Model-Based Techniques in Motor Learning and Task Optimization

by

Sherif Maher Botros

B.Sc., Cairo University (1985)
M.Sc., Case Western Reserve University (1988)

Submitted to the Department of Brain and Cognitive Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1994

Signature of Author .....
Department of Brain and Cognitive Sciences
January 28, 1994

Certified by .....
Christopher G. Atkeson
Associate Professor
Thesis Supervisor

Accepted by .....
Gerald Schneider
Chairman, Departmental Committee on Graduate Students

Model-Based Techniques in Motor Learning and Task Optimization

by

Sherif Maher Botros

Submitted to the Department of Brain and Cognitive Sciences
on January 28, 1994, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis we address the problem of learning to perform dynamic tasks optimally with respect to given objectives and subject to the constraints imposed by the environment. We focus in particular on indirect methods. These methods are based on first acquiring a model of the system to be controlled, and then using this model to find the optimal actions. Key problems associated with learning task optimization include the representation of the acquired knowledge, algorithms of adaptation and learning, exploration and identification of the environment and optimal exploitation of previous experiences. Using computer simulations, we propose and test different approaches to the solution of these problems. We use HyperBasis Functions (HyperBF) to model the dynamics of the system. We propose a heuristic method for the estimation of the HyperBF parameters. This method separates the estimation of the norm metric from the other parameters. We address the exploration-exploitation problem in the context of the optimization of multi-dimensional functions. We propose and test a method for the solution of this problem which is based on the confidence in the acquired model. We also explore the problem of learning open and closed loop trajectory optimization. We discuss and test different approaches to learning optimal closed loop controllers. Finally, we explore the problem of finding the optimal actions in noisy environments. We conclude the thesis with a discussion of possible links between the algorithms used here and those used by biological systems.

Thesis Supervisor: Christopher G. Atkeson
Title: Associate Professor

Acknowledgments

I would like to express my sincere gratitude to the many people who have contributed, in their own way, to making this thesis possible.

First, I would like to thank my advisor, Professor Chris Atkeson, for his valuable help and financial support during the completion of this thesis. I would like also to thank the other members of my committee: Professor Michael Jordan, Professor Tomaso Poggio and Professor Ron Williams for their valuable suggestions, criticisms and encouragements. I am also very grateful to Professor Emilio Bizzi and Jan Ellertsen for solving many of the problems I encountered during my study at MIT.

Special thanks are due to my family and my dear friends who provided me with the moral support and encouragement.

Finally, I am grateful for the Fairchild foundation for their financial support during part of this research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Learning a new motor task can be defined as using previously collected knowledge and experience of the task to guide the system's future actions and responses. There are many different ways by which a dynamical system, such as a human being, can acquire this knowledge and experience to improve his actions. For example, the human being may get verbal instructions from a teacher, can watch skilled (or even unskilled) individuals perform the task, may physically practice the task, or even mentally practice it. Previous experience on a closely related task may sometimes help, but also can sometimes interfere with the learning of the new task. Motor learning is not only knowledge acquisition and retrieval, but also includes the efficient use of this knowledge to infer and plan new ways of solving a motor task. Humans and animals are not only capable of learning the basic skills required to perform complex motor tasks but they are also able to optimize their performance by practicing.

1.1 Different Levels of Learning and Optimization

We can distinguish different levels of motor learning. All the levels of learning involve some transformation of data and previous experiences into conclusions and models

of the environment and the use of these conclusions to guide actions. At the lowest level, what is known in the neural network literature as supervised learning, the system is given the desired output response, by a teacher for example, and is required to find the actions which will produce the desired behavior. An example of this is the learning of trajectory following by a robot arm. This problem is essentially solved if we are able to build an association between the different sets of possible actions and the outcomes of those actions under the different states of the environment and the dynamical system, i.e. we would like to find a map

$$f : a \times s \longmapsto o \tag{1.1}$$

where $a \in A$, $s \in S$ and $o \in O$; A is the set of all possible actions, S is the set of all possible states of the environment and the system and O the set of all possible outcomes (typically $O = S$).

There are many theoretical and practical issues that are involved in building such a map. In order to make the problem of learning this map tractable in terms of the amount of training needed, the learning system has to find among the almost infinite attributes of the possible actions and states, the ones that are relevant to the particular task that it is required to learn. This process is called dimensionality reduction. Another important issue is the issue of generalization, that is how to infer which actions to take based on previous experience when the learning system is faced with new situations that it has not encountered before. To solve this problem it is generally assumed that actions that are close in some space produce similar outcomes given similar states of the environment. The problem here is how to automatically find this measure of closeness based on the available data. Finally the question of finding which representation of the previous experiences is most useful remains.

At a second level of learning, the learning system is given only the goal of the task and is required to learn the actions that best achieve this goal. This level of learning is more general than the first level and includes it as a special case (e.g. the goal of

first level learning can be defined as: reproduce the desired response as accurately as possible). Learning task optimization usually comprises the following components:

- A data collection mechanism for exploring the environment and determining its response to the different action alternatives. This mechanism is responsible for action generation and exploration.
- A mechanism for the evaluation of actions with respect to a goal. When a sequence of actions is involved in achieving a goal, this process of temporal credit assignment is usually difficult.
- A mechanism for the modification of actions. This mechanism is responsible for finding and choosing the actions that are more successful at achieving the desired goal.

The learning system may also include other elements in addition to the above. For instance, it may include creating and identifying a model of the environment. This model can then be used for predicting the response to novel actions and for action modification.

To illustrate the issues and problems involved with this type of learning, which we call here task optimization, consider the following example. The way a tennis player chooses to hit a ball may have the goal of maximizing his chance of winning the point under such constraints as the time he/she has to react, the quality of the racket he/she has, the uncertainty she/he has in observing and predicting the trajectory of the ball and in performing the desired movement; and given her/his knowledge and previous experience about the dynamics of the ball and racket and his/her model about the opponent's strengths and weaknesses. Intuitively, having more knowledge and more accurate models of the task should result in better performance with respect to the same goal and under the same constraints. The knowledge that the player uses may be just the previous successes and failures when he/she previously applied different actions in the same situation, or a more detailed dynamical model that he/she formed

with experience. Since more relevant knowledge about the task will probably result in better performance, the player may seek to acquire more knowledge as he/she plays. However the goal of acquiring more knowledge requires different exploratory actions from the player that may be in conflict with the actions that maximize immediate success. The choice of the goal, hence the action, depends on the situation, for example whether the player is winning or whether the game is a practice game or in the world championship. This choice of goal also may depend on the player previous experience, whether he/she "feels" that he/she "knows" enough and is confident in his/her knowledge, or that he/she needs to explore more actions. This also brings another important issue in task optimization, which is how to maximally exploit and use the experience gained from previous trials to reduce the amount of search needed to find the optimal action. The player may elect to use some form of a gradient search. For example if he/she notices that lower hits resulted in success, then probably she/he will try even lower hits. Another technique he/she may use is some kind of random search. Random search techniques would probably work better if the outcomes of the different actions are stochastic.

The above example illustrates the major differences between optimal control and learning optimal control. Unlike optimal control, in learning the optimal action we do not have full a priori information about the consequences of the different actions under the different states of the system and environment. This information is built and updated as the learning system performs the task. In this case, the system's actions should be a mixture of exploration as well as optimizing performance under the given goal and constraints. This mixture of exploration and optimization has been recognized by control theorists and has been given the name of dual control by Feldbaum [Feldbaum, 1965]. Another issue, illustrated by the tennis player example, is the issue of choosing an action. We have seen that this action depends on an immediate goal and that this goal may depend on an even higher goal. This issue has also been mentioned by Bellman [Bellman, 1978]. For example, in the tennis player

example, the immediate goal could be maximizing information gain or exploration, or maximizing the probability of winning a point. The choice between these goals may be governed by the higher goal of winning the game, which may still be governed by more ulterior motives and goals.

We may think of yet another level of optimization, which I prefer to call here development. In this type of learning, the constraints themselves that restrict the learning behavior or restrict the possible set of actions are modified in order to improve performance. For example a better representation of previous experiences, or better choice of relevant measurable parameters may lead to better generalization. A better choice of hardware parameters may increase the range of available actions. In the case of the tennis player for example, muscle properties may change with practice, so that the player may gain more speed and strength and flexibility. Indeed a big part of the training of athletes in different sports is designed to improve the biological hardware of the system. In the case of a robot arm, for example, this type of learning for example could guide the choice of the different parameters of the arm, e.g. masses, inertias, number of joints and their shapes. This type of task optimization is beyond the scope of this thesis and will not be addressed further.

All the levels of learning and task optimization mentioned above can be viewed as a closed loop feedback system. Some feedback from the environment drives the learning and the change in the actions performed. There is another type of learning, called unsupervised, where the learner tries to find the similarities between the experiences in order to better classify and represent them. However, this type of learning is not explored here.

1.2 Thesis Objectives

The goal of this work is to explore the practical problems of learning and task optimization through simulation of simple motor tasks. Algorithms for the solution of

these problems will be proposed based on techniques adopted from different fields. We will focus primarily on model based task optimization techniqes. The main role of the model is to allow the learner to simulate the outcome of its actions before acting. Through this mental practice, the learner can find good solutions. This is particularly desirable when real experimentation is expensive.

This research can be divided into two main parts. In the first part I will focus in particular on the problem of representation of previous experiences and building models. I will study and implement a particular non-parametric technique called radial basis functions (RBF) and its variants, generalized basis functions (GRBF) and Hyper basis functions (Hyper BF) [Poggio and Girosi, 1990]. I will show how this technique can be successful in learning the inverse dynamics of robot arms [Botros and Atkeson, 1991], and how it can be implemented efficiently. I will also demonstrate how to apply this technique, or any other function approximation technique for the optimization of static and dynamic systems. The second part of this research will focus on the problem of task optimization: how to learn the best possible action relative to a particular objective, given our current knowledge and experience, and how to increase this knowledge and modify it to achieve better performance through the use of a model. I will address the problem of learning actions in both deterministic and non-deterministic environments. I will explore the problems involved in non-deterministic environments by simulating a simple problem of basketball shooting. I will also talk about how to optimize actions and action parameters by simulating the task of bouncing a ball with a racquet. I will show through simulation how the trajectory of the racquet can affect the success rate and quality of bouncing, and how the learning system can optimize those parameters automatically.

The main approach I am taking in this research is a synthetic approach, implementing learning and optimization algorithms on artificial systems. We hope, this approach will help us appreciate the problems involved in biological motor learning and task optimization and evaluate the possible different approaches to solving these

problems. From the technological point of view, our ultimate objective is to try to automate the design of machines (both hardware and software) based on learning and task optimization, especially in the case where the machine and the task to be achieved is difficult to model a priori.

1.3 Why Model Based ?

Although many of the task optimization problems we present in this thesis can be solved without using a model, we will mainly focus here on model based techniques for the exploration and optimization of actions. We believe that building models of the response of the environment can serve many useful purposes in task optimization and results in better use of the previous experiences.

- It allows for a better directed search for the optimal action, for example along a gradient.

- Models may also help in better credit assignment, that is finding which actions are responsible for the achieved performance. This credit assignment may be difficult in cases where the effect of actions is delayed.

- Models also allow for the transfer of knowledge in the case when the goal of the task changes.

- For humans, the availability of a mental model aids in the simulation and prediction of the response of the environment. This will help in better planning of actions. For example, in the case of the tennis player, the availability of a mental model of the dynamics of the ball will allow the player to anticipate where the ball will land and then plan his/her movement so that he/she can intercept the ball from the best possible position.

  Work on mental imagery and mental rehearsal has shown that humans can improve their performance by merely imagining the performance of a task

[Adams, 1990]. A model based task optimization may partially explain this behavior.

A model can have many forms based on the nature of the task and the available information. In this thesis, we will only explore tasks that have continuous state and action spaces. We will use general function approximation techniques to represent the models built from experience.

1.4 Function Approximation in Motor Learning and Task Optimization

There are many different ways that function approximation techniques such as neural networks may be applied to motor learning and task optimization problems. These techniques are summarized by Atkeson [Atkeson, 1991]. A recent survey for the application of neural networks in control is given by Hunt et al. [Hunt *et al.*, 1992]. Over the past few years, there have been many attempts for applying neural networks in control, using different control and network architectures, with varying degrees of success. [Psaltis et al., 1988; Khalid and Omatu, 1992; Narendra and Mukhopadhyay, 1992; Schiffmann and Geffers, 1993; Chen and Khalil, 1992; Pao et al., 1992; Levin and Narendra, 1993; Kuschewski et al., 1993; Sanner and Slotine, 1992; Jordan and Rumelhart, 1992; Atkeson, 1991] and the papers cited therein are among the recent papers that have examined the application of neural networks in control.

The most direct way for using a neural network as a controller is what is called an inverse model. This is simply an association, having as its input the desired output(s) o_d and the states of the system s, and as its output the action(s) that should be applied as shown in Figure 1-1. One problem of this approach is that this mapping may not exist as in the case where there are many possible actions that have the same desired outcome. In this case preprocessing of the data may be necessary to prevent such conflict and erroneous results. In the case where the inverse model is used with an

optimal control scheme, only one action is chosen as optimal for each desired outcome and state of the system. In this case the inverse model is just a way of encoding an optimal control policy as used by Jordan for example [Jordan, 1989]. Psaltis, Sideris and Yamamura [Psaltis et al., 1988] have also distinguished between generalized and specialized inverses. Kawato and Gomi proposed another training algorithm for learning the inverse model by feeding back a function of the error between desired and actual performance [Kawato and Gomi, 1993]. However, the feedback-error-learning algorithm does not necessarily result in a correct inverse model. Kawato and Gomi also proposed that the feedback-error-learning may be a good model for cerebellar learning, where the feedback error is provided by the climbing fibers input to the Purkinje cells of the cerebellum [Kawato and Gomi, 1992]. Based on experimental measurements during eye-tracking experiments, inverse dynamics computations have also been proposed as a possible model for the processing of eye position, velocity and acceleration by the Purkinje cells of the ventral paraflocculus of the cerebellum [Shidara, Kaeano, Gomi and Kawato, 1993].

Another possible way for using function approximation to encode the previous experiences is to encode the outcome of an action as a function of the states and actions. This has been called forward modeling. Unlike inverse models, forward models always exist for deterministic systems. When using a forward model, the answer to a particular query, i.e. to find the action to take to achieve a certain desired outcome for a given set of states, is not as straightforward to obtain as in the case of the inverse model and can be performed using root finding or optimization techniques [Atkeson, 1991]. Ito proposed another possible configuration in which the forward dynamics model is used in an internal feedback loop [Ito, 1993]. In this configuration, the forward model receives an efferent copy of the motor command, and provides error feedback which improves the motor command. This internal loop acts in conjunction with the limb feedback loop.

There may exist more than one function approximation network in a single control

system. For example, forward models can be used for the plant dynamics and for a controller. The role of the plant dynamics forward model is to predict the response of the environment and to propagate back the error in the output[Jordan, 1989]. More generally, it can be used to propagate back any performance gradient in order to adjust the controller parameters. An inverse model may be used to make an initial guess for the root finding algorithm for the forward model and also to correct the errors in approximating the forward model.

Function approximators used as classifiers are another potential application for function approximation in control. The classifier acts as a nonlinear switch, which switches between a discrete set of controllers based on the states and desired outputs of the system. The switch may also change smoothly from one controller to the next. This can be thought of as a more generalized method of gain scheduling which is used in the control of nonlinear systems. The controllers themselves need not be fixed and can be some other function approximators, which may be useful, for example, when the controller function varies considerably in the different areas of the state and output space (e.g. the controller or the behavior of the system depends on the different sets of states in different areas of the state space.). This is one possible application for the competitive networks paradigm described by Jacobs et al. [Jacobs, Jordan, Nowlan and Hinton, 1991]. Narendra and Mukhopadhyay [Narendara and Mukhopadhyay, 1993] have also suggested the use of neural network classifiers as a switch to select a controller for the case when it is known a priori that the controlled plant can only be in one of a finite number of configurations.

Neural networks may also be used as state observers for state feedback control when not all the states can be measured. Recurrent neural networks have been previously proposed and tested for such tasks.

Another potential use of function approximation in control is to build a model of some measure of performance of the system as a function of the states, actions and outputs, and then use this model to find the actions that optimize this measure of

performance using optimization techniques such as dynamic programming or calculus of variations.

$$a^* = \{a | \min_a f(a, s, o)\} \tag{1.2}$$

subject to the constraint:

$$s_{k+1} = g(s_k, a_k) \tag{1.3}$$

where a^* is the optimal action or action sequence that minimizes the approximated cost function f. The function g describes the state evolution of the system. Both f and g can be modeled using function approximation. Reinforcement learning techniques are examples of such use of function approximation techniques [Sutton, 1992]. Pao, Phillips and Sobajic [Pao et al., 1992] have proposed a similar approach for finding the optimal control trajectory with respect to some cost function, by modeling this cost function as a function of the control trajectory using a neural network and then optimizing it. Some researchers have also studied and applied Hopfield Networks for finding the optimal control law of linear systems [Mears et al., 1993; Lan and Chand, 1990].

1.5 Thesis Contributions

- We have studied the approximation of forward and inverse dynamics using Radial Basis Functions. It is shown using simulations that the norm metric used to scale the experiences plays a very important role for good modeling and generalizations from examples. Previous heuristic methods to choose this norm metric did not take into account the relationship between the inputs and outputs of the system and used only the distribution of the inputs to find the metric. Through simulations, it is shown that this is not sufficient to obtain good approximation, especially when the dependency of the output of the network on the different inputs varies considerably. We have developed an efficient heuristic method for

finding a good norm metric for the case of Gaussian HyperBFs that takes into account the sensitivity of the output to the different inputs. It is found that the results obtained with this method approximate those obtained by using nonlinear optimization techniques, but the new method is much faster.

- We have developed a neural network based function optimization algorithm. This algorithm attempts to model the objective function using a neural network. The algorithm adaptively selects an exploration strategy based on the estimated prediction error of the network. Using many simulations, it is found that this algorithm can reduce the amount of experimentation needed to find the optimal parameters. In addition it is less prone to noise in observation than gradient based optimization algorithms.

- We have developed and tested a general method for learning the optimal actions of an unknown dynamical system given a performance index. This method works by incrementally updating a model of the dynamical system and then using this model to find the open loop optimal actions. We have identified and solved some of the problems associated with the implementation of this approach, and we have implemented it successfully in a variety of simulations. We have also proposed and tested, using simulations, different possible approaches for learning a closed loop optimal controller.

- We have implemented and demonstrated the use of model based techniques in learning the optimal actions for systems with stochastic outcomes. We show how function approximation can be used to learn both probabilities of success/failure and expected reinforcement which then can be optimized subject to any additional constraints.

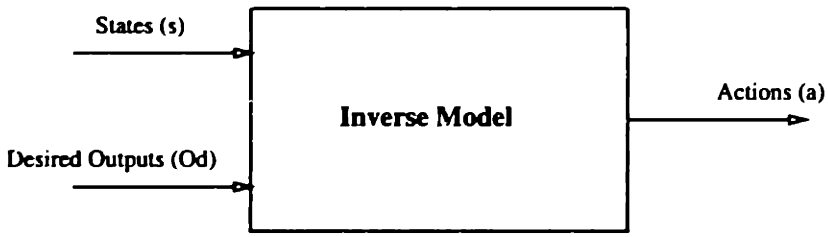

Figure 1-1: Learning the inverse model

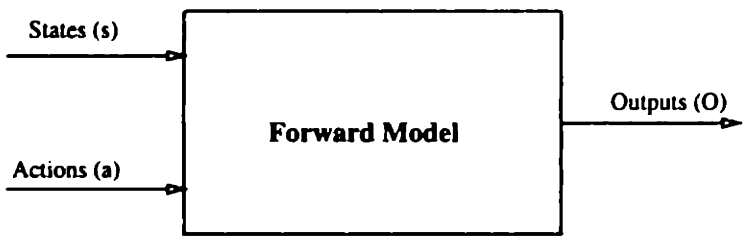

Figure 1-2: Learning the forward model

Chapter 2

Knowledge Representation

2.1 Introduction

In order to make use of past experiences, both in the case of supervised learning and task optimization, the learning system has to have a mechanism of knowledge representation in order to store those experiences in an efficient way where the information can be easily modified, added and retrieved. There are many possible different paradigms that exist for knowledge representation derived from different fields such as classical artificial intelligence, statistics and mathematics. Among these methods are:

1. Symbolic representations where the experiences are transformed and encoded into a set of rules. Decision trees may be considered as one example of those techniques.

2. Statistical representations where the knowledge is represented by a set of probability distributions that are modified by experience. Bayesian techniques may fall under this category.

3. Fuzzy representations, where the knowledge is coded as fuzzy associations between inputs and outputs.

4. Function approximation techniques of knowledge representation which include methods such as parametric structured models, nearest neighbors, local regression, feedforward neural nets, generalized splines, other basis functions and other non-parametric methods.

Hybrid techniques which combine the best features of some of these techniques may also be used. The best method to use for a particular knowledge representation application depends on the quality and amount of data that is available and the properties of the function to be approximated. The amount of the a priori knowledge incorporated into the function approximation reduces the search for the best parameters and the number of examples required. For example a structured dynamical model with few parameters to estimate is best if we can accurately model the dynamics of the system. It requires relatively very few experiences and generalizes very well. Non parametric techniques are generally more flexible about what functions they can model but require much more training. The ability to generalize from examples depends on the degree of smoothness of the function to be approximated, the number of examples available, the number of dimensions of the input space and the approximation method used. The simplest example that illustrates the effect of the different factors is the well known digital uniform sampling of a continuous function. We can only recover the original function from its samples exactly if the frequency of sampling in each dimension is higher than twice the highest frequency in that dimension and if we convolve the samples with a sinc function which has a width that depends on the frequency content of the function along that dimension. This example illustrates that the number of samples needed to recover the function increases linearly with the frequency content of the function in a given direction and exponentially with the number of dimensions. In the case of non-uniform samples the problem is much more complex. In general, the optimal rate of convergence of a function as a function of the smoothness p, the number of samples n and the number of dimensions d is of the form $\epsilon_n = n^{-\frac{p}{2p+d}}$ [Poggio and Girosi, 1989]. This equation shows the problem of the curse

of dimensionality, as the number of dimensions increases, the number of samples n required to achieve a specified error rate increases almost exponentially (if p is small). Therefore if the number of dimensions increases, it becomes very important to exploit the smoothness of the function in the different directions and detect directions where the function is slowly varying. This will help us achieve better generalization for the same amount of data.

In this thesis, we focus primarily on the method of radial basis functions (RBF) and its variations, generalized RBFs and Hyper-BFs. Some researchers have criticized the use of the local radial basis functions such as Gaussians on the ground that the number of RBFs required to cover the space becomes very large with the increase in the number of dimensions. This is based on the need to cover the whole input space where there is data with basis functions, regardless of irrelevant dimensions [Weigend et al., 1990; Hartman and Keeler, 1991]. Other researchers believe that the problem of scaling the width of RBFs in different dimensions is hard [Hartman and Keeler, 1991]. In this chapter, I will show how it is possible to efficiently improve the generalization properties of RBFs by finding efficient methods for scaling the variables in the different dimensions. The scaling factors are also a measure of the relative dependence of the function on the different input variables. Moreover, with a good norm metric it is possible to cover the input space with relatively very few RBFs.

However, it should be emphasized that the method of RBFs with one global set of scaling parameters is not always suitable for all applications. For example, it does not work well if the distribution of the data in the input space is not very uniform. Some prefiltering of the data is sometimes needed to remove data that are very close together in the input space, so that the problem does not become ill-conditioned.

2.2 Radial Basis Functions for Function Approximation

2.2.1 Background

The Radial Basis Functions (RBF) approach to approximating functions consists of modeling an input output mapping as a linear combination of radially symmetric functions [Powell, 1987; Poggio and Girosi, 1990; Broomhead and Lowe, 1988; Moody and Darken, 1989]. It was first developed as an exact interpolation approach, that is, it reproduces the outputs of the given examples exactly. The output of the interpolating function is described by the following equation :

$$y(\mathbf{x}) = \sum_{i=1}^{n} C_i \phi(\|\mathbf{x}_i - \mathbf{x}\|) \tag{2.1}$$

For the exact interpolation case, n is equal to the number of examples, C_i 's are the coefficients to be estimated, $\mathbf{x}_i$ is the vector of inputs at the example i. Sometimes a polynomial term of the form $\sum_{j=1}^{p} \mu_j P_j^m(\mathbf{x}_i)$ is added to the above equation for certain types of RBFs. In this case, since the number of parameters is larger than the number of data, the following extra constraints are added to make the parameter estimation problem well posed [Powell, 1987].

$$\sum_{j=1}^{n} C_j P_i^m(\mathbf{x}_j) = 0 \qquad i = 1, \ldots, p \tag{2.2}$$

Examples of RBFs include :

- Gaussians $\phi(r_{ij}) = \exp(\frac{-r_{ij}^2}{c^2})$
- Hardy Multiquadrics [HMQ] $\phi(r_{ij}) = \sqrt{r_{ij}^2 + c^2}$
- Hardy Inverse Multiquadrics [HIMQ] $\phi(r_{ij}) = \frac{1}{\sqrt{r_{ij}^2 + c^2}}$

- Thin Plate Splines [TPS] $\phi(r_{ij}) = \left\{ \begin{array}{ll} r_{ij}^{(2k-d)} \log r_{ij} & d \ even \\ r_{ij}^{(2k-d)} & d \ odd \end{array} \right\} \quad 2k > d$

  where d is the dimension of the inputs and k is a smoothness parameter.

- Cubic Splines [CS] $\phi(r_{ij}) = r_{ij}^3$

- Linear Splines [LS] $\phi(r_{ij}) = r_{ij}$

where $r_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$. Some of these RBFs (e.g. Gaussians and multiquadrics) have an explicit width parameter c that needs to be determined. However we can also fix this width parameter, for example to have the value 1 and scale the data instead.

To find the coefficients C_j for exact interpolation, we have to invert a square matrix which is theoretically guaranteed to be nonsingular for a wide class of radial basis functions given of course distinct data [Micchelli, 1986]. Since the equations are linear, there are a number of batch and recursive algorithms that exist for finding the exact value of the coefficients. The linearity of the function with respect to the coefficients C_j guarantees the convergence to the globally optimal parameters. Since many of the optimization algorithms require the inversion of a matrix of rank n, the computational complexity for finding the optimal parameters is $O(n^3)$, where n is the number of data points and also the number of coefficients in the case of exact interpolation. Exact interpolation may not be desirable if the data are noisy or if the computational burden is high.

2.2.2 Mathematical Interpretation of RBFs

As mentioned by Poggio and Girosi [Poggio and Girosi, 1989], many of the radial basis functions are the Green functions obtained by solving different regularization problems of the form :

$$\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 + \lambda \|Pf\|^2 \tag{2.3}$$

as $\lambda \rightarrow 0$. Where P in the above equation is a radially symmetric differential operator, and $\|\cdot\|$ is the L^2 norm. For example Gaussian RBFs result from operators P of the

form:

$$\int_{R^d} dx \sum_{m=0}^{\infty} a_m (P^m f(x))^2 \tag{2.4}$$

where $P^{2m} = \nabla^{2m}$ and $P^{2m+1} = \nabla\nabla^{2m}$, ∇^2 is the Laplacian operator and the coefficients $a_m = \frac{\sigma^{2m}}{m!2^m}$.

Interpolation using thin plate splines minimizes the functional:

$$\mathcal{J}_k(f) = \int_{R^d} dx \sum_{|\nu|=k} (D^\nu f)^2 \tag{2.5}$$

Since regularization is also related to Bayesian estimation, we can think of RBFs as a special case of Bayesian estimation [Girosi, Poggio and Caprile, 1990], where the prior probability of the function f is assumed to be

$$P(f) \propto \exp^{-\lambda \|Pf\|^2} \tag{2.6}$$

Another interesting interpretation for some forms of RBFs made by [Schagen, 1980] is to regard the given training examples as point realizations of a stationary stochastic process $Z(\mathbf{x})$. The stationarity of $Z(\mathbf{x})$ implies that the mean and variance of the process at any point are constants and that the covariance between two points is only a function of the difference between these two points. If we make the stronger assumption that the covariance depends only on the distance between two points, that is $Cov[Z(\mathbf{X}_a), Z(\mathbf{X}_b)] = \sigma^2 g(\|\mathbf{X}_a - \mathbf{X}_b\|)$, where σ^2 is the variance of the process, and given that the function $g(r)$ should satisfy the covariance properties, namely $g(0) = 1$, $g(r) \leq 1$ for $r \geq 0$ and that the covariance matrix should be nonnegative definite, some RBF solutions may be interpreted as the best linear unbiased estimate of the stochastic process given the data points. For the derivation of this result we refer the reader to the paper by Schagen [Schagen, 1980]. Gaussian RBFs satisfy all these assumptions and constraints. Note that not all RBF functions mentioned above satisfy all the assumptions required for this interpretation. RBFs with increasing function

values as the distance from the center increases can not model a covariance since $g(r) \geq g(0)$ for some $r > 0$

Both of the above interpretations of radial basis functions make some a priori assumptions about the degree of smoothness of the function to be approximated. These a priori assumptions determine the shape of the radial basis function used.

We can also think of RBF approximation as another adaptive linear kernel technique for approximation, similar to local weighted averaging. The approximation at a particular input can be viewed as a weighted average of the neighboring data points. The shape of the kernel is determined entirely by the distribution of the input data and varies from one point to the next. It is very important to differentiate between the shape of the RBF bases and the shape of the associated kernel. The kernel associated with a particular RBF $\phi(\mathbf{x})$ can be expressed mathematically in matrix form by the following equations:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{n} \psi(\mathbf{x}, \mathbf{x}_i) y_i \tag{2.7}$$

$$\psi(\mathbf{x}, \mathbf{x}_i) = \Phi^T(\mathbf{x})(A(\mathbf{x})^T A(\mathbf{x}))^{-1} A(\mathbf{x})^T \tag{2.8}$$

where $A(\mathbf{x})$ for an RBF network with n_c centers and n_p data points is an $n_p \times n_c$ matrix with elements

$$A_{ij} = \phi(\|\mathbf{x}_i - \mathbf{c}_j\|), \quad i = 1, \cdots, n_p, \quad j = 1, \ldots, n_c \tag{2.9}$$

and $\Phi(\mathbf{x})$ is an n_c- dimensional vector representing the response of the different RBF units at a particular input $\mathbf{x}$. An example of a kernel in one dimension for a data set uniformly spaced in the input space from [-1, 1] is shown in figure 2-1 for the cubic RBF. Note that although for a cubic RBF $\phi(r) \rightarrow \infty$ as $r \rightarrow \infty$, the kernel associated with it is local. The shape of the kernel depends on the distribution of the data and the centers and on the location of the query point. This effect is higher near the boundaries of the data set. The localization properties of RBFs are discussed in

more detail in [Powell, 1988; Jackson, 1988].

2.2.3 Extensions to RBF: GRBF and HyperBF

To reduce the problems of exact interpolation, many researchers have suggested using a smaller number of basis functions [Broomhead and Lowe, 1988; Poggio and Girosi, 1989; Moody and Darken, 1989]. In this case it is not possible to reproduce the exact outputs in general. To choose the centers of the basis functions we can use optimization techniques [Poggio and Girosi, 1989] or also heuristic algorithms based on the distribution of the data [Moody and Darken, 1989]. The RBF approximation with movable centers has been called Generalized Radial Basis Functions (GRBF) [Poggio and Girosi, 1989]. The estimation of the location of the centers of the GRBFs using least square error optimization techniques is not an easy problem. This is because the error surface is not convex and the number of parameters to be estimated is large. From my personal experience with different computer simulations and using second order nonlinear optimization techniques, I found that it is very hard to adjust the centers and that usually the gain in performance is small.

Another extension to the RBF approach, described also by Poggio and Girosi [Poggio and Girosi, 1989] is known as Hyper Basis Functions (HyperBF). This is a further generalization of the GRBF technique which includes using radial basis functions of different widths or also non-radial basis functions. Similar types of basis functions have been described by Saha et al. [Saha et al., 1991] for image coding and analysis and have been termed Oriented Non-Radial Basis Functions (ONRBF). Saha et al. suggested a gradient descent algorithm to find the parameters of the ONRBFs. Varying the widths of the RBFs is also equivalent to using a general norm rather than the Euclidean norm to compute the distance of a point from the center of the basis function. The equation describing the output in terms of the basis functions and the

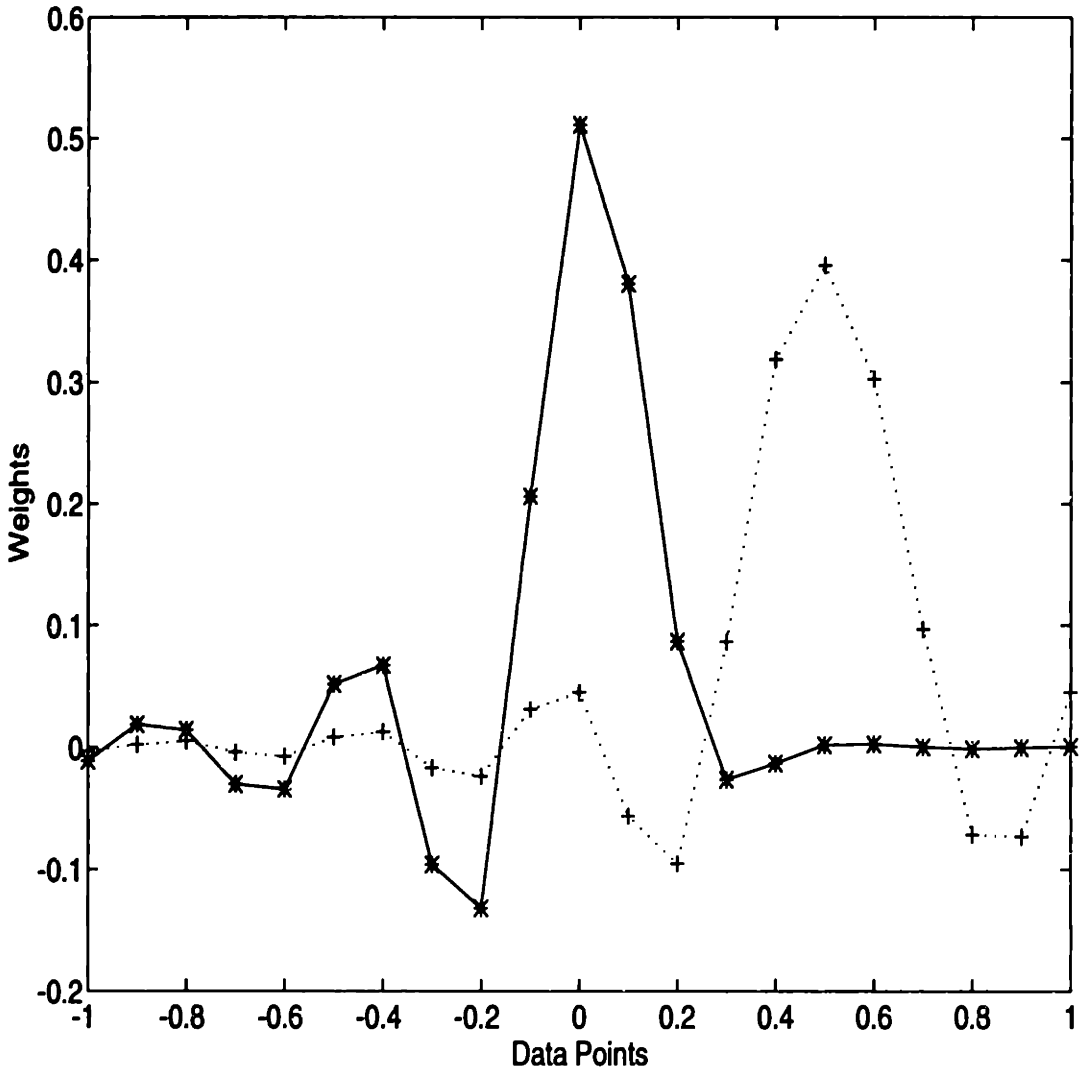

Figure 2-1: Kernel shapes $\psi(\mathbf{x}, \mathbf{x}_i)$ for query points at 0.05 (solid lines) and 0.5 (dotted lines) using a cubic RBF. Location of data points are marked. RBF centers are located at every other data point.

different inputs is as follows :

$$y(\mathbf{x}) = \sum_{i=1}^{n} C_i \phi(\|\mathbf{x}_i - \mathbf{x}\|_W) \tag{2.10}$$

where $\|\mathbf{x}_i - \mathbf{x}_j\|_W^2 = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{W}^T \mathbf{W} (\mathbf{x}_i - \mathbf{x}_j)$ and $\mathbf{W}$ is a square matrix. Since the matrix $\mathbf{Q} = \mathbf{W}^T\mathbf{W}$ is symmetric and at least positive semidefinite, it is possible to factorize it into $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is a lower triangular matrix. Therefore only $d(d+1)/2$ elements need to be estimated. The rest of the elements are redundant. It is also possible to estimate the elements of the symmetric matrix $\mathbf{Q}$ directly, but in this case there is no guarantee that the matrix Q will be positive definite and matches the definition of a general norm. Another important point is that if the matrix $\mathbf{W}$ is singular, it is important to redefine the input variables and remove the irrelevant dimension, otherwise Micchelli's results about the uniqueness of the coefficients of the basis functions C_i do not apply, since in this case two different input vectors may appear to be the same if the difference between the two input vectors lie in the null space of the matrix $\mathbf{W}$. From practical experience, it is found that the $\mathbf{W}$ matrix plays a very important role in the quality of generalization. This is especially true for functions which do not meet the smoothness assumptions implied a priori by using a certain type of RBFs. In the next section I will describe different possible ways for estimating the $\mathbf{W}$ matrix, and in section 2.2.5 I will show the results of applying these techniques for approximating different multidimensional functions.

2.2.4 Estimating the Weight Matrix W for Gaussian RBFs

There are many possible ways for estimating the weight matrix $\mathbf{W}$. One class of methods, based on optimization techniques, is to find a $\mathbf{W}$ which minimizes the sum of the square of the errors between the output of the RBFs and the training set output. Nonlinear optimization techniques that could be used include gradient descent, second order nonlinear optimization techniques or variations of random search. Gradient and

second order methods are not guaranteed to converge to the global minima and are sensitive to the initial choice of parameters. Also, for large amounts of data and RBFs, the amount of computation involved in second order methods becomes very large. In random search the amount of computation for each step is relatively small, but a very large number of steps may be needed to converge, especially when the number of parameters to be estimated is large. The main advantage of random search is that it can escape from local minima. Caprile and Girosi present a simple random search technique that have been found to work well in practice [Caprile and Girosi, 1990].

Another alternative method for determining the best $\mathbf{W}$ for diagonal $\mathbf{W}$ is by using cross-validation techniques to estimate the W's in the different directions. This has the advantage, over minimum training error techniques, that it will attempt to minimize the predicted mean square error as opposed to the mean square error of the training set only and therefore may generalize better. Hutchinson et. al. [Hutchinson, 1984] have attempted to use generalized cross validation to find scaling parameters for one input variable. More work is needed to generalize this technique to more than one input variable in an efficient way. However, in general, cross validation techniques tend to be computationally very expensive and it may be very difficult to adapt these techniques to real time applications.

Empirical estimation of Gaussian RBF widths

Many researchers have explored some heuristic methods for the estimation of the RBF widths and center locations. Moody and Darken [Moody and Darken, 1989] describe methods based on adaptive clustering of the input data. In their analysis, they totally ignore the characteristics of the function to be approximated. Platt describes a resource allocating network that adaptively adds basis functions based on a novelty measure [Platt, 1991]. The novelty measure is based on two factors: the accuracy of the approximation and the distance of the new experience from the previous data points. The width of the RBFs is proportional to the distance to the k-nearest

neighbor. Although the output data in this method are used in the choice of the center locations, the estimation of the RBF widths is still completely dependent on the input distribution only. Hutchinson proposes a heuristic algorithm for finding a reasonable set of initial values of the parameters of the RBFs [Hutchinson, 1993]. His algorithm is a generalization of Moody and Darken algorithm and allows for the possibility of estimating the widths of RBFs based on the output as well as the input data. Methods that depend only on the input distribution to determine the RBF width parameters may not work well if the dependence of the function to be approximated in the different directions is not uniform. Mel and Omohundro describe a method that depends on the second order derivatives of the function to be approximated with respect to the different input variables [Mel and Omohundro, 1991]. In this chapter I will describe new methods for estimating a diagonal $\mathbf{W}$ for Gaussian RBFs based on approximating the first order partial derivatives of the function to be approximated with respect to the different input variables. Although this class of methods is not proven to optimize any cost function, it is found to approximate and sometimes surpass the results obtained using nonlinear optimization techniques. The diagonal width parameters are assumed to depend on the average variation of the function in each direction, as measured by the sum of the square of the first partial derivatives in each direction, in addition to the variance of the input data in the different dimensions. Let

$$\mathbf{g} = \int_{R^d} \left[\left(\frac{\partial f}{\partial x_1}\right)^2 \left(\frac{\partial f}{\partial x_2}\right)^2 \cdots \left(\frac{\partial f}{\partial x_n}\right)^2 \right]^T d\mathbf{x} \tag{2.11}$$

and $\overline{\mathbf{g}} = \frac{\mathbf{g}}{\|\mathbf{g}\|}$. We found empirically that a good approximation for the diagonal of $\mathbf{W}$ is as follows :

$$w_{ii} = \overline{g}_{ii} \frac{k}{\sqrt{E\{(x_i - t_i)^2\}}} \tag{2.12}$$

where the subscript i denotes the i^{th} input variable, E denotes the expected value and t_i is the i^{th} component of the centers of the RBFs. k is a constant that can be determined by cross validation or least mean square optimization. From simulations

we found that the approximation is not very sensitive to a range of the values of k. However, the choice of a good k is important for the conditioning of the coefficients of the RBF. The bigger the value of k, the smaller will be the equivalent width of the RBFs and the estimation of the coefficients will be less singular.

To understand why this suggested form of $\mathbf{W}$ makes sense, we can divide the equation for $\mathbf{W}$ into two terms. The first term is the normalized gradient functional and the second is a normalization factor that normalizes the input space so that the different inputs have approximately similar range of values. The gradient functional term results in making the first order terms in the regularizer operator for Gaussian RBFs (equation 2.4) to have approximately equal magnitude. This, in turn, satisfies the assumptions made by the regularizer. Intuitively, the width of the Gaussian functions will be smaller in the directions where the approximated function varies the most. Note also that if one variable is irrelevant, its derivative function will be zero and therefore the corresponding w component will also be zero.

The idea of separating the estimation of the norm metric from the estimation of the other function approximation parameters has been recognized by many other researchers [Girosi, 1992; Moody and Darken, 1989; Samarov, 1991; Li 1992; Zhao, 1992]. Some of these researchers have also suggested the use of different forms of derivative functionals for other function approximation techniques [Samarov, 1991; Zhao, 1992; Li, 1992]. Both Samarov and Li have described methods for estimating these derivative functionals or expected derivatives from the data and mentioned the assumptions under which these estimations are valid.

Iterative Estimation of the Diagonal Weight Matrix W

I will describe here a new iterative procedure for estimating the derivative functional that seems to work well in practice. This procedure starts by first estimating the function using a diagonal $\mathbf{W}$ matrix equal to the inverse of the variance of the input data in the different directions for example, and then estimating the derivative functional

using the approximated function. We then use the derivative functional to update the value of the **W** matrix and use this latter to improve the function approximation. In practice, this procedure usually converges in 3 or 4 iterations. However a more detailed mathematical analysis is needed to understand the convergence properties of this algorithm. From the limited number of simulations I have done using this algorithm, it is found that it always converges to some value for **W**, however in cases where the original approximation is very bad (high RMS error), it may not find the best answer.

The second term in the suggested approximation for **W** is added to normalize the input variables originally so that they will have approximately similar variances.

2.2.5 Test Results

Approximating a 2-D function

The first numerical example is the approximation of the Rosenbrock's banana function [Luenberger, 1973]. This is a function of two variables with a steep ravine shaped like a banana described by the following equation:

$$f(x) = 100({x_1}^2 - x_2)^2 + (1 - x_1)^2 \tag{2.13}$$

We used 40 training and test points distributed randomly and uniformly over the range from [-2, 2] in each of the input dimensions, and used 25 Gaussian RBFs to approximate the function under the following conditions:

1. The centers and shapes of the Gaussians are fixed.

2. The centers are fixed, but the diagonal weight matrix **W** is estimated using the sequential technique described above.

3. As in 2, however in this case the true function is used to estimate the derivative functional.

4. As in 2, but a Levenberg-Marquardt optimization technique is used to estimate the parameters.

The average RMS errors on the test set of 10 independent simulations (each one has a different training and test sets) are shown in table 2.1 for the 4 different methods described above, together with their standard deviations. As is shown in the table, the results obtained using the derivative functional are better than the other methods and are less varied in both the cases where we use the true expression for the derivative functional and when we use the new sequential method. The average RMS error when using the Levenberg-Marquardt algorithm to estimate **W** (column 4 in table 2.1) is worse than the sequential method and the RMS error is more varied. The reason for this large variation in the error is that the optimization technique sometimes converges to a local minimum and is also sensitive to the initial conditions. It is important to note here that the sequential algorithm usually converges in 2 to 3 iterations and the computation is much faster than the optimization algorithm. An advantage that the sequential algorithm has over the optimization one is that in the sequential algorithm we can set the number of centers equal to the number of data in the training set, since we do not use optimization to estimate **W**. This is not possible in the direct optimization case since the number of parameters to estimate will be larger than the number of data points, and the problem will be singular. Setting the number of centers equal to the number of data points reduces the error in the training set to zero and also considerably improves the error in the test set and the algorithm still converged faster than the Levenberg-Marquardt optimization technique. Another important point to mention here is that although the error on the test set for using Gaussians with fixed centers and shapes is very bad in this case, this method will give good results if the number of training data is increased and the number of Gaussians increased, i.e. the effect of a good choice of a weight matrix **W** is much less important when data and RBFs are abundant.

number of simulations = 10	1	2	3	4
mean	0.2445	0.0967	0.0933	0.1285
std	0.1437	0.0406	0.0439	0.1101

Table 2.1: The mean and standard deviation of the normalized RMS error on the test set using different RBF optimization techniques (See text).

