PERFORMANCE CHARACTERIZATION AND CONTROLLER SCHEDULING FOR DYNAMIC SYSTEMS USING NEURAL NETWORKS

A DISSERTATION
SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
for the degree
DOCTOR OF PHILOSOPHY
in
ELECTRICAL ENGINEERING

by

ENRIQUE D. FERREIRA

Department of Electrical and Computer Engineering

CARNEGIE MELLON UNIVERSITY

Pittsburgh, Pennsylvania, August 1998

Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890

Typeset by the author using LaTeX 2_ε.

Performance Characterization and Controller Scheduling for Dynamic Systems Using Neural Networks

by
Enrique D. Ferreira

Submitted to the Department of Electrical and Computer
Engineering on August 3rd, 1998, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering

Abstract

In this thesis we explore the use of neural networks to evaluate the performance of nonlinear systems and to switch among a pool of controllers to achieve a desired closed-loop performance. This method is motivated on the needs of a automated way to generate controller schedules and the development of a control architecture for evolvable systems called SIMPLEX. The method is based on two neural network estimates for each controller, one for its region of stability and another for a performance index. Neurodynamic programming techniques are used for neural network training providing a way to realize the architecture in real-time. Several types of estimates are developed to handle uncertainties and disturbances. Convergence of the algorithms and confidence intervals are analyzed. The architecture and controller scheduling algorithm developed in this thesis allows us to assure stability of the closed-loop system and improve closed-loop performance by combining tested and new controller capabilities. Computational complexity is analyzed. Results are presented in a simulation example and a submerged vessel application. Future lines of research are discussed.

Thesis Supervisor: Prof. Bruce H. Krogh
Title: Professor of Electrical and Computer Engineering

Thesis Advisor: Prof. John P. Lehoczky
Title: Professor of Statistics

Thesis Advisor: Prof. William C. Messner
Title: Professor of Mechanical Engineering

Thesis Advisor: Prof. B. Erik Ydstie
Title: Professor of Chemical Engineering

Preface

Quite some time has passed since I entered Carnegie Mellon University with a Ph.D. goal in mind. I gave my best to do it right but I have gained much more than that. In this section I am trying to give something back by saying thanks to all the people I have been in touch with during this time. Here is a list of them, no order in special but no random either.

The support of my parents Dumas and Gladys, my sisters Ana and Beatriz, my brothers-in-law Martín and Pablo, my niece Florencia, my aunt Mirta and my uncles Edilio and Rómulo, even though far away, has been invaluable. My closest relatives, Marcelo, Eduardo, Luisa, Mercedes and Miguel have given me an incredible support and the opportunity to know more of the United States and its people.

To my friends Fernando and Nirvana, Mario, Bernardo and the *Minired* enginering school group for the interesting discussions and moments of joy in between the research. From my former school I have many professors I would like to thank, for what I have learned from them, and for the opportunity they gave me to come here, specially my former advisors Rafael Canetti and Ramón Sosa.

Once at CMU, I found many friends and people to whom I could discuss research as well as life issues. This thesis would have been much harder, to say the least, without the constant help from my advisor Bruce Krogh and his ability to see through all the issues. J. Lehockzy, W. Messner, E. Ydstie have contributed with useful suggestions to develop the topic. Danbing Seto, Lui Sha and all

the INSERT team at the Software Engineering Institute were very helpful in the development of the experiments.

I also appreciate the contributions of my office mates Jim Reich, Xu Cheng, Jun Li and Alongkrit Chutinan. To my other friends at CMU: Juan Huerta, Teck-Khim Ng, Marcel Bergerman, John Steele, Edwin Intong and his wife Marilou, and specially to Priyadarshee Mathur, "Pd" for everybody, for all the long hours and dinners with very interesting control discussions. I must not forget my advisor's family, Margie, Heather, Michelle and Robby for their friendship. Special thanks to Mathias Rausch and his family, Eva, Octavia and Magdalena, for being very good friends during their stay at Pittsburgh.

My appreciation to many of the staff at the ECE Department, specially Lynn Philibin, Elaine Lawrence, Debbie Scappatura and Carol Patterson for their advices and help with all the paper work. Also to Lisa Minetti from the Intercultural Communication Center and Linda Melville from the Office of International Education for their help in getting me adjusted to the language and issues related to foreign students in USA.

From outside CMU my very special thanks to my housemate Mark Gredler and his parents for getting along with me so well despite my student way of life. To John Guiver from NeuralWare Inc. for his friendship and sincere interest in my home country.

Besides research and social life the other important factor in student life is money and for that I have to thank the government of my home country Uruguay, through the Consejo Nacional de Investigaciones Científicas Y Técnicas (CONICYT), the Organization of American States and Carnegie Mellon University for their financial support to my research interests.

Contents

List of Figures

List of Tables

Notation

List of standard notation used throughout the thesis.

$\mathcal{Z}, \mathfrak{R}$	Integer and real spaces.
$\mathfrak{R}^n, \mathcal{C}^n$	Real and complex vector spaces.
D, R, S	Domain sets.
$\mathbf{x}, \mathbf{y}$	Vector signals.
$f(.), g(.)$	Scalar and Vector functions.
$W, \hat{W}$	Neural network weight matrices.
$\Phi(.), \phi(.)$	Neural networks functions.
J_δ^i	Performance index for controller i with discount factor δ.
$\Psi(.)$	Basis functions on linear approximation expansions.

Chapter 1

Introduction

1.1 Motivation

We explore the use of neural networks to evaluate the performance of autonomous systems and to switch among a pool of controllers to achieve a desired closed-loop performance.

There are several motivations for this research. Neural networks have been shown to be effective in the control of many complex real world systems because they can provide robust approximation of nonlinear functions. Neural networks can also be adaptive, keeping track of changes in the environment to help improve the overall performance of the system. Controller scheduling, mainly gain scheduling, has been used successfully for many years in practice, but the way it has been designed and applied has been almost always empirical. Neural networks offer the possibility to automate some of these empirical procedures.

Another motivation comes from the possible application of the proposed scheme to a fault-tolerant system called SIMPLEX being developed at the Software Engineering Institute at Carnegie Mellon University. SIMPLEX operates within a real-time multitasking operating system to help control a complex process using a set of controllers. All of the controllers run concurrently as separate tasks, and SIMPLEX monitors the performance of the controllers and the plant to determine which

controller should actually send control inputs to the plant. This monitoring and switching activity guarantees safety and performance, provided the switching rules are viable. The neural network approach proposed in this PhD research project may be a useful way to realize the SIMPLEX switching rules.

1.2 Thesis overview

The thesis is organized in the following way. Chapter 2 covers the work that has already been done related to the subject and describes the concepts and notation to be used throughout the rest of the thesis. Chapter 3 presents a neural network approach to estimate the region of estability of a class of dynamic systems. Chapter 4 develops several methods to estimate a cost-to-go function for a dynamic system, based on neurodynamic programming. Chapter 5 combines the results of Chapters 3 and 4 and describes a controller scheduling method, the main result of this work. Chapters 6 and 7 show simulation and experimental results on two applications, an inverted pendulum and a submerged vessel system. Finally, Chapter 8 summarizes the contributions of this thesis and points out several lines of future research. Appendix A contains the details of the more involved theorem proofs to make the flow of the ideas easier to read and understand.

Chapter 2

Background

2.1 Introduction

Throughout this century, but especially in the last decades, a great amount of work has been done in the theory of nonlinear dynamical systems and in neural network theory. This chapter concentrates on the concepts and results from these areas that are relevant to our work.

2.2 Lyapunov stability theory

Lyapunov stability theory remains the most important method to study stability for nonlinear systems and thus it is used extensively in areas such as optimal and adaptive control. There are many publications that cover Lyapunov theory (e.g. [LM94, KB60]). We give a summary of some of its concepts and important results. Consider an autonomous system described by

$$\dot{\mathbf{x}} = f(\mathbf{x}) \tag{2.1}$$

with $\mathbf{x} \in \Re^n$, $f(.)$ smooth and $f(0) = 0$. The origin is referred to as an *equilibrium point* for (2.1). The trajectory of the system (2.1) with initial state $\mathbf{x}_o$ is denoted by $\mathbf{x}(t, \mathbf{x}_o)$. First, we

introduce some definitions that will be used throughout this section and afterwards some well known definitions and theorems for Lyapunov analysis (e.g. from [SL91])..

Definition 2.1 (Class $\mathcal{K}$ function). A continuous function $\varphi : \Re^+ \to \Re^+$ is said to belong to class $\mathcal{K}$ if

(i) $\varphi(0) = 0$

(ii) φ is strictly increasing in $\Re^+$.

Definition 2.2 (Class $\mathcal{K}_\infty$ function). A class $\mathcal{K}_\infty$ function is a class $\mathcal{K}$ function which is onto $\Re^+$. Therefore, its inverse is also a class $\mathcal{K}_\infty$ function.

Definition 2.3 (Invariant Set). A set $\Omega \subseteq \Re^n$ is invariant with respect to the system (2.1) if every solution of (2.1) starting in Ω remains in Ω for all time.

Definition 2.4 (Lyapunov function). A continuous, positive definite function $V : \Re^n \to \Re$ with continuous partial derivatives is a Lyapunov function of (2.1) if its derivative along any state trajectory is negative semi-definite, i.e.,

$$\dot{V}(\mathbf{x}(t)) \leq 0, \qquad \forall t$$

Theorem 2.1 (Lyapunov's Direct Method). *System* (2.1) *is locally (globally) stable at the origin iff there exist a locally (globally and radially unbounded) Lyapunov function V for it. Furthermore, the origin is asymptotically stable iff the Lyapunov function V is strictly decreasing along any system trajectory.*

An extension of Lyapunov's Direct Method is Lasalle's theorem.

Theorem 2.2 (Invariant Set Theorem). *Given system* (2.1)*, let V be a scalar function on $\Re^n$ with continuous partial derivatives such that*

- *For some $l > 0$ the region Ω_l defined by $V(\mathbf{x}) < l$ is bounded.*

- $\dot{V}(\mathbf{x}) \leq 0 \; \forall \mathbf{x} \in \Omega_l$

Let R be the set of all points in Ω_l where $\dot{V}(\mathbf{x}) = 0$, and M be the largest *(with respect to set containment) invariant set in R. Then, every solution $\mathbf{x}(t)$ originating in Ω_l tends to M as $t \to \infty$.*

Definition 2.5 (Non-smooth Lyapunov function). A continuous, positive definite function $V : \Re^n \to \Re$ is a non-smooth Lyapunov function of an autonomous system (2.1) if it satisfies:

$$V(x(t_1)) < V(x(t_2)), \quad \forall \; t_1 > t_2$$

along any state trajectory $x(t)$ of (2.1).

Analogous results to Lyapunov and Lasalle's theorems hold for non-smooth Lyapunov functions by using the concept of generalized gradient [Cla83].

We introduce also an extended concept of stability

Definition 2.6 (Robust Uniform Asymptotic Stability). Given a system $\dot{\mathbf{x}} = f(\mathbf{x}, w)$ with $w(t) \in W \subseteq \Re^l$ for all $t \geq 0$ and a compact subset $\Omega \subset \Re^n$, define $\|\mathbf{x}\|_\Omega = inf\{\|\mathbf{x} - \mathbf{y}\|, \mathbf{y} \in \Omega\}$. Then the system is RUAS with respect to Ω or *RUAS* $-\ \Omega$ if the following conditions hold.

1. For every $\mathbf{x}(0) \in \Re^n$ and $w(t) \in W$, the solution $\mathbf{x}(t)$ is defined for all t.
2. Uniform stability: There exists a $\mathcal{K}_\infty$ function δ s.t. for any $\epsilon > 0$, $\|\mathbf{x}(t)\|_\Omega < \epsilon$ for all $t \geq 0$ whenever $\|\mathbf{x}(0)\|_\Omega \leq \delta(\epsilon)$, for all $w(t) \in W$.
3. Uniform Attraction: For any $r, \epsilon > 0$, there exists $T > 0$, s.t. for every $w(t) \in W$, $\|\mathbf{x}(t)\|_\Omega < \epsilon$ whenever $\|\mathbf{x}(0)\|_\Omega < r$ and $t \geq T$.

These concepts help in dealing with systems with uncertainty and noise which is an important issue in applications.

2.3 Stability regions for nonlinear systems

The problem of estimating the region of stability for the stable equilibria of autonomous nonlinear systems is fundamental in the theory of dynamic systems, and has been studied for many years [Zub64] [NL95]. In applications, knowledge of regions of stability is essential for the safe operation of many complex dynamic systems, such as power systems and nuclear reactors [ZHZL88]. Despite many years of theoretical attention to this problem, and its clear practical importance, the existing methods for estimating stability regions remain limited. They often can be applied only to low-dimensional systems, and they are generally very conservative.

Definition 2.7 (Region of attraction). Given the dynamic system (2.1) with equilibrium state $\mathbf{x}_e$ (i.e., $f(\mathbf{x}_e) = 0$), the region of attraction or domain of attraction for $\mathbf{x}_e$ is the set of initial conditions, $\mathbf{x}_o$, such that

$$\lim_{t \to \infty} \mathbf{x}(t, \mathbf{x}_o) = \mathbf{x}_e$$

Several methods have been proposed to compute inner (i.e., conservative) approximations to the boundary of regions a attraction (e.g., [CHW88] and references therein). All of these methods assume a model of the system dynamics is given. Consequently, for real applications the system dynamics must be known precisely or measures must be taken to make the boundary estimate conservative enough to account for model uncertainties.

Most of the proposed methods involve the computation of a Lyapunov function for the system. For example, Davison and Kurak [DK71] developed an algorithm to find a quadratic Lyapunov function that maximizes the volume of the stability region. Noldus and Loccufier [NL95] proposed a technique for improving a given Lyapunov function estimate by integrating the system dynamics backwards in time, away from the Lyapunov boundary at specific points, to enlarge the estimation. Marpaka [Mar91] proposed that for power system applications where fast real-time estimates are needed, one could train a feedforward neural network to characterize a *given* Lyapunov function.

Although Marpaka's work addresses the problem of on-line computation via a neural network, it does not advance any methods for actually obtaining the Lyapunov function itself.

In the next sections we explain with more detail two methods to compute stability regions that will be used later for comparison purposes.

2.3.1 Maximal Lyapunov functions

Vannelli and Vidyasagar [VV85] estimated the region of stability from the development of a "maximal" rational Lyapunov function for a given dynamical system (2.1) with $\mathbf{x} = 0$ an asymptotic stable equilibrium point with domain of attraction S.

Definition 2.8. A function $V_m : \Re^n \to \Re^+ \cup \{\infty\}$ is called a *maximal Lyapunov function* (MLF) for the system (2.1) with domain of attraction $S \subseteq \Re^n$ if

i. $V_m(0) = 0,\ V_m(\mathbf{x}) > 0 \quad \mathbf{x} \in S \backslash 0$

ii. $V_m(\mathbf{x}) < \infty \iff x \in S$

iii. $V_m(\mathbf{x}) \to \infty$ as $\mathbf{x} \to \partial S$ and/or $|\mathbf{x}| \to \infty$

iv. $\dot{V}_m$ is well defined and negative definite over S.

