1996

Neural Network Implementation of Non Linear Control Using Radial Basis Functions

Francis Andersson
University of Rhode Island

NEURAL NETWORK IMPLEMENTATION OF NON LINEAR CONTROL USING RADIAL BASIS FUNCTIONS

BY

FRANCIS ANDERSSON

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

1996

MASTER OF SCIENCE THESIS

OF

FRANCIS ANDERSSON

APPROVED:

Thesis Committee

Major Professor

R. J. Vaccaro

A. R. Lindgren

James D. Kowalski

Thomas J. Rochett

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

1996

Abstract

This research is concerned with the design of radial basis function neural networks to implement a controller for nonlinear systems. Nonlinear systems are of particular interest given the fact that most real life systems are nonlinear in nature and control schemes for such systems are not as developed as their linear counterparts and involves a lot of heuristics. We show the ability of *radial basis function networks* (RBF) to serve as a single unifying model incorporating both nonlinear and linear methodologies.

We focus on the problem of the inverted pendulum on a cart system, which is a classic problem in a lot of control literatures. The problem is to swing the pendulum from a given initial state, which is typically the hanging down position, to the up position and then to keep it balanced in the up position. In swinging the pendulum, the cart to which the pendulum is attached is moved back and forth on a track until the pendulum is in the up position. This system is a very useful model in that it demonstrates a multi-variable highly nonlinear system that belongs to a class of nonlinear systems that cannot be controlled by traditional nonlinear techniques such as *feedback linearization.*

In training the RBF network, we explore several different control schemes to produce the training data. These control schemes could also be easily extrapolated to work with other multi-variable nonlinear systems. We first design a neural controller for the second order system describing the pendulum dynamics only. The controller is able to drive the state variables from any permissible state of the system to zero, and to keep it stabilized in that equilibrium state. Secondly, we again show the network's ability to implement nonlinear control of the fourth order pendulum/cart system. We

further demonstrate how the Kohonen self organizing feature map algorithm can be used to make the network more efficient and adaptive.

Acknowledgments

Firstly, I would like to thank God for His many blessings and uplifting me in those times when the problems seemed insurmountable. Secondly, many thanks to my advisor, Dr. Richard Vaccaro, for his advice and optimism throughout this project. Also, I would like to express my gratitude to Dr. Allen Lingren and Dr. James Kowalski for their valuable inputs in this research and for serving on my thesis committee. Finally, special thanks to my family for their encouragement and moral support during this project.

Contents

List of Figures

Chapter 1

Introduction

The work reported in this thesis represents a subset of a rapidly growing interest in the use of neural networks as a paradigm for the control of nonlinear systems or the representation of systems in system identification problems. Recent research [1], [2],[3],[4], [5], [6] has attempted to define the role of neural networks in control theory. A major focus of these research efforts has been to establish a mathematical formulation of these network architectures through which a general control methodology could be developed. It is worth noting that neural networks are natural to use for nonlinear control and identification methods due to the fact that these networks lend themselves easily in performing *nonlinear* mappings in multidimensional space. Assuming that there exists an input-output mapping that achieves the control objective, the network is trained in a *supervised* fashion by modifying the synaptic weights so as to minimize the difference between the desired response and the actual response produced by the input signal. In addition, neural networks are able to *generalize*; generalization refers to the networks producing reasonable outputs for inputs not encountered during training (learning). This implies that neural networks are *robust* and can be easily retrained to *adapt* their synaptic weights to compensate for minor changes in the environmental conditions under which they are operating. In this day and age of parallel and distributed computation, these networks lend themselves easily as practical tools both in hardware and software implementations. In the hardware form neural networks have the potential to be inherently *fault tolerant* in the sense

that its performance is gracefully degraded under adverse operating conditions. For example, if a neuron or its connecting links are damaged, recall of a stored pattern is impaired in quality. However, due to the distributed nature of information in the network, the damage has to be extensive before the overall performance of the network is degraded seriously.

Consider a system described by the following equations;

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}) + g(\mathbf{x})u \\ y &= h(\mathbf{x}) \end{aligned} \tag{1.1}$$

where $\mathbf{x}$ is a vector-valued state vector, $f(.)$, $g(.)$ and $h(.)$ are nonlinear functions and u and y are the input and output of the system, respectively. The objective of the control problem is to determine the input, u, so that the system behaves in a desired fashion. For example, there are two ways in which the system (plant) can be controlled: regulation and tracking. In the former, the main goal is to stabilize the plant around a fixed operating point, typically referred to as an equilibrium point. In the latter, the aim is to make the output, y, follow an input signal asymptotically.

As stated above the goal of the control problem is to make the plant behave in a certain, deterministic way. But the nature of the plant itself provides the framework for the control mechanism. Systems are generally characterized as being either linear or nonlinear. Linear control techniques have over the past few decades been well documented and successfully implemented [7] and [8]. There exists numerous linear controllers (eg. pole placement, PID, LQR, H_∞, etc.) available for linear systems. Nonlinear control, on the other hand, is not as developed and most designs of controllers still rely heavily on heuristic methods. Probably the most well-known nonlinear control scheme has been *feedback linearization* in which the system is made to be locally equivalent to a linear system, after which linear control techniques can be utilized. However, feedback linearization can be applied only to a limited class of systems (eg. the Inverted Pendulum system does not fall in this class [9]). This presents a challenge to the controls engineer because most systems of interest (eg. robotic arms, helicopters, etc.) are nonlinear in nature. It is against this backdrop that alternative methods such as neural networks have been explored.

An obvious disadvantage of neural networks is that they are highly nonlinear in their parameters. Learning must be based on nonlinear optimization techniques, and parameter estimates may become trapped at a local minimum of the chosen optimization criterion during the learning procedure when a gradient descent algorithm is used. [10] demonstrates this disadvantage. Other optimization techniques, such as genetic algorithm, Newton's method and simulated annealing, although capable of achieving a global minimum, require extensive computation.

A viable alternative to highly nonlinear-in-the-parameter neural networks is the radial basis function (RBF) network. An RBF network can be regarded as a special two-layer network which is linear in the parameters by fixing all RBF centers and nonlinearities in the hidden layer. Thus the hidden layer performs a fixed nonlinear transformation with no adjustable parameters and it maps the input space onto a new space. The output layer then implements a linear combiner on this new space and the only adjustable parameters are the weights of this linear combiner. These parameters can therefore be determined using the linear least square (LS) method, [11] [3]. An RBF network that performs the mapping $f_r : \mathbf{R^n} \rightarrow \mathbf{R}$ is represented by the following;

$$f_r(\mathbf{x}) = \mathbf{w_o} + \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{w_i}\phi(\|\mathbf{x} - \mathbf{c_i}\|) \tag{1.2}$$

where $\mathbf{x} \in \mathbf{R^n}$ is the input vector, $\phi(\cdot)$ is the nonlinear activation function, w_i are the weights, and $\mathbf{c_i} \in \mathbf{R^n}$ are the known RBF centers.

1.1 Problem Statement

As stated previously, the objective of a controller is to produce an input signal, preferably an optimal one, that would move the states of the system in desired trajectories. The goal of this research is to design a neural controller for the inverted pendulum system that *would swing the pendulum from any given state in its allowable state space to the equilibrium state of standing upright and drive all other state variables to zero.* Furthermore, the controller should be self correcting given any perturbation. This goal exceeds that of [4] which is to get the pendulum from the equilibrium state

of hanging down to the target equilibrium state of standing erect.

The mathematical model used for the inverted pendulum system is given by Vaccaro, [7]. A schematic of this system is shown in Fig. 1.1.

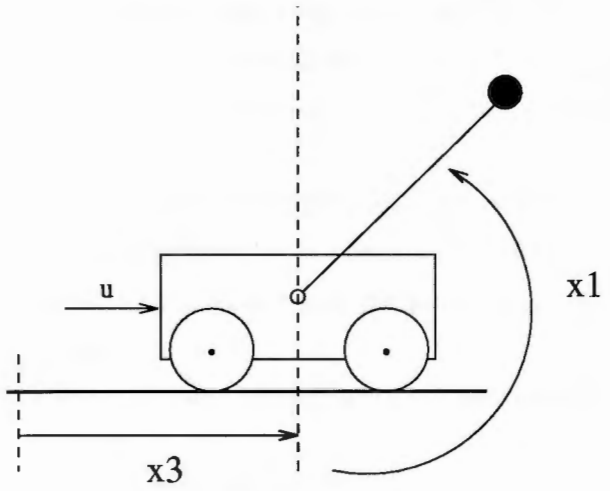

Figure 1.1: The inverted pendulum on a cart system

The nonlinear system model is represented by the following differential equations;

$$\dot{\mathbf{x}} = f(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} \tag{1.3}$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) \tag{1.4}$$

where:

$$f(\mathbf{x}) = \begin{bmatrix} x_2 \\ -A\sin(x_1) + \frac{A}{ng}\cos(x_1)Cx_4 \\ x_4 \\ -Cx_4 \end{bmatrix} \tag{1.5}$$

$$g(\mathbf{x}) = \begin{bmatrix} 0 \\ -\frac{A}{ng}\cos(x_1)D \\ 0 \\ D \end{bmatrix} \tag{1.6}$$

$$h(x) = \begin{bmatrix} x_1 \end{bmatrix} \tag{1.7}$$

$$
\begin{aligned}
A &= 23.1\ rad/sec_2 \\
C &= 25.0\ sec^{-1} \\
D &= 2{,}633\ rad/(volt - sec^2) \\
n &= 495\ \text{rad/sec} \\
g &= 9.81\ m/s^2
\end{aligned}
$$

where x_1 is the angular position of the pendulum, x_2 is the angular velocity of the pendulum, x_3 is the motor position, x_4 is the motor velocity and u is the control input. The standard orientation is that when the pendulum is hanging down, $x_1 = 0$ and when it is pointing up, $x_1 = \pi$.

The linearized state-space model of the inverted pendulum/cart system is given by;

$$
\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -A & 0 & 0 & \frac{AC}{ng} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -C \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{AD}{ng} \\ 0 \\ D \end{bmatrix} u(t) \tag{1.8}
$$

1.2 An Alternate Neural Network Approach to the Pendulum-cart Problem

In the paper by Suykens et al [4], a control law is proposed using either a feed-forward or recurrent neural networks to switch a multi-variable nonlinear plant between equilibrium points and to stabilize the plant at the target equilibrium point. This network design incorporates a linear controller to stabilize the plant at the target equilibrium point. As an illustration of this control strategy Suykens uses the inverted pendulum-cart system in which the task is to swing the pendulum from down to up and to locally stabilize it in the up position. In this section we present only the feedfoward network design given in that paper, as well findings of [10] in designing and implementing this network to control the pendulum-cart system.

In order to determine the weights, Suykens suggests an optimization scheme in which optimal weights are found such that a cost function is minimized using a steepest gradient descent algorithm (e.g *Constr* in Matlab). This training scheme does not require that an input-output mapping is known beforehand, but instead determines the mapping during training.

Given a nonlinear system, as in (1.1), the optimal control problem is to minimize a cost function over the weights of the network. The cost function is given by

$$C = \|\mathbf{x}(T)\| + \int_0^T \zeta(\mathbf{x}(t))dt \tag{1.9}$$

where T is the final time, $\zeta(\mathbf{x}(t)) = \mathbf{x}(t)^T\mathbf{x}(t)$ (quadratic control) or $\zeta = 0$ (terminal control).

The input-output relationship of the neural network is given by

$$u = \alpha \tanh(\mathbf{w}^T \tanh(\mathbf{V}\mathbf{x})) \tag{1.10}$$

where α is the maximum amplitude of the control signal, $\mathbf{w}^T$ is the weight vector for the output layer, $\mathbf{V}$ is the weight matrix for the input layer. In the linear region the tanh can be dropped because $\tanh x \approx x$ if x is small. Therefore (1.10) in the linear region can be written as

$$u = \alpha\mathbf{w}^T\mathbf{V}\mathbf{x} \tag{1.11}$$

If we let $\mathbf{w}^T$ be a function of $\mathbf{V}$ such that

$$\mathbf{w}^T = -\frac{\mathbf{L}}{\alpha}\mathbf{V}^{-1} \tag{1.12}$$

then (1.11) simplifies to

$$u = -\mathbf{L}\mathbf{x} \tag{1.13}$$

where $\mathbf{L}$ is a row vector of full state feedback regulator gains. Note that eqn.[1.13] is the standard expression for a full state feedback regulator. This gains vector can be calculated according to linear control theory such as pole placement (see appendix A), PID, LQR etc...

