve desired closed-loop pole locations based on a state-space model. However, these type of controllers were not able to produce satisfactory results when sufficient link flexibility was present.

Schmitz [38] used optical sensing of the end-point to achieve increased bandwidth for the closed-loop system and greater end-point disturbance rejection. However, since the system was nonminimum phase, gain margin was very low for the controller and a 60% increase in gain led to instability.

Optimal Control

In optimal control schemes, desired pole placement is achieved through a suitably designed state feedback regulator whose gains are selected by optimizing a chosen cost function. Karkkainen [20] used a combined inverse dynamics/optimal regulator control scheme to control a forestry manipulator. Nominal control was provided by inverse dynamics based on an assumed rigid link, and flexure compensation was provided by an optimal control scheme. In [45], a prescribed degree of stability was introduced in the closed-loop system by modifying the cost function. Schn tz [38] used a low-order compensator to avoid the complexity involved in full state feedback, and was able to achieve good control in the absence of modeling errors. However, the performance deteriorated with the change of tip mass and called for a more robust controller.

In conclusion, fixed parameter controllers, based on a linear, time-invariant model of a flexible manipulator, are unable to perform satisfactorily in the presence of modeling errors which arise from configuration and load changes of the arm.

Adaptive controllers can solve these problems to some extent and are discussed in the next subsection.

1.3.2 Adaptive Control

The adaptive control methods can be classified into the following groups.

Model Reference Adaptive Control (MRAC)

In MRAC, a control input is generated to force the system to behave in a prescribed manner specified by a reference model (usually, a stable, linear, time-invariant system). Siciliano et. al. [40] used this method with two control inputs — one for the nominal control, and the other for the adaptive control which is devoted to ensure the stability of the whole system. This method is, however, not so promising for multi-link arms where the coupling terms in the joint variables are quite significant.

Self-tuning Adaptive Control (STAC)

In STAC, a constant order model with unknown parameters is assumed *a priori*, and the control scheme forces the parameters and the outputs to converge to their true values. However, this scheme needs the parameter estimator to operate faster than the rate of change of system dynamics and this poses a severe problem in certain cases. Nemir et. al. [31] used this method with a pole placement approach and assumed a second-order model (with a specified natural frequency and damping ratio) for the single flexible link. Lambert [25] has also used a similar approach with a Generalized Predictive Control (GPC) scheme.

Linear Perturbation Control

Nelson and Mitra [30] used this method to achieve load-adaptive optimal control for a single-link flexible arm. The arm was accurately modeled as a complex time-varying system and instead of re-calculating the parameters in real-time (nominal values of the parameters were assumed to be known *a priori*), they have designed a number of controllers off-line for various configurations. A supervisory controller was used to switch between these controllers or schedule the gain as the arm configuration is changed.

1.4 Conclusions

In all the modeling and control strategies discussed so far, the performance of the controller is largely dependent on the model accuracy of the manipulator. In other words, discrepancies between the model and actual system plays a very significant role in the deterioration of the performance of the controler. Such discrepancies include the effects of truncated higher order elastic modes, unmodeled friction and backlash effects, and, for linearized models, system nonlinearities. As a matter of fact, control design for flexible-link manipulators is not so well advanced as their dynamic modeling. Most of the control strategies developed so far, have not made use of the special features of flexible manipulators and ignored the practical limitations of manipulator actuation. For example Meirovitch [27] assumed a large number of distributed actuators and Singh [41] assumed the availability of tip actuators to demonstrate good control.

Another problem in controlling a flexible-link manipulator is due to its nonminimum phase behavior which makes it difficult to ensure closed-loop stability. In some recent works [33, 13], this problem has been solved by a "transmission zero assignment" technique. This method is based on designing a feed-through compensator $T(s)$ for a plant $G(s)$, such that the augmented plant $\hat{G}(s) = G(s)+T(s)$ is minimum phase and has transmission zeros at desired locations in the left half of the complex plane. As pointed out in [13], the design scheme using transmission zero assignment fails when substantial amount of Coulomb friction is present in the manipulator's hub. This is due to the approximation made in designing $T(s)$ using a linearized model based on an approximate model, which does not incorporate the friction terms. To overcome this difficulty, we have adopted an **Artificial Neural Network (ANN)** based learning strategy in this thesis to control the end-point of a flexible-link manipulator. This approach is motivated by the fact that the inherent learning ability of neural nets can be utilized to approximate highly nonlinear functions, while their massively parallel architectures assure fault tolerant operation.

Research efforts to date have produced various neural network based control approaches for rigid link manipulators [21, 28], flexible-joint manipulators [49, 50], flexible-link manipulators [9], and vibration suppression in flexible space structures [6, 18]. Majority of the neural network based control concepts developed are based on state feedback, an approach limited by the availability of the system states. In a flexible-link manipulator, the system states (modal coordinates) are difficult to measure without employing elaborate, model dependent state estimators and sensing devices. This makes the implementation of the control algorithms based on state feedback significantly difficult. These drawbacks have motivated the present research towards development of a neural network based control scheme that makes use of only output measurement as it becomes available from sensor readings. This scheme also surpasses the control strategy of [13] in its ability to compensate for the unmodeled and/or inaccurately modeled dynamics present in a flexible-link manipulator. This is done by training a neural network off-line using data from both the *Approximate Mathematical Model* and an *Accurate Model* which is the simulated representation of an experimental set up in our laboratory. The resulting system performs set-point tracking with reasonable accuracy.

Chapter 2

DYNAMIC MODEL AND THE CLOSED-LOOP SYSTEM

2.1 Introduction

In this chapter, we will use the assumed modes approach and Euler-Bernoulli beam theory to derive the dynamic equations of a single flexible-link manipulator. The nonlinear state-space model will then be linearized and transformed to a transfer function form to facilitate the control design in subsequent sections. Then we will develop a neural network based modeling and control strategy to meet our major design objectives, i.e. to perform reasonably accurate set-point tracking of the end point of the manipulator while ensuring closed-loop stability. The proposed

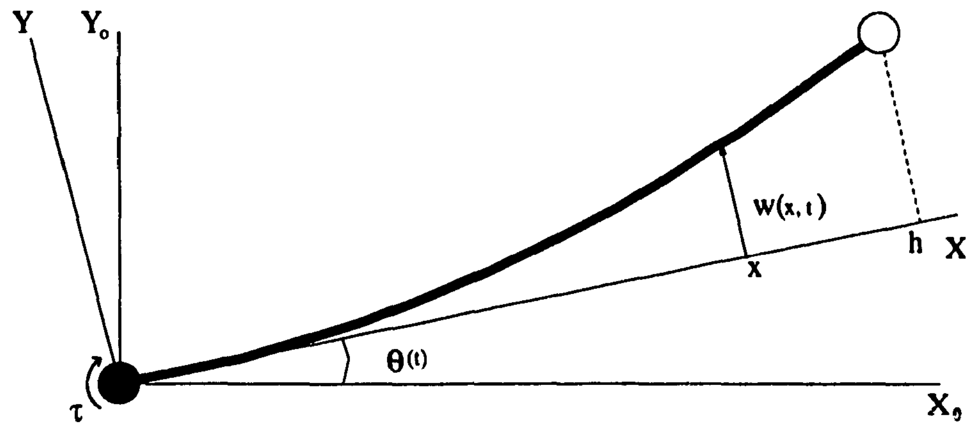


Figure 2.1: Flexible Link

controller uses a partitioned strategy, wherein the inner loop stabilizes the plant, and the outer loop (servo portion) provides set-point tracking. A detail discussion will be provided on the design strategy for the different compensators present in these two loops. The basic structure of the controller is similar to that in [14] with slight modifications. In order to overcome the deficiency in the controller, e.g. in handling the effects due to friction, we have used a neural network based approach to effectively identify the unmodeled and/or inaccurately modeled dynamics.

2.2 Analytical Model of a Single Link Arm

As shown in Figure 2.1, the flexible link is connected to a motor at one end (hub) and is driven by a torque τ. The other end is free to move, and has a small mass M_P as a payload. It is assumed that the length of the beam, h, is much greater than the width, w, thus restricting the beam to oscillate in the horizontal direction. All deflections of the beam are assumed to be small, and the effect of shear deformation and rotary inertia are neglected for simplicity. The beam has a moment of inertia I_h, and a linear mass density γ.

Using the assumed modes approach and Euler-Bernoulli beam theory, and neglecting the effects of shear deformation and rotary inertia, the dynamic behavior of a single flexible-link manipulator can be expressed by the following set of equations

$$\begin{aligned}
\ddot{\theta}[J_h + \frac{\gamma h^3}{3} + \gamma\sum_{i=1}^{n} q_i^2 + M_P(h^2 + \sum_{i=1}^{n}\phi_i(h)q_i)^2] &\quad + \\
\dot{\theta}[2\gamma\sum_{i=1}^{n} q_i\dot{q}_i + M_P\sum_{i=1}^{n}\sum_{j=1}^{n}\phi_i\phi_j(\dot{q}_i q_j + q_i\dot{q}_j)] &\quad + \\
b\dot{\theta} + C_{coul}(\frac{2}{1+\epsilon^{-\kappa\dot{\theta}}} - 1) &\quad + \\
\sum_{i=1}^{n}\ddot{q}_i[\gamma\int_0^h \phi_i x dx + M_P h\phi_i(h)] &= \tau(t) \qquad (2.1)
\end{aligned}$$

$$\begin{aligned}
\ddot{\theta}[\gamma\int_0^h \phi_j x dx + M_P h\phi_j(h)] + \gamma\ddot{q}_j + M_P\phi_j(h)\sum_{i=1}^{n}\phi_i(h)\ddot{q}_i &\quad + \\
c_j\dot{q}_j - [\dot{\theta}^2\gamma - EI\int_0^h(\frac{d^2\phi_j}{dx^2})^2 dx]q_j - M_P\dot{\theta}^2\phi_j(h)\sum_{i=1}^{n}\phi_i(h)q_i &= 0 \qquad (2.2)
\end{aligned}$$

These equations may be expressed in state-space form as

$$M(\boldsymbol{q})\begin{pmatrix}\ddot{\theta}\\ \ddot{\boldsymbol{q}}\end{pmatrix} + C(\boldsymbol{q},\dot{\boldsymbol{q}})\begin{pmatrix}\dot{\theta}\\ \dot{\boldsymbol{q}}\end{pmatrix} + K(\dot{\theta})\begin{pmatrix}\theta\\ \boldsymbol{q}\end{pmatrix} + \begin{pmatrix}F_c\\ \mathbf{o}\end{pmatrix} = \begin{pmatrix}\tau(t)\\ \mathbf{o}\end{pmatrix} \qquad (2.3)$$

with the net tip position output

$$y = \begin{bmatrix} h & \boldsymbol{\Phi}(\boldsymbol{h}) \end{bmatrix} \begin{pmatrix} \theta \\ \boldsymbol{q} \end{pmatrix} \tag{2.4}$$

where, $\theta \in \Re$, $\boldsymbol{q} \in \Re^n$, n is the number of elastic modes, $M(\boldsymbol{q})$ represents the inertia matrix, $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ represents the Coriolis and viscous damping matrix, $K(\dot{\theta})$ is the matrix of centrifugal forces, and $\boldsymbol{\Phi}(\boldsymbol{h})$ is the vector representing various mode shapes, i.e.

$$\boldsymbol{\Phi}(\boldsymbol{h}) = [\phi_1(h) \;\; \phi_2(h) \;\; \dots \;\; \phi_n(h)]$$

The various terms contained in the matrices can be found in [14], where the model is identical to that in (2.3) and (2.4), except for the Coulomb friction term F_c which has been modeled here as a sigmoid function (instead of a pure signum function, which has a discontinuity at the origin). The term F_c is given by

$$F_c = C_{coul}(\frac{2}{1 + e^{-\kappa\dot{\theta}}} - 1) \tag{2.5}$$

where $C_{coul} = \begin{cases} C^1_{coul} & \text{for } \dot{\theta} < 0 \\ C^2_{coul} & \text{for } \dot{\theta} > 0 \end{cases}$ represents the Coulomb friction coefficient.

The model of the flexible-link manipulator, as represented in equations (2.3) and (2.4) is highly nonlinear. In addition to the Coulomb friction term F_c, we note the following nonlinearities.

- *Inertia Matrix $M(\boldsymbol{q})$*: m_1 comprises of the products q_i^2 and $q_i q_j$, which are the terms indicating the change in the rigid body inertia of the manipulator as a result of deflection of the arm.

- *Coriolis and Viscous Damping Matrix $C(\boldsymbol{q}, t\boldsymbol{q})$*: Post-multiplying this matrix by $\dot{\theta}$ gives rise to some terms in C_1 which are of the type $q_i\dot{q}_i$ and $q_i\dot{q}_j$. These represent Coriolis forces.

- *Centrifugal Force Matrix $K(\dot{\theta})$*: Forming the product $\dot{\theta}^2 K_2 \boldsymbol{q}$ yields the vector of centrifugal forces.

All these nonlinearities play a very significant role in influencing the behavior of the manipulator, specially when the neighboring linear terms are relatively small in magnitude. We will consider these issues in the forthcoming sections.

It is worth noting in this context that accurate modeling of various nonlinear friction terms is an extremely difficult task. Moreover, a model such as (2.3), (2.4) may also have some unmodeled dynamics since it is a finite dimensional representation of a distributed parameter system. However, henceforth we will assume that the nonlinear model described by (2.3) and (2.4) accurately represents the behavior of the real flexible link manipulator, and we will call it the *Actual System*. A less accurate version of the model, which does not take into account the effect of friction will be called the *Approximate Model*.

These two nonlinear models can be represented by the following set of equations.

Actual System

$$\dot{\boldsymbol{V}} = \begin{pmatrix} \mathbf{o} & & \boldsymbol{I} \\ \cdots & \cdots & \cdots \\ & & \\ -M^{-1}K & & -M^{-1}C_{ac} \end{pmatrix} \boldsymbol{V} + \begin{pmatrix} \mathbf{o} \\ \cdots \\ M^{-1}\begin{bmatrix} \tau(t) - F_c \\ \mathbf{o} \end{bmatrix} \end{pmatrix} \tag{2.6}$$

and

$$y_{ac}(t) = \begin{bmatrix} h & \boldsymbol{\Phi}(\boldsymbol{h}) & \vdots & \mathbf{o}^T \end{bmatrix} \boldsymbol{V}(t) \tag{2.7}$$

Approximate Model

$$\dot{\boldsymbol{V}} = \begin{pmatrix} \mathbf{o} & & \boldsymbol{I} \\ \cdots & \cdots & \cdots \\ & & \\ -M^{-1}K & & -M^{-1}C_{ap} \end{pmatrix} \boldsymbol{V} + \begin{pmatrix} \mathbf{o} \\ \cdots \\ M^{-1}\begin{bmatrix} \tau(t) \\ \mathbf{o} \end{bmatrix} \end{pmatrix} \tag{2.8}$$

and

$$y_{ap}(t) = \begin{bmatrix} h & \boldsymbol{\Phi}(\boldsymbol{h}) & \vdots & \mathbf{o}^T \end{bmatrix} \boldsymbol{V}(t) \tag{2.9}$$

where, $\boldsymbol{V} = [\theta \ \boldsymbol{q} \ \vdots \ \dot{\theta} \ \dot{\boldsymbol{q}}]^T$ represents the states of the system. The Coriolis and viscous damping matrices, C_{ac} and C_{ap}, are almost identical, except for the viscous damping coefficient b, which is present in the first partitioning matrix C_1 of C_{ac} but absent in C_{ap}.