Theorem 2.3 states a necessary and sufficient condition for a set to be the domain of attraction of a system in terms of the existence of a MLF.

Theorem 2.3. *Suppose f is Lipschitz continuous on S. Then, in order for an open set A containing 0 to be the domain of attraction for* (2.1)*, it is necessary and sufficient that there exists a continuous function $V : A \to \Re^+$ and a positive definite function ϕ such that*

i. $V(0) = 0,\ V(\mathbf{x}) > 0 \quad \mathbf{x} \in A \backslash 0$

ii. The function

$$\dot{V}(x_o) = \lim_{t \to 0^+} [V(\mathbf{x}(t, x_o)) - V(\mathbf{x}_o)]/t$$

is well defined for all $\mathbf{x} \in A$ *and satisfies*

$$\dot{V}(x_o) = -\phi(\mathbf{x}) \text{ for all } \mathbf{x} \in A.$$

iii. $V(\mathbf{x}) \to \infty$ *as* $\mathbf{x} \to \partial A$ *and/or* $|\mathbf{x}| \to \infty$

Moreover, Vanelli and Vidyasagar also state a condition for the existence of a MLF.

Lemma 2.4. *Suppose* f *is Lipschitz continuous on S, and suppose there exist* $\delta > 0$ *such that* V *is a continuous function on some closed ball* $\bar{B}_\delta$ *with* $V(0) = 0$ *and* $\dot{V}$ *is negative definite over* $\bar{B}_\delta$. *Then, there exists a MLF* V_m *that agrees with* V *in* $\bar{B}_\delta$.

In [VV85] the authors develop a computational method with rational functions as MLFs and find approximations to the stability region to a certain error in terms of the order of the polynomials used in the rational expansion. This procedure relies on a model of the system for the solution and it is computationally expensive. Also, the development for higher order systems has not been done completely yet. This method is applied to the low order examples in Chapter 6 and comparisons with our work are drawn.

2.3.2 LMI Approach

Even though Linear Matrix Inequalities (LMI) have been known for decades, only in recent years they have become an important tool for a control engineer due to major advances in how to solve them. LMIs are flexible to manage many types of problems more quickly than other tools. A good reference on the possibilities of LMIs is [BGFB94]. [Ter96] covers the mathematical aspects of the interior point method, the main optimization procedure used for LMI computations.

The main disadvantage of LMIs is that they can not be applied to nonlinear systems unless everything can be set up as linear matrix inequalities, which is highly unlikely for many problems.

The approach taken by Seto et al. to overcome this limitation is fairly simple and very similar to the one by Davison and Kurak [DK71]. It consists first in solving for an ellipsoid of maximal volume that solves the Lyapunov stability equation of the linearized system, subject to possible state and input constraints. This ellipsoid acts as a first guess of the stability region. The next step is the verification that this region is indeed in the stability region of the nonlinear system by looking at the derivative vector field on the ellipsoid surface. If this is not the case the ellipsoid is shrunk until this condition is satisfied. These operations can be done very fast using LMIs and gives a very easy formula to evaluate whether a state belongs to the stability region. This method is applied to the examples in Chapters 6 and 7 and comparisons with our work are explained in each chapter.

2.4 Controller Scheduling

Controller scheduling strategies have been used for a long time. Recently, some works on controller scheduling have emerged from the adaptive control community with a combinations of ideas that relate to the work in this thesis.

Although it is strictly for linear systems, the work of Morse [Mor95, Mor96] on supervisory control contains some key ideas for any scheduling system. The ”supervisor” switches between a set of linear controllers to control a plant assumed to belong to a certain family of plants. It is assumed that for each plant in the family a stabilizing robust controller is given. The plant and controller parameterization may be countable or uncountable. The switching logic is realized by a hybrid system which computes an $\mathcal{L}_2$ performance measure for the family. The best performance decides which plant in the family is the closest at that particular instant and the controller designed for it is applied to the system. The procedure also makes use of a dwell-time, i.e., a minimum time that a controller would be applied to avoid infinite switching in a finite time span. In [Mor96] it is proved

that under a constant disturbance the supervisor is able to follow a step reference signal with no steady-state error and moreover that under piecewise constant disturbance and noise all signals in the system are uniformly bounded.

Narendra and Balakrishnan [NB97, BN95b] developed several control architectures with the goal of improving the transient response of a switching system in an adaptive control framework. The schemes proposed in [NB97] are based on a family of linear models for a linear unknown plant and a family of controllers, fixed or adaptive, plus a supervisory scheme to minimize a performance index by switching controllers. The performance index for the j^{th} model is given by

$$J_j(t) = \alpha e_j^2(t) + \beta \int_0^t e^{-\lambda(t-\tau)} e_j^2(\tau)\, d\tau, \qquad \alpha \geq 0, \beta, \lambda \geq 0$$

where $e_j(t)$ is the error between the output of the plant and the output of the j^{th} model. Stability and robustness are proven. The use of neural networks in those architectures is also assessed in [BN95a], but using them as system model estimators rather than directly for controller scheduling.

A high-level approach for scheduling controllers for a regulation problem in discrete time developed by Kolmanovsky and Gilbert [KG97] presents many similarities to this work. Basically, they address the problem of designing a supervisor to switch between N output feedback controllers based on the computation of positive invariant sets O_∞^i for the closed-loop system generated by applying controller i to the plant. O_∞^i si defined in the augmented state-space for the controllers. A block diagram of the approach is shown in Fig. 2.1. The controllers are ranked from 1 to N by preference, 1 being the less preferable and N the most desirable controller to use. An important concept in the design is the set of safe indices $I(\mathbf{x}_p)$ that groups the indices of the controllers that are within their safe set $\Pi^i G^i$ at the plant state $\mathbf{x}_p$ with $G^i \subset O_\infty^i$ and Π^i being the projector operator of the augmented state space for each controller over the plant state space.

Definition 2.9 (Switching rule.). Set $i(0) = \max I(\mathbf{x}_p(0))$. For $t > 0$ apply the following procedure: if $I(\mathbf{x}_p(t)) = \emptyset$ or $\max(I(\mathbf{x}_p(t)) < i(t-1)$, set $i(t) = i(t-1)$; else, set $i(t) = \max I(\mathbf{x}_p(t))$.

Theorem 2.5 states the main result in [KG97].

Theorem 2.5. *Assume* $G^i \subset O^i_\infty,\ i = 1, \ldots, N$, *and the initial state* $\mathbf{x}_p(0) \in \bigcup_{i=1,\ldots,N} \Pi^i G^i$. *Let* $i(t)$ *and* $u(t)$ *be defined respectively by Def. 2.9 and*

$$\begin{aligned} \mathbf{x}_c^{i(t)}(t) &= f^{i(t)}(\mathbf{x}_c^i(t-1), \mathbf{y}(t-1)), \quad i(t) = i(t-1),\ t > 0 \\ &= Z^{i(t)}(\mathbf{x}_p(t)), \quad t = 0, \text{ or } i(t) \neq i(t-1), \\ u(t) &= h^{i(t)}(\mathbf{x}_c^{i(t)}(t), \mathbf{y}(t)) \end{aligned}$$

with the function Z^i *satisfying* $(\mathbf{x}_p, Z^i(\mathbf{x}_p)) \in G^i$. *Then* $i : \mathcal{Z}^+ \to \{1, \ldots, N\}$ *is nondecreasing. Further, if for each* $i \in \{1, \ldots, N-1\}$ *there exists* $j > i$ *and* $T_i \in \mathcal{Z}^+$ *s.t.* $\Pi^i H^i(T_i) \subset \Pi^j G^j$ *with* $H^i(T) \subset O^i_\infty$ *defined by*

$$\begin{gathered} (\mathbf{x}_p^i(t, \mathbf{x}_p, \mathbf{x}_c^i, w),\ \mathbf{x}_c^i(t, \mathbf{x}_p, \mathbf{x}_c^i, w)) \in H^i(T) \\ \forall (\mathbf{x}_p, \mathbf{x}_c^i) \in O^i_\infty,\ w \in W,\ t \geq T \end{gathered}$$

Then, $i(t) \to N$.

In words, Theorem 2.5 states that under the assumption of being able to calculate the safe indices for each state and the connectivity of the corresponding safe sets for the available controllers, the switching rule defined in Def. 2.9 guarantees that the system will eventually switch to the N^{th} controller which by assumption was the most desirable controller.

Further developments for the case of linear systems are described in [KG97] but a deeper analysis needs to be done for the case of nonlinear systems where the implementation of the rule 2.9 may require a complete knowledge of the system as well as strong computational resources. Another point to mention is that this switching rule is based completely upon the assumption the system is stable for each controller. The performance factor is introduced in the ordering of the controllers.

2.4.1 Lyapunov theory approaches

Lyapunov theory remains the most powerful tool for analysis of nonlinear systems and it has been applied to the case of switching systems. Malmborg et al. [MBA96] consider the case of having

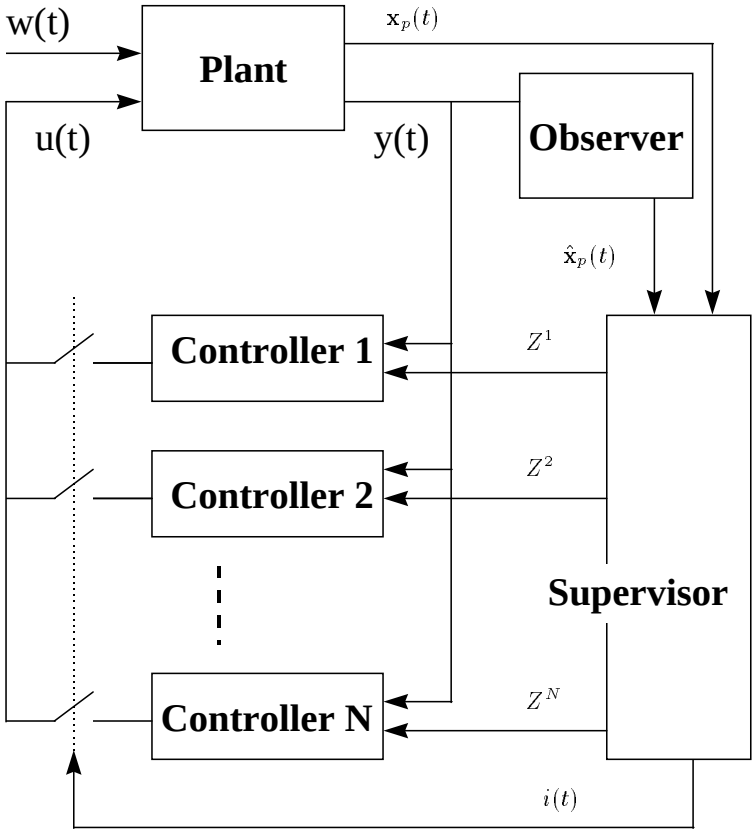

Fig. 2.1: Block diagram of the multimode regulator approach by Kolmanovsky and Gilbert.

multiple controllers for a given plant. Their principal purpose is to design a switching strategy that guarantees the closed-loop stability through the use of Lyapunov functions available for each controller (that is, available for each closed-loop system that results from the application of each controller independently). State and control constraints are not dealt with explicitly in the switching strategy. Presumably the control constraints are taken into account in the design of the controllers. Their *min-switching* strategy chooses the control(s) corresponding to the least-valued Lyapunov

function(s). This creates implicitly a (non-smooth) Lyapunov function for the closed loop system from the set of Lyapunov functions for the individual controllers. This control scheme serves as a principal motivation for the performance-based switching strategy proposed in this thesis. Formally, the switching strategy is given as follows.

Definition 2.10 (Min-Switching strategy). Given a nonlinear dynamical system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, $\mathbf{x} \in \mathfrak{R}^n$, $\mathbf{u} \in \mathfrak{R}^m$, and a set of control laws $\mathbf{u}_i(\mathbf{x})$, $i = 1, \ldots, M$ and a set of performance index functions $I_i : \mathfrak{R}^n \rightarrow \mathfrak{R}, \quad i = 1, \ldots, M$ that characterize the control laws, the Min-Switching strategy is defined as the application of the control signal $\mathbf{u}^*(t) = \mathbf{u}_i(\mathbf{x}(t)), i = arg\ \underset{j}{min}\{I_j(\mathbf{x}(t))\}$.

Theorem 2.6 presents the main result in [MBA96].

Theorem 2.6. *Given a dynamical system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, $\mathbf{x} \in \mathfrak{R}^n$, $\mathbf{u} \in \mathfrak{R}^m$, suppose the given controllers $\mathbf{u} = c_i(\mathbf{x})$ stabilize the system in open domains $\Omega_i \subset \mathfrak{R}^n$, and for each pair (c_i, Ω_i) there exists a C^1 Lyapunov function $V_i(\mathbf{x})$. Assume the collection of sets Ω_i covers $\mathfrak{R}^n$. Furthermore, at any forced transition from c_i to c_j the corresponding Lyapunov functions are equal, i.e. $V_i(\mathbf{x}) = V_j(\mathbf{x})$. Then, the min-switching rule defined by the set of controllers and using their associated Lyapunov functions as their performance index functions stabilize the system and $W = min(V_1, \ldots, V_n)$ constitutes an associated non-smooth Lyapunov function for the overall system.*

Remark 2.1. From the proof given in [MBA96] we can see that the system may switch smoothly from one controller to the other, but in certain cases it may slide on the boundary between two or more controllers having the same value for the given Lyapunov function V_i. Although still stable, that will give undesired chattering.

For systems in which the nonlinearities are a few and can be identified, McConley et al. [MADF97] developed a robust scheduling approach using control Lyapunov functions (CLFs) to solve the tracking problem. The CLFs provide the means of doing the switching between a sequence of setpoints along a desired trajectory.

2.5 Hybrid systems

A gain scheduling system can be thought of as a hybrid system, that is, a system with both continuous and discrete state variables. The states of the plant and controllers evolve in a continuous state space whereas the decision strategy introduces a discrete state. The dynamics of the continuous-state trajectory changes each time the decision strategy switches to a different controller. Using a typical model for hybrid system dynamics, a switching control system can be described by the equations

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}(t), q(t)) \\ q(t) &= \Phi(\mathbf{x}(t), q(t^-)) \end{aligned}$$

where the continuous state vector $\mathbf{x}(t)$ includes the states of all controllers as well as the plant, and the discrete state $q(t)$ represents the index of the controller selected at each instant.