The inverted pendulum mathematical model used by Suykens is given by;

$$f(\mathbf{x}) = \begin{bmatrix} x_2 \\ \frac{\frac{3}{4}mlx_4^2 \sin x_3 - \frac{mg}{2}\sin 2x_3}{\frac{4}{3}m_t - m\cos^2 x_3} \\ x_4 \\ \frac{m_t g \sin x_3 - \frac{ml}{2}x_4^2 \sin(2x_3)}{l(\frac{4}{3}m_t - m\cos^2 x_3)} \end{bmatrix} \tag{1.14}$$

$$g(\mathbf{x}) = \begin{bmatrix} 0 \\ \frac{4}{3}\frac{1}{\frac{4}{3}m_t - m\cos^2 x_3} \\ 0 \\ -\frac{\cos x_3}{l(\frac{4}{3}m_t - m\cos^2 x_3)} \end{bmatrix} \tag{1.15}$$

$$h(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \tag{1.16}$$

where x_1 is motor position, x_2 is motor velocity, x_3 is pendulum position, x_4 is pendulum velocity, m is the mass of pendulum and equals 0.1kg, m_t is the total mass of the pendulum and cart and equals 1.1kg, l is the half pole length and equals 0.5m, and g is the acceleration due to gravity.

Given this pendulum model Suykens and his colleagues were able to demonstrate the ability of the feedfoward neural network to swing the pendulum from down to up and to stabilize the pole at the up position using the *linear quadratic regulator (LQR)* from linear control theory. A diagram of this feedforward network is shown is Fig. 1.2. In [10] the authors highlight the difficulties with this optimization based controller. Their results indicate the weight matrix, $\mathbf{V}$, is highly dependent on the system parameters l, m_t, and m. For example, the authors were able to determine the optimal weights that balanced the pendulum cart model used by Suykens, but when the pendulum half length was changed from 0.5m to 0.55m or the total mass from 1.1kg to 1.25kg the controller could not achieved the control objective given the same weight matrix $\mathbf{V}$. Further results showed that the authors were unable to find the optimal weights to swing up and balance the pendulum-cart model given by Vaccaro (see (1.5)).

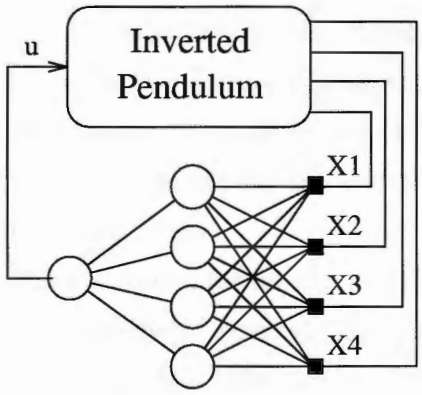

Figure 1.2: Inverted pendulum controlled by a feed forward neural network.

In concluding this section, let us review the pros and cons of this neural control law,

Pros The incorporation of the linear controller into the neural network is seamless and is mathematically well-defined. In addition, the optimal control input is determined during training of the network which means that input-output mappings are not needed in training like it would be if backpropagation or a least square training method had been used.

Cons There is no guarantee a priori that the linearized region will be entered. The network may become stuck at a local minimal depending on the initial weights matrix chosen or the step size of the gradient descent algorithm. Furthermore, there is no reliable way to pick this initial weight matrix. Suykens suggest using a random martix which is normally distributed with a variance between 0 and 1. But as discussed in [10] this rule of thumb choice is not at all reliable.

1.3 Organization of Chapters

In chapter two, we present the mathematical model for *radial basis neural networks* and demonstrate their ability to generalize. Also, we show how these networks can be more efficient by using *self-organizing neural networks* to determine their centers.

In chapter three, we design a controller for the second order model for the pendulum system. We explore the use of the *dynamic programming algorithm* along with techniques from linear control theory to produce the training data for the network and we investigate two schemes for placing the centers, fixed selection of the centers and self-organized selection of the centers.

In chapter four, we present the controller for the fourth order model of the pendulum system. We utilize a new method to generate the training data for the network using the energy information of the system. Results are given using the self-organizing placement of centers scheme discussed in the previous two chapters.

Finally in chapter five, we summarize our work and propose ideas for future work.

Chapter 2

Neural Network Implementation of Nonlinear Functions

2.1 Introduction

In this chapter we approach the design of a neural network as a *curve-fitting or approximation problem* to implement nonlinear functions in a high dimensional space. According to this design strategy, training the network is equivalent to finding a surface in multidimensional space that provides the best fit to the training data. Correspondingly, generalization is equivalent to the use of this multidimensional surface to interpolate the test data.

On a historical note, Broomhead and Lowe,[12], were the first to use radial basis functions in the design of neural networks. Other major contributions to the theory, design, and application of radial basis function networks include works by Moody and Darken, [13], Poggio and Girosi, [14] and Chen, [11].

2.2 Radial Basis Function Networks

The construction of a *radial basis function network* in its most basic form involves three different layers as shown in Fig. 2.1. The input layer is made up of source nodes. The second layer is a hidden layer of high enough dimension and constitutes

The problem is to find an approximation $\hat{Y} = \hat{F}(\mathbf{x})$ of the mapping (2.1) value for any argument $\mathbf{x} \in \rho$.

From numerical analysis we know that the most convenient way of representing an unknown nonlinear function is to present it as a linear expansion

$$\hat{\mathbf{Y}} = \hat{F}(\mathbf{x}) = \sum_{i=1}^{N_a} w_i \phi(\|\mathbf{x} - \mathbf{c}_i\|) \tag{2.3}$$

where $\{\phi(\|\mathbf{x} - \mathbf{c}_i\|) | i = 1, 2, ..., N_a\}$ is a set of N_a *radial basis functions*, $\|.\|$ denotes the Euclidean norm, w_i are weights of the expansion, and $\mathbf{c}_i \in \mathbf{R}^p, i = 1, 2, ..., N_a$ are the centers of the radial basis functions.

Two commonly used radial basis functions are

1. *Inverse multiquadrics*

$$\phi(x) = \frac{1}{(x^2 + c^2)^{\frac{1}{2}}} \tag{2.4}$$

2. *Gaussian functions*

$$\phi(x) = exp(-\frac{(x - c)^2}{2\sigma^2}) \tag{2.5}$$

Theoretical investigations and practical results suggest that the choice of radial basis functions is not crucial to the performance of the RBF network [11]. Our choice of radial basis functions in this thesis is that of the Gaussian functions, which is generally expressed as

$$\phi(\|\mathbf{x} - \mathbf{c}_j\|_{\mathbf{\Sigma}_i^{-1}}) = exp[-\frac{1}{2}(\mathbf{x} - \mathbf{c}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \mathbf{c}_i)] \tag{2.6}$$

where $\mathbf{\Sigma}^{-1}$ is the inverse covariance matrix of the Gaussian distribution and can be expressed in terms of a norm weighting matrix $\mathbf{C}_i$, [16], [17]

$$\mathbf{\Sigma}_i = \mathbf{C}_i^T \mathbf{C}_i \tag{2.7}$$

2.2.1 Exact RBF Network

In the exact RBF network implementation of the mapping in (2.1), we set $N_a = N$ and we take the known data points $\mathbf{x}_i, i = 1, 2, ..., N$ to be the centers of the radial

basis functions. We can therefore rewrite (2.3) as

$$\hat{Y}_i = \hat{F}(\mathbf{x}_j) = \sum_{j=1}^{N} w_i \phi(\|\mathbf{x}_j - \mathbf{x}_i\|) \quad i = 1, 2, ...N \tag{2.8}$$

Equation (2.8) can be expressed in matrix notation as

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} \tag{2.9}$$

where

$$\phi_{ji} = \phi(\|\mathbf{x}_j - \mathbf{x}_i\|), \quad j, i = 1, 2, ..., N \tag{2.10}$$

Let

$$\begin{aligned} \mathbf{Y} &= [Y_1, Y_2, ..., Y_N]^T \\ \mathbf{\Theta} &= [w_1, w_2, ..., w_N]^T \\ \mathbf{\Phi} &= \{\phi_{ji} | j, i = 1, 2, ..., N\} \end{aligned} \tag{2.11}$$

Then we can expressed (2.9) in a more compact form

$$\mathbf{Y} = \mathbf{\Phi}\mathbf{\Theta} \tag{2.12}$$

The N-by-1 vectors $\mathbf{Y}$ and $\mathbf{\Theta}$ represent the desired response vector and linear weight vector respectively. The N-by-N matrix $\mathbf{\Phi}$ is the interpolation matrix.
If $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ *are distinct points in* $\mathbf{R}^p$*, then the interpolation matrix,* $\mathbf{\Phi}$*, is positive definite [18].* Provided that this statement is true, we can obtain the weight vector $\mathbf{\Theta}$ by

$$\mathbf{\Theta} = \mathbf{\Phi}^{-1}\mathbf{Y} \tag{2.13}$$

2.2.2 Generalized RBF Network

In the generalized RBF network implementation of (2.1), we set $N_a \leq N$ and we consider that the centers of the network do not necessarily coincide with the training data points. The network expansion is depicted in (2.3).

Assuming the networks centers are known and fixed, let us fit the training set data in (2.2) using the network in eqn. [2.3]. Utilizing the same notations in (2.11) and (2.12), we can represent the fitting problem in the regression form

$$\mathbf{Y} = \mathbf{\Theta\Phi} + \varepsilon, \quad \mathbf{\Phi} = \{\phi(\|\mathbf{x}_j - \mathbf{c}_i\|)\}_{j,i=1}^{N,N_a} \tag{2.14}$$

where $\varepsilon = \begin{bmatrix} e_1 & \dots & e_N \end{bmatrix}^T$ is a residual error vector. Since $\mathbf{\Phi}$ is not guaranteed to be well conditioned or even a full rank matrix, we will look for a regularized least squares solution to (2.14) that minimizes

$$\|\varepsilon\|_F^2 + \alpha\|\mathbf{\Theta}\|_F^2, \quad 0 < \alpha \leq 1 \tag{2.15}$$

where $\|.\|$ is the Frobenius norm and α is a scalar regularization parameter introduced to compensate for ill-conditioned problems. Solving eqns.[2.14] and [2.15] yields

$$\mathbf{\Theta} = \mathbf{Y}\Phi^T(\alpha\mathbf{I} + \Phi\Phi^T)^{-1} \tag{2.16}$$

2.3 Self Organizing Feature Map Networks

The performance of an RBF network critically depends upon the chosen centers. The RBF centers should suitably sample the input domain of the network and reflect the data distribution. Furthermore, due to obvious reasons in considering real time implementations of these networks, it is prefered to have as few basis functions (reduction of the dimensionality of the hidden layer space) as possible- hence reducing the computational time of the network. But the question arises as how to best select appropriate centers.