2.3 Parameter Estimation

The parameters used in this thesis for computer simulations are all based on an actual experimental set-up in our laborarory. These parameter values have been determined in an earlier work [14]. They are listed below in a tabular format. We have used 4 elastic modes for simulation, which means $n = 4$.

I_h	0.3	$Kg.m^2$
b	0.59	$N-m/rad.s^{-1}$
C_{coul}^1	4.77	$N-m$
C_{coul}^2	4.74	$N-m$
h	1.2	m
γ	1.208	$Kg.m^{-1}$
EI	1.94	$N-m^2$
M_P	0.03	Kg
ω_1	3	$rad-s^{-1}$
ω_2	19	$rad-s^{-1}$
ω_3	52	$rad-s^{-1}$
ω_4	102	$rad-s^{-1}$
k_1	1.5387	
k_2	3.8734	
k_3	6.4062	
k_4	8.9721	
c_1	0.4	
c_2	4.0	
c_3	2.0	
c_4	5.0	

Table 2.1: Flexible-link Parameters

2.4 Linearization of the Dynamic Model

Having developed the nonlinear dynamic model of the flexible-link manipulator, we now focus our attention on linearizing this model around a chosen equilibrium point. This will serve as a starting point for designing various controlers in subsequent chapters. This linear model is however an approximation of the original nonlinear system and remains valid only in the vicinity of the equilibrium point. The approximation deteriorates with the increase of the range of operation.

This local linearization is performed by using a Taylor series expansion of each term of (2.1) and (2.2), and neglecting second and higher order terms in the expansion. We choose an equilibrium point at $\theta_0 = \dot{\theta}_0 = \ddot{\theta}_0 = 0$ and $\boldsymbol{q}_0 = \dot{\boldsymbol{q}}_0 = \ddot{\boldsymbol{q}}_0 = \mathbf{o}$. Physically, this implies a *rigid* manipulator whose hub velocity and hub acceleration are zero.

Let us denote the small variations of τ, θ and $\boldsymbol{q}$ as $\delta\tau$, $\delta\theta$ and $\delta\boldsymbol{q}$ respectively. Partitioning $M(\boldsymbol{q})$, $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $K(\dot{\theta})$ as

$$M(\boldsymbol{q}) = \begin{bmatrix} m_1(\boldsymbol{q}) & \vdots & \boldsymbol{m}_2^T \\ \dots & \vdots & \dots \\ \boldsymbol{m}_2 & \vdots & M_3 \end{bmatrix}, C(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} C_1 & \vdots & \mathbf{o}^T \\ \dots & \vdots & \dots \\ \mathbf{o} & \vdots & C_2 \end{bmatrix}, \text{and } K(\dot{\theta}) = \begin{bmatrix} 0 & \vdots & \mathbf{o} \\ \dots & \vdots & \dots \\ \mathbf{o} & \vdots & K_1 - \dot{\theta}^2 K_2 \end{bmatrix},$$

and linearizing about the specified operating point yields the perturbed state-space model

$$\dot{V} = \left(\begin{array}{ccc} \mathbf{o} & & I \\ \cdots & \cdots & \cdots \\ -M_L^{-1}K_L & & -M_L^{-1}C_L \end{array} \right) V + \left(\begin{array}{c} \mathbf{o} \\ \cdots \\ M_L^{-1} \begin{bmatrix} \delta\tau \\ \mathbf{o} \end{bmatrix} \end{array} \right) \tag{2.10}$$

$$\delta y = \begin{bmatrix} h & \Phi(h) & \vdots & \mathbf{o}^T \end{bmatrix} V \tag{2.11}$$

where, $V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$, $V_1 = \begin{bmatrix} \delta\theta \\ \delta q \end{bmatrix}$, $V_2 = \dot{V}_1$,

$M_L = \begin{bmatrix} m_1(q_0) & m_2^T \\ m_2 & M_3 \end{bmatrix}$, $C_L = \begin{bmatrix} b + \frac{\kappa C_{coul}}{2} & \mathbf{o}^T \\ \mathbf{o} & C_2 \end{bmatrix}$, and $K_L = \begin{bmatrix} 0 & \mathbf{o}^T \\ \mathbf{o} & K_1 \end{bmatrix}$. This perturbed state-space model will be used in the subsequent sections for designing a feedthrough compensator in the closed-loop control system.

2.5 Bottlenecks of Controller Design

As discussed in the previous chapter, the control design of flexible-link manipulators is significantly difficult for a number of reasons : nonminimum phase behavior, unmodeled and/or inaccurately modeled dynamics, model truncation, and control and observation spillover. Considering all these problems, the first task of the designer is to choose between two broad areas of control, viz. linear and nonlinear control, to proceed further with the development of the compensators. Working with the nonlinear model has the immediate advantage of obviating the need for linearization. This is significant because a controller based on a locally linearized model does not usually perform well when the region of operation is far from the operating point. However, nonlinear systems are difficult to handle mathematically, and a wealth of techniques exist in the literature for the design and analysis of linear systems. For these reasons, controllers are designed using the linearized model of the flexible-link manipulator. To this end, we also have to decide the nature of linearization to be performed. A *globally linearized* system, i.e. one which is designed to transform the nonlinear system into one that behaves in a linear fashion over the entire range of operation, outperforms a *locally linearized* system. But for a flexible-link manipulator, global linearization is not so easy to perform. It requires either output feedback or state feedback to accomplish the task. Flexible-link manipulators have unstable zero dynamics which cause the closed-loop system to become internally unstable, thereby eliminating the possibility of output feedback. Further, the states are difficult to measure, which in effect makes it difficult to implement a state feedback strategy. Considering these problems, we will design the controllers based on a locally linearized model of the flexible-link manipulator.

2.6 System Structure

The control strategy has been described earlier in [8]. The proposed controller consists of three major parts. The trained network and the plant together form what we call the *Compensated Model*, so named because it compensates for the presence of unmodeled and/or inaccurately modeled dynamics to give a representation that is close to a given

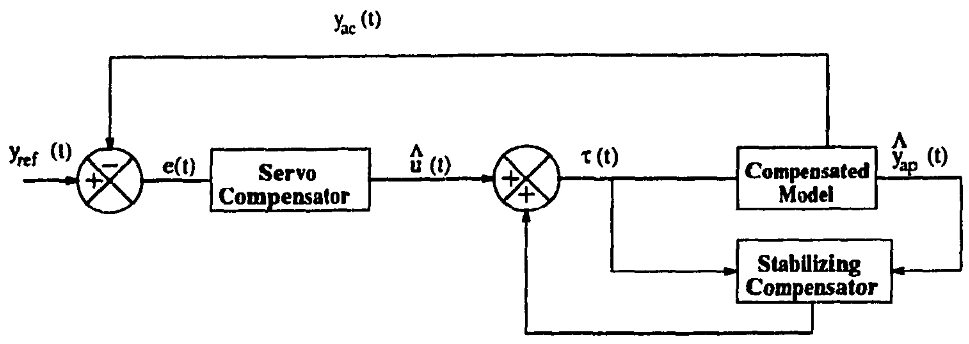


Figure 2.2: Proposed Control Scheme

mathematical model (approximate) of the flexible-link manipulator. The second portion of the controller is the inner loop whose sole responsibility is to stabilize the plant using a transmission zero assignment technique similar to that in [13]. The third portion is the outer loop, commonly referred to as the servo compensator, whose task is to achieve set-point tracking. Figure 2.2 illustrates the structure of the controller.

This partitioned control strategy satisfies both of our major design objectives. By employing a neural network previously trained off-line by the procedure to be described in Chapter 4, the *Compensated Model* closely matches its output $\hat{y}_{ap}(t)$ to the output $y_{ap}(t)$ of the *Approximate Model* of the flexible-link manipulator (which does not take into account the effect of friction). The *Compensated Model* is used in the design of the stabilizing compensator via a transmission zero assignment technique. As pointed out in [13], this technique fails when substantial amount of Coulomb friction is present in the manipulator's hub. This is due to the approximation made in designing the feed-through compensator in the stabilization loop (explained in section 2.6.2) using the

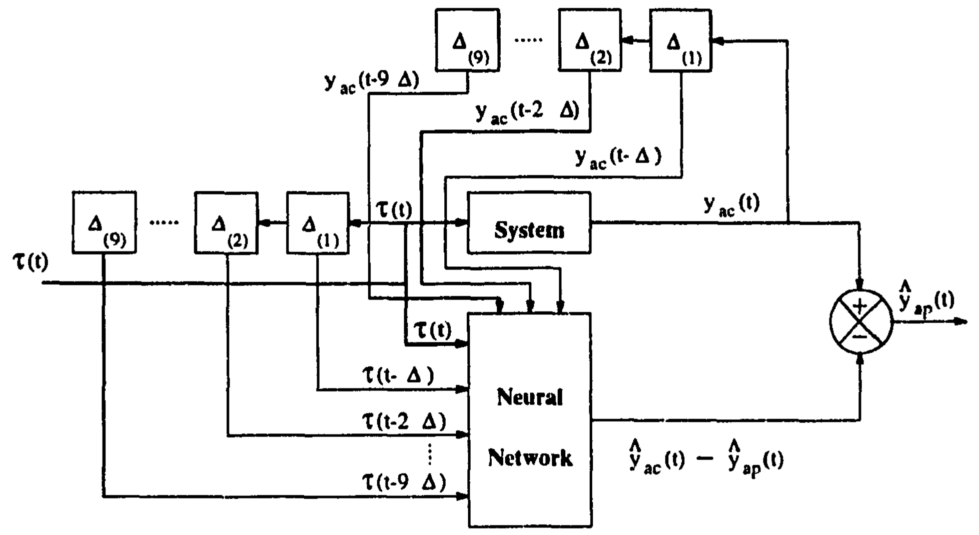


Figure 2.3: The Compensated Model

linearized model based on the *Approximate Model*, which does not incorporate friction terms. But the *Compensated Model*, when treated as a black box in terms of inputs and outputs, behaves in much the same way as the *Approximate Model*. Therefore, the corrections via the neural network which resulted in the *Compensated Model*, makes the the application of the feed-through compensator and various servo gains more accurate. The development of the different compensators is achieved as described below.

2.6.1 The Compensated Model

The structure of the *Compensated Model* is shown in Figure 2.3. It essentially consists of the plant in parallel with the neural network to cancel out the effect of the frictional terms. The net is trained using the delayed values of the torque $\tau(t)$ and the tip deflection $y_{ac}(t)$. The detail procedure for off-line training is described in Chapter

3. The same set of inputs are generated on-line when the network is used in the recall phase for closed-loop control. Several tapped-delay lines can be used to get these delayed signals.

2.6.2 The Stabilizing Compensator

As discussed previously, the flexible-link manipulator is a nonminimum phase system with one or more zeros in the right-half of the s-plane. [1] This type of system cannot be controlled just by applying constant gain output feedback because the system becomes unstable with even a moderate gain. One way to design the compensator is based on assigning the transmission zeros of the system to desired locations in the complex plane using a feed-through compensator. When output feedback is applied in the inner loop, the inner loop poles approach these transmission zeros. Thus the inner loop poles can be placed further within the left-half plane than the poles of the plant. When the outer loop is also closed and the poles of the complete closed-loop system begin to migrate toward the right-half plane zeros of the plant, the added margin of stability generated by the inner loop guarantees the stability of the closed-loop system for a large range of feedback gains. This approach is taken from [13] where the authors have successfully applied this method to control a flexible-link manipulator without considering frictional effects. The design is carried out in the following way.

[1]The reason for this nonminimum phase behavior is that the system is *non-collocated*, which means the actuator (at the hub) is not located at the same place as the sensor (at the tip).

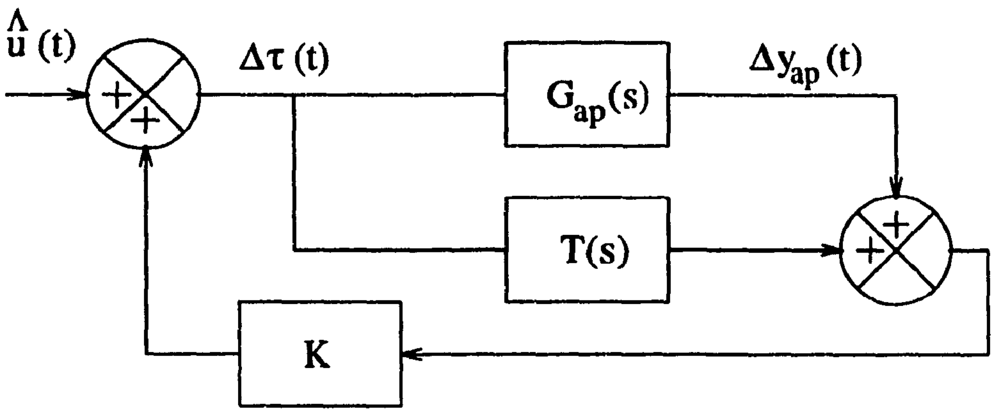


Figure 2.4: Stabilizing Compensator

Consider the structure of the inner stabilizing loop illustrated in Figure 2.4. Taking the Laplace transform of the linearized state-space model of (2.10) and (2.11), and setting b and C_{coul} to zero, we get the open-loop transfer function of the *Approximate Model* of the flexible-link manipulator as follows.

$$G_{ap}(s) = \frac{\Delta y_{ap}(s)}{\Delta\tau(s)} = \frac{p(s)}{q(s)} \tag{2.12}$$

where, $\Delta y_{ap}(s)$ and $\Delta\tau(s)$ represent the Laplace transform of of $\delta y(t)$ and $\delta\tau(t)$ respectively. They are written as two polynomial functions $p(s)$ and $q(s)$. Our objective is to design a feed-through compensator $T(s) = \frac{p_t(s)}{q_t(s)}$ such that the augmented plant $\hat{G}(s) = G_{ap}(s) + T(s)$ is minimum phase and has transmission zeros at desired locations in the left-half plane. Now,

$$\hat{G}(s) = \frac{p_t(s)q(s) + q_t(s)p(s)}{q_t(s)q(s)} \tag{2.13}$$

	ZEROS						POLES					
$G_{ap}(s)$	-1.9 ±76.9i	-1.1 ±47.9i	-20.1	14.3 ±3.9i	-7.9		0	-144.3	-2.8 ±102.4i	-0.8 ±52i	-1.8 ±18.9i	-0.2 ±2.9i
$\hat{G}(s)$	-3.4 ±102.4i	-10.1 ±54.4i	-46.4	-12.2 ±21.2i	-19.9	-10.5 ±3.2i	0	-144.3	-2.8 ±102.4i	-0.8 ±52i	-1.8 ±18.9i	-0.2 ±2.9i
$G_0(s)$	-1.9 ±76.9i	-1.1 ±47.9i	-20.1	14.3 ±3.9i	-7.9		-144.9	-9.9 ±102.1i	-9.9 ±52.1i	-10 ±19.1i	-18.7	-10.2 ±2.9i

Table 2.2: Pole-Zero Locations

The inner loop transfer function is given by

$$G_0(s) = \frac{\Delta y_{ap}(s)}{\hat{u}(s)} = \frac{G_{ap}(s)}{1 - K\hat{G}(s)} = \frac{q_t(s)p(s)}{q_t(s)q(s) - K[p_t(s)q(s) + q_t(s)p(s)]} \tag{2.14}$$

This expression is simplified further by choosing $q_t(s) = q(s)$, which reduces (2.14) to

$$G_0(s) = \frac{p(s)}{q(s) - K[p(s) + p_t(s)]} \tag{2.15}$$

Table 2.2 shows the pole-zero locations of $G_{ap}(s)$, $\hat{G}(s)$ and $G_0(s)$. These values are based on the values of $p(s)$ and $q(s)$ (which are known *a priori*) and for $K = -100$. The polynomial $p_t(s)$ is selected such that roots of $p_t(s) + p(s)$ (zeros of $\hat{G}(s)$) are at the locations shown in Table 2.2. This ensures adequate damping for the poles of $G_0(s)$, which is needed for closed-loop stability. The pole-zero locations are plotted in Figures 2.5 through 2.7.