To show the relation between the **W** matrix obtained using the derivative functional and the Levenberg-Marquardt optimization, I plotted w_{11} vs. w_{22} in figure 2-2. In this figure, the slope of the line drawn by the "x" symbol represents the ratio of w_{22} to w_{11} computed using the derivative functional from the true function. The open circles represent the values of **W** plotted at each iteration of the Levenberg-Marquardt algorithm. The "*" symbol represents the values of **W** after each iteration of the sequential method. The second iteration for that particular run falls almost exactly on the true derivative functional line. It is also interesting to note that the values obtained from the Levenberg-Marquardt method approaches this same line asymptotically. This is the case in most of the simulations performed, given that the optimization does not get stuck in a local minimum.

Approximation of Wood's function

Wood's function is another function used to test optimization algorithms. It is a 4 dimensional function described by equation 2.14.

$$\begin{aligned} y &= 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2 + 90(x_4 - {x_3}^2)^2 + (1 - x_3)^2 + \\ &\quad 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \end{aligned} \tag{2.14}$$

Like in the banana function case, the rate of change of the function is different in the different directions. This is a good example to illustrate the effect of the norm metric and the choice of **W** in approximating this function using HyperBFs. The range of the input data is assumed to be from [-2, 2] in all directions. The training set contained 100 data points uniformly and randomly distributed in the input

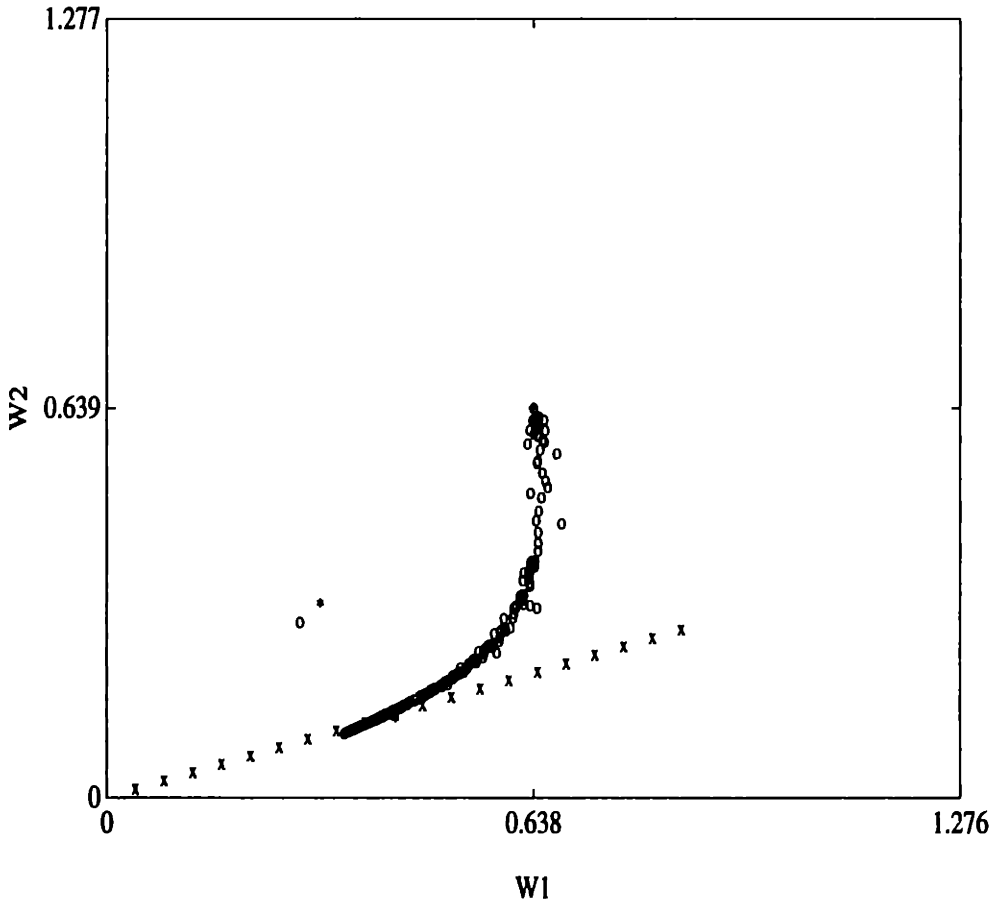

Figure 2-2: The values of the parameters of **W** used in approximating the banana function, using different optimization methods. Note how the parameters converge towards an asymptotic line that is estimated using the heuristic method for the choice of parameters described in the text.

range. We used 40 HyperBF functions with fixed centers, distributed randomly in the 4 dimensional space. We compared the use of nonlinear optimization techniques with the heuristic sequential method we developed. These results confirm the results obtained in approximating the 2-D banana function. The trajectory of the estimated parameters obtained using the Levenberg-Marquart nonlinear optimization technique is shown superimposed on the parameters obtained using the heuristic method in figure 2-3. As shown in the figure the parameters converge to approximately the same values.

Modeling the Inverse Dynamics of a Robot Arm

In this example I will describe the modeling of the ideal inverse dynamics of a simulated two joint planar robot arm. The inverse dynamics equations of the arm model are given in equation 2.15.

$$\begin{aligned} T_1 &= \ddot{\theta}_1(I_1 + I_2 + 2m_2c_{x_2}l_1\cos\theta_2 - 2m_2c_{y_2}l_1\sin\theta_2) \\ &\quad +\ddot{\theta}_2(I_2 + m_2c_{x_2}l_1\cos\theta_2 - m_2c_{y_2}l_1\sin\theta_2) \\ &\quad -2l_1\dot{\theta}_1\dot{\theta}_2(m_2c_{x_2}\sin\theta_2 + m_2c_{y_2}\cos\theta_2) \\ &\quad -l_1\dot{\theta}_2^{\,2}(m_2c_{x_2}\sin\theta_2 + m_2c_{y_2}\cos\theta_2) \\ T_2 &= \ddot{\theta}_1(m_2c_{x_2}l_1\cos\theta_2 - m_2c_{y_2}l_1\sin\theta_2 + I_2) + \ddot{\theta}_2I_2 \\ &\quad +l_1\dot{\theta}_1^{\,2}(m_2c_{x_2}\sin\theta_2 + m_2c_{y_2}\cos\theta_2) \end{aligned} \tag{2.15}$$

where θ_i, $\dot{\theta}_i$, $\ddot{\theta}_i$ are the angular position, velocity and acceleration of joint i. T_i is the torque at joint i. I_i, m_i, l_i, c_{x_i} and c_{y_i} are respectively the moment of inertia, mass, length and the x and y components of the center of mass location of link i. The values chosen for the different parameters are shown in table 2.2.

The input vector is formed of 6 variables $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2)$. The outputs to be estimated are the two joint torques. Separate function approximation networks are used, one for each of the two torques. The training and test sets are formed of one

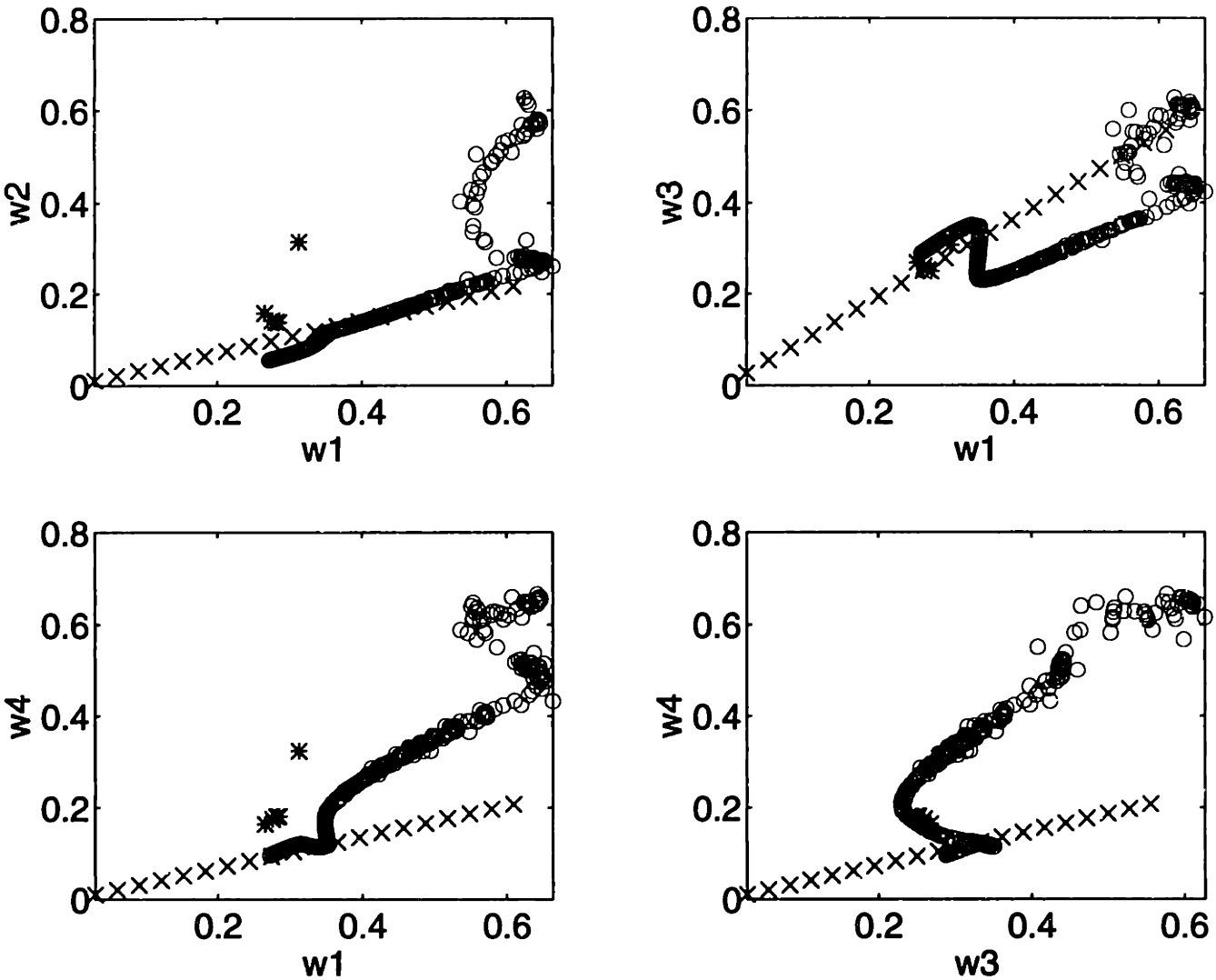

Figure 2-3: The values of the parameters of **W** used in approximating Wood's function, using HyperBFs and different optimization methods. The parameters converge towards an asymptotic line that is estimated using the estimated average derivatives of the Wood's function with respect to the different input variables.

thousand random noiseless experiences each; uniformly distributed across the space of the inputs. The different inputs are selected from the following ranges: [-4, 4] for the joint angles, [-20, 20] for the joint angular velocities and [-100, 100] for the joint angular accelerations. Three different methods of optimization for approximating the two torques T_1 and T_2, given joint angles, velocities and accelerations, have been tried and compared :

1. Exact interpolation with centers located at the training examples and using different radial basis functions. No distance metric is used in this case. The exact interpolation is repeated at different widths of the radial basis functions to determine the effect of the width parameter on the generalization of the RBFs. The coefficients of the RBFs are computed using matrix inversion. A Singular value decomposition matrix inversion algorithm is used to avoid singular matrices.

2. Using less centers than examples and using a diagonal matrix $\mathbf{W}$. The non zero elements of $\mathbf{W}$ are estimated using a nonlinear least square Levenberg-Marquardt algorithm.

3. As in 2, but using the derivative functional to obtain $\mathbf{W}$. The derivative functional is estimated using the iterative method described above. We also compared the derivative functionals obtained with the iterative method with the ones estimated directly assuming we know the true dynamic equations.

For the exact interpolation case, the input variables are scaled such that the input space is limited to the six dimensional hypercube $[-1, 1]^6$. This resulted in reducing the error on the test set.

The results of using exact interpolation for different types of RBFs are shown in figure 2-4. c^2 represents the width parameter when relevant. The normalized error in

estimating the test set is computed using the following equation:

$$E = \sqrt{\frac{\sum_{k=1}^{2}\sum_{i=1}^{n}(T_{ki} - \hat{T}_{ki})^2}{\sum_{k=1}^{2}\sum_{i=1}^{n}T_{ki}^2}} \tag{2.16}$$

where $\hat{T}_{ki}$ is the approximated torque at the k^{th} joint for test point i. The error on the training set is zero in this case, since this method performs exact interpolation. The results for LS and CS shown in this figure are obtained after the addition of a first order polynomial to the RBFs. A third order polynomial for TPS is also tried.

As shown in this figure, the normalized error is more sensitive to the width parameter (i.e. c^2) for the Gaussian RBFs than for Hardy multiquadrics and inverse multiquadrics. This is in agreement with Franke's observation (Franke, 1982). The best normalized error for any RBF that has been tested here is 0.338 for HMQ with a value of $c^2 = 4$. Also, contrary to our expectations and to results reported by others (Franke, 1982), the TPS with a third order polynomial had a normalized error of 0.5003. This error value did not change significantly when only lower order polynomials are added to the $(r^2 \log r)$ RBFs. Using Generalized Cross Validation (Bates *et al.*, 1987) to optimize the tradeoff between smoothness and fitting the data, we got similar normalized error for TPS. These results are much worse than the results obtained using other methods on the same problem and using the same data sets. (See for example Atkeson, 1989 or Zhao, 1992). Using 500 Gaussian RBFs (250 Gaussians only for the recursive method) and a diagonal matrix $\mathbf{W}$, the error on the test set using the different methods to estimate $\mathbf{W}$ are shown in table 2.3. Shown also in this table are the values of the six diagonal elements of $\mathbf{W}$. As shown in the table, the value of w_{11} which corresponds to the joint angle θ_1 is very close to zero which reflects the irrelevancy of this variable to the output and which also justifies the use of the derivative functional in the estimation of $\mathbf{W}$. The same is true for the value of w_{44} for the 2^{nd} joint which corresponds to $\dot{\theta}_2$ which indeed does not affect the computation of the torque for the second joint as it is obvious from equation 2.15. The normalized

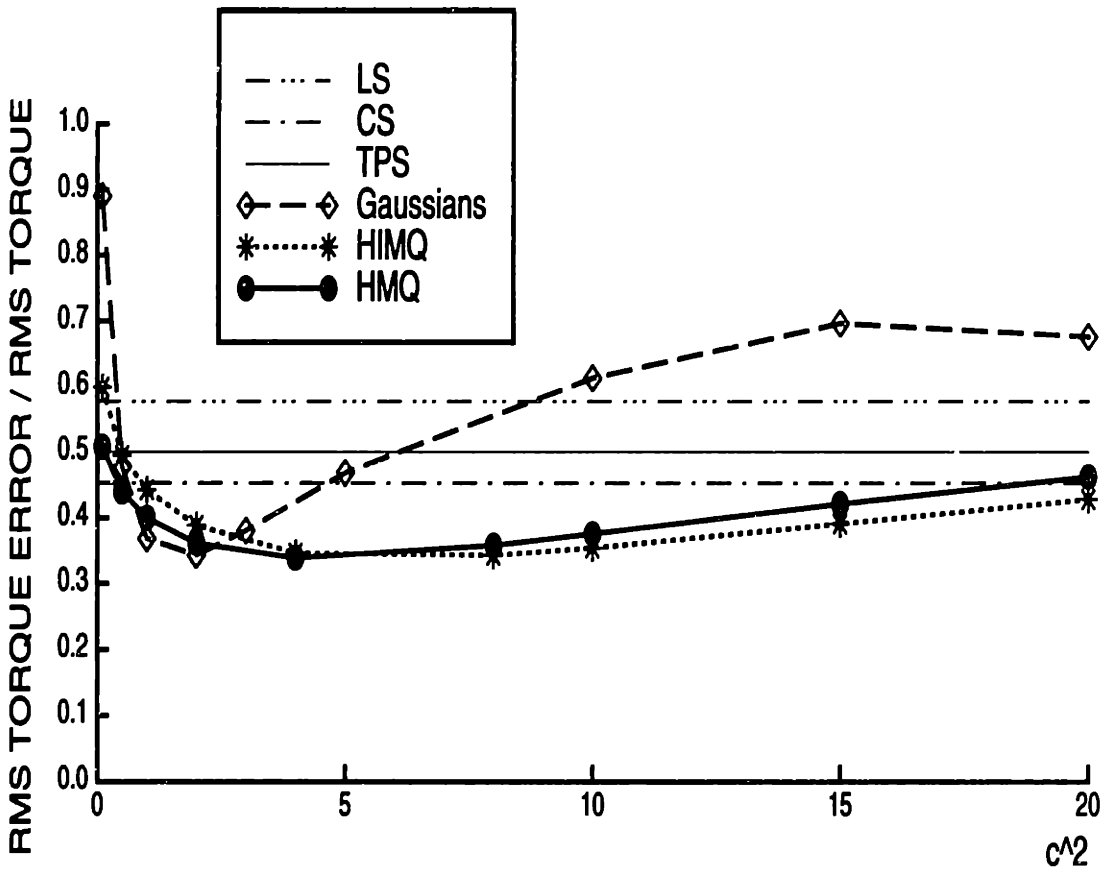

Figure 2-4: Normalized RMS error on a test set using exact interpolation with different radial basis functions as a function of the width parameters.

RMS error at each iteration for the iterative method is shown in figure 2-5, using 100 Gaussian radial basis functions only. As shown in this figure, the error approaches zero after about 4 iterations. Using 250 Gaussian radial basis functions for the iterative method instead of a 100, makes the error on the test set converges almost to zero in one iteration only. It is important to mention here that the convergence depends also on the initial condition. If the initial set of parameters, or the number of centers and their widths is not appropriate, then this recursive method will converge to a wrong solution, albeit always better than the original solution. The RMS error on the test set obtained by optimizing the metric used in the RBF approximation was smaller than any other function approximation method that have been tried on the same problem using the same data sets.

I_1, I_2	0.33333
m_1, m_2	1.0
c_{x1}, c_{x2}	0.5
c_{y1}, c_{y2}	0.0

Table 2.2: Parameter values used in the simulation of the robot equations

W	L-M ALG.		TRUE FUNC.		REC. METH.	
	Joint 1	Joint 2	Joint 1	Joint 2	Joint 1	Joint 2
$W_{11}(\theta_1)$	0.000021	5.48237e-06	0.000000	0.000000	5.82823e-08	4.23936e-09
$W_{22}(\theta_2)$	0.382014	0.443273	0.456861	0.456449	0.449958	0.449939
$W_{33}(\dot{\theta_1})$	0.004177	0.0871921	0.005531	0.010150	0.0002817	0.0010549
$W_{44}(\dot{\theta_2})$	0.004611	0.000120948	0.007490	0.000000	0.000560	0.000000
$W_{55}(\tilde{\theta_1})$	0.000433	0.00134168	0.000271	0.000110	0.0000025	0.00000065
$W_{66}(\tilde{\theta_2})$	0.000284	0.000955884	0.000059	0.000116	0.00000017	0.00000041
RMS	0.0098		0.0001		0.003079	

Table 2.3: Scaling Weights and Errors Using Different Methods. 500 centers are used in the Levenberg-Marquardt algorithm and true function method, but only 250 centers are used in the recursive method

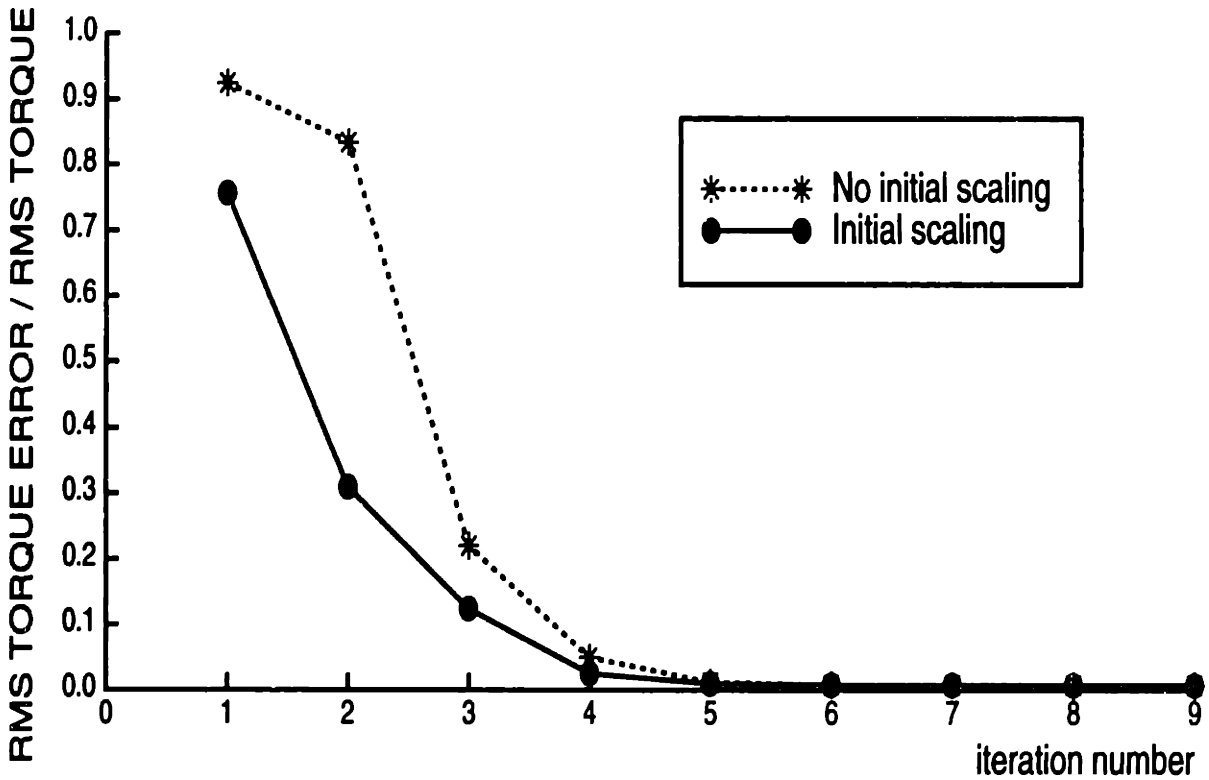

Figure 2-5: Normalized RMS error on the test set using the recursive method described in the text as a function of the iteration number

2.3 Conclusion

Knowledge representation for continuous input / output systems may be regarded as function approximation. One major issue for function approximation is the removal of irrelevant variables and the appropriate scaling of all the relevant variables. This is very important, especially in applications where the data are sparse relative to the number of dimensions. Appropriate scaling will improve the quality of approximation considerably and will reduce the amount of data required to achieve good generalization. We have investigated and tested this claim on one particular function approximation technique called radial basis functions. Although many radial basis functions could be interpreted as optimal solutions to some approximation problems, they do not perform well in practice because the assumptions associated with RBFs inherently assume that the sensitivity of the underlying function is the same in all the directions in the input space. One way to change these assumptions is by changing the norm metric used to compute the distance of the input vector from the center of the RBF by making an affine transformation of variables [Poggio and Girosi, 1989]. The optimization of the norm metric can be done using nonlinear optimization techniques (e.g. gradient descent, second order methods, random search, ...). However these techniques are usually slow to converge. A method that works well for Gaussian RBFs was described and successfully applied in different test problems. Although this heuristic approach was successful in many test problems, there remain some important questions that need to be answered. Among these questions are:

- What are the limitations of this approach? For example what are the types of functions that it can and cannot approximate? What are the convergence properties of the iterative method described above? One possible problem that may arise with the approach described above is that one global scaling of the input variables may not be adequate for some functions where the effect of some variables in one area of the space is different than the effect of the same

variables in another area of the input space. A possible solution to this problem is to use a local RBF method, where we use only the "neighbors" to the point at which we want to determine the output. The definition of neighbors may be adaptively redefined after scaling.

- How to automate the choice of the width of the Gaussians? A very small width results in poor approximation (large oscillations) which result in a bad initial estimate of the derivatives needed to estimate the metric norm which, in turn, results in bad final approximation. A large width of the Gaussian makes the system of equations almost singular and results in very large coefficients. In the examples above, a heuristic criterion is used based on the average distance between the data points to estimate a reasonable width. This is related to criteria used by other researchers [Moody and Darken, 1989 ; Platt, 1991]

- Can we find better methods that improve the choice of the centers of the RBFs and the number of basis functions to use? Current methods either use optimization techniques or heuristic methods based on the distribution of the inputs.

- Can we extend this approach to other types of radial basis functions? One way to possibly extend this approach is to look at the different terms in the equivalent regularizers of the different radial basis functions and then make these terms of the same order of magnitude by making an affine transformation of the input variables.

From the computational point of view, the iterative heuristic method described here saves a considerable amount of time relative to the nonlinear optimization techniques, however since the norm metric obtained depends on the output values, the norm metric used for the network associated with each output can be different. This requires that we treat each output as a different problem which results in an increase in computation linearly as the number of outputs increase. This is not the case for other heuristic methods to determine the norm metric (Moody and Darken, 1989)

which do not depend on the outputs and are determined exclusively using the distribution of the input variables.

Chapter 3

Optimization Using Function Approximation

3.1 Introduction

The goal in task optimization problems is to find the set of actions or action parameters that results in the best possible performance with respect to some objective. When there is only limited knowledge about the response of the environment, the search for the optimal set of actions is performed by first trying an action and observing its performance, then selecting a new action based on the collection of prior performances. In this chapter we will explore and compare the different techniques of searching for the optimal actions assuming no prior knowledge about the environment. It is assumed that the actual physical evaluation of the performance of a particular action is possible but expensive. Therefore the objective here is to find the optimal set of parameters with the minimum physical experimentation. This goal is different than classical nonlinear optimization techniques where the objective is to find algorithms that minimize computation time. In this chapter we will only consider the optimization of objective functions that can be described by static, deterministic cost functions that have a continuous parameter space. Dynamic and stochastic

system optimization will be addressed in later chapters. Dynamic systems with parameterized control variables can still be optimized using the approaches that will be described in this chapter.

3.2 Active Exploration

The strategy used for exploring the environment depends on the goal of exploration. For finding the optimal actions given a certain goal, the role of exploration may be to find the optimal answer with a certain accuracy while minimizing the amount of search. If the goal function is assumed to be smooth, exploration may be performed along the gradient for example. On the other hand, for learning a nonlinear mapping, the role of exploration may be defined as minimizing the degree of uncertainty about the environment in the particular region of interest in the input space. In this case, a more distributed exploration may be needed, with more data points located where there is large variations in the function output. A related problem is addressed by the theory of optimal experiment design for parameter identification. The goal of selecting the data in this case is to minimize the uncertainty in estimating the parameters of a model of the environment. Prior assumptions about the structure of the model and the error distribution restrict the application of these techniques to the active exploration problem for general nonlinear systems. Beyer and Smieja define and compare density based versus error based exploration for function approximation [Beyer and Smieja, 1993]. Intuitively, effective exploration should take into account all previous experiences. This has been called reflective exploration [Beyer and Smieja, 1993]. In addition to the goal of exploration and effective use of previous experiences, exploration should also depend on the data representation and all knowledge known about the environment to be explored (e.g. number of variables, sensitivity to different variables, smoothness).

3.3 Function Approximation in Optimization

One possible approach for finding the optimal parameters of a general nonlinear cost function is to approximate the cost function and constraints using function approximation techniques, then use classical nonlinear optimization techniques to find the optimal parameters which minimize the approximated cost function given the approximated constraints. Similar approaches have been previously proposed and explored by many researchers for the purpose of global function optimization [Devroye, 1978; Schagen, 1980, 1984, 1986] using different types of function approximation techniques. Powell has suggested that RBF interpolation may be a highly successful technique for function optimization due to its ability of approximating non-uniformly distributed data points[Powell, 1987]. The use of function approximation in optimization may be most useful when the objective function and its derivatives are expensive to measure or when the measurements are noisy and some kind of averaging is required. Another important advantage of this technique over other optimization techniques is that, unlike other methods, this method can potentially make use of all the previous searches for the optimum whereas other techniques forget most of the previous searches. This can potentially reduce the number of function evaluations required to achieve a specified performance.

Although the work of previous researchers has shown some advantages for using function approximation techniques for optimization over other nonlinear methods in some examples, it is not yet clear how function approximation methods for optimization will perform for general nonlinear optimization problems. One of the major problems associated with the use of function approximation is the choice of the exploration strategy. There is always a conflict between choosing an action that increases the knowledge of the learner, and an action that exploits the current knowledge of the learner about the environment in order to find the estimated best action. This problem has been termed the exploitation-exploration problem. Thrun [Thrun, 1992] presents a survey and a taxonomy of the different approaches to the exploitation-

exploration problem. He classifies the different exploration strategies into directed and undirected techniques. He proposes the method of selective attention as a basis for switching between exploration and exploitation. His method works well for problems with a finite number of states and actions. In addition to the goal of exploration, the exploration strategy may also depend on the acquired knowledge and its representation. For a model-based exploration, the role of the exploration may be to minimize the uncertainty in the objective function approximation at the beginning of the search. At later stages of the search, the emphasis of exploration may be shifted towards finding the parameter values that optimize the objective function. Schagen suggested and implemented a method of exploration based on estimating a weighting factor which strikes a balance between the conflicting goals of exploring unknown regions and optimizing the function in known regions [Schagen, 1984]. The weighting factor used depends on the number of data points evaluated so far, as well as the apparent variation of the objective function. In another paper [Schagen, 1986], Schagen also examined the use of another criteria for exploration suggested by Mockus [Mockus, 1989] based on minimizing the expected deviation from the optimum. Both these exploration strategies are based on approximating the objective function using Gaussian RBFs and interpreting the Gaussian RBFs as the correlation function between data points.

In this chapter, we will explore the use of function approximation in the optimization of an unknown multivariable objective function. We propose an algorithm for multivariable function optimization using RBF approximation. Our approach for exploring the function and representing previous experiences is similar to the approach used by Schagen. However, instead of having a weighting factor that balances between optimization and exploration, we use two different objectives, one for exploratory actions and the other for optimizing actions. The probability of switching between these objectives is based on a parameter that reflects the average error in prediction of the RBF network. Using many computer simulations, we will try to compare optimiza-

tion using function approximation with general nonlinear optimization techniques. We will explore the effect of the nature of the problem, for example the number of dimensions, degree of nonlinearity and smoothness, desired accuracy and presence of local optima on the performance of the different techniques. We will also explore and compare the effect of using different function approximation techniques and different exploration strategies on the amount of search.

3.4 Sequential Optimization Algorithm

The algorithm we propose here is closely related to Schagen's optimization algorithm [Schagen, 1984]. To balance the often conflicting goals of exploratory and optimizing actions, Schagen used a cost function formed of two terms, weighted by a factor that depends on the size of the unexplored region relative to the estimated variability of the function as measured by the estimated width of the Gaussian RBFs used to approximate the function [Schagen, 1984]. The two terms in the cost function represent the optimization and exploration phases respectively. In the first few iterations of the algorithm, when only few data points are available, the exploration term is dominant and as more data are accumulated, the optimization term becomes dominant. One problem with this approach is that the objective function that is optimized does not reflect the function we want to optimize until the optimization term becomes dominant, and even then it will not be completely accurate. Another difficulty of the approach is that the weighting factor used to balance between exploration and optimization is based on many assumptions about the function to be optimized such as assuming that it is a stochastic process with symmetric Gaussian correlation between points. These assumptions may not necessarily hold for general nonlinear functions.

In the algorithm proposed here, a new data point is selected based on optimizing one of two separate objective functions. One objective function $J_1(\mathbf{x})$ represents the approximated model of the real function to be optimized. We use Gaussian RBFs with

variable diagonal weight matrix to approximate this function as shown in equation 3.1.

$$\mathbf{x}_o = argmin \quad J_1(\mathbf{x}) = \sum_{i=1}^{N} C_i \exp(-\|\mathbf{x} - \mathbf{t}_i\|_W^2) \tag{3.1}$$

where N is the number of Gaussian RBFs used. In the simulations that follow, we choose N to be equal to $k/2$, where k is the iteration number and also the number of data points accumulated. The coefficients C_i and the diagonal W matrix are estimated using the iterative algorithm developed in the previous chapter. However, instead of using straightforward least squares, a weighted least squares algorithm is used, where the weights are inversely correlated with the true value of the function as described in equation 3.2. We have found that this cost function works better in practice, since it tends to increase the accuracy of estimation for small values of the function, which is the area of interest. We assume here that the minimum is a small positive value, which is often true for quadratic cost criteria.

$$E(\mathbf{C}) = \sum_{i=1}^{N} \frac{1.}{(y_i^2 + c^2)} * (\hat{y}_i - y_i)^2. \tag{3.2}$$

where $\mathbf{C}$ is a vector of the parameters to be estimated. The centers of the RBFs $\mathbf{t_i}$ are chosen randomly to span the range of input variables. The goal of the minimization of the second objective function $J_2(\mathbf{x})$ shown in equation 3.3 is to select new data points as far as possible from each other in the desired range of data.

$$\mathbf{x}_o = argmin\ J_2(\mathbf{x}) = \sum_{i=1}^{k} \exp(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{\sigma^2}) + \sum_{i=1}^{d} \exp(-\frac{(a_i - x_i)^2}{\sigma^2}) + \exp(-\frac{(b_i - x_i)^2}{\sigma^2}) \tag{3.3}$$

where d is the input dimension and x_i is the i^{th} component of the vector $\mathbf{x}$. The parameters a_i and b_i, $i = 1, \cdots, d$ define the region of search. Equation 3.3 is similar to the exploratory term used by Schagen [Schagen, 1984]. It provides a much better distribution of data compared to uniform random selection. This better distribution is due to the dependence of the current selection of data points on all the previous

selections, unlike a random uniform distribution which does not have a memory. The exploration based on optimizing $J_2(\mathbf{x})$ is based only on the density of the data in the different areas of the input space. Another possible strategy for exploration may take into account the performance at the different data points. This may be accomplished, for example, by multiplying the different terms in the summation of J_2 by the corresponding true value of the function at the data point. A comparison between these two different strategies for exploration and data selected randomly and uniformly in two dimensions is shown in figure 3-1. From simulation results, we decided against the use of value-based exploration since it resulted in poor identification of the objective function and a larger number of trials to achieve the same level of performance. The best exploration strategy found was the density based spacing of the data. In all the simulations that follow we used this method of exploration.

The switch between the exploratory and optimizing objective functions is based on a probability that depends on the average normalized prediction error of the approximation network. At each iteration of the algorithm, the average normalized prediction error $E_p(k)$ is estimated using equation 3.4.

$$E_p(k) = aE_p(k-1) + b\max(\frac{|y_o(k-1) - y_e(k-1)|}{|y_o(k-1)| + c}, 1) \tag{3.4}$$

where a, b, c, are constants chosen to be 0.9, 0.1 and 0.1 respectively. $y_o(k)$ and $y_e(k)$ are the observed and predicted value of the function at iteration k. $E_p(0)$ is chosen to be equal to one. For the values of a, b and c chosen the equation for $E_p(k)$ above is guaranteed to be stable. Using $E_p(0) = 1$ guarantees that the value of $E_p(k)$ will lie in the range (0, 1]. The probability of choosing to optimize the approximated cost function $J_1(\mathbf{x})$ instead of using the exploration cost $J_2(\mathbf{x})$ is determined using a monotonically decreasing function of $E_p(k)$. In the following simulation, we chose

$$P(opt, k) = 1 - E_p(k) \tag{3.5}$$

where $P(opt, k)$ is the probability of selecting to optimize the approximated function at iteration k. As the predicted error decreases, the probability of choosing the optimization objective over the exploration objective increases. In summary the proposed algorithm works as follow :

1. Initialize the algorithm. This includes the selection of variables $\mathbf{x}$, the range of interest and the initialization of $E_p(0) = 1$.

2. At iteration k, find $\mathbf{x}(k)$ that optimizes either $J_1(\mathbf{x})$ or $J_2(\mathbf{x})$ based on the probability $P(opt, k)$ described by equation 3.5.

3. Perform an experiment using the value of $\mathbf{x}(k)$ found, and observe the output $y_o(k)$. Also predict the value of the output $y_e(k)$ using $J_1(\mathbf{x})$.

4. Add the new data point $\{\mathbf{x}(k), y_o(k)\}$ to the training set and update the function approximation.

5. Find $E_p(k+1)$ and $P(opt, k+1)$ using equations 3.4 and 3.5 respectively.

6. Go to 2.

We did not use an explicit stopping criterion for the algorithm described above. There are many possible stopping criteria that can be chosen depending on the nature of the problem. In the case when the value at the optimal point is known, the algorithm can be stopped when the optimal value obtained approaches the correct value. In the case when the optimal value is not known a priori, we can choose to stop the algorithm when the average predicted error falls below a certain level and then choose the optimal value found so far. From our experience with the simulations below, we found that the algorithm finds the optimal value with a good accuracy while the average predicted error is still relatively high (e.g. above 0.5 in most cases). This is due to the relatively high density of data points near the optimal value compared to other areas of the input space, which results in an increased accuracy near the optimal point.

3.5 Test Results

We tested the sequential model-based optimization algorithm described above on the optimization of many different objective functions. The objective functions chosen were selected because they represent classical benchmarks in the nonlinear and global optimization literature. Branin, and Hartman functions are taken from [de Biase and Frontini, 1978]. The Rosenbrock's, helical and Wood's functions can be found in most nonlinear optimization textbooks (for example [Scales, 1985]). In all the following examples we compared the performance of the model-based algorithm with the Broyden-Fletcher-Golfarb-Shanno (BFGS) gradient based minimization algorithm. The BFGS algorithm is chosen due to its superior performance in many of the test problems used here [Scales, 1985]. The BFGS algorithm used here requires a much fewer number of function evaluations, including function evaluations required to compute the derivatives, than methods of optimization that do not require derivatives. For example Powell's method requires 125 function evaluations to reach an error of less than 0.001 for optimizing the Rosenbrock Banana function [Fletcher, 1965]. Gradient methods and the model-based method proposed here require only about 30 function evaluations to reach the same accuracy (see below). For all the examples shown below, the number of function evaluations reported for the BFGS algorithm includes the number of function evaluations needed to compute the derivatives and represents the average of many trials. The number of function evaluations reported for the sequential model-based algorithm is for one trial only. We found that this is adequate since this algorithm is relatively insensitive to changes in the starting point.

3.5.1 Optimization of Branin's RCOS Function

Branin's RCOS function is a 2-dimensional function described by 3.6

$$J(x_1, x_2) = a(x_2 - b{x_1}^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e \tag{3.6}$$

$$a = \pi \; ; \; b = 5.1/(4.0\pi^2) \; ; \; c = 5.0/\pi \; ; \; d = 6.0 \; ; \; e = 10.0\pi \; ; \; f = 1.0/(8.0\pi)$$

There are three global minima with minimum value equal to 1.25 in the region $[-5 \leq x_1 \leq 10 \; ; \; 0 \leq x_2 \leq 15]$. The coordinates of the minima are $(-\pi, 12.275)$, $(\pi, 2.275)$ and $(3\pi, 2.475)$. The contour of the function together with the output of 35 iterations of the sequential optimization described above is shown in figure 3-2. As shown in figure 3-2, the algorithm was able to discover 2 of the three global minima in less than 35 iterations. The algorithm was able to discover the first minimum with accuracy less than 0.01 in 29 iterations. In the first 35 iterations of the algorithm, 22 iterations are exploratory and the other 13 are optimizing (represented in figure 3-2 by the symbols 'x' and 'o' respectively). The probability $P(opt, k)$ for performing optimization rather than exploration is shown in figure 3-3 as a function of iteration number. The average number of function evaluations to reach only one of the global minima using a non-linear second order technique such as the Broyden-Fletcher-Golfarb-Shanno (BFGS) method for 10 trials with random starting points is equal to 28.2, which is comparable to the number of function evaluations for the proposed sequential model based optimization. The number of function evaluations includes 2 function evaluations to estimate the gradient at each iteration.

3.5.2 Optimization of Rosenbrock's Banana Function

The Rosenbrock's Banana function is a 2-dimensional function, characterized by a banana shaped curvature described by equation 3.7

$$J(x_1, x_2) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2 \tag{3.7}$$

This function has one global minimum at the point $(1, 1)$ which has a value of 0. The region of interest is defined to be $[-2 \leq x_1 \leq 2 \; ; \; -1 \leq x_2 \leq 3]$. The first 30 iterations of using the sequential algorithm is shown in figure 3-4. The first 30

iterations consisted of 19 exploratory actions reprsented by the symbol "x" and 11 optimizing actions represented by the symbol "o". The desired accuracy was set to $\epsilon = 0.001$. The algorithm reached a value of 0.0092 at the point $(1.035, 1.073)$ after 26 iterations. The performance of the sequential model based algorithm was much better in this case than the performance obtained using the BFGS method which reached the minimum in an average of 86.4 function evaluations (including derivative calculations), based on 100 different trials with random starting points.

3.5.3 Optimization of Wood's Function

Wood's function is a 4-dimensional function described by equation 3.8

$$\begin{aligned} J(\mathbf{x}) &= 100({x_1}^2 - x_2)^2 + (1 - x_1)^2 + 90({x_3}^2 - x_4)^2 + (1 - x_3)^2 \\ &\quad +10.1(x_2 - 1)^2 + 10.1(x_4 - 1)^2 + 19.8(x_2 - 1)(x_4 - 1) \end{aligned} \tag{3.8}$$

There is one global minimum at $\mathbf{x} = (1, 1, 1, 1)$. We limited the search to the hypercube $[-2, 2]^4$. Based on an average of 10 trials, the BFGS nonlinear optimization method reached a value lower than 0.001 after 197.7 function evaluations. Using the sequential model based optimization we reached a value of 0.0036 after 160 observations with $\mathbf{x} = (0.9738, 0.9414, 1.0208, 1.0505)$. No further improvement could be made with more experimentation, up to the iteration limit of 200 iterations. Out of 200 iterations, 89 were optimizing steps and the rest was exploratory.

3.5.4 Optimization of the Helical Function

This is a function of 3 variables which has a helical valley. It is described by equation 3.9.

$$J(\mathbf{x}) = 100((x_3 - \theta)^2 + (r - 1)^2) + x_3^2 \tag{3.9}$$

where

$$\begin{aligned} 2\pi\theta &= \tan^{-1}(x_2/x_1) \\ r &= ({x_1}^2 + {x_2}^2)^{1/2} \end{aligned}$$

The minimum for this function occurs at (1, 0, 0). The BFGS algorithm requires an average of 343 function evaluations (including gradient evaluations) to reach a value of $\|J(\mathbf{x}) - J(\mathbf{x}^*)\| \leq 0.001$ based on the average of 10 trials with random starting points in the region $[-2, 2]^3$ and an average of 251 function evaluations to reach a value below 0.005. Using the sequential RBF based optimization algorithm, the minimum value reached was 0.0044 after 108 iterations with input values [0.9377, 0.3424, 0.0520]. However, the error level did not improve with further iteration of the algorithm up to the iteration limit of 200. The number of optimization steps in 200 iterations is 91.

3.5.5 Optimization of Hartman's Family of Functions

Hartman's family of functions consists of shifted and inverted multidimensional Gaussian functions of different amplitudes. The number of local minima is controlled by the number of Gaussain functions and the location of the minima is controlled by the centers of the Gaussians. The shape and amplitude of attraction regions is controlled by the coefficients α_{ij} in equation 3.10.

$$J(\mathbf{x}) = -\sum_{i=1}^{m} c_i \exp(-\sum_{j=1}^{N} \alpha_{ij}(x_j - {p_j}^i)^2) \tag{3.10}$$

We tested the optimization algorithm on a seven dimensional Hartman function containing five local minima. The parameters of the function are chosen randomly and are shown below : $c_1, c_2, c_3, c_4, c_5 = 0.6641, 0.3234, 0.8692, 0.4424, 0.9180$

$a_1 = 0.5533, 1.9035, 0.2345, 0.8320, 1.6728, 1.9251, 0.6659$

$a_2 = 1.0524, 1.0719, 1.6362, 1.8598, 1.3258, 0.4865, 1.8829$

$a_3 = 1.1934, 0.2521, 0.4939, 1.7735, 0.3924, 0.3259, 1.6720$

$a_4 = 1.2048, 1.2465, 1.6774, 1.3433, 0.7831, 1.7426, 1.3347$

$a_5 = 0.2312, 0.0, 1.3521, 1.2575, 0.9091, 1.1487, 0.0645$

$m_1 = -0.5944, 1.4203, -.5167, 0.6457, 0.3057, -1.7602, 0.9779$

$m_2 = -0.8820, -0.2295, -1.2943, -1.6328, 1.5890, 1.9697, 0.3766$

$m_3 = -1.8680, 0.7637, -0.0512, 0.2774, 1.5829, -0.2620, 0.7544$

$m_4 = -1.1874, -0.0731, -0.9102, 1.7885, -1.0228, 1.8130, -0.0922$

$m_5 = -1.4916, 1.8457, 1.4286, -1.1304, -1.9720, 1.1299, -1.7984$

$minima = -0.6827, -0.3235, -0.8693, -0.4424, -0.9180$

Using the BFGS algorithm and based on 10 trials, the global minimum was reached only 3 times at the value of -0.918. Two times the algorithm converged to a local minimum (-0.6827). In the other 5 times, the performance did not improve. This reflects the large volume of the parameter space where the cost function is relatively flat. The region of attraction of the global minimum is relatively small compared to the volume of the input space of interest. The average number of function evaluations, including derivative calculations, based on the 3 successful trials is 167.6. Using 250 iterations of the sequential model based optimization the minimum value reached was -0.712 after 232 iterations. The sequential method was unable to locate the global minimum. The prediction error as computed using equation 3.4 hovered around 20% after a fast drop in the first 50 iterations.