In this section we propose using a *self-organizing feature-mapping (SOFM) algorithm*, developed by Kohonen [19] in which the topography of the input domain is learned in an *unsupervised* fashion and the centers of the RBF network are then taken to be the weights of the SOFM. The SOFM algorithm draws striking resemblance to the k-means clustering algorithm, which is well documented in a lot of pattern classification literature [5]. To begin the discussion of the SOFM algorithm let us

define an input matrix, $\mathbf{X}$, representing the set of input vectors over time, denoted by

$$\mathbf{X} = \left[\begin{array}{cccc} \mathbf{x}_1, \mathbf{x}_2 & ... & \mathbf{x}_N \end{array} \right] \tag{2.17}$$

where

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix}, \quad i = 1, 2, ..., N$$

and synaptic weight matrix denoted by

$$\mathbf{W} = \left[\begin{array}{cccc} \mathbf{w}_1 & \mathbf{w}_2 & ... & \mathbf{w}_Q \end{array} \right] \tag{2.18}$$

where

$$\mathbf{w}_j = \begin{bmatrix} w_{jp} \\ \vdots \\ w_{jp} \end{bmatrix}, \quad j = 1, 2, ..., Q$$

Note that $Q \leq N$ where Q is the number of neurons (centers) and N is the number of training input data vectors

To find the best match of the input vector $\mathbf{x}$ with the weight vectors $\mathbf{w}_j$, we define the best matching criterion to be the *minimum Euclidean distance* between vectors.

$$i(\mathbf{x}_i) = arg \min_j \|\mathbf{x}_i - \mathbf{w}_j\|, \quad j = 1, 2, ..., Q, \;\; i = 1, 2, ...N \tag{2.19}$$

where $i(\mathbf{x})$ is the index that identifies the neuron that best matches the input vector. This neuron is classified as the the winning neuron and is part of a topological neighborhood, denoted by $\Lambda_{i(\mathbf{x})}(n)$. An example of a neighborhood topology is illustrated in Fig. (2.2). Given this winning neuron, the idea is then to adjust it along with its neighboring neurons to move closer to the input vector in a Euclidean sense. Kohonen's SOFM algorithm is summarized by the following steps;

1. *Initialization.* Choose random values for the initial weight vectors $\mathbf{w}_j(0)$. The only restriction here is that the $\mathbf{w}_j(0)$ be different for $j = 1, 2, ..., Q$, where Q is the number of neurons.

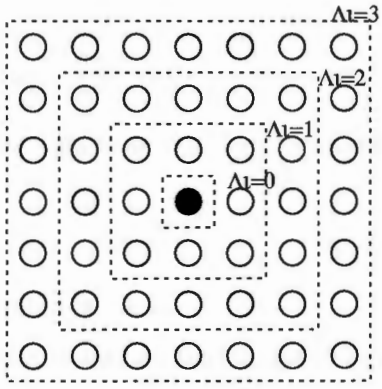

Figure 2.2: Square topological neighborhood Λ, of varying size, around "winning" neuron, identified as black circle

2. *Sampling.* Draw a sample $\mathbf{x}$ from the input distribution with a certain probability.

3. *Similarity Matching.* Find the best matching (winning) $i(\mathbf{x})$ at time n, using

$$i(\mathbf{x}) = arg\min_j \|\mathbf{x} - \mathbf{w}_j\|, \qquad j = 1, 2, ..., N$$

4. *Updating.* Adjust the weight vectors of all neurons, using the update formula

$$\mathbf{w}_j(n+1) = \begin{cases} \mathbf{w}_j(n) + \eta(n)[\mathbf{x}(n) - \mathbf{w}_j], & j \in \Lambda_{i(\mathbf{x})}(n) \\ \mathbf{w}_j(n), & \text{otherwise} \end{cases}$$

where $\eta(n)$ is the learning rate parameter, and $\Lambda_{i(\mathbf{x})}(n)$ is the neighborhood function centered around the winning neuron.

5. *Continuation.* Continue with step 2 until no noticeable changes are observed.

2.4 A Nonlinear Function Implementation Example

Let us consider a system of equations given by

$$x_1(t) = e^{-\frac{1}{2}t}\cos(t) \tag{2.20}$$

$$x_2(t) = \dot{x}_1(t) = -\frac{1}{2}e^{\frac{1}{2}t}[\cos(t) + \sin(t)] \tag{2.21}$$

where

$$t = \begin{bmatrix} 0 & 0.1 & \ldots & 9.9 & 10 \end{bmatrix}$$

and let us also define another function u to be the linear combination of x_1 and x_2

$$u = x_1 + x_2 \tag{2.22}$$

The task is to obtain a mapping $\mathbf{x} \rightarrow u$.

In order to achieve this mapping, we first utilize the SOFM network to determine the locations of the centers of a generalized RBF network. We batched the input vectors as in (2.17)

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_{101} \end{bmatrix}$$

We trained the SOFM network using the 101 data points for 1000 epochs[1] to produce 15 center locations. Fig (2.3) shows the result of the training of the SOFM network. Note how the SOFM network places more centers in the area of greater transient activities and fewer centers when the transient activities are smaller and gradually decaying.

Now that we have determined and fixed the centers, we obtain the weights of the generalized RBF network by solving (2.16). To get the interpolation matrix $\mathbf{\Phi}$, we fixed the covariance matrix from (2.6) and (2.7) to be $\mathbf{\Sigma} = dI_2$, where $d = 0.8271$. d is taken to be the average distance between the neighboring center nodes [3]. We set $\alpha = 0$, because the interpolation matrix is well-conditioned. Figure 2.4 shows the RBF network's approximation to (2.22).

[1] An epoch is the presentation of the set of training (input and/or target) vectors to a network and the calculation of new weights and biases. Note that the training vectors may be presented one at a time or all together in a batch.

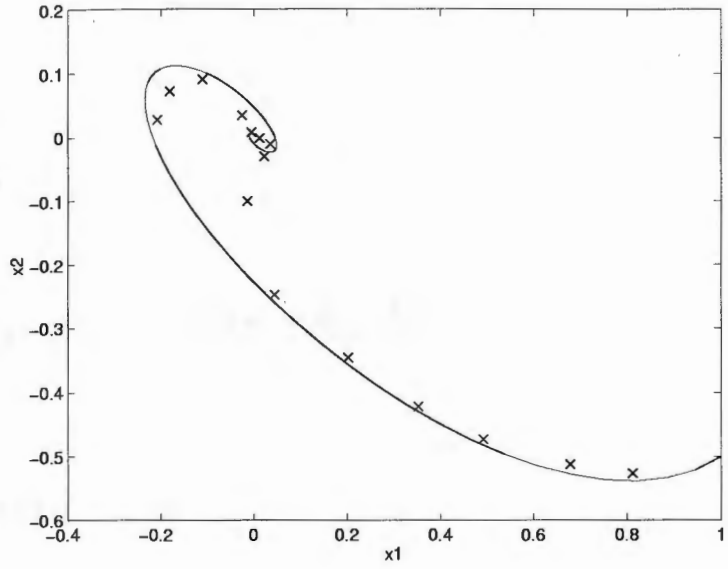

Figure 2.3: Placement of centers by SOFM along the input trajectory where 'x' show the center locations

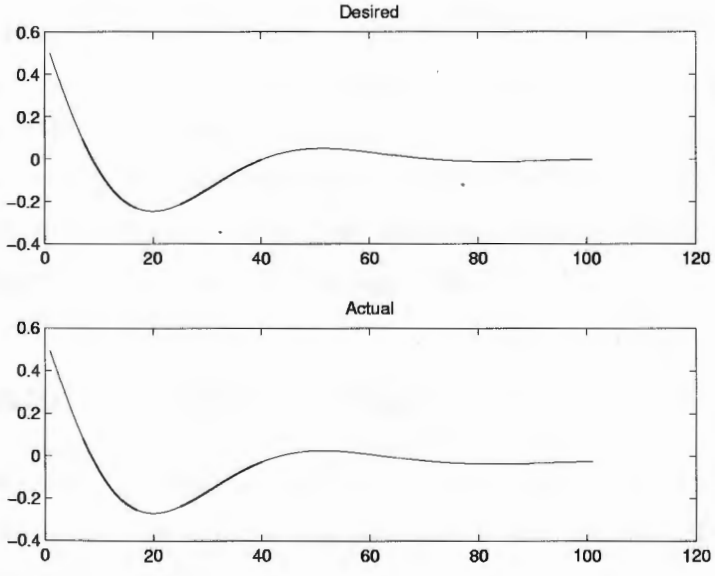

Figure 2.4: Illustration of the approximating ability of the RBF network with centers determined by the SOFM network

Chapter 3

The Second Order Model

3.1 Introduction

Given the equations for the pendulum without regard to the dynamics of the cart

$$\begin{aligned} \dot{x_1}(t) &= x_2(t) \\ \dot{x_2}(t) &= -A\sin(x_1(t)+\pi) - \frac{A}{g}\cos(x_1+\pi)\hat{u}(t)) \end{aligned} \tag{3.1}$$

and where $\hat{u}(t)$ is the equivalent to the acceleration of the cart, we would like in this chapter to design the RBF network such that it is optimal and balances the pendulum in the upright position given any initial state. For ease of computation the orientation of the pendulum is changed from the hanging down position $x_1 = 0$ to $x_1 = \pi$, and the upright position $x_1 = \pi$ to $x_1 = 0$. The training data is generated by the *dynamic programming algorithm* developed by Richard Bellman [20].

3.2 Dynamic Programming

The method of dynamic programming is a process by which the performance measure of a system is minimized by using a concept called the *principle of optimality.* This principle is described as, [21];

> An optimal policy that has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal

policy with regard to the state resulting from the first decision.

The dynamic programming algorithm is summarized as follows;

1. Quantize the state space, considering the maximum possible ranges of state values. Each state variable, x, is quantized as follows;

$$\begin{aligned} x &\in (x_{min}, x_{max}) \\ x &= x_{min} + k\Delta x \quad k = 0, 1, ..., M, \quad M = \frac{x_{max} - x_{min}}{\Delta x} \end{aligned} \tag{3.2}$$

   This requires an *a priori* knowledge of the state space.

2. Quantize the control effort (input), considering the maximum allowable range.

$$\begin{aligned} \hat{u} &\in (\hat{u}_{min}, \hat{u}_{max}) \\ \hat{u} &= \hat{u}_{min} + k\Delta\hat{u} \quad k = 0, 1, ..., M, \quad M = \frac{\hat{u}_{max} - \hat{u}_{min}}{\Delta\hat{u}} \end{aligned} \tag{3.3}$$

   This range of the input signal could represent hardware limitation(s) of the controller.

3. Determine the value of the performance index (cost function) of all quantized states at the final time, N.

$$J_{NN}(\bar{x}(N)) = \bar{x}^T(N)W\bar{x}(N) \tag{3.4}$$

   where W is a penalty matrix.

4. Go back one time index, for each quantized state determine the optimal control input vis-a-vis the optimal cost for all quantized control inputs. The optimal cost at this point is the smallest sum of the cost of being in the present state plus the cost of the states arrived at given all possible inputs. If the state arrived at is not a quantized state then interpolating costs from adjacent quantized states are used to determined its cost. This is shown by the recurrence equation below;

$$J^*_{N-k,N}(\bar{x}(N-k) = \min_{u(N-k)} [J_{k,N}(\bar{x}(N-k), u(N-k)) + J^*_{N-(k-1),N}(\bar{x}(N-k+1))] \tag{3.5}$$

5. Repeat step 4 until the time index is zero. For each state at each time index, the optimal control input is stored in a table or matrix.

3.3 Results from Dynamic Programming and the RBF Network

For the pendulum system, the following parameters were used; $x_{1max} = 5$ rad, $x_{1min} = 0$ rad, $\Delta x_1 = 0.5rad$, $x_{2max} = 5\frac{rad}{s}$, $x_{2min} = -8\frac{rad}{s}$, $\Delta x_2 = 0.5\frac{rad}{s}$, $\hat{u}_{max} = 8\frac{m}{s^2}$, $\hat{u}_{min} = -8\frac{m}{s^2}$ and $\Delta\hat{u} = 0.5\frac{m}{s^2}$

The performance index is given by;

$$J = \bar{x}^T(N)W\bar{x}(N) + \sum_{k=1}^{N-1}(\bar{x}^T(k)Q\bar{x}(k) + u^2(k)) \tag{3.6}$$

where $N = \frac{T}{\Delta T}$, $T = 3$ seconds (final time), $\Delta T = 0.2$ second (sampling interval). The penalty matrices W and Q were defined as

$$W = 10^3 \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}.$$

In this section we use the look-up table from dynamic programming, which contains optimal trajectories from all permissible states at any time index, along with a linear controller to train the RBF network. The insertion of the linear controller ensures that the system remains stabilized when face with small perturbations. The linear controller of choice is a vector of gains that places the closed loop poles of the system within the unit circle (see appendix A for details) The following steps illustrate the process;

1. Select an initial state and then look up in the table for the corresponding input value. If the initial state is not a quantized state then interpolate inputs at neighboring quantized states for input. Use this input to determine the next state and then repeat the process until the state variables are within the linear region.

2. Once inside the linear region, switch to the linear controller. Let the system run with the linear controller for a while so that the training data contain a few presentations in the linear region.