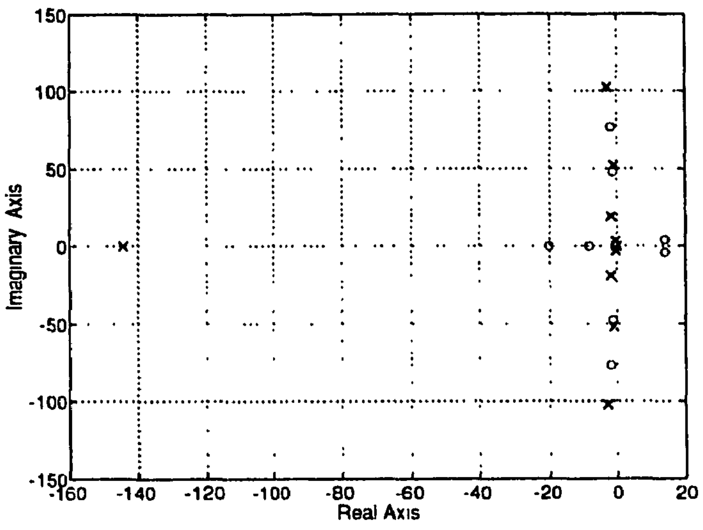


Figure 2.5: Pole-Zero locations of $G_{ap}(s)$

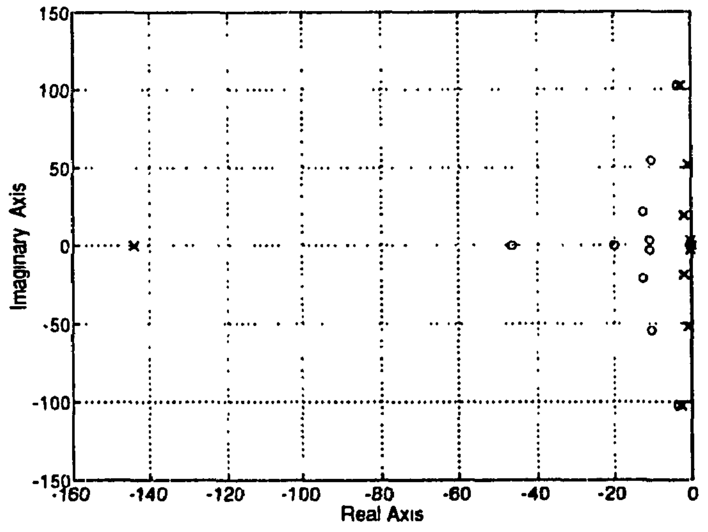


Figure 2.6: Pole-Zero locations of $\hat{G}(s)$

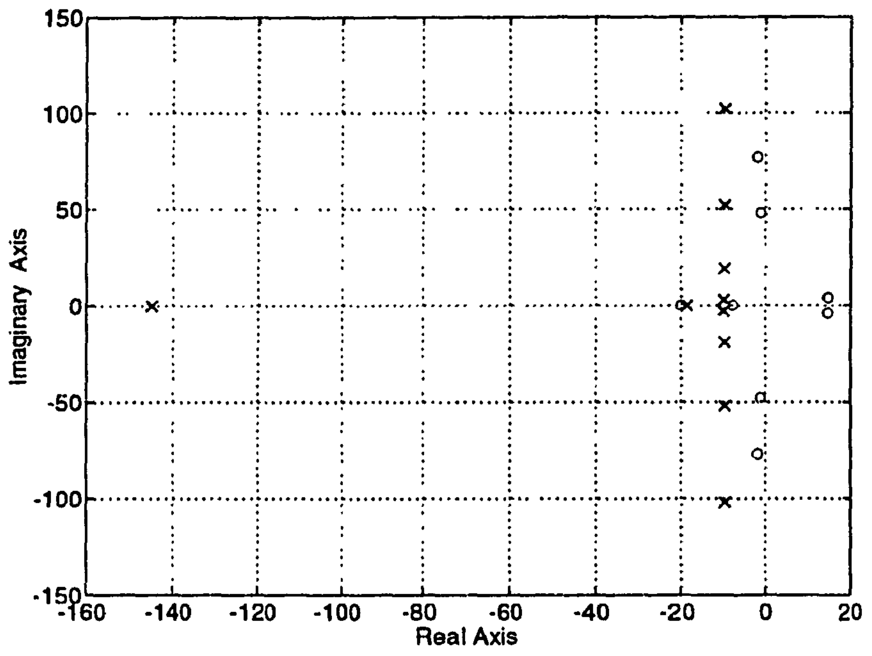


Figure 2.7: Pole-Zero locations of $G_0(s)$

2.6.3 Servo Compensator

The servo compensator comprises of a proportional and integral controller, as shown in Figure 2.8. This compensator takes the error $e(t)$ as its input and

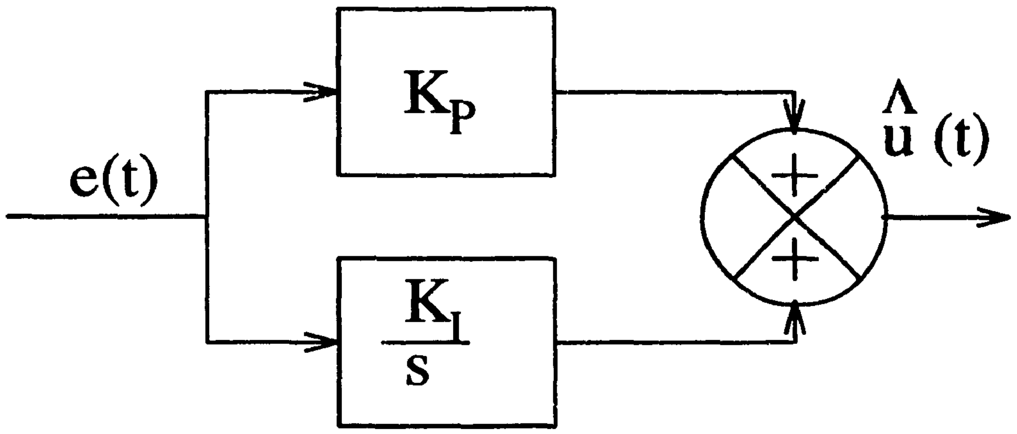


Figure 2.8: Servo Compensator

generates the output $\hat{u}(t)$ which drives $e(t)$ asymptotically to zero. The integral term helps to reduce the steady-state error while the proportional term attempts to reduce the oscillations . The values of these coefficients were chosen as $K_P = 8.5 \times 10^5$ and $K_I = 2 \times 10^6$.

Finally, we combine all these three blocks and illustrate the detailed structure of the closed-loop system in Figure 2.9.

Simulation results are provided in Chapter 4 to demonstrate the performance of the controller.

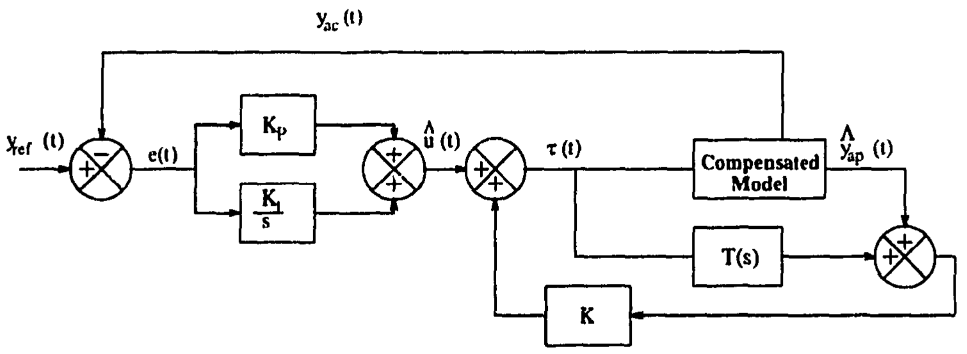


Figure 2.9: Closed-loop System

Chapter 3

NEURAL NETWORK TRAINING

3.1 Introduction

In this chapter, we will discuss the off-line training procedure for the artificial neural network to learn some of the unmodeled and/or inaccurately modeled dynamics present in the flexible-link manipulator. This network is embedded in the *Compensated Model* that was described in Chapter 2. We will choose a suitable network to meet our design objectives and discuss various aspects of its training, such as the learning rule, the network topology, choice of training signals, and techniques that ensure rapid convergence of the training.

3.2 Goal of Training

There is a wide variance in the approaches used in designing a neural network based robot controller. Several researchers have successfully trained neural nets to model the inverse dynamics of rigid-link/joint manipulators [21, 22], as well as flexible-joint manipulators [49, 50], and used the network for closed-loop control. This approach, however, fails for flexible-link manipulators because of its nonminimum phase characteristics, which render the inverse dynamics unstable. This, in effect, means that even if the overall closed-loop system is stable, the neural network itself could saturate, causing severe clipping errors. As pointed out in [48], one way of dealing with this problem is to introduce a sufficiently large delay in the system which stabilizes the delayed inverse model while preserving its amplitude response. Unfortunately, this method generally causes significant changes in the phase response and transient response, and is not an acceptable solution in real-time robot control.

Faced with the problem outlined above, we initially thought of devising a control scheme where a neural network will be trained to identify the entire forward dynamics of the flexible-link manipulator. However, a closer observation revealed that it was unsuitable for flexible-link manipulators because of their tendency to be only marginally stable. Their linearized models have complex conjugate pairs of poles close to the imaginary axis of the s-plane. This means that during the

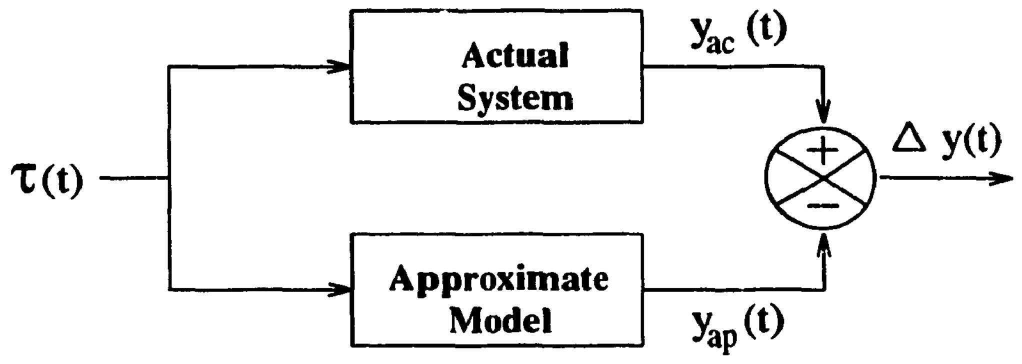


Figure 3.1: Off-line Training of the Neural Network

early stages of learning, the weight changes taking place within the neural network will tend to reinforce the oscillatory tendency of the plant. We then investigated a method of teaching the input-output mapping to the network which involves only *a part* of the forward dynamics. Our objective was to use the neural network as a learning controller which would add a compensating term to an *Approximate Model* so as to make it behave as close as possible to the *Actual System*.

The training scheme is shown in Figure 3.1. The goal of the training is to identify the input-output mapping between the applied torque $\tau(t)$ and the tip deflection error $\Delta y(t)$, which is the difference between the deflections of the *Actual System* and the *Approximate Model*. The output of the network, i.e. $\Delta y(t)$, can then be subsequently added to the output of the *Actual System* in the closed-loop system to compensate for the unmodeled and/or inaccurately modeled dynamics.

3.3 Selecting a Network Paradigm

It is widely accepted that presently there is no such thing as a "generic neural network". The designer usually chooses a specific network only for a particular application, and is even free to create hybrid paradigms by combining characteristics from various standard paradigms. One can also introduce an entirely new learning algorithm if there is a need for that.

Although the network design process is very heuristic, and involves some degree of trial and error, there are some criteria which must be considered when designing a net [51]

1. Associativity (hetero- or auto-)
2. Resolution (continuous, bi-state, tri-state, or discrete multiple states)
3. Learning rule
4. Number of layers
5. Number of nodes per layer
6. Direction of flow (feedforward or feedback)
7. Node input function
8. Node output function

9. Simulation control scheme

10. Amount of theoretical knowledge available on the paradigm selected (such as proof of stability or proof of convergence)

Referring to the criteria listed above, we now proceed to select our specific network. Noting that the network maps input torque $\tau(t)$ into tip deflection error $\Delta y(t)$, which is clearly a mapping from one space into another entirely different space, we conclude that the net has to be hetero-associative. We also would like to use a supervised learning strategy because the target vector, $\Delta y(t)$, can be known *a priori*. Further, the net should have a continuous input-output mapping. The latter implies that we need a continuous neural network, as opposed to a discrete net such as Counterpropagation.

After narrowing down our choice to a smaller group of networks, we focus our attention on a specific net that has gathered much importance in the recent years thanks to its ability to accomplish the task of system identification with a greater efficiency and ease. This is the ***backpropagation*** network which has associated with it some proofs of convergence and applicability to boost the confidence of the designer. Hecht-Nielsen [16] has shown that:

Theorem 3.1 *Given any $\epsilon > 0$ and any L_2 function $\boldsymbol{f} : [0,1]^n \in \Re^n \rightarrow \Re^m$, there exists a three-layer backpropagation network that can approximate $\boldsymbol{f}$ to within ϵ mean-squared error accuracy.*

This result has been corroborated by several others, such as Stinchombe and White [42]. We now select this network and proceed further to examine and resolve any problems that might be associated with it.

3.4 Neural Net Inputs and Outputs

Though a three-layer backpropagation network has been proven capable of approximating any arbitrary function, provided it has a sufficient number of nodes in the hidden layers, the only problem in using this network is the lack of "memory" which hinders the learning of temporal patterns in a dynamic system. The backpropagation network performs a strictly static mapping. Though the learning process is dynamical because of the ability of the net to self-adapt, the dynamics of learning is not directly related to the dynamics of system. The dynamics of learning are governed by network parameters and the learning algorithm, which are all selected by the designer. The net's dynamics are therefore used only to learn some unknown static mapping. This raises the fundamental question of finding a feasible way to use this network for learning the unmodeled dynamics of the flexible-link manipulator.