Among the issues that are current research topics in hybrid systems theory, stability is the most relevant to this work. Branicky [Bra94a, Bra94b] developed mathematical results for analysis of hybrid systems. In particular, [Bra94b] covers the stability of a class of hybrid systems called *switched systems* given by

$$\dot{\mathbf{x}} = f_i(\mathbf{x}(t)), \quad i \in \{1, 2, \ldots, N\}, \tag{2.2}$$

with $\mathbf{x} \in \Re^n$, $f(.)$ Lipschitz, and the further assumption that the selection of the indices i is such that there are finite number of switches in finite time. Defining the switching sequence $S = (i_o, t_o), (i_1, t_1) \ldots$ and $S|i$ as the sequence of t_k such that $i_k = i$, Branicky introduces the following definitions

Definition 2.11. Given a strictly increasing sequence of times $T = t_o, t_1, \ldots$, the *interval completion* $\mathcal{I}(T)$ is defined by

$$\mathcal{I}(T) = \bigcup_{j \in \mathcal{Z}+} (t_{2j}, t_{2j+1})$$

and the *even sequence* $\varepsilon(T)$ as

$$\varepsilon(T) = \{t_{2j} : j \in \mathcal{Z}^+\}$$

Definition 2.12 (Lyapunov-like functions). Given a strictly increasing sequence of times T in $\Re$ ($\mathcal{Z}$) we say that V is a Lyapunov-like function for function f and trajectory $\mathbf{x}(.)$ ($\mathbf{x}[.]$) over T if

- $\dot{V}(t) \leq 0$ ($V(\mathbf{x}[t+1]) \leq V(\mathbf{x}[t])$) for all $t \in \mathcal{I}(T)$ ($t \in T$).
- V is monotonically nonincreasing on $\varepsilon(T)$ (T).

Theorem 2.7. *Suppose we have candidate Lyapunov functions $V_i, i = 1, 2, \ldots, N$, for* (2.2) *with $f_i(0) = 0$ for all i. Let S be the set of all switching sequences associated with the system. If for each $s \in \mathcal{S}$ we have that for all i, V_i is Lyapunov-like for f_i and $\mathbf{x}_S$ over $S|i$, then the system is stable in the sense of Lyapunov.*

Caines and Ortega [CO96] have worked with a similar type of systems they have called *Differential Hybrid Systems* and have assessed stability by looking at the existence of a so-called π-Lyapunov function.

As with Lyapunov theory for standard continuous-state systems, the difficulty on applying Theorem 2.7 to prove stability or to implement the switching strategy is to find the candidate Lyapunov functions. For hybrid systems with continuous dynamics that are linear or affine, numerical methods have been developed to construct piecewise quadratic Lyapunov functions using LMIs [PL96, JR96]. Stability is proven for piecewise linear systems but no performance measures are computed yet.

2.6 Neural networks

In the following section we describe the basics of neural networks. It is not the purpose to do a detailed survey but to introduce the concepts and results that are used in this dissertation. We

examine first the principal characteristics of neural networks and give a more detail description of some special procedures directly relevant to this work.

2.6.1 Neural network architectures

With the development of this area an enormous number of architectures have been designed to deal with different theories and/or applications.

Multilayer network

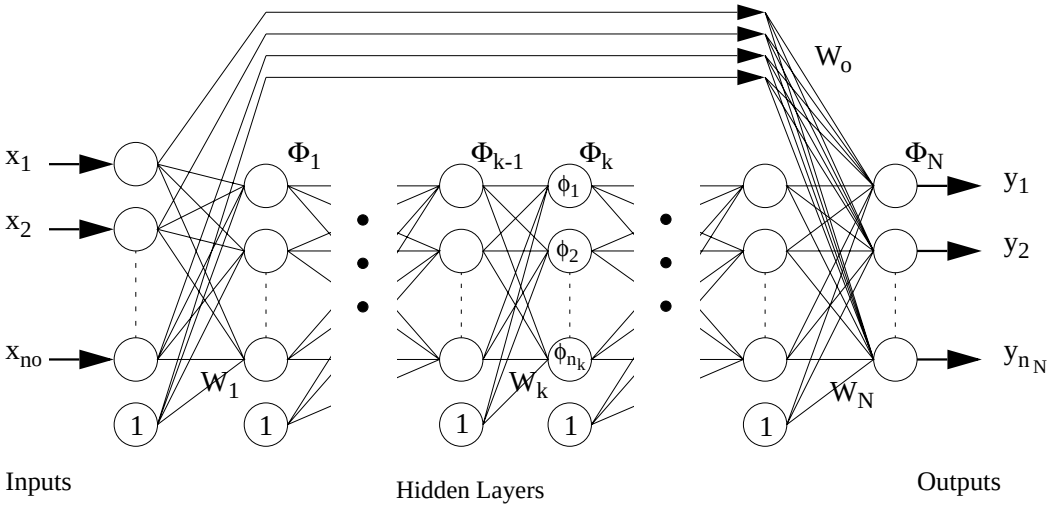

Fig. 2.2: Typical multilayer neural network architecture.

A typical structure for a multilayer network is shown in Fig. 2.2. Multilayer networks are memoryless nonlinear mappings. Neural network units are grouped into ordered layers. We use the following notation: let N be the number of layers of units from input to output. Each layer k is composed of n_k units. We use n_o to note the number of inputs to the network. Let $\mathbf{x} \in \Re^{n_o}$ be the vector of inputs to the neural network and $\mathbf{y} \in \Re^{n_N}$ the output vector. The output of the k^{th} layer

is represented by the following expressions

$$\begin{aligned} \mathbf{s}_k &= \Phi_k(\mathbf{x}) \\ &= \phi_k(W_k^T \mathbf{s}_{k-1}), \quad 1 \leq k < N \\ \mathbf{y} &= \mathbf{s}_N = W_o^T \bar{\mathbf{x}} + W_N^T \mathbf{s}_{N-1} \end{aligned}$$

where $W_k \in \Re^{(n_{k-1}+1)\times n_k}$, $\mathbf{s}_k \in \Re^{n_k+1}$, and

$$\begin{aligned} \Phi_o(\mathbf{x}) &= \bar{\mathbf{x}} \\ \bar{\mathbf{x}}^T &= [1 \quad \mathbf{x}^T] \\ \phi_k^T(\mathbf{x}) &= [1 \quad \phi_{k,1}(x_1) \ldots \phi_{k,n_k}(x_{n_k})] \end{aligned}$$

The functions $\phi_{k,i} : \Re \to S \subseteq \Re$, called activation functions, are usually bounded monotonic non-decreasing functions like the *signum* function, linear with saturation or sigmoidal type functions. All the units in the same layer have usually the same activation functions but they may differ with respect to other layers in the network. The layers and units in them are usually called input, hidden and output, according to their position in the structure of the network.

Kolmogorov's theorem (cited in [Bis95]) implies a two-hidden layer neural network can represent any continuous mapping as accurately as desired as long as strictly increasing activation functions and a sufficient number of units are used.

Recurrent network

For more complex problems, e.g. system identification, multilayer neural networks are many times insufficient [LN96] due the lack of memory. Recurrent neural networks fill this gap. Actually, recurrent neural networks are often considered as a generalization of multilayer networks. For this type of networks there is no restriction in their interconnectivity: any unit output can contribute to the input signal of any other unit. In particular, the possibility of having the output unit signals

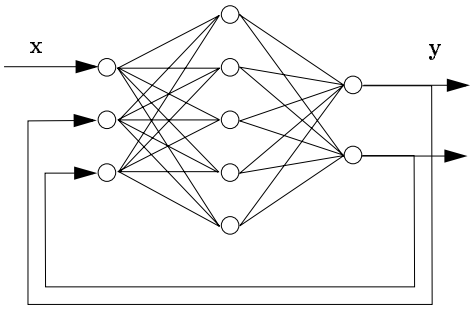

Fig. 2.3: A recurrent neural network structure.

fed back to the input layer is the most common structure, as shown in Fig. 2.3. We can still make the differentiation between input, hidden and outputs layers according to their interaction with the environment but interconnections can happen between any two units in any two layers.

System Identification is the major area in which recurrent neural networks have been applied. Narendra [NP90] developed architectures and algorithms for several system identification models. These models have had wide acceptance in certain areas like chemical systems and process control where lack of accurate first principle models, or the use of strictly linear models had made the industry to rely only on costly experimental exploration and trial and error.

2.6.2 Neural network training

Training usually means the methods used to adapt the weights W of a neural network structure to be able to reproduce a desired mapping or behavior. Training methods can be classified in three different types:

Supervised. The most used method of the three, this refers to the direct use of input-output patterns we wish the neural network to reproduce to adapt the parameters of the network. Those patterns are used to generate an output error function parameterized on the network weights.

Many procedures are developed to minimize this error function. It is the typical mapping training method, very much used in pattern recognition tasks [Bis95].

Unsupervised. This training method, mainly used in classification problems, is the opposite of the previous one discussed. It was first developed with the idea of self organization of a neural network by Kohonen [Koh89]. Clustering the input data is the main goal for feature extraction and dimensionality reduction to help organizing the input data. In this method, only a set of input vectors is presented to the network and the training methods will set the outputs according to some predefined measure of proximity of the input patterns. In a self-organizing map [Koh89] the network architecture is that of a one-layer network and the training is performed such that the inputs are associated with a specific output unit or a group of units. On presenting the trained network with a particular input, only a unit or group of units would be activated, denoting that the input belongs or is closer to the cluster represented by that unit or groups of units.

Reinforcement. Reinforcement learning algorithms are useful when the information provided to the adaptation procedure is based on performance, not on the desired outcome [Wer91]. Fig. 2.4 shows an scheme of what reinforcement learning is. The algorithm implements an estimation of the correlation between the outcomes and the weights of the network. An error between the desired outcomes and the network prediction is then expressed as a function of the weights and minimized. Given the state of the environment $\mathbf{x}(t)$ and a reference signal $\mathbf{s}(t)$, the reinforcement system gives an action to be performed $\mathbf{u}(t)$ on the environment. In a supervised learning procedure, samples of both $\mathbf{x}(t)$ and $\mathbf{u}(t)$ would be given to the network for training. In this method the feedback on the performance of the actions applied is given by the reinforcement signal $\mathbf{r}(t)$. The reinforcement learning algorithm uses the reinforcement signal to correlate the current value $\mathbf{r}(t)$ with past actions applied and to estimate the future performance of the closed-loop system. The weights of the neural network that determine the actions are then modified to maximize the performance estimate.

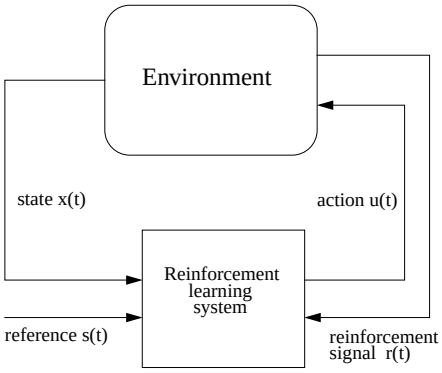

Fig. 2.4: Reinforcement learning scheme.

2.6.3 Neuro-dynamic programming

As the term suggests, neuro-dynamic programming refers to the use of neural networks in dynamic programming approaches. A deep analysis has been developed recently by Bertsekas and Tsitsiklis [BT96] in a stochastic setting. In the following we will summarize the main ideas. The dynamic programming problem to solve consists in finding the optimal control for a markov process either in a finite -or infinite- state system. In a finite-state application, the term optimal refers to the minimization of the cost-to-go function for each state i defined in terms of the cost of the transitions between states given by

$$J^{\pi}(i) \quad = \quad \liminf_{N\to\infty} E\left[\sum_{k=0}^{N-1} \alpha^k g(i_k, \mu_k(i_k), i_{k+1}) \middle| i_o = i\right] \tag{2.3}$$

where the function $g(.)$ is the transition cost function and $E[.|.]$ is the conditional expected value. The policy $\pi = \{\mu_o, \mu_1, \dots\}$ is a sequence of functions μ_k mapping states into controls with $\mu_k(i)$ belonging to a finite set of controls $U(i)$. Of particular interest are the set of *stationary proper* policies. For a stationary policy the mapping functions $\mu_k = \mu$ for all k. A stationary policy μ is proper if, using this policy there is positive probability that the termination state will be reached after at most n stages, regardless of the initial state. The solution of the problem, i.e. the optimal

policy π^* is given by Bellman's equation

$$J^*(i) \quad = \quad \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(g(i,u,j) + \alpha J^*(j)), \tag{2.4}$$

with p_{ij} being the transition probability elements. Dynamic programming solves this problem by going backwards in time but that is not possible in many applications. Sutton [Sut88] developed a method that makes it possible to solve the dynamic programming problem in a forward manner using approximations to the cost-to-go functions. The term neurodynamic comes from the fact that neural networks have been used to approximate cost-to-go functions but any other nonlinear approximators may be used.

Temporal differences method

The main purpose of the seminal work due to Sutton [Sut88] was to generate an algorithm to predict the outcome of a function dependent on a time sequence. Let x be the time sequence and $P(x_t, w)$ be the outcome estimate, dependent on the sequence value x_t at time t and unknown parameters w. Sutton defined the TD(λ) procedure to update the parameters w as

$$w_{t+1} \quad = \quad w_t + \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \tag{2.5}$$

Several authors have continued working on proving and applying the capabilities of the TD(λ) procedures [Day92, TR97].

In applying TD(λ) algorithms to the estimation of cost-to-go functions, the convergence of the method is demonstrated in [BT96] when the estimation function is linear in the unknown parameters. More specifically, if the estimator is written as

$$\hat{J}(i,r) \quad = \quad r^T \psi(i) \tag{2.6}$$

where r is the parameter vector and ψ are the basis functions used, the *temporal difference* d_t is defined as

$$d_t = g(i_t, i_{t+1}) + \alpha \hat{J}(i_{t+1}, r_t) - \hat{J}(i_t, r_t) \tag{2.7}$$

The update equation for the parameters is then

$$r_{t+1} = r_t + \gamma_t d_t \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \psi(i_k) \tag{2.8}$$

or in other way

$$\begin{aligned} z_t &= \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \psi(i_k) \\ r_{t+1} &= r_t + \gamma_t d_t z_t \\ z_{t+1} &= \alpha\lambda z_t + \psi(i_{t+1}) \end{aligned} \tag{2.9}$$

Theorem 2.8. [BT96] *Under the following assumptions,*

1. *The markov chain i has a unique invariant distribution π.*
2. *The transition cost satisfy $E[g(i_t, i_{t+1})] < \infty$ where $E[.]$ denotes expectation under π.*
3. *The matrix $\Phi = \{\psi(i)\}$ has full row rank.*
4. *Each element of Φ satisfy $E[\Phi_{ij}^2] < \infty$.*
5. *The adaptive coefficient sequence satisfies*

$$\sum_{t=0}^{\infty} \gamma_t = \infty, \qquad \sum_{t=0}^{\infty} \gamma_t^2 < \infty$$

The following holds:

1. *For any λ in $[0, 1]$ the TD(λ) algorithm* (2.9) *converges with probability one.*

2. *The limit of convergence r^* is the unique solution of the equation*

$$\Pi T^{(\lambda)}(\Phi^T r^*) = \Phi^T r^*$$

3. *Furthermore,*

$$\|\Phi^T r^* - J^*\|_D \leq \frac{\alpha(1-\lambda)}{1-\alpha\lambda}\|\Pi J^* - J^*\|_D$$

where D is the diagonal matrix with entries $\pi(i)$ and Π is the projection on the basis function subspace given by

$$\Pi = \Phi^T(\Phi D \Phi^T)^{-1}\Phi D$$

Bertsekas and Tsitsiklis noted the importance of on-line sampling in convergence of the algorithm. The fact is that convergence depends on the transition probability used in the algorithm and it is shown that divergence may occur if something other than the real probability transition matrix is used. One way of assuring the use of the real probability distribution is by doing on-line sampling. No general results are shown on nonlinear-in-the-parameters approximators though.