3. Repeat the two steps above starting with different initial conditions to obtain different trajectories. The objective is to have these trajectories span as much of the state space as possible. Figure 3.1 show the simulation results for trajectories of six different initial states, $\{x_1, x_2 | (3,0), (1.5,-8), (3.5,4), (2.5,-5), (0.4,-0.1), (1.5,-2)\}$.

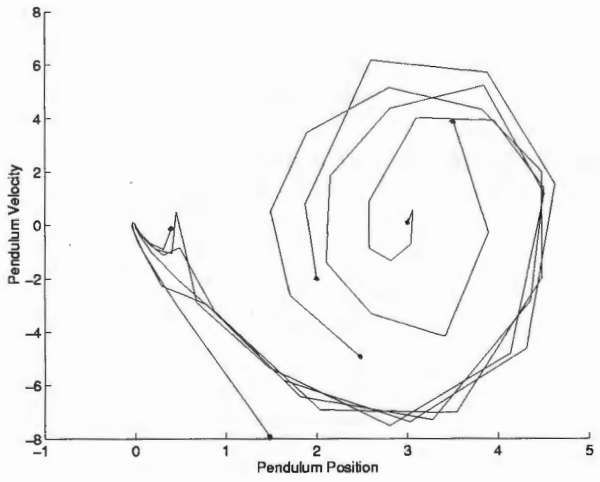

Figure 3.1: Training trajectories for different initial conditions.

4. Use these optimal trajectories to train the network by setting up input and target matrices and solving for the weights in a least squares sense.

5. Design and insert a linear controller for the linear region to guarantee that the states become stabilized once they enter the linear region (see appendix A for the design for the pole placement linear controller). To better approximate the linear controller add more centers in the linear region as shown in Fig. 3.2. We also see from Fig. 3.3 that the pendulum remains stabilized in the upright position.

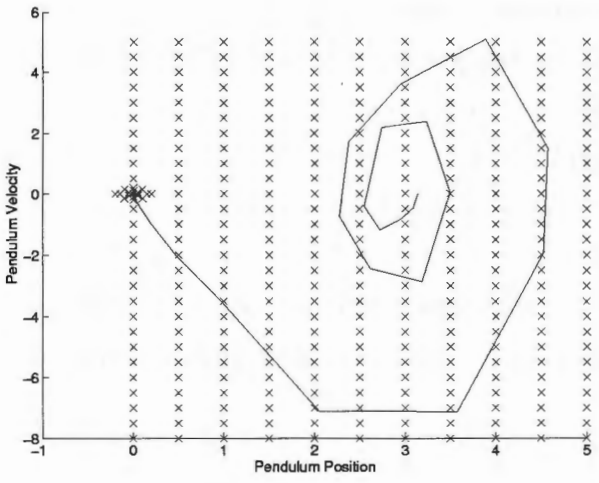

Figure 3.2: Plot of the trajectory of pendulum from hanging down to standing erect with additional linear region centers

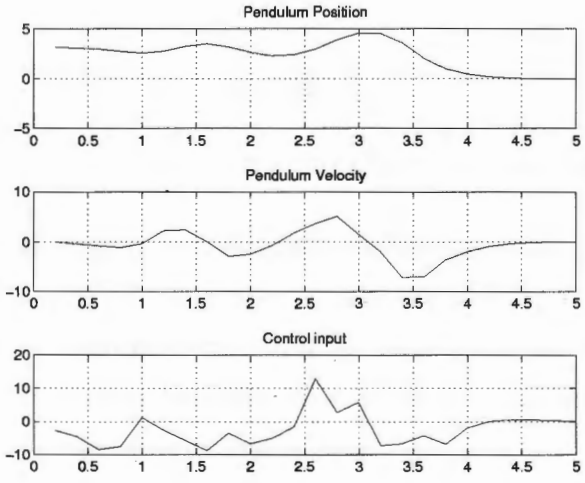

Figure 3.3: Plot showing the stabilization of the system at the target equilibrium point.

3.3.1 Fixed Centers Selected on a Grid

Given equation (2.16) in Chapter two the RBF network was designed with α chosen to be 9×10^{-7}. The training data for the network are the six trajectories shown in Fig. 3.1. The centers, $\mathbf{c}$, are placed in fixed intervals on a rectangular grid with the covariance matrix, Σ equal to $0.25I_2$. There are 297 basis functions that span the permissible state space. Given this setup each basis function operates pretty much independently and locally. Figures 3.4 and 3.5 show the distribution of the centers and the plot of the trajectory from the hanging down state, $x_1 = \pi$, to the upright position, $x_1 = 0$ and its corresponding weight surface, respectively.

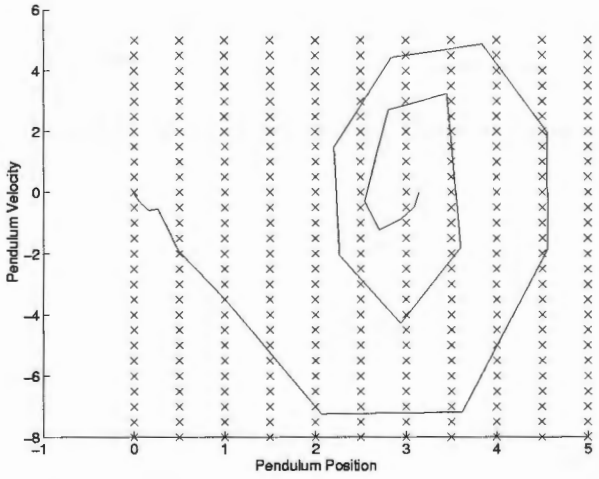

Figure 3.4: Plot of the trajectory of pendulum from hanging down to standing erect

As discussed earlier, the dynamic programming algorithm exploits the principle of optimality which in concise terms goes something like this; if path *abc* is optimal from a to c then path *bc* is optimal from b to c. This concept is demonstrated in Fig. 3.6. Note that we inserted more basis functions about the linear region to ensure that the linear controller is better approximated.

Figure 3.7 shows the network's ability to achieve generalization. Even though the initial state, $\mathbf{x}_o = [3.5, -1]^T$ and its corresponding trajectory were not used for training, the network is able to produce a mapping that achieved the control objective.

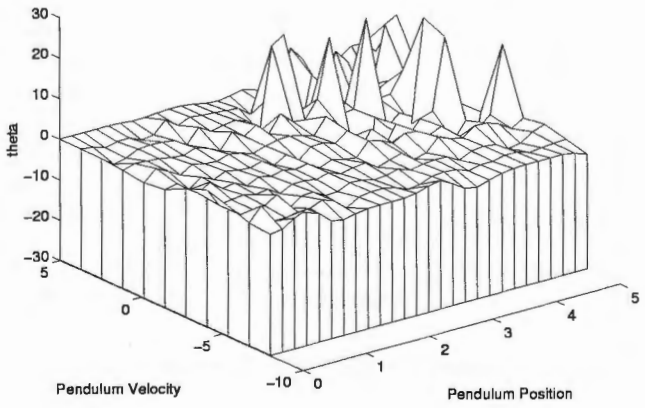

Figure 3.5: Weight surface of the state space

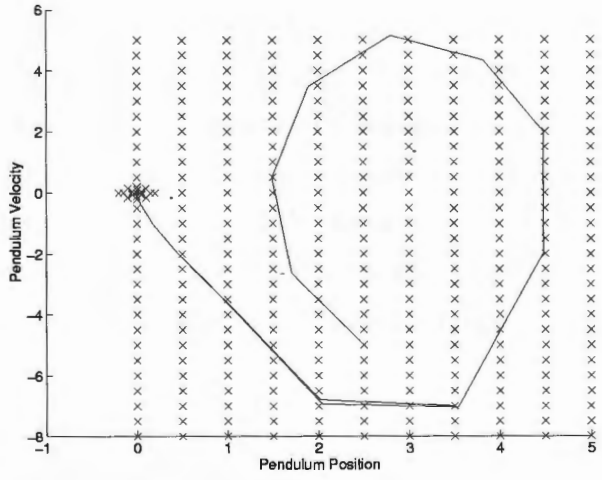

Figure 3.6: Plot showing the principle of optimality. One trajectory starts at $\mathbf{x}_o = [2.5, -5]^T$ and the other at $\mathbf{x}_o = [3.5, -7]^T$

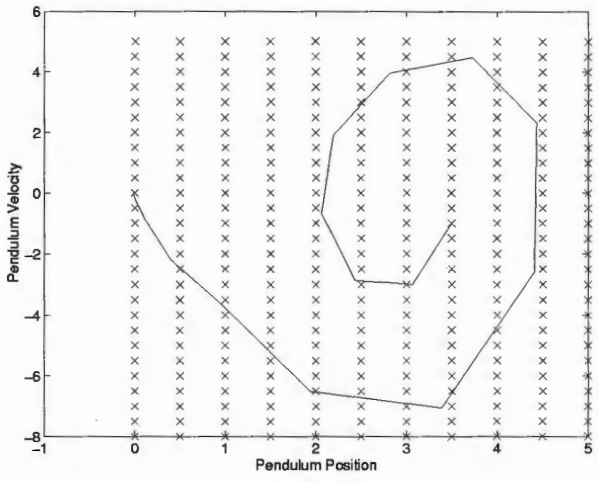

Figure 3.7: Plot showing the network's ability to generalize with an untrained intial vector of $\mathbf{x}_o = [3.5, -1]^T$.

3.3.2 Self-Organized Selection of Centers

In the previous section we placed the centers of the radial basis functions on a lattice of fixed interval grid points. This task was trivial given the dimensionality of the state space. However, in dimensions of three or higher, ones ability to visualize the state space of a system becomes impaired and placing centers on hyper-spheres is by no means trivial. We now use the SOFM algorithm discussed in the previous chapter and see how it compares with the results of fixed grid centers.

The SOFM network was trained with the same six trajectories used to train the fixed grid RBF as shown in Fig. 3.1. We used a single layered network with 160 neurons to produce the centers of the RBF network. The result of that training for a thousand epochs is displayed in Fig. 3.8. The covariance matrix used is;

$$\mathbf{\Sigma} = dI_2 \tag{3.7}$$

where $d = 2.7$ is the average distance between centers.

Figures 3.9, 3.10 and 3.11 show the ability of the network to approximate an optimal trajectory and to implement the linear controller.