3.6 Summary and Conclusion

In this chapter we explored through many simulations the use of function approximation techniques for the optimization of nonlinear functions. One of the major problems for using function approximation in optimization is the choice of the sequence of input variables to explore. To solve the conflicting goals of optimization and exploration, we represented the problem as a two-objective optimization problem,

one for exploring the area of interest for better identification of the function, and the other objective function represents the optimization process for finding the optimal input variables. We proposed a probability of performing an optimization step which is a monotonically decreasing function of the prediction error. We used Gaussian RBFs with a variable scaling of the inputs, to represent the approximated objective function. The scaling of the inputs was performed using the iterative algorithm proposed in the previous chapter. The coefficients of the RBFs were estimated using singular value decomposition to obtain the pseudoinverse of the matrix of RBFs.

Table 3.1 shows a summary of the number of function evaluations required to find the optimal value for the different functions tested, together with the results obtained using the Broyden-Fletcher-Golfarb-Shanno (BFGS) gradient based nonlinear minimization algorithm. It is obvious from this table, that the algorithm performed well in many of the test problems. However, the performance of the model-based optimization deteriorates, relative to the second order gradient based algorithms, as the number of dimensions increases especially in the case of highly varying functions such as Wood's function. This is not surprising since the number of samples required to achieve a certain accuracy in approximation is determined by the degree of variation of the function and the number of dimensions. The algorithm performs best for functions of low dimensions or slowly varying functions which may contain many local minima. In such a case, the model-based optimization may recover most of the local minima in a much lower number of function evaluations. Another case where model-based function optimization might be more efficient, is when the physical observations of the function to be optimized are noisy. Gradient based optimization techniques may not work in such cases. The performance of the algorithm depends on the approximation of the objective function. We found that a weighted least squares, where low value data are weighted more than observations of higher values, works slightly better than plain least squares. Since it approximates the data better near the minima. Other possible improvement to the objective function approxima-

	RBF		BFGS	
	# of evaluations	accuracy	# of evaluations	accuracy
Branin's	29	< 0.01	28.2	< 0.01
Rosenbrock	26	0.001	86.4	< 0.001
Wood	160	0.0036	197.7	< 0.001
Helical	108	0.0044	251	< 0.005
Hartman	232	-0.712	167.6	-0.918

Table 3.1: Comparison between the RBF model-based optimization with BFGS method

tion, may result in an improvement in the optimization algorithm. Possible ways to improve approximation may include limiting the range of input space adaptively based on previous performance and using local function estimators.

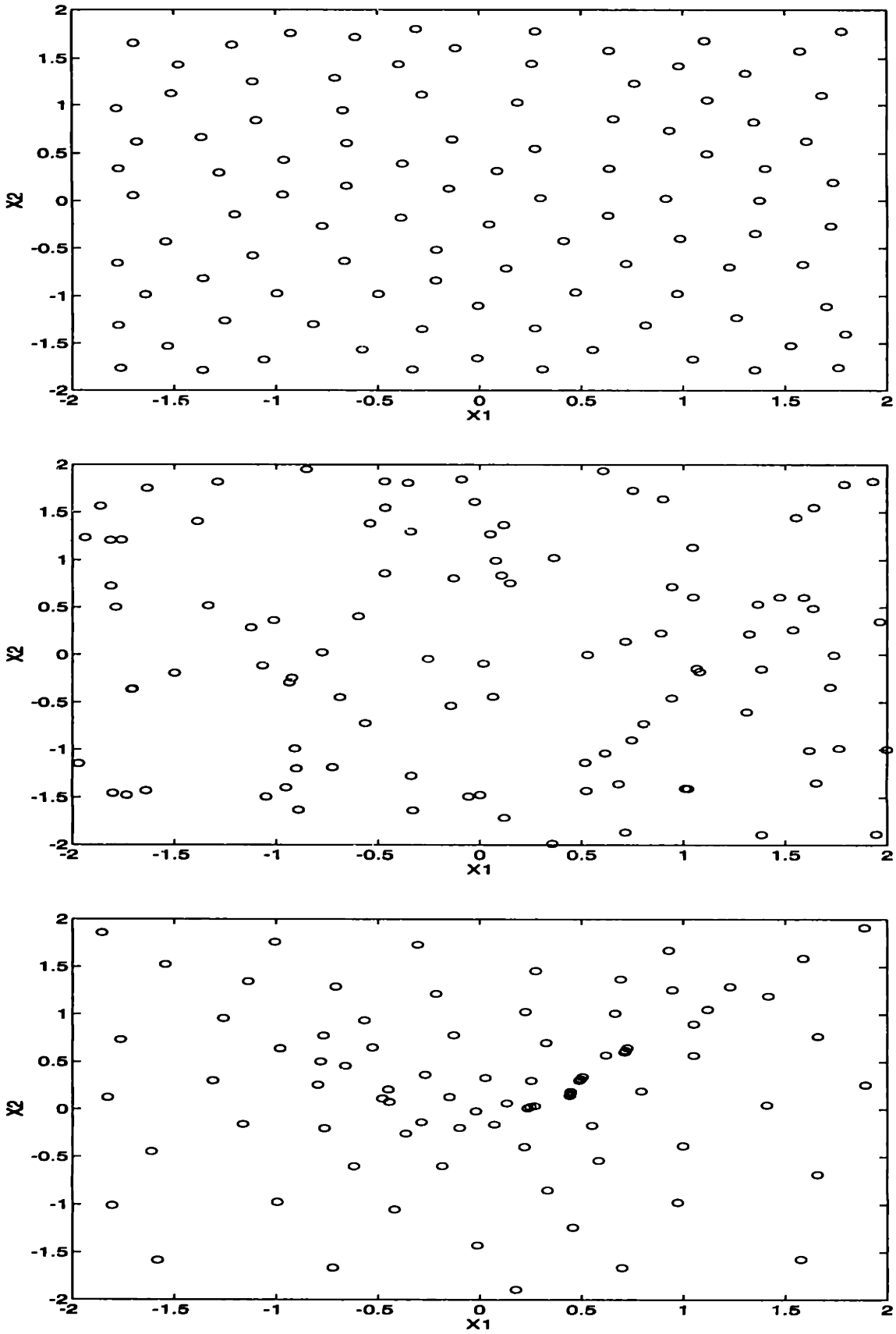

Figure 3-1: Comparison between density based spacing (upper graph), randomly spaced (center) and value-based spacing (lower graph) of the data for the Rosenbrock's banana function.

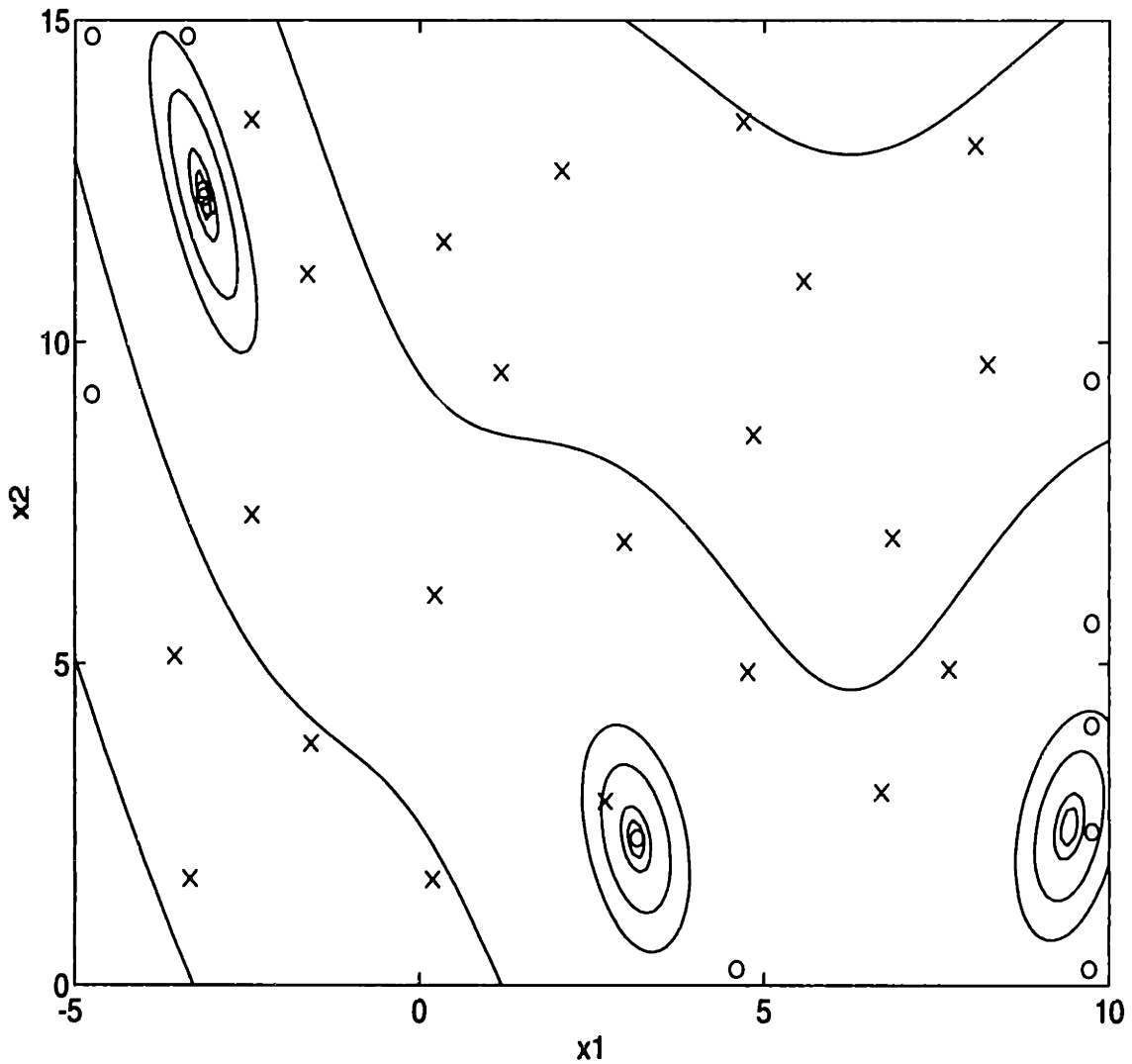

Figure 3-2: Contour of Branin's Function and result of 35 iterations of the sequential optimization algorithm. The symbol 'x' represents an exploration step and the symbol 'o' represents an optimization step.

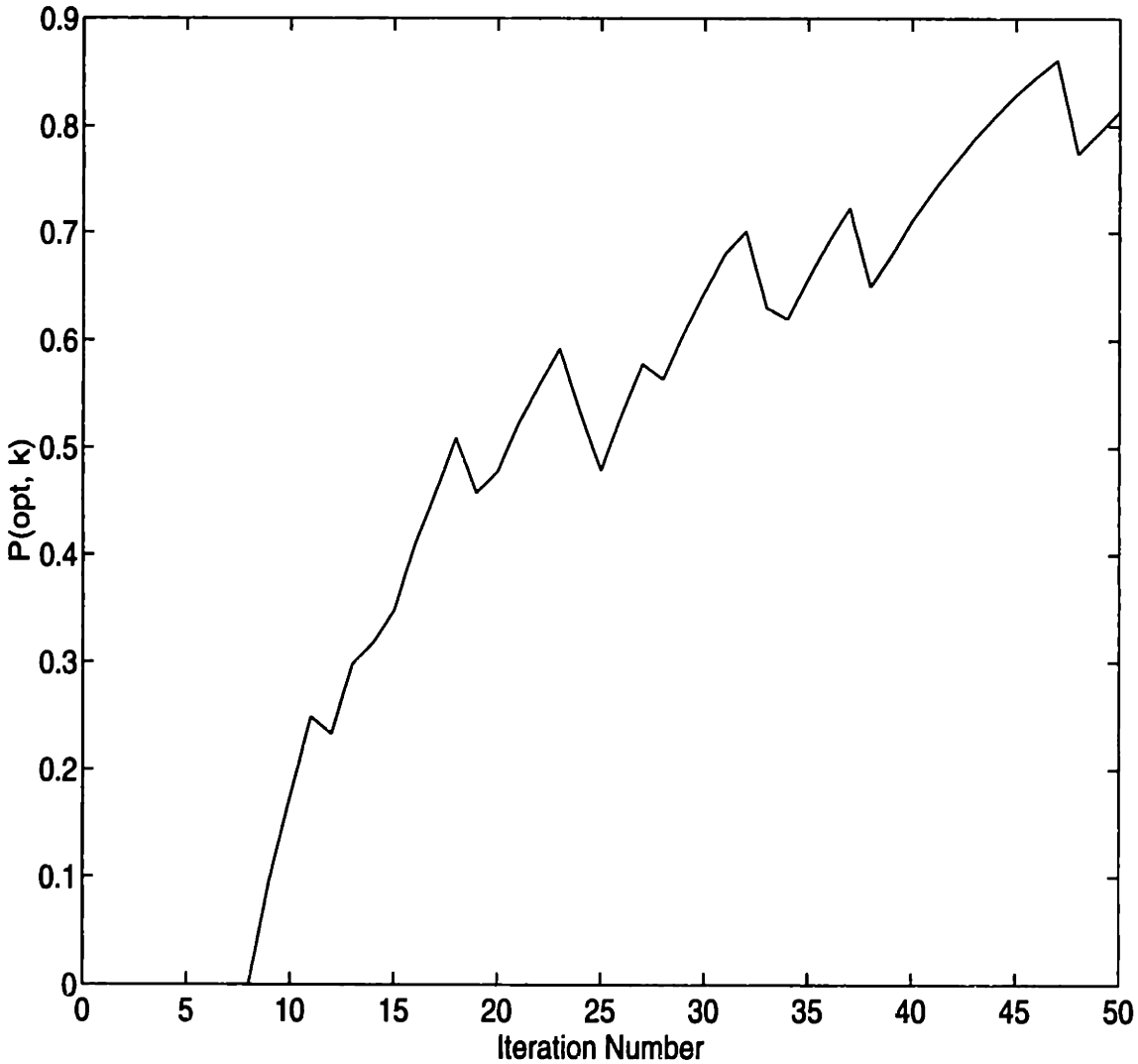

Figure 3-3: Probability $P(opt, k)$ as a function of iteration number k for 50 iterations

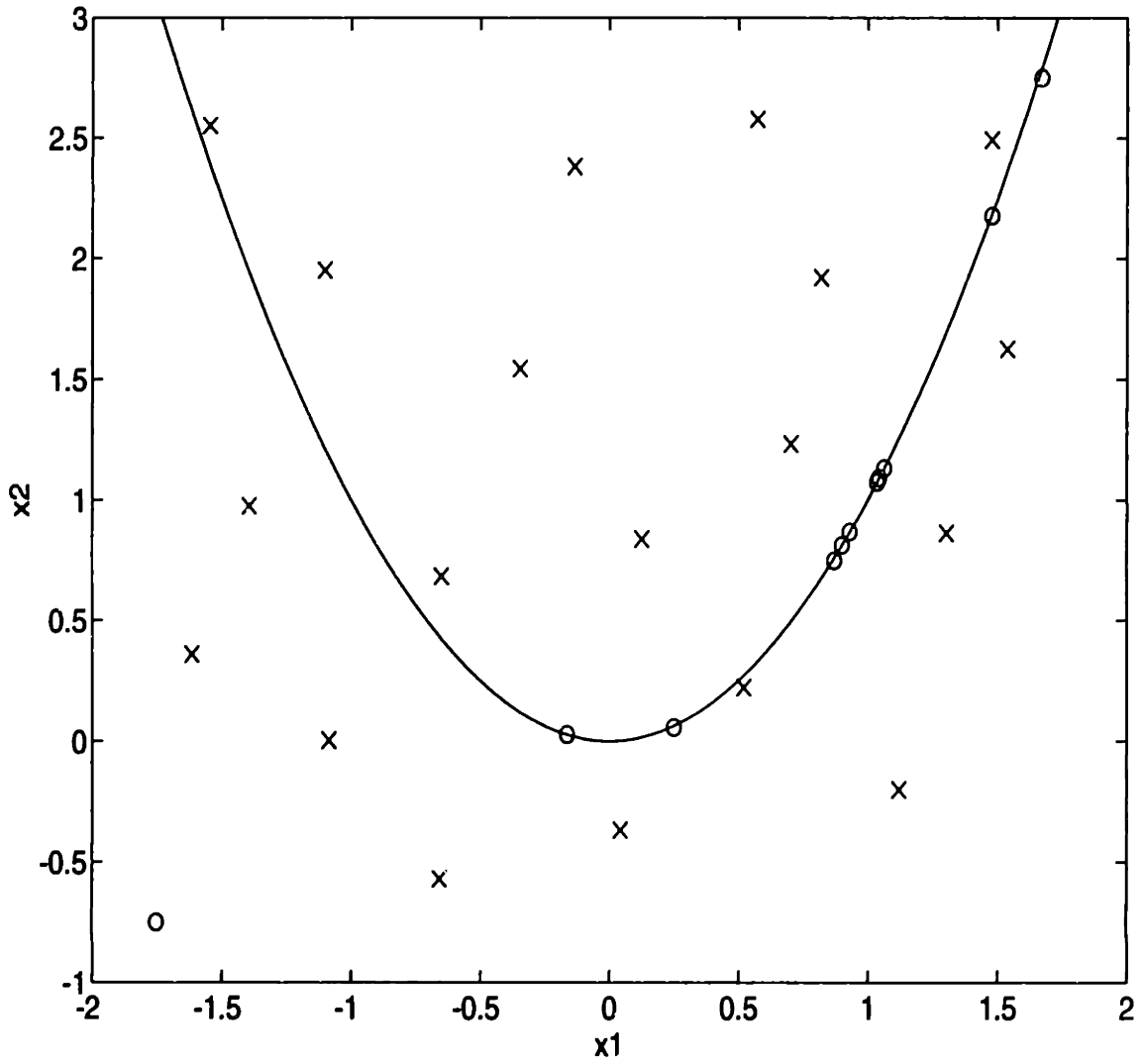

Figure 3-4: Result of the first 30 iterations of the Gaussian RBF based optimization of the Rosenbrock's banana function. The solid curve represents the minimization of the higher cost term of the banana function. The symbol ' x ' and ' o ' represent an exploration and optimization step respectively.

Chapter 4

Learning Trajectory Optimization

4.1 Introduction

In the previous chapter we discussed different methods for searching for the best action given a certain objective. However most motor activities involve a sequence of actions. The learning system has to plan the sequence of actions that optimizes a given goal. Very limited knowledge is assumed initially about the dynamics of the environment to be controlled. This limited knowledge varies with the method used to find the optimal control.

As an example of learning trajectory optimization, consider a violin player. The patterns of motion of his/her arms are optimized in such a way as to obtain the best music quality. This process of optimization and learning may take many years of practice. Many other examples of learning optimal motion by practice abound in the different fields of sports and performing arts to optimize many different objective functions such as speed, force, accuracy, gracefulness and smoothness of motion.

What makes the learning and optimization of trajectories a difficult problem is the dynamic nature of the system. An action taken at one instant in time will affect the behavior of the system at future times. Therefore in order to determine the best action to take now, it is necessary to assess its contribution to the future behavior of the

system. This problem has been termed temporal credit assignment. Many different approaches for learning and optimizing a sequence of actions have been proposed in the literature. These approaches can be generally divided into direct and indirect methods. Direct methods involve finding the control actions or control laws directly without explicitly forming a model of the dynamics of the system that we want to control. Methods such as reinforcement learning [Sutton, 1992] and genetic algorithms [Goldberg, 1989] are capable of direct optimization of the control action or control law and may be classified under this category, although these methods can also make use of a model of the system dynamics. Indirect methods, on the other hand, involve an explicit modeling of the dynamics of the system to be controlled and then use this model to find the optimal trajectory (open loop control) or the optimal control law as a function of the states of the environment (closed loop control), using dynamic optimization techniques such as calculus of variations and dynamic programming. In this chapter we will focus only on indirect methods. We will only explore the variational indirect methods based on differential dynamic programming and calculus of variations. First I will give a brief introduction to the learning optimal control problem that I will attempt to solve and then describe in more detail the solution approach that will be followed. Some simulation examples of trajectory optimization of robot arm motions will then be presented to illustrate the applicability of the approach. The problem of choosing the objective functions to be optimized will not be discussed here. These objective functions are sometimes given explicitly by a teacher (e.g. jump as high as possible, or run as fast as possible), although more often, the criteria governing the choice of the objective function are not as clear and may depend on many factors and personal preferences (e.g walk or move comfortably or in an aesthetically pleasant manner). It is important to mention that the objective functions may be a function of either or both the control and output variables. Jordan [Jordan, 1989] views some objective functions in the context of motor learning as "active constraints on the process of learning" and gives some examples of these

constraints that can be applied in different coordinate systems. Objective functions used in most of the engineering applications are quadratic functionals of the states and controls. These are selected primarily for the ease of computation.

Unlike dynamic programming methods, which generate an optimal control law (closed loop control), trajectory optimization techniques based on variational principles generate only an open loop optimal trajectory which is generally only optimal in the neighborhood of some initial state. Nevertheless, finding a good open loop control trajectory is very important in fast motions especially when there exists a delay in the feedback loop. In addition, a good open loop optimal trajectory may help simplify the feedback loops and reduce the feedback gains required to achieve a desired performance level. More importantly, to obtain a closed loop optimal solution for general nonlinear systems using dynamic programming is still a challenge for real world applications involving more than few states. This is due to the curse of dimensionality, which is the exponential increase in storage and computation requirements as the number of dimensions increases [Bellman, 1962]. From a biological point of view, it has been argued that learned fast movements are probably performed in an open loop fashion. This argument is based on the long delay associated with feedback loops (usually of the order of 100 msec.). Moreover, feedback control by itself can not explain the ability of deafferented monkeys to make fairly accurate movements [Polit and Bizzi, 1979]. However, open loop control alone is not desirable due to the presence of disturbances. If the effect of these disturbances is not reduced by the use of feedback, they may get integrated over time and we may end up with a different trajectory than the optimal one. This is a serious problem especially in the cases where the model used to obtain the optimal trajectory is not very accurate and when the system near the optimal trajectory is unstable. We will discuss in this chapter different possible techniques for extending the open loop learning optimal control methods developed here to closed loop optimal and suboptimal control and give different simulation examples of robot arm motion optimization. Also, if it is

possible to obtain an almost real-time open loop optimal control, it may be possible to compute this open loop control more often based on the current state of the system, generating a feedback optimal control. Methods for parallelization of the computation of the open loop trajectory, such as a Hopfield network implementation, may prove to be useful to implement the optimal control. We will discuss at the end of the chapter, how to map an open loop optimal control problem into a stable Hopfield network. The optimizing network connections will depend on the goal of the movement and the dynamic model of the environment. Starting from any initial conditions, the network will converge to the optimal trajectory in a short time that also depends on the strength of the connections.

4.2 Learning Optimal Trajectories

The problem we would like to address in this chapter can be stated as follows : Given an unknown dynamical system with known initial conditions, we would like to find the optimal control trajectory $\mathbf{u}(t)$ that minimizes a cost function by practicing. The unknown dynamical system can be represented by a set of ordinary nonlinear differential equations of the form 4.1. The cost function is usually a functional of the states and controls histories as shown in equation 4.2. In these equations we assume that the system to be controlled is time invariant.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{4.1}$$

$$J = \Phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t))dt \tag{4.2}$$

where $\mathbf{x}(t) \in \mathcal{R}^n$ describes the state of the system as a function of time, and $\mathbf{u}(t) \in \mathcal{R}^m$ represents the set of controls as a function of time. The different components of the state and control vectors needed to uniquely describe the system evolution as a function of time are assumed to be known and to be fully observed. This assumed

prior knowledge is needed to model the dynamical system but may not be necessary in the case of direct optimization techniques. $L(\mathbf{x}(t), \mathbf{u}(t))$ is the cost accrued at each instant of time and $\Phi(\mathbf{x}(t_f), t_f)$ is the cost of being in a certain final state at a certain final time.

We search for the optimal trajectory by repeating the motion several times and at each time refining and adjusting the control trajectory based on the accumulated experiences from the previous trials which are used to build a forward model of the dynamics of the system. The approach we are taking can be divided into two separate phases as shown in figure 4-1. The first phase is a learning phase and the second an optimization phase. These two phases are iterated until the cost functional stops improving. During the n^{th} iteration, a movement is generated using the control action obtained from the $(n-1)^{th}$ iteration with some noise added to it. The addition of the noise is important to avoid the convergence of the learning algorithm to a local minimum and to insure the continuous addition of information at each iteration. This is also very important for the learning phase of the algorithm, especially when the measurements are noisy, to avoid overfitting of the dynamic function in one small area of the input space and very poor fitting in other areas. This will result in instability. The level of the noise is reduced gradually as the number of iterations grows so as to allow the algorithm to eventually converge to the optimal solution. The dynamics of the system are represented using function approximation techniques to represent the rate of change of the state vector as a function of the states and controls (i.e. $\mathbf{f}(\mathbf{x}, \mathbf{u})$ in equation 4.1). We used the HyperBF technique with Gaussian basis functions to represent the different components of $\mathbf{f}(\mathbf{x}, \mathbf{u})$ using one separate network for each component. The forward model is used to predict the effect of the current control trajectory on the final cost, and to estimate the rate of change of the cost with respect to this trajectory. This information is then used to update the control sequence. We use classical optimal control theory based on variational calculus to estimate the gradient of the cost functional with respect to the control and

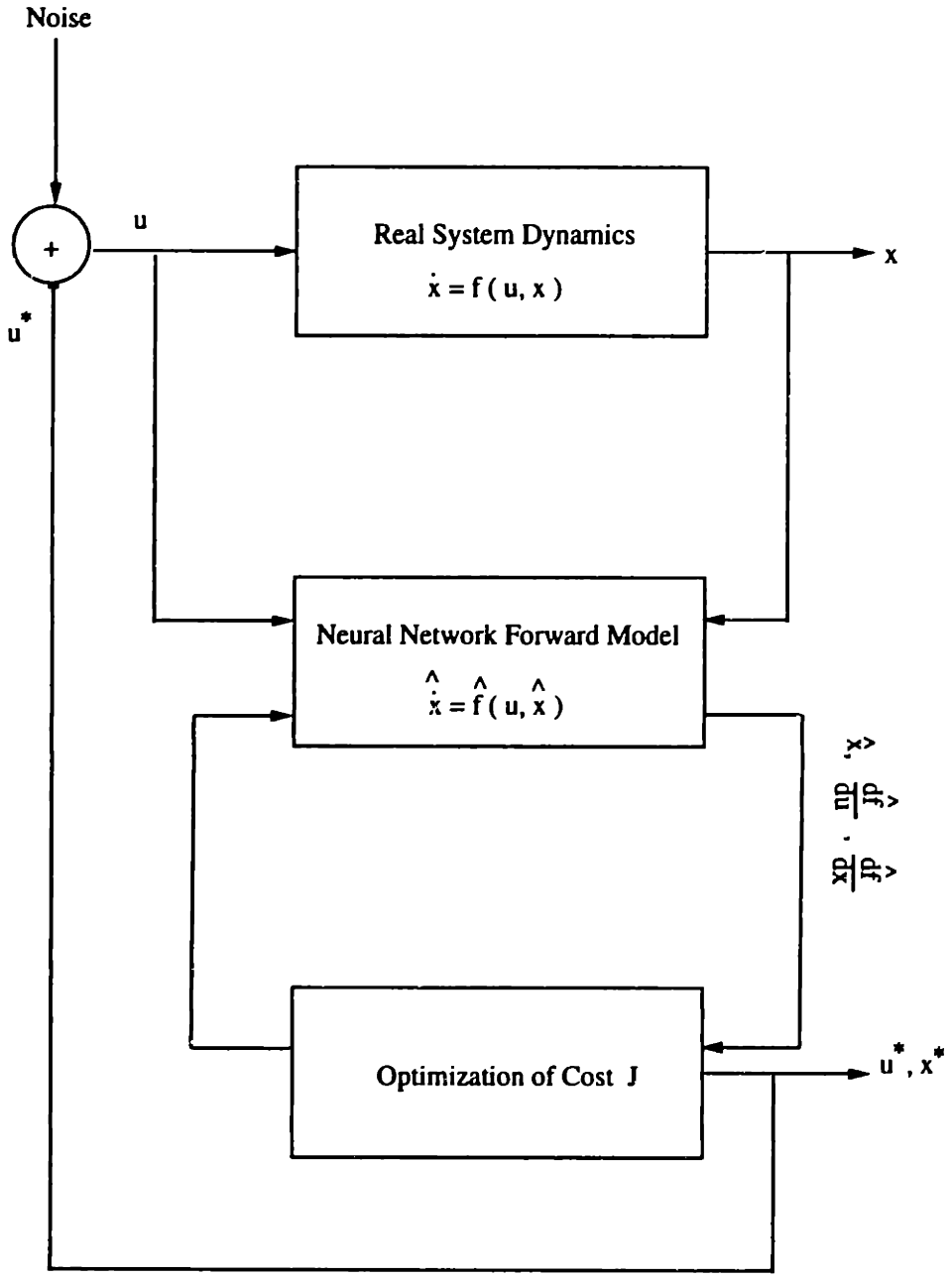

Figure 4-1: Schematic representation of the learning optimal control algorithm

to find the open loop optimal control trajectory. This is done using the dynamical system constraints 4.1, which are approximated by the Gaussian HyperBF networks, in addition to any other extra constraints on the control or the states of the system. The estimation of the Gaussian HyperBF distance metric parameters is performed using the iterative algorithm described in chapter 2.

4.3 The Exploration Problem

In order to build a forward model of the system to be optimized, it is necessary to be able to explore the effect of different inputs in the area of the state space of interest. As mentioned in the previous chapter the goal of exploration conflicts with the goal of finding the optimal control. For example, if the purpose of the optimal controller is to hold the state at a fixed value, the controller will not be able to do this in a stable and robust manner without first knowing what is the response of the system for different input values. In the previous chapter, we discussed the active exploration problem as it applies to the optimization of cost functions that do not depend on dynamics. In such a case, it is possible to freely choose to explore any region of interest in the input space. However, the behavior of the optimization algorithm depended on the exploration strategy used. We proposed an algorithm for exploration, which memorizes the previous areas explored and tend to explore more in areas of low density of data points. We also tested a modification of this algorithm which tended to explore areas of the state space where the outcome has been more favorable, in addition to the density of data points. We found that for the global RBF function approximator used, exploration based on density only worked better than exploration based on the outcome of the function in addition to the density of points. For dynamic systems, however, there are many different problems that may restrict the exploration of the input space. Issues such as stability, and reachability of the different states from a given initial condition and initial time play a major role

in the choice of exploration strategy. On the other hand, in order to better identify the model of the system it is necessary to always excite the system to discover all the possible modes. This issue has been recognized early by adaptive control and systems identification theorists and has been called persistence of excitation. Another constraint on the exploration is that we are interested in finding the optimal control, therefore we have to strike a balance between identification and optimization. This is the exploitation-exploration problem discussed in the previous chapter. It has also been termed the dual control problem in the control literature [Feldman, 1966]. As also discussed in the previous chapter, the system identification part should not be treated separately and should depend on the purpose of the model that we would like to develop, and the representation of the model. For example, if the need is adaptive regulatory control, stability issues are very important, and indeed modern theories of adaptive control use Lyapunov stability results in order to design a stable control system [Narendra and Annaswamy, 1989; Sastry and Bodson, 1989]. For iterative learning of the optimal control, as is the objective we would like to achieve in this chapter, stability is not critical since we test the system only for a short period of time. The persistence of excitation requirement depends on the representation of the model. For models represented with a linear combination of nonlinear functions of the states, such as the RBF forward modeling approach we use here, in order to identify the linear coefficients of the nonlinear functions, it is necessary to use inputs that excite those functions. This has been termed functional persistence of excitation by Sanner and Slotine [Sanner and Slotine, 1992]. Although these constraints on the control input have been well known and studied by adaptive control theorists, there is no general methodology for the choice of inputs. Control inputs are chosen mostly in a heuristic fashion in order to satisfy the persistence of excitation requirement. Guez, Rusnak and Bar-Kana propose the use of a two-objective function optimization, one representing the control objective and the other representing the estimation objective given some a priori information about the class of tasks that the system may be

required to do in the future [Guez, Rusnak and Bar-Kana, 1992]. In the simulations we report here, we use a heuristic approach to the exploration problem. Exploration is performed along the gradient of the estimated cost function, in addition to a small random component. We found that such a heuristic scheme works fine for most problems.

4.4 Solution of the Optimal Control Problem

In this section we will describe briefly the theory of dynamic optimization based on the calculus of variations and Pontryagin maximum principle, as applied to dynamical systems described by HyperBF functions. We will only focus here on continuous, time invariant systems. Many of the mathematical relations will be stated here without proof. Proofs of these relations and a detailed analysis of the theory can be found in [Bryson and Ho, 1975]. Dyer and McReynolds offer a parallel analysis based on the principle of optimality and dynamic programming approach [Dyer and McReynolds, 1970].

Given the cost functional J described by equation 4.2 above, and the constraints that govern the evolution of the dynamical system approximated by the HyperBF functions, we would like to find the optimal control $\mathbf{u}(t)$ as a function of time. Based on Lagrange theory, this constrained optimization of the cost functional J is equivalent to the unconstrained optimization of the augmented cost functional J' given by adjoining the constraints to the cost equation J through the use of Lagrange multipliers.

$$J' = \Phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t)) + \lambda(t)^T(\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) - \dot{\mathbf{x}})\, dt \qquad (4.3)$$

The vector $\lambda(t) \in \mathcal{R}^n$ represents the Lagrange multipliers. Since the constraints represented by the dynamical system equations should be satisfied at all times from t_0 to t_f, the Lagrange multipliers also have to be functions of time. Any additional static

constraints, such as constraints on the final state of the system could be adjoined to the cost functional in the same way, but using a constant Lagrange multiplier instead. However, we will only focus on the special case described by equation 4.3 to simplify the analysis. The Lagrange multipliers $\lambda(t)$ have many different interpretations. It can be shown that, on the optimal trajectory, $\lambda(t)$ represent the partial derivatives of the cost functional $J^*(t)$ with respect to the different components of the state vector $\mathbf{x}(t)$ while holding the control $\mathbf{u}(t)$ constant; where $J^*(t)$ is the optimal cost to go from time t to t_f. $\lambda(t)$ is also referred to in the literature as influence functions [Bryson and Ho, 1975] or costates. If we define the Hamiltonian function as:

$$H(\mathbf{x}(t), \mathbf{u}(t)) = L(\mathbf{x}(t), \mathbf{u}(t)) + \lambda(t)^T \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.4}$$

Then the augmented cost functional J' could be rewritten as in equation 4.5

$$J' = \Phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} (H(\mathbf{x}(t), \mathbf{u}(t)) - \lambda(t)^T \dot{\mathbf{x}})\, dt \tag{4.5}$$

The Hamiltonian can also be interpreted on the optimal trajectory as the negative of the partial derivative of the optimal cost to go with respect to time (equation 4.6). This relation could easily be derived from calculus of variations principles.

$$\frac{\partial J^*}{\partial t} = -H^*(x, \frac{\partial J^*}{\partial x}) \tag{4.6}$$

The superscript $*$ defines the different variables at the optimal trajectory. Equation 4.6 is also called the Hamilton-Bellman-Jacobi equation [Bryson and Ho, 1975] and it provides the link between the calculus of variations approach and dynamic programming. The optimal value of the cost functional J' occurs when the increment dJ' is equal to zero. This is a necessary condition for optimality. If we ignore second

and higher order terms, dJ' can be represented using equation 4.7

$$dJ' = (\Phi(\mathbf{x}(t_f), t_f) - \lambda(t))^T d\mathbf{x}(t_f) + \int_{t_0}^{t'} [(H_x + \dot{\lambda})^T \delta x + H_u{}^T \delta u + (H_\lambda - \dot{\mathbf{x}})^T \delta\lambda] dt \quad (4.7)$$

The necessary condition of optimality leads to the following canonical optimal control equations :

State equation :

$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \lambda} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \quad (4.8)$$

Costate Equation :

$$-\dot{\lambda} = \frac{\partial H}{\partial x} = (\frac{\partial f}{\partial x})^T \lambda + \frac{\partial L}{\partial x} \quad (4.9)$$

Stationarity Condition :

$$\frac{\partial H}{\partial u} = 0 \quad (4.10)$$

Boundary conditions :

$$\mathbf{x}(t_0) = \mathbf{x}_0 \quad (4.11)$$

$$\lambda(t_f) = \Phi_x(t_f) \quad (4.12)$$

In deriving these equations, we have assumed a specified final time t_f and no additional hard constraints on the state. For more general continuous optimization problems, only the boundary conditions are changed, the other canonical equations of the states and costates remain the same. [Bryson and Ho, 1975] provides an excellent introduction to the more general dynamic optimization cases. The solution of the optimization problem reduces to the solution of the $2n$ first order ordinary differential equations representing the state and costate equations with mixed boundary conditions, in addition to m algebraic equations representing the stationarity conditions. There are many classical iterative numerical methods for solving such problems. In general, these methods work by guessing one of the unknowns, such as initial or terminal states or a control trajectory, and then integrating the differential equations

and compute the different trajectroies, and then use the necessary conditions to iteratively improve the guess until all the necessary conditions are satisfied or no further improvement is obtained. For example if the optimal control $\mathbf{u}(t)^*$ could be written in terms of $\mathbf{x}(t)$ and $\lambda(t)$ using the stationarity conditions (as is the case in linear and bilinear systems for example), then the problem reduces to a normal two-point boundary value problem and could be solved by any of the numerical methods available for solving such problems such as the shooting or mesh methods [Numerical Algorithms Group, 1984]. In the following simulations, only general first order and second order gradient methods have been used. These start by guessing a control $\mathbf{u}^o(t)$ and then integrating the state and costate equations forward and backward respectively, then update the control in a direction that reduces the objective function. The simplest of such methods, gradient descent, updates the control in the direction of the gradient.

$$\mathbf{u}^{j+1} = \mathbf{u}^j - \epsilon(\frac{\partial H}{\partial u}) \qquad \epsilon > 0 \tag{4.13}$$

Second order gradient methods, such as the successive sweep algorithm, take into account the second derivative of the objective function with respect to u and x. The second order terms are computed recursively from the final time t_f backwards. The change δu in this case is similar to a time varying linear feedback law. Second order methods are much more computationally expensive, requiring the solution of a matrix Riccati equation ($n(n+1)/2$ differential equations) in addition to the solution of the n state equations. These methods have very good convergence near the optimal point, but require that the initial guess $\mathbf{u}^o(t)$ satisfies the constraint that H_{uu} be positive definite. In all the simulations that follow, a conjugate gradient algorithm has been used. The conjugate gradient method behaves like a first order gradient method in the first few iterations and then gradually approximates a second order method as the number of iterations increases. However, unlike second order methods, it does not require the computation of second order partial derivative matrices and is computationally much simpler. The basic conjugate gradient algorithm for optimal

control is illustrated by the following steps :

1. At iteration j, use the control $\mathbf{u}^j(t)$, and integrate the state equations forward from the given initial conditions x_0.

2. Integrate the costate equations backward starting from the final conditions $\lambda(t_f) = \Phi_x(\mathbf{x}(t_f))$, and compute $\lambda^j(t)$

3. Simultaneously with setp 2, compute the gradient $\frac{\partial H^j}{\partial u}$, using the relation $\frac{\partial H}{\partial u} = L_u + \lambda^T \mathbf{f}_u$

4. Set $\mathbf{u}^{j+1}(t) = \mathbf{u}^j(t) + l^j \mathbf{a}^j$, where the direction $\mathbf{a}^j$ is a conjugate direction and is found by the equation : $\mathbf{a}^{j+1} = -\frac{\partial H^{j+1}}{\partial u} + m^j \mathbf{a}^j$, l^j is a scalar that minimizes $J(u^j + l^j \mathbf{a}^j)$, and $m^j = \frac{\|\frac{\partial H^{j+1}}{\partial u}\|}{\|\frac{\partial H^j}{\partial u}\|}$

The conjugate gradient algorithm is used here because of its relative ease of computation over the second order methods, and the relatively faster convergence compared to the first order algorithms. Moreover, since the computed gradients are derived from the Gaussian HyperBF function approximation and are not very accurate, there is practically no advantage of using methods with higher convergence rates, such as second order methods. In addition, the second order methods require approximating the second derivatives and further care to guarantee the stability of the Riccati equations. In the simulations that follow, second order optimization techniques are only used after a few iterations of the optimization-learning cycle to compute optimal linear feedback gains in the vicinity of the open loop trajectory.

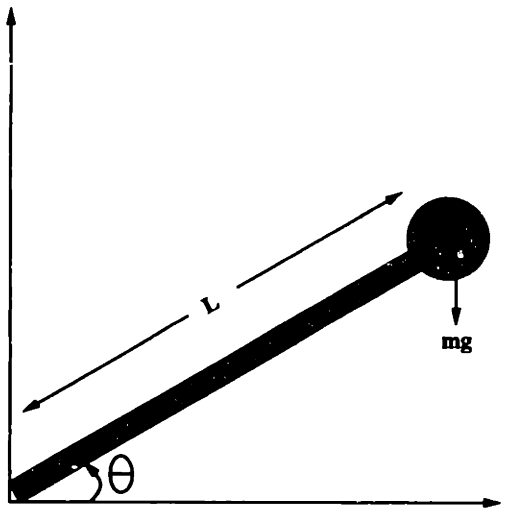

Figure 4-2: One Link Manipulator

4.5 Computer Simulations

4.5.1 One link minimum torque change trajectory

The first simulation represents a simple one link ideal manipulator shown in figure 4-2 and whose dynamics are described by equation 4.14.

$$\ddot{\theta} = T - Lmg\cos\theta \tag{4.14}$$

where T represent the torque, L is the length of the link, m represents the mass, and g is the acceleration due to gravity. The objective is to find the sequence of torques that moves this link from rest at $\theta_0 = 0$ to rest at $\theta_f = \frac{\pi}{2}$, while minimizing the change in torque applied over time. This minimum torque change criterion, described by equation 4.15 has been proposed by Uno et al. [Uno et al., 1989] as a possible criterion governing human arm trajectory planning. Uno has shown, using this criterion, that it is possible to simulate motions similar to those observed in

humans and to explain the asymmetry of the motions under gravity.

$$J = \int_{t_0}^{t_f} (\frac{\partial T}{\partial t})^2 \, dt \tag{4.15}$$

Other possible criteria have also been proposed [Nelson, 1983; Hogan and Flash, 1987], among them the minimum jerk hypothesis has been the most popular. The minimum jerk hypothesis is purely kinematic and does not depend on the configuration of the arm. For the simple one link manipulator under gravity, as in this example, and for a large range of movements, the minimum torque change criterion yields results very close to the minimum jerk criterion, and the minimum torque change trajectory appears to be very close to symmetric.

The forward model is represented by a Gaussian HyperBF network with three inputs (θ, $\dot{\theta}$, T) and one output ($\ddot{\theta}$). Although the angular velocity $\dot{\theta}$ does not appear in equation 4.14, and therefore does not have any effect on the evolution of the system, it is included here as one of the inputs to the HyperBF network to show the ability of the HyperBF approximation to deal with irrelevant variables. Since the cost function J contains derivatives of the torque and hence does not have the canonical form described in equation 4.2, the state equations are augmented with a fictitious state described by equation 4.16, so that the cost function will be a function of the states and controls only and reduce to the form of the cost described in equation 4.2. This augmentation of the state equations can also be used for higher order derivatives of the states or controls. It is assumed here that the learner knows the correct order of the system.

$$\dot{T} = u \tag{4.16}$$

$$J = \int_{t_0}^{t_f} u(t)^2 \, dt \tag{4.17}$$

$$\tag{4.18}$$

To initialize the HyperBF model, we used 50 random data points near posture to

estimate the HyperBF coefficients. We used 10 Gaussian RBFs distributed randomly in the same range as the data points. The centers of the RBFs remain fixed, but tneir number increases as more data are accumulated. The distribution of the initial data points as well as the location of the initial centers are shown in figure 4-3. This figure also shows the final optimal trajectory obtained after learning. As shown in the figure, the range of velocities and accelerations used in the initial estimation are very small and close to posture. The range of the initial velocities used to learn an initial model is about 2% of the range of values of the final learned trajectory. The range of initial acceleration used is about 10% of the range of accelerations of the optimal trajectory. Figure 4-3 also shows the ability of Gaussian HyperBF networks to linearly extrapolate from a narrow range of data. We used a zero-torque initial trajectory to initialize the learning-optimization algorithm, that is we let the arm swing freely under gravity. We then used this trajectory to update the HyperBF forward model, and used the conjugate gradient algorithm to find the optimal trajectory. The trajectory obtained after the first iteration of the algorithm is shown in figure 4-4. As shown in the figure, the trajectory found by the algorithm approximates very well the ideal trajectory. This ideal trajectory is obtained numerically using the exact model and the same conjugate gradient dynamic optimization. Also shown in the figure is the ideal minimum jerk trajectory obtained analytically. This latter trajectory can be modeled by a fifth order polynomial for the position. All the subsequent motions after the first trial were essentially equal to the ideal movement.