2.6.4 Q-learning

Q-learning was developed by Watkins in his Ph.D thesis in Cambridge University [Wat89]. It is an incremental method to solve the dynamic programming problem. The method is based the so-called *Q* function which is used as a means of obtaining the optimal control strategy. Using the same notation as in § 2.6.3 the Q-factors are defined as

$$Q^*(i,u) = \sum_{j=0}^{n} p_{ij}(u)(g(i,u,j) + J^*(j)) \tag{2.10}$$

In terms of the Q-factors it is easy to rewrite Bellman's equation

$$J^*(i) = \min_{u \in U(i)} Q^*(i,u) \tag{2.11}$$

The main result can be stated in the following way

Theorem 2.9. [BT96] *Given*

$$\sum_{t=0}^{\infty} \gamma_t = \infty, \qquad \sum_{t=0}^{\infty} \gamma_t^2 < \infty, \qquad \forall i, u \in U(i)$$

and the Q-learning method defined as

$$\begin{aligned} Q_{t+1}(i,u) &= (1-\gamma_t(i,u))Q_t(i,u) + \\ &\quad \gamma_t(i,u)\left(g(i,u,\bar{i}) + \min_{v \in U(i)} Q_t(\bar{i},,v)\right) \\ Q(0,u) &= 0, \qquad \forall t \end{aligned} \tag{2.12}$$

with $\bar{i}$ picked randomly with probability $p_{i\bar{i}}(u)$. Then, $Q_t(i,u)$ converges with probability 1 to $Q^*(i,u)$ for every i and u if all policies are proper.

2.6.5 Confidence intervals for neural network approximation

After using neural networks for estimation of a function it is important to assess the quality of the estimate. A first measurement of quality is already performed during training by looking at the error on the training and validation sets. A second approach is to use statistics to compute confidence intervals for the network outputs. In general, we have to look into the nonlinear parameter estimation and detection theory. Usual methods applied are Least squares, Maximum likelihood and Bayesian estimation [Bar74, Tre68].

A nonlinear least squares approach to neural network estimation has been proposed by Chrysolouris et al. [CLR96] as follows. Consider a multilayer neural network to estimate an input-output mapping h from noisy data

$$y_{obs} = \phi(\mathbf{x}, W^*) + \epsilon_1 + \epsilon_2$$

where ϕ represents the neural network function, W^* the optimum weights of the network, and ϵ_1, ϵ_2 refer to modeling and observation errors, respectively, which are assumed uncorrelated, independent

and identically distributed gaussian random vectors. The confidence interval $100 * (1 - \alpha)$ for a predicted value $\hat{y}$ can be computed as

$$\begin{aligned} \Delta y &= \hat{y} \pm t_{n-p}^{\alpha/2} (s_1^2 + (s_1^2 + \sigma_2^2) \hat{J}^T (\hat{G}^T \hat{G})^{-1} \hat{J})^{\frac{1}{2}} \\ s^2 &= \frac{\|y - \phi(\mathbf{x}, \hat{W})\|^2}{n - p} \\ s_1^2 &= s^2 - \sigma_2^2 \\ \hat{J} &= \frac{\partial \phi(\mathbf{x}, \hat{W})}{\partial \hat{W}_j}, \quad j = 1, \ldots, p \\ \hat{G} &= \frac{\partial \phi(\mathbf{x}_i, \hat{W})}{\partial \hat{W}_j}, \quad i = 1, \ldots, n; \quad j = 1, \ldots, p \end{aligned} \tag{2.13}$$

where $t_{n-p}^{\alpha/2}$ is the Student distribution, n is the number of data points used for estimation, p is the number of parameters W in the network and σ_2^2 is the variance of the output measurement noise.

A Bayesian solution has been presented in Bishop [Bis95]. Consider a multilayer neural network to estimate an input-output mapping h from noisy data

$$y_{obs} = \phi(\mathbf{x}, W^*) + \epsilon$$

where ϕ represents the neural network function, W^* the optimum weights of the network, and ϵ is the output error. Assuming a gaussian error model and a gaussian prior distribution of weights, the maximum likelihood solution $\hat{W}$ minimizes the squared error

$$S(W) = \beta \sum_{i=1}^{n} (y_i - \phi(\mathbf{x}_i, W))^2 + \alpha \sum_{i=1}^{p} W_i^2$$

where $T = (\mathbf{x}_i, y_i)$ is the training set, n is the number of data points used for estimation and p is the number of parameters W in the network. The posterior distribution of outputs for a given input

and given training set T is gaussian and can be expressed as

$$\begin{aligned} p(y|\mathbf{x},T) &\sim \mathcal{N}(\hat{y},\sigma_t^2) \\ \hat{y} &= \phi(\mathbf{x},\hat{W}) \\ \sigma_t^2 &= \sigma^2 + \hat{J}^T\hat{H}^{-1}\hat{J} \\ \hat{J} &= \frac{\partial\phi(\mathbf{x},\hat{W})}{\partial\hat{W}_j}, \quad j=1,\ldots,p \\ \hat{H} &= \frac{\partial^2\phi(\mathbf{x},\hat{W})}{\partial\hat{W}^2} \end{aligned} \tag{2.14}$$

where σ^2 is the variance of the gaussian noise model. Both of the above results are based on a linearization of the approximation around the solution. A comparison on these approaches shows that equations (2.13) rely only on the computation of the Jacobian of the network with respect to the weights while equations (2.14) need the Jacobian as well as the Hessian matrix to compute the confidence intervals.

2.7 The SIMPLEX architecture

One of the principal motivations for developing the controller switching strategy presented in this thesis is to provide a method for implementing the switching rules in the SIMPLEX Architecture, a technology developed at the Software Engineering Institute at Carnegie Mellon University to support safe, reliable on-line upgrades and modifications to real-time control systems [Sha95]. Fig. 2.5 shows a typical configuration for SIMPLEX in which there are three controllers: a *safety* controller, a reliable *baseline* controller, and the *experimental* controller representing the new untested control module. The basic idea of the SIMPLEX system is to guarantee that the baseline controller performance is maintained if there are problems in the experimental controller. This is accomplished by monitoring the control outputs and system performance when the experimental controller is installed, and switching control back to the baseline controller if problems are detected. The safety

controller is invoked when it is necessary to take more extreme action to return the system to the operating region for the baseline controller. The SIMPLEX Architecture has been applied successfully

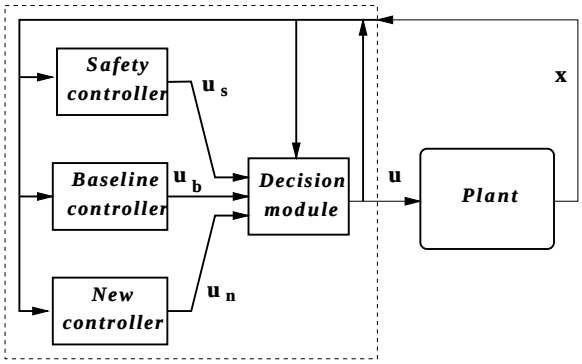

Fig. 2.5: SIMPLEX architecture.

to the control of several physical systems including single and multiple inverted pendulum systems, a plasma deposition system, and a level-control system for a submerged vessel – the example presented below. From a control perspective, SIMPLEX corresponds precisely the multi-controller scenario considered in this paper. The controllers are designed for a given plant with similar control objectives, in most cases to stabilize the system, but with different regions of stability and different performance characteristics. Clearly the ability for the SIMPLEX system to provide the desired protection against errors in the experimental controller depends entirely on the rules used to switch to the safety and baseline controllers. These rules are very difficult to create and maintain, even for small systems. In the current applications of SIMPLEX, the switching rules have been developed using physical insight and extensive experimentation resulting in ad hoc and conservative characterizations of the regions of stability for the safety and baseline controllers.

The neural network approach proposed in this dissertation provides a means of obtaining less conservative estimates of the stability regions for the controllers, and also a method for determining when to switch from the safety controller back to the baseline controller based on estimates of their performance. The on-line learning capabilities of the neural networks also suggest the possibility

of acquiring knowledge of the performance of an experimental controller, leading to the eventual transition of the experimental controller to the new baseline controller once it has been completely tested and verified.

Chapter 3

Neural networks estimates of regions of stability

3.1 Introduction

In this chapter we present a new method to estimate the region of attraction of a stable equilibrium for an autonomous nonlinear system using a neural network.

A formulation of the problem and the notation used is introduced in § 3.2. The neural network approximation is presented in § 3.3 while the main theoretical results are presented in § 3.4. Computational issues are presented in § 3.5. Discussion of the results of the method on standard examples and comparisons with previous methods presented in Chapter 2 are shown in § 3.6.

3.2 Problem formulation

Let the state-constrained nonlinear autonomous system be:

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k) \\ x &\in D \subseteq \Re^n \\ f(0) &= 0 \end{aligned} \tag{3.1}$$

with $\mathbf{x} \in \Re^n$, $\mathbf{x} = 0$ being an asymptotically stable equilibrium point. The nonlinear function f is assumed smooth on the constraint set D. Let us define the trajectory of the system (3.1) with the initial state $\mathbf{x}_o$ at time step $k = 0$ as $\mathbf{x}_k(\mathbf{x}_o)$. A trajectory is acceptable only if $\mathbf{x}_k(\mathbf{x}_o) \in D$ for all *k*. The stability region R for $\mathbf{x} = 0$ is a connected, invariant set defined as:

$$R \;=\; \{\, \mathbf{x}_o : \mathbf{x}_k(\mathbf{x}_o) \in D, \quad \forall k \geq 0 \quad \text{and} \lim_{k\to\infty} \mathbf{x}_k(\mathbf{x}_o) = 0 \,\}$$

The problem of interest is to design a classifier that determines whether a given state is in R. This classifier should be conservative, but as accurate as possible, and fast enough to be used on-line.

3.3 Stability region estimation

This section describes the neural network architecture chosen and algorithms used to estimate the stability region R.

3.3.1 Neural network architecture

Fig. 3.1 shows the architecture of the proposed neural network to estimate stability regions. A multilayer feedforward network was selected for its capacity as an universal approximator with a size that is small relative to the size of the data set [SS92]. Even though one hidden layer is sufficient to achieve accurate approximations, the use of two hidden layers plus a linear input-output component gives more robust results [MZT96], albeit at the cost of longer training.

Using the notation developed in Chapter 2 for a multilayer neural network, the input-output relationship of the architecture in Fig. 3.1 is expressed by

$$\mathbf{y} \;=\; W_o^T \bar{\mathbf{x}} + W_3^T \Phi_2(\mathbf{x}) \tag{3.2}$$

The functions $\phi_i(.)$ for the hidden units of the neural network are all chosen to be the hyperbolic tangent functions applied to each component of their input vectors (i.e. $\phi_i(\mathbf{u}) = \tanh(\mathbf{u}_i)$).

The inputs to the neural network are the components of the system state vector, **x**, and the output is a two-dimensional vector, **y**. The components of **y** will range roughly between -1 and 1 in the region of approximation, giving an indication of the membership of the state vector in R. The ideal output values are $(y_1, y_2) = (1, -1)$ when **x** is R and $(y_1, y_2) = (-1, 1)$ when **x** is not in R. This two-variable codification of the classes was selected to give more feedback to the network during training. It follows a typical configuration which assigns as many output units as classes to separate and sets up the ideal case with a different unit active per class [HKP91, Bis95].

To make a distinct classification in the non-ideal case (i.e., when the components of **y** are not equal to ± 1), two positive threshold parameters, θ and δ, are selected to implement the following decision rule:

Declare **x** belongs to *R* if:

$$\begin{aligned} &1. \quad y_1(\mathbf{x}) - y_2(\mathbf{x}) > \theta \\ &2. \quad \|\nabla_{\mathbf{x}}(y_1(\mathbf{x}) - y_2(\mathbf{x}))\| < \delta \end{aligned} \tag{3.3}$$

where $\nabla_{\mathbf{x}}$ denotes the gradient with respect to $\mathbf{x}$.

The motivation for the classification rules is based on the necessity of obtaining conservative approximations for the stability regions. The parameter θ is chosen after the network is trained so that the classification is correct for all the training and validation data. The parameter δ is chosen to be much smaller than the maximum of the norm of the gradient of the network output over the domain. The second rule does not accept $\mathbf{x}$ as being in R in regions where the sensitivity of the outputs is high, as will be the case near the boundary of the stability region estimate.

3.3.2 Training procedure

The training of the neural network for the stability region estimator is based on supervised and reinforcement learning techniques (see Chapter 2). To initialize the training, three disjoint regions $A, B \text{ and } C$ are defined, based on a priori knowledge of the autonomous system behavior:

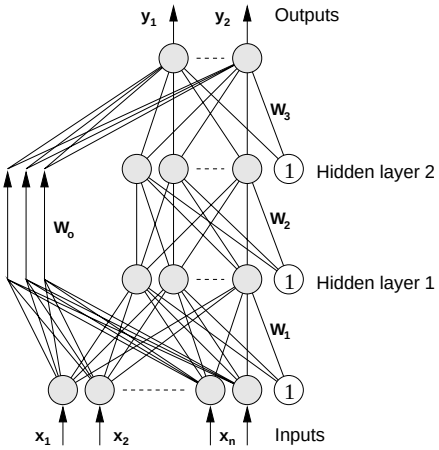

Fig. 3.1: Neural network architecture for stability region estimation.

1. *Inner region A.* A very conservative region which includes all the states from which convergence to the origin is certain. This region may be obtained from a linearized model, for example.

2. *Study region B.* The region on which the training procedure is going to be conducted.

3. *Unsafe region C.* The bounding region in which the system is either unstable or the state is outside the operating region of interest.

To begin the training, data are selected from regions A and C and supervised training is applied to generate a first approximation to the stability region R. Afterwards, experiments are performed with the system starting at states belonging to region B, and observing if the evolution leads the system to regions A or C, data are obtained for region B to train the neural network.

During each experiment reinforcement learning techniques are used for training. This is because in this case we do not know a desired value, i.e. if the current state belongs to the stability region or not, beforehand. Therefore a temporal differences method (TD_λ) [Sut88] as defined in Chapter 2 is applied to generate a prediction of the desired outputs for the network.

The update equation for the network weights W at step k is

$$\Delta W_k = \eta(P_{k+1} - P_k).\sum_{i=1}^{k} \lambda^{k-i}\nabla_w P_i, \qquad k = 1, \ldots, N.$$

$$P_k = \mathbf{y}_k, \quad k = 1, \ldots, N. \qquad P_{N+1} \equiv \mathbf{y}^*$$

with $\mathbf{y}_k$ the network output at step k and $\mathbf{y}^*$ the desired value. This algorithm is much faster than just waiting until the end of an experiment to get precise data to train the network [Wer91]. At the end of each experiment when the result is known, those data points can be added to the supervised training set.

The next element to consider in the training procedure is the design of the data to use with the algorithms described above. In an ideal situation in which experiments are possible and cheap, we can define a fine grid to cover the variation of the function by means of the sampling theorem. Training samples are taken in a square mesh inside the region selected to approximate the function. For reinforcement learning the initial states are taken uniformly random in the region to investigate.