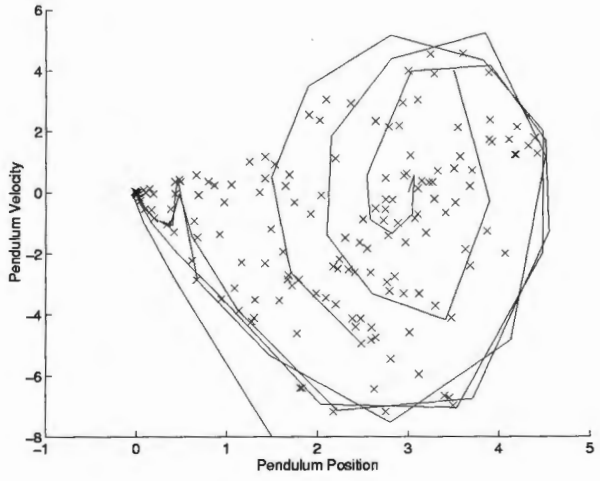

Figure 3.8: Placement of centers to match input trajectories

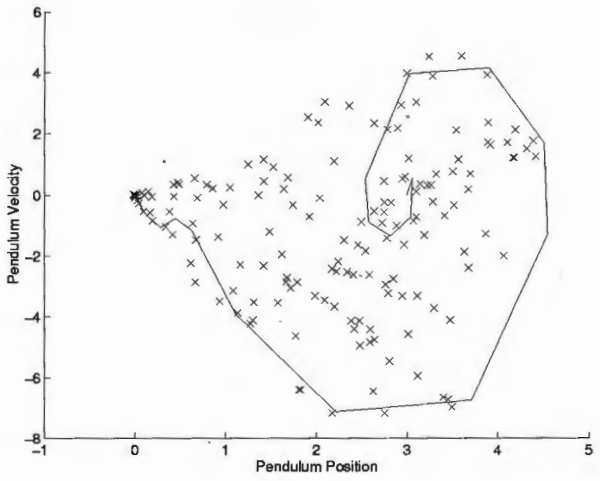

Figure 3.9: Plot of the trajectory from hanging to the upright position

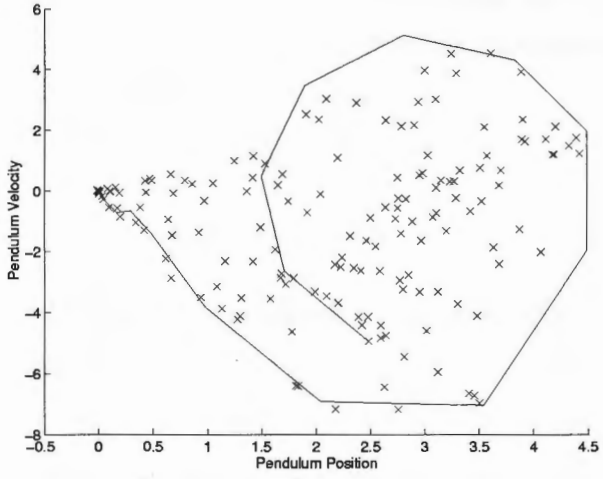

Figure 3.10: Plot for a non-equilibrium initial state

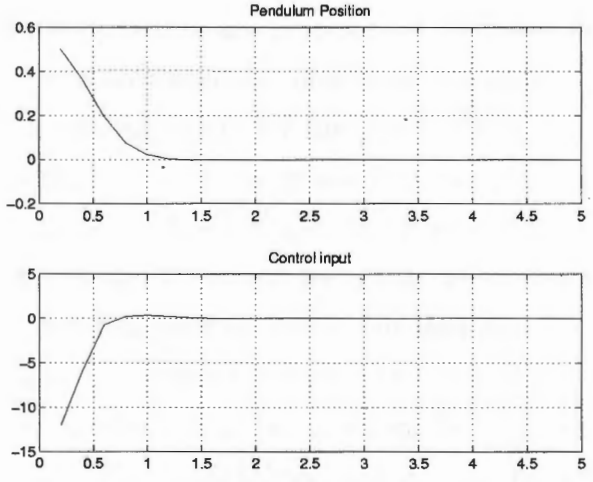

Figure 3.11: Approximation of the linear controller

3.4 Control by Segmentation

In this section we briefly explore the possibility of using the results of the previous sections to implement a control law for the fourth order pendulum-cart system. Recall the equations for the cart dynamics from (1.5)

$$\dot{x_3} = x_4$$
$$\dot{x_4} = -Cx_4 + Du$$

Also recall from (3.1) that $\hat{u}$ is equivalent to the acceleration of the cart. Substituting it for $\dot{x_4}$ in yields

$$\dot{x_4} = \hat{u} = -Cx_4 + Du \tag{3.8}$$

where u is the input to the cart system. Solving for u produces

$$u = \frac{n\hat{u} + Cx_4}{D} \tag{3.9}$$

where x_4 is obtained by numerical integration of $\hat{u}$:

$$x_4(t) = \int_0^t \hat{u}(\tau)d\tau + C_1. \tag{3.10}$$

Observe that the cart dynamics are represented by linear equations whereas the pendulum dynamics are represented by nonlinear equations. Given that we have already determined the control inputs for the pendulum system using dynamic programming, which was fairly easy to do given that the plant was second order, we can now compute the input to the cart system. Figure 3.12 show a block diagram of this control by segmentation. It would seem that we have solved the fourth order problem and can pack our bags and go home, but that may be foolhardy. There are two serious drawbacks to this control scheme. The first is that there does not seem to be a way to place a threshold limit on u, which becomes a considerable point in hardware implementation. The second drawback is when the acceleration of the cart goes to zero, the cart could still be moving at a constant velocity. Another drawback to this control law is the errors introduced in performing the integration in (3.10); more precisely in determining the constant of integration.

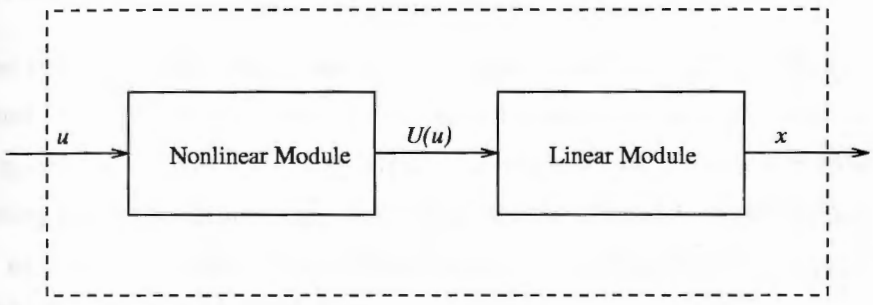

Figure 3.12: Block diagram showing the segmentation modules

Chapter 4

The Fourth Order Model

4.1 Introduction

In Chapter two *dynamic programming* was used to generate the training data for the network and lend itself as a powerful nonlinear control design tool, producing optimal control trajectories. However, for higher dimensions of three or greater *dynamic programming* becomes almost impractical and suffers from what Bellman [20] called *the curse of dimensionality.* What this means is that the number of quantized state vectors, which is the product of each quantized state variable, becomes exceedingly large requiring a lot of memory for storage and tremendously increases computational time. It is given this drawback that we present in this chapter a new control law that will produce the training data for the fourth order system (see Chapter one).

4.2 Energy Controller

In this section we develop a control law to regulate the swinging energy of the pendulum without regard to the cart dynamics. The resulting control system is such that the swinging energy will converge to the desired energy trajectory from almost all initial conditions. This controller design is based on a paper by Chung and Hauser, [9], in which they proposed a control law that would regulate the swing energy of the pendulum-on-a-cart system by maintaining a desired periodic orbit.

Given the equations for the pendulum dynamics in (1.5)

$$\begin{aligned} \dot{x_1}(t) &= x_2(t) \\ \dot{x_2}(t) &= -A\sin(x_1(t)) - \frac{A}{ng}\cos(x_1(t))(-Cx_4(t) + Du(t)) \end{aligned} \tag{4.1}$$

We would like to design a feedback control u so that the swing energy of the pendulum, defined by the kinetic and potential energy of the rod,

$$H(\theta,\omega) = \frac{1}{2}(l\omega)^2 + mgl(1-\cos(\theta)) \tag{4.2}$$

is regulated to a desired swing energy $\hat{H}$. Note that $\theta = x_1$ and $\omega = x_2$. m and l are mass and length of the rod, respectively and g is acceleration due to gravity. $\hat{H}$ is a time-varying function that depends on the energy of the initial states, $\theta(t_o)$ and $\omega(t_o)$, and on the energy at the final states, $\theta(t_f) = \pi$ and $\omega(t_f) = 0$. $\hat{H}$ is given by

$$\hat{H}(t) = \pm H(\theta(t_o),\omega(t_o)))e^{-\beta t} - mgl(1-\cos(\pi)) \tag{4.3}$$

where β is a design parameter. This function is used to drive the total energy, H, gradually from the initial energy determined by the initial state to a final energy when the pendulum is inverted in the linear region and from where linear control can then be employed. For example, if the initial energy is lower than at the final state, $\hat{H}$ would increase the total swinging energy until it reaches the final energy level. On the other hand, if the initial energy is higher than at the final state, $\hat{H}$ would decrease the total swing energy until it reaches the final energy level.

Next we define the error function

$$E(\theta,\omega) = H(\theta,\omega) - \hat{H}$$

and if we choose the feedback control law to be

$$u = \alpha\omega\cos\theta E \tag{4.4}$$

then in the limit as t becomes large, $E(t)$ goes to zero [9].

4.2.1 Simulation Results of the Energy Controller

The energy control law by itself could not balance and stabilize the pendulum in the upright position (linear region); so we insert the *pole placement* linear controller (see Appendix A) when the pendulum-cart system enters the linear region (see Chapter one) to stabilize and to compensate for minor perturbations in the system. Also, the energy controller has to be given a nonzero initial state or it will remain at rest with $u = 0$.

• The energy controller was designed with $m = 0.25kg$, $l = 0.16m$, $g = 9.81\frac{m}{s^2}$, and $\beta = 2$. In the linear region, the control law is

$$u = -\mathbf{Lx}$$

where

$$\mathbf{L} = \begin{bmatrix} 10.7688 & 1.6143 & -0.0125 & -0.0160 \end{bmatrix}$$

• Simulation results for the controller are shown in Fig. 4.1 to Fig.4.6. Figures 4.1 and 4.2 show the plots of the control input and state variables, respectively from the hanging down position, $x_1 = \frac{\pi}{10}$, to up position, $x_1 = \pi$. The controller is able to balance and stabilize the pendulum and cart even when the initial states that are far from the final states as illustrated in Fig. 4.3 to Fig. 4.6.

4.3 RBF Network Training Simulation Results

We trained the RBF network using two trajectories, one whose initial state vector $\mathbf{x}_o = [\frac{\pi}{10}, 0, 0, 0]^T$ is in the nonlinear region, and the other whose initial state vector $\mathbf{x}_o = [(\pi - 0.4), 0, 0, 0]^T$ is in the linear region. The latter is to ensure that the network has enough presentations in the linear region so that it would learn to mimic the linear controller.

We chose 100 radial basis functions whose centers are determined by the SOFM network, and a fixed inverse covariance matrix of these centers was selected to be

$$\mathbf{\Sigma}^{-1} = diag[(c_{1_{max}}\sigma_1^2, c_{2_{max}}\sigma_2^2, c_{3_{max}}\sigma_3^2, c_{4_{max}}\sigma_4^2]$$

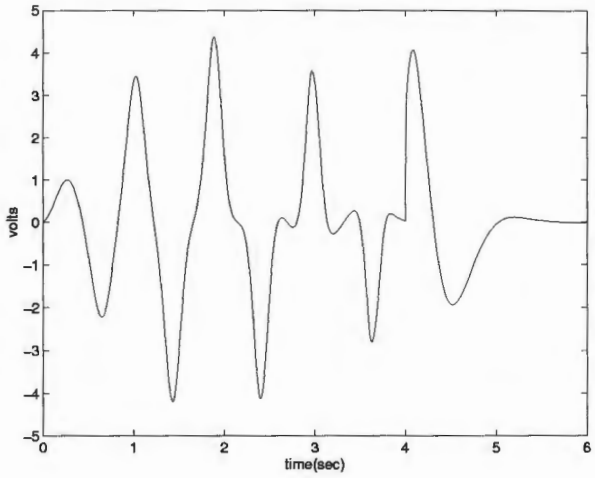

Figure 4.1: Plot of the control input that moves the pendulum from hanging down,$x_1 = \frac{\pi}{10}$, to the upright position, $x_1 = \pi$

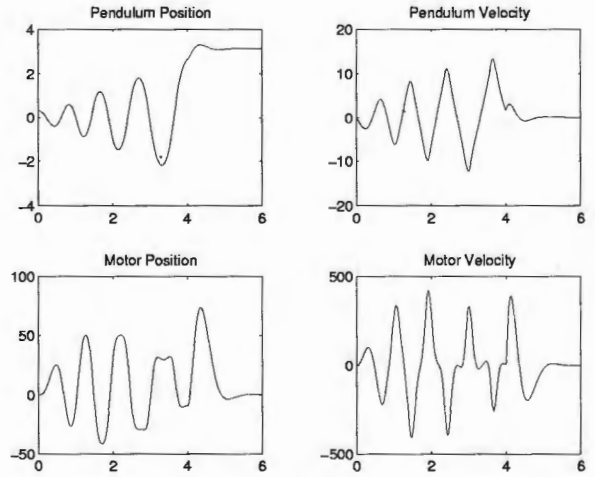

Figure 4.2: Plot of the state variables with initial state $\mathbf{x}_o = [\frac{\pi}{10}, 0, 0, 0]^T$