4.5.2 Minimum torque change of a two link manipulator

This example shows how to apply the algorithm described above to learn the minimum torque change trajectory for a two link manipulator. The goal is to plan a trajectory from a starting initial position to a final position in a specified time so as to minimize the sum of the rate of change of torques applied at the joints, without prior knowledge of the dynamics of the arm, except for a complete measurement of the states as a

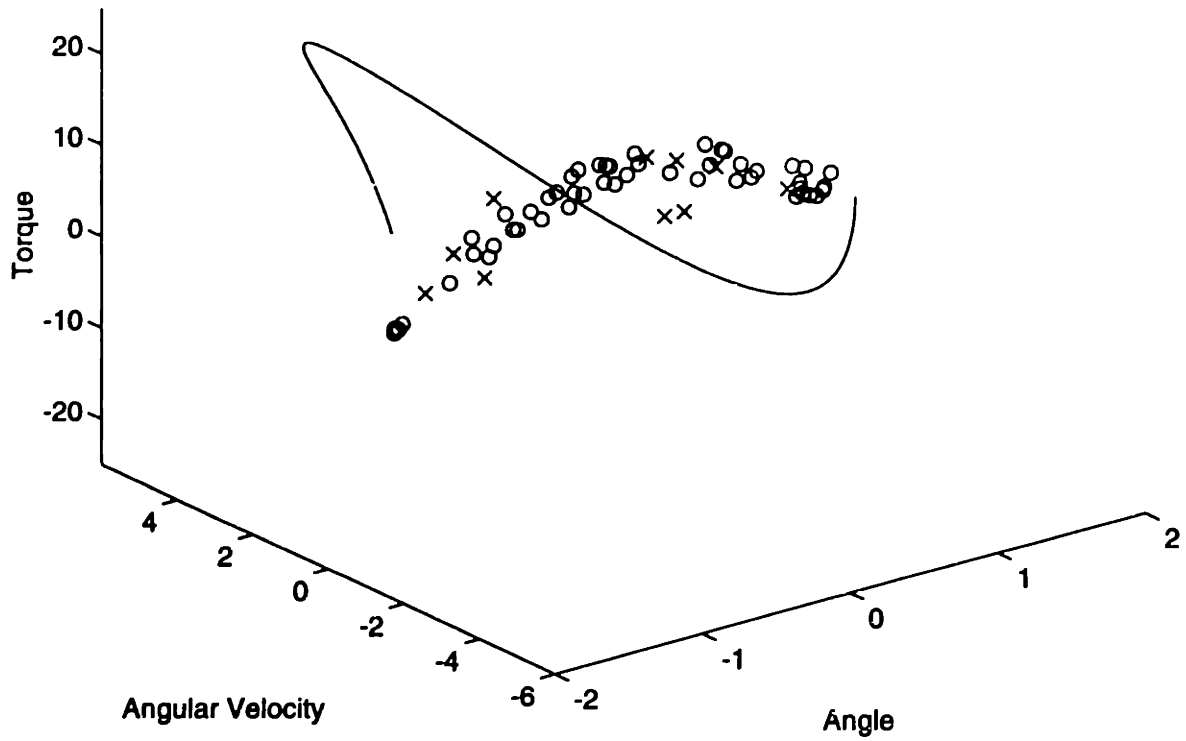

Figure 4-3: Initial distribution of angles, angular velocities and torques used to generate the initial HyperBF model. The symbol 'o' represents the location of the data points, the symbol 'x' represents the initial location of the centers of the HyperBFs. The continuous trajectory represents the learned minimum torque change trajectory.

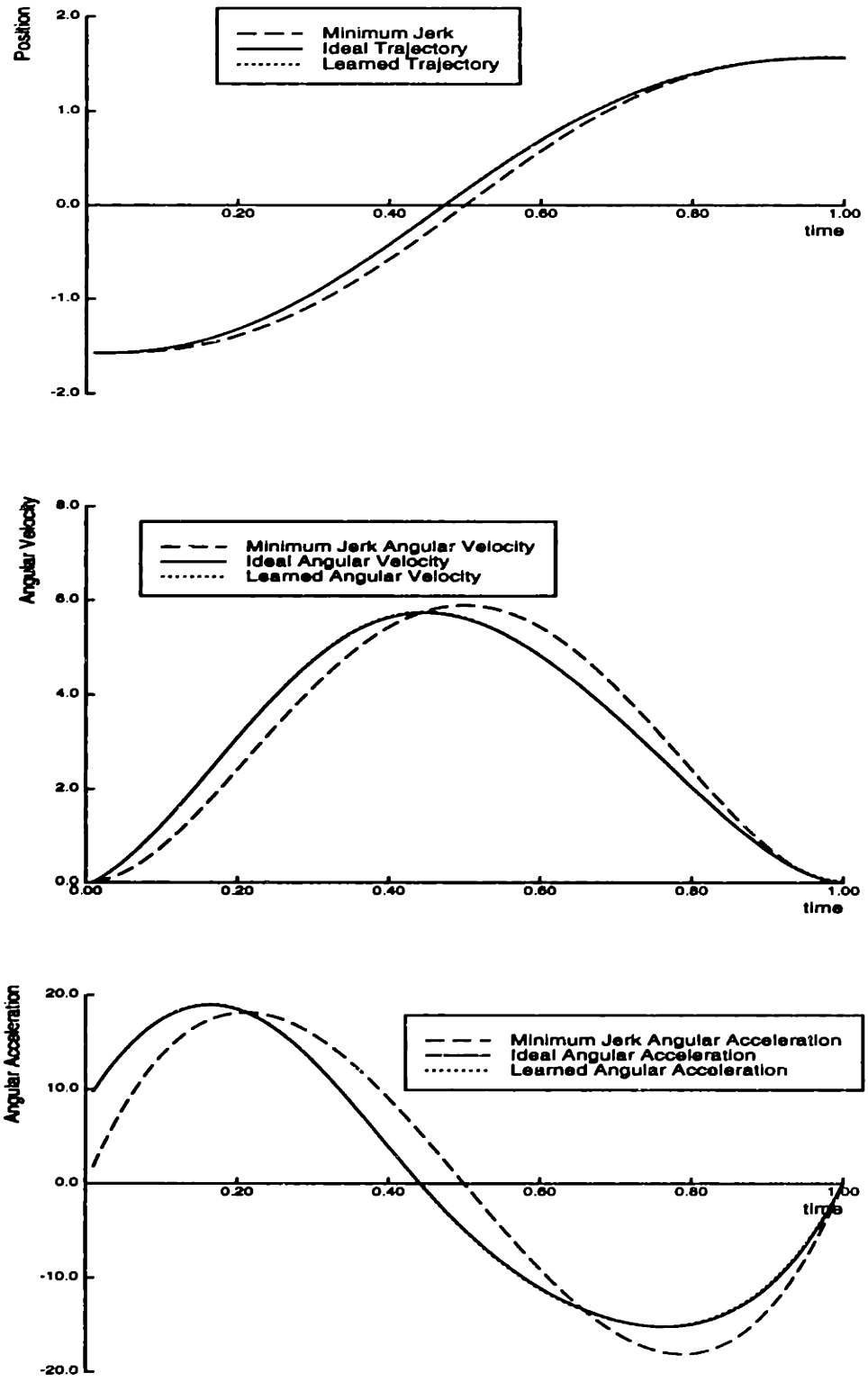

Figure 4-4: Learned Minimum Torque Change Trajectory for a One Link Manipulator. Also shown in the figure are the correct model minimum torque change and minimum jerk trajectories for the same movement for comparison

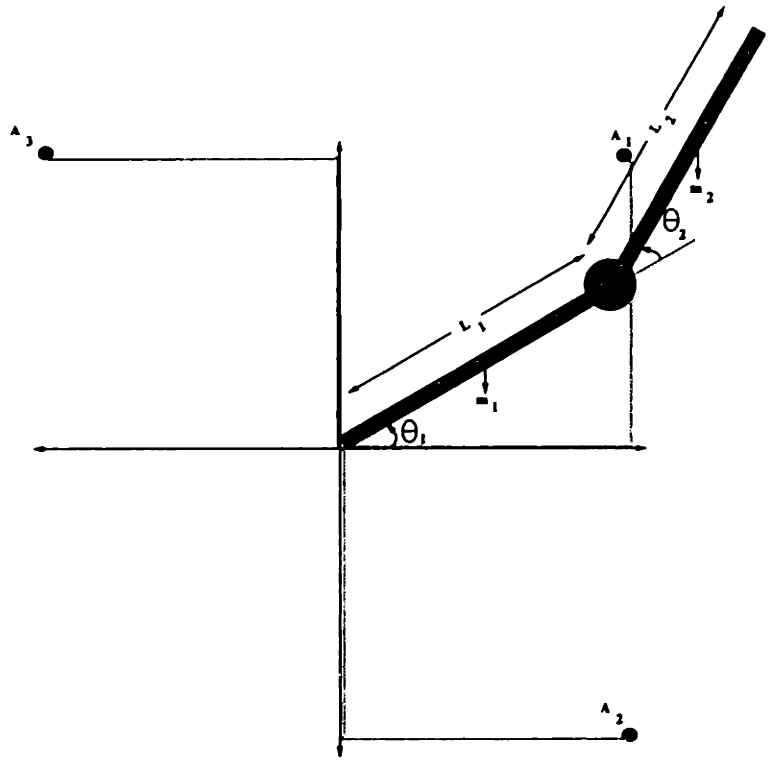

Figure 4-5: A two link manipulator. Points A_1, A_2 and A_3 are used as initial or final position in the simulations described in the text

function of time (e.g. angular positions and velocities), in addition to the torques and accelerations. Three points A_1, A_2 and A_3 in figure 4-5 are selected to be either the starting or the final position in all the simulations that follow. In the simulations, the forward model of the dynamics is represented by two Gaussian HyperBF networks, the outputs of which model the angular accelerations at the two joints. Each HyperBF network has six inputs representing the angles, angular velocities and torques at the two joints. The HyperBF networks are also used to generate the derivatives of the angular accelerations with respect to the different states and torques. These will be used by the dynamic optimization subroutine. Since the objective function is not in the general canonical form and contains derivatives of torques, two fictitious states are added to the forward model. The time derivatives of the torques are considered

to be the control variables, and the torques at the joint are considered to be two additional state variables. This representation reduces the problem to the general canonical form.

We initialize the HyperBF networks using 120 random data points near posture selected at relatively low accelerations and velocities. The ranges of velocities and accelerations are the same as used in the one link case. An initial movement is then generated using zero torques at the joint and starting from the initial states. The torques are then updated using the conjugate gradient algorithm and the forward Gaussian HyperBF networks. The new torques are then applied after the addition of random white noise and the forward model is updated using the new experiences. A new data point is only added to the pool of experiences if the distance between this point and all the previous points is larger than a minimum distance δ. The distance metric used to define δ is the same as the one used in the HyperBF approximation. We iterate the procedure until the trajectory stops improving. The torques, angular positions, velocities and accelerations at the two joints are shown in figures 4-6 and 4-8 at four different iteration numbers. As observed from the figures, the learned trajectory landed at the desired states in relatively few iterations of the algorithm (less than 20). The trajectory of the end point for the movements represented in figures 4-6 and 4-8 are shown in figures 4-7 and 4-9 respectively. As clear from the figures, these trajectories are not symmetric, and curved and depend on the starting and final positions, unlike the minimum jerk trajectories which are straight lines and symmetric.

Unlike the minimum jerk trajectories, there is no closed form solution for the minimum torque change trajectories. Therefore, in order to test the accuracy of the learned minimum torque trajectories obtained, we compare them with those obtained numerically using the exact model of the two link arm and the same numerical conjugate gradient optimization algorithm with the same initial conditions. Figure 4-10 and 4-12 are plots of the trajectories that minimize the sum of the rate of change

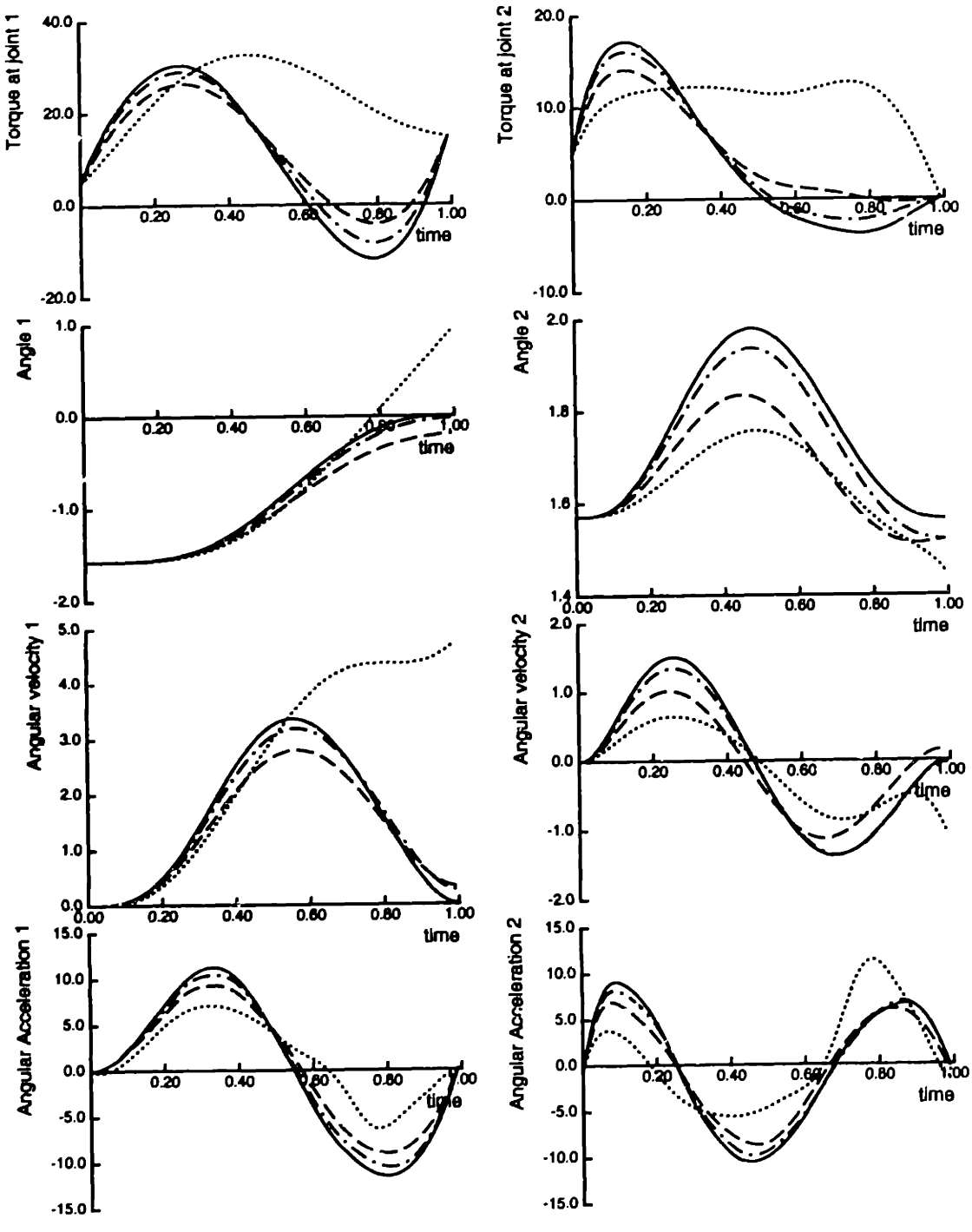

Figure 4-6: Torques, angular positions, velocities and accelerations at both joints at four different iteration numbers. The dotted, dashed, dotdash and solid lines represent iteration 1, 5, 10 and 20

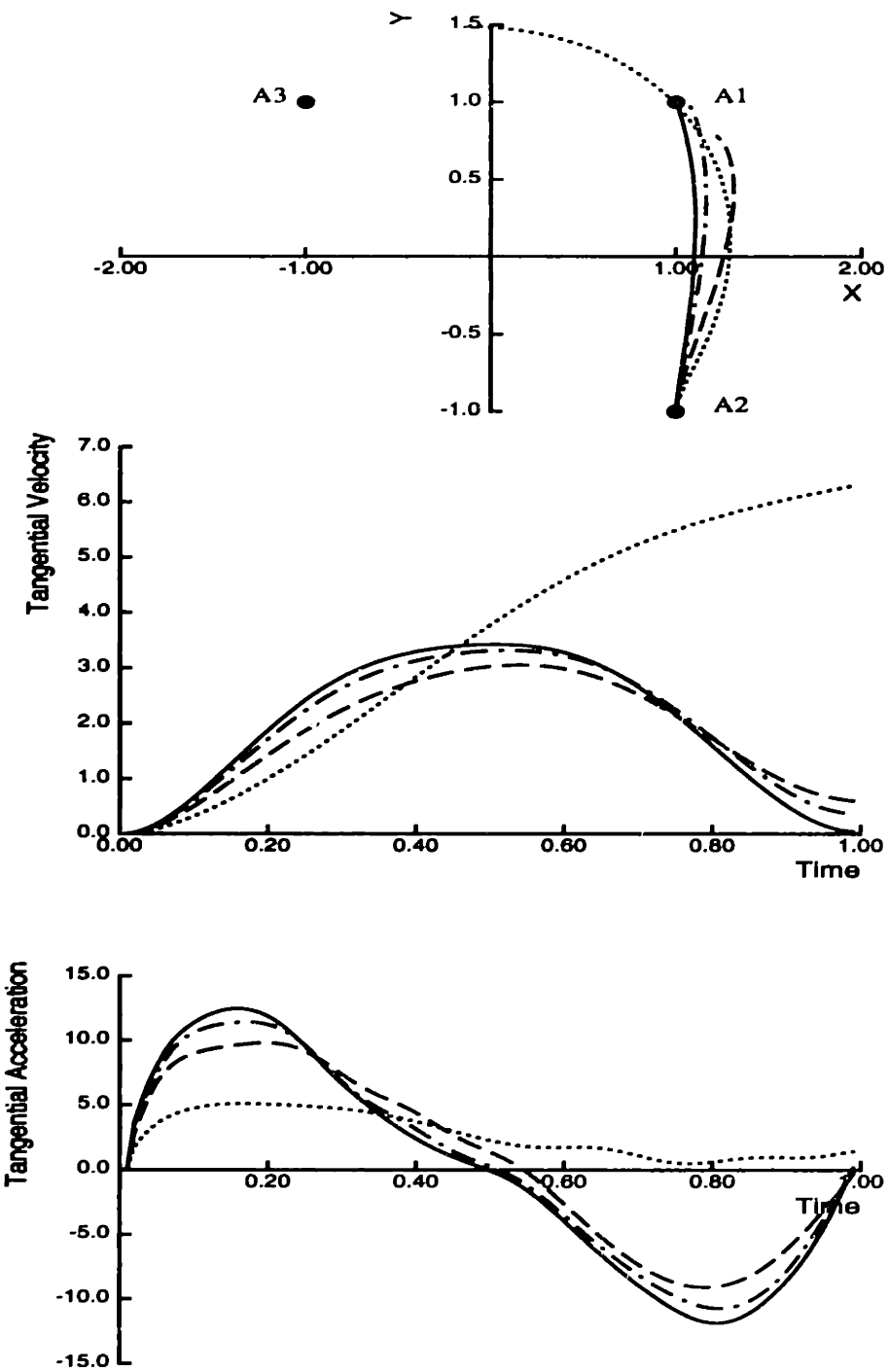

Figure 4-7: X, Y positions, tangential velocities and accelerations for the movements shown in the previous figure. The dotted, dashed, dotdash and solid lines represent iteration 1, 5, 10 and 20 respectively.

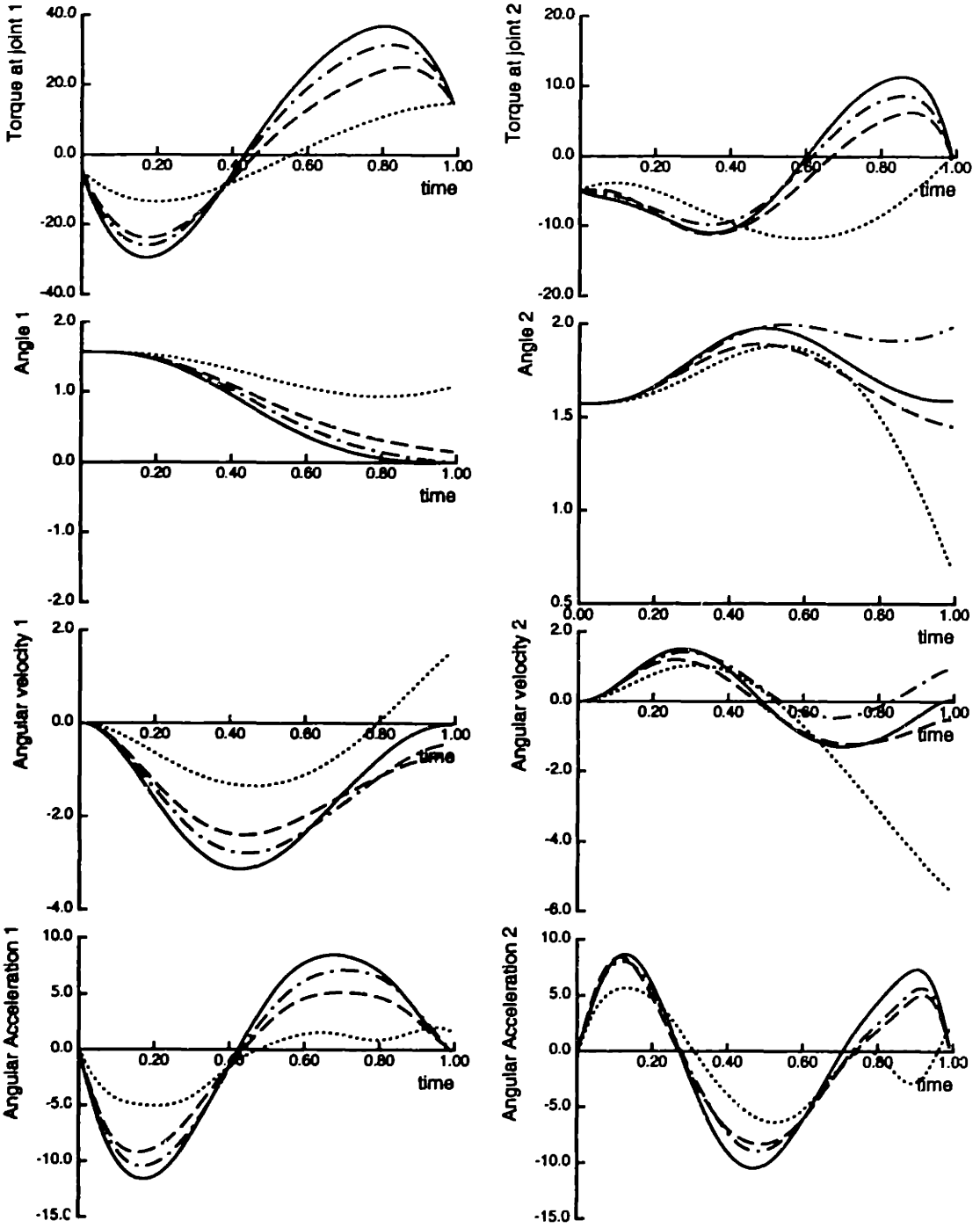

Figure 4-8: Torques, angular positions, velocities and accelerations at both joints at four different iteration numbers. The dotted, dashed, dotdash and solid lines represent iteration 1, 5, 10 and 20

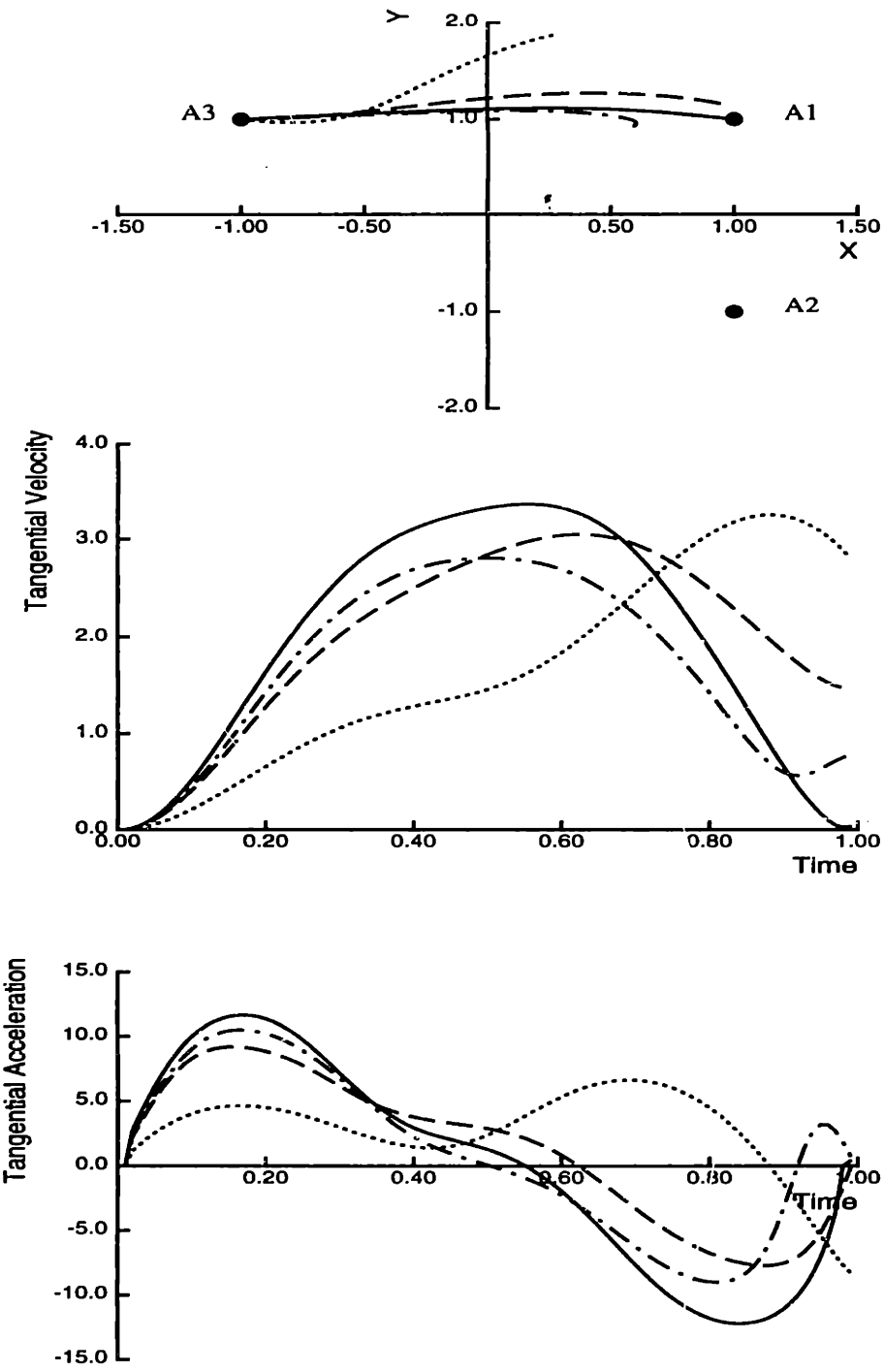

Figure 4-9: X, Y positions, tangential velocities and accelerations for the movements shown in the previous figure. The dotted, dashed, dotdash and solid lines represent iteration 1, 5, 10 and 20 respectively.

of torques obtained by numerical optimization on the exact model and the learning optimal control algorithm described above. As shown in the figures, both trajectories, although not exactly the same, agree very well, except for the second joint trajectory when moving the arm upwards. The reason for this discrepancy is due to the presence of local minima that result from the use of high penalties in the cost function to enforce the end constraints. This is illustrated by figure 4-11 which shows a comparison of the accumulated torque change cost between the ideal and the learned trajectories of figure 4-10. As shown in this figure, the accumulated cost is approximately the same at the end of the trajectory.

It is important to stress here that the forward model of the dynamics obtained after learning is only accurate around the optimal trajectory. Although the forward model predicts very well the dynamics near the optimal trajectory, it looses its accuracy in other areas of the state space.

4.5.3 Trajectory Following

There are many different methods for learning trajectory following using function approximation techniques or neural networks. These include directly modeling the inverse dynamics and/or modeling the forward dynamics as described in chapter 2. In this section, we describe another method for trajectory following using forward modeling and dynamic optimization, by representing the problem as an optimal control problem. The cost function in this case is a measure of the error between the desired and actual trajectories in addition to any other constraints or costs of the control variables. We use the learning optimal control paradigm to generate the open loop control trajectory for a two link planar robot, so that the end-point follows a vertical circular trajectory with a constant angle of rotation. The cost function is described in terms of the end-point cartesian coordinates. Although the objective function could also be described in terms of the joint coordinates, there are some advantages for using the end-point cartesian coordinates instead. First, errors in end-point coordi-

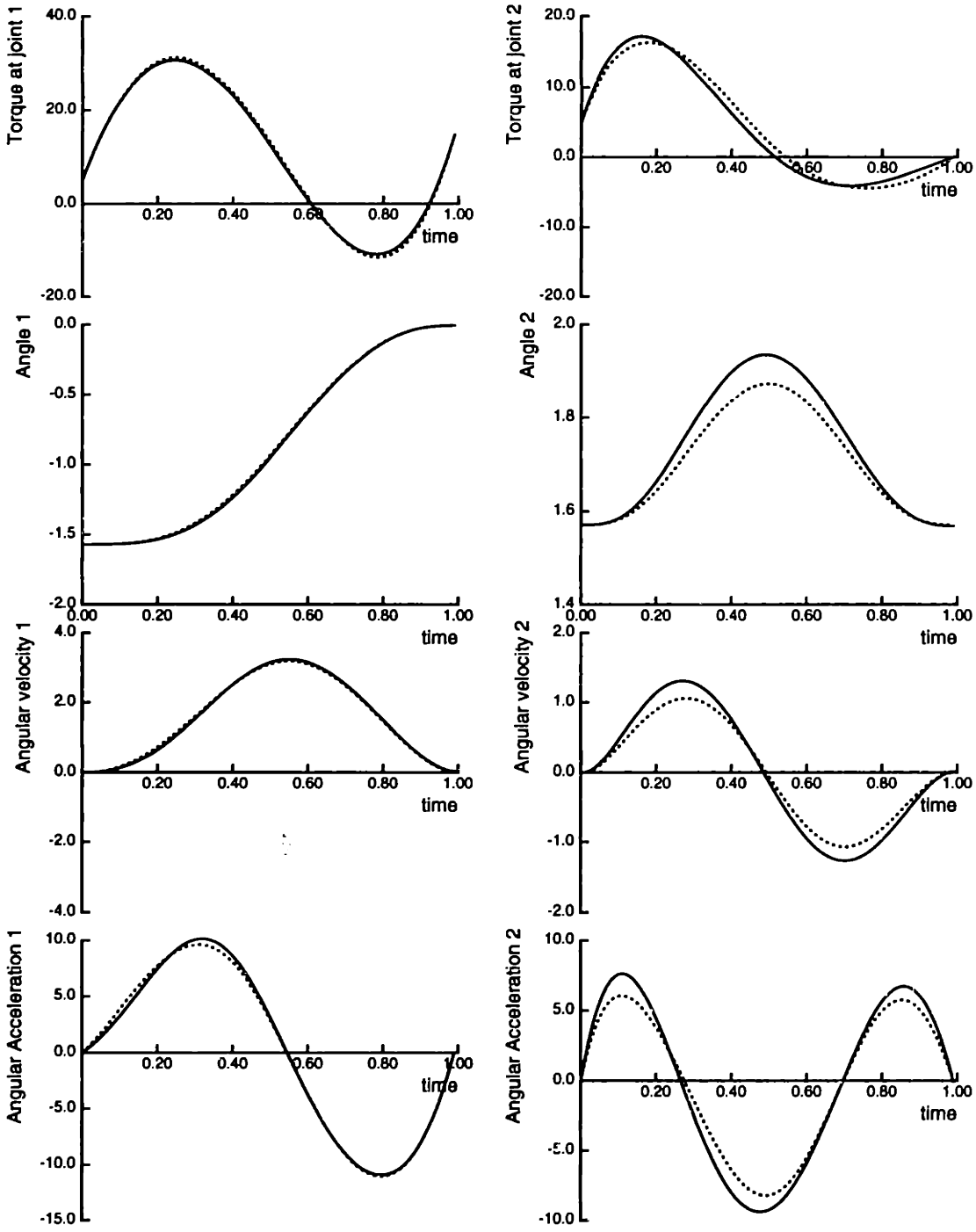

Figure 4-10: Comparison of torques, angular positions, velocities and accelerations using dynamic optimization on the exact and the learned models. The solid lines represent the minimum torque change trajectories obtained using the exact model and the dashed lines represent the trajectories after 20 iterations of the learning optimization algorithm. These trajectories represent moving the arm vertically upwards.

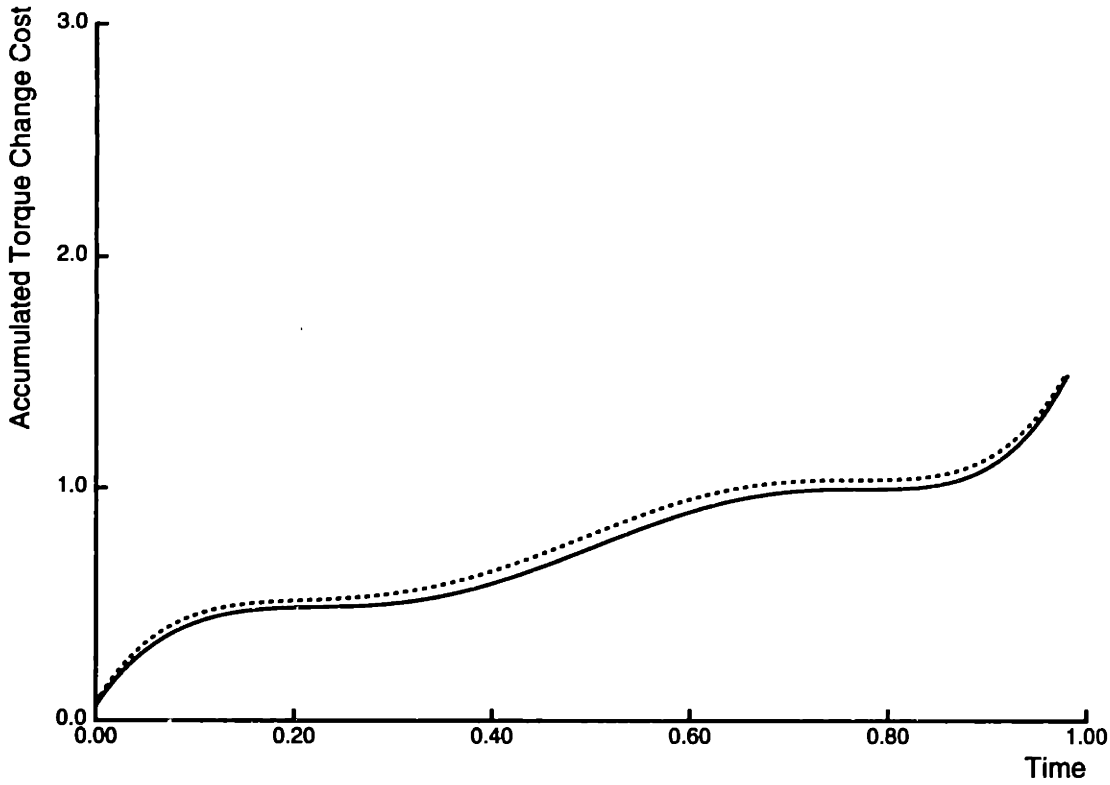

Figure 4-11: The accumulated minimum torque change cost as a function of time for the trajectories generated by the HyperBF controller and the ideal ones. The solid lines represent the computed minimum torque change cost using the exact model, while the dotted lines show the minimum torque change cost obtained by optimizing the learned dynamic model.

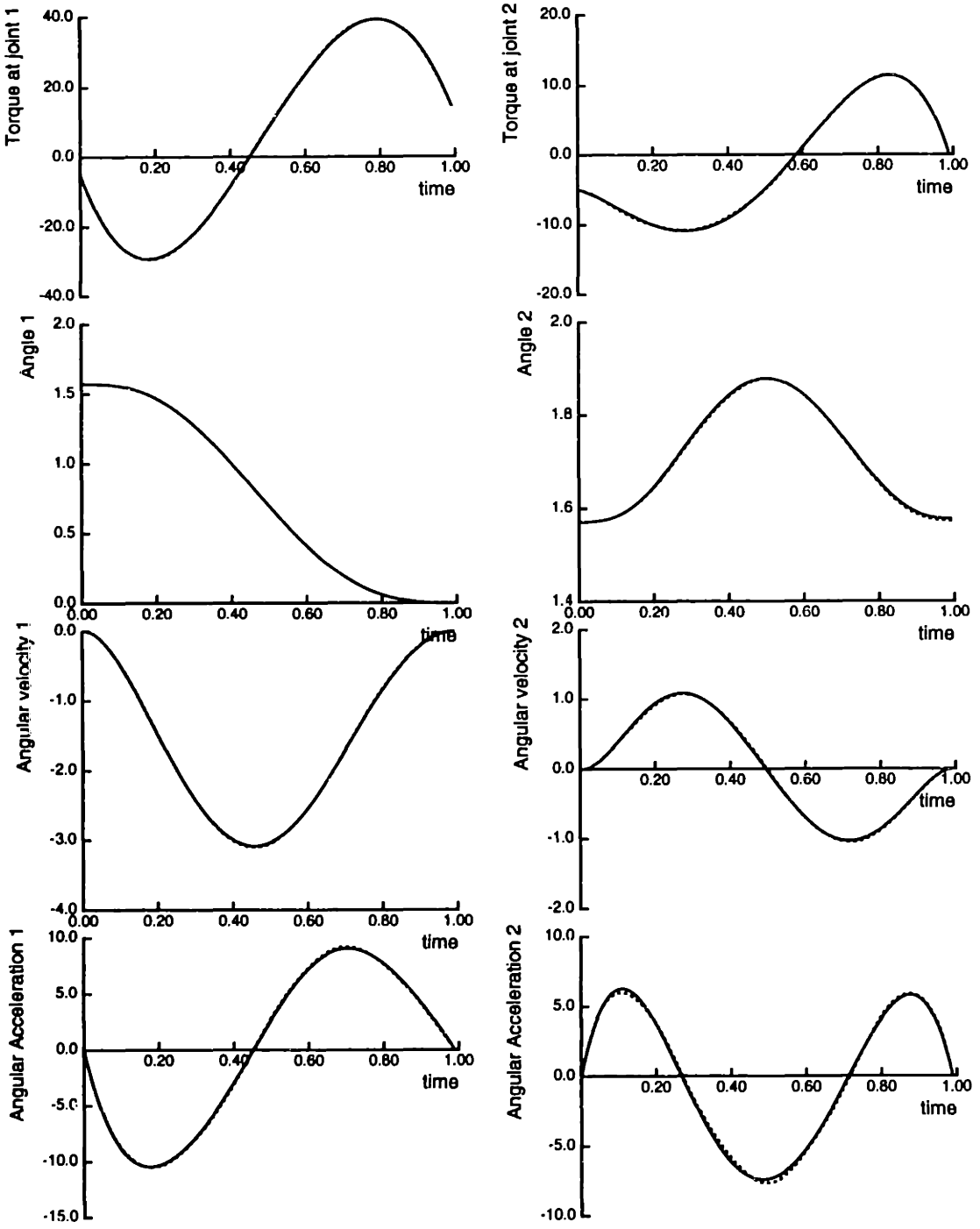

Figure 4-12: Comparison of torques, angular positions, velocities and accelerations using dynamic optimization on the exact and the learned models. The solid lines represent the minimum torque change trajectories obtained using the exact model and the dashed lines represent the trajectories after 20 iterations of the learning optimization algorithm. These trajectories represent moving the hand horizontally from left to right.

nates are in general a nonlinear function of the errors in joint coordinates, and what matters most for this task is to keep the magnitude of the errors in end-point coordinates as small as possible. In addition there may be more than one solution for the trajectory following, and using the end point coordinates will result in not restricting ourselves to any particular solution a priori. However, for the particular two joint arm simulation described here, there are only two possible solutions. Which one will be chosen depends on the initial conditions and/or the limitations on the joint angles. The cost function used is described by equation 4.19 :

$$J = \int_{t_0}^{t_f} (\mathbf{x}_{ep}(\theta, \dot{\theta}) - \mathbf{x}_{ref})^T Q (\mathbf{x}_{ep}(\theta, \dot{\theta}) - \mathbf{x}_{ref}) + \tau^T R \tau dt \qquad (4.19)$$

where $\mathbf{x}_{ref}$ is a vector $\in \mathcal{R}^4$ describing the reference x, y positions and velocities, $\mathbf{x}_{ep} \in \mathcal{R}^4$ represents the end point x and y positions and velocities and is a function of the joint angles positions and velocities. The term $\tau^T R \tau$ represents the torque cost. We assume complete knowledge of the kinematics of the robot, although these also may be learned from examples.

The simulation results for following a vertical circular trajectory are shown in figure 4-13. This figure shows the open loop simulation, using the learned optimal trajectory for the first, 5^{th} and 20^{th} iteration of the algorithm. As shown in the figure, at around the 5^{th} iteration, the open loop trajectory obtained is already very close to the reference one. Another point to note from the figure is that even after 20 iterations, there exist a very small error in the open loop trajectory. This reflects of course the small inaccuracies of the forward dynamics model and shows the need for feedback to obtain exact trajectory tracking.

4.6 Feedback Control

All the simulations of the previous section were done using the optimal open loop control trajectory obtained from the dynamic optimization routine. However, in

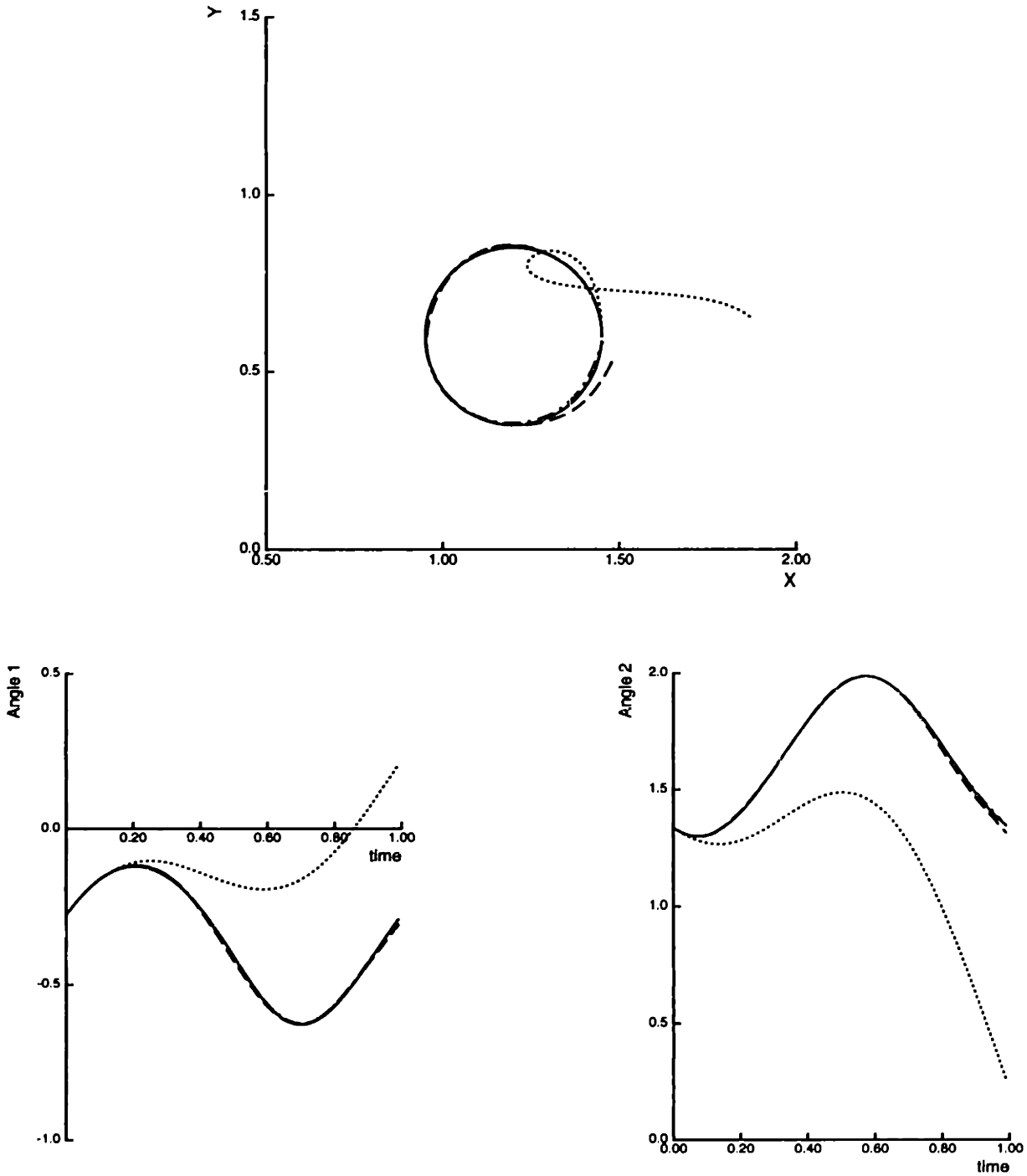

Figure 4-13: Circular trajectory tracking for a 2 joint manipulator. Dotted, dashed and dotdash curves represent trials number 1, 5 and 20 respectively. Solid curves represent the reference trajectory.

practical applications, open loop control is undesirable due to its lack of robustness in the presence of uncertainties and noise. There are many possible ways we can include feedback control in order to make the current open loop method more robust. We will discuss some of the possibilities and show, using simulations, how these techniques can be used to obtain an optimal feedback controller. These simulations will also show the difficulties associated with the use of these techniques.

4.6.1 Dynamic programming around the optimal trajectory

Dynamic programming provides a global optimal control law. Unfortunately, due to the curse of dimensionality, dynamic programming can not be used for more than few dimensions. Both the amount of memory and the number of computations needed to compute the optimal law grow exponentially as the number of dimensions increase. However, one of the advantages of dynamic programming is that the problem becomes simpler as the set of possible actions and states is reduced. Therefore, if the number of input and control dimensions is small (3 or 4), we use dynamic programming in a narrow tube around the optimal trajectory. The advantage is that we can discretize the state space at a higher resolution inside this tube and we can obtain a higher resolution control law. However, the curse of dimensionality problem is not solved. We can also add a supervisory control whose role is to return the state of the system to the tube around the optimal trajectory, if it leaves that tube. Sliding control could be used for that purpose. This type of feedback is beyond the scope of this thesis and will not be addressed further.