The characteristic function we try to represent is not continuous on the domain and we try to recognize where the discontinuity arises. This is usually not very suitable for estimators based on continuous functions. However, we do not care so much about an exact representation of the function but rather want a correct decision of the state being in R instead. Moreover, this estimation has to be carried out in a compact set and not the whole state-space $\Re^n$. These facts are very important and can be used to our advantage. In these conditions Sanner and Slotine showed that it is possible to uniformly approximate the function [SS92] and provide bounds on the number of grid points needed and bounds on the error in the approximation based on the smoothness of the nonlinear system function f (3.1) and the application of the sampling theorem.

To enhance the approximation, as experiments are done, by selecting training patterns close to the boundary of R, the estimation becomes more refined where we need it. This procedure is carried out initially using off-line data, but training can continue on line as the system operates.

A conjugate-gradient, Polak-Ribiere minimization procedure is used [Min86]. The initial values of the weights are chosen randomly. Weight decay and early stopping using a test set is used to prevent overtraining. A pruning algorithm is run to optimize the size of the network by getting rid of the hidden units which do not correlate well enough with the output.

Comparisons of the neural network stability region estimator with other approaches to stability region approximation are presented in § 3.6. We have found that in all cases, with a reasonable amount of training the neural network obtains an estimate of the stability region which is much less conservative than most other methods. Moreover, since it is not model-based, it can be applied to systems using empirical data with no a priori limits on the dimension of the state space.

3.4 Estimation properties

3.4.1 Guidelines for grid design

Applying results on the sensitivity of discrete-time system trajectories we can use the data we can get from experiments more efficiently to guarantee a conservative estimation of the stability region. First, we state a well known result on the sensitivity of a discrete-time system trajectory to initial conditions.

Theorem 3.1. [Aga92] *Consider the dynamical system given by*

$$\mathbf{x}_{k+1} - \mathbf{x}_k \quad = \quad F(k, \mathbf{x}_k) \tag{3.4}$$

with F continuous and satisfying

$$\|F(k, \mathbf{x}) - F(k, \mathbf{y})\| \leq g(k, \|\mathbf{x} - \mathbf{y}\|) \tag{3.5}$$

with $g : \mathcal{Z} \times \Re \to \Re$, such that g(k,r) is nondecreasing in r for all k. Let $\mathbf{x}_o^1, \mathbf{x}_o^2 \in \Re^n$ be two different state vectors such that $\|\mathbf{x}_o^1 - \mathbf{x}_o^2\| \leq z_o$ and define $\mathbf{x}_k^i$ as the step-k state of the trajectory of the system (3.4) *starting at $\mathbf{x}_o^i$ for k=0. Then, $\|\mathbf{x}_k^1 - \mathbf{x}_k^2\| \leq z_k$ where $z_{k+1} - z_k = g(k, z_k)$.*

The system form used in Theorem 3.1 is sligthly different from the form we have adopted. However, we can translate (3.4) to (3.1) by defining $F(k, \mathbf{x}_k) = f(\mathbf{x}_k) - \mathbf{x}_k$.

The following proposition applies Theorem 3.1 rather directly.

Proposition 3.2. *Consider the dynamical system given by* (3.1) *satisfying the conditions of Theorem 3.1 with* $F(k, \mathbf{x}_k) = f(\mathbf{x}_k) - \mathbf{x}_k$. *Assume there exists* $\epsilon > 0$ *such that the ball* $B(\mathbf{x}_f, \epsilon) \subset R$ *for some* $\mathbf{x}_f \in D$. *If there exists a trajectory* $\mathbf{x}_k(\mathbf{x}_o)$ *of* (3.4) *with* $\mathbf{x}_N = \mathbf{x}_f$ *then the ball* $B(\mathbf{x}_o, z_o) \subset R$ *with* z_o *the solution of*

$$z_{k+1} - z_k = g(k, z_k), \qquad z_N = \epsilon \tag{3.6}$$

Proof. Follows directly from Theorem 3.1. □

This result can be used in the following way in our experiments: if our initial state $\mathbf{x}_o$ lies in the region B and the system trajectory goes to region A as defined in § 3.3.2, by applying Proposition 3.2 we can compute the radius z_o of a ball around $\mathbf{x}_o$ of states which also belong to R. To find z_o we have to solve (3.6) backwards in time. This information can be used in several ways. First, we can eliminate from the training set states which are included in $B(\mathbf{x}_o, z_o)$. Also, z_o is an estimate for the separation between points around $\mathbf{x}_o$ in the training set. This is only local information, and leads to what we called a non-uniform grid.

Another use for this bound is to estimate a local bound on the norm of the gradient of the neural network around $\mathbf{x}_o$. Since the neural network gives a continuous function, in order to reach the threshold right outside $B(\mathbf{x}_o, z_o)$ we should have

$$\|\nabla_x(y_1 - y_2)\| \geq \frac{(y_1 - y_2) - \theta}{z_o}.$$

This result may seem quite conservative, especially when N is large, but as the estimation of R becomes more accurate the values of N are reduced and the neighborhoods increase in size. Similar conclusions can be applied if the experiments lead to region C.

It is worth noting that it is not necessary to know the system (3.1) exactly. A bound on the function f is all that we know to apply the result for on-line experiments.

3.4.2 Confidence intervals

Since it is not possible to have noise-free data we have to establish confidence intervals in the values obtained from the neural network and use them in our conservative decision. This result comes from the application of Bayesian techniques described in Chapter 2.

We generalize the results on confidence intervals presented in § 2.6.5 by introducing uncertainty in the measurements. Assume we would like to approximate a function $h(\mathbf{x})$ using a known parametric estimation function $\phi(\mathbf{x}, W)$ that depends on a set of parameters W, and N_p input-output data values $(\mathbf{x}_i, y_i)$ perturbed by gaussian noise. Furthermore, we assume there exists a particular set of values W^* that perfectly models the unknown function $h(\mathbf{x})$. The problem can be formulated with the following expressions,

$$\begin{aligned} y^* &= h(\mathbf{x}^*) \quad = \quad \phi(\mathbf{x}^*, W^*) \\ y_i &= \phi(\mathbf{x}_i^*, W^*) + e_i,\ i = 1, \ldots, N_p \\ \mathbf{x}_i &= \mathbf{x}_i^* + \mathbf{v}_i \\ \hat{y}_i &= \phi(\mathbf{x}_i, \hat{W}) \end{aligned} \tag{3.7}$$

with the pair $(\mathbf{x}^*, y^*)$ representing the true input-output values, N_p the number of patterns used for the estimation and $W^*, \hat{W}$ the optimum and trained neural network weights, respectively. The input noise $\mathbf{v}_i$ and output error e_i are assumed to be uncorrelated gaussian random vectors with

$$\mathbf{v}_i \sim \mathcal{N}(0, \sigma_v^2 I), \qquad e_i \sim \mathcal{N}(0, \sigma_e^2) \tag{3.8}$$

The output error has two components due to the modeling error between the function h and the family of functions ϕ and a measurement error. Under these assumptions we show next that the

posterior output distribution can be approximated by a gaussian distribution. Its variance can be used as a confidence interval.

Using the same reasoning as in § 2.6.5 [Bis95], the conditional density function for the neural network output given a particular weight set and true input vector is expressed as

$$p(y|\mathbf{x}^*, W) \quad \sim \quad \mathcal{N}(\phi(\mathbf{x}^*, W), \sigma_e^2).$$

Taking into account the gaussian noise at the input of the network we can express the input-output conditional density function as

$$p(y, \mathbf{x}|\mathbf{x}^*, W) \quad \sim \quad \mathcal{N}(\phi(\mathbf{x}^*, W), \sigma_e^2)\, \mathcal{N}(\mathbf{x}^*, \sigma_v^2 I).$$

Assuming the input-output noisy patterns $T_i = (\mathbf{x}_i, y_i)$ are independent to each other, the conditional density function for the pattern set $T = \{T_i,\ \forall i\}$ is

$$p(T|X^*, W) \quad = \quad \prod_{i=1}^{N_p} p(y_i, \mathbf{x}_i|\mathbf{x}_i^*, W) \tag{3.9}$$

with $X^* = \{\mathbf{x}_i^*, \forall i\}$. Applying Bayes' formula we get the a posteriori probability distribution of the weights

$$\begin{aligned}
p(W|X^*, T) &= \frac{p(T|X^*, W)\, p(W|X^*)}{p(T|X^*)} \\
&= \frac{\exp(-S(W)) \exp\left(\alpha_x \sum_{i=1}^{N_p} \|\mathbf{x}_i - \mathbf{x}_i^*\|^2\right)}{\int p(T|X^*, W) p(W|X^*) dW} \\
&\sim \exp(-S(W)) \\
S(W) &= \beta \sum_{i=1}^{N_p} (\phi(\mathbf{x}_i^*, W) - y_i)^2 + \alpha \sum_{i=1}^{N_w} W_i^2
\end{aligned}$$

where $S(W)$ is the squared error over the target data. If the neural network is trained, $\hat{W}$ minimized $S(W)$ and

$$\begin{aligned}
S(W) &\simeq S(\hat{W}) + \frac{1}{2}(W - \hat{W})^T H_w (W - \hat{W}) \\
\phi(\mathbf{x}^*, W) &\simeq \phi(\mathbf{x}, \hat{W}) + \nabla_x^T (\mathbf{x}^* - \mathbf{x}) + \nabla_w^T (W - \hat{W}) \\
&\simeq \hat{y} + y_x^T \tilde{x} + y_w^T \tilde{W}
\end{aligned}$$

in a neighborhood of the optimum value W^*. Using the a posteriori conditional density function for the weights we can find the output conditional density function for a given input and training data.

$$\begin{aligned} p(y|\mathbf{x},T) &= \int p(y,\mathbf{x}^*|\mathbf{x},W)p(W\,|X^*,T)\;dW\,d\mathbf{x}^* \\ &\simeq \int \exp\left[\frac{-1}{2\sigma_e^2}(y-\hat{y}-y_x^T\tilde{\mathbf{x}}-y_w^T\tilde{W})^2-\frac{1}{2\sigma_v^2}\tilde{\mathbf{x}}^T\tilde{\mathbf{x}}-\frac{1}{2}\,\tilde{W}^T H_w\tilde{W}\right] dW\,d\mathbf{x}^* \end{aligned}$$

Completing squares in W in the integrand and integrating first in the variable W we get

$$\begin{aligned} p(y|\mathbf{x},T) &\simeq \int \exp\left[-\frac{1}{2\sigma_v^2}\tilde{\mathbf{x}}^T\tilde{\mathbf{x}}-\frac{1}{2\sigma_w^2}(y-\hat{y}-y_x^T\tilde{\mathbf{x}})\right] d\mathbf{x}^* \\ &\simeq \exp\left[-\frac{1}{2\sigma_y^2}(y-\hat{y})^2\right] \\ p(y|\mathbf{x},T) &\sim \mathcal{N}(\phi(\mathbf{x},\hat{W}),\sigma_y^2) \end{aligned} \tag{3.10}$$

with

$$\begin{aligned} \sigma_y^2 &= \sigma_w^2+\sigma_v^2\,\nabla_x^T\phi\,\nabla_x\phi \\ \sigma_w^2 &= \sigma_e^2+\nabla_w^T\phi\,H_w^{-1}\,\nabla_w\phi \\ H_w &= \left.\frac{\partial^2 S(W)}{\partial W^2}\right|_{\hat{W}} \end{aligned} \tag{3.11}$$

Given the training set and a proposed threshold value θ, we can compute the probability of error in the decision rule (3.3) by using the gaussian distribution for the output of the neural network (3.10). This idea gives us a way to improve the estimate for θ by

$$\begin{aligned} \theta &\in (\theta_1(\epsilon),\theta_2(\epsilon)) \\ \theta_1(\epsilon) &= \max_{\mathbf{x}\in T,\mathbf{x}\notin R}\;\theta_1(\mathbf{x},\epsilon) \\ \theta_2(\epsilon) &= \min_{\mathbf{x}\in T,\mathbf{x}\in R}\;\theta_2(\mathbf{x},\epsilon) \\ \theta_1(\mathbf{x},\epsilon) &= \min_{\vartheta}\{\vartheta\mid p((y_1-y_2)>\vartheta|\mathbf{x},T)\le\epsilon\} \\ \theta_2(\mathbf{x},\epsilon) &= \max_{\vartheta}\{\vartheta\mid p((y_1-y_2)<\vartheta|\mathbf{x},T)\le\epsilon\} \end{aligned} \tag{3.12}$$

for a given error ϵ. It may not be possible to find a value of θ that meets these conditions. This would show that our estimation might not be very good, either because training was not sufficient,

there is lack of data in some region of the state space, or the size of the network is too small. The result in (3.10) also shows the influence of the derivative of the output with respect to the input for the confidence interval what confirms the usefulness of the second rule. As a matter of fact, the term in the expression of σ_y^2 in (3.11) that depends on the derivative can be used to set up an estimate for δ.

Going a little bit further on the computation of confidence intervals in the estimation of a stability region, a more intelligent filter can be used on the input patterns, e.g. a Kalman filter, as a way of improving the confidence interval estimation. We have not pursued this idea in this project.

3.5 Computational issues

Computation costs in neural networks are measured in terms of the number of weights N_w in the network and the number patterns N_p used in the training procedures. Furthermore, the number of weights depends on the number of units per layer and it is given by the expression

$$N_w = (n_o + n_2 + 1)(n_3 + n_1) + n_2 \tag{3.13}$$

where n_i are the number of units in the i^{th} layer. The computational complexity of the supervised learning using the backpropagation algorithm to compute the derivatives is $\mathcal{O}(N_w N_p)$ [Bis95].

The confidence intervals introduced in § 3.4.2 require the computation of the Hessian of the sum of the squared errors for the patterns with respect to the weights of the neural network. This computation is more intensive, with a complexity of $\mathcal{O}(N_w^2)$. However, it is not that important since it is only performed after the network has been trained. An optimized algorithm to compute the inverse of the Hessian matrix directly was used developed by Hassibi and Stork [HS93]. It is based on the outer product approximation [Lev44, Mar63].

3.6 Two-dimensional examples and comparisons

In this section we examine the results of the proposed method for simple second-order systems and compare the results with the ones obtained by Davison and Kurak [DK71] and Vannelli and Vidyasagar [VV85]. These examples are continuous time systems. To be able to do the comparison, a standard ordinary differential equation solver is used to compute the trajectory for each sample period. An object-oriented approach was developed in C++ [Lip95] to handle the discrete-time conversion and different examples in a modular way. Further analysis and display was done using MATLAB ™.

3.6.1 Van der Pol equation

The state equations for this example are:

$$\begin{aligned} \dot{x}_1 &= -x_2 \\ \dot{x}_2 &= x_1 - x_2 + x_1^2 x_2 \end{aligned}$$

Knowing the region of stability in this case, the regions for training were taken as:

$$\begin{aligned} A &= \{(x_1, x_2) \in \Re^2 \mid \ |x_1| \leq 0.5 \text{ and } |x_2| \leq 0.5\} \\ C &= \{(x_1, x_2) \in \Re^2 \mid \ |x_1| \geq 2.5 \text{ or } |x_2| \geq 3.0\} \\ B &= A^c \cap C^c \end{aligned}$$

where the superscript c refers to the complement set. The sample time used was $T = 0.01$. The error values and parameters were averaged over 5 runs. Table 3.1 shows the results of the proposed method. Fig. 3.2 shows the estimation of that region according to the three methods. The network approximation is closer to the actual stability region than the other methods. The ideas developed in § 3.4.1 were not necessary in this very small simulation example and satisfactory results were obtained with the use of a uniform grid. However, we have to look more closely into the results

Training samples:	3721
Testing samples:	961
Units in each layer:	2 - 10 - 2 - 2
Training set MSE:	0.105
Test set MSE:	0.152
θ, δ :	0.9 / 1.1

Table 3.1: Van der Pol equation: Estimation results.