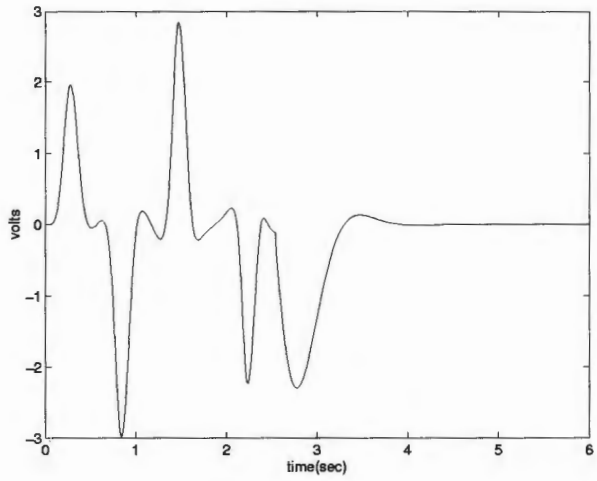

Figure 4.3: Plot of the control input when the initial state is $\mathbf{x}_o = [\frac{\pi}{2}, 0, 100, 0]^T$

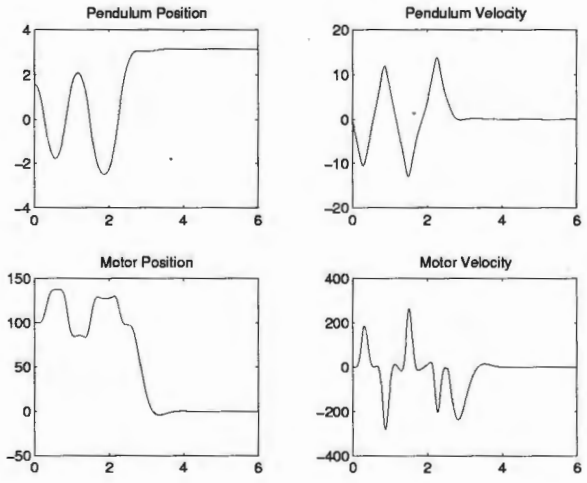

Figure 4.4: Plot of the state variables with initial state $\mathbf{x}_o = [\frac{\pi}{2}, 0, 100, 0]^T$

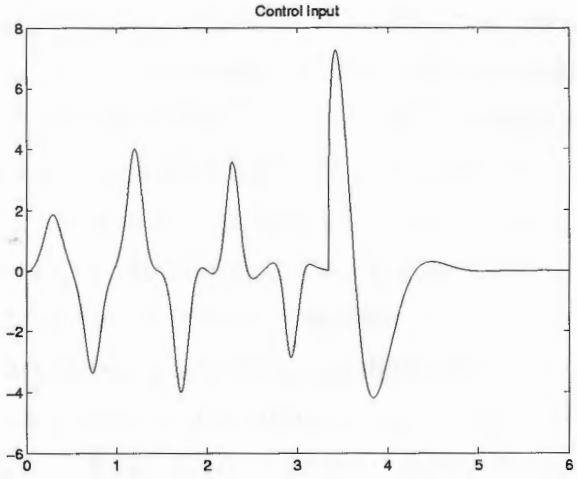

Figure 4.5: Plot of the control input when the initial state is $\mathbf{x}_o = [\frac{\pi}{4}, 1, 10, 20]^T$

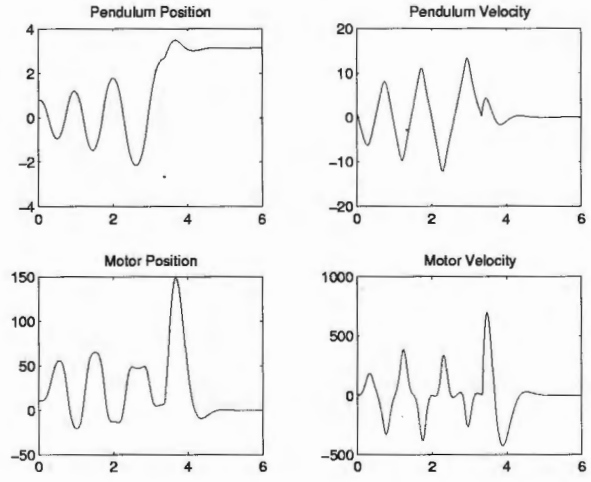

Figure 4.6: Plot of the state variables with initial state $\mathbf{x}_o = [\frac{\pi}{4}, 1, 10, 20]^T$

where $c_{i_{max}}$ is the maximum valued center for the ith state vector, and σ_i^2 is the variance associated with that ith state vector. The result of the training is shown in Fig. 4.7. The closed-loop simulation results of the network are displayed in Figures 4.8 and 4.9. Figures 4.10 and 4.11 show the networks ability to generalize given an untrained initial state of $\mathbf{x}_o = [\frac{\pi}{5}, 0, 0, 0]^T$. Though the network is able to achieve some generalization and obtained the designed objective, its overall ability to produce closed-loop results similar to that of the training set is relatively poor. This poor showing could be attributed to the choice of the fixed covariance matrix, $\mathbf{\Sigma}$, which probably does not accurately represent the variance distribution of the centers. The poor generalization is further manifested when we provided the network with four training trajectories; the four initial points for these trajectories were $\mathbf{x}_o = \{(\frac{\pi}{10}, 0, 0, 0); (\frac{\pi}{4}, 1, 10, 20); (\frac{\pi}{2}, 0, 100, 0); (\pi - 0.4, 0, 0, 0)\}$. The closed-loop simulation results are shown in Figures 4.12, 4.13 and 4.14. Note from Fig. 4.12 that the approximation to the training set is good, but when the network is placed in closed loop it is not able to achieve the design objective as depicted in Fig. 4.13 and Fig. 4.14.

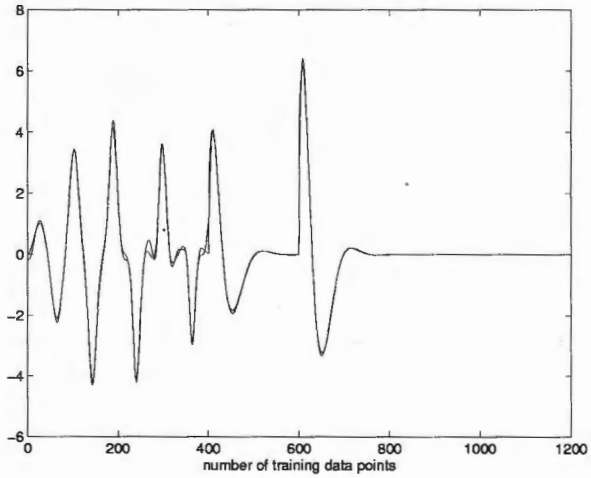

Figure 4.7: The RBF network approximation to the input trajectories.

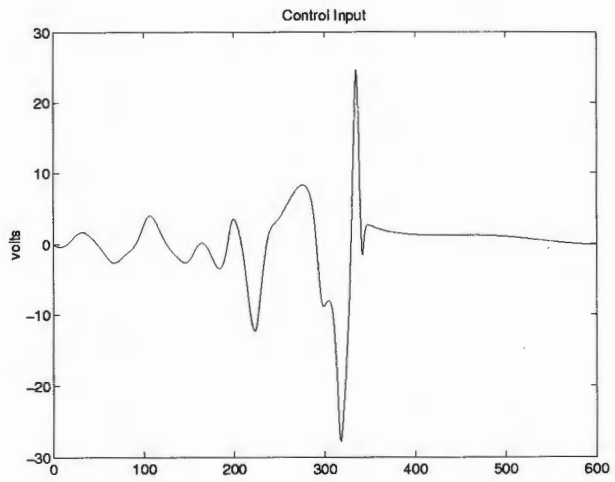

Figure 4.8: Closed-loop simulation of the RBF network showing the control input

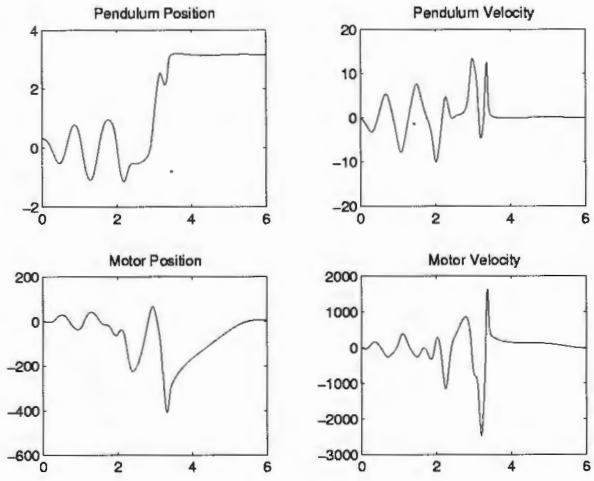

Figure 4.9: Closed-loop simulation of the RBF network showing the state variables

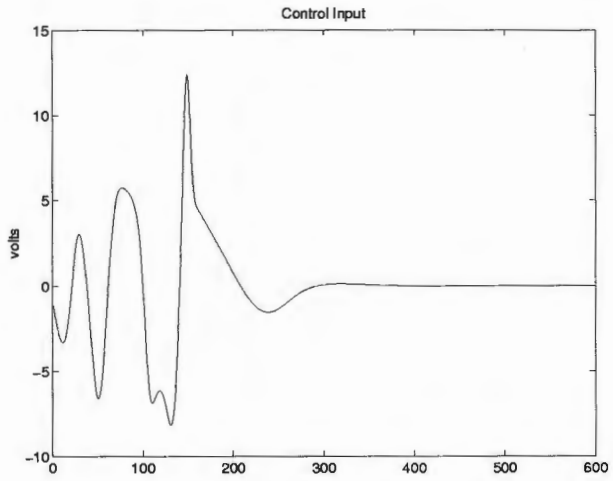

Figure 4.10: Illustration of the RBF network ability to generalize, showing the state variables with initial state $\mathbf{x}_o = [\frac{\pi}{5}, 0, 0, 0]^T$

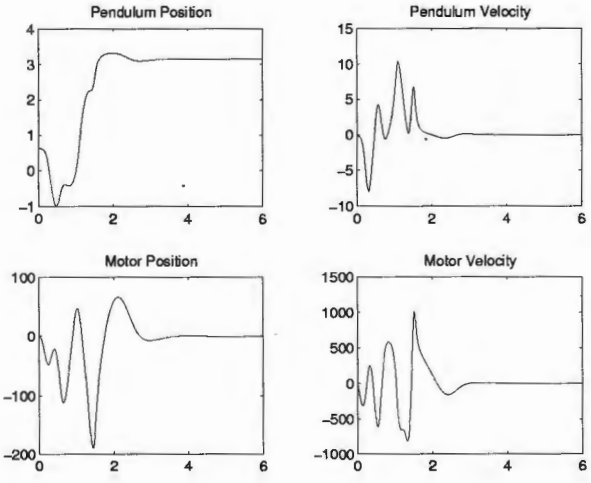

Figure 4.11: Illustration of the RBF network ability to generalize , showing the control input

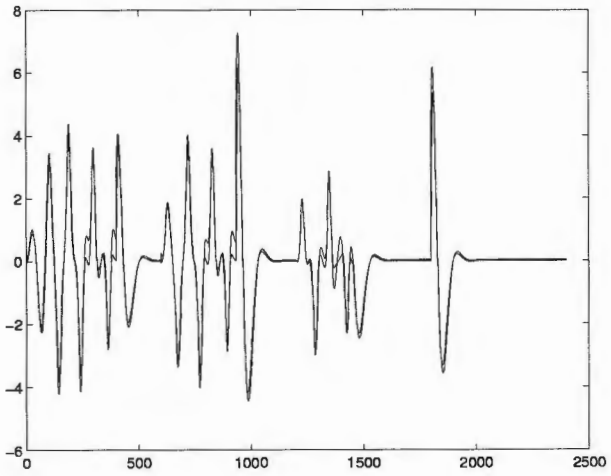

Figure 4.12: The RBF network approximation to the four training trajectories.