4.6.2 Learn an optimal feedback law using the field of extremals

We have seen that for more than 3 or 4 dimensions, dynamic programming becomes unpractical due to the curse of dimensionality. Basically, the problem is that as the

number of dimensions increase, the state/control space volume increases exponentially, and the number of possible alternatives at each instant of time becomes very large. One possibility to solve this problem is to sample this huge volume randomly near the desired trajectory and then use the generalization power of function approximation to obtain an approximate control law in the neighborhood of interest. The algorithm for learning a control law using approximated dynamic programming works as follows :

1. First, a model of the dynamic system is learned by approximating the experiences from previous trials using a neural network (HyperBF for example).

2. Using the calculus of variation and the model developed in 1, we generate many optimal trajectories using many different initial conditions to the same final hypersurface.

3. Again use function approximation, or neural networks, to approximate the optimal control $\mathbf{u}^*$ as a function of the states $\mathbf{x}$. In general, optimal trajectories do not intersect except in very special circumstances, which makes the control $\mathbf{u}^*$ as a function of the different states defined. However, one important point to keep in mind is that the optimal policy $\mathbf{u}^*$ is in general a function of time, in addition to the states, if the final time is specified. This is true even if the instantaneous cost and the dynamic system are not functions of time. Therefore in that case time should be used as an input variable in addition to the state variables, to determine the optimal policy $u(\mathbf{x}, t)$. The optimal policy is independent of time when the final time is unspecified and when both the instantaneous cost and the dynamic equations are not explicit functions of time, as is the case of minimum time trajectories of time invariant systems.

Although the use of this approach is much more robust than open loop trajectory generation, there exist some difficulties associated with this approach: for example, the extremal fields generated by the dynamic optimization routine should represent

the same local minima, otherwise the control law may not be smooth, and the approximated control law will be very inaccurate. Another difficulty, is that in the case where the control law is time varying, and there are hard constraints to be met exactly at the final time, the control law is very sensitive to the states near the final time, which makes it very hard to represent using neural networks. Therefore, we do not recommend using this approach by itself for finite horizon systems since the dependence on time may complicate the approximation. It should be augmented with additional feedback controllers to prevent instabilities due to inaccuracies of modeling the controller.

We used this approach to learn the minimum torque change optimal control law for the two link robot arm. The optimal trajectories used to learn the optimal control were generated using the exact model of the robot. The learned HyperBF optimal controller had 7 inputs (angular positions, velocities and torques in addition to time), and 2 outputs (one-step-ahead torques). An example of the trajectories generated using this approach to learning optimal feedback is shown in figure 4-14 for moving the two joint arm vertically upward. The trajectory generated using the optimal control law approximated by the Gaussian HyperBF network differs from the optimal one generated using open loop optimal control with full knowledge of the dynamics. The difference in the cost function between the optimal trajectory and the approximated suboptimal one, is shown as a function of time in figure 4-15. As obvious from the figure, although the final state reached is very close to the desired state, the final cost is about 25% higher when the HyperBF network optimal controller is used. We attribute this higher cost to the difficulties mentioned above. It is interesting to note that most of the increase in cost was generated during approximately the first fifth of the trajectory, which shows the problem of the controller near the start of the trajectory.

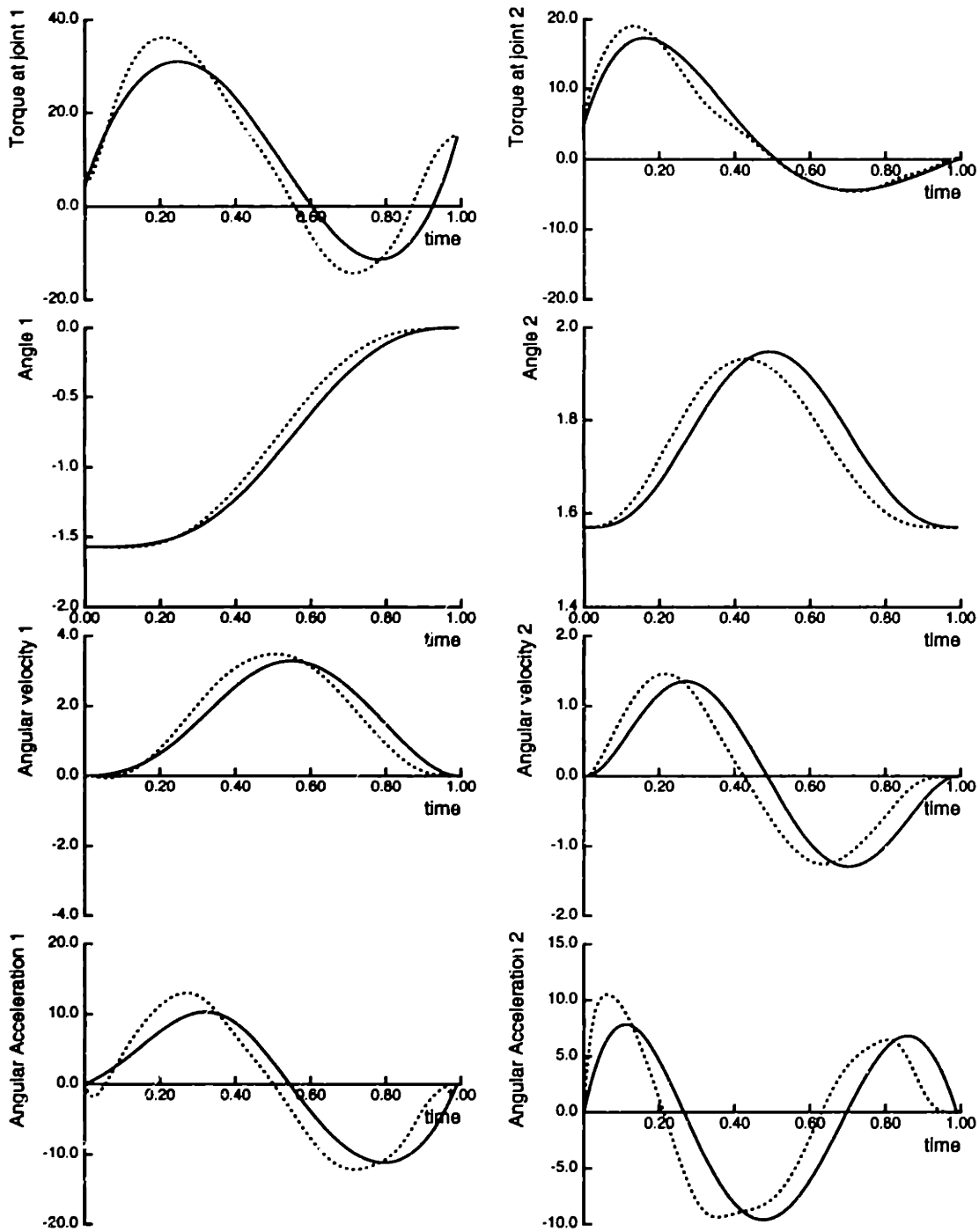

Figure 4-14: Performance of the learned optimal controller. The solid lines represent the minimum torque change trajectories obtained using the exact model and the dashed lines represent the trajectories generated by the optimal controller. These trajectories represent moving the hand vertically upwards.

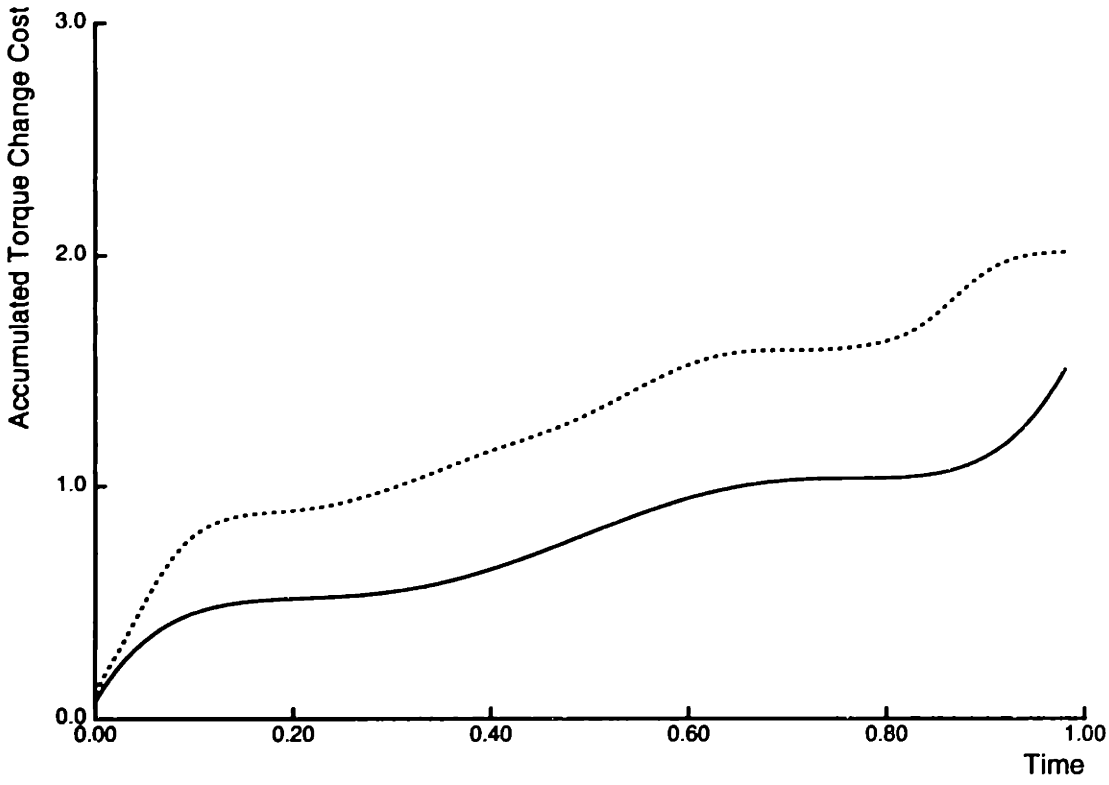

Figure 4-15: The accumulated minimum torque change cost as a function of time for the trajectories generated by the HyperBF controller and the ideal ones. The solid lines represent the computed minimum torque change cost using the exact model, while the dotted lines show the minimum torque change cost using the HyperBF controller.

4.6.3 Use linear optimal feedback

The use of calculus of variations with linear systems and quadratic cost functions results in an optimal linear feedback law. By linearizing the state dynamic equations of a nonlinear system around the optimal trajectory, and expanding the cost function to second order, we obtain an equivalent linear quadratic problem. We can then use the same linear quadratic dynamic optimization theory to obtain a suboptimal linear feedback gain around the optimal trajectory. This method will work as long as the perturbations around the optimal trajectory are kept small. The optimal linear time varying gains can be generated using the learned model of the dynamic system. We tested the application of this method on the robot trajectory tracking simulation shown in the previous section. Figure 4-16 shows a comparison between the open loop and the closed loop trajectories using the optimal linear feedback gains generated using this technique. As shown in the figure, when there are perturbations in the states, the open loop performance deteriorates considerably, while the closed loop error decreases with time.

4.6.4 Use a real-time open loop controller

One possibility for obtaining a closed loop feedback control, is to use an open loop optimal controller which is updated often relative to the rate of change of the system states. Practically, this is a difficult task since the amount of computation involved in computing the optimal control makes it very hard to be useful for real time applications. Recently, however, there have been an increased interest in solving optimization problems using parallel processing and some neural networks models such as Hopfield Networks. Parallel processing may allow us to solve optimal control problems in real time. In this section we will discuss some possible alternatives for implementing optimal control problems in parallel analog hardware. In particular, we propose different Hopfield like neural networks that implement the optimal control problem for linear systems with quadratic costs. For linear systems, these networks are guaranteed to

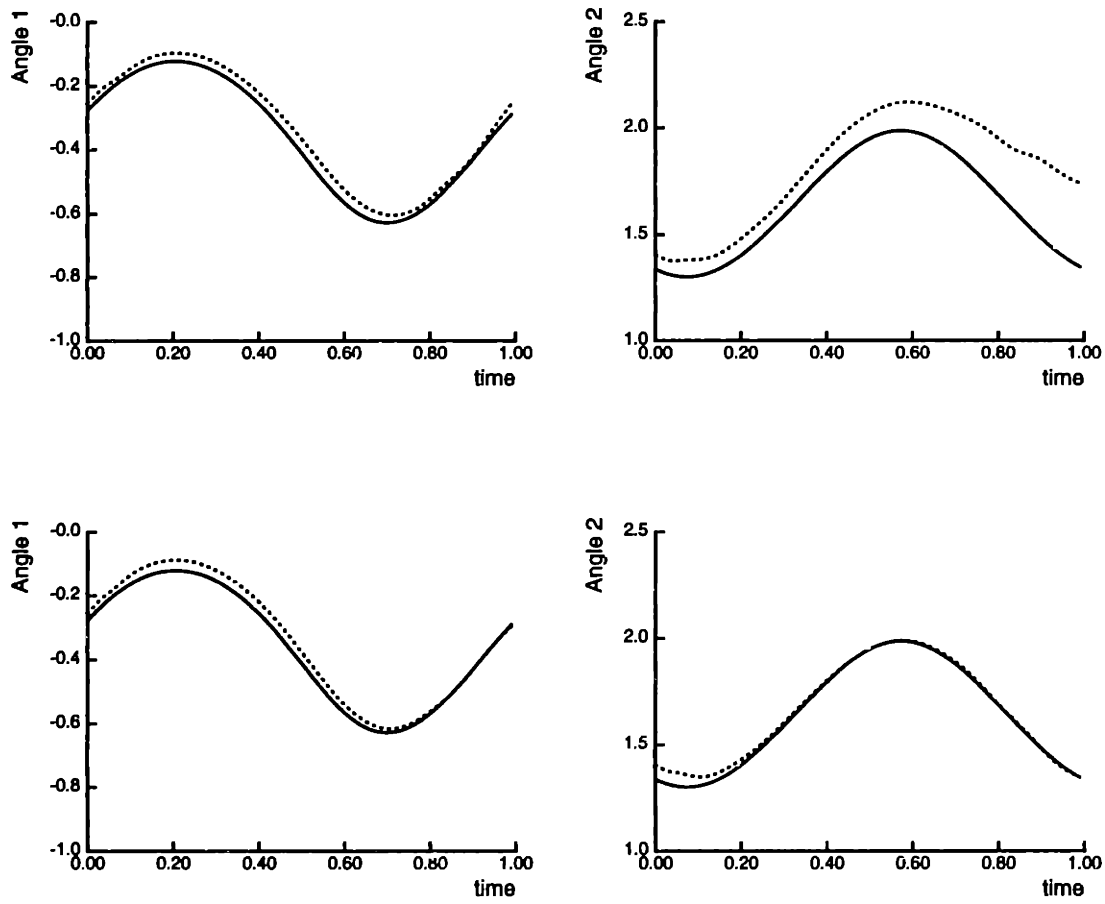

Figure 4-16: Comparison of the open loop and the optimal linear feedback closed loop angular positions of the simulated 2-joint robot arm as a function of time. The upper plots represent the open loop performance, while the lower plots represent the closed loop optimal linear feedback performance at the two links. The solid lines represent the reference trajectories and the dotted lines represent the trajectories generated using the optimal controls.

be stable and converge to local minima (Global minima for the case of linear systems with quadratic costs). Other researchers have also proposed similar techniques for the solution of optimal control problems [Lan and Chand, 1990; Mears, Smith, Chandler and Pachter, 1993]. We will also explore extending the same techniques to non-linear optimization problems. Unlike linear systems, stability of the optimization process and convergence to the global minima are not automatically guaranteed for nonlinear systems. A block diagram that describes a general closed loop optimal nonlinear controller based on a Hopfield type optimization network is shown in figure 4-17. System identification is used to estimate the forward model. The learned forward model and its partial derivatives with respect to the states of the system and controls, are then used to estimate the nonlinear connectivity matrix of the Hopfield network.

Hopfield optimal control implementation using penalty functions

Lan and Chand [Lan and Chand, 1990] and Mears et al. [Mears et al., 1993] have proposed transforming the dynamic optimization problem to a static one by using a penalty function to append the dynamic constraints into the cost function as shown in equation 4.20.

$$J'(\mathbf{x},\mathbf{u}) = \sum_{k=1}^{N} \mathbf{x}_k^T Q \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k + \kappa \|\mathbf{x}_{k+1} - A_k \mathbf{x}_k - B_k \mathbf{u}_k\| \tag{4.20}$$

where κ is a scalar in this case that does not depend on time. Using the update formula 4.21, they were able to prove Lyapunov stability, provided that the matrices Q and R are positive definite.

$$\dot{\mathbf{x}}_k = -\epsilon \frac{\partial J'}{\partial \mathbf{x}_k} \tag{4.21}$$

$$\dot{\mathbf{u}}_k = -\epsilon \frac{\partial J'}{\partial \mathbf{u}_k} \tag{4.22}$$

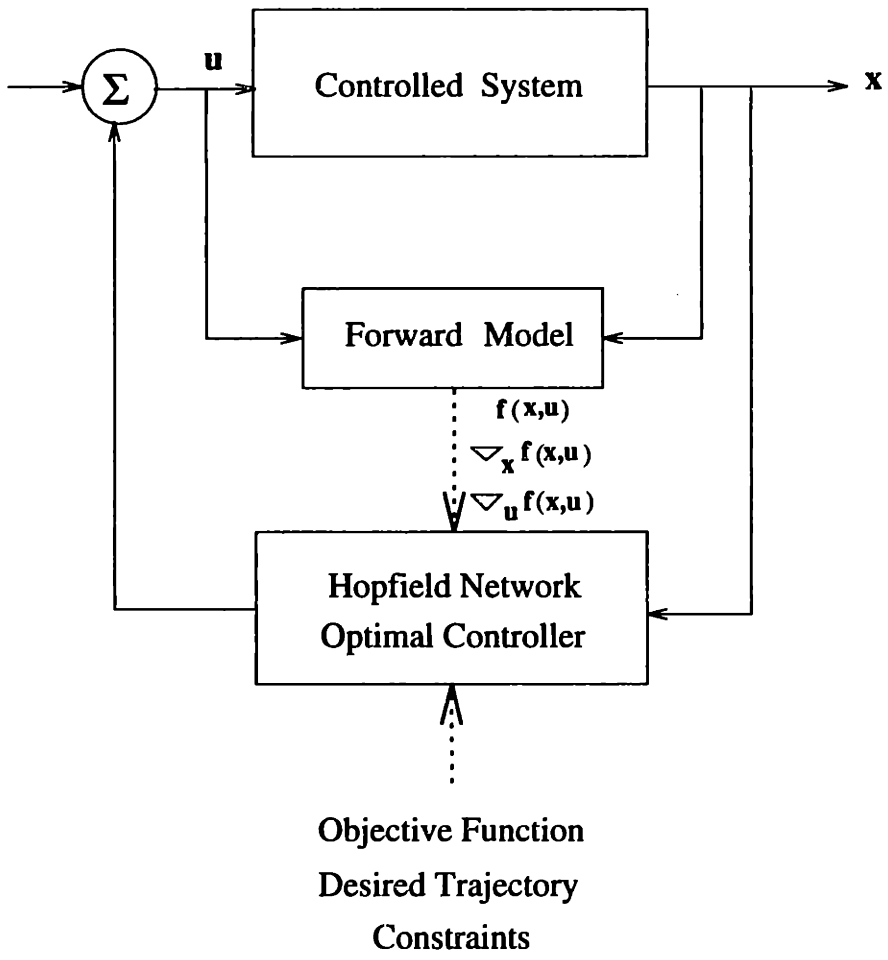

Figure 4-17: A closed loop controller based on a Hopfield type optimizing network

Stability can be proven by showing that $\frac{dJ'}{dt}$ is negative. They proposed the use of a Hopfield Network to implement the static optimization problem. There are some disadvantages to the above procedure due to the penalty function implementation of the dynamic equations. In order not to violate the dynamic constraints, the penalty coefficients should be very high. This makes the optimization problem ill-conditioned and consequently results in very slow convergence to the optimal point. Reducing the penalty coefficient on the other hand may result in completely erroneous results. To see this, consider the simple example shown in equation 4.23

$$x_{k+1} = 0.9x_k + 0.1u_k \qquad x_0 = 1; \;\; x_1 = 0 \tag{4.23}$$

$$J(x,u) = \sum_{k=0}^{10}({x_k}^2 + {u_k}^2)dt \tag{4.24}$$

$$\tag{4.25}$$

One simple solution that will minimize the augmented cost function J' is to have $u_k = 0$ and $x_k = 0$ for $k > 0$. This of course will violate the dynamic constraint at $k = 0$. In the next section, we propose different Hopfield implementation of the linear quadratic optimal control problem, that exactly implement the dynamic constraints and which are also guaranteed to be stable and converge to local optima. All the techniques proposed may also be extended to nonlinear systems. In all these techniques, the optimal control problem is considered as a static nonlinear programming problem, and then standard first order gradient based techniques are used to map the problem to a neural network.

Constraint satisfying Hopfield Networks

We discuss in this section few different implementations of Hopfield networks optimal controllers which are based on well known constrained optimization techniques [Luenberger, 1984]. In particular, the networks we propose here are based on the following methods:

- Gradient projection [Kirk, 1970; Luenberger, 1984]
- Lagrange multipliers [Bertsekas, 1982]
- Augmented Lagrange multipliers (Method of multipliers) [Bertsekas, 1982]

We compare the advantages and disadvantages of each method. Using simulations, we show how these methods can be used for the very fast computation of the optimal control.

1. Gradient Projection Hopfield Network

One possible exact numerical solution to the optimal control problem is through the use of gradient projection [Kirk, 1970]. The idea of this approach is to first discretize the cost function so that it is transformed into a static optimization problem, then use a form of gradient descent to update the states and controls. The gradients of the objective function are first projected into the hypersurface representing the dynamic constraints. This procedure is summarized by the following equations :

$$minimize J(\mathbf{x},\mathbf{u}) = \sum_{k=1}^{N} \mathbf{x}_k^T Q \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k \tag{4.26}$$

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \quad k = 0, \ldots, N \tag{4.27}$$

The dynamic equations can be transformed into the following form :

$$\mathbf{M}^T \mathbf{v} = \mathbf{c} \tag{4.28}$$

where : $\mathbf{M}^T$ is an $Nn \times N(n+m)$, N is the number of time steps, n the dimension of the states and m the dimension of the controls.

$$\mathbf{M}^T = \begin{bmatrix} I & & & -B & & & \\ -A & I & & & -B & & \\ & -A & I & & & -B & \\ & & & & & & \\ & & -A & I & & & -B \end{bmatrix} \tag{4.29}$$

$$\mathbf{v} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \\ \mathbf{u}_0 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix} \tag{4.30}$$

and the vector $\mathbf{c}$ depends on the boundary conditions. Define the projection matrix $\mathbf{P}$:

$$\mathbf{P} = \mathbf{I} - \mathbf{M}(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T \tag{4.31}$$

The projection matrix projects any change in $\mathbf{x}$ or $\mathbf{u}$ to the hypersurface of the constraints 4.29, provided that we start from a valid solution that satisfies the constraints [Kirk, 1970]. The projection matrix $\mathbf{P}$ is symmetric, idempotent and positive semidefinite. If we then use the updating rule 4.32, we are guaranteed to converge to a stable local minimum that satisfies the constraints, given a positive definite quadratic cost

and an initial feasible trajectory.

$$\begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \\ \vdots \\ \dot{\mathbf{x}}_N \\ \dot{\mathbf{u}}_0 \\ \dot{\mathbf{u}}_1 \\ \vdots \\ \dot{\mathbf{u}}_{N-1} \end{bmatrix} = -\epsilon \quad \mathbf{P} \quad \begin{bmatrix} \frac{\partial J}{\partial \mathbf{x}_1} \\ \frac{\partial J}{\partial \mathbf{x}_2} \\ \vdots \\ \frac{\partial J}{\partial \mathbf{x}_N} \\ \frac{\partial J}{\partial \mathbf{u}_0} \\ \frac{\partial J}{\partial \mathbf{u}_1} \\ \vdots \\ \frac{\partial J}{\partial \mathbf{u}_{N-1}} \end{bmatrix} \tag{4.32}$$

Proof

Using Lyapunov stability theorem, if there exists an energy function $E(\mathbf{v})$ which is bounded from below, and $dE(\mathbf{v})/dt < 0$, the system of differential equations 4.32 is stable.

Let

$$E(\mathbf{v}) = J(\mathbf{x}, \mathbf{u}) \tag{4.33}$$

$$\frac{dE}{dt} = \frac{\partial E^T}{\partial \mathbf{v}} \frac{d\mathbf{v}}{dt} \tag{4.34}$$

Using equation 4.32, equation 4.34 becomes:

$$\begin{aligned} \frac{dE}{dt} &= -\epsilon \quad \frac{\partial E^T}{\partial \mathbf{v}} \quad \mathbf{P} \quad \frac{\partial E}{\partial \mathbf{v}} \\ &\leq 0 \end{aligned} \tag{4.35}$$

In this derivation we made use of the fact that the projection matrix $\mathbf{P}$ is positive semidefinite.

For quadratic cost functions, we can substitute for the value of $dE/d\mathbf{v}$ in the update equation 4.32. In this case the update rule can be expressed by equation 4.36

$$\dot{\mathbf{v}} = -\epsilon \quad \mathbf{W} \ \mathbf{v} \tag{4.36}$$

where $\mathbf{W}$ is a symmetric positive semidefinite matrix represented by equation 4.37

$$\mathbf{W} = \mathbf{P} \ \mathbf{H}$$

where

$$\mathbf{H} = \begin{bmatrix} Q & & & & \\ & Q & & & \\ & & Q & & \\ & & & R & \\ & & & & R \end{bmatrix} \tag{4.37}$$

Equation 4.37 may be implemented using an analog Hopfield network formed of simple RC elements. The value of the coefficient ϵ together with the eigenvalues of the connectivity matrix $\mathbf{W}$ determine the rate of convergence to the local optima.

Equivalence between gradient projection and penalty methods

It is important to note here that the gradient projection method may be viewed as a modified penalty method with an infinite penalty coefficient, but well-conditioned connectivity matrix. Let us rewrite the energy function of the penalty method (equation 4.20) in terms of the $\mathbf{M}$, $\mathbf{v}$, $\mathbf{H}$ and $\mathbf{c}$ matrices defined above:

$$\mathbf{J}'(\mathbf{x}, \mathbf{u}) = \mathbf{v}^T\mathbf{H}\mathbf{v} + \kappa(\mathbf{M}^T\mathbf{v} - \mathbf{c})^T(\mathbf{M}^T\mathbf{v} - \mathbf{c}) \tag{4.38}$$

A gradient descent update rule for the above energy yields :

$$\frac{d\mathbf{v}}{dt} = -\mathbf{W}\mathbf{v} + \mathbf{M}\mathbf{c} \tag{4.39}$$

where $\mathbf{W} = \mathbf{H} + \kappa\mathbf{M}^T\mathbf{M}$, the matrices $\mathbf{H}$ and $\mathbf{M}$ have the same definitions as above, and κ is the penalty coefficient. Our goal is to modify the update rule in the above equation without changing the equilibrium point. This can be easily done by multiplying the right hand side of the above equation by a nonsingular matrix $\mathbf{F}$:

$$\frac{d\mathbf{v}}{dt} = -\mathbf{F}\mathbf{W}\mathbf{v} + \mathbf{F}\mathbf{M}\mathbf{c} \tag{4.40}$$

The matrix $\mathbf{F}$ which has the same dimension as $\mathbf{W}$ is added to improve the eigenvalue ratio of the linear system of equations 4.40. Since the ill-conditioning is mainly due to the penalty coefficient, one way to reduce this ill-conditioning is by approximately cancelling the effect of κ by using

$$\mathbf{F} = [\mathbf{E} + \kappa\mathbf{M}^T\mathbf{M}]^{-1} \tag{4.41}$$

where $\mathbf{E}$ is a symmetric positive definite matrix. The best value of $\mathbf{E}$ is of course $\mathbf{H}$ which will result in all eigenvalues being equal. Another possibility is to use

$$\mathbf{E} = \mathbf{I} \tag{4.42}$$

where $\mathbf{I}$ is the identity matrix. Using the matrix inversion lemma [Bertsekas, 1982] and taking the limit as $\kappa \rightarrow \infty$, the matrix $\mathbf{F}$ reduces to :

$$\mathbf{F} = \mathbf{I} - \mathbf{M}(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T \tag{4.43}$$

which is the same as the projection matrix $\mathbf{P}$ defined in equation 4.31 above.

Example:

We have tested the gradient projection method for the generation of the optimal control trajectory of an unstable second order system with a quadratic cost. The continuous system **A**, **B**, **Q** and **R** matrices and the initial conditions $\mathbf{x}_o$ are given below:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 2 & -1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{R} = 0.5, \quad \mathbf{x}_o = \begin{bmatrix} -4 \\ 4 \end{bmatrix} \tag{4.44}$$

This example is taken from [Kirk, 1970]. For a time horizon of 6 seconds and a sampling rate of 50 Hz, the number of variables of the equivalent Hopfield network is 900. The output of the Hopfield network after convergence of the network is shown in figure 4-18. The trajectories obtained are similar to the trajectories obtained using an LQR regulator. The convergence behavior of the Hopfield network is shown in figure 4-19 which is a plot of the Lyapunov energy function defined above as a function of time for a value of $\epsilon = 10^5$.

This example shows the feasibility of using the gradient projection method to obtain a constraint satisfying Hopfield neural network which is capable of computing the optimal control trajectory in a small fraction of the sampling time. However, there may be some practical disadvantages for using this technique. First, unlike the penalty method, the connection weight matrix **W** generated using the gradient projection method is in general not sparse. This is due to the fact that the projection matrix **P** is not sparse. This high connectivity makes it more difficult to implement the network in hardware. The second disadvantage is that the connectivity matrix is only positive semidefinite. This is because the projection matrix restricts the trajectory of the network to the hypersurface of the constraints. This may present a problem if the network initial conditions do not satisfy the dynamic constraints. Moreover, any imprecision in the implementation of the connectivity matrix **W** may result in an unstable system. Another disadvantage with this approach is that, al-

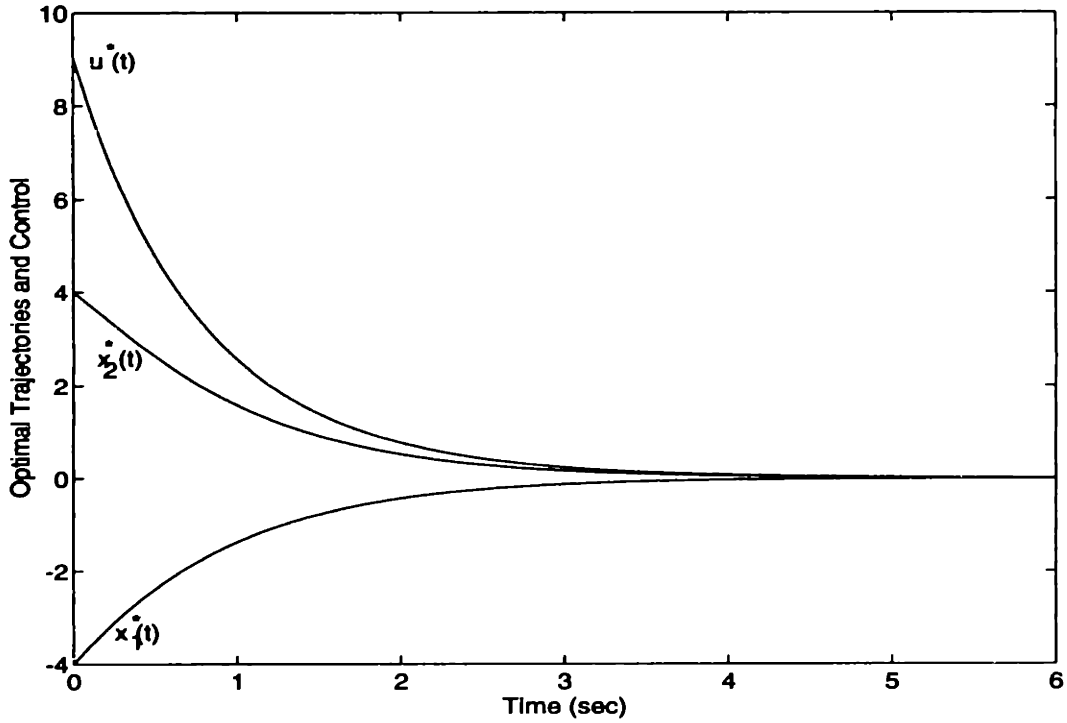

Figure 4-18: Optimal control $u^*(t)$ and state trajectories $x_1{}^*(t)$, $x_2{}^*(t)$ obtained using the gradient projection Hopfield network

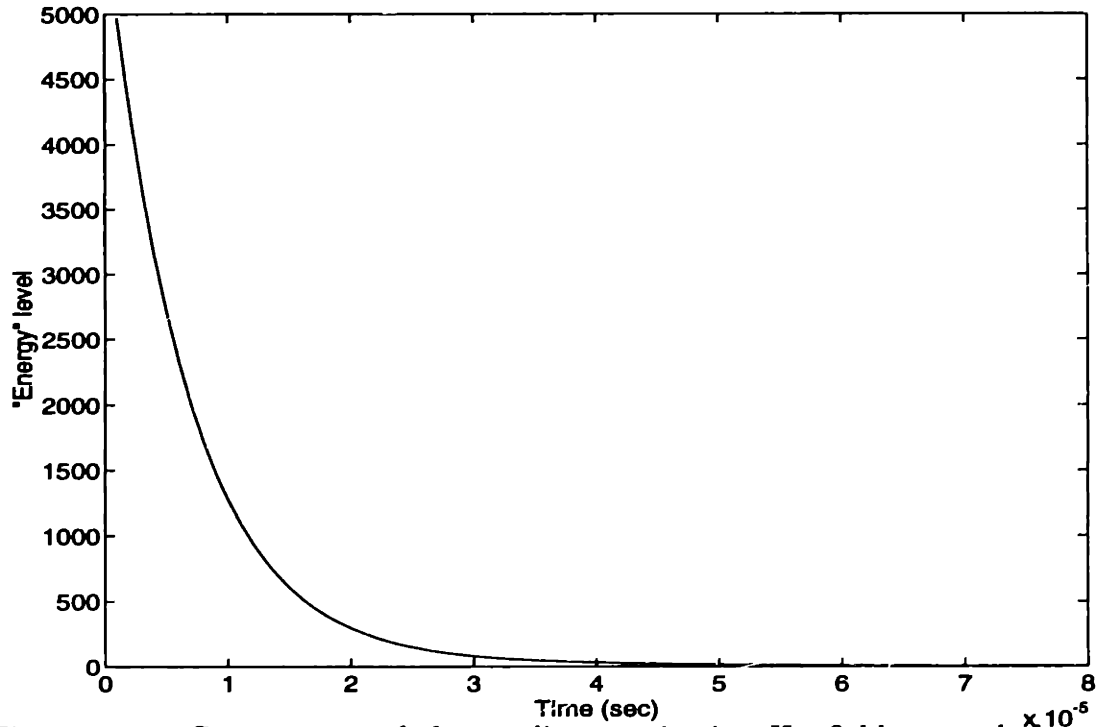

Figure 4-19: Convergence of the gradient projection Hopfield network for a time constant $(1/\epsilon) = 0.01msec$

though the Hopfield network converges very fast to the optimal trajectory, the initial computational time required to find the connectivity matrix $\mathbf{W}$ may be high and consequently cannot be implemented in real time. This is due to the time it takes to invert the sparse block tridiagonal matrix ($\mathbf{M}^T\mathbf{M}$). For example, for the two dimensional example shown above, it takes approximately 35 seconds to find the weight matrix on a moderately used SUN Sparcstation10 computer. It is important to note here that once the weight matrix is computed for given system parameters, it can be updated much faster for small changes in those parameters using small perturbations techniques.

2. Lagrange multipliers Hopfield network

Another approach for mapping a discrete linear quadratic optimal control problem to a Hopfield neural network is through the use of Lagrange multipliers.

If we define

$$\mathbf{L} = \begin{bmatrix} \mathbf{H} & -\mathbf{M} \\ -\mathbf{M}^T & \mathbf{0} \end{bmatrix} \tag{4.45}$$

where the matrices $\mathbf{H}$, $\mathbf{M}$ and $\mathbf{c}$ have the same definitions as before; then the necessary conditions of optimality can be expressed in matrix form as follows :

$$\mathbf{L}\begin{bmatrix} \mathbf{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix} \tag{4.46}$$

where λ is the $Nn \times 1$ vector of the discretized Lagrange multipliers.

The matrix $\mathbf{L}$ is nonsingular if the optimal trajectory is unique. One possible solution for the linear system of equations 4.46 is to construct the energy function

$$\mathbf{E} = \left\{\mathbf{L}\begin{bmatrix} \mathbf{v} \\ \lambda \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix}\right\}^T \left\{\mathbf{L}\begin{bmatrix} \mathbf{v} \\ \lambda \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix}\right\} \tag{4.47}$$

If we then construct the connectivity matrix

$$\mathbf{W} = -\mathbf{L}^T\mathbf{L} \tag{4.48}$$

and use the update rule :

$$\frac{d}{dt}\begin{bmatrix} \mathbf{v} \\ \lambda \end{bmatrix} = \epsilon\,\mathbf{W}\begin{bmatrix} \mathbf{v} \\ \lambda \end{bmatrix} + \mathbf{L}^T\begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix} \tag{4.49}$$

the dynamic system 4.49 will be stable and will converge to the optimal state, control and costate trajectories.

One serious problem with the above approach is that, although all the eigenvalues of the matrix $\mathbf{W}$ are guaranteed to be negative and real, the condition number of the matrix $\mathbf{W}$ is the square of the condition number of the matrix $\mathbf{L}$. This may result in ill-conditioning of the matrix $\mathbf{W}$ and consequently a very slow convergence to the optimum. A better alternative that works well for linear systems with quadratic costs is to use the duality property of the Lagrange function [Bertsekas, 1982], which states that the solution of the constrained minimization problem is equivalent to the minimization over $\mathbf{x}$ and the maximization with respect to λ of the Lagrangian function. We can then use a gradient descent (ascent) algorithm to minimize (maximize) the Lagrangian function with respect to $\mathbf{x}$ (λ) . This is guaranteed to converge for linear systems with quadratic costs, since the Lagrange function is convex with respect to $\mathbf{x}$. For example, we can choose the connectivity matrix $\mathbf{W}$ to be

$$\mathbf{W} = \begin{bmatrix} -\mathbf{H} & \mathbf{M} \\ -\mathbf{M}^T & \mathbf{0} \end{bmatrix} \tag{4.50}$$

In this case the condition number of $\mathbf{W}$ will be similar to that of the matrix $\mathbf{L}$. The eigenvalues of $\mathbf{W}$ will have a negative real part but they are complex in general.

Although the Hopfield network implementation of the Lagrange multipliers method has a bigger size than the network implemented using the gradient projection method, it is much more sparse, which makes it much easier to implement in hardware. Also, since the connectivity matrix $\mathbf{W}$ is directly computed from the system matrices, the computation time required to build the Hopfield network implementing the Lagrange method is very small.

Example :

We used the Hopfield net generated using the Lagrange method described in the previous section to find the optimal control, state and costate trajectories for the unstable second order linear system 4.44 described above, using the same sampling rate and the same time horizon as in the gradient projection method. The time constant $\frac{1}{\epsilon}$ was set to 10^{-6}. The optimal trajectories and the convergence of the network as measured by the norm of the derivatives of the state variables, are shown in figures 4-20 and 4-21 respectively. Comparing with the gradient projection algorithm, the convergence rate here is an order of magnitude slower, but still the network converged in less than 1 msec.

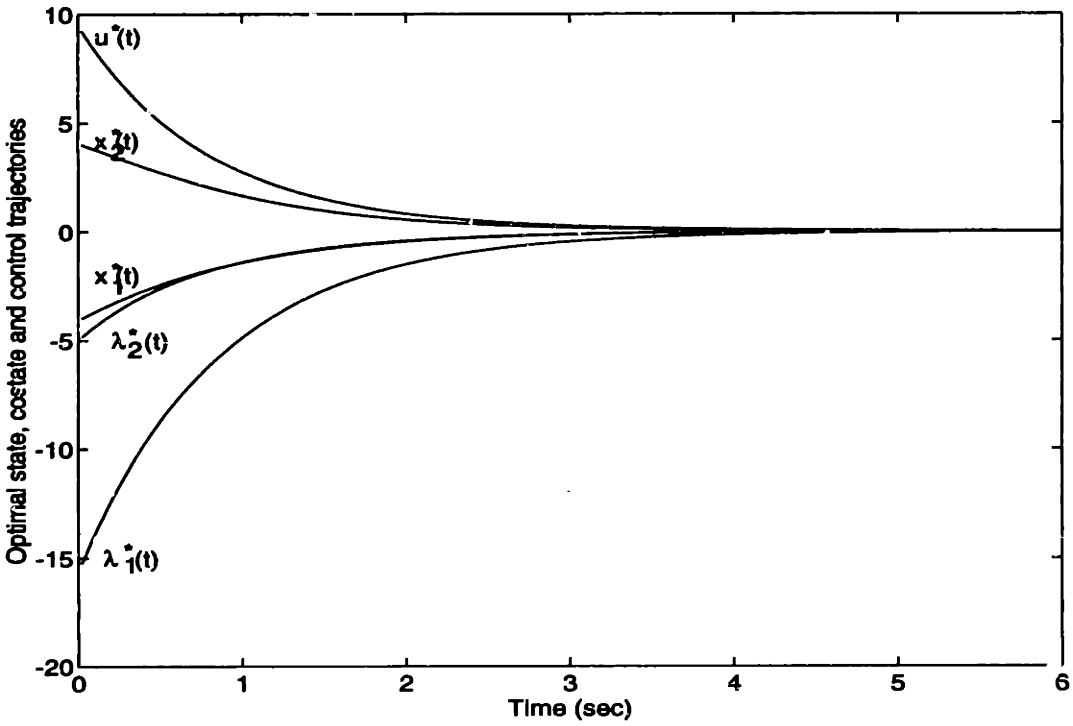

Figure 4-20: Optimal control $u^*(t)$ and state and costate trajectories $x_1{}^*(t)$, $x_2{}^*(t)$, $\lambda_1{}^*(t)$ and $\lambda_2{}^*(t)$ obtained using the Lagrange Hopfield network

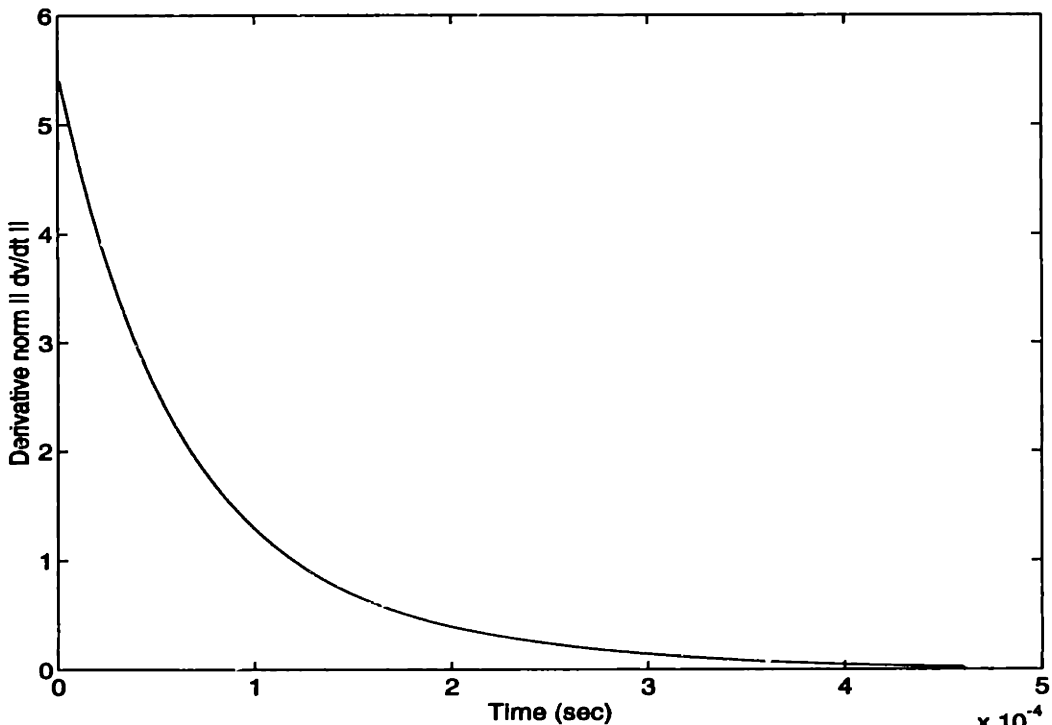

Figure 4-21: Convergence of the Lagrange multiplier Hopfield network as measured by the norm of the time derivatives of the state, for a time constant $(1/\epsilon) = 0.001msec$

Augmented Lagrange multipliers (multiplier methods)

In all the previous analysis of Hopfield implementations of open loop optimal controllers, we have only focused on linear systems with quadratic costs. This represents a serious limitation for the control of motor systems, since these latter are highly nonlinear in general. The linear techniques described above can be extended to slowly varying nonlinear systems, where the system equations can be reasonably linearized for the time horizon at which we compute the optimal control. Moreover, the gradient projection and penalty methods can be modified and extended for nonlinear systems [Scales, 1989]. In that case the resulting connectivity matrix $\mathbf{W}$ of the corresponding Hopfield network is not constant, but a function of the states and controls. The Lagrange multipliers method, as mentioned above is not guaranteed to work for general nonlinear systems, due to its possible non-convexity near the stationary points.