Currently, there are two approaches for addressing this problem. The first one is to use the states of the system as input to the network to encode the dynamics. This is done as follows. Consider equations (2.6) and (2.7) again. They can be written as

$$\dot{V} = f_{ac}(V, \tau) \tag{3.1}$$

$$y_{ac}(t) = g(V) \tag{3.2}$$

From (3.1) and (3.2), we can write

$$y_{ac}(t) = g(h_{ac}(\dot{V}, \tau)) = F_{ac}(\dot{V}, \tau) \tag{3.3}$$

Again, from equation (2.6) through (2.9), we can conclude that $y_{ap}(t)$ is a function of $y_{ac}(t)$. Hence

$$\Delta y(t) = y_{ac}(t) - y_{ap}(t) = \Phi(y_{ac}(t)) \tag{3.4}$$

This means,

$$\Delta y(t) = \Phi(F_{ac}(\dot{V}, \tau) = H(\dot{V}, \tau) \tag{3.5}$$

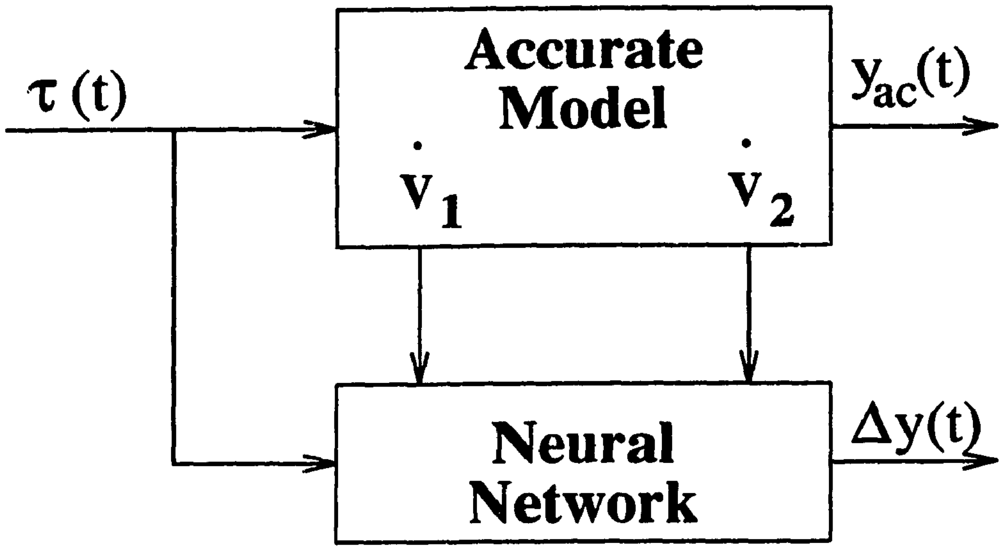


Figure 3.2: Net Inputs and Outputs Using the States of the System

We can train a neural network to identify this static function $H(\bullet)$ whose arguments may be given to the network as its inputs. The proposed scheme is shown in Figure 3.2.

This technique, however, has serious implementational problems. It requires all the states to be measurable which is possible only if they are real quantities and sensors can be designed to monitor them. For a flexible-link manipulator, the states (modal coordinates) are considerably difficult to measure. Further, there is a significant noise problem associated with measuring the time-derivative of the signals because there is currently no sensor available to measure $\ddot{\theta}$ and $\ddot{q}$ directly. To circumvent this problem, we select the second method. This is essentially to use a time delay network, i.e. one whose inputs consist of delayed values of $\tau(t)$

and $y_{ac}(t)$. These values are all computed off-line. This approach is motivated by considering the ARMA (Autoregressive Moving Average) difference equation that represents the dynamics of the manipulator.

Consider equations (2.6) through (2.9) again. Following the same argument that $y_{ap}(t)$ is a function of $y_{ac}(t)$, we can write $\Delta y(t)$ as

$$\Delta y(t) = y_{ac}(t) - y_{ap}(t) = \Phi(y_{ac}(t)) \tag{3.6}$$

Keeping in mind that $\boldsymbol{V} = [\theta \; \boldsymbol{q} \; \dot{\theta} \; \dot{\boldsymbol{q}}]^T \in \Re^n$ represents the states of the system with $n = 10$ (corresponding to 4 elastic modes), we can get the ARMA model as

$$\Delta y(t) = F(\tau(t),\; \tau(t-\Delta),\; \ldots \tau(t-9\Delta),\; y_{ac}(t-\Delta),\; y_{ac}(t-2\Delta),\; \ldots y_{ac}(t-9\Delta)) \tag{3.7}$$

where $F(\bullet)$ is some nonlinear function.

We attempt to train the neural network to identify this function $F(\bullet)$, whose arguments are given to the network as inputs. The scheme is shown in Figure 3.3. The strength of this approach stems from the fact that we no longer need to measure all the states of the system, instead we require only the tip deflection which can be measured relatively easily and accurately.

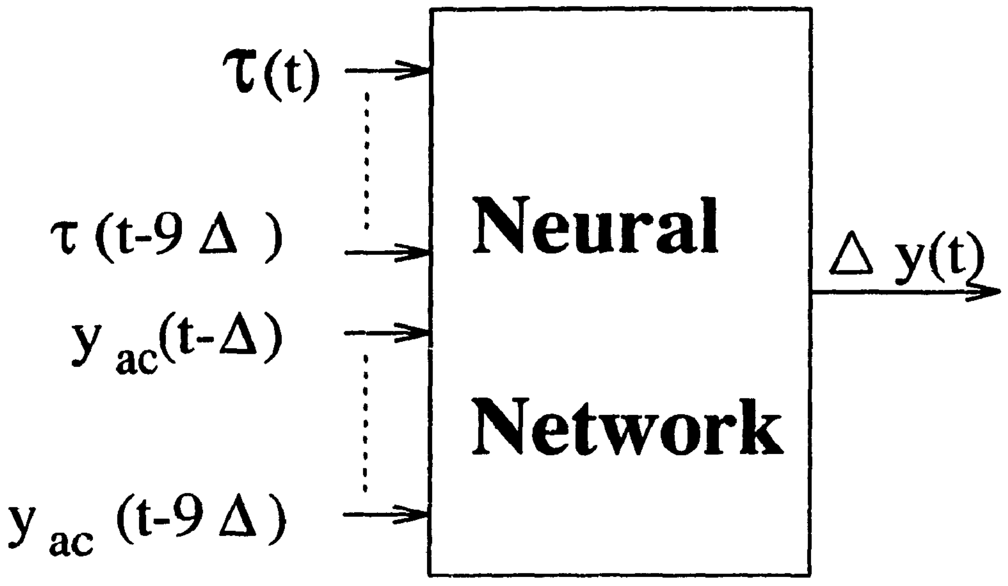


Figure 3.3: Net Inputs and Outputs Using the ARMA Model

3.5 Network Topology

In this research, we have experimented with many different network configurations. Some of the attempts were aborted because the network failed to converge to a minimum even after a sufficiently large number of epochs. But this gave us good insight into the dynamics of learning which helped in the choice of a better configuration in subsequent attempts.

In order to enhance the speed and accuracy of training, we decided to use a fully connected network. The initial guess for the size of the network was made using a result based on Kolmogorov's theorem [24], which states that:

Theorem 3.2 *Given any continuous function $f : [0,1]^n \in \Re^n \rightarrow \Re^m$, there exists a three-layer feedforward network having n fan-out neurons in the first layer, $(2n+1)$ neurons in the hidden layer, and m neurons in the output layer which can approximate f to any desired degree of accuracy.*

This was only a first step . In our system, n and m were eq al to 19 and 1 respectively, which indicated the presence of at least 39 neurons in the hidden layer. Once we started with this configuration, we noticed the failure of the network to converge to a minimum. A closer observation revealed that almost $\frac{1}{3}$ of the neurons did not change their weights very much from their initial values. These strongly suggested the need to prune those neurons and create a second hidden layer because those nodes were obviously not participating in the learning process. After this change was made, and we applied some advanced methods of learning convergence (mentioned in section 3.7), we got an acceptable solution with a network of size 19-25-20-1.

The structure of the network is shown in Figure 3.4. We have added an input buffer layer between the input and the first hidden layer. This layer does not perform any learning. It is used only to scale the net input signals so that the

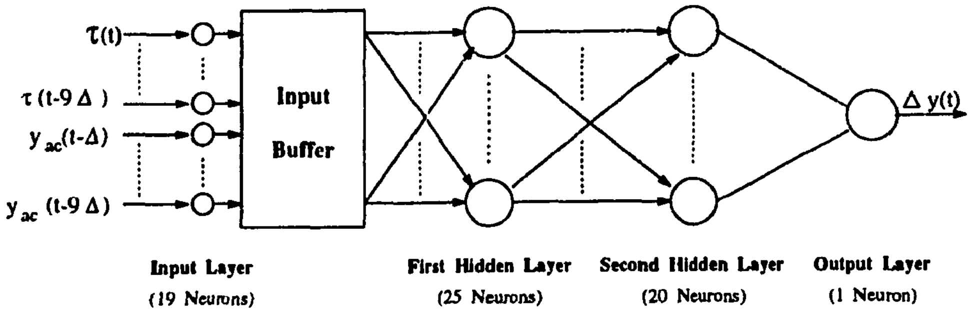


Figure 3.4: Topology of the Neural Network

nodes in the first hidden layer are not initially driven to saturation [1]. In our network, we have used the hyperbolic tangent function in the hidden layers to get a bipolar output in the range [-1,1]. So, in the buffer layer, we have scaled the inputs between -0.9 and +0.9 to allow some safety margin at both ends. This attenuation is done by passing the input signals through some fixed gains which are selected based on the observed operating range of the net inputs.

The output layer consists of a single neuron which generates the tip deflection error $\Delta y(t)$ as its output. We used a linear activation function in this layer instead of a sigmoid or hyperbolic tangent function. This was done to avoid further scaling of the output which would have been needed otherwise.

We have added a bias signal to all the neurons in the input and hidden layers to provide a fixed offset. During the training, if all the net inputs become zero

[1]Since the weight change is proportional to the derivative of the activation function, maximum changes occur for those neurons which are operating near the midrange of their activation function.

at some instant, but the desired net output is nonzero, then the presence of this bias signal becomes absolutely necessary for successful training. This offsets the origin of the logistic function, producing an effect that is similar to adjusting the threshold of the perceptron neuron, thereby permitting a more rapid convergence of the training process. This is incorporated by adding a weight to each neuron such that the source of the weight is always +1, instead of the output of a neuron in the previous layer.

3.6 Learning Rule

The backpropagation neural network has been trained using the generalized delta rule, which essentially implements gradient descent in sum-squared error for semi-linear activation functions [37]. We will use the following notation to describe this rule by considering the effect of a single learning iteration on the weights of the jth node in a given layer.

D_j	is the desired value of the node's output;
w_{jk}	is the current weight on the kth input line;
Δw_{jk}	is the change in w_{jk} as a result of the current learning iteration;
n_j	is the number of inputs to node j, where

$1 \leq k \leq n_j$;

I_{jk}	is the value of the kth input to node j;
A_j	is the activation function of node j;
O_j	is the output of node j.

Considering the input function for any node j as the weighted summation

$$A_j = \sum_{k \in K} w_{jk} I_{jk} \; + \; w_{jk} B \tag{3.8}$$

where,

K	is the set of all nodes in the layer below node j;
B	is the output of the bias node $(B = 1)$;
b	is the index of the bias node.

The weight change is formulated as

$$\Delta w_{jk} = \eta * \delta_j * I_{jk} \tag{3.9}$$

Here δ_j is the error signal, and η is a parameter, called the learning rate, which is set by the designer. For each node j in the output layer, the error signal is given by

$$\delta_j = (D_j - O_j) * f'_j(A_j) \tag{3.10}$$

where $\acute{f}_j(A_j)$ is the derivative of the jth node's output function

$$O_j = f_j(A_j) \tag{3.11}$$

For each node j not in the output layer, the error signal is given by

$$\delta_j = \acute{f}_j(A_j) * \sum_{q \in Q} (\delta_q * w_{qj}) \tag{3.12}$$

where the summation is taken over all nodes in the layer directly above node j, and w_{qj} is the weight on the connection from node j to node q.

The backpropagation algorithm applies the generalized delta rule to perform the learning. Figure 3.5 illustrates the flowchart of the error backpropagation training algorithm for a three-layer network such as that in Figure 3.4. The procedure shown in the flowchart is self-explanatory. After the inputs are applied and fed forward through the network, the error signals are computed for each node starting with the output layer, and the learning rate is re-adjusted (according to an algorithm explained in section 3.7.2). Then the weights of all nodes in that layer are updated, followed by updating of all the preceeding layer weights in a backward direction until the first hidden layer is reached. The cumulative cycle error is computed in step 3 as a sum-squared error over all outputs in the entire training set. The final error value for the entire training cycle is calculated after each completed pass through the training set. The learning procedure stops when the final error falls below the upper bound, E_{max}, set by the designer.

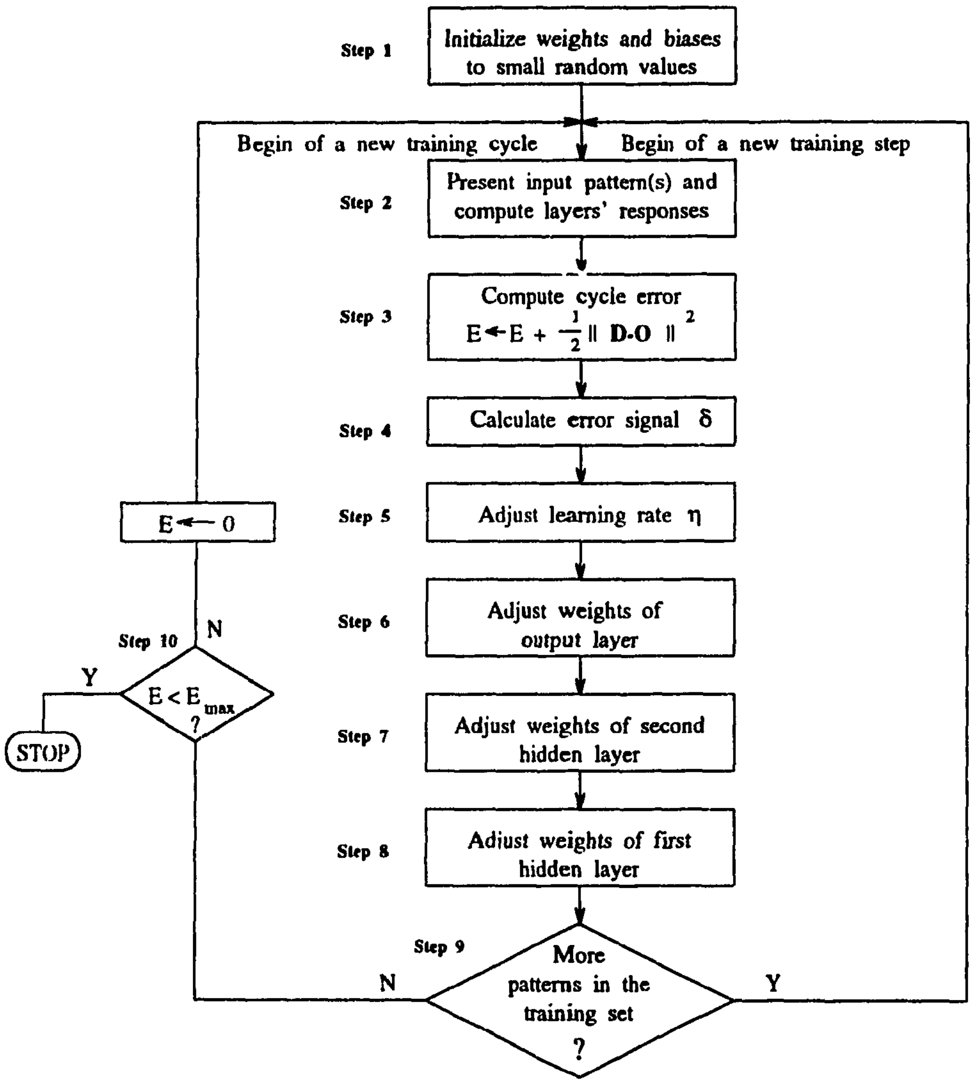

Step 1
Initialize weights and biases to small random values
Begin of a new training cycle
Begin of a new training step
Step 2
Present input pattern(s) and compute layers' responses
Step 3
Compute cycle error
E ← E + 1/2 ‖ D-O ‖²
Step 4
Calculate error signal δ
Step 5
Adjust learning rate η
Step 6
Adjust weights of output layer
Step 7
Adjust weights of second hidden layer
Step 8
Adjust weights of first hidden layer
Step 9
More patterns in the training set ?
N
Y
E ← 0
Step 10
E < E max ?
N
Y
STOP

Figure 3.5: Flowchart of the Error Backpropagation Training Algorithm

3.7 Problems Encountered using the Backpropagation Network

Despite the many successful applications of the backpropagation network, it is not a panacea. The greatest difficulty lies in the uncertain training process which may take a very long time, and at the end of it, the network may not be trained at all. As pointed out in [47], there are two major factors responsible for unsuccessful training.