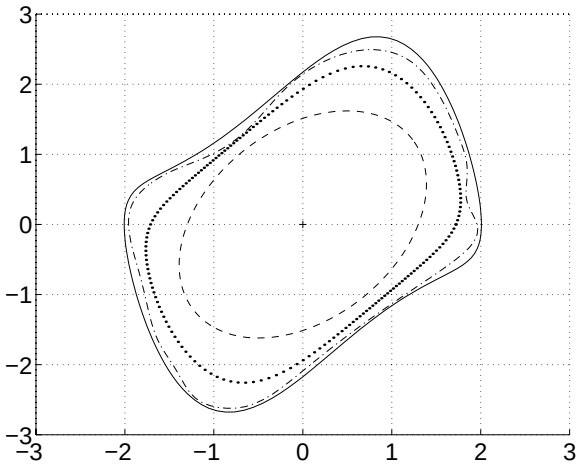

Fig. 3.2: Van der Pol equation: Estimates comparison.

solid:	actual stability region.
dotted:	Vannelli and Vidyasagar method
dashed:	Davison and Kurak method
dash-dotted:	neural network estimation.

obtained in § 3.4.2 to be able to get values for θ and δ with reasonable confidence intervals. If we try to design the value of θ by applying the results of § 3.4.2 and use equations (3.12) directly, we come up with no solution for an error ϵ of 5 percent or less. Examining the results we note that the confidence intervals are greatly affected at some points in state space for which there is a large

derivative of the output of the network with respect to the input. This was expected from the form of the expression (3.10). Therefore, the second rule was checked first and the remaining points were only tested to determine the value for θ.

If we do not take these points that violate rule 2 into account and use

$$\begin{aligned} \epsilon &= 0.01 \\ \sigma_v &= 0.05 \\ \sigma_e^2 &= \frac{1}{N_p - N_w} \sum_{i=1}^{N_p} (\mathbf{y}_i - \phi(\mathbf{x}_i, \hat{W}))^2 \end{aligned}$$

we find a good trade-off for

$$\theta = 1.0 \qquad \delta = 4.2$$

Fig. 3.3 shows the values obtained for θ_1 and θ_2 for the training patterns. We can observe that the maximum value of θ_1 is larger than the minimum of θ_2, implying a solution that correctly classifies all the patterns with a one percent accuracy is not possible. However, if we take $\theta = \theta_1$, to assure $\epsilon = 0.01$ accuracy on the classification for $\mathbf{x} \notin R$, the percentage of patterns wrongly classified as not in the region of stability is very small. Another look at the same data, shown in Fig. 3.4, displays a contour plot of the values of θ_1 and θ_2 in the state space. The figure shows, as expected, that the large confidence intervals are located very close to the stability boundary. Finally, Fig. 3.5 displays the contour plots of both rules and relative to the real stability boundary. In this case the derivative rule is the most conservative rule. Actually, the contour for the threshold rule and the real boundary lie inside the band set up by the derivative rule.

3.6.2 Second example

This example is also taken from [DK71]. The state equations are:

$$\begin{aligned} \dot{x}_1 &= -x_1 + 2x_1^2 x_2 \\ \dot{x}_2 &= -x_2 \end{aligned}$$

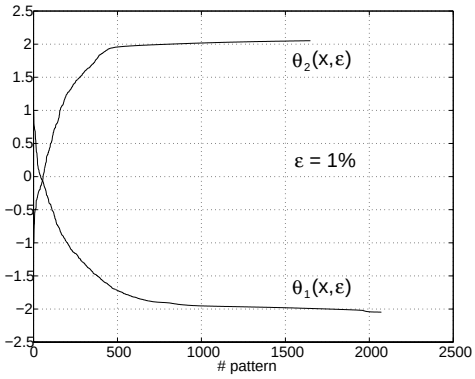

Fig. 3.3: Van der Pol: Confidence interval for threshold θ for the training patterns.

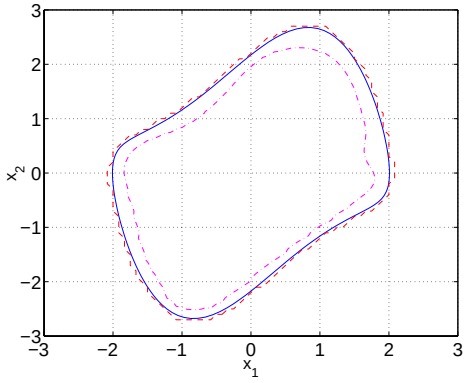

Fig. 3.4: Van der Pol: Confidence interval threshold contours.

solid:	actual stability region
dashed:	contour for θ_1.
dash-dotted:	contour for θ_2.

In this case the region of stability is unbounded:

$$R = \{\mathbf{x} : \ x_1.x_2 < 1\}$$

The method developed in [VV85] gives a perfect estimation of the region. The method reported in this work is essentially designed for a bounded region. Therefore, the intersection of the stability region and the region, given by

$$|x_1| \leq 3.0, \qquad |x_2| \leq 3.0$$

was estimated. Table 3.2 shows the results of the proposed method. Fig. 3.6 shows the estimation of the stability region according to the three methods and the contour plots for the decision rules using the neural-based stability region estimation.

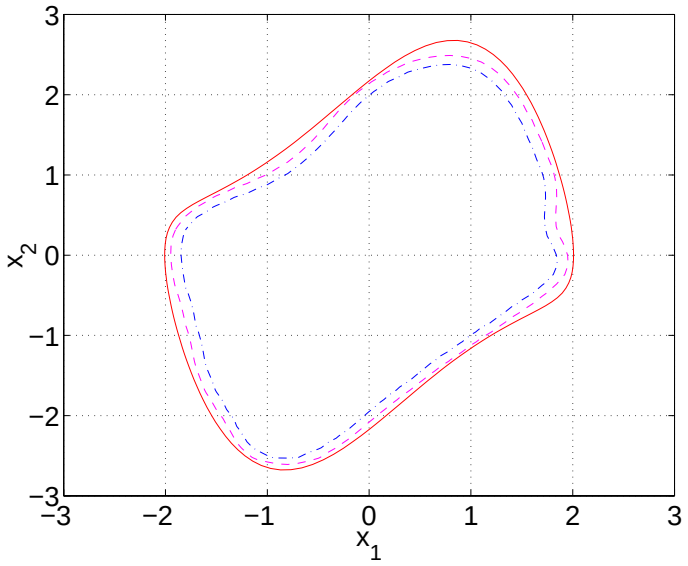

Fig. 3.5: Van der Pol equation: Confidence interval threshold θ for the training patterns.

solid:	actual stability region.
dashed:	contour for threshold rule ($\theta = 1.0$).
dash-dotted:	contour for derivative rule ($\delta = 4.2$).

Training samples:	3721
Testing samples:	1024
Units in each layer:	2 - 8 - 2 - 2
Training set MSE:	0.0128
Test set MSE:	0.0144
θ, δ :	0.2 / 1.3

Table 3.2: Network estimation results

3.7 Discussion

A method to estimate the stability region of an autonomous nonlinear system using neural networks is presented. Although the model of the system is not used directly, any a priori knowledge might

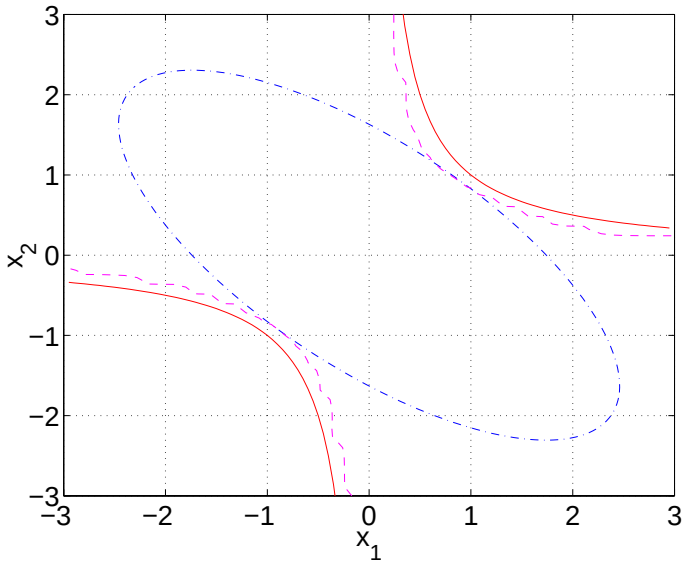

Fig. 3.6: Contour plots for decision rules compared to other estimates.

solid:	actual stability region and
	Vannelli and Vidyasagar method
dash-dotted:	Davison and Kurak method
dashed:	decision rules ($\theta = 0.2,\ \delta = 1.3$).

be used for training. For example, the knowledge of bounds on the dynamic equation function may lead to a better design of the grid to train the neural network. Confidence regions are computed as a way of making a conservative decision whether a point in the state space belongs to the stability region. Some basic comparisons with other methods on standard low-order systems is given as a preliminary study. Further comparisons with other methods and applications will be made in Chapters 6 and 7.

In applications, the knowledge of a region of stability can have several uses. If we think of the nonlinear autonomous system as a closed-loop system with state and input constraints, the stability region estimation can allow us to make a decision to shutdown the system or change controllers when the system goes out of the stable region. It is necessary to perform experiments to estimate

the region of stability, but actual data from the system operation can be used as training data for the network to improve its estimation.

Chapter 4

Neural network estimates of performance indices

4.1 Introduction

There is no doubt as to the importance of performance indices in the development of the control systems theory. An effective performance index measures the quality of significant features of the system behavior. To be useful for control system analysis and design, the performance index must also be computable and lead to effective numerical methods for computing optimal control laws. We can cite the optimal control theory and more recently the robust control theory as examples of effective definitions and uses of performance indices. It is clear from those theories that there are always tradeoffs between the class of systems one can study, linear systems in the case of optimal and robust control, and the possibilities for developing effective computational tools. It is more difficult to derive analytical results and numerical methods for nonlinear systems. In this chapter we present results on the estimation of a certain type of performance indices, cost-to-go functions, by the use of neural networks. In general, the methods presented here can be applied to an arbitrary nonlinear system. Moreover, constraints on control and state variables can be included.

4.2 Problem formulation

As a general formulation of the problem being considered, let a state-constrained and control-constrained nonlinear discrete-time system in its state-space formulation be given by

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{x}_k \in S, &\qquad \mathbf{u}_k \in D \end{aligned} \tag{4.1}$$

where $\mathbf{x}_k \in \Re^n$ is the state vector and $\mathbf{u}_k \in \Re^m$ the control input vector. The nonlinear function f is assumed to be smooth. The discrete-time state equations reflect the sampled-data nature of the computer control system. We assume that the state $\mathbf{x}_k$ is observable. The control signal $\mathbf{u}_k$ to be applied on the system can be generated by M different state feedback controllers of the form:

$$\mathbf{u}_k^i = g^i(\mathbf{x}_k), \quad i = 1, 2, \ldots, M. \tag{4.2}$$

The goal of the controllers is to take the system to the origin.

Let us denote the trajectory of the system (4.1) with the initial state $\mathbf{x}_o$ at time step k by the application of the state feedback control $\mathbf{u}_k^i$ as $\mathbf{x}_k^i(\mathbf{x}_o)$. We consider performance indices for the controllers of the form

$$J_\delta^i(\mathbf{x}_o) = B^i + \sum_{k=0}^{\infty} \delta^k U^i(\mathbf{x}_k^i(\mathbf{x}_o)), \qquad i = 1, \ldots, M \tag{4.3}$$

where $0 < \delta \leq 1$ is a discount factor and $U : \Re^n \rightarrow \Re$ represents the cost function of being in a particular state.

In this chapter we develop a method to estimate $J_\delta^i(\mathbf{x})$ using neural networks.

4.3 Performance index properties

We first analyze some properties of the performance indices defined by (4.3) that will be useful in later developments.

Property 4.1. *For any state* $\mathbf{x} \in S$*, and any two discount factors* δ_1, δ_2 *such that* $0 < \delta_1 < \delta_2 \leq 1$*,*

$$J^i_{\delta_1}(\mathbf{x}) \leq J^i_{\delta_2}(\mathbf{x})$$

Lemma 4.1. *For any given state* $\mathbf{x}_o$ *and integer* $k \geq 0$*, if the performance index* (4.3) *is well defined, i.e.* (4.3) *converges,* J^i_δ *satisfies:*

$$J^i_\delta(\mathbf{x}^i_k(\mathbf{x}_o)) \;=\; (1-\delta)B^i + U^i(\mathbf{x}^i_k(\mathbf{x}_o)) + \delta\, J^i_\delta(\mathbf{x}^i_{k+1}(\mathbf{x}_o)) \tag{4.4}$$

Proof. It follows by direct substitution of (4.3) into (4.4). □

Lemma 4.2. *For a given state* $\mathbf{x}_o$*, if* (4.3) *converges for* δ_1 *and* δ_2*, the difference between the performance indices for a given controller* g^i *for two different values of the discount factor can be expressed by:*

$$J^i_{\delta_1}(\mathbf{x}_o) - J^i_{\delta_2}(\mathbf{x}_o) \;=\; (\delta_1 - \delta_2) \sum_{j=0}^{\infty} \delta_1^j\, J^i_{\delta_2}(\mathbf{x}^i_{j+1}(\mathbf{x}_o)) \tag{4.5}$$

Proof. Equation (4.5) follows directly from the definition of the performance indices (4.3).

$$\begin{aligned}
J^i_{\delta_1}(\mathbf{x}_o) - J^i_{\delta_2}(\mathbf{x}_o) &= \sum_{k=0}^{\infty} (\delta_1^k - \delta_2^k) U^i(\mathbf{x}^i_k(\mathbf{x}_o)) \\
&= (\delta_1 - \delta_2) \sum_{k=0}^{\infty} \left(\sum_{j=0}^{k-1} \delta_1^j\, \delta_2^{k-1-j} \right) U^i(\mathbf{x}^i_k(\mathbf{x}_o)) \\
&= (\delta_1 - \delta_2) \sum_{j=0}^{\infty} \sum_{k=j+1}^{\infty} \delta_1^j\, \delta_2^{k-1-j}\, U^i(\mathbf{x}^i_k(\mathbf{x}_o)) \\
&= (\delta_1 - \delta_2) \sum_{j=0}^{\infty} \sum_{l=0}^{\infty} \delta_1^j\, \delta_2^l\, U^i(\mathbf{x}^i_{l+j+1}(\mathbf{x}_o)) \\
&= (\delta_1 - \delta_2) \sum_{j=0}^{\infty} \delta_1^j\, J^i_{\delta_2}(\mathbf{x}^i_{j+1}(\mathbf{x}_o))
\end{aligned}$$

□

From this Lemma we can draw the following propositions about the continuity of J_δ^i with respect to the parameter δ.