In this section, we will show how to use the augmented Lagrange multipliers method [Bertsekas, 1982; Scales, 1989] to obtain an efficient Hopfield network implementation of optimal controllers that can be applied to general nonlinear dynamic systems. The augmented Lagrange multipliers method, also known as multiplier methods, is a combination of the penalty and the basic Lagrange method described above. It is an exact method, unlike the penalty method and at the same time does not suffer from the excessive ill-conditioning problem that plagues the penalty method. Moreover, it solves the problem of possible non-convexity associated with the basic Lagrange multipliers method. The basic idea behind the augmented multipliers method comes from the observation that the optimization problem

$$\min_{\mathbf{x}} J(\mathbf{x}) = f(\mathbf{x})$$

subject to

$$\mathbf{h}(\mathbf{x}) = \mathbf{0} \tag{4.51}$$

is exactly equivalent to

$$\min_{\mathbf{x}} J(\mathbf{x}) = f(\mathbf{x}) + \kappa \|\mathbf{h}(\mathbf{x})\|^2$$

subject to

$$\mathbf{h}(\mathbf{x}) = \mathbf{0} \tag{4.52}$$

since the penalty term is exactly zero on the constraint surface. We can then form the Lagrange function for the equivalent optimization problem and then use the duality property of the Lagrangian function to compute the optimal variables and Lagrange multipliers. If κ is sufficiently large the Lagrange function will be convex near the local minima. It must be noted here that the penalty term is mainly needed to make the Lagrange function convex near the optimal point and not to enforce the constraints. Therefore a large value of κ may not be required and this will reduce the ill-conditioning problem associated with the penalty method.

Using the augmented Lagrange multiplier method and gradient descent (gradient ascent to compute the optimal Lagrange multipliers), the update equations for the optimal control problem

$$\min_{\mathbf{x},\mathbf{u}} \sum_{k=0}^{N} \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \tag{4.53}$$

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \qquad k = 1, \cdots, N-1 \tag{4.54}$$

are as follows

$$\begin{bmatrix} \frac{d\mathbf{v}_k}{dt} \\ \frac{d\lambda_k}{dt} \end{bmatrix} = -\epsilon \begin{bmatrix} \mathbf{H} & \mathbf{M}(\mathbf{v}) \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \lambda \end{bmatrix} - \begin{bmatrix} \kappa \mathbf{M}(\mathbf{v}) \\ \mathbf{I} \end{bmatrix} \mathbf{h}(\mathbf{v}) \tag{4.55}$$

where

$$\mathbf{M}(\mathbf{v}) = \begin{bmatrix} \mathbf{I} & -\mathbf{A}(\mathbf{v}_1)^T & & & & \\ & & \mathbf{I} & -\mathbf{A}(\mathbf{v}_j)^T & & \\ & & & \mathbf{I} & -\mathbf{A}(\mathbf{v}_{n-1})^T & \\ & & & & \mathbf{I} & \\ -\mathbf{B}(\mathbf{v}_0)^T & & & & & \\ & & -\mathbf{B}(\mathbf{v}_j)^T & & & \\ & & & & -\mathbf{B}(\mathbf{v}_{N-1})^T & \end{bmatrix} \tag{4.56}$$

and

$$\mathbf{A}(\mathbf{v}_j) = \nabla_{x_j}\mathbf{f}(\mathbf{x}_j, \mathbf{u}_j) \tag{4.57}$$

$$\mathbf{B}(\mathbf{v}_j) = \nabla_{u_j}\mathbf{f}(\mathbf{x}_j, \mathbf{u}_j) \tag{4.58}$$

Figure 4-22 shows a network that performs the above optimization. Note that many of the weights in this network are nonlinear and depend on the forward model of the dynamics and its derivatives with respect to the controls and states at each instant of time. This configuration is very similar to Kawato's cascade network for finding the minimum torque change trajectory [Kawato et al., 1990], however the proposed network is a more general one and can be used with any optimal control problem.

We tested the above network in the simulated two-joint tracking of a circle described above. The results of the optimization are as shown in figures 4-23 and 4-24. Figure 4-23 shows the two joint angle trajectories superimposed on the reference trajectory after the network optimization. It is clear from the figure that some residual error remains. This is due to the fact that the optimization network implements a gradient descent algorithm which is very slow to converge to the optimal values. Figure 4-24 shows the convergence of the network as a function of time, as measured by the norm of the gradient.

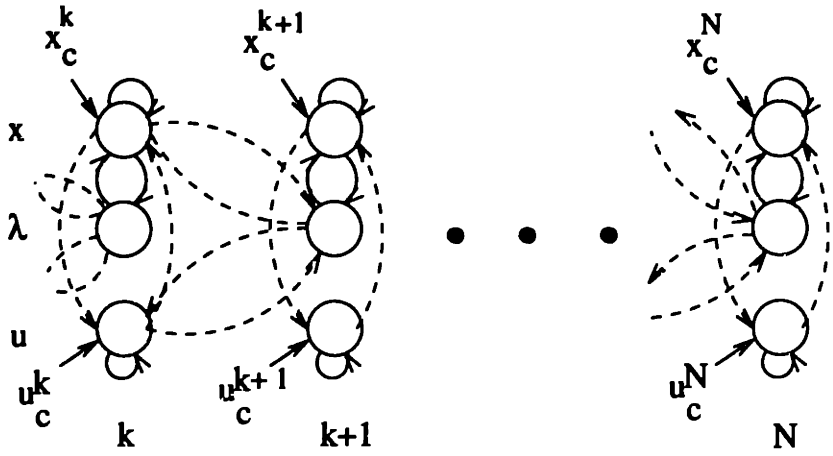

Figure 4-22: Nonlinear optimal control network. x_k, λ_k and u_k are the states, costates and controls at time step k. Dashed lines represent nonlinear weights that are computed using the learned forward model and its partial derivatives with respect to x_k and u_k. Solid lines represent linear weight values. ${x_c}^k$ and ${u_c}^k$ represent any constraint on the values of x_k or u_k.

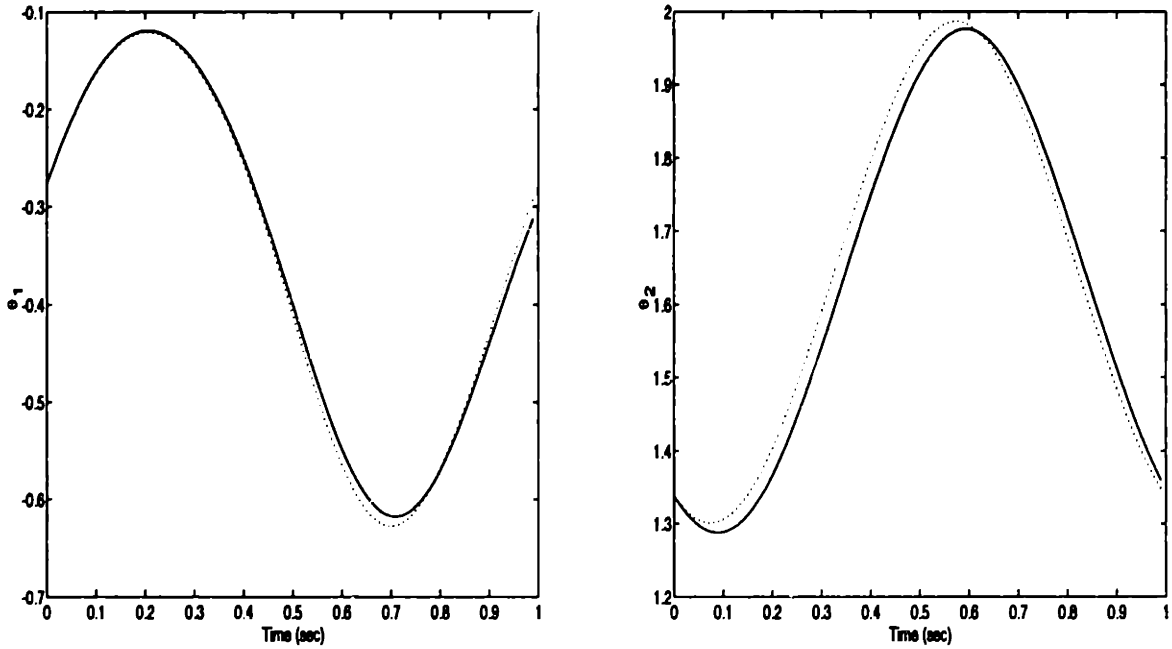

Figure 4-23: Trajectories of the joint angles θ_1 and θ_2 obtained using a dynamic neural network implementing augmented Lagrange multipliers (solid lines) superimposed on the reference trajectories (dotted lines)

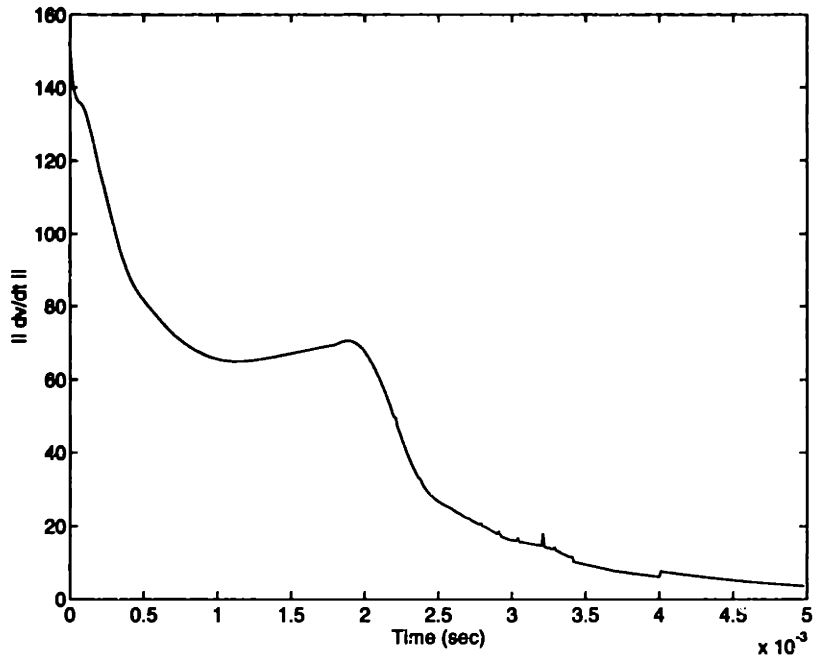

Figure 4-24: Convergence of the network as measured by the norm of the gradient of all the units. The time constant $(1/\epsilon)$ used in this simulation is 0.01 msec.

4.7 Finding Control Trajectories For Difficult Control Problems

There are many control problems where the constraints on the task makes the control problem hard. For example, limits on the degrees of freedom in the control space will make the search for the optimal control harder. Examples of such control problems include the control of a two link arm from an initial to a final position using only one torque generator at one of the joints or the control of a unicycle. Another example is the control of a pendulum on a cart using only force on the cart. To lift the pendulum to a vertical position by controlling the movement of the cart is a hard control problem [Slotine and Li, 1991]. Similar control problems also arise in human motor control when the control variables do not directly control the desired states, but these latter are controlled through nonlinear couplings with other states. An example is the gymnast on a horizontal bar performing a giant swing by using torso and hip muscles. Other than the active torques that the gymnast exerts using the lower body muscles, s/he performs a sequence of weight shifts that allow him/her to perform the swing. For example, s/he extends the body during the downswing phase and flexes the body during the upswing phase to increase the acceleration and decrease the deceleration respectively. All this is done within the limits of acceptable form of course [Jensen and Schultz, 1977].

In this section, I will explore, using simulations, the possibility of applying learning optimal control technique described above to learn to lift a two link arm as high as possible from a vertically down position, using only one torque generator at the shoulder. This can be thought of as a simplified model of the swinging action of the gymnast described above. There are also constraints on the maximum value of the torque exerted and on the range of joint displacements. The joint angle at the shoulder is restricted from zero (fully extended) to π (fully flexed). The same two-link model used to generate the minimum torque trajectory is used here, with the

exception that the torque at the shoulder is set to zero. The cost function that we want to optimize is described by equation 4.59.

$$J = K \ (\theta_1^o - \theta_1(t_f)) + \int_0^{t_f} (k_1 \tau(t)^2 + k_2(\theta_2(t) - L)^2)dt \qquad (4.59)$$

where K and k_1 are positive constants, k_2 is equal to zero if $\theta_2(t)$ is within the allowed range of θ_2 and a large positive number otherwise.

We used Gaussian RBF networks to represent the forward dynamics model. We used 20 initial sinusoidal torque sequences with different amplitude and phase to obtain an initial forward model of the system.

Using the cost function above and the true model equation, the best trajectory that we could obtain is shown in figure 4-25. This figure shows that it was possible to get the upper arm just above the horizontal using the cost function 4.59, without exceeding the joint limits on θ_2. Shown also in figure 4-25 is the best performance that could be obtained using the open loop learning optimal control method described above. This performance is much worse than the best performance that could be obtained using the real model. The discrepancy in performance is due mainly to the difficulty in identifying the system. The problem with the identification is that most of the state space is very difficult to reach with random inputs, and therefore the identification is poor. We could not overcome this identification problem using different types of torque sequences.

4.8 Relation to other methods

There has been previous research that addressed the problem of learning the optimal actions, especially as related to the problem of solving the excess degrees of freedom problem in the field of motor control. Jordan [Jordan, 1989] has developed a related approach to learn and optimize motor actions in the presence of excess degrees of freedom. His model is formed of two subnetworks, a forward model and a controller.

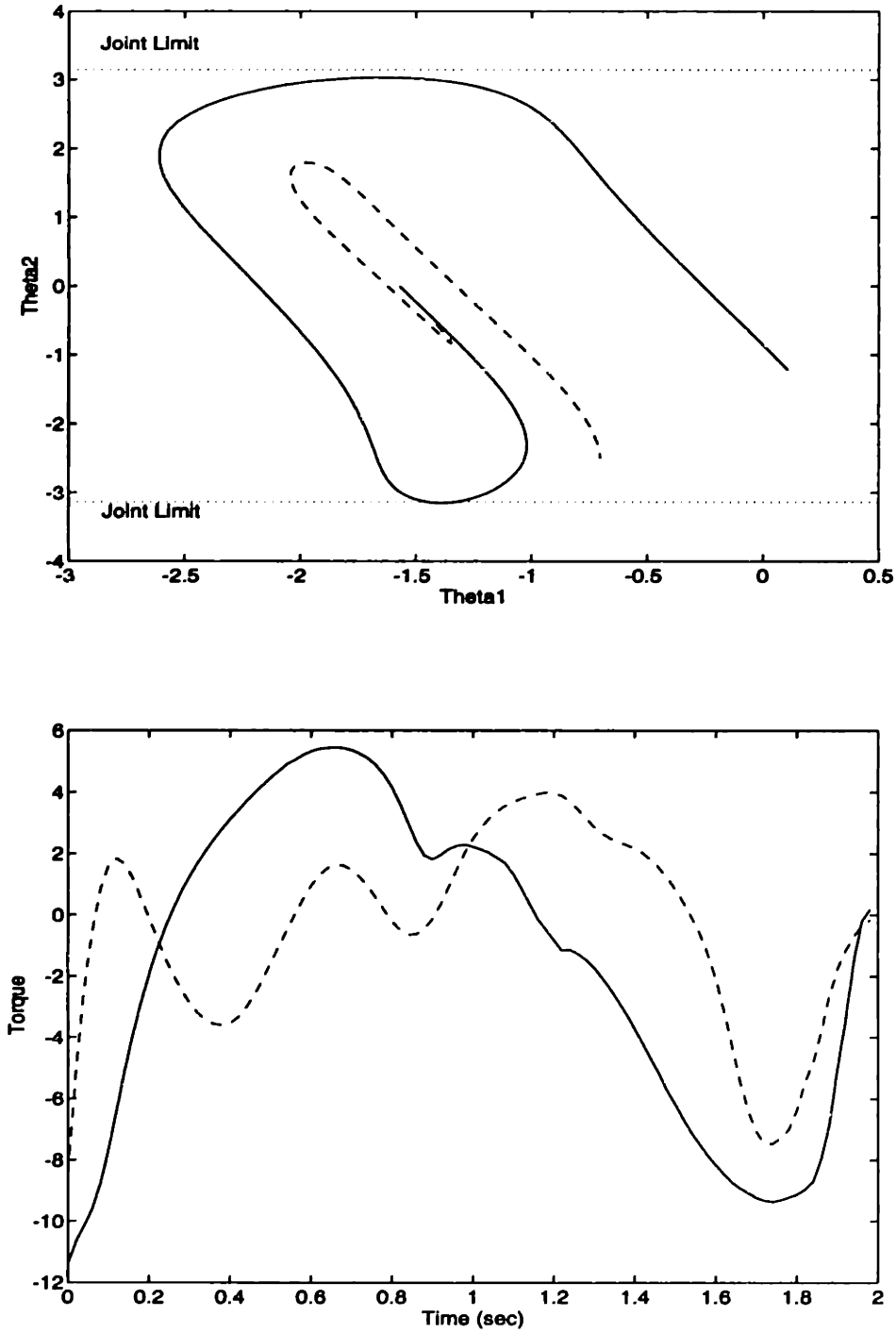

Figure 4-25: Phase plot of the computed optimal trajectory (upper plot), and the computed optimal torque as a function of time (lower plot). The solid lines represent the trajectory computed using the ideal model and the dashed lines represent the trajectory obtained using the approximated forward dynamics.

The forward model transforms the control actions ("articulatory space") to task outputs at any given state. The controller generates the optimal actions as a function of the states. This is similar in structure to our closed loop approach using the extremal fields. The difference between both approaches is in the learning of the control law ("motor program"). While Jordan method attempts to learn the "motor program" directly as a function of the intrinsic and external states, by backpropagating the errors through the forward model, we try in our approach to first generate the open loop optimal control and then learn the closed loop control as a function of the states and time. Kawato et al. [Kawato et al. 1990] developed a cascade neural network model that is specifically designed to generate the minimum torque change arm trajectories proposed by Uno, Kawato and Suzuki [Uno et al., 1989]. Their model also employs a forward model of the dynamics. However, this model is reproduced at each time step. The minimum torque change is realized by adding resistive connections between the units representing the torques at each time step. The minimum torque change torques are then obtained by backpropagating the error through this cascade structure. This method of trajectory generation is very similar to our trajectory optimization method, and can be shown to be exactly equivalent to first order gradient dynamic optimization methods. More recently, the same group [Wada et al., 1992] suggested another method to find the optimal trajectory using both a forward and an inverse model. This method is more general than the cascade network and can be applied to other optimal control problems. Unlike methods based on the calculus of variations, this method does not require the backpropagation in time of the error or the reinforcement signal, but requires the learning of both an inverse and a forward model.

Many other researchers have also suggested the use of function approximation and neural networks to approximate the optimal feedback law by modeling the optimal control as a function of the states [Peterson, 1992; Goh, 1993]. In both of these papers, the optimal control was computed using the dynamic model of the system. In Goh's

work, the goal was to synthesize a stable optimal controller for nonlinear regulator problems assuming full knowledge of the dynamics. The conditions of optimality were used to train the neural networks. Two sets of neural networks were used, one set to represent the optimal feedback controllers, and the other to represent components of the return function. In Peterson's paper the optimal open loop control sequence obtained by solving the equations resulting from the necessary conditions of optimality at a given initial state is used to train a neural network. This approach is similar to one of the approaches discussed in this chapter.

Unconstrained and constrained optimization using dynamic neural networks, such as Hopfield networks, have been proposed by many researchers (e.g. [Hopfield, 1984; Platt and Barr, 1988]). In particular, Platt and Barr propose using Lagrange multipliers methods for constrained optimization. This approach to solve linear and nonlinear programming problems, together with proofs of convergence, has also been known since at least the late 1950's [Arrow, 1959], although the solution was not given in the context of dynamic neural networks. Also, many researchers have proposed to solve optimal control problems using Hopfield Networks [Lan and Chand, 1990; Mears et al, 1993]. In these papers, however, the problem was treated as an unconstrained optimization problem with the constraints incorporated as an extra quadratic penalty. As pointed out previously, these approach does not work well in practice, due to the very large number of constraints that have to be satisfied.

The problem of learning a desired trajectory using iterative techniques has also been addressed by many researchers. A review of these techniques is presented in [Atkeson, 1986]. The main disadvantage of these techniques is the requirement to return to the same initial conditions after each iteration. As pointed out in [Atkeson et al., 1988], iterative techniques which rely on a model will lead to faster convergence and correction of the errors.

Chapter 5

Optimization in Stochastic Environments

5.1 Introduction

In many real life motor control problems, the result of an action applied by an agent is not completely deterministic. Unlike deterministic systems, the result of one trial is embedded in noise and may not yield much information. To assess the success or failure of an action, the action is applied several times and then we compute some statistics about its degree of success or failure to achieve the desired goal. Several factors may contribute to this randomness. One such factor is that not all the relevant components of states and the actions applied to a system are measured. Sometimes they are not even observable. Another possible source for this non deterministic response could be intrinsic variability and noise within the system. This is obviously true in humans and living systems, where there is a large variability in actions from trial to trial, even though the goal is the same. Such a system is modeled in figure 5-1.

Another problem where the result of actions may appear non deterministic is the case when the result of an action is delayed and depends also on the future actions and states of the system which could be unpredictable. For example for games like chess,

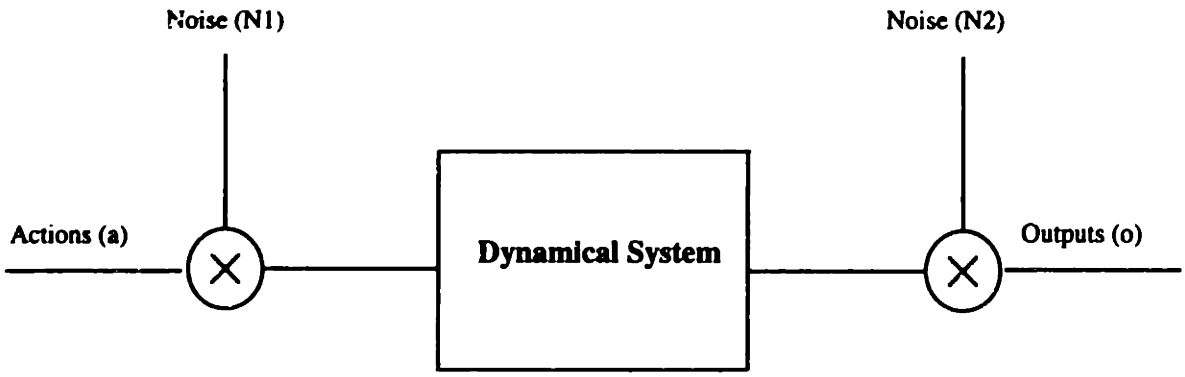

Figure 5-1: A general system with stochastic response

the effect of an action is not immediate and depends also on the actions taken by the opponent which are unpredictable. However in this case the action could be made deterministic if we include all the states and actions which affect the output. From the above examples, it is important to note that the description of an environment as stochastic or deterministic may not necessarily be an absolute property of the environment but could depend also on the ability of the learner to measure the relevant states and actions. Therefore an environment that one learner may view as stochastic may be viewed by another learner as deterministic. However, for practical reasons, it may be cheaper to consider an environment as stochastic, if the price to be paid to achieve determinism is higher than the gain in accuracy.

In this chapter, we would like to explore different methods for optimizing actions that are suitable for systems with non-deterministic response. Some of these methods could also be used for deterministic environments as well, however there may be more efficient methods that are more suited for deterministic systems. After reviewing different techniques for finding optimal actions in stochastic systems, we will describe in more detail a particular method to find the optimal values by first building a forward model of the system and/or a model of an evaluation function that estimates the expected cost or utility of an action, and then finding optimal actions. By a

forward model we mean a model of the parameters that describe the statistics of the output as a function of the different actions and states. An example of such a model could be a model of the probabilities of success of different actions or a model of the means and variances of the outputs observed when applying different actions. The question that we would like to address in this chapter is can we find an action that is optimal with respect to the goal of the task given the variability in the system response?

Different optimization objectives include finding the actions that maximize the probability of success or maximize the expected return. It must be emphasized that, for non-deterministic systems, it is very hard to locate the absolute optimal action given only a small number of experiences. This is due to the fact that it requires a large number of actions to differentiate between the ranking of two actions, if their probabilities for success are very close. In our examples, we view success as a function (usually assumed to be binary) of the outcome which could be a continuous stochastic variable. We will consider both the cases when the output is binary (for example 1 represents the success of the trial and 0 for failure) and when the output is a continuous random variable such as a reinforcement variable. Sometimes the success or failure is a function of another intermediate output. For example success could be defined by the following relation.

$$t(o) = \begin{cases} 1 & \alpha \leq o \leq \beta \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Where $t(o)$ is the success function, o is the outcome, and α and β are constants that may be known or unknown. Mathematically we can express the problem as the following optimization problem :

$$a^* = \max_{a \in A} P(t(o(a,s)) = 1|a) \tag{5.2}$$

where a^* represents the optimal action, o is the outcome, s the state, and A is the set of all admissible actions. In general, the functions $t(\cdot)$ and $o(\cdot)$ are not known a priori. If the outcome o is unobservable or does not exist, we may express $t(\cdot)$ directly as a function of a and s. The distinction between t and o is important if we can measure both t and o and if we can model the transformation $t(o)$ and if $t(o)$ is binary. In that case, the knowledge of o provides more information such as the distance from success or the direction of smaller cost. However in many applications, the only information available to the learner is success or failure.

The problem is to find an optimal action a^* in the most efficient way, i.e. in the smallest number of trials. In the next section we will review different methods for obtaining a^* and we will summarize the advantages and disadvantages of each method. We will then present simulation results for finding the optimal actions for two simple motor control problems. The first example is a basketball aiming problem, where the task is to find the optimal shooting angle and speed that maximize the probability of success given uncertainties in the actions. The second simulation example is the periodic hitting of a ball with a racquet to generate a periodic trajectory of the ball with a given maximum height. The task is to find the best periodic open loop trajectory of the racquet to keep the ball in the air given different external perturbations on the trajectory of the ball.

5.2 Methods for Finding the Optimal Actions

In general, we can divide the algorithms that could be used to find the optimal actions into two main classes: direct and indirect optimization methods. The direct methods do not require forming any model of the response of the system as a function of its states and inputs. These algorithms directly attempt to change the inputs, or a controller parameters, based on previous responses. The actions "settle" into the optimal action with practice. These algorithms may converge to a global or a

local optimum depending on the algorithm used or even the learning parameters of these algorithms. This will be demonstrated by an example later. Techniques such as learning automata [Narendra and Thathachar, 1989], reinforcement learning and direct optimization over the input variables fall under this class and will be described below.

The indirect techniques attempt to model the statistics of the output, or sometimes model a cost function, given the inputs and states, and then find the actions that maximize the desired objective function, for example the probability of success as defined in the equation for a^* above.

5.2.1 Direct Optimization Techniques

General nonlinear function optimization algorithms could be used to determine the optimal action. However, function optimization techniques which rely on the gradient to determine the next action, are not well suited for optimizing non-deterministic functions because the gradient information is erroneous and not easily available, thus we have to perform many function evaluations at neighboring points in the action space to obtain more reliable information. Another problem may be that we can not get a gradient at all, if all the information that is available is a function of the output $t(o)$ which could be binary or not continuous, for example success or failure. In addition, these methods do not necessarily find the best action and may only find a locally optimal one. However, these algorithms could still be used but in a less greedy fashion, for example with a small learning rate and this learning rate should diminish with more examples in order for the algorithms to converge. The other class of function optimization techniques relies on some variation of a random search. I will describe two different examples of such function optimization techniques and discuss the advantages and disadvantages of each and propose methods to improve them.

Learning Automata and Reinforcement Learning

The basic idea behind learning automata and reinforcement learning is that we start with an a priori probability distribution for choosing an action, and then we increase or decrease the probability of choosing that action in the future based on whether this action was successful or not in the past. This is also motivated by human and animal learning experiments where a reward or penalty increases or decreases respectively the choice of an action in the future. It has its roots in psychology with the work of Thorndike and the "Law of Effect" [Thorndike, 1927]. Learning automata are also closely related to the work on bandit problems in operations research. There are many algorithms for reinforcement learning which differ in the rule by which the probability of choosing an action is modified. The basic version of these algorithms is indifferent to how close the outcome is to success. The information that is used to update the probabilities of choosing an action is either success or failure. However, some versions allow a real number, between [0, 1] for example, as a reinforcement signal. Many of the algorithms are theoretically proven to converge to the action with the highest probability of success as the number of trials $n \rightarrow \infty$. For more information about the different algorithms for reinforcement learning and learning automata refer to the book by Narendra and Thathachar [Narendra and Thathachar, 1989]. Reinforcement learning techniques were originally designed for the case when the number of actions is finite and small. For a continuous action space, we can discretize the space. However, if the number of possible actions is large, the learning rate should be made very small to guarantee convergence in practice. In this case the convergence rate will be very slow. Millington addressed this problem [Millington, 1991] and developed a reinforcement learning algorithm which could be used for continuous variables. Another disadvantage of this approach is that it does not take advantage of the closeness of an experience to success. To take advantage of the continuity of the outcome as a function of the actions taken (i.e neighboring actions have similar outcomes) a modification to the basic algorithm could be made where the outcome of

an action not only modifies the probability of choosing that action, but also modifies the probability of choosing neighboring actions as well.

Gullapalli [Gullapalli, 1990] described another method for optimizing real valued functions directly using an associative reinforcement scheme [Barto and Anandan, 1985], by choosing the action from a probability distribution whose parameters are adjusted according to the reinforcement received and its relation to the expected reinforcement. Gullapalli's method could be easily applied to find the optimal action given a certain state of the environment (associative reinforcement learning tasks). However, it becomes more difficult to use when there is more than one action parameter to optimize since the different optimal parameters are interdependent, and therefore we expect that this method will require a large amount of training. Another case where this method cannot be easily used is when there are constraints on a subset of the actions that should be satisfied by the optimal actions in addition to their instantaneous utility. For example in the case of planning a trajectory with desired end points, in addition to the instantaneous utility at each instant of the trajectory (for example the instantaneous cost could be a smoothness cost), we have some end point constraints that have to be satisfied. These final constraints are the integration of a function of a subset of actions. One method to solve this problem is to translate the end point constraints into instantaneous cost or utility, using for example the method of temporal differencing (TD) developed by Sutton [Sutton, 88]. Another method to propagate the constraints would be through the help of a model of the dynamics by integrating backward in time. Gullapalli demonstrated the use of these stochastic reinforcement networks in simple problems including the control of a robot arm to a final position [Gullapalli, 1990].

Depending on the learning rate, and the algorithm used, it may be possible to find the best action. For very small learning rates, many actions are wasted in exploration, but the chance of finding the best action is higher. If the learning rate is very high, the algorithm will increase rapidly the probability of applying an action which was

successful and will apply this action more often at the expense of searching for a better action. This is one example of the identification/control conflict.

Metropolis Methods with Simulated Annealing

The basic idea behind this method is to first choose some initial action and then choose the next action based on some probabilistic rule. If the new action is better, accept it with probability 1, and if the outcome of the new action is worse, accept it with a probability that depends on how much worse the outcome was. The parameter which determines the variance of this probability distribution, also called the temperature, is decreased as the number of trials increase. This technique, also called simulated annealing, has been used in many optimization problems with good results. However, it is computationally expensive and requires a large number of function evaluations. Another disadvantage of this algorithm is that, unlike the reinforcement learning, the actions chosen do not depend much on the probabilities of their previous successes, therefore many trials could be wasted looking for a better action. This algorithm requires the knowledge of the closeness of the action from success or how successful it was, unlike reinforcement methods. This technique can not be used in this form for associative reinforcement tasks [Barto and Anandan, 1985] where the best actions are functions of the states of the environment as is the case in many motor control tasks. Nevertheless this technique is very valuable for optimization of functions.

5.2.2 Indirect Methods

Indirect methods usually involve the formation of some stochastic model of the input output data and/or the utility of the instantaneous actions. After the formation of an approximate model, a search is performed to find the inputs which optimize the desired performance index such as the probability of success, for example. One difference between this class of methods and direct methods is that direct methods explicitly solve the exploration part. The choice of actions is determined directly by

the algorithm used. This is not the case for the indirect methods. In this class of methods, actions chosen must allow for better identification of the system in addition to being optimal. In order to build a good approximation of the stochastic system, we need to collect a lot of examples for each action many of them could be unsuccessful trials. This is not very desirable, especially when mistakes are expensive. Efficient methods should use the knowledge collected so far to compromise between exploratory and optimal actions. In a previous chapter, we have proposed and implemented a method to guide exploration for deterministic systems which makes use of previous experiences. This approach can also be applied here. There is however an important difference between exploration in deterministic and stochastic systems. There is no gain in information for trying the same action in deterministic systems, while trying the same action, or nearby actions, in stochastic systems improves the estimation of the statistics of the system. To reduce the number of trials needed, we have to take advantage of the continuity of the outcome (or success) statistics as a function of the different actions, assuming of course that such continuity exists.

Approximating the Probability of Success or Failure

For simplicity let us assume for the time being that the environment is context free. Then we can estimate the probabilities of success of the different actions $P(s|a)$ given previous experiences of success or failure of the different actions. Assuming that this probability function is continuous as a function of the actions (i.e. neighboring actions will have similar probabilities of success), we can use the information at nearby actions to estimate the probabilities of success for a given action value. A more efficient way is to use a function approximation technique such as radial basis functions, together with a suitable criterion such as maximum likelihood to estimate the probability of success. Such a method to compute the probabilities of success of the different actions is equivalent to the soft classification techniques described in detail by Wahba [Wahba, 1993], where a regularization factor is added to the maximum likelihood objective

function and cross-validation is used to estimate the weighting of the regularization factor. We will give a simple 2-D example of the use of this technique, using HyperBFs to estimate the probabilities of success. However, for simplicity we will not use a regularization factor. In addition, we believe that such regularization factors may sometimes have a small added value at a computationally very high cost, especially in cases where the function to be estimated depart significantly from the regularization assumptions. The regularization factor obtained in these cases using cross validation will be very small or zero. However, as pointed out by Wahba, without a regularization factor, the minimization of the negative log likelihood objective function will tend to push the estimation of $\hat{P}(\mathbf{a})$ to 1.0 or -1.0. In order to circumvent that problem, the number of free parameters should be kept low compared to the number of data points. Regularization will still be very important when the number of experiences is still small as is the case in the first few iterations.

We will briefly describe here how to use HyperBFs to estimate the probabilities of success of different actions by modeling different monotonic functions of the probability of success. For a more detailed description of the different statistical methods to model binary data refer to the book by Cox and Snell [Cox and Snell, 1989] and the paper by Wahba [Wahba, 1993].

Let us define the logit function $f(\mathbf{a}) = \ln \frac{P(\mathbf{a})}{(1-P(\mathbf{a}))}$ where $P(\mathbf{a})$ represent the probability of success given an action vector $\mathbf{a}$. We can then estimate the logit function $f(\mathbf{a})$ using a HyperBF network of Gaussian units, for example, and previous experiences.

$$\hat{f}(\mathbf{a}) = \sum_{i=1}^{\#centers} C_i \exp(\|\mathbf{a} - \mathbf{t}_i\|_W) \tag{5.3}$$

where $\|\mathbf{a} - \mathbf{t}_i\|_W$ represent the generalized norm : $(\mathbf{a} - \mathbf{t})^T W^T W (\mathbf{a} - \mathbf{t})$, and $\mathbf{t}_i$ represent the locations of the centers of the Gaussian functions. The logit function is used instead of the direct probabilities because it transforms the probability values to $(-\infty, \infty)$, which is more suitable for approximation by neural networks. In addition,

the use of the logit function, amplifies the error for small and large probabilities, which results in better estimation of both low and high probabilities. However, the use of the logit has the disadvantage that the function becomes much less smooth which makes it harder to approximate. In addition regularization is needed to prevent very high function variation.

If we define the Likelihood function of the data to be the probability of occurrence of the data (assuming independence) given a probability model, then the likelihood function L can be mathematically represented as follows :

$$\mathcal{L} = \Pi_{i=1}^{\#data} P(\mathbf{a}_i)^{s_i}(1 - P(\mathbf{a}_i))^{1-s_i} \tag{5.4}$$

where s_i is either 1 or 0 depending on the success or failure of action a_i respectively. Maximizing $\mathcal{L}$ is equivalent to minimizing the negative of the loglikelihood function. The objective function to be minimized could then be expressed as a function of the logit function $f(\mathbf{a})$.

$$J = \sum_{i=1}^{\#data} \ln(1 + \exp(\hat{f}(\mathbf{a}_i))) - s_i \hat{f}(\mathbf{a}_i) \tag{5.5}$$

where $\hat{f}(\mathbf{a}_i)$ is the HyperBF approximation of the logit function. We can use any nonlinear optimization method, such as gradient descent for example, to estimate the values of the parameters C_i and the W matrix. Assuming a diagonal W, the parameters are updated as follows :

$$\begin{aligned}
\Delta C_i &= -\epsilon \frac{\partial J}{\partial C_i} \qquad i = 1, \cdots, \#centers \\
\Delta w_{dd} &= -\epsilon \frac{\partial J}{\partial w_{dd}} \qquad dd = 1, \cdots, \#dimensions \\
\frac{\partial J}{\partial C_i} &= \sum_{j=1}^{\#data} \left(\frac{\exp(\hat{f}(\mathbf{a}_j))}{1 + \exp(\hat{f}(\mathbf{a}_j))} - s_j\right) \exp(\|\mathbf{a}_j - \mathbf{t}_i\|_W) \\
\frac{\partial J}{\partial w_{dd}} &= \sum_{j=1}^{\#data} \left(\frac{\exp(\hat{f}(\mathbf{a}_j))}{1 + \exp(\hat{f}(\mathbf{a}_j))} - s_j\right) \sum_{i=1}^{\#centers} -2w_{dd}(a_{j_{dd}} - t_{i_{dd}})^2 C_i \exp((\|\mathbf{a}_j - \mathbf{t}_i\|_W)
\end{aligned}$$

(5.6)

The factor $\frac{\exp(\hat{f}(\mathbf{a}_j))}{1+\exp(\hat{f}(\mathbf{a}_j))}$ in equations 5.6 is the famous logistic function and is equal to $\hat{P}(\mathbf{a}_j)$, the estimated probability of success of the action vector $\mathbf{a}_j$. The interpretation of equations 5.6 is very simple: if the current action results in a success, change the parameters of the RBFs in a direction so as to increase the probability of success, with each parameter increased proportional to its sensitivity of the probability to that parameter. On the other hand, if the action vector results in a failure, change the parameters in the opposite direction to reduce the estimated probability of success.

It is important to note that we could have obtained similar results without the use of the logistic function, by simply using least squares regression directly on the binary observations. If the variances are all the same as a function of the action space, this will result in finding the expected values of the output which is in this case equivalent to the probabilities of success. Unfortunately, the variances are not all the same and are related to the probabilities of success as a function of actions by the equation 5.7.

$$Var(o(a)) = P(a)\,(1 - P(a)) \tag{5.7}$$

However, if the probabilities of success as a function of the action space are restricted for example within the range $0.2 \leq P(a) \leq 0.8$, the variation of the variance will be small and consequently the loss of accuracy will be negligible [Cox and Snell, 1989]. In addition, even if the probabilities are outside this range, we can use iterative techniques and weighted least squares. We first obtain $\hat{P}(a)$ and then estimate the variance using equation 5.7, and weight the squared error values by the inverse of the estimated variance. The use of the logistic function automatically includes this scaling by giving more weight to actions of both low and high probabilities.

The following simple example illustrates the use of HyperBFs to estimate the probabilities of success both using logistic functions and also directly from the data using the iterative method to estimate the variance and then using weighted least

squares.

Example 1 :

The action vector a used here is 2-D and is in the range $[-2, 2]^2$. The probability of success is assumed to be represented by equation 5.8.

$$P(a_1, a_2) = \frac{1}{2}(\cos {a_1}^2 + \sin {a_2}^2) \tag{5.8}$$

This ensures that the probabilities of success at any point is in the range $[0, 1]$. A 3-D plot of the probability of success as a function of the action space is shown in figure 5-2. Each training example is given by the triplet (a_{1i}, a_{2i}, s_i). Samples of training examples are : $(-1.124, -1.812, 0)$, $(-1.969, -0.466, 1)$, $(-0.33, 0.747, 1)$, ... We used 1000 training examples uniformly distributed in the input space. The distribution of the data is shown in figure 5-3. A success is shown by a '1' and a failure is represented by an 'o' symbol. 50 Gaussian Basis Functions whose centers were randomly and uniformly distributed over the space were used to estimate either the logit function of the probability of success from which this probability is then derived, or the probability of success directly in the case of the weighted least squares method. The same second order optimization technique is used to find the parameters of the HyperBF network for both methods. The estimated probability of success for both methods is shown in figure 5-4. As shown in figure 5-4, the estimated probability of success function approximates the ideal one, although the peaks are not perfect. The direct iterative scheme converges after one iteration only and gives better results. The normalized error between the estimated function and the ideal one defined by equation 5.9 is equal to 0.161 for the method using the logistic function and 0.0959 for the iterative direct method.

$$E = \sqrt{\int_{\mathcal{A}} \frac{(P(\mathbf{a}) - \hat{P}(\mathbf{a}))^2}{P(\mathbf{a})^2}} \tag{5.9}$$

where $\mathcal{A}$ is the domain of the action set.

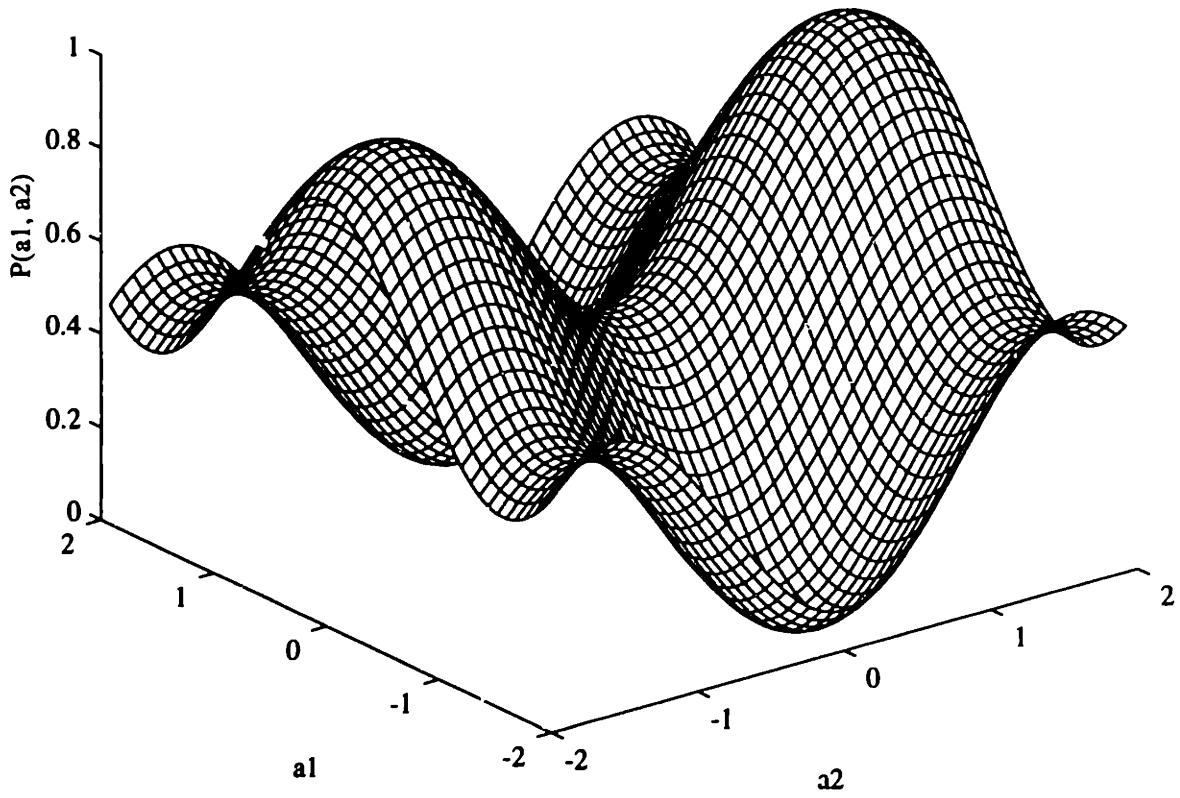

Figure 5-2: Probability of success as a function of the action space (a_1, a_2) for the model of example 1.