Network Paralysis

As the network trains, the weights can become very large. This forces the neurons to operate in a region where the derivative of the activation function is very small. Since the error sent back for training is proportional to this derivative, the training process may virtually come to a standstill. Some researchers have tried to avoid this problem by using a small learning rate η, but that extends the training time considerably. Presently, there is no acceptable solution to this problem.

Local Minima

The backpropagation learning algorithm is essentially a multidimensional optimization problem employing a gradient descent method, i.e. it follows the slope

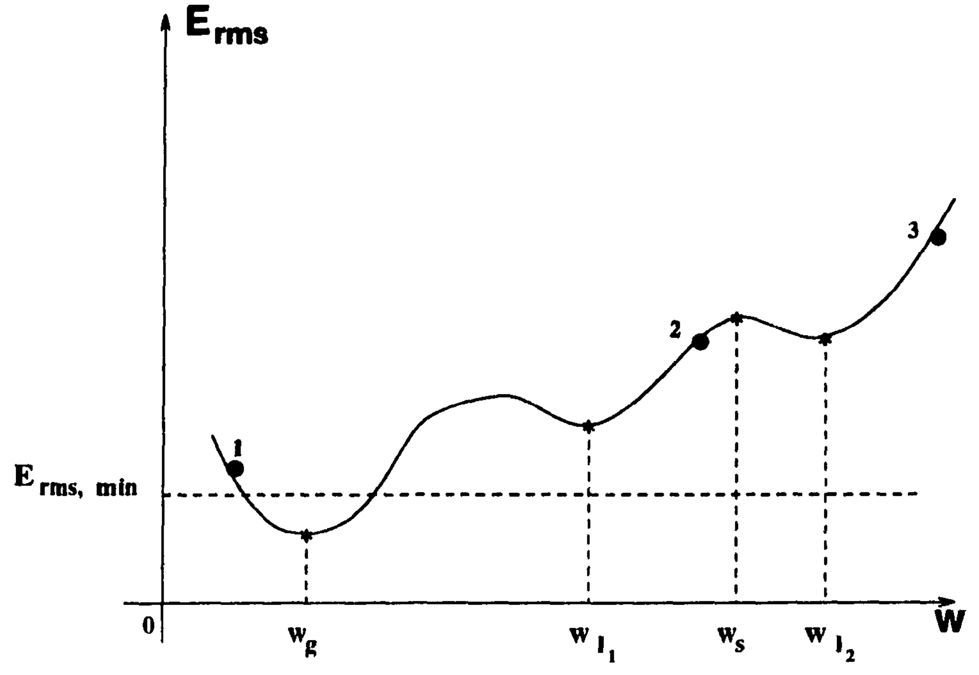


Figure 3.6: Minimization of the Error E_{rms} as a function of a single weight

of the error surface downward, constantly adjusting the weights in the process. The error surface of a complex network is highly convoluted, full of hills, valleys, folds and gullies in high-dimensional space. The network may get trapped in one of these local minima (a shallow valley) before reaching a much deeper global minimum, and may produce an erronious result. If the minimization process starts from an inappropriate initial condition, then it might not be possible for the network to reach a global minimum at any time.

As indicated in Figure 3.6 (a graph showing a cross-section of a hypothetical error surface in weight space), if the minimization starts from an initial point 2 or 3, then it will stop prematurely at the local minima w_{l_1} or w_{l_2}, located on both sides of the stationary point w_s. But if it is started from 1, then the global minimum, w_g, may be reached. Quite often, an appropriate choice of some learning parameters helps the network to get out of a plateau (local minimum) even if it gets trapped in one. But the optimum choice of network parameters is usually a very complex problem, and presently it is done by trial and error in most cases.

3.8 Measures taken for rapid convergence

In this thesis, we have taken a number of measures to ensure rapid convergence of the network to a minimum in a reasonable amount of time. They are as follows.

3.8.1 The Momentum Method

This method involves adding a term to the current weight adjustment that is proportional to the amount of the previous weight change. This additional term tends to keep the weight changes going in the same direction — hence the name *momentum*. This term typically helps to speed up convergence, and to achieve an efficient and more reliable learning profile. The weight adjustment equation (3.9) is then modified to the followinag:

$$\Delta w_{jk}(t) = \eta * \delta j * I_{jk} \; + \; \alpha * \Delta w_{jk}(t-1) \qquad (3.13)$$

Here α is the momentum coefficient which is usually set to a positive value less than 1. In our simulations, we have used a value of $\alpha = 0.8$.

3.8.2 Adaptive learning rate

Selection of a value for the learning rate parameter η, has a significant effect on the network performance. Usually, η is chosen to be a small number, of the order of 10^{-3} to 10, but the choice of a suitable value depends strongly on the nature of the learning problem and on the network architecture.

A large value of η generally speeds up convergence at the initial stage of learning. But as the error gets closer to a minimum, there is usually no further reduction. Further, the weights tend to fluctuate a great deal with a high learning rate, and the resulting oscillatory behavior can lead to system instability. There is also a danger of jumping from the global minimum's region of convergence to a local minimum's region of convergence. On the other hand, a small value of η yields a smooth trajectory of descent along the error surface, but the total training time (and hence the number of epochs) to settle down to a solution is considerably longer.

Many researchers have tried to exploit the benefits of both low and high η by the process of *simulated annealing* [23, 17]. In this method, the learning rate is initially set to a high value so as to progress quickly to the neighbourhood of a global minimum, while escaping any local minimum or flat region that might come along the way. Thereafter, the learning rate is set to a small value to facilitate smooth descent towards the global minimum and to reduce the amount of overshoot. The learning rate is, therefore, changed as a function of the number of epochs according to some "annealing schedule". This method is, however, quite heuristic in nature. There is no specific rule to guide the designer to choose various η values at different stages of training. Moreover, the same set of η values, chosen after a different number of epochs, may lead to entirely different results.

In this research, we have used the concept of an *adaptive learning rate*, which is essentially to make a trade-off between a high learning rate and a small learning rate in a continuous fashion. In this method of training, the ratio of the current output error to the previous one is checked at every step to determine whether the training shows a convergent or divergent trend. Based on this, the current learning rate is increased or decreased by specified factors. (Typically, we have chosen the values of decision ratio, incrementing factor and decrementing factor as 1.04, 1.05 and 0.7 respectively).

This method may provide a near optimal learning rate throughout the training process. It increases the learning rate at times to speed up the training, but does so only to the extent that the network can learn without large error increases. When the learning rate gets too high to ensure a further decrease in error, it gets reduced

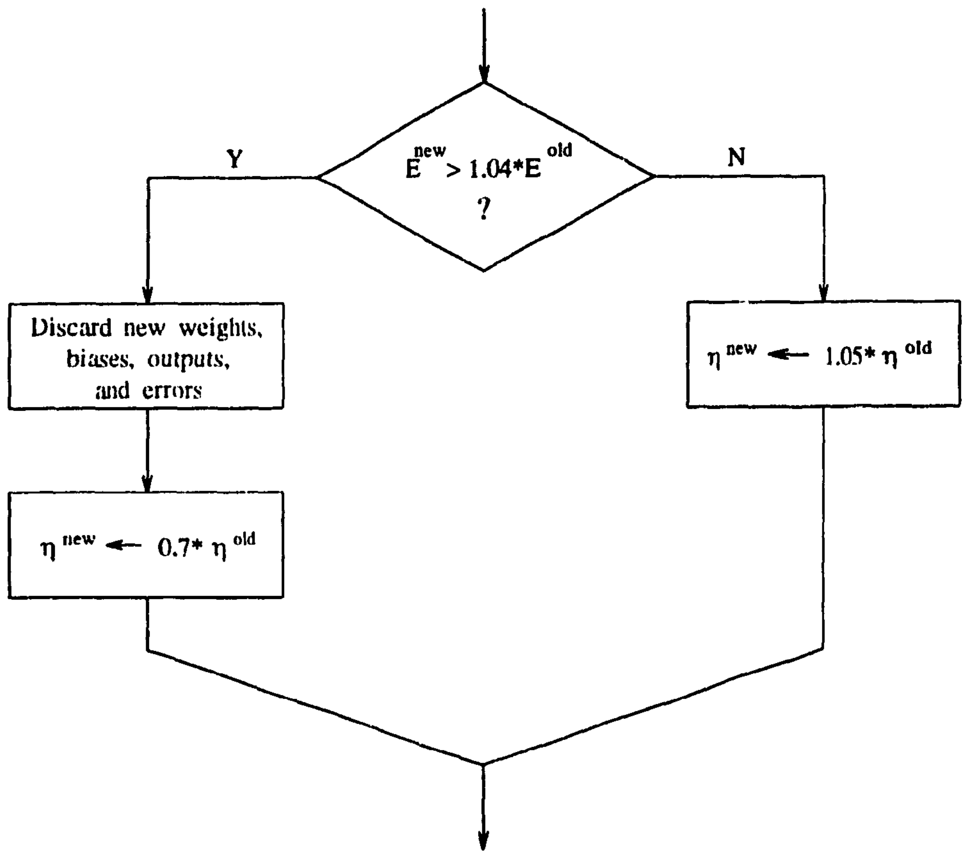

$E^{new} > 1.04 * E^{old}$?
Y
N
Discard new weights, biases, outputs, and errors
$\eta^{new} \leftarrow 0.7 * \eta^{old}$
$\eta^{new} \leftarrow 1.05 * \eta^{old}$

Figure 3.7: Flowchart of Learning Rate Adjustment Algorithm

automatically until stable learning resumes. This method surpasses simulated annealing in its ability to change the learning rate dynamically throughout the training, without having to go through the heuristic "annealing schedule" which updates the learning rate only at some pre-supposed discrete stages.

3.8.3 Better initial conditions

The choice of initial weights and biases plays a significant role in the training of the network towards an acceptable error minimum. Typically, they are initialized to small random values (to prevent an early saturation of neurons) without taking into consideration the dynamics of learning. Quite often, this leads to a very long training time, and in some cases the network may not converge at all.

In this thesis, we have used an improved method of picking up initial weights and biases that are better than pure random values. This scheme was proposed by Nguyen and Widrow [32], and it formulates a method for initialization based on the range of inputs and outputs.

It is well known that during training the network learns to implement a desired function by building piecewise linear approximations (of individual neuron outputs) to the function. The pieces are then summed to form the complete approximation. This indicates that the weights need to move in such a manner that the region of interest is divided into small intervals. It is then reasonable to consider speeding up the training process by setting the initial weights of the hidden layer so that each hidden node is assigned its own interval at the start of the training. The network is trained in the usual way, each hidden node still having the freedom to adjust its interval size and location during training. However, most of these adjustments are small since the majority of the weight movements have been eliminated by the method of setting their initial values.

Consider a network with H nodes in the first hidden layer. The network has $X \in [-1,\ 1]^n$ as the input vector, $W_i \in \Re^n$ as the weight vector of the i th node of the hidden layer, and W_{b_i} as the bias weight of the i th hidden node. Following Nguyen and Widrow's method, the elements of W_i are assigned values from a uniform random distribution between -1 and +1 so that its direction is random. Next, the magnitude of the weight vectors are adjusted as

$$||W_i|| = H^{\frac{1}{N}} \tag{3.14}$$

In our simulations, we set the magnitude of W_i to $0.7H^{\frac{1}{N}}$ to allow some overlap between the intervals. Then we locate the center of the interval of each node's activation function in a random manner by setting the value of W_{b_i} as a random number between $-||W_i||$ and $||W_i||$.

With the weights initialized as above, the network, in general, achieves a lower mean squared error in a much shorter time.

3.9 Training Signal

As explained in section 3.3, the backpropagation network has a total of 19 inputs comprising of torque $\tau(t)$ along with its past 9 values, and the past 9 values of tip deflection $y_{ac}(t)$. Our goal is to train the network with a fairly rich set of inputs so that it performs well for any arbitrary input that we may later present to it. The most general type of training signal would have been white noise. But, keeping in mind that our experimental set-up is bandlimited to approximately 1 kHz., we used a pseudo-random sequence of low-pass white noise as the training input set for the network.

The motivation behind the choice of this type of training signal is that it generates a uniformly populated input space during training. As a result, during the recall phase, any unknown input presented to the network stays relatively close to some of the inputs used during training, resulting in a smaller distance of interpolation for the net [35]. This type of signal also alows for an unordered presentation of inputs during training, a condition highly desirable for converging to a global minimum using the generalized delta rule.

Chapter 4

SIMULATION RESULTS

4.1 Introduction

This chapter presents the simulation results to corroborate the theoretical concepts developed in Chapters 2 and 3. The organization of this chapter is as follows. Section 4.2 illustrates the behavior of the network during training. Section 4.3 demonstrates the performance of the trained network in the recall phase. Section 4.4 presents the closed-loop system behavior when the trained network is used to control the manipulator, as well as the response in the absence of the network. Different reference trajectories are used to compare the tracking performance of the controller. Finally, in section 4.5, we have developed a robust controller for improved disturbance rejection, and provided the results for its tracking performance in section 4.6. The simulations were conducted on a SUN SPARC 2 computer using MATLAB and SIMULINK. Four elastic modes and one rigid body mode were used to model the manipulator. For this simulation study, the flexible-link manipulator, i.e. the *Actual System*, was represented by an *Accurate Model* incorporating a friction model. This is the model represented by equations (2.6) and (2.7). The *Approximate Model* did not include any friction effects. The parameters used in the *Accurate Model* were taken from [14], and are based on an actual experimental set-up in our laboratory.