Proposition 4.3. *For a given state $\mathbf{x}_o$ and controller g^i, a sufficient condition for $J_\delta^i(\mathbf{x}_o)$ to be continuous with respect to δ at $\delta = \delta_o$ is that $J_\delta^i(\mathbf{x})$ is well defined in a neighborhood of δ_o and*

$$\sum_{j=0}^{\infty} J_{\delta_o}^i(\mathbf{x}_{j+1}) < \infty$$

Proof. If we take $|\delta_o - \delta_1| < \epsilon_1$ and apply Lemma 4.2

$$\begin{aligned} |J_{\delta_o}^i(\mathbf{x}_o) - J_{\delta_1}^i(\mathbf{x}_o)| &< |\delta_o - \delta_1| \quad |\sum_{j=0}^{\infty} \delta_1^j J_{\delta_o}^i(\mathbf{x}_{j+1})| \\ &< \epsilon_1 \quad \sum_{j=0}^{\infty} J_{\delta_o}^i(\mathbf{x}_{j+1}) \\ &< \epsilon \end{aligned}$$

□

Proposition 4.4. *For a given state $\mathbf{x}_o$ and controller g^i, a sufficient condition for $J_\delta^i(\mathbf{x}_o)$ to be continuous from the left at $\delta = \delta_o$ is*

$$\sum_{j=0}^{\infty} \delta_o^j J_{\delta_o}^i(\mathbf{x}_{j+1}) < \infty$$

Proof. The idea is the same as in the previous proposition, using Lemma 4.2 and Property 4.1. If we take $\epsilon_1 > 0,\ (\delta_o - \delta_1) < \epsilon_1$,

$$\begin{aligned} J_{\delta_o}^i(\mathbf{x}_o) - J_{\delta_1}^i(\mathbf{x}_o) &= (\delta_o - \delta_1) \quad \sum_{j=0}^{\infty} \delta_o^j J_{\delta_1}^i(\mathbf{x}_{j+1}) \\ &< \epsilon_1 \sum_{j=0}^{\infty} \delta_o^j J_{\delta_o}^i(\mathbf{x}_{j+1}) \\ &< \epsilon \end{aligned}$$

□

4.4 Estimation procedure

The output $\mathbf{y}$ of the neural network estimator of the performance index (4.3) for each controller is a scalar. A multilayer neural network is used with two hidden layers with hyperbolic tangent activation functions, linear output units and a linear input-output layer. Using the notation developed in Chapter 2 the estimate performance index for each controller is given by

$$\hat{J}_\delta^i(\mathbf{x}, \hat{W}) \quad = \quad \hat{W}_o^T \bar{\mathbf{x}} + \hat{W}_3^T \Phi_2(\mathbf{x}) \tag{4.6}$$

Estimating performance indices such as (4.3) is a standard problem in Neuro-dynamic programming [BT96]. A Heuristic Dynamic Programming (HDP) algorithm [Wer91] is used to train the networks. The training algorithm uses an estimation of the cost-to-go at $\mathbf{x}_k$ given by

$$J_\delta^{i*}(\mathbf{x}_k, \hat{W}_k) = U^i(\mathbf{x}_k) + \delta \hat{J}_\delta^i(\mathbf{x}_{k+1}, \hat{W}_k) \tag{4.7}$$

where $J_\delta^{i*}(\mathbf{x}_k, W_k)$ is the estimated desired value for the network for state $\mathbf{x}_k$ based on the current values of the weights $\hat{W}_k$. In our applications, the positive definite state cost function selected is a standard quadratic form,

$$U^i(\mathbf{x}) = \mathbf{x}^T \, P^i \, \mathbf{x}, \qquad P^{iT} = P^i > 0, \; i = 1, \ldots, M.$$

Equation (4.7) is motivated by the definition of $J_\delta^i(\mathbf{x})$ (4.3) neglecting the bias term B^i which is added directly to the output of the network. The HDP training procedure is illustrated in Fig. 4.1. The *error* signal ε_k is defined as

$$\varepsilon_k \quad = \quad J_\delta^{i*}(\mathbf{x}_k, \hat{W}_k) - \hat{J}_\delta^i(\mathbf{x}_k, \hat{W}_k). \tag{4.8}$$

The basic HDP algorithm uses a gradient descent update to attempt to minimize ε_k^2 for each k. Therefore, the update equations for the weights of the neural network becomes

$$\hat{W}_{k+1} \quad = \quad \hat{W}_k - \eta_k \, \varepsilon_k \, \nabla_{\hat{W}_k} \varepsilon_k \tag{4.9}$$

with η_k the training coefficient. Note that the HDP algorithm can be applied both off-line and on-line.

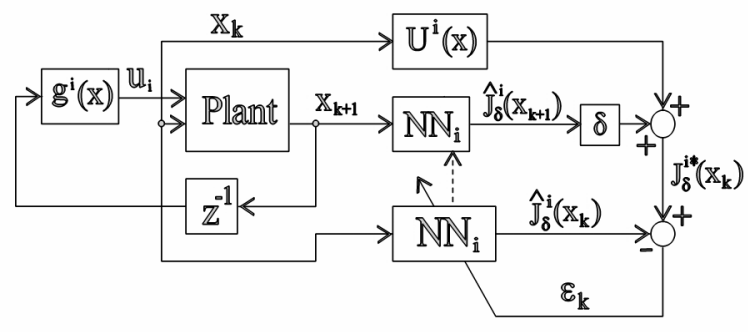

Fig. 4.1: HDP learning scheme.

4.5 Convergence results

We are unaware of any results in the literature providing sufficient conditions for convergence of the neural network parameters for performance estimators of the type described above. Our experience applying the HDP algorithm to estimate the performance index for systems in simulation and using experimental data from a physical system (see Chapters 6 and 7) has been encouraging, however. Moreover, analytical results for related problems in the context of Q-learning [BYB94] and temporal differences [Day92], [TR97] indicate that convergence should be expected under rather mild conditions. A main difference between our approach and most work on learning cost-to-go performance indices is that the control signal in our case does not depend on the performance index (for one of the given controllers). We assume, rather, that each of the given controllers stabilizes the system and the feedback law remains fixed.

The following result provides a sufficient condition under which convergence of the neural network parameters in the output layer is guaranteed for the problem being considered in this paper.

Theorem 4.5. *Given the nonlinear system* (4.1) *under the action of a prespecified control law* $g(\mathrm{x})$ *that stabilizes the system and the performance index* $J(\mathrm{x})$ *defined by* (4.3)*, assume that* $J(\mathrm{x})$ *can*

be expressed as a linear combination of basis functions

$$J_\delta(\mathbf{x}) = W^T \Psi(\mathbf{x}) \tag{4.10}$$

where W is a parameter vector and $\Psi(.)$ is the vector of known bounded basis functions. If we define the estimator for $J(\mathbf{x})$ as

$$\hat{J}_\delta \;\; = \;\; \hat{W}^T \Psi(\mathbf{x}), \tag{4.11}$$

use the HDP algorithm given by (4.7)-(4.9) *to adapt the parameter vector $\hat{W}$ and,*

$$\eta(1+\delta^2)\Psi_M^2 < 2, \quad \|\Psi(.)\| < \Psi_M \tag{4.12}$$

where δ is the discount factor in (4.3) *and η is the training coefficient in* (4.9)*, then the magnitude of the vector $\tilde{W}_k = W - \hat{W}_k$ is monotone nonincreasing.*

Proof. See Appendix A. □

Theorem 4.5 relies on the assumption of having a set of basis functions Ψ that satisfies (4.10). To apply Theorem 4.5 to our two-level neural networks, the functions Ψ in (4.11) can represent the outputs of the last hidden layer of the neural network plus the bias terms, and the vector W then represents the coefficients in (4.6) for the output layer. The functions Ψ can be determined by fixing the weights in the first hidden layers after an initial step of training. These weights should be set so that $\Psi(\mathbf{x}) = 0$ if and only if $\mathbf{x} = 0$. The proof of Theorem 4.5 indicates that the iterations to compute the weights in (4.6) will converge to the correct values provided the expression $\tilde{W}_k^T(\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k))$ is non-zero infinitely often. This condition can be viewed as a type of "persistent excitation" requirement for the neural network training procedure. In this case, the x_k sequence is independent of the neural network weights – it is simply the state sequence resulting from the application of one of the given controllers. As there is no requirement in the theorem that the state sequence be from a single state trajectory, the neural network training can be performed continuously with different state trajectories.

This method being a steepest descent-type algorithm has the general convergence properties of steepest descent algorithms. It is well known [Min86] that in the case of the error being approximately a quadratic function in the training parameters, which is true near the origin, the ratio of convergence is linear, i.e.

$$\frac{\|\tilde{W}_{k+1}\|}{\|\tilde{W}_k\|} \quad \leq \quad \frac{M-m}{M+m} \tag{4.13}$$

with M, m being the largest and smallest eigenvalues of the quadratic form. From the proof of Theorem 4.5 in Appendix A the error to adapt the parameters of the network can be written as

$$\varepsilon_k^2 = \tilde{W}_k^T \Theta_\delta(\mathbf{x}_k) \Theta_\delta^T(\mathbf{x}_k) \tilde{W}_k$$

where

$$\Theta_\delta(\mathbf{x}_k) \quad = \quad \delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}), \tag{4.14}$$

giving an ill-conditioned quadratic function with $m = 0$ that possibly leads to very slow convergence, since we can only assure from (4.13) that

$$\frac{\|\tilde{W}_{k+1}\|}{\|\tilde{W}_k\|} \quad \leq \quad 1$$

4.5.1 Adding a momentum term

The first extension to the algorithm develop in Theorem 4.5 with the goal of improving its convergence is to add a so-called momentum term [Min86]. This introduces a slight change in the adaptation equation for the network weights (4.9) to

$$\hat{W}_{k+1} \quad = \quad \hat{W}_k - \eta_k \, \varepsilon_k \, \nabla_{\hat{W}_k} \varepsilon_k + \nu_k (\hat{W}_k - \hat{W}_{k-1}) \tag{4.15}$$

where ν_k is the momentum update coefficient. It is straightforward to show that the error dynamics is now given by

$$\begin{bmatrix} \tilde{W}_{k+1} \\ \tilde{W}_k \end{bmatrix} \quad \left[\begin{array}{c|c} I(1+\nu_k) - \eta_k \Theta(\mathbf{x}_k)\Theta(\mathbf{x}_k)^T & -\nu_k I \\ \hline I & 0 \end{array}\right] \quad \begin{bmatrix} \tilde{W}_k \\ \tilde{W}_{k-1} \end{bmatrix}$$

with $\Theta(\mathbf{x}_k)$ given by (4.14). Computationally, nothing greater is added to the algorithm. However, for the case in which η_k and ν_k are the same for all k the rate of convergence is known to improve to

$$\frac{\|\tilde{W}_{k+1}\|}{\|\tilde{W}_k\|} \quad \leq \quad \frac{\sqrt{M}-\sqrt{m}}{\sqrt{M}+\sqrt{m}}$$

with M, m the largest and smallest eigenvalues of $\Theta(\mathbf{x}_k)\Theta(\mathbf{x}_k)^T$ [Min86]. Unfortunately, this case is ill-conditioned as before and in general we can not assure exponential convergence either. Although it seems we do not gain much on doing this, it is usually advisable since the approximation procedure is smoother than using gradient descent only. The next sections describe other algorithms that have been developed to approximate the cost-to-go function.

4.6 Embedded filtering

As mentioned before, the approximation taken, linear in the parameters, is very limited in terms of the accuracy that can be reached with respect to the number of parameters involved. In this section and the next section we suggest different approaches that may improve the approximation.

A further extension in the performance estimation procedure may include some filtering in the adaptive algorithm to obtain a smoother approximation. Fig. 4.2 shows a diagram of this procedure. The cost-to-go function for the closed-loop system i satisfies

$$J_\delta^i(\mathbf{x}_k) \quad = \quad U(\mathbf{x}_k) + \delta J_\delta^i(\mathbf{x}_{k+1}) \tag{4.16}$$

which is equivalent to

$$J_\delta^i(\mathbf{x}_{k+1}) \quad = \quad (\lambda+\mu)J_\delta^i(\mathbf{x}_k) - (1/\delta)U(\mathbf{x}_k) \tag{4.17}$$

when $\delta(\lambda+\mu)=1$. We can split the λ and μ terms and think of this expression as a first-order filter equation by introducing a new variable $\varphi_\delta^i(k)$ given by

$$\varphi_\delta^i(k+1) \quad = \quad \mu\varphi_\delta^i(k) + \lambda J_\delta^i(\mathbf{x}_k) - (1/\delta)U(\mathbf{x}_k) \tag{4.18}$$

The coefficients λ and μ can change on each iteration as long as they satisfy

$$\begin{aligned} 0 < \mu < 1 \qquad & \lambda > 0 \\ \delta(\mu + \lambda) \;\; = \;\; & 1 \end{aligned}$$

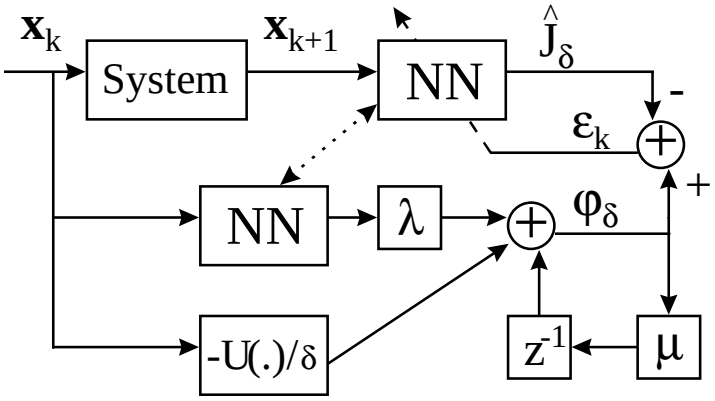

Fig. 4.2: Embedded filter adaptive scheme.

Theorem 4.6. *Assuming a linear approximator function for the cost-to-go function as in* (4.11)*, the embedded filter adaptive scheme can be expressed by the following equations:*

$$\begin{aligned} \hat{W}_{k+1} &= \hat{W}_k - \eta_k(\Theta_\lambda - \mu_k\phi_k)(U(\mathbf{x}_k)/\delta + \hat{W}_k^T\Theta_\lambda - \mu_k\varphi_k) \\ \varphi_{k+1} &= \mu_k\varphi_k + \lambda_k\hat{W}_k\Psi(\mathbf{x}_k) - U(\mathbf{x}_k)/\delta \\ \phi_{k+1} &= \mu_k\phi_k + \lambda_k\Psi(\mathbf{x}_k) \\ \Theta_\lambda &= \Psi(\mathbf{x}_{k+1}) - \lambda_k\Psi(\mathbf{x}_k) \end{aligned} \tag{4.19}$$

Furthermore, if $\mu_k < 1$ *and* $\lambda_k,\ \eta_k,\ \mu_k$ *satisfy*

$$\sigma_2 \begin{bmatrix} I - \eta_k(\Theta_\gamma - \mu_k\phi_k)\Theta_\lambda^T & \eta_k\mu_k(\Theta_\gamma - \mu_k\phi_k) \\ \lambda_k\Psi^T(\mathbf{x}_k) & \mu_k \end{bmatrix} \;\; < \;\; 1 \tag{4.20}$$

with $\sigma_2[.]$ *denoting the second largest singular value and*

$$\begin{aligned} \Theta_\gamma &= \Psi(\mathbf{x}_{k+1}) - \gamma\Psi(\mathbf{x}_k) \\ \gamma &= \lambda_k + \mu_k = 1/\delta \end{aligned}$$

the stability of the system (i.e. the closed-loop system under a given controller) implies the boundedness of the discrete signals in (4.19)*, and* $\hat{W}_k \to W$ *and* $\varphi_k \to J^i_\delta(\mathbf{x}_k)$.