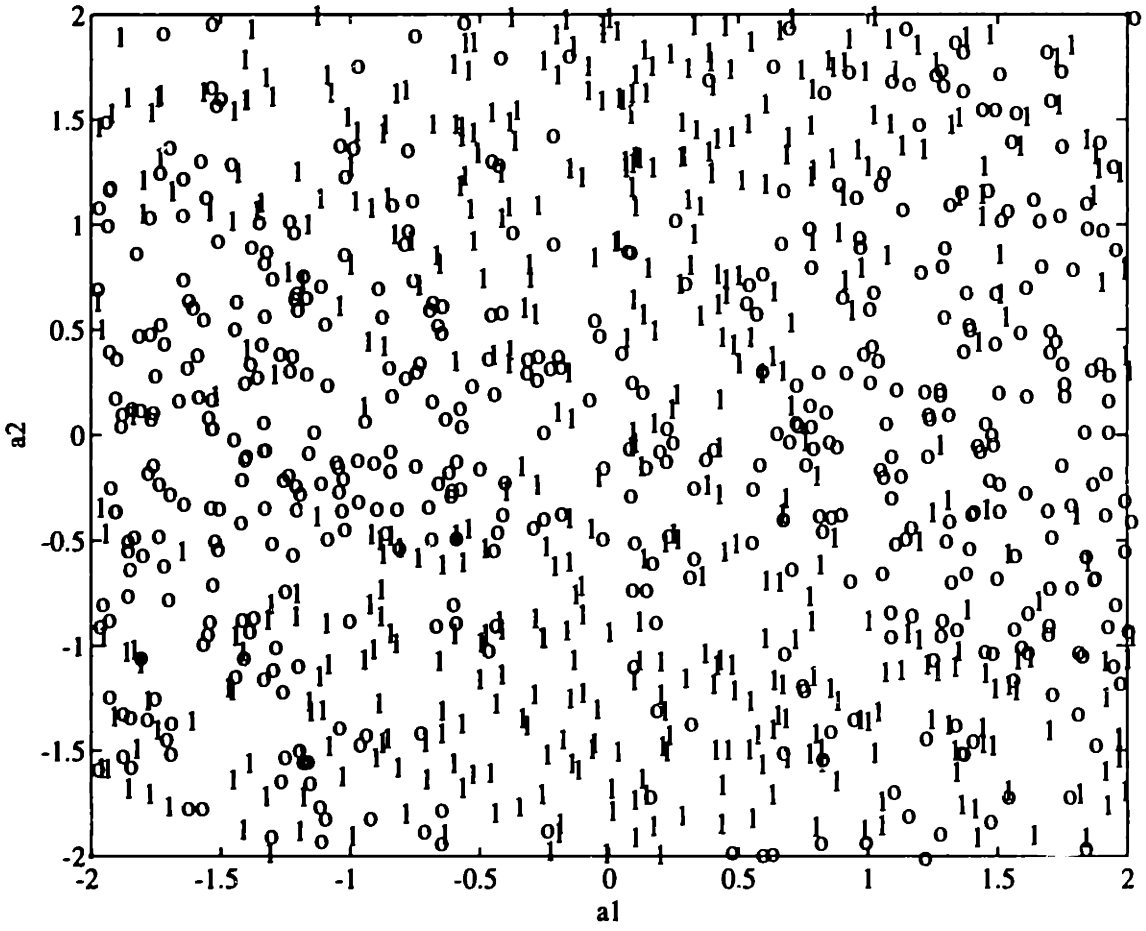

Figure 5-3: Distribution of the training examples used to train the HyperBF network of example 1; '1' represents a success and 'o' represents a failure.

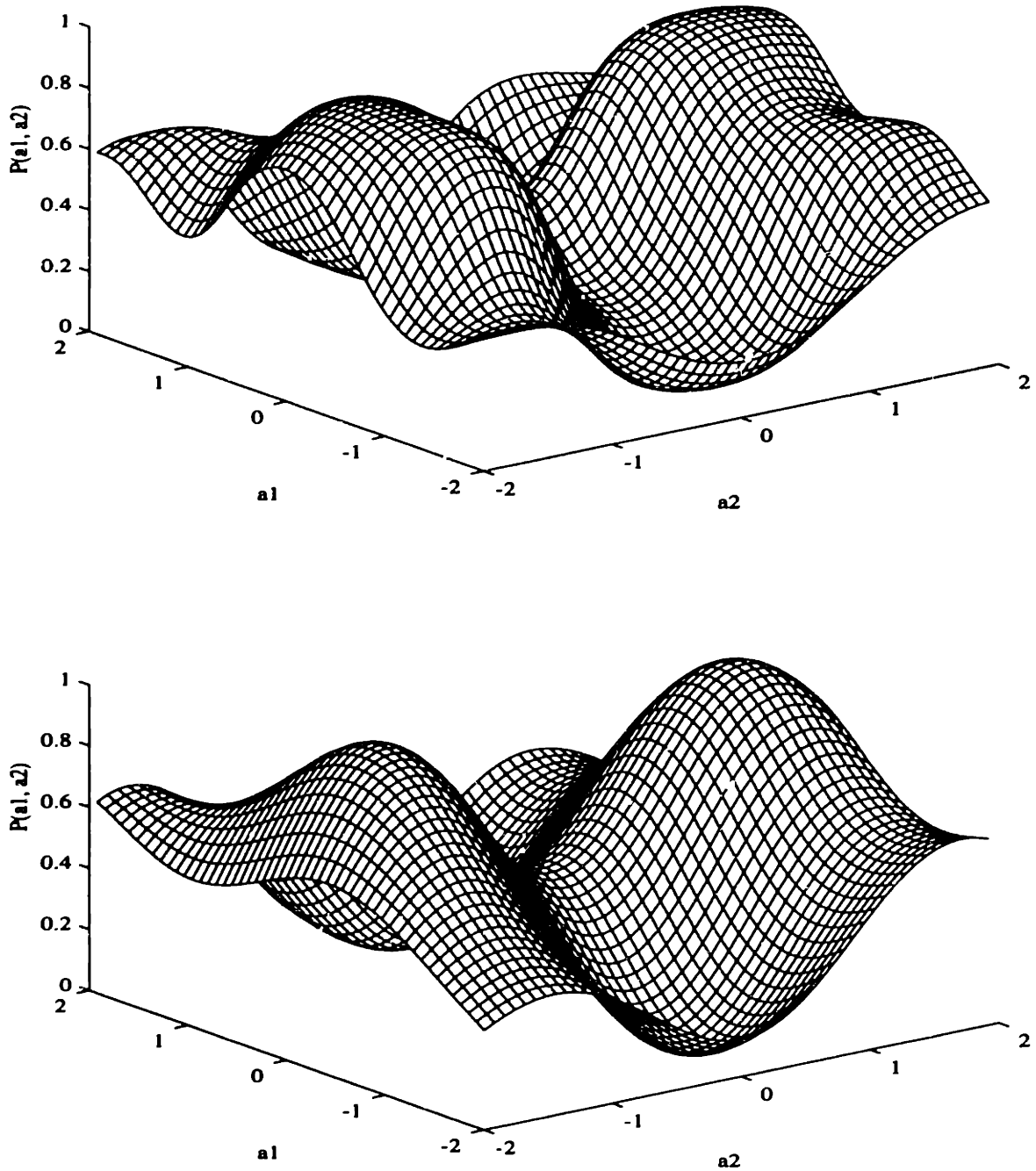

Figure 5-4: Estimated probability of success for example 1 using 1000 data points in the training set. The top figure is the output of the HyperBF network trained using the logistic function, and the lower figure is the output of the HyperBF network using the iterative direct method.

While the use of the logistic function guarantees that the results remain between (0, 1), the direct method as mentioned above does not have such a guarantee. Indeed the maximum value for the estimated probability in figure 5-4 is found to be 1.001. This is obviously not a serious problem in this case, since we can always truncate any value outside the range (0, 1). Although this error value is high with respect to a 2-D deterministic system, it is acceptable for a stochastic system given the amount of data. If we increase the training set to include 5000 data points, randomly and uniformly distributed as before, the normalized error is decreased to 0.0814 for the logistic function estimation and to .0526 for the weighted least squares case. The estimated probability of success for 5000 training examples as a function of the 2-D action space is shown in figure 5-5. This performance is much better than using data in the neighborhood to estimate the probability of success of an action vector. Figure 5-6 shows the performance using neighborhood estimation of the probability of success, using the same 5000 training examples. The action space is divided into bins, and the probability of success of each bin is computed separately, then a smoothing filter is used to reduce the variability of the probability of success function. The smoothing filter used is a simple weighted averaging filter with the desired bin having double the weight of all the neighboring bins. The computed normalized error in this case is 0.193, which is even higher than the error obtained using 1000 training examples when we used Gaussian HyperBFs.

Although directly modeling the data using iterative weighted least squares gives better results for example 1 above, when the number of data points is small compared to the number of parameters this algorithm is not stable and results in a substantially higher error when compared to the logistic function method. For example the normalized error for using only 100 data points in the training set is 0.27 when using the logistic function and 0.40 when using the direct weighted least squares. Since in most of the applications that we will describe in this chapter, the initial number of training points is usually small, we will model the logit function of the probabilities instead

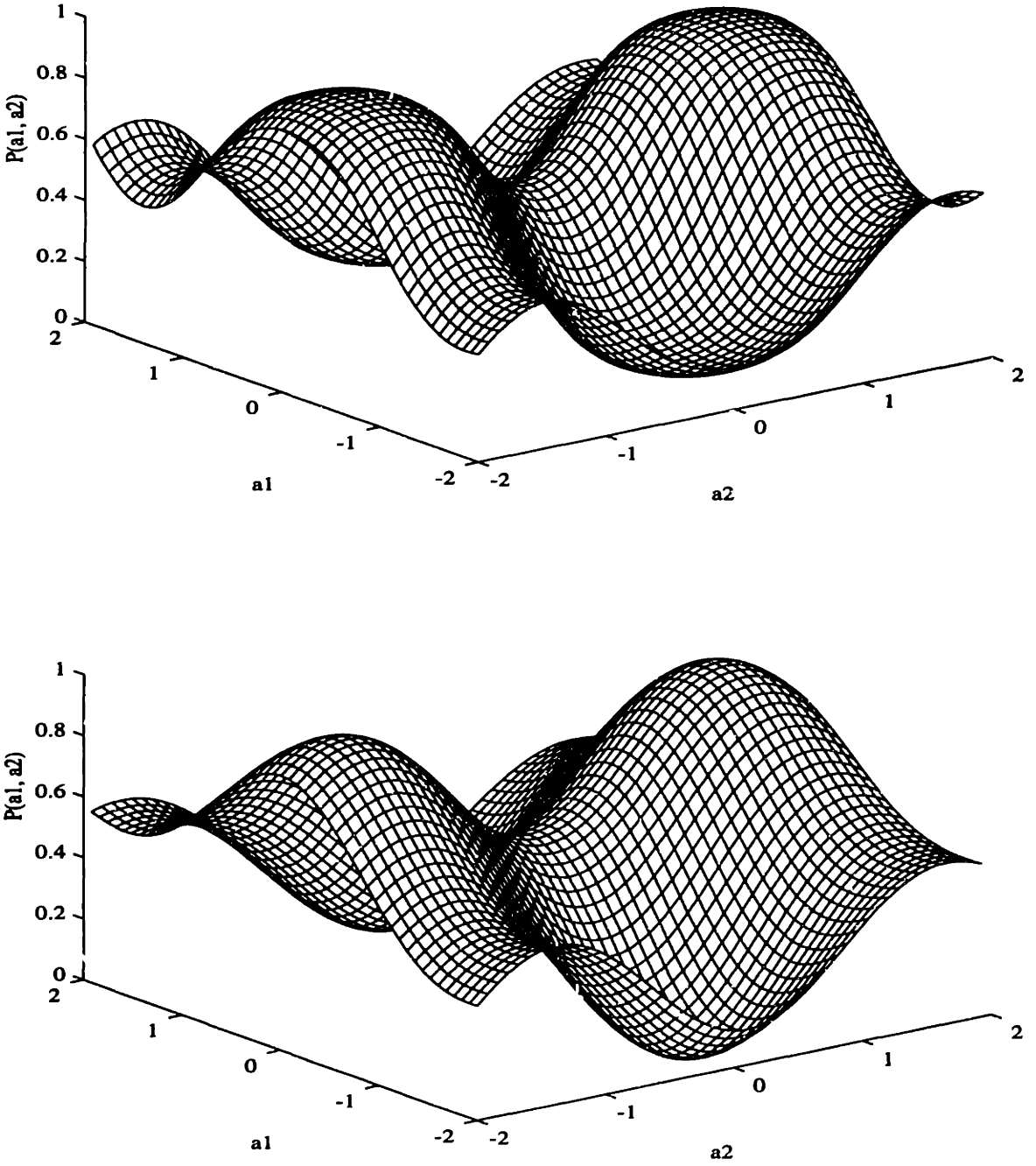

Figure 5-5: Estimated probability of success for example 1 using 5000 data points in the training set. The top figure using the logistic function and the lower figure using the iterative direct method.

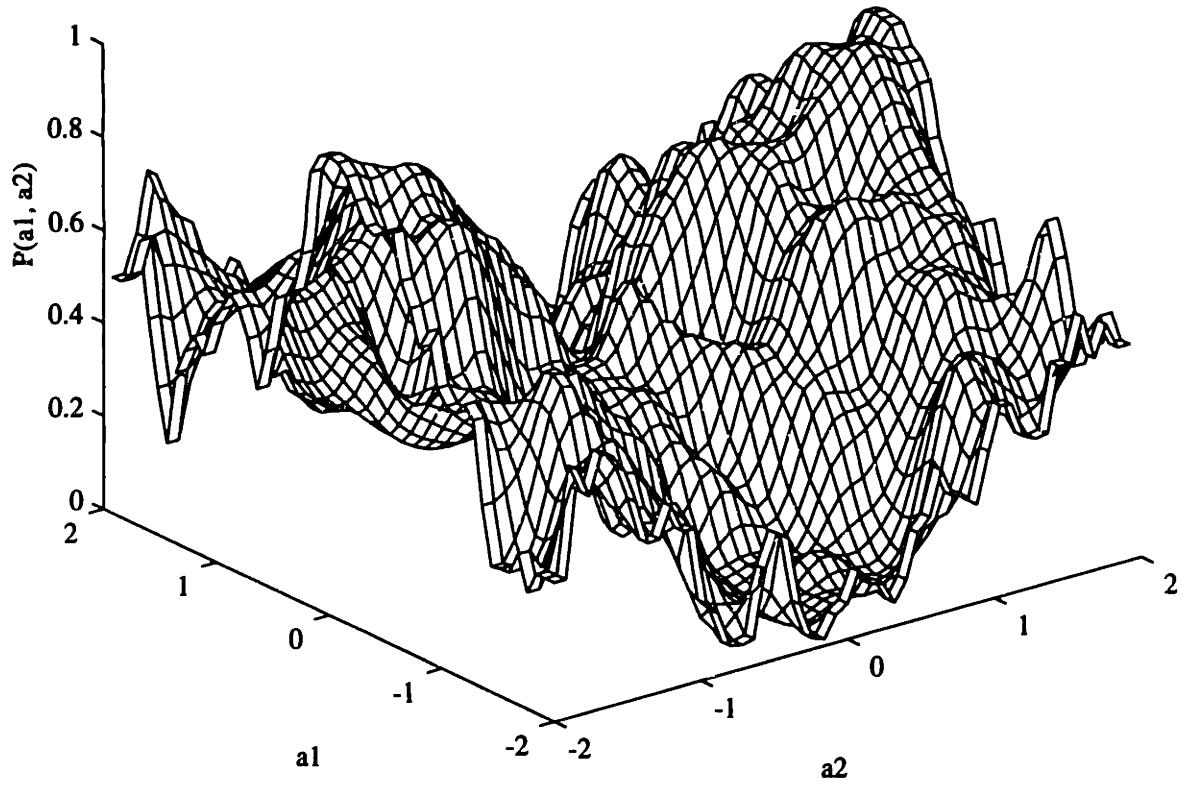

Figure 5-6: Estimated probability of success for example 1 using 5000 data points in the training set and using smoothed bins.

of the direct modeling of the probabilities to ensure better stability of the algorithm. In addition, a regularization factor is important when the amount of data is small. The regularization factor that we chose is a minimization of the sum of squares of the coefficients of the HyperBFs.

Example 1 shows how we can represent the probabilities of success using function approximation techniques, it remains now to find an efficient way for exploration so that we do not waste actions in areas of the action space where the probability of success is expected to be very low but, at the same time, allow for accurate identification of the parameters of the model used to describe the environment. This task is done

automatically for most of the direct methods. For the indirect methods, in addition to finding the optimizing action two other factors have to be considered. First, the whole space of actions has to be explored, if a global optimum is desired. Secondly from the systems identification point of view, the actions should not be constant in order to obtain a more stable model of the effect of actions. One method to perform this exploration for the indirect methods is to locate the optimal action so far and add to it a Gaussian random vector whose standard deviation decreases as the number of actions increases. In addition a completely random action is performed every n steps of optimal actions, so that we can obtain a balanced model. We tested this idea with the function of example 1 and using the logistic model. We started the algorithm by performing 100 random actions uniformly distributed over the action space. Since the purpose is to find the action which gives the highest probability of success, we are not concerned in approximating the probabilities accurately in areas of estimated low probability of success. Analytically, the maximum value of the function of example 1 occurs at $(0,\ \pi/2)$ and $(0,\ -\pi/2)$ in the range of actions used. The location of the optimal actions obtained using this algorithm, are shown in figure 5-7 superimposed on the real contours of equal probability of success. As shown in the figure, most of the optimal values obtained have a probability of success above 0.95. However, at the beginning there were few points who had a probability of success of about 0.5. This plot does not show the random actions which were executed every 5 optimal actions. The algorithm could not find any of the local minima exactly. This is always the case with stochastic systems since there will always be an uncertainty associated with the optimal action chosen. The model probability of success of the chosen actions as a function of iteration number is shown in figure 5-8. This figure also does not include the exploration/identification action executed every fifth optimal action. As shown in the figure, initially the optimal action chosen had a probability of success of more than 0.8. Then there was a jump to a worse action. The reason for this is that, at the beginning when there are only few experiences, an action which results in a relatively

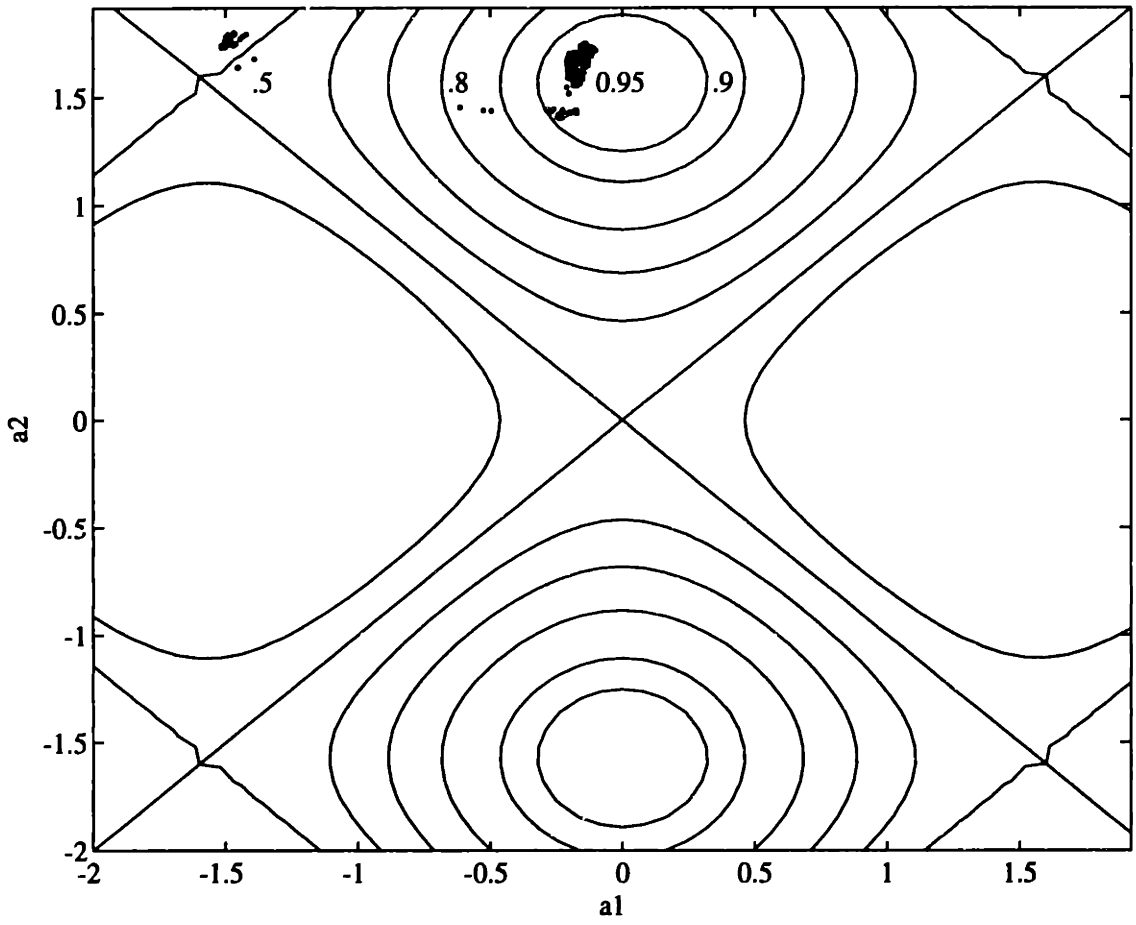

Figure 5-7: Location of the optimal actions for example 1. The optimal actions are superimposed on the model equal probability curves.

unlikely outcome may bias the model considerably.

There are also other schemes to improve identification and optimization of the model that we will use in the simple simulations that we will describe later in this chapter. For example, one can assume that all unexplored actions are initially successful. This will encourage the *curiosity* of the learner. We can also penalize actions based on the density of the previous experiences in that area. This will reduce repetition and avoid local optima. The repetition penalty should be reduced gradually to allow convergence to one action.

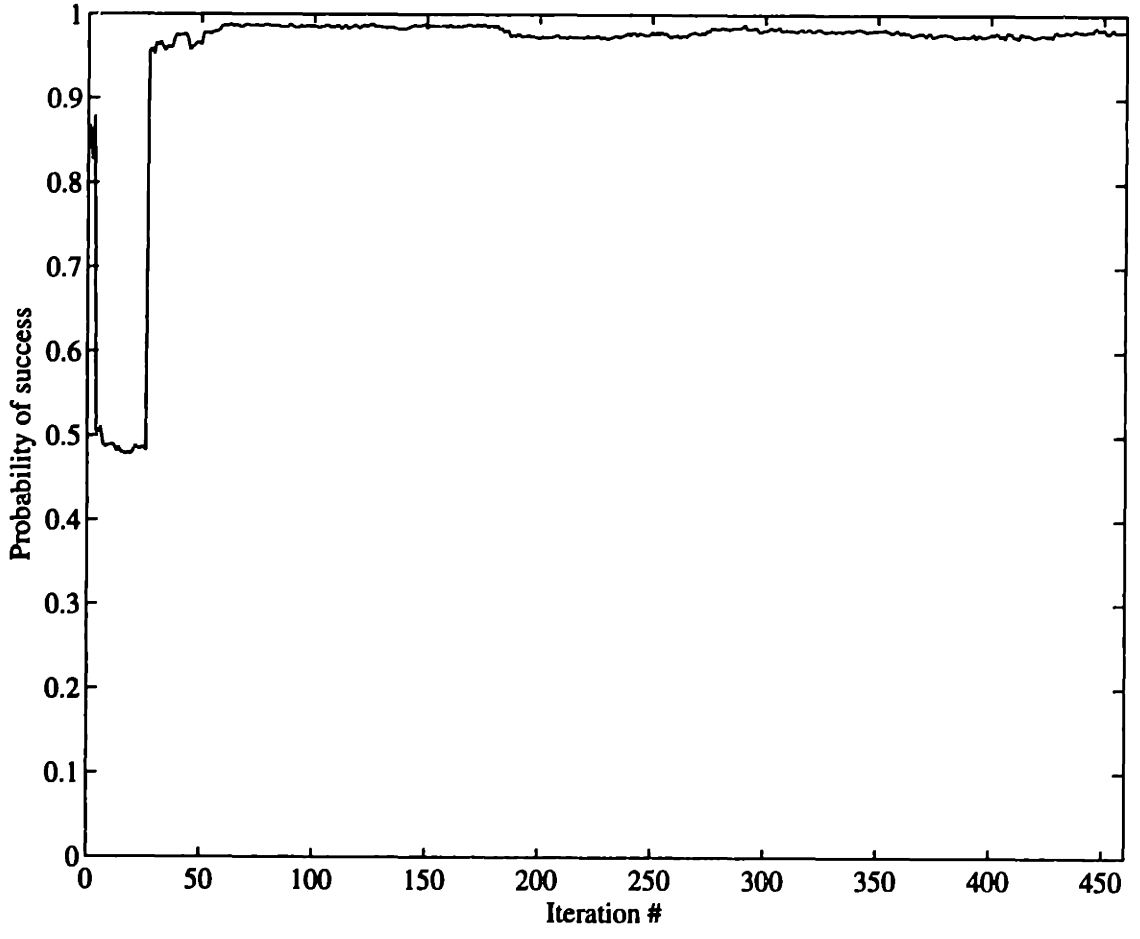

Figure 5-8: Ideal probabilities of success as a function of iteration # for the problem described in example 1.

Maximizing average reinforcement

In many cases the outcome of an action that the learner can measure is in the form of a real valued reinforcement (or cost) signal as opposed to a binary success or failure information. For example, in aiming at a target, the information that the learner gets is not only whether s/he was successful in hitting the target, but also how far was s/he close to the target.

In the previous section, we have shown how HyperBFs could be used to construct a probability of success as a function of actions that we can then optimize and obtain the actions with highest probabilities of success. Similarly in the case when the outcomes are stochastic real values, a simple way for determining the best action based on previous experiences is to directly construct a mapping of the expected cost or expected reinforcement resulting from applying the different actions, and then find the action that optimizes this expected cost or reinforcement. It must be emphasized here that these cost or reinforcement functions are sometimes subjective and only reflect the different priorities of the agent or the decision maker if we have many objectives we would like to achieve. In the case of continuous mappings of stochastic systems, in addition to the usual problems of function approximation mentioned before, namely generalization and learning complexity we have the problem of filtering the noise. We have to estimate an average reinforcement or cost. This is particularly hard when the variances of the outputs are a function of the action space, especially when we have only few data points. One possible way to overcome this problem, is to first assume that the variance has a constant value, and then estimate expected values using least squares error optimization. We can then estimate the variance at each point using the estimated expected values. Finally we update the expected values estimated using weighted least squares, with the weights being the inverse of the estimated variances.

As mentioned before, the agent or the decision maker has active control over the experiences and can therefore choose actions for the purpose of exploration and better estimation of their utility or cost. The other important problem that exists

when using some global function approximators such as RBFs is that the data are not evenly distributed and are clustered around what is considered to be more optimal areas in the parameter space which is a problem for the system identification. One simple way to overcome the problem of clustering and to allow for exploration of different areas of the space, is to add a cost to the action that depends on the average density of the previous experiences but which decays over time to allow convergence. This will encourage early exploration of the space and avoid initial bad distribution of the data. Another method to encourage exploration is to assume initially that unexplored areas have favorable outcomes.

To find the optimal cost or reinforcement we can use any optimization technique such as gradient descent for example. However, since the cost or reinforcement function is only very approximate anyway, there is no need for an accurate optimization technique at least for the first few trials. Not only an approximate technique of optimization will reduce the amount of computation of the search for the optimal parameter set; but it is even more desirable in terms of the long term optimization since it encourages exploration of a larger area of the parameter space which tend to have a low cost. This will minimize the risk of early convergence of the algorithm to a fictitious minimum that resulted only from the poor approximation of the cost function. The accuracy of the optimization algorithm used to find the minimum cost of the approximated cost function could be increased as the number of experiences accumulated is increased and the cost function approximation becomes more accurate.

In the remainder of this chapter, we will describe two simulations of action optimization by building action/outcome models using the techniques described above, and then we will compare these techniques with direct optimization methods such as Gullapalli's stochastic reinforcement algorithm.

5.3 Basketball Shooting : Learning the Best Launching Velocity

There is theoretically a continuum of initial angles θ_0 and speeds v_0 by which we can throw a basket ball from a fixed position into the hoop. One way for a basket ball player to shoot a successful shot, is to have a perfect model of the system, good measurements of the distances between him and the basket and good control of his muscles to generate the exact initial velocity. Another possibility is to have an associative controller which generates an action based on the measurements of the distances from the basket and the other context variables. Even with a high degree of practice, it is unlikely that the player can estimate and control these quantities accurately especially in the presence of adversary players. Therefore, in order to be successful, the player has to choose his action in such a way that it maximizes the probability of success given the different sources of uncertainties and disturbances. There are different strategies that the player uses to maximize his success, for example he may try to run closer to the hoop and jump to reduce the uncertainties. Also he can give the ball a little back spin and aim at the back board instead of aiming directly at the hoop. In a rebound shot, a spin in the correct direction will help keep the ball closer to the backboard and the effect of errors in aiming will be reduced. Also, he can choose an angle and speed of throwing the ball that are less sensitive to noise. The choice of the angle and speed of throwing that optimize success are related. Intuitively a ball falling verically sees a bigger target than when the launching angle become more acute. However, highly arched shots are more prone to error. First, because it is a longer shot, therefore any initial error is magnified. Secondly, because the ball is dropped from a higher position, it gains higher velocity when it reaches the target, and therefore if it hits the rim it will have a bigger rebound and lower probability of success [Broer and Zernicke, 1979]. In this section we will simulate the problem of shooting at a basket from a fixed position, and we will show the result

of applying some of the different optimization methods discussed above for finding the best launching velocity that maximizes the probability of success or maximizes expected reinforcement given random errors in the speed and angle of throwing. We will not address here the other strategies that the player may use to improve the probability of success. The problem as it is stated here is context free, that is, we are only interested in finding the best action for only one fixed position. However, this problem and the different methods of solution presented here could be very easily extended to the context sensitive case by making the model a function of the states as well as of the actions.

Assuming the effects of air drag and spin are negligible, we can approximate the trajectory of the ball in flight using the dynamic equation of a projectile. The relation between the initial speed v_0, initial angle θ_0 and the horizontal and vertical position of the ball $x(t)$ and $y(t)$ respectively could be described by the following equation:

$$v_0{}^2 = \frac{gx}{2\cos\theta_0{}^2(\tan\theta_0 - \frac{y}{x})} \tag{5.10}$$

where g represents the acceleration due to gravity, x and y are the distance and the height coordinates of the trajectory respectively. This equation is derived by eliminating time from the simple dynamic equations of the ball in the x and y direction, with gravity being the only force acting in the negative y direction. In the simulations it is assumed that the ball is thrown from a fixed horizontal distance from the center of the target of $5m$ and a vertical distance of $0.6m$, the target is a horizontal line in the xy plane of width $1.0m$, and only the trajectories that reach the target in their descending phase are considered successful. Gaussian noise is added to the initial speed and angle of launch of the ball. The standard deviation of the noise added to the initial launch angle is constant, but the noise added to the initial speed of the ball is assumed to be proportional to the launching speed. Given the above assumptions, the contour curves of the probability of success as a function of launching speed and angle are shown in figure 5-9. As shown in this figure, there is a single peak for the

probability of success function. This highest probability of success is achieved near the minimum launching speed which is due to the small variance in the initial speed near the minimum. It is also obvious from the figure that the probability of success is more sensitive to the initial speed than to the initial angle of launch. The highest probability of success is about 0.6 given the level of noise used in the simulation.

The first simulation assumes that the player gets only a binary feedback signal representing success or failure of his/her action. The probability of having a success if the actions are uniformly and randomly distributed in the action space is about 0.055. This makes it hard to model using only success or failure information and few examples, since most of the examples are failures and we need few hundred data points to get a reasonable initial approximation. We use HyperBF to represent the probability of success as described in example 1 above, and then optimize this probability to obtain the best action. The initial model of the probability of success is generated using 100 initial experiences uniformly distributed across the action space. The velocity range is $[0, 20]$ m/sec and the angle range is $[0, \frac{\pi}{2}]$ *Radians*.

The location of the estimated optimal actions for 100 additional iterations are shown in figure 5-10 superimposed on the contours of equal probability of success. Except for few points in the first few iterations, practically all the estimated optimal angles and velocities have a probability of success greater than 0.5. Figure 5-11 shows the probability of success of the estimated optimal value against the true optimal value. This figure was smoothed using a moving average window of a width of 10 points to remove the noise. As shown in the figure, the optimal action chosen has a higher probability of success as the number of iterations is increased. However, the probability of success does not reach the optimal one. The contours of the final estimated probability of success model used to compute the optimal action is shown superimposed on the contours of corresponding probability contours for the true model in figure 5-12 where the solid lines represent the estimated probabilities and the dotted lines represent the true probabilities. This figure shows that the es-

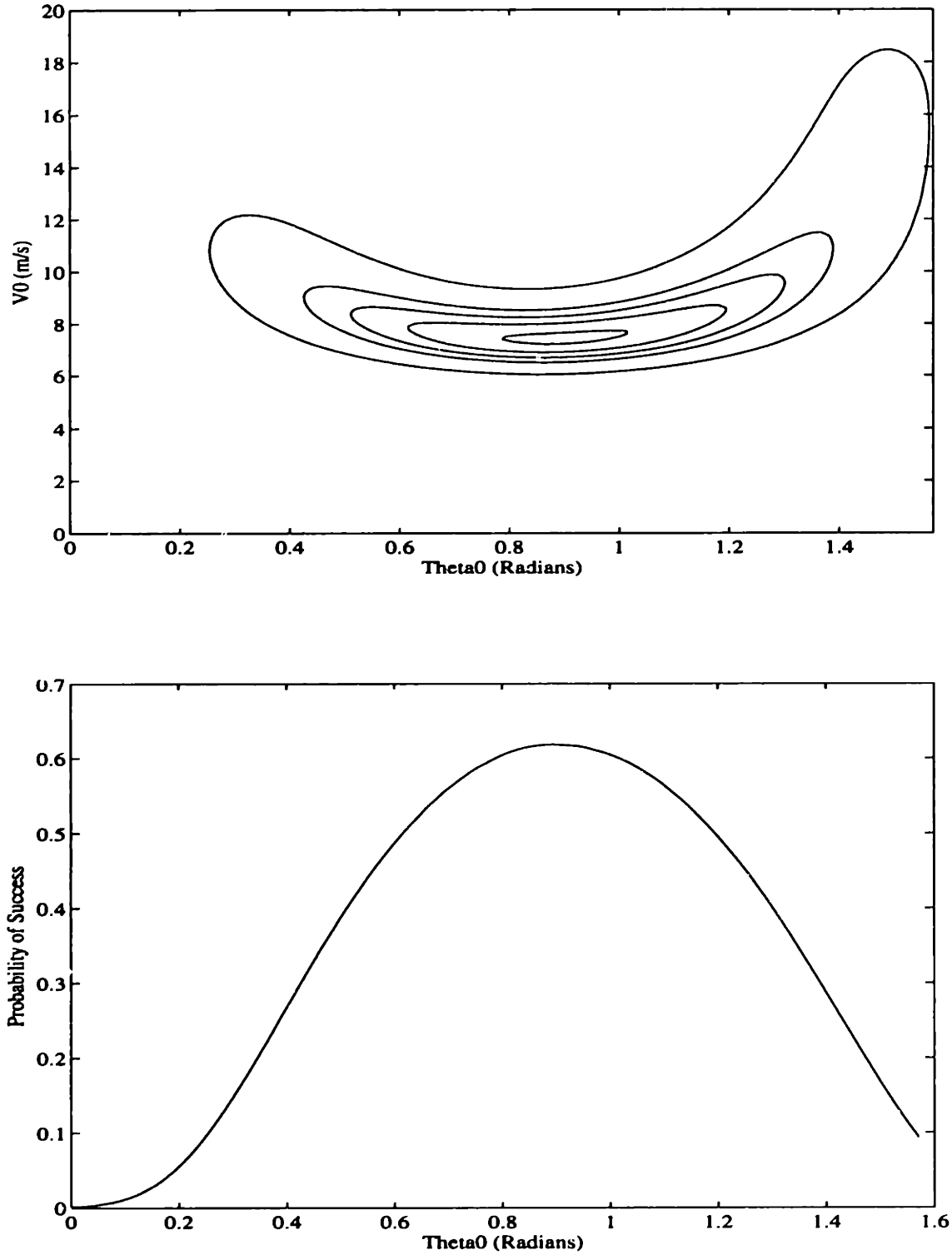

Figure 5-9: Contours of simulated probabilities of success as a function of the launching speeds and launching angles. The highest probabilities of success are then projected as a function of the launching angles

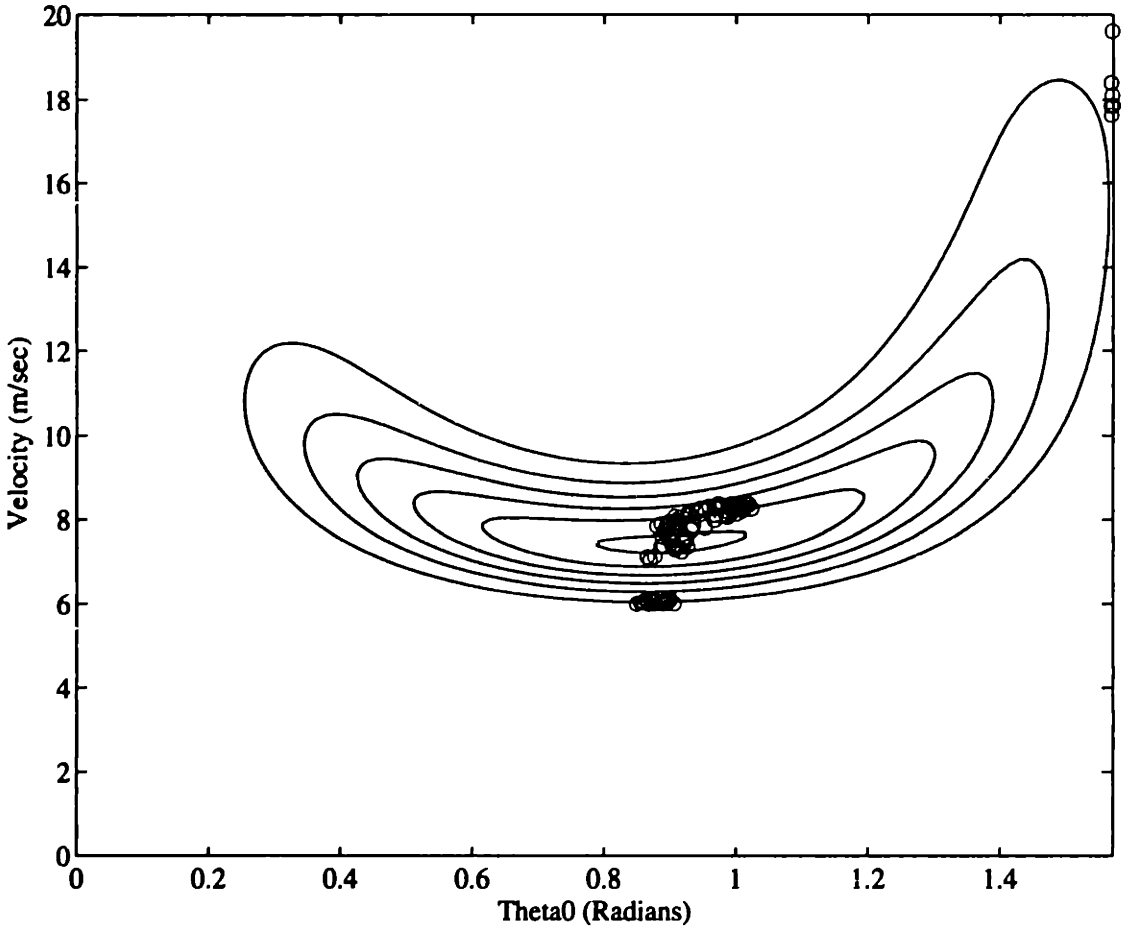

Figure 5-10: Estimated optimal actions for the basket ball simulation. The optimal actions are superimposed on the model equal probability contours.

timated location of the probability peak occurs at a slightly higher velocity than in the true model.

We would like here to show how the algorithm described above may be related to stochastic learning automata type algorithms. Although the term "Learning Automaton" is mostly used to describe a system with finite action space, we will generalize this description to include systems with continuous action and state spaces. Stochastic learning automata algorithms find the best action by directly updating the probability of using any action in the future based on previous responses. The algorithm for updating these probabilities is called reinforcement. Based on the re-

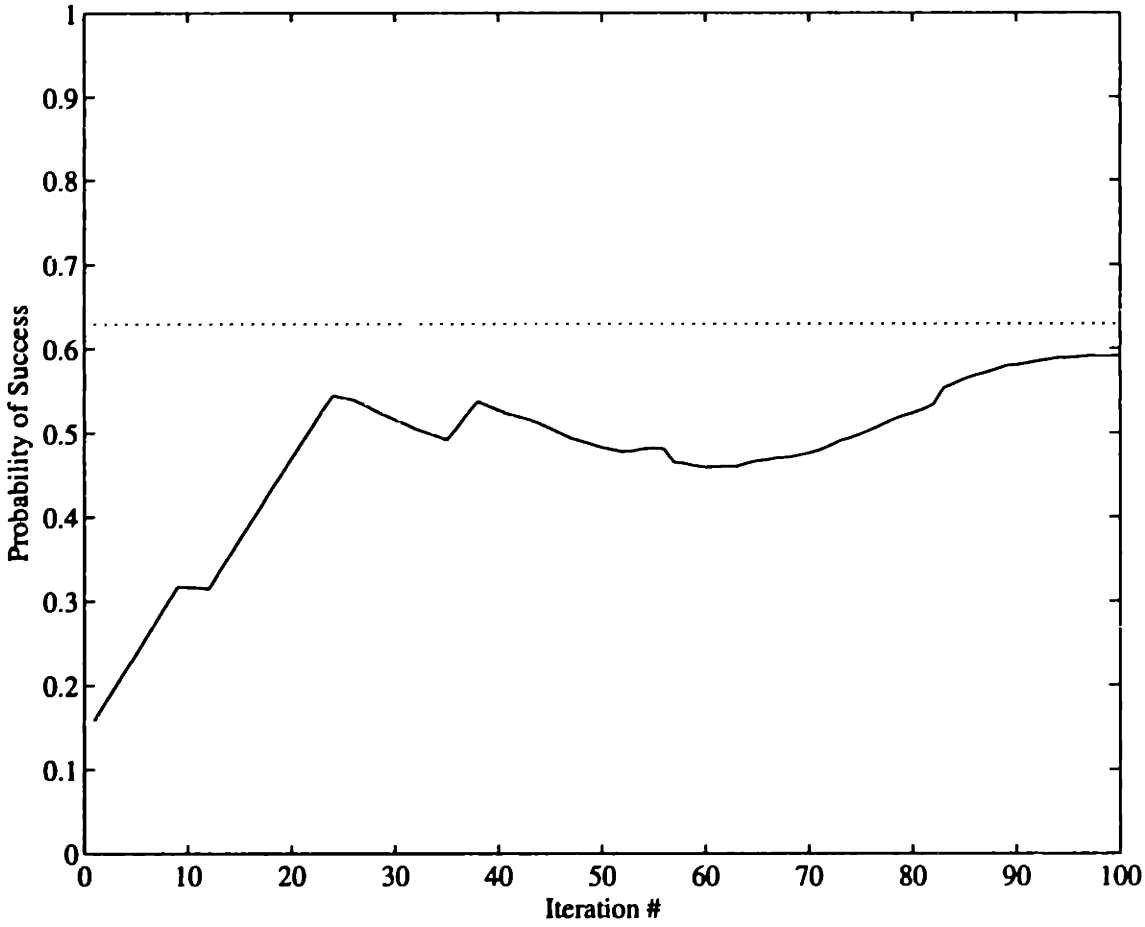

Figure 5-11: Estimated probabilities of success of the optimal actions for the basketball simulation as a function of iteration number.

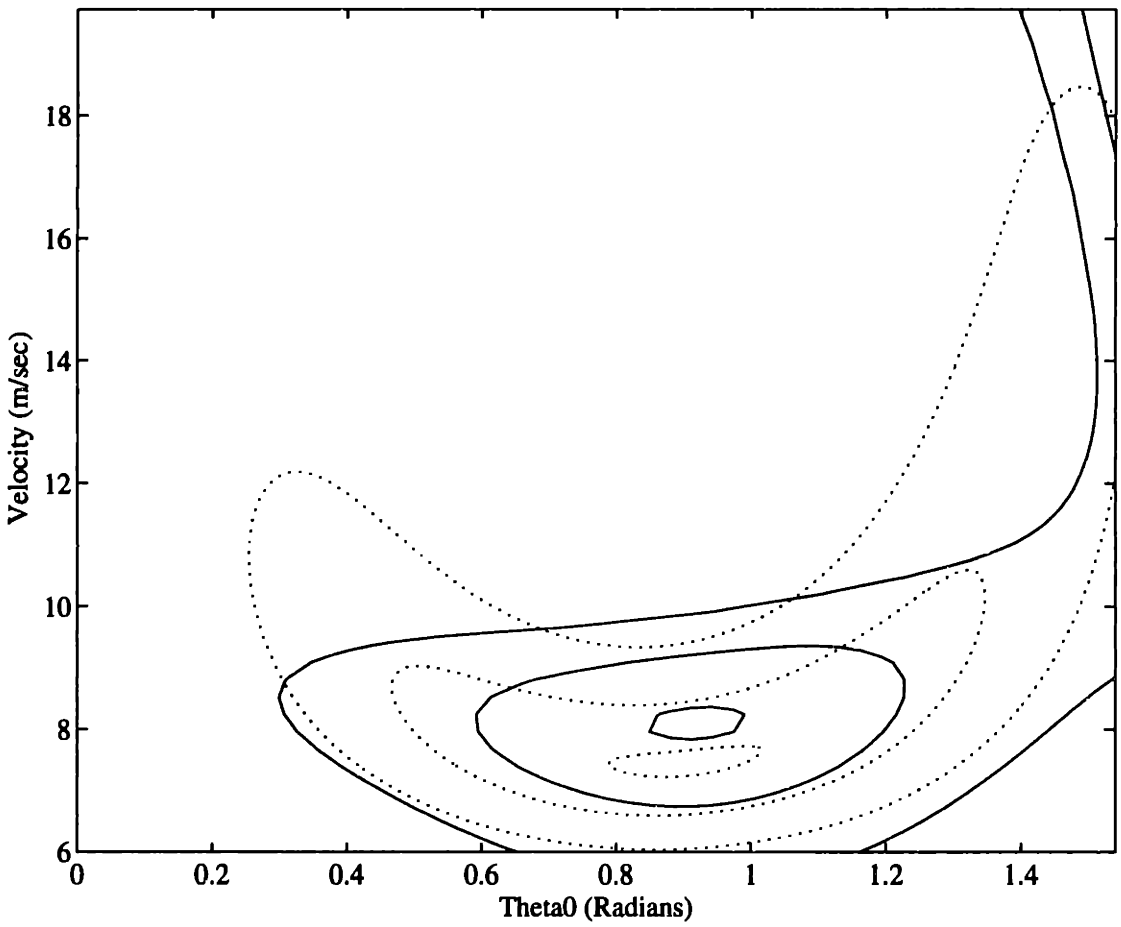

Figure 5-12: Estimated probability contours (solid lines) versus true model probability contours (dotted lines) for the basket ball simulation.

inforcement scheme, different types of behavior emerge. Algorithms which tend to reward success and penalize failure symmetrically, for example linear reward-penalty (L_{R-P}) algorithms [Narendra and Thathachar, 1989], generally do not converge to the optimal action, although their behavior is called "expedient", that is, better than pure chance. The probability for choosing an action for these algorithms usually converges to a probability distribution related to the probability of success (or failure) of the action. In the case of L_{R-P} algorithm, the probability of choosing an action after a large number of iterations is inversely proportional to the probability of failure of that action. Although this algorithm tends to choose more successful actions more often, it is not optimal. Knowing the probability of success or failure, we can generate a sequence of actions with the same probabilities as the L_{R-P} algorithm. Other reinforcement schemes reward successes and penalize failures in a nonsymmetric fashion. If success is rewarded more than failure is penalyzed, these algorithms tend to converge to more optimal actions. At the limit, when we do not penalize failure, for example the linear reinforcement - inaction (L_{R-I}) algorithm, the algorithm converges to the optimal action for discrete action spaces. Although this algorithm is optimal, its convergence may be slow, because it does not make use of the experiences which resulted in a failure. Modeling the probability of success and then finding the optimal action will converge to the optimal action as the model for the probability of success improves. The advantage offered by direct optimization schemes is that they combine the identification process with the process of selecting new actions in such a way as to improve overall performance. The schedule of actions we use to obtain a better probability of success model is different from that used by the reinforcement algorithms. Also we make symmetric use of all the actions, whether they resulted in success or failure, to update the model. The disadvantage of using the indirect method of first modeling the probability of success is the time it takes to find the optimal action from the model which makes it not suitable for applications which require fast action.