4.2 Network Training

The network was trained in three stages. The first stage was from epoch 1 to 1000, the second stage was from epoch 1001 to 6000, and the third stage was from epoch 6001 to 9000. As shown in Figure 4.1, at the end of each stage, we set the learning rate to its initial value of 4 and started the training afresh. This was done to reduce the training time. Since an "adaptive learning rate" scheme was employed in the training, we could afford to increase the learning rate at times to speed up the training without running the risk of subsequent oscillatory behavior which could be caused if the learning rate was held at that high value for long. The learning curve is demonstrated in Figure 4.2. It shows that the network settled down to a minimum after 9000 epochs. An interesting observation can be

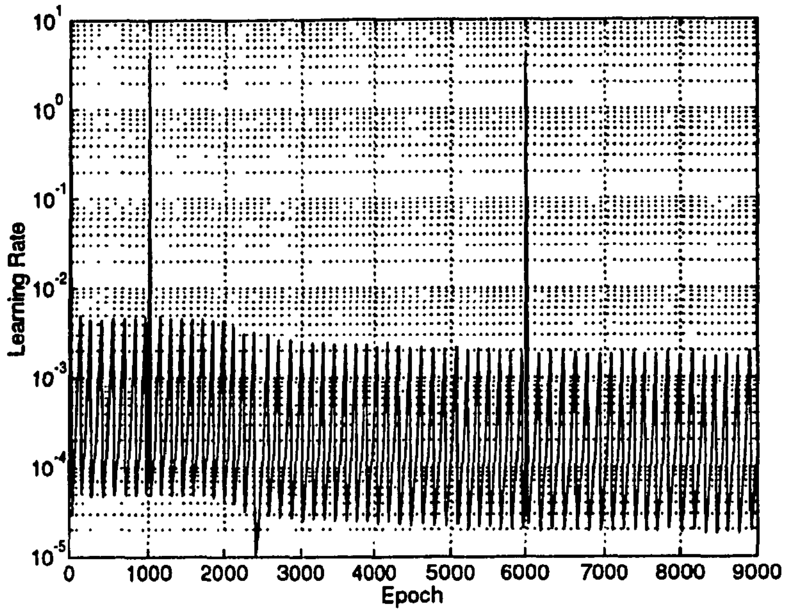


Figure 4.1: Adaptive Learning Rate

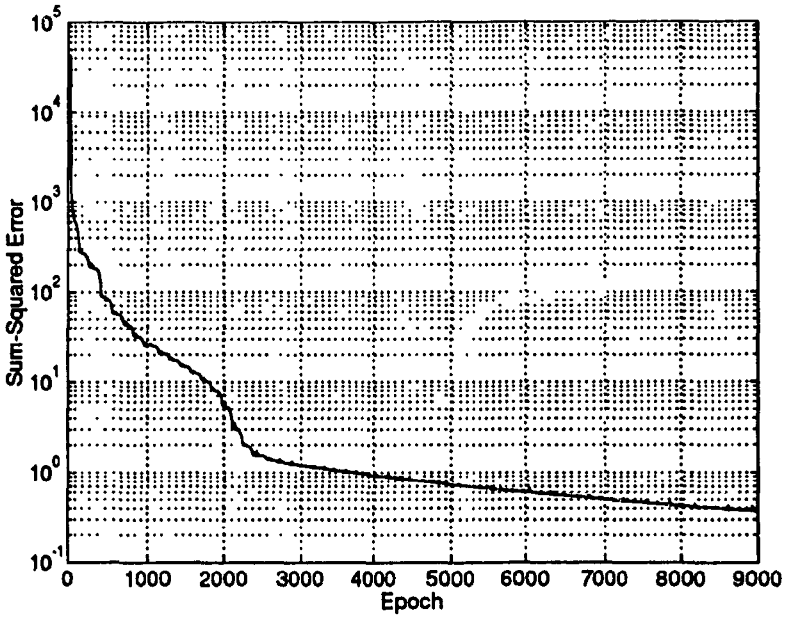


Figure 4.2: Learning Curve

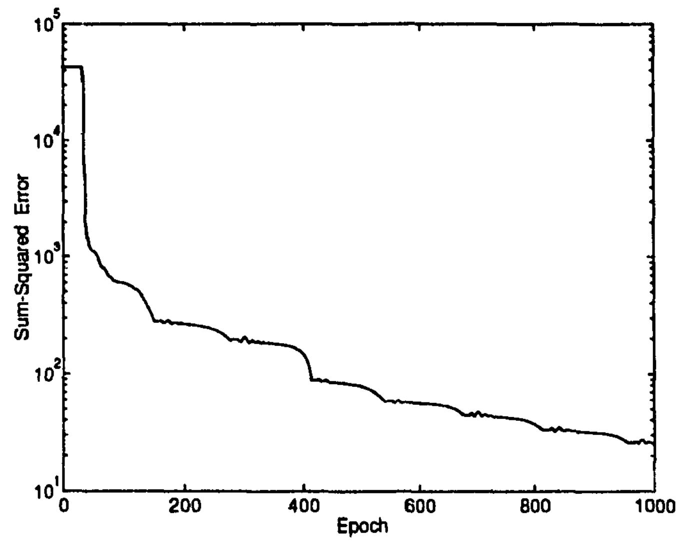


Figure 4.3: Network Error for First 1000 Epochs

made in this context. Instead of following a monotonically decreasing pattern, the error often increases sharply as training proceeds, and then decreases again gradually at a similar rate. A part of the learning curve is magnified in Figure 4.3 to reveal this behavior more clearly. Such fluctuations are due primarily to the pseudo-random nature of the training signal. If the training signal were not random in nature, we could expect the error to follow a more regular pattern.

4.3 Network Recall

Having trained the network off-line, we were interested in testing its performance. This was to ensure overall closed-loop stability when the neural network would

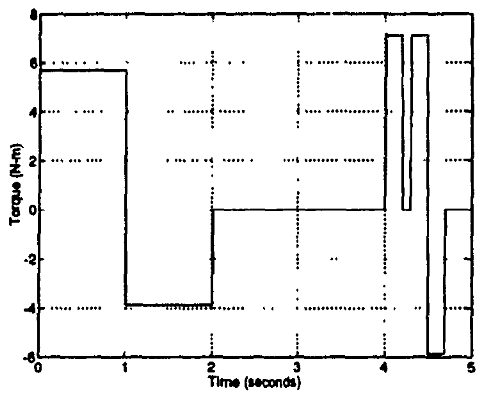


Figure 4.4: Test Input

be used as a controller in the system. For this purpose, we used a set of pulses for the torque input, as shown in Figure 4.4, to both the network and the set-up shown in Figure 3.1.

This input was unknown to the network, but was within its training input space. We also used past values of the torque, $\tau(t - i\Delta)$, and obtained the tip deflections, $y_{ac}(t - i\Delta)$, $i = 1, 2, \ldots 9$ off-line. They were applied as inputs to the network. We then compared the two responses as shown in Figure 4.5. The proximity of the two responses demonstrates conformity to the expected performance.

4.4 Closed-Loop Response

The closed-loop with the *Compensated Model* is shown in Figure 4.6, where y_{ref} is shown by the dashed line and y_{ac} is shown by the solid line.

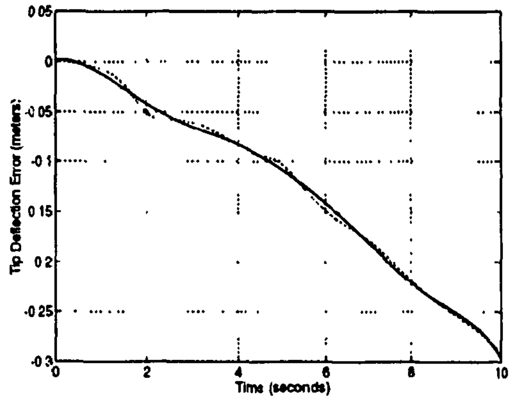


Figure 4.5: Function Approximation

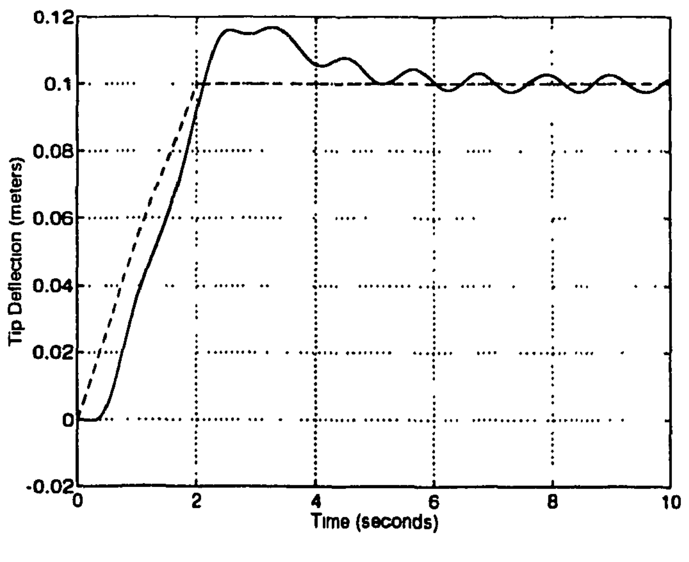
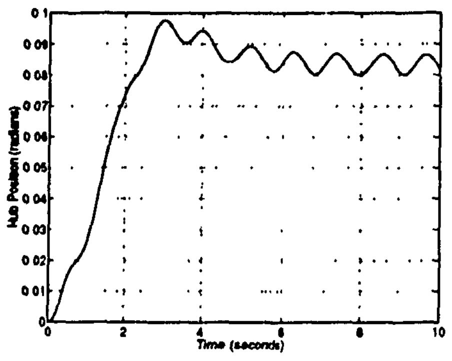
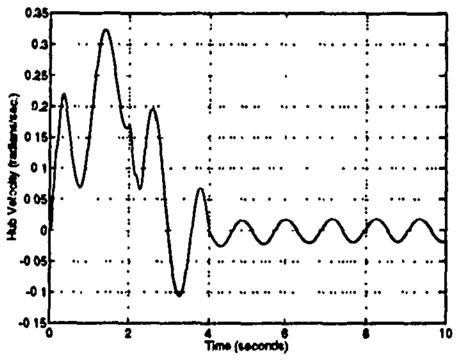


Figure 4.6: Closed Loop Response of $y(h,t)$ with the Network



(a)



(b)

Figure 4.7: (a) Variation of θ and (b) Variation of $\dot{\theta}$

A closer examination reveals that the tip deflection initially exhibits a negative going excursion before attempting to follow the reference trajectory. This is due to the phase delay characteristic of nonminimum phase systems which increases with frequency. That is why the effect is more pronounced in the transient portion (which contains the high frequency components) of the trajectory.

The steady state part of the trajectory shows a 3% oscillation about the nominal value. To explain this oscillatory behavior, we have to look at the hub angle and hub velocity profile in Figure 4.7.

The hub velocity $\dot{\theta}(t)$ shows a change from 0 to 0.33 radians/sec. during the first 1.7 second of simulation. This large variation causes the tip deflection, and hence the vector $q(t)$, to increase in magnitude. As a result, the centrifugal force vector

$\dot{\theta}^2 K_2 q$ becomes significantly large, and the controller, whose design is based on the linearized plant model for which $\dot{\theta}_0(t) = 0$, fails to perform satisfactorily. Further, the variation of $\dot{\theta}_0(t)$ exerts a time-varying influence upon the damping of the plant model, and this causes a disturbance effect upon the controller which has been designed for a linear time invariant system. This perturbation is manifested as an oscillation of the tip deflection that also appears in the net tip position. Another reason for the oscillation is ascribed to the numerical integration performed during simulation. Such discrete integration necessarily introduces small discontinuities for the hub position values. As a result, when the hub velocity is calculated by numerically differentiating the hub position, these discontinuities inject some amount of noise in the system. This noise becomes dominant towards the end of simulation (when nominal value of hub velocity approaches zero, and the noise is large compared to that value), and causes a variation in the sign of the hub velocity. Since, computation of control torque involves the term $C_{coul}\ sigmoid(\dot{\theta})$, this positive and negative going excursions results in large swings of control torque, which manifests itself as an oscillation at the tip. Last but not the least, though there is a finite amount of tracking error, the observed difference in performance is actually within the same order of magnitude as the fluctuations caused by pseudo-random training, and the slight error in function approximation by the neural network should be accepted because the *Compensated Model* can never be

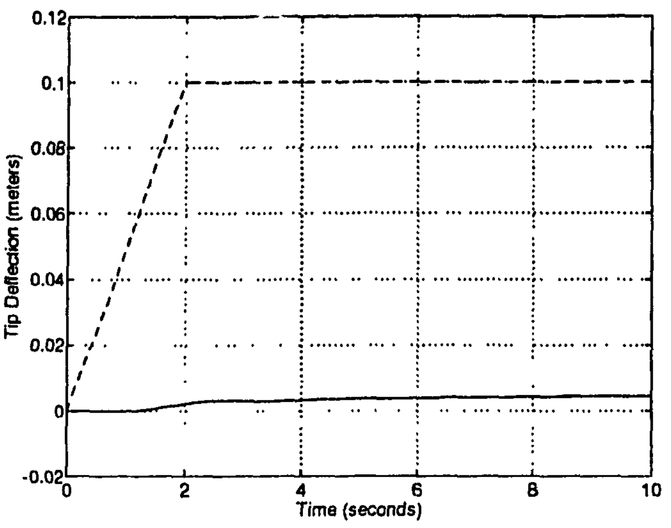


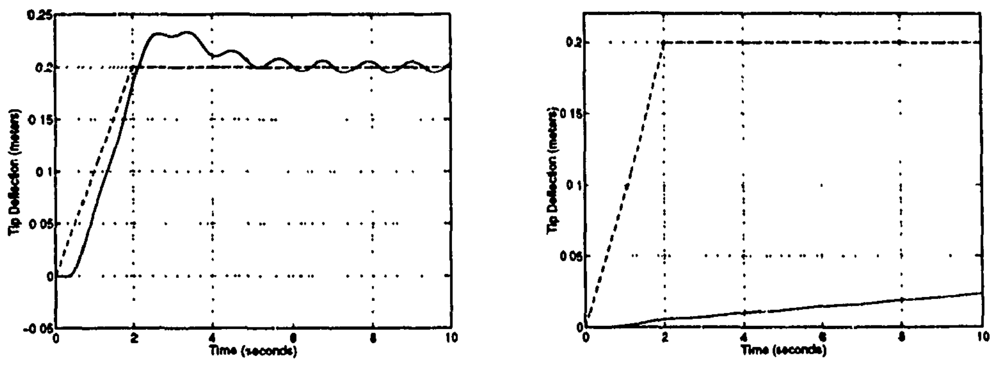
Figure 4.8: Closed-Loop Response of $y(h,t)$ without the Network

an *exact replica* of the *Approximate Model* of the manipulator.

The response of the system when the neural network is omitted is shown in Figure 4.8.