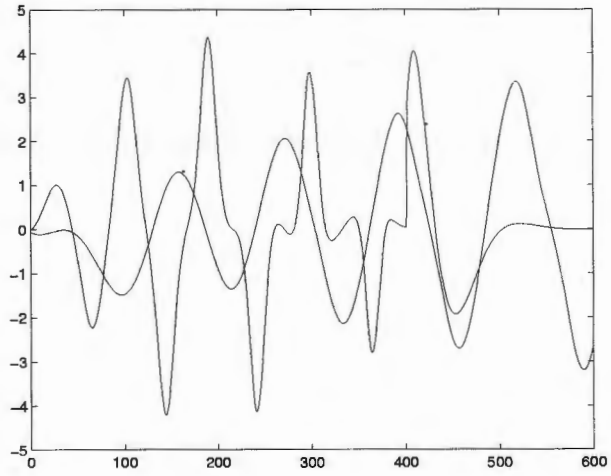

Figure 4.13: Closed-loop simulation of the RBF network showing the control input

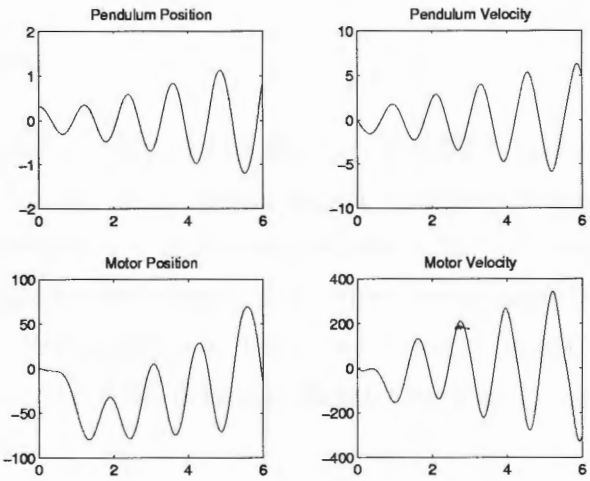

Figure 4.14: Closed-loop simulation of the RBF network showing the state variables

Chapter 5

Conclusions

5.1 Summary

In this research we demonstrated the ability of *radial basis function networks* to implement control of nonlinear systems, given that there exists training data that achieves the design objectives. Furthermore, the RBF network controller was successful in meshing together seamlessly controllers from nonlinear control theory and linear control theory. We also showed how a self-organizing feature map can be used to place the centers for the RBF network, thereby making the network more efficient and possibly adaptive.

In both Chapters three and four, the control laws presented are able to balance and stabilize the pendulum-cart system from *all* permissible initial states. The *dynamic programming algorithm* discussed in chapter three is a powerful control scheme that guarantees optimal results for any nonlinear control problem. Unlike gradient descent algorithms use in optimal control theory dynamic programming cannot get stuck in a region of local minimal, but instead produces global results. Furthermore, a nice feature of the algorithm is that we are able to put constraints on the control input as well as the state variables. The curse of dimensionality, which is the only major but considerable drawback of dynamic programming, limits the algorithm to lower dimensional problems because in higher dimensions computation become expensive in terms of a huge requirement for computer memory and processor time. On the

other hand, the energy control law presented in chapter four is fast to converge to a result, but the result is not optimal and is specific to the pendulum problem.

In retrospect of this research to use RBF networks to implement nonlinear control systems, the author believes that the single most important thing to improve with these networks, is determining a strategy for selecting the spreads of the centers or inverse covariance matrix. An accurate determination of this parameter is crucial for the network's ability to generalize and achieve the design objectives. For dimensions in which visualization of the distribution of the network's centers is impossible, such a strategy is very much needed. However, if generalization and minimal network configuration are not issues of the application of interest, then exact RBF networks, which are not sensitive to this covariance parameter, can be used.

5.2 Further Work

As noted in the concluding remarks of the previous section, the single most important thing that needs improvement with the generalized RBF networks is determining the spread of the centers; hence, it is basis for further work. The author proposes the following strategies for obtaining the spread of the centers;

1. Define an initial inverse covariance matrix for all centers

$$\mathbf{\Sigma}^{-1} = d(\mathbf{C}^T\mathbf{C})$$

where d is the average distance between centers, $\mathbf{C}$ is a norm weighting matrix that is diagonal and is used to normalize the input data in a unit hypersphere.

Next, we minimize a cost function over the inverse covariance matrix

$$\min_{\Sigma^{-1}} \mathcal{E} = \|\mathbf{Y} - \hat{\mathbf{Y}}\|$$

where $\mathbf{Y}$ is the desired output vector and $\hat{\mathbf{Y}}$ is the approximated output vector.

Using this scheme we can get an optimal inverse covariance matrix that gives a general representation of the spreads of all the centers. It should be noted that the position of the centers are fixed and determined by the SOFM network.

2. Another approach for the selection of the spreads of the centers is to find the inverse covariance matrix associated with each center. This parameter and all other free parameters of the network are determined using a gradient-descent learning procedure instead of a linear regression technique. The first step in the development of such a learning procedure is to define the instantaneous value of the cost function

$$\mathcal{E} = \frac{1}{2}\sum_{j=1}^{N} e_j^2$$

where N is the number of training examples used to undertake the learning process, and e_j is the error signal, defined by

$$e_j = y_j - \sum_{i=1}^{N_a} w_i \mathbf{\Phi}(\|\mathbf{x}_j - \mathbf{c}_i\|_{\mathbf{\Sigma}_i^{-1}})$$

The requirement is to find the free parameters w_i, $\mathbf{c}_i$, and $\mathbf{\Sigma}_i^{-1}$ so as to minimize $\mathcal{E}$. The results of this minimization are summarized as follows;

(a) *Linear weights*

$$\frac{\partial \mathcal{E}(n)}{\partial w_i(n)} = \sum_{j=1}^{N} e_j \mathbf{\Phi}(\|\mathbf{x}_j - \mathbf{c}_i\|_{\mathbf{\Sigma}_i^{-1}})$$

$$w_i(n+1) = w_i(n) - \eta_1 \frac{\partial \mathcal{E}(n)}{\partial w_i(n)} \quad i = 1, 2, ..., N_a$$

(b) *Positions of centers*

$$\frac{\partial \mathcal{E}(n)}{\partial \mathbf{c}_i(n)} = 2w_i(n)\sum_{j=1}^{N} e_j \mathbf{\Phi}(\|\mathbf{x}_j - \mathbf{c}_i\|_{\mathbf{\Sigma}_i^{-1}})\mathbf{\Sigma}_i^{-1}[\mathbf{x}_j - \mathbf{c}_i(n)]$$

$$\mathbf{c}_i(n+1) = \mathbf{c}_i(n) - \eta_2 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{c}_i(n)} \quad i = 1, 2, ..., N_a$$

(c) *Spreads of centers*

$$\frac{\partial \mathcal{E}(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)} = -w_i(n)\sum_{j=1}^{N} e_j \mathbf{\Phi}(\|\mathbf{x}_j - \mathbf{c}_i\|_{\mathbf{\Sigma}_i^{-1}})\mathbf{\Sigma}_i^{-1}[\mathbf{x}_j - \mathbf{c}_i(n)][\mathbf{x}_j - \mathbf{c}_i(n)]^T$$

$$\mathbf{\Sigma}_i^{-1}(n+1) = \mathbf{\Sigma}_i^{-1}(n) - \eta_3 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)} \quad i = 1, 2, ..., N_a$$

Finally, the author would like to see some work done in further developing the *control by segmentation* methodology proposed in Chapter three.

Appendix A

Summary of Pole Placement

In this appendix we provide a summary of pole placement by digital state feedback. [7] and [8] give a more detailed development of this strategy from linear control theory.

Given a continuous-time system represented by;

$$\dot{\mathbf{x}} = \mathbf{Ax(t)} + \mathbf{bu(t)} \tag{A.1}$$

1. Choose appropriate s-plane pole locations and sampling interval, T.

2. Obtain the *zero order hold*, ZOH, of the plant

$$\mathbf{x[k+1]} = \mathbf{\Phi x[k]} + \mathbf{\Gamma u[k]} \tag{A.2}$$

where $\Phi = \exp \mathbf{AT}$ and $\Gamma = \int_0^T e^{\mathbf{A}\tau}\mathbf{b}d\tau$

Then calculate the denominator ploynomial of the ZOH model.

$$det(z\mathbf{I} - \mathbf{\Phi}) = \mathbf{a(z)} = \mathbf{z^n} + \mathbf{a_1 z^{n-1}} + ... + \mathbf{a_n} \tag{A.3}$$

If the poles, s_i, of the continuous-time plant are, then $a(z)$ can be computed as follows

$$a(z) = \prod_{i=1}^{n}(z - e^{s_i T}) \tag{A.4}$$

3. Map the s-plane pole locations chosen in Step 1 to the z-plane using the ZOH pole mapping formula $z_i = e^{s_i T}$. Multiply out the z-plane pole locations to obtain $p(z)$

$$p(z) = \prod_{i=1}^{n}(z - z_i) = z^n + p_1 z^{n-1} + ... + p_n \tag{A.5}$$

4. Form the controllability matrix $\mathbf{W_c}$ of the plant (ZOH), and form the controllability matrix $\bar{\mathbf{W}}_\mathbf{c}$ of the controllable canonical form.

$$\mathbf{W_c} = \left[\begin{array}{cccc} \Gamma & \Phi\Gamma & \ldots & \Phi^{n-1}\Gamma \end{array} \right] \tag{A.6}$$

$$\bar{\mathbf{W}}_\mathbf{c} = \left[\begin{array}{cccc} \bar{\Gamma} & \bar{\Phi}\bar{\Gamma} & \ldots & \bar{\Phi}^{n-1}\bar{\Gamma} \end{array} \right] \tag{A.7}$$

where $\bar{\Phi} = T_c \Phi T_c^{-1}$ and $\bar{\Gamma} = T_c \Gamma$. T_c is a transformation matrix that takes the controllable system to controllable canonical form.

5. Calculate the feedback vector

$$\bar{\mathbf{L}} = \left[\begin{array}{cccc} (p_1 - a_1) & (p_2 - a_2) & \ldots & (p_n - a_n) \end{array} \right] \bar{\mathbf{W}}_\mathbf{c} \mathbf{W}_\mathbf{c}^{-1} \tag{A.8}$$

Appendix B

Computer Code

This appendix provides some of the Matlab codes used in the design and simulation of the neural controller.

B.1 The dynamic programming algorithm

```
function [COST,UOPT]=dyprog(N,Ts)
% **********************Initialization******************
global U
[x,u,C,S]=getgrid;
COST=zeros(N,S);
UOPT=zeros(N,S);
W=1e3*diag([10 1 10 1]);  %defines the penalty matrix for the final state.
%*********************************************************************
for i=1:S
    COST(N,i)=x(:,i)'*W*x(:,i);
end
for K=1:N-1
   for i=1:S
     COSMIN=1e6;
```

```
    for j=1:C
        U=u(j);
        [T,Y]=ode23('odepen',(i-1)*Ts,i*Ts,x(:,i));
        [l,w]=size(Y);
        X=round(Y(l,:)'*1000)./1000;
        JNN=inter(X,COST(N-K+1,:));
        Copt=cost(x(:,i)) + JNN;
        if Copt <= COSMIN
           COSMIN = Copt;
           UMIN=U;
        end
     end
    COST(N-K,i)=COSMIN;
    UOPT(N-K,i)=UMIN;
  end
end


function [x,u,C,S]=getgrid()
%**********************Initializations************************
Umax=8; Umin=-8;
delx1=0.5;
delx2=2;
delx3=50;
delx4=100;
x1max=6;x1min=0;
x2max=10;x2min=-10;
x3max=200;x3min=-200;
x4max=500;x4min=-500;
s1=round((x1max-x1min)/delx1) +1;
s2=round((x2max-x2min)/delx2) +1;
```

```
s3=round((x3max-x3min)/delx3) +1;
s4=round((x4max-x4min)/delx4) +1;
u=[Umin:-2 -1.5:0.5:1.5 2:Umax];
C=length(u);
S=s1*s2*s3*s4;
x=zeros(4,S);
%*************************************************************
x1=x1min:delx1:x1max;
x2=x2min:delx2:x2max;
x3=x3min:delx3:x3max;
x4=x4min:delx4:x4max;
count=0;
for i=1:length(x1)
      for j=1:length(x2)
         for k=1:length(x3)
            for l=1:length(x4)
                count=count+1;
                x(:,count)=[x1(i);x2(j);x3(k);x4(l)];
            end
         end
      end
end

function Japprox=inter4(X,J)
epsilon=0.01;
x1=[floor(2*X(1))/2 ceil(2*X(1))/2];
x2=[floor(0.5*X(2))/0.5 ceil(0.5*X(2))/0.5];
x3=[floor(0.02*X(3))/0.02 ceil(0.02*X(3))/0.02];
x4=[floor(0.01*X(4))/0.01 ceil(0.01*X(4))/0.01];
%*************************************************************
count=0;xi=zeros(4,16);
```

```
for i=1:length(x1)
    for j=1:length(x2)
        for k=1:length(x3)
            for l=1:length(x4)
                count=count+1;
                xi(:,count)=[x1(i);x2(j);x3(k);x4(l)];
                dist(count)=norm(X-xi(:,count));
            end
        end
    end
end
%************************************************************
if X(1)<0 | X(1)>6 | X(2) <-15 | X(2)>15 | X(3)<-200 | X(3)>200 |
   +  X(4) <-500 | X(4)>500 % these constraints ensure that the state
                             % vectors do not exceed their boundaries.
   Japprox=0;
elseif min(dist) < epsilon
   tmp=xi(:,find(dist==min(dist)));
   Japprox=J(getindex(tmp(:,1)));
else
   s=1/sum(1./dist(1:length(xi)));
   Japprox=0;
   for i=1:length(xi)
       tmp=xi(:,i);
       Japprox=(s/norm(X-tmp))*J(getindex(tmp)) + Japprox;
   end
end
```