5.4 Bouncing a Ball with a Racquet: Learning the Optimal Periodic Trajectory

In this part we are interested in answering the following question: given the task of bouncing a ball with a racquet, what would be the most robust periodic open loop trajectory of motion of the racquet, given external perturbations on the trajectory of the ball. The trajectory of the ball should also be close to periodic, having the same period as the trajectory of the racquet. In general, if the coefficient of restitution of the racquet is smaller than one, for each constant velocity of the racquet there is a stable height that the ball will reach. However in this case we do not have any control over the phase of the hit (at which time of the periodic cycle we hit the ball) and the robot may miss to hit the ball in one cycle or hit more than once during the same cycle. A smaller coefficient of restition results in a more stable ball trajectory. Intuitively, to stabilize the phase relationship, we would like to hit harder at the beginning of the phase and softer near the end to move the phase of hit to the middle of the cycle. This intuitive idea has been implemented and tested by Atkeson and has been shown to work [Atkeson, 1991].

A one dimensional ball bouncer is initially simulated using a constant speed trajectory of the racquet. This speed corresponds to a stable height of one meter, given a coefficient of restitution equal to 0.9. The ball is dropped from any random height in the range [0.m, 2.m] and at any random phase of the trajectory, then the bouncing of the ball is simulated until failure. The phases at which consecutive hits occur are then analyzed and the trajectory of the racquet is modified in a direction that makes the ball bouncing more stable. Two different methods were used to learn the optimal trajectory : a direct method where the trajectory is adapted directly after each failure, and an indirect method where a model of the cost of each action as a a function of the phase of the racquet trajectory is estimated using a HyperBF network. This model is then used together with dynamic optimization to generate the best trajec-

tory. The cost used is heuristic and depends on whether the hit trajectory resulted in improving the trajectory of the ball in the next cycle. This cost function is also stochastic since the behavior of the ball in future cycles depends on its behavior in the past which is affected by random perturbations. In both methods tried, we added the additional constraints that the position of the racquet at the beginnning and end of the cycle are fixed. The velocity of the racquet is also constrained to be within a specified range. The presence of these constraints makes the use of direct methods less straightforward.

5.4.1 Direct method

The trajectory of the racquet is discretized into 20 bins of equal velocity. We then simulate the ball bouncing until failure using the discretized trajectory. The velocity of the bin when failure occured is then increased or decreased depending on whether the hit resulted in a shorter cycle or a longer cycle respectively. Only the direction of the error is used, but not its magnitude. In order to satisy the above mentioned constraints and keep the range of movement constant, the velocity of the mirror bin is adjusted in the opposite direction so that the integral of the velocity remains constant. By doing this, we implicitly force the trajectory of movement of the racquet to be symmetric. Figure 5-13 shows the trajectory of the racquet for one cycle after 500 trials. The velocity is approximated as a linear function of the phase which is expected. No restriction was made in the learning algorithm to insure that the resulting velocity is smooth. Also information on the performance of one bin affected only the velocity of that bin and its mirror image.

5.4.2 Optimizing expected cost

In this method, the system is simulated until failure. At failure, an instantaneous cost is computed as a function of the phase at which the hit occured and the instantaneous velocity of the racquet at ths time. A dynamic optimization algorithm is then used

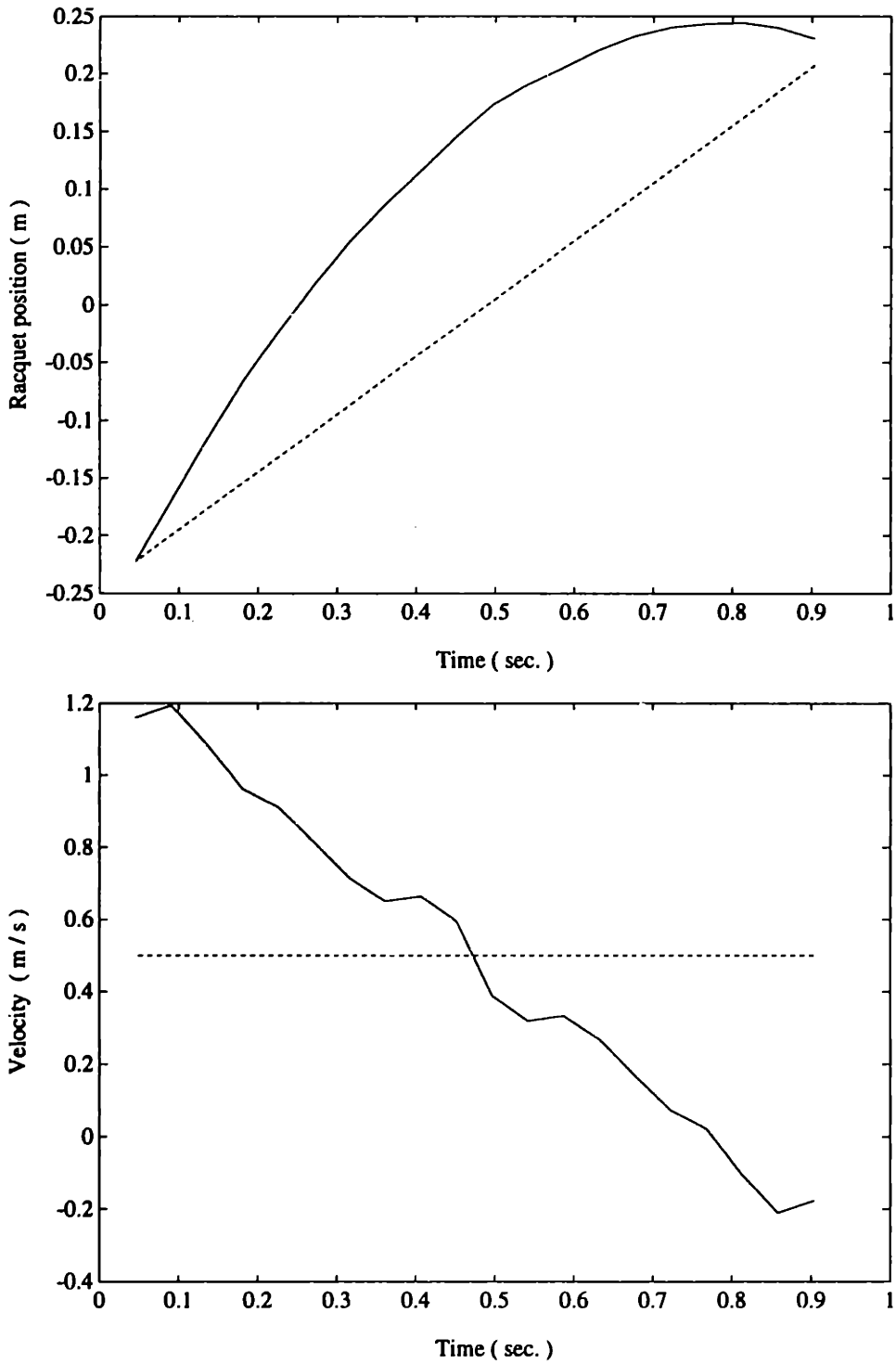

Figure 5-13: Position and velocity of the racquet as a function of time before (dashed line) and after (solid line) learning

to minimize the integral of the cost over the whole period while at the same time satisfying the end point constraints on the position of the racquet and the constraints on the maximum magnitude on the velocity of the racquet. The cost at each hit is given by equation 5.11

$$cost(t(n), v(t(n)) = (t(n+1) - 0.5)^2 \tag{5.11}$$

Where $t(n)$ represents the phase at which the racquet hits the ball during the n^{th} cycle and $v(t)$ is the velocity of the racquet as a function of the phase. This cost encourages hitting the ball near the center of the cycle. We represented the cost using a HyperBF network with 30 centers randomly and uniformly distributed. We then used dynamic optimization, described in detail in the previous chapter, to optimize the integral of the cost over the whole period under the end point constraints. The optimization procedure is as described by equations 5.12- 5.15. Cost function :

$$v(t)^* = argmin J(v(t)) = \int_0^T cost(t, v(t))\, dt \tag{5.12}$$

System equation :

$$\dot{p} = v(t) \qquad p(0) = -0.2 \quad p(T) = 0.2 \tag{5.13}$$

Hamiltonian :

$$H = cost(t, v(t)) + \lambda(t) v(t) \tag{5.14}$$

Costate equation :

$$\dot{\lambda} = -\frac{\partial H}{\partial p} = 0; \tag{5.15}$$

500 hits were used in the estimation of the cost function and its optimization. The estimated cost function as a function of the phase and velocity of the racquet is shown

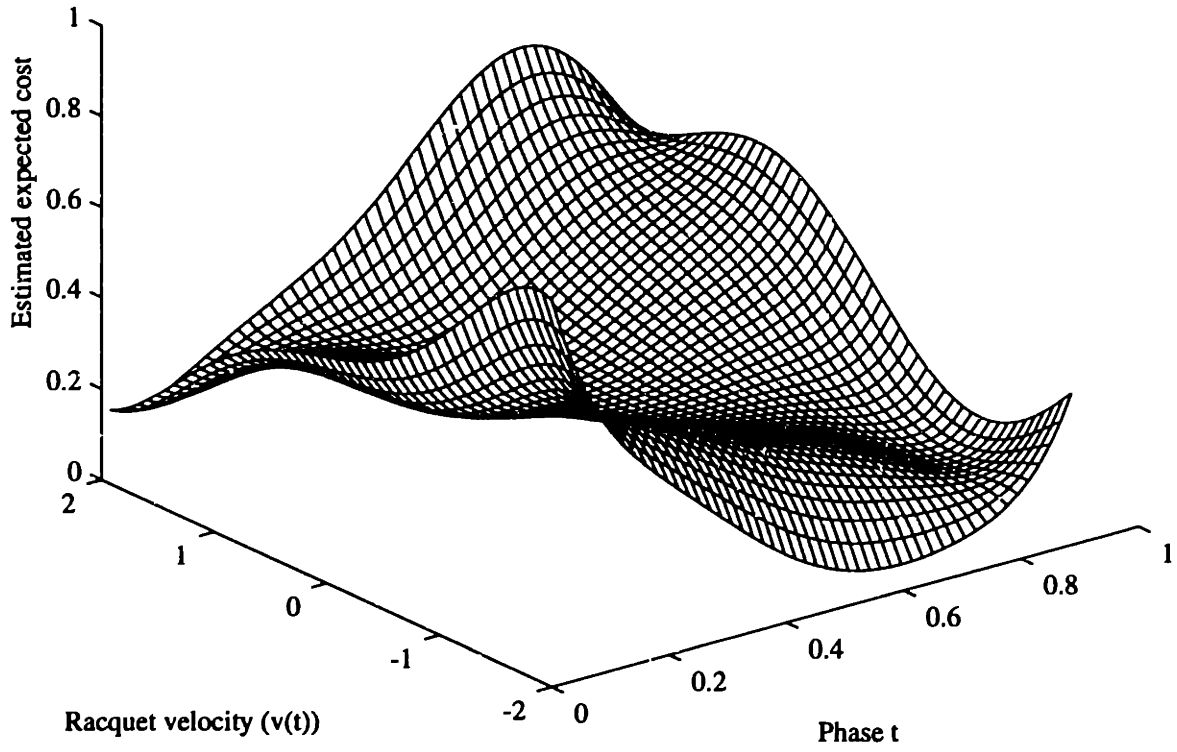

Figure 5-14: Expected cost estimated using a HyperBF network with 30 centers.

in fig 5-14 As shown in this figure, a higher speed at a low phase has a smaller expected cost, conversely at a phase close to one, a negative speed has a smaller cost. It is important to note that this cost function represents the estimated expected cost, the actual cost measured by the learner is a random variable. To show the relation between the actual and estimated expected cost, we plotted the estimated cost at a phase of 0.05 and the actual cost for experiences which had a phase less than 0.1 in figure 5-15. It is obvious from this figure that the variance in the measured cost is very high. The optimal position and velocity of the racquet computed using the

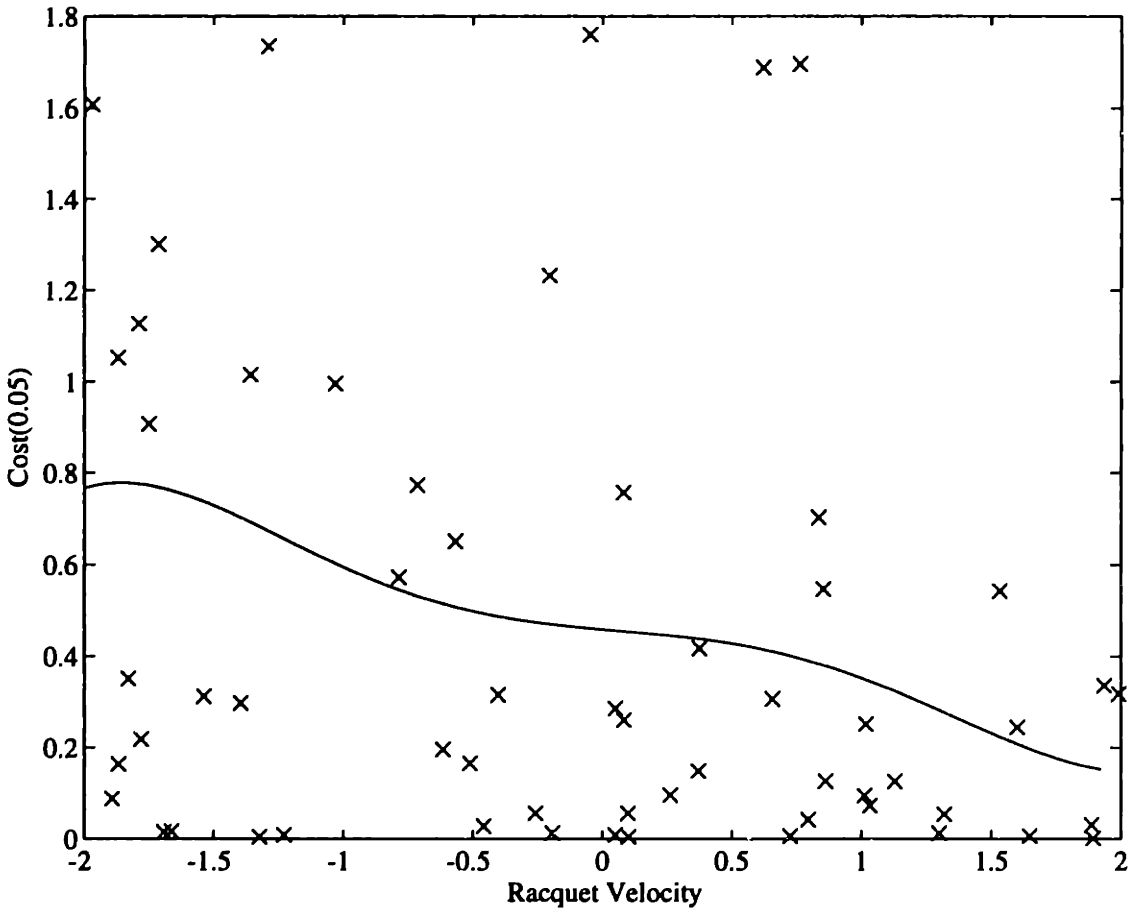

Figure 5-15: Comparison between the real cost data and the estimated expected cost as a function of the velocity of the racquet. The expected cost is estimated at a phase of 0.05, while the real data are the measured cost for experiences with phases that lie between zero and 0.01.

dynamic optimization procedure described above on the estimated expected cost and the constraints on the position of the racquet is shown in figure 5-16. As shown in the figure, the racquet starts the cycle at a high speed, than the speed is reduced almost linearly in such a way to satisfy the constraints on the position of the racquet at the end of the phase. It is important to realize that this trajectory profile of the racquet is dependent on the nature of the disturbances. In this case the disturbance on the height of the ball was uniformly distributed and symmetric around the ideal height. The velocity profile in this case is almost symmetric around the ideal constant speed velocity which occur almost in the middle of the phase. If we choose another disturbance distribution, we would expect to obtain a different trajectory of the racquet.

5.5 Conclusion

The main focus of this chapter is to test the indirect methods for optimizing actions in stochastic environments. Expected values for nonlinear stochastic functions are hard to model because the variance of the noise is different at each point. We describe here an iterative technique for overcoming this problem which uses previous estimates of the expected values to improve the estimate of the variance of the noise and then use the latter to improve the estimation of the expected values.

We have shown the possibility of optimizing actions or a sequence of actions in stochastic environments by first building a model that describes an average behavior of the outcome as a function of actions and states and then optimizing this model using known optimization algorithms. There are many advantages to such an approach. One advantage is that the knowledge gained from all the previous experiences is not lost and can be used when the objective changes. This is unlike direct methods of optimization, such as reinforcement learning and gradient based methods where the experience gained is represented relative to the desired objective.

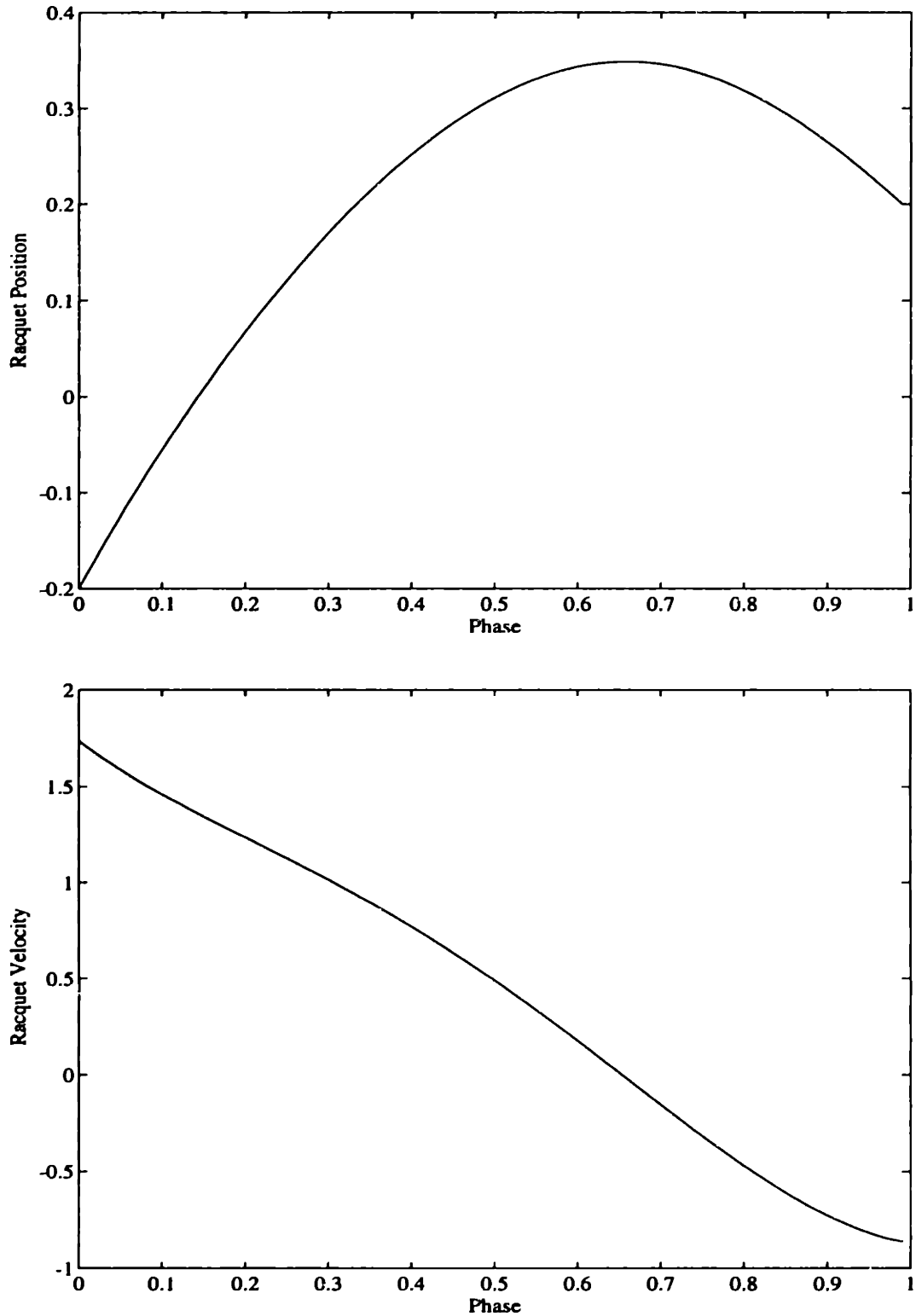

Figure 5-16: Position and velocity of the racquet as a function of time using estimated expected cost and dynamic optimization and 500 experiences.

One main feature of indirect methods is the separation of the identification and control processes (also called the exploration-exploitation problem). This problem does not arise in direct optimization methods since the selection of actions is part of the optimization algorithm. For indirect methods we have to explicitly define a strategy for exploration which should converge to the optimal action. This problem has been addressed in detail in a previous chapter.

Chapter 6

Conclusion

6.1 Practice and Optimization

In this thesis we focus on the acquisition of motor skills by practicing. Practicing is viewed here as an active optimization process in a multidimensional space with respect to a given goal of movement [Gel'fand and Tsetlin, 1971]. There are many different ways to perform this optimization. The approach we have investigated here is to observe previous performances and then use past experience to build and select hypotheses about the task and the environment. Future practice is then based on the current hypotheses. We should balance between the need to validate and improve these hypotheses on one hand, and the need to improve the performance of the task. As the confidence in the current hypothesis increases, as measured by the accuracy of its predictions, future practice should shift to optimizing the performance. We have implemented this approach and tested it in the context of multivariable function optimization. We found that this approach works well compared to gradient approaches. However, it must be understood that the best search strategy used depends on the nature of the task and the function to be optimized. For simple convex functions, for example, it may be more appropriate to use a gradient search since the direction of the search will always lead to the best action. On the other hand, when performance

results are noisy, it may be better to form a model of the average performance. Moreover, it has been shown experimentally that sometimes emphasis on accuracy may impair learning for an unpracticed subject [Newell and McDonalds, 1992].

Finding the best sequence of actions is a much harder optimization problem. This is mainly due to two main reasons: the difficulty of attributing performance to a particular action in time (temporal credit assignment) and the constraints imposed on the exploration strategy, for example actions must result in a continuous trajectory. This second reason results in the difficulty of exploring the effects of actions in some regions of the state space that are hard to reach. The method that we used for optimal trajectory generation estimates a model of the dynamics which is then used to compute the gradient of the cost function with respect to the action trajectory. The new action trajectory is then selected along the gradient in addition to a noise term which decreases with practice. This search strategy can be termed as a local search procedure. It is prone to being stuck in local minima. Also, it does not solve the second problem mentioned above, namely it does not guarantee the exploration of all the regions of the action-state space.

The most commonly used strategies to find the best actions under uncertainty are based on reinforcement learning. In this thesis we explored the use of an alternative strategy which, instead of modifying the probability of choosing an action based on its performance, it represents some parameters of the statistics of previous performance as a function of the different actions and uses these statistics to search for the optimal action given a certain performance measure. We tested this approach on two different problems and it was found to perform well. However a more detailed comparison between this approach and reinforcement learning approaches is needed. The strategy for choosing actions that we used was heuristically chosen to alternate between random and optimizing searches. A better search strategy may improve the results obtained using this approach.

6.2 Storage and Recall of Optimal Actions

In most of this thesis we focused our attention mostly on how to search for the optimal actions, we did not discuss in detail how these optimal actions are then stored to be used in the future. One possible approach is to build a static mapping from the current state of the environment and the goal of the movement to the optimal action found that can achieve this goal. Another approach may be that we never store the optimal actions. The experience we gain from practice is only used to geneate a model of the task. We then use this model to generate the optimal experience in real time. Dynamic recurrent neural networks may be capable of performing such an optimization in real time. The connection strengths between the units of such a network directly depend on the model of the task generated from previous experience in addition to the goal of the task. This approach is appealing from a biological point of view since most of the neural networks in the central nervous system responsible for motor functions tend to be recurrent dynamic networks. [Lukashin and Georgopoulos, 1993] trained a dynamical neural network to perform neural population coding. They showed that the dynamical behaviour of the network resembles the experimentally observed dynamics of the motor cortex neurons.

6.3 Relation to Models of Motor Learning

Many psychological models of learning have been proposed to explain the numerous observations made about animal and human motor learning. [Adams, 1990] provides a historical review of the evolution of the psychological motor learning theory. Earlier models of motor learning have emphasized a non-cognitive approach and the automatic reinforcement of behavior based on the knowledge of results [Adams, 1990]. Such theories can not explain some modes of learning such as observational learning and mental practice. A model-based approach to learning can explain these modes of learning although other interpretations are also possible. Knowledge of results and

observation of performance are still necessary for the learner to be able to form a model, however the search for the optimal action can be performed on the learned model.

Although many experiments have been done that measure the improvement in performance, there have been relatively few research studies in motor control which focused on the search strategies that human or animal subjects use to find the best actions [Newell and McDonald, 1992]. One such study is reported by [Newell and McDonal, 1992] and uses protocols similar to those developed by Krinskii and Shik [Krinskii and Shik, 1964]. In this series of studies, a function of the subject arm joint angles is measured and results are shown to the subject. The shape of the function is unknown to the subject. The subject is required to move his/her arm to minimize this function. The search trajectory performed by the subject are then recorded and analyzed. These studies showed that subjects use different search strategies. Some subjects vary just one coordinate at a time while others use complex pattern search [Newell and MacDonald, 1992]. More experiments need to be done to determine how the nature of the task affect the search strategy and whether the search strategy used depends on the amount of previous experience.

Bibliography

[1] J. A. Adams, 1990. "The changing face of motor learning". *Human Movement Science*, 9: 209-220

[2] K. J. Arrow, L. Hurwicz and H. Uzawa, 1958. "Studies in Linear and Nonlinear Programming" Stanford University Press, Stanford, California.

[3] C. G. Atkeson, 1986. "Roles of knowledge in motor learning" Ph.D. Thesis, Massachusetts Institute of Technology.

[4] C. G. Atkeson, E. W. Aboaf, J. McIntyre, D. Reikensmeyer, 1988. "Model-Based Robot Learning". Massachusetts Institute of Technology, Artificial Intelligence Memo 1024.

[5] C. G. Atkeson, 1989. "Using local models to control movement" In D. S. Touretzky, editor, *Neural Information Processing Systems.* Morgan Kauffman, San Mateo, Ca.

[6] C. G. Atkeson, 1990. "Memory based techniques for task-level learning in robots and smart machines". *Proceedings of the 1990 American Control Conference*, p. 2815-2820, San Diego, Ca.

[7] C. G. Atkeson, 1991 "Memory-based learning control". *Proceedings of the 1991 American Control Conference*, p. 2131 - 2136, Boston, MA.

[8] C. G. Atkeson, 1991. "Using locally weighted regression for robot learning". *Proceedings of the 1991 IEEE International Conference on Robotics and Auomation* v.2, p. 958 - 963, Sacramento, Ca.

[9] A. G. Barto and P. Anandan, 1985. "Pattern Recognizing Stochastic Learning Automata". *IEEE Transactions on Systems, Man and Cybernetics* 15:360-375.

[10] D. M. Bates, M. J. Lindstorm, G. Wahba, and B. S. Yandel, 1987. "Gcvpack - routines for generalized cross validation". *Commun. Statist.-Simula.*, 16(1):263–297.

[11] R. E. Bellman, 1978. "An introduction to artificial intelligence: Can computers think?". Boyd and Fraser, San Francisco.

[12] R. E. Bellman and S. E. Dreyfus, 1962. "Applied Dynamic Programming". Princeton University Press, Princeton, NJ.

[13] D. Bertsekas, 1982. "Constrained optimization and Lagrange multiplier methods". Academic Press, New York.

[14] U. Beyer and F. Smieja, 1993. "Learning from Examples using Reflective Exploration". Electronic copy, available via anonymous FTP from archive.cis.ohio-state.edu as file pub/neuroprose/beyer.explore.ps.Z

[15] S. M. Botros and C. G. Atkeson, 1991. "Generalization Properties of Radial Basis Functions". In *Advances in Neural Information Processing Systems 3*, R. P. Lippman, J. E. Moody and D. S. Touretzky, eds., Morgan Kaufmann, San Mateo, Ca.

[16] S J. Bradtke, 1993. "Reinforcement Learning Applied to Linear Quadratic Regulation". *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, San Mateo, CA.

[17] A. E. Bryson and Y. Ho, 1975. "Applied Optimal Control : Optimization, Estimation and Control". Hemisphere Pub. Corp., New York.

[18] M. R. Broer and R. F. Zernicke, 1979. "Efficiency of human movement". W. B. Saunders Company, Philadelphia.

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

[20] B. Caprile and F. Girosi, 1990. "A nondeterministic minimization algorithm". Artificial Intelligence Memo 1254, Massachusetts Institute of Technology.

[21] F.-C. Chen and H. K. Khalil, 1992. "Adaptive Control of Nonlinear Systems Using Neural Networks". *International Journal of Control* **55**, 6:1299-1317.

[22] B. W. Choi, J. H. Won, and M. J. Chung, 1992. "Optimal Redundancy Resolution of a Kinematically Redundant Manipulator for a Cyclic Task". *Journal of Robotic Systems* **9** (4) : 481 - 503

[23] D. R. Cox and E. J. Snell, 1989. "Analysis of Binary Data." 2nd Ed. Chapman and Hall, New York, N.Y.

[24] L. de Biase and F. Frontini, 1978. "A Stochastic Method for Global Optimization". *Compstat*, p. 355-361.

[25] L. P. Devroye, 1978. "The Uniform Convergence of Nearest Neighbor Regression Function Estimators and Their Application in Optimization" *IEEE Transactions on Information Theory* **IT-24** (2):142-151.

[26] L. P. Devroye, 1987. "A course in density estimation". Progress in Probability and Statistics, Vol. 14. Birkhauser, Boston.

[27] P. Dyer and S. R. McReynolds, 1970. "The Computation and Theory of Optimal Control" Academic Press, New York and London.

[28] A. A. Feldbaum, 1965. "Optimal Control Systems", Academic Press, New York.

[29] R. Fletcher, 1965. "Function minimization without evaluating derivatives". *The computer journal*, 8:33-41 (April).

[30] R. Franke, 1982 "Scattered data interpolation: tests of some methods". *Math. Comp.*, 38:181–200.

[31] I. M. Gelfand and M. L. Tsetlin, 1971. "Mathematical modeling of mechanisms of the central nervous system". In *Models of the structural-functional organization of certain biological systems*, Gelfand, Gurfinkel, Fomin and Tsetlin (eds.), MIT Press, Cambridge, MA.

[32] F. Girosi, 1992. "Some Extensions of Radial Basis Functions and their Applications in Artificial Intelligence". *Computers and Mathematics with Applications*, Vol 24, No. 12 : 61 - 80.

[33] F. Girosi and T. Poggio, 1989. "Networks and the best approximation property" MIT AI Memo 1164.

[34] F. Girosi, T. Poggio and B. Caprile, 1990. "Extensions of a theory of networks for approximation and learning : outliers and negative examples". MIT AI Memo 1220.

[35] C. J. Goh, 1993 "On the Nonlinear Optimal Regulator Problem" *Automatica* **29**(3) : 751 - 756

[36] D. E.Goldberg, 1989. "Genetic algorithms in search, optimization, and machine learning" Addison-Wesley Pub. Co, Reading, MA.

[37] A Guez, I. Rusnak and I. Bar-Kana, 1992. "Multiple Objective Optimization Approach to Adaptive and Learning Control". *International Journal of Control,* **56**(2):469-482.

[38] V. Gullapalli, 1990. "A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions". *Neural Networks,* 3: 671 - 692.

[39] E. Hartman and J. D. Keeler, 1991. "Predicting the Future: Advantages of Semilocal Units". *Neural Computation* 3 : 566 - 578.

[40] N. Hogan and T. Flash, 1987. " Moving gracefully: quantitative theories of motor" coordination. *TINS* **10**(4): 170-174

[41] K. J. Hunt, D. Sbarbaro, R. Żbikowski and P. J. Gawthrop, 1992. "Neural Networks for Control Systems - A Survey". *Automatica*, 28(6):1083-1112.

[42] J. M. Hutchinson, 1993. "A radial basis function approach to financial time series analysis". Ph.D thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[43] M. Hutchinson,J. Kalma and M. Johnson, 1984. "Monthly estimates of windspeed and wind run for Australia". *J. Climatology*, 4:311-324.

[44] M. Ito, 1993. "New concepts in cerebellar function" *Revue de Neurologie (Paris)*, 149(11):596-599.

[45] I. R. H. Jackson, 1988. "Convergence Properties of Radial Basis Functions". *Constructive Approximation* 4:243-264.

[46] R. A. Jacobs and M. I. Jordan, 1991. "A competitive modular connectionist architecture". In *Advances in Neural Information Processing Systems 3*, R. P. Lippman, J. E. Moody and D. S. Touretzky, eds., Morgan Kaufmann, San Mateo, Ca.

[47] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton, 1991. "Adaptive Mixtures of Local Experts". *Neural Computation* 3: 79-87.

[48] C. R. Jensen and G. W.Schultz, 1977. "Applied Kinesiology" McGraw Hill book Company.

[49] M. I. Jordan, 1989. "Indeterminate Motor Skill Learning Problems". In M. Jeannerod (ed.) *Attention and Performance XIII.* Hillsdale, NJ: Lawrence Erlbaum.

[50] M. I. Jordan and D. E. Rumelhart, 1992. "Forward Models : Supervised Learning with a Distal Teacher". *Cognitive Science* 16 : 307 - 354

[51] M. Kawato, Y. Maeda, Y. Uno and R. Suzuki, 1990. ' 'Trajectory Formation of Arm Movement by Cascade Neural Network Model Based on Minimum Torque Change Criterion." *Biological Cybernetics*, 62: 275 - 288.

[52] M. Kawato and H. Gomi, 1993. "Feedback-error-learning model of cerebellar motor control". In*Role of the Cerebellum and Basal Ganglia in Voluntary Movement*, N. Mano, I. Hamada and M. R. DeLong editors. Elsevier Science Publishers.

[53] M. Kawato and H. Gomi, 1992. "A computational model of four regions of the cerebellum based on feedback-error learning" *Biological Cybernetics* 68:95-103.

[54] D. E. Kirk, 1970. " Optimal control theory; an introduction". Englewood Cliffs, N.J., Prentice-Hall

[55] V. I. Krinskii and M. L. Shik, 1964. "A simple motor task". *Biophysics*, 9:661-666

[56] J. G. Kuschewski, S. Hui and S. H. Żak, 1993. "Application of Feedforward Neural Networks to Dynamical System Identification and Control". *IEEE Transactions on Control Systems Technology* **1**, No. 1:37-49.

[57] M. Lan and S. Chand, 1990. "Solving Linear Quadratic Discrete-Time Optimal Controls Using Neural Networks", *Proceeding of the 29th Conference of Decision and Control*, Vol. 5, 2770-2772.

[58] Ker-Chau Li, 1992. "On Principal Hessian Directions for Data Visualization and Dimension Reduction: Another Application of Stein's Lemma." *Journal of the American Statistical Association* 87 : 1025 - 1039

[59] A. U. Levin and K. S. Narendra, 1993. "Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization". *IEEE Transactions on Neural Networks* **4**, 2:192-206.

[60] D. Luenberger, 1973. "Introduction to linear and nonlinear programming". Addison-Wesley, Reading, Massachusetts.

[61] D. Luenberger, 1984. "Linear and nonlinear programming". Addison-Wesley, Reading, Massachusetts.

A. V. Lukashin and A. P. Georgopoulos, 1993. "A dynamical neural network model for motor cortical activity during movement: population coding of movement trajectories". *Biological Cybernetics*, 69:517-524

[62] K. Marzuki and S. Omatu, 1992. A Neural Network Controller for a Temperature Control System. *IEEE Control Systems Magazine* **6**:58-64

[63] B. W. Mel and S. M. Omohundro, 1991. "How Receptive Field Parameters Affect Neural Learning". In *Advances in Neural Information Processing Systems 3*, R. P. Lippman, J. E. Moody and D. S. Touretzky, eds., Morgan Kaufmann, San Mateo, Ca.

[64] M. J.Mears, R. Smith, P. R. Chandler and M. Pachter, 1993. "A Hopfield Neural Network for Adaptive Control", *AIAA Guidance, Navigation and Control Conference*, Monterey, CA.

[65] C. A. Micchelli, 1986. "Interpolation of scattered data: distance matrices and conditionally positive definite functions". *Constr. Approx.*, 2:11–22.

[66] P. .J. Millington, 1991. "Associative reinforcement learning for optimal control". Master thesis, Department of aeronautics and astronautics, MIT, Cambridge, Massachusetts.

[67] J. Mockus, 1989. Bayesian Approach to Global Optimization. Theory and Applications. Mathematics and Its Applications. Soviet Series. Kluwer Academic Publishers

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

[69] A. Moore, 1991. "Knowledge of knowledge and intelligent experimentation for learning control". *Proceedings of the 1991 International Joint Conference on Neural Networks.*

[70] S. Mori, H. Nishihara and K. Furuta, 1976. "Control of unstable mechanical system. Control of pendulum." *International Journal of Control*, **23**(5) : 673 - 692.

[71] K. S. Narendra, and M. A.L. Thathachar, 1989. "Learning automata : an introduction". Prentice Hall, Englewood Cliffs, N.J.

[72] K. S. Narendra and A. M. Annaswamy, 1989. "Stable adaptive systems". Prentice Hall, Englewood Cliffs, N.J.

[73] K. S. Narendra and S. Mukhopadhyay, 1992. "Intelligent Control Using Neural Networks". *IEEE Control Systems Magazine* **4**:11-18

[74] W. L Nelson, 1983. "Physical principles for economies of skilled movements". *Biological Cybernetics* 46:135-147.

[75] K. M. Newell and P. V. McDonald, 1992. "Searching for solutions to the coordination function: Learning as exploratory behavior". In *Tutorials in motor Behavior II*, G. E. Stelmach and J. Requin (editors).

[76] Numerical Algorithms Group, 1983. "Fortran Library Manual", Mark 11, Volume 2.

[77] Y.-H. Pao, S. M. Phillips and D. J. Sobajic, 1992. "Neural-Net Computing and the Intelligent Control of Systems". *International Journal of Control* **56**, 2:263-289.

[78] J. Peterson, 1992. "On-line Estimation of Optimal Control Sequences: Pontryagin Estimators", *International Conference on Artificial Neural Networks in Engineering,* ANNIE 92.

[79] J. Peterson, 1993. "On-line estimation of the Optimal Value Function: HJB Estimators", *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, San Mateo, CA.

[80] D. A. Pierre, 1986. "Optimization theory with applications". Dover Publications, New York.

[81] J. C. Platt and A. H. Barr, 1988. "Constrained Differential Optimization for Neural Networks". Technical Report, TR-88-17, California Institute of Technology. Pasadena, California.

[82] J. Platt, 1991. "A Resource-Allocating Network for Function Interpolation". *Neural Computation*, **3**:213-225.

[83] T. Poggio and F. Girosi, 1989. "A theory of networks for approximation and learning". MIT AI Memo 1140.

[84] T. Poggio , 1990a. "A theory of how the brain might work". MIT AI Memo 1253.

[85] T. Poggio and F. Girosi, 1990b. "Extensions of a theory of networks for approximation and learning: dimensionality reduction and clustering". MIT AI Memo 1167.

[86] A. Polit and E. Bizzi, 1979. "Characteristics of the motor programs underlying arm movements in monkeys". *Journal of Neurophysiology*, 42:183-194

[87] M. J. D. Powell, 1987. "Radial basis functions for multivariable interpolation: A review". In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 143–167. Clarendon Press, Oxford.

[88] M. J. D. Powell, 1988. "Radial Basis Function Approximations to Polynomials". In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1987*, 223 - 241, Longman Scientific and Technical.

[89] D. Psaltis, A. Sideris and A. A. Yamamura, 1988. "A Multilayered Neural Network Controller" *IEEE Control Systems Magazine* **4**:17-21

[90] A. Saha, J. Christian, D. S. Tang and C.-L. Wu, 1991. In *Advances in Neural Information Processing Systems 3*, R. P. Lippman, J. E. Moody and D. S. Touretzky, eds., Morgan Kaufmann, San Mateo, Ca.

[91] A. M. Samarov, 1991. "Exploring regression structure using nonparametric functional estimation". TR-69, Sloan School of Management, M.I.T.

[92] R. M. Sanner and J.-J. Slotine, 1992. "Gaussian Networks for Direct Adaptive Control". *IEEE Transactions on Neural Networks* **3**, No. 6:837-863.

[93] S. Sastry, and M. Bodson, 1989. "Adaptive control : stability, convergence, and robustness". Prentice Hall, Englewood Cliffs, N.J.

[94] L. E. Scales, 1985. "Introduction to non-linear optimization". Springer-Verlag, New York

[95] I. P. Schagen, 1980. "Stochastic Interpolating Functions: Applications in Optimization". *J. of Inst. Math. Appl.* **26**:93-101.

[96] I. P. Schagen, 1984. "Sequential Exploration of Unknown Multi-dimensional Functions as an Aid to Optimization". *IMA Journal of Numerical Analysis* **4**:337 - 347

[97] I. P. Schagen, 1986. "Internal Modelling of Objective Functions for Global Optimization". *Journal of Optimization Theory and Applications* **51**:345-353.

[98] W. H. Schiffman and H. Willi Geffers, 1993. "Adaptive Control of Dynamic Systems by Back Propagation Networks". *Neural Networks*, **6**:517-524

[99] M. Shidara, K. Kawano, H. Gomi and M. Kawato, 1993. "Inverse Dynamics Model Eye Movement Control by Purkinje Cells in the Cerebellum". *Nature*, **365**, Number 6441:50-52

[100] B. W. Silverman, 1986. "Density estimation for statistics and data analysis". Chapman and Hall, London, New York.

[101] J.-J. Slotine and W. Li 1991. "Applied Nonlinear Control". Prentice Hall, Englewood Cliffs, New Jersey.

[102] R. Sutton, 1988. "Learning to predict by the method of temporal differences". *Machine Learning*, 3:9-44.

[103] R. Sutton, ed., 1992. "Reinforcement Learning". Kluwer Academic Publishers, Boston, Massachusetts.

[104] E. L. Thorndike, 1927. "The law of effect". *American Journal of Psychology*, 39: 212 - 222.

[105] S. B. Thrun, 1992. "The role of exploration in learning control" In:*Handbook of Intelligent Control: Neural, fuzzy and adaptive approaches*, D. A. White and D. A. Sofge editors. Van Nostrand, Reinhold, New York.

[106] Y. Uno, M. Kawato, R. Suzuki, 1989. "Formation and Control of Optimal Trajectory in Human Multijoint Arm Movement." *Biological Cybernetics*, 61: 89 - 101.

[107] Y. Wada and M. Kawato, 1992. "A Neural Network Model for Trajectory Formation of Arm Movement by Using Forward and Inverse Dynamics Models". In *Artificial Neural Networks, 2* I. Aleksander and J. Taylor (Eds.) Elsevier Science Publishers B.V.

[108] G. Wahba, C. Gu,, Y. Wang, and R. Chappell, 1993. "Soft Classification, a. k. a. Risk Estimation, via Penalized Log Likelihood and Smoothing Spline Analysis of Variance", preprint. To appear in *the proceedings of the Santa Fe Workshop*, D. Wolpert and A. Lapedes, eds., Addison-Wesley.

[109] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart, 1990. "Predicting the future: A connectionist approach". *International Journal of Neural Systems* **1** : 193.

[110] P. J. Werbos, 1990. "Consistency of HDP Applied to a Simple Reinforcement Learning Problem" *Neural Networks* **3** : 179 - 189.

[111] Y. Xia and H. Inooka, 1992. "Application of Tree Search to the Swinging Control of a pendulum" *IEEE Transactions on Systems, Man and Cybernetics* **22** (5) : 1193 - 1198.

[112] Y. Zhao, 1992. "On Projection Pursuit Learning", Ph.D. thesis, Department of Mathematics, M.I.T.