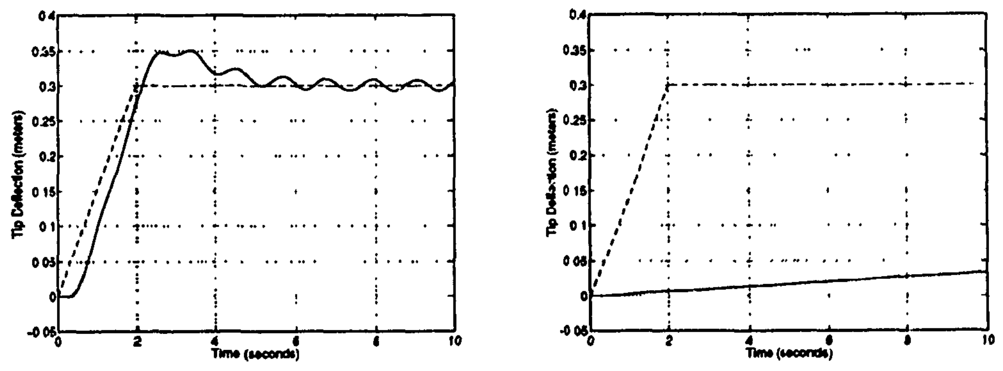
The controller parameters T, K, K_P and K_I were the same as those used for the *Compensated Model*. It is clear that the neural network provides significant improvement in the closed-loop performance by compensating for friction and other unmodeled dynamics.

Next, we would like to observe the tracking performance of the controller with respect to similar reference trajectories but with higher steady state values. Figure 4.9 and 4.10 show the closed-loop behavior for two such trajectories.



(a) (b)

Figure 4.9: (a) Closed-Loop Response with the Network (for $y_{max} = 0.2$ m) and (b) Closed-Loop Response without the Network (for $y_{max} = 0.2$ m)



(a) (b)

Figure 4.10: (a) Closed-Loop Response with the Network (for $y_{max} = 0.3$ m) and (b) Closed-Loop Response without the Network (for $y_{max} = 0.3$ m)

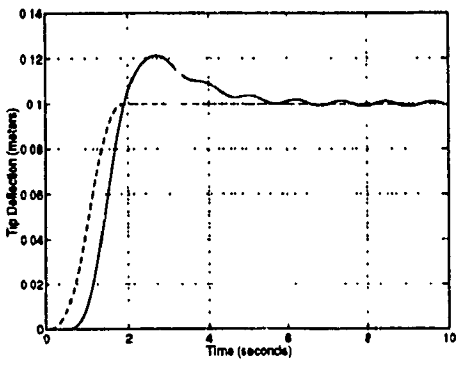
We note an increase in the amplitude of oscillation as the reference input goes higher. This is expected since the increased values of $q(t)$, $\dot{q}(t)$, $\theta(t)$ and $\dot{\theta}(t)$ play a more significant role in causing the system to deviate from the nominal operating point and thereby deteriorating its performance.

A further increase in the value of the reference input makes the system unstable. This is due primarily to the combined effect of decreased damping of flexible dynamics caused by higher hub velocity, larger drift away from the operating point, and gross violation of the Euler-Bernoulli requirement for small deflection.

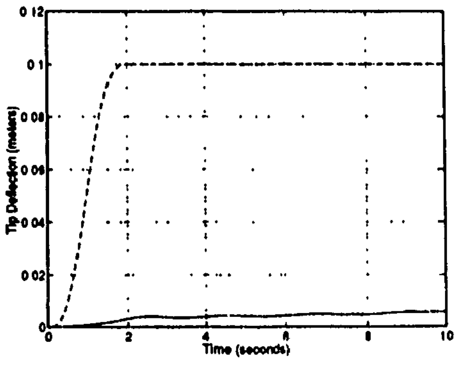
Figure 4.11 through 4.13 illustrate the performance of the controller with respect to a different kind of reference trajectory. This smooth trajectory is constructed in a way that it has a continuous position and velocity everywhere on it. This is generated as folllows.

$$y_{ref}(t) = \begin{cases} \frac{\beta}{2\pi}(y_f - y_0)[t - \frac{1}{\beta}cos(\frac{3\pi}{2} + \beta t)] + y_0 & ; 0 \leq t \leq \frac{2\pi}{\beta} \\ y_f & ; t > \frac{2\pi}{\beta} \end{cases}$$

For our simulations, we have used three similar set of trajectories with a rise time $t_r = \frac{2\pi}{\beta} = 2$ sec., total duration $t_t = 10$ sec., and initial value $y_0 = 0$ meter. The only difference between them was the steady state value, y_f, which was chosen as 0.1, 0.2, and 0.3 meters respectively. This was done to compare it with the corresponding linear segment trajectories.

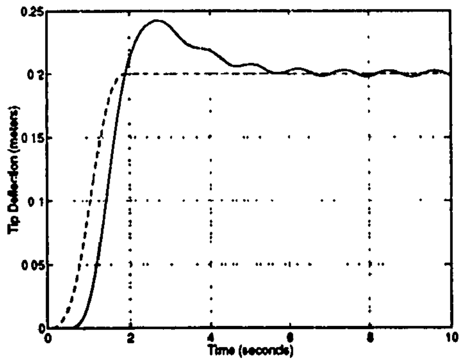


(a)

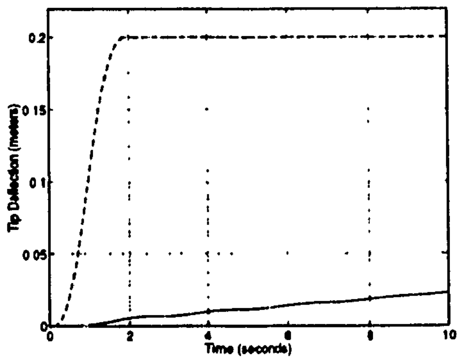


(b)

Figure 4.11: (a) Closed-Loop Response with the Network (for $y_{max} = 0.1$ m) and (b) Closed-Loop Response without the Network (for $y_{max} = 0.1$ m)

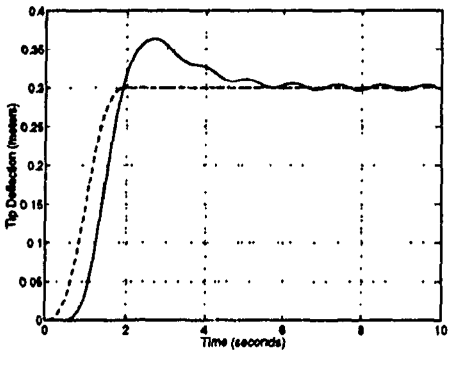


(a)

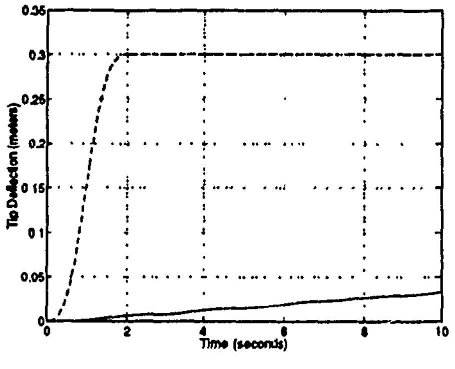


(b)

Figure 4.12: (a) Closed-Loop Response with the Network (for $y_{max} = 0.2$ m) and (b) Closed-Loop Response without the Network (for $y_{max} = 0.2$ m)



(a)



(b)

Figure 4.13: (a) Closed-Loop Response with the Network (for $y_{max} = 0.3$ m) and (b) Closed-Loop Response without the Network (for $y_{max} = 0.3$ m)

In all the above simulations, we observe a reduced amplitude of oscillation compared to their linear segment counterparts. This is not unexpected since, by avoiding the discontinuity of $\dot{y}_{ref}(t)$ at 2 second, we have been able to eliminate the discontinuity of the hub velocity and acceleration. This results in less perturbation for the control torque, and hence ~ improved tracking performance.

Further, we are interested to see the effect of an input that excites the higher order modes of the manipulator. Recalling that the network was trained with a low-pass noise signal, our objective is to test the robustness of the controller in presence of high frequency components. Figures 4.14 and 4.15 illustrate the performance of the controler for the reference trajectories $0.2 \sin t$ and $0.2 \sin 5t$ respectively.

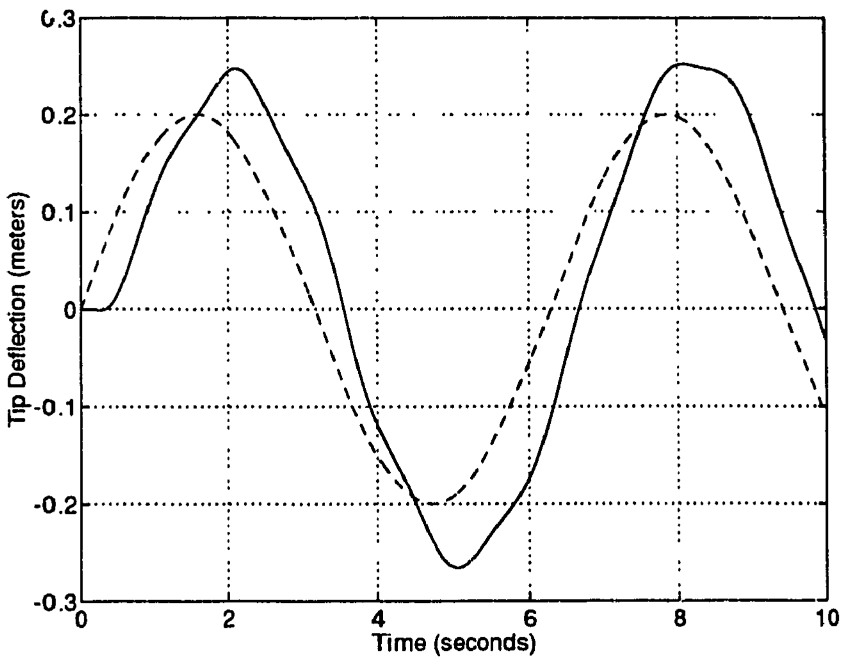


Figure 4.14: Closed-Loop Response for $y_{ref}(t) = 0.2 \sin t$

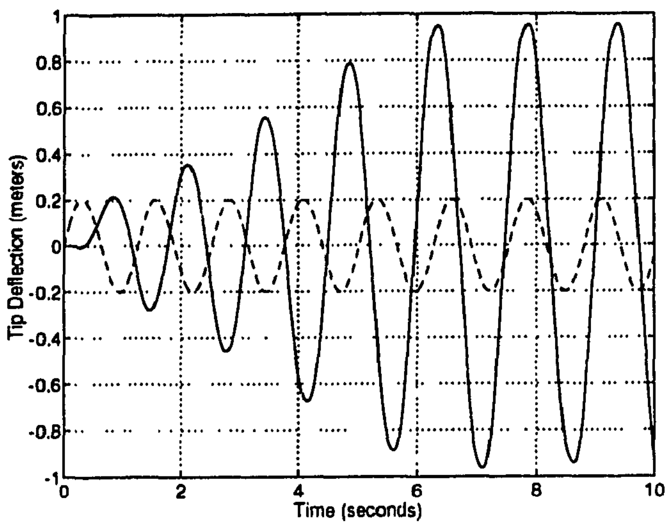


Figure 4.15: Closed-Loop Response $y_{ref}(t) = 0.2 \sin 5t$

We observe a larger error in the first case, and with further increase in frequency the system becomes unstable. These signals essentially excite the higher order modes of the system which were neglected in the derivation of the linearized model. This, in effect, makes the feed-through compensator less effective. In addition, the performance of the neural controller degrades considerably as it was not trained with these high frequency input signals. Consequently, we see poor tracking performance.

4.5 Designing a Robust Servo Controller

The motivation for designing a robust servo controller arises from the fact that any controller design for a multivariable system, based on a linearized model, usually neglects the effect of perturbations in the model, which, in effect, leads to a degradation in the closed-loop performance. We indeed observe this effect in the simulation results based on the scheme described in section 2.6. The steady-state response is significantly oscillatory, and calls for designing a robust servo controller which can reject these disturbances in a more efficient way while ensuring closed-loop stability.

If we review the linearization procedure for the dynamic model presented in section 2.4, it becomes apparent that the high frequency modes (represented by second and higher order derivatives) are omitted in the derivation. Further, the operating point may drift about its nominal value due to perturbations in the system. All these makes it necessary to solve the robust servo-mechanism problem in an efficient way. Our method is adapted from [34] where the *Internal Model Principle* has been applied to design the robust con.roller so that it contains a certain duplicated model of its "environment", i.e. of the disturbances and the reference inputs acting on the system.

Upon examining the closed-loop response of Figure 4.10(a), we can see that the steady state oscillation of the tip position is *almost* sinusoidal in nature of mag-

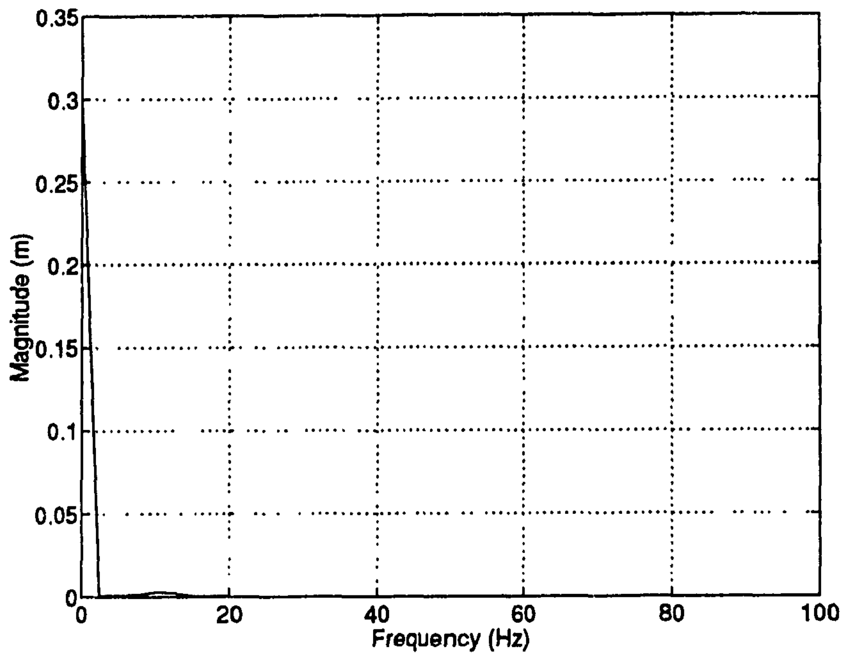


Figure 4.16: Fourier Transform of $y(h, t)$

nitude 0.01 m. In order to observe it more closely, we take the fast Fourier transform of this signal and for a magnitude of 0.01 m, we get the frequency of oscillation around $\omega_n = 5.65$ radians/sec (0.9 Hz.). This is illustrated in Figure 4.16. Although the Fourier transform shows the presence of other harmonics, we would be interested to suppress the most dominant frequency component, i.e. the one at $\omega_n = 5.65$ radians/sec.