B.2 The energy controller algorithm

```
global U;
```

```
%**********************Declaration of Variables************************
L=getl4;  % determines the state feedback gains vector
M=.25;    % mass in kilograms of pendulum rod
l=0.16;   % distance in meters from pivot to center of mass of pendulum
g=9.81;   % gravitational acceralation (m/s^2)
Ts=0.01;  % sampling time
B=2;      % a design parameter that gives the time constant of the target
          % swing trajectory
ftime=6;  % simulation time
alpha=1;  % design parameter
% ***********************************************************************
m=round(ftime/Ts);
t=[0:m]*Ts;
x(:,1)= input(' Enter the initial state vector ')
C=(.5*(l*x(2,1))^2 + m*g*l*(1-cos(x(1,1)))) - 0.7652;
if C>=0.7652
   C=-C;
end
H_hat=C*exp(-B*t)+0.7652; %target swing trajectory with max energy equals
x=zeros(4,m);             %to 0.7652 at pi
for j=1:m
   if abs(x(1,j)) > (pi-0.5)
      U=-L*[x(1,j)-pi;x(2:4,j)];  %insert linear controller when rod
   else                           %enters linear region
      E(j)=(0.5*M*(l*x(2,j))^2 + M*g*l*(1-cos(x(1,j)))) - H_hat(j);
      U=alpha*x(2,j)*cos(x(1,j))*E(j);
   end
   [T,Y]=ode23('odepend',(j-1)*Ts,Ts*j,x(:,j));
   [l,w]=size(Y);
   u1(j)=U;
   x(:,j+1)=Y(l,:)';
```

```
end
%*********************************************************************
subplot(2,2,1), plot(t,x(1,:));
subplot(2,2,2), plot(t,x(2,:));
subplot(2,2,3), plot(t,x(3,:));
subplot(2,2,4), plot(t,x(4,:));
```

B.3 The training algorithm

```
% ****************Training*********************************
<load training data>
<define input and output matrices, x and y respectfully.>
Q=sofm(x);   % function call to the self-organizing routine to place
             % the centers. This function simply makes a call to Matlab´s
             % function for Kohonen´s self-orgaining feature map.
Q=Q´;
n=length(y);
m=length(Q);
alpha=0.000000000009;  %design parameter to compensate for ill conditions
%*******Method I to determine the inverse varaince parameter***************
% d=diag(Q´*Q);
% [d1,i1]=max(d);
% [d2,i2]=min(d);
% C=2*m/(norm(Q(:,i1)-Q(:,i2)))^2;
%*******Method II to determine the inverse varaince parameter**************
C=inv(diag([max(abs(Q(1,:)))^2*cov(Q(1,:)) max(abs(Q(2,:)))^2*cov(Q(2,:))
  + max(abs(Q(3,:)))^2*cov(Q(3,:)) max(abs(Q(4,:)))^2*cov(Q(4,:))]));
%*********************************************************************
I=eye(m);
for i=1:m
```

```
    for j=1:n
       PHI(i,j)=exp(-0.5*((x(:,j) - Q(:,i))´ *C* (x(:,j) - Q(:,i))));
    end
end
theta=y*PHI´*inv(alpha*I + PHI*PHI´);
Y_o=theta*PHI;
delE=norm(y-Y_o)        %measure of the ``goodness´´ of the approximation.
save <filename> theta Q C;
```

B.4 Closed-loop simulation

```
% This program assumes that the centers,Q, weights vector, theta,
% and invariance, C, have already been determined.
global U;
< load Q,theta & C>
m=length(Q);
% ******************************Simulation*********************************
Ts=0.01;
ftime=6;
n=round(ftime/Ts);
t=[0:n-1]*Ts;
x=zeros(4,n);
u=zeros(1,n);
x(:,1)=input(´ Enter the initial state vector ´)
for i=1:m
   phi(i,:) = exp(-0.5*((x(:,1) - Q(:,i))´ * C * (x(:,1) - Q(:,i))));
end
u(1) = theta*phi;
for j=1:n-1
   U=u(j);
```

```
   [T,Y]=ode23('odepend',(j-1)*Ts,Ts*j,x(:,j));
   [l,w]=size(Y);
   x(:,j+1)=Y(l,:)';
   for k=1:m
      phi(k,:) = exp(-0.5*((x(:,j+1) - Q(:,k))' * C * (x(:,j+1) - Q(:,k))));
   end
   u(:,j+1)=theta*phi;
end
subplot(2,2,1), plot(t,x(1,:));
subplot(2,2,2), plot(t,x(2,:));
subplot(2,2,3), plot(t,x(3,:));
subplot(2,2,4), plot(t,x(4,:));
```

References

[1] A. U. Levin and K. S. Narendra, "Control of nonlinear dynamical systems using neural networks: Controllability and stabilization," *IEEE Transactions on Neural Networks*, vol. 4, pp. 192–206, March 1993.

[2] K. S. Narendra, J. Balakrishnan, and M. K. Ciliz, "Adaptation and learning using multiple models, switching, and tuning," *IEEE Control Systems Magazine*, vol. 15, pp. 37–51, June 1995.

[3] D. Gorinevsky, "On the persistency of excitation in radial basis function network identification," *IEEE Transactions on Neural Network*, vol. 6, pp. 1237–1244, September 1995.

[4] J. A. K. Suykens, B. L. D. Moor, and J. Vandewalle, "Static and dynamic stabilizing neural controllers, applicable to transition between equilibrium points," *Neural Network*, vol. 7, no. 5, pp. 819–831, 1994.

[5] S. Chen, S. A. Billings, and P. M. Grant, "Recursive hybrid algorithm for nonlinear system identification using radial basis function networks," *International Journal of Control*, vol. 55, no. 5, pp. 1051–1070, 1992.

[6] E. Tzirkel-Hancock and F. Fallside, "Stable Control of Nonlinear Systems Using Neural Networks," tech. rep., Cambridge University, Cambridge, England, July 1991.

[7] R. J. Vaccaro, *Digital Control: A State Space Approach.* McGraw-Hill Book Company, 1995.

[8] F. L. Lewis, *Applied Optimal Control and Estimation. Digital Design and Implementation.* Prentice Hall, 1992.

[9] C. C. Chung and J. Hauser, "Nonlinear control of a swinging pendulum," *IEEE Transactions on Automatic Control*, vol. AC-35, pp. 851–862, 1993.

[10] M. Kutter and F. Andersson, "Balancing the Inverted Pendulum Using a Feed Forward Neural Network," tech. rep., University of Rhode Island, Department of Electrical Engineering, Kingston, Rhode Island, May 1995.

[11] S. Chen, C. F. Cowen, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1237–1244, 1995.

[12] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.

[13] J. E. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1989.

[14] T. Poggio and F.Girosi, "Networks for approximating and learning," *Proceedings of the IEEE*, vol. 78, pp. 1481–1497, 1990.

[15] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 326–334, 1965.

[16] S. Haykin, *Neural Networks, A Comprehensive Foundation.* Macmillan College Publishing Company, Inc, 1994.

[17] S. M. Botros and C. G. Atkeson, "Generalization properties of radial basis basis functions," in *Advances in Neural Information Processing Systems* (R. P. Lippmann, J. E. Moody, and D. S. Touretzky, eds.), pp. 707–713, IEEE, 3 ed., 1993.

[18] W. A. Light, "Some aspects of radial basis function approximation," in *Approximation Theory, Spline Functions and Applications* (S. P. Singh, ed.), vol. 256 of *NATO ASI Series*, pp. 163–190, Kluwer Academic Publishers, 1992.

[19] T. Kohonen, "Self-organized formation of topological correct feature maps," *Biological Cybernetics*, vol. 43, pp. 58–69, 1982.

[20] R. Bellman, ed., *Modern Analytic and Computation Methods in Science and Mathematics*, pp. 2–192. Elsevier Publishing Company, 1970.

[21] D. E. Kirk, *Optimal Control Theory.* Electrical Engineering Series, Prentice-Hall, 1970.

Bibliography

Bellman, R., ed., *Modern Analytic and Computation Methods in Science and Mathematics*, pp. 2–192. Elsevier Publishing Company, 1970.

Botros, S. M. and Atkeson, C. G., "Generalization properties of radial basis basis functions," in *Advances in Neural Information Processing Systems*, (Lippmann, R. P., Moody, J. E., and Touretzky, D. S., eds.), pp. 707–713: IEEE, 1993.

Broomhead, D. S. and Lowe, D., "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.

Chen, S., Billings, S. A., and Grant, P. M., "Recursive hybrid algorithm for nonlinear system identification using radial basis function networks," *International Journal of Control*, vol. 55, pp. 1051–1070, 1992.

Chen, S., Cowen, C. F., and Grant, P. M., "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1237–1244, 1995.

Chung, C. C. and Hauser, J., "Nonlinear control of a swinging pendulum," *IEEE Transactions on Automatic Control*, vol. AC-35, pp. 851–862, 1993.

Cover, T. M., "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 326–334, 1965.

Gorinevsky, D., "On the persistency of excitation in radial basis function network identification," *IEEE Transactions on Neural Network*, vol. 6, pp. 1237–1244, September 1995.

Haykin, S., *Neural Networks, A Comprehensive Foundation.* Macmillan College Publishing Company, Inc, 1994.

Kirk, D. E., *Optimal Control Theory.* Prentice-Hall, 1970.

Kohonen, T., "Self-organized formation of topological correct feature maps," *Biological Cybernetics*, vol. 43, pp. 58–69, 1982.

Kutter, M. and Andersson, F., "Balancing the Inverted Pendulum Using a Feed Forward Neural Network," Technical report, University of Rhode Island, Department of Electrical Engineering, Kingston, Rhode Island, May 1995.

Levin, A. U. and Narendra, K. S., "Control of nonlinear dynamical systems using neural networks: Controllability and stabilization," *IEEE Transactions on Neural Networks*, vol. 4, pp. 192–206, March 1993.

Lewis, F. L., *Applied Optimal Control and Estimation. Digital Design and Implementation.* Prentice Hall, 1992.

Light, W. A., "Some aspects of radial basis function approximation," in *Approximation Theory, Spline Functions and Applications*, (Singh, S. P., ed.), pp. 163–190: Kluwer Academic Publishers, 1992.

Moody, J. E. and Darken, C. J., "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1989.

Narendra, K. S., Balakrishnan, J., and Ciliz, M. K., "Adaptation and learning using multiple models, switching, and tuning," *IEEE Control Systems Magazine*, vol. 15, pp. 37–51, June 1995.

Poggio, T. and F.Girosi, "Networks for approximating and learning," *Proceedings of the IEEE*, vol. 78, pp. 1481–1497, 1990.

Suykens, J. A. K., Moor, B. L. D., and Vandewalle, J., "Static and dynamic stabilizing neural controllers, applicable to transition between equilibrium points," *Neural Network*, vol. 7, pp. 819–831, 1994.

Tzirkel-Hancock, E. and Fallside, F., "Stable Control of Nonlinear Systems Using Neural Networks," Technical report, Cambridge University, Cambridge, England, July 1991.

Vaccaro, R. J., *Digital Control: A State Space Approach.* McGraw-Hill Book Company, 1995.