Proof. See Appendix A. □

Remark 4.1. This approach avoids one of the problems found in the algorithm described in section 4.5 with respect to the convergence because the adaptive parameters may be selected to avoid getting stuck in the direction $\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)$.

Remark 4.2. This procedure is computationally more expensive. The equation (4.20) is not a linear matrix inequality (LMI) and its feasibility remains to be shown.

4.7 Higher order approximation

The algorithms presented in the previous sections are based on the first-order equation for the cost-to-go function (4.16). This equation can be generalized to a higher-order equation and adaptive algorithms derived using a similar procedure to HDP.

The following lemma is a generalization of Lemma 4.1.

Lemma 4.7. *For any given state* $\mathbf{x}_o$ *and integers* $k, r \geq 0$, *if the performance index* (4.3) *is well defined, i.e.* (4.3) *converges,* J^i_δ *satisfies:*

$$J^i_\delta(\mathbf{x}^i_k(\mathbf{x}_o)) = (1-\delta)B^i \sum_{j=0}^{r-1} \delta^j + \sum_{j=0}^{r-1} \delta^j U^i(\mathbf{x}^i_{k+j}(\mathbf{x}_o)) + \delta^r J^i_\delta(\mathbf{x}^i_{k+r}(\mathbf{x}_o)) \qquad (4.21)$$

Proof. Equation (4.21) is deduced directly by applying (4.4) of Lemma 4.1 r times. □

Without lost of generality we can assume $B^i = 0$. We can rewrite (4.21) as,

$$\begin{aligned} J^i_\delta(\mathbf{x}^i_k(\mathbf{x}_o)) &= \sum_{j=0}^{r-1} \delta^j U^i(\mathbf{x}^i_{k+j}(\mathbf{x}_o)) + \delta^r\, J^i_\delta(\mathbf{x}^i_{k+r}(\mathbf{x}_o)) \\ &= S^i_{\delta kr}(\mathbf{x}_o) + \delta^r\, J^i_\delta(\mathbf{x}^i_{k+r}(\mathbf{x}_o)) \end{aligned} \tag{4.22}$$

Since (4.22) is valid for all $r > 0$ we can use a convex combination of terms. Then,

$$J^i_\delta(\mathbf{x}^i_k(\mathbf{x}_o)) = \sum_{j=1}^{r} \alpha_j S^i_{\delta k j}(\mathbf{x}_o) + \sum_{j=1}^{r} \alpha_j \delta^j\, J^i_\delta(\mathbf{x}^i_{k+j}(\mathbf{x}_o)) \tag{4.23}$$

$$\sum_{j=1}^{r} \alpha_j = 1 \qquad \alpha_j > 0 \quad \forall j$$

Moreover, we can rewrite (4.23) back in terms of $U^i(\mathbf{x}^i_k(\mathbf{x}_o))$,

$$\begin{aligned} J^i_\delta(\mathbf{x}^i_k(\mathbf{x}_o)) &= \sum_{j=0}^{r-1} \beta_j \delta^j U^i(\mathbf{x}^i_{k+j}(\mathbf{x}_o)) + \sum_{j=1}^{r} \alpha_j \delta^j\, J^i_\delta(\mathbf{x}^i_{k+j}(\mathbf{x}_o)) \\ \beta_j &= \sum_{i=j+1}^{r} \alpha_i \end{aligned} \tag{4.24}$$

In summary, the recursive, higher-order equation for the cost-to-go function given by (4.23) or (4.24) would allow the computation of J^i_δ using dynamic programming with r step delays. Equation (4.24) can be rewritten in a vector form as,

$$\Gamma^i_{\delta r,k} = \Upsilon^i_{rk} + A_{\delta r}\Gamma^i_{\delta r,k+1} \tag{4.25}$$

with

$$\begin{aligned} \Gamma^i_{\delta r,k} &= [\, J^i_\delta(\mathbf{x}^i_k)\; J^i_\delta(\mathbf{x}^i_{k+1})\; \ldots\; J^i_\delta(\mathbf{x}^i_{k+r-1})\,]^T \\ \Upsilon^i_{rk} &= [\, \sum_{j=0}^{r-1} \beta_j \delta^j U^i(\mathbf{x}^i_{k+j}(\mathbf{x}_o)) \quad 0\; \ldots\; 0\,]^T \end{aligned}$$

$$A_{\delta r} = \begin{bmatrix} \alpha_1\delta & \alpha_2\delta^2 & \dots & \dots & \alpha_r\delta^r \\ 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & \dots & \dots & 0 \\ \vdots & & \ddots & & 0 \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

Using (4.25) allows us to generalize Theorem 4.5 fairly easy. Following the reasoning for HDP, we would generate an estimate $\hat{\Gamma}^i_k(\hat{W}_k)$ for $\Gamma^i_{\delta r,k}$, and equations (4.7)-(4.9) would get modified to,

$$\Gamma^{i*}_k(\hat{W}_k) = \Upsilon^i_{rk} + A_{\delta r}\hat{\Gamma}^i_{k+1}(\hat{W}_k) \tag{4.26}$$

$$\epsilon_k = \Gamma^{i*}_k(\hat{W}_k) - \hat{\Gamma}^i_k(\hat{W}_k) \tag{4.27}$$

$$\hat{W}_{k+1} = \hat{W}_k - \eta_k \left(\nabla_{\hat{W}_k}\epsilon_k\right)\epsilon_k \tag{4.28}$$

Expressing the higher-order method in vector notation as a first-order equation makes easy to generalize the result of Theorem 4.5. The following theorem generalizes Theorem 4.5 in establishing the convergence properties of the weights of a radial basis network for the higher-order approximation case.

Theorem 4.8. *Given the nonlinear system* (4.1) *under the action of a prespecified control law* $g(\mathbf{x})$ *that stabilizes the system and the performance index* $J(\mathbf{x})$ *defined by* (4.3)*, assume that* $J(\mathbf{x})$ *can be expressed as a linear combination of basis functions* (4.10) *where* W *is a parameter vector and* $\Psi(.)$ *is the vector of known bounded basis functions. If we define the estimator for* $J(\mathbf{x})$ *as in* (4.11) *and use the higher-order HDP algorithm given by* (4.26)-(4.28) *to adapt the parameter vector* $\hat{W}$ *and,*

$$r^2 + \left(r - 1 + \sqrt{\sum_{i=1}^{r}\alpha_i^2\delta^{2i}}\right)^2 < \frac{2}{\eta\,\Psi_M^2}, \quad \|\Psi(.)\| < \Psi_M \tag{4.29}$$

where δ *is the discount factor in* (4.3)*,* η *is the training coefficient in* (4.28)*,* r *is the order of the HDP and* α_i *the coefficients of the convex combination in* (4.24)*, then the magnitude of the vector* $\tilde{W}_k = W - \hat{W}_k$ *is monotone nonincreasing.*

Proof. See Appendix A for the details. □

4.8 Example

To illustrate the training procedures described in this chapter we use the Van der Pol example used in § 3.6.1. Fig. 4.3 shows a tridimensional plot of the cost-to-go function (4.3) for the discretized Van der Pol system for

$$\begin{aligned} \delta &= 0.9 \\ U(\mathbf{x}) &= \mathbf{x}^t P \mathbf{x} \\ P &= 0.1I \end{aligned}$$

where I is the identity matrix. Training is performed for random trajectories starting at states be-

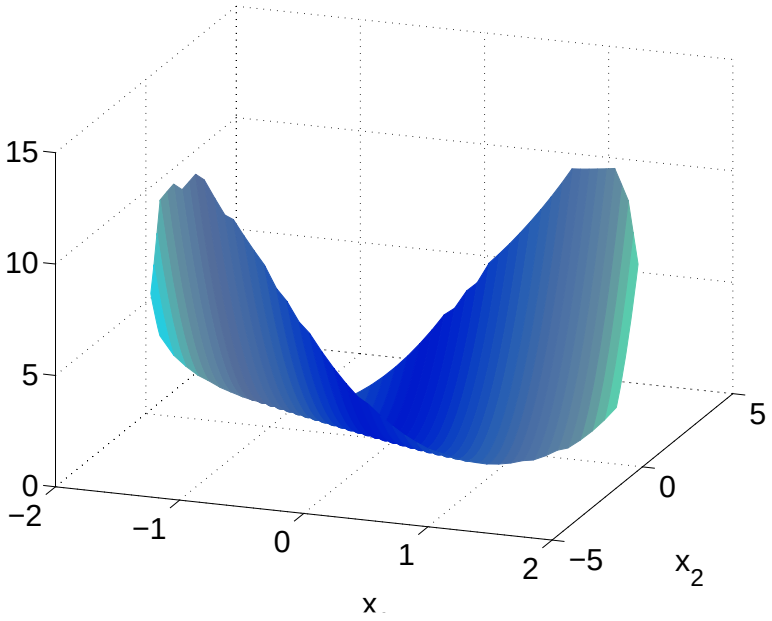

Fig. 4.3: Cost-to-go function for the Van der Pol equation.

longing to the stability region of the Van der Pol equation. The results are averaged over five independent experiments. To accelerate the estimation process, a window over the trajectories and a buffer of previous trajectories is kept to generate the patterns for the training algorithm. Table 4.1 gives information on the training procedures analyzed. Fig. 4.4 shows the neural network estimate and Fig. 4.5 shows the error surface when the first-order HDP with momentum algorithm is used. Fig. 4.6 shows the error surface when the higher-order HDP with momentum algorithm is used

Procedure	1st-order-HDP	3rd-order-HDP	Emb. Filtering
Training samples:	2300	2300	2300
Testing samples:	480	480	480
Units in each layer:	2/20/12/1	2/20/12/1	2/20/12/1
Training set MSE:	0.01	0.01	0.01
Test set MSE:	0.065	0.015	0.035
Momentum	0.9	0.9	n/a
Window size	10	10	10
Buffer size	100	100	100
Real cost MSE	0.135	0.0216	0.0565

Table 4.1: Van der Pol equation: Estimation results for the three methods.

while Fig. 4.7 shows the embedded filter estimation.

The best approximation in terms of the smaller mean squared error in a grid of points in the state space is achieved using the higher order HDP procedure. The embedded filter method gives an error surface that is smoother than the other two methods. Even though the filter operation introduced in the procedure improves the results with respect to the plain first-order method, it is still a first-order method in the sense that the error ϵ_k in Fig. 4.2 is evaluated from just one-step delayed values. With respect to computation effort, either the first -or higher- order procedures have the same order of

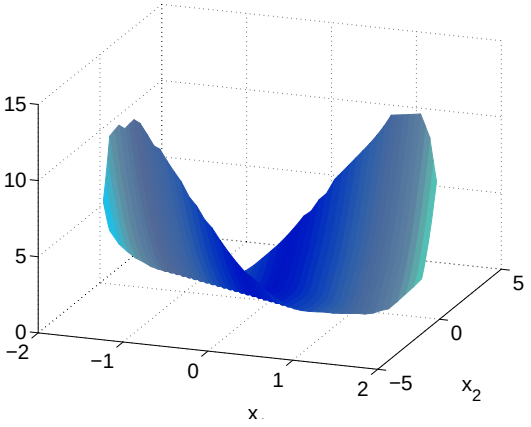

Fig. 4.4: Estimate of the performance index.

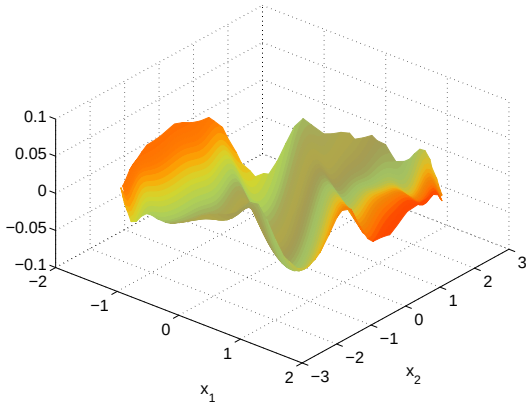

Fig. 4.5: Error surface for first-order HDP with momentum.

magnitude in computation time. For the higher-order method the only extra element required is a queue to store the r previous values of the states and cost-to-go estimates. The embedded filtering technique is considerably slower since each step involves a search for good values of λ and μ with singular value decomposition besides the usual computations.

4.9 Discussion

In this chapter we have presented methods to estimate performance indices for a closed-loop system. The performance index chosen is a cost-to-go function for the system. Properties of the cost-to-go functions are examined. The estimation is based on neurodynamic programming techniques. Some extensions are presented to cope with some of its shortcomings in terms of nonlinear function approximation and convergence. The methods are illustrated with the Van der Pol equation, showing that a good approximation may be achieved. In Chapters 6 and 7 we apply these techniques to other types of nonlinear systems.

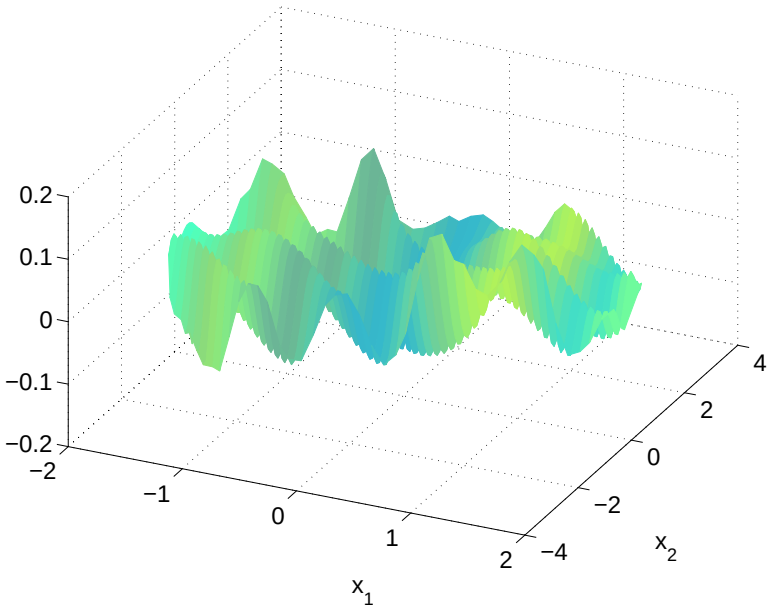

Fig. 4.6: Error surface for the higher-order HDP procedure.