Recalling that the Laplace transform of a sinusoidal signal $\sin \omega t$ is $\frac{\omega}{s^2+\omega^2}$, we design our robust servo controller as $G(s) = \frac{K_\omega}{s^2+\omega^2}$, which is essentially a model of the disturbance signal. We will further use a proportional block in parallel with

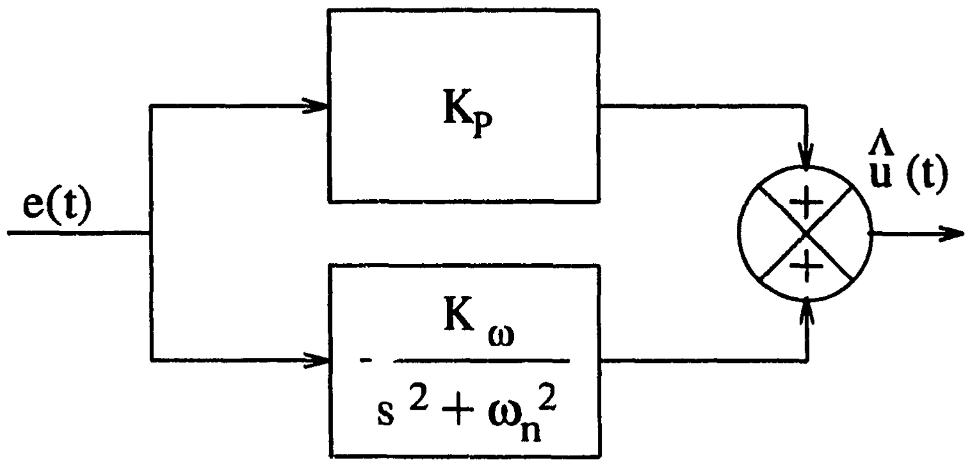


Figure 4.17: Robust Servo Controller

$G(s)$ to reduce the oscillations caused by other harmonics which are not considered in the design of the robust controller. The structure of the controller is illustrated in Figure 4.17. This time we tune the gains K_P and K_ω to the values 7.9×10^5 and 4.2×10^3 respectively. Simulation results show a considerable reduction in oscillation.

4.6 Robust Controller Response

As discussed in the previous section, the robust servo controller has been designed to minimize the effect of disturbances present in the earlier simulations. This has been done using ***internal model principle*** where a model of the disturbance signal has been incorporated in the servo compensator. Figure 4.18(a) and 4.18(b) illustrate the performance of the controller.

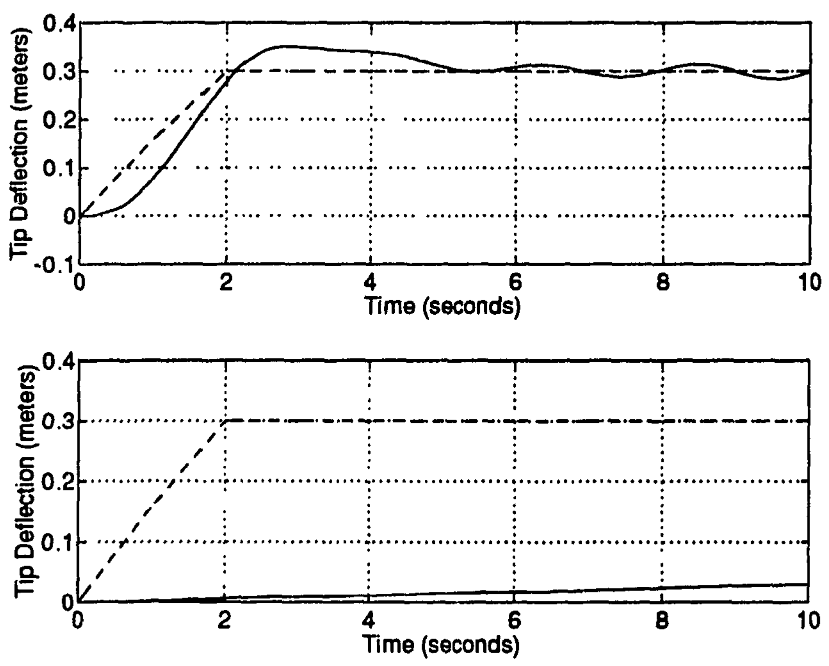


Figure 4.18: (a) Closed-Loop Response of $y(h,t)$ with the Network and (b) Closed-Loop Response of $y(h,t)$ without the Network

When this response is compared with the response in Figures 4.6 and 4.8, we clearly see a reduction in the amplitude and frequency of oscillation.

Chapter 5

CONCLUSIONS AND FUTURE RESEARCH

5.1 Conclusions

In this thesis, we have explored a neural network based controller design that was motivated by difficulties in controlling a flexible-link manipulator subject to unmodeled and/or inaccurately modeled dynamics using only end-point (output) measurements. The closed-loop response of the proposed control strategy has demonstrated conformity to the performance specification, i.e. to track a reference trajectory with reasonable accuracy while ensuring closed-loop stability.

In recent years, several researchers have focused their attention on the development of neural network based modeling and control strategies for a variety of systems which are difficult to control by conventional methods. These difficulties arise from a number of factors such as unmodeled and/or inaccurately modeled dynamics, parametric fluctuations, changes of configuration (due to loading or disturbances) etc. By virtue of their unparalleled ability to learn the dynamics of an arbitrarily complex system with considerable uncertainty, neural networks have proved to be indispensable for controlling these types of systems. In our application, the backpropagation neural network has demonstrated good performance by effectively modeling friction and other unmodeled dynamics present in an actual manipulator. Further, our system is nonminimum phase, and calls for specific measures to ensure closed-loop stability. We have successfully employed a transmission zero assignment technique to achieve this goal.

However, the proposed neural network based control strategy has two fundamental limitations. The first is that it cannot reject disturbances very efficiently. By using a robust servo controller, the problem was solved partially, but it needs further investigation. The second shortcoming is that the controller has a very limited range of operation. Attempting to increase the range often results in a significant change in the linearized model which renders the feed-through compensator less effective. In addition, it may cause a violation of the Euler-Bernoulli constraint for small tip deflections which was assumed in deriving the dynamic model.

Despite the limitations outlined above, the neural network based controller has done a commendable job in controlling the motion of the flexible arm. Considering the inherent complexity involved with this system, and the difficulties that standard techniques have in meeting *all* the design objectives outlined in section 1.1, we believe that neural networks provide a valid approach to the control of flexible-link manipulators.

5.2 Future Research

A particular aspect of the present scheme that deserves further investigation is *robust performance*. The servo compensator should be efficient enough to reject various disturbances that might be present in the system. The present method can be extended to incorporate a neural network based adaptation scheme comprising of two components: a feed-through neural network controller which is trained off-line so as to ensure that the resulting system (consisting of the enhanced model and the feed-through controller) exhibits minimum-phase behavior, and a PID-type neural network based servo-controller whose gains are adjusted on-line. The latter may be constructed with a Competitive Learning Network using Kohonen layer neurons which are capable of performing unsupervised learning in a winner-take-all mode. This scheme is likely to exhibit more robust performance.

Another aspect that could be probed further is to observe the effect of a payload at the tip. The present modeling scheme is incapable of handling a varying payload, but this is an issue of significant practical interest.

Last but not the least, the neural network based control scheme need to be implemented on an actual test-bed to corroborate the theoretical results, and to see if any additional problem crops up or not.

Bibliography

[1] W. J. Book. *Modelling, Design and Control of Flexible Manipulator Arms.* PhD thesis, M. I. T., 1974.

[2] W. J. Book. Analysis of massless elastic chains with servo controlled joints. In *Trans. A.S.M.E. Journal of Dynamic Systems, Measurement and Control*, pages 187–192, September 1979.

[3] W. J. Book. Recursive lagrangian dynamics of flexible manipulator arms. *International Journal of Robotics Research*, pages 87–101, 1984.

[4] W. J. Book, O. Maizzo-Neto, and D. E. Witney. Feedback control of two beam, two joint systems with distributed flexibility. *Trans. A.S.M.E. Journal of Dynamic Systems, Measurement and Control*, pages 424–431, 1975.

[5] W. J. Book and M. Majette. Controller design for flexible distributed parameter mechanical arms via combined state-space and frequency domain

techniques. *Trans. A.S.M.E. Journal of Dynamic Systems, Measurement and Control*, pages 245–254, 1983.

[6] D. Boussalis and S. J. Wang. Multivariable neural network vibration control based on output feedback. In *Second IEEE Conf. on Control Applications*, September 1993.

[7] R. H. Cannon and E. Schmitz. Initial experiments on the end-point control of a flexible one-link robot. *Intl. Journal of Robotics Research*, pages 62–75, 1984.

[8] A. Chaudhuri, R. V. Patel, and K. Khorasani. Neural network based modeling and control of a flexible-link manipulator. In *World Automation Congress*, volume 1, pages 449–454, Maui, Hawaii, August 13-17 1994.

[9] W. Cheng and J. T. Wen. A neural controller for the tracking control of flexible arms. In *IEEE Int. Conf. on Neural Networks*, pages 749–754, 1993.

[10] F. L. Chernous'ko. The dynamics of controlled motions of an elastic manipulator. *Eng. Cybernetics*, pages 101–111, 1981.

[11] X. Cyril, J. Angeles, and A. K. Misra. Dynamics of flexible multibody mechanical systems. In *Trans. of the Canadian Society of Mechanical Engineering*, pages 235–249, 1991.

[12] Anthony R. Fraser and Ron W. Daniel. *Perturbation Techniques for Flexible Manipulators.* Kluwer Academic Publishers, Norwell, MA, 1991.

[13] H. Geniele, R. V. Patel, and K. Khorasani. Control of a flexible-link manipulator. In *Proc. 4th International Symposium on Robotics and Manufacturing*, Santa Fe, NM, November 11-13 1992.

[14] Howard Geniele. Control of a flexible-link manipulator. Master's thesis, Concordia University, Montreal, Canada, 1994.

[15] G. G. Hastings and W. J. Book. Verification of a linear dynamic model for flexible robot manipulators. In *Proc I.E.E.E. Int. Conf. on Robotics and Automation*, 1986.

[16] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Proc. Int. Joint Conf. on Neural Networks*, volume I, pages 593-605, Washington D. C., 1989.

[17] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Tech. rep. cmu-cs-84-119, Carnegie-Mellon University, 1984.

[18] D. C. Hyland. A nonlinear vibration control design with a neural network realization. In *Proc. 29th Conf. on Decision and Control*, December 1990.

[19] H. Kanoh and H. Gil Lee. Vibration control of a one-link flexible arm. In *Proc. 24th Conf. on Decision and Control*, 1985.

[20] P. Karkkainen. Compensation of manipulator flexibility effects by modal space techniques. In *Proc. I.E.E.E. Int. Conf. on Robotics and Automation*, 1985.

[21] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.

[22] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki. Hierarchical neural network model for voluntary movement with application to robotics. *IEEE Control Systems Magazine*, 8:8–15, April 1988.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecci. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[24] A. N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Russian*, pages 953–956, 1957.

[25] M. Lambert. *Adaptive Control of Flexible Systems.* PhD thesis, Oxford University, 1987.

[26] L. Meirovitch. *Analytical Methods in Vibration.* The Macmillan Co., 1967.

[27] L. Meirovitch and H. Baruh. Control of self-adjoint distributed parameter systems. *A.I.A.A.J. Guidance and Control*, pages 60–66, 1982.

[28] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft. Real-time dynamic control of an industrial manipulator using a neural network based learning controller. *IEEE Journal of Robotics and Automation*, 6:1–9, February 1990.

[29] G. Nagathan and A. H. Soni. Nonlinear flexibility studies for spatial manipulators. In *Proc. I.E.E.E. Int. Conf. on Robotics and Automation*, 1986.

[30] W. L. Nelson and D. Mitra. Load estimation and load adaptive optimal control for a flexible robot arm. In *Proc. 25th Conf. on Decision and Control*, 1986.

[31] D. Nemir, A. J. Koivo, and R. L. Kashyap. Control of gripper position of a compliant link using strain gauge measurements. In *Proc. 25th Conf. on Decision and Control*, 1986.

[32] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *International Joint Conference of Neural Networks*, pages 21–26, July 1990.

[33] R. V. Patel and P. Misra. Transmission zero assignment in linear multivariable systems, part ii: The general case. In *Proc. American Control Conf.*, 1992.

[34] R. V. Patel and N. Munro. *Multivariable System Theory and Design.* Pergaman Press, 1982.

[35] D. Psaltis, A. Sideris, and A. A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, pages 17-21, April 1988.

[36] F. Raksha and A. A. Goldenberg. Dynamic modelling of a single-link flexible robot. In *Proc. I.E.E.E. Int. Conf. on Robotics and Automation*, 1986.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representatins by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, chapter 8, pages 318–362. Cambridge: MIT Press, 1986.

[38] E. Schmitz. *Experiments on the End-point Control of a very Flexible One-Link Manipulator.* PhD thesis, Stanford University, 1985.

[39] B. Siciliano, W. J. Book, and G. De Maria. An integral manifold approach to control of a one-link flexible arm. In *Proc. 25th Conf. on Decision and Control*, 1986.

[40] B. Siciliano, B. Yuan, and W. J. Book. Model reference adaptive control of a one-link flexible arm. In *Proc. 25th Conf. on Decision and Control*, 1986.

[41] S. N. Singh and A. A. Schy. Control of elastic robotic systems by non-linear inversion and modal damping. *Trans. A.S.M.E. Journal of Dynamic Systems, Measurement and Control*, pages 180–189, 1986.

[42] M. Stinchcombe and H. White. Approximating and learning unknown mappings using multilayered feedforward networks with bounded weights. In *Proc. Int. Joint Conf. on Neural Networks*, volume III, pages 7–16, San Diego, 1990.

[43] W. Sunada and S. Dubowski. On the dynamic analysis and behavior of industrial robot manipulators with elastic members. *Trans. A.S.M.E. Journal of Mech., Trans. and Aut. in Design*, pages 42–51, 1983.

[44] A. Truckenbrodt. Truncation problems in the dynamics and control of flexible mechanical systems. In *Proc. I.F.A.C. 5th Triennial World Congress on Control Science and Technology*, 1981.

[45] P. B. Usoro, R. Nadira, and S. S. Mahil. Control of lightweight flexible manipulator arms: A feasibility study. Technical report, N. S. F., 1986.

[46] P. B. Usoro, R. Nadira, and S. S. Mahil. A finite element/lagrange approach to modelling lightweight flexible manipulators. *Trans. A.S.M.E. Journal of Dynamic Systems, Measurement and Control*, pages 198–205, 1986.

[47] P. D. Wasserman. *Neural Computing: theory and practice.* Van Nostrand Reinhold, New York, 1989.

[48] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[49] V. Zeeman, R.V. Patel, and K. Khorasani. A neural network based control strategy for flexible joint manipulators. In *Proc. 28th IEEE Conf. on Decision and Control*, pages 1759–1764, Tampa, FL, 1989.

[50] V. Zeeman, R.V. Patel, and K. Khorasani. A neural network controller for flexible joint manipulators. In *Proc. American Control Conf.*, San Diego, CA, 1990.

[51] Vladimir Zeman. A neural network based approach to the control of flexible joint manipulators. Master's thesis, Concordia University, Montreal, Canada, 1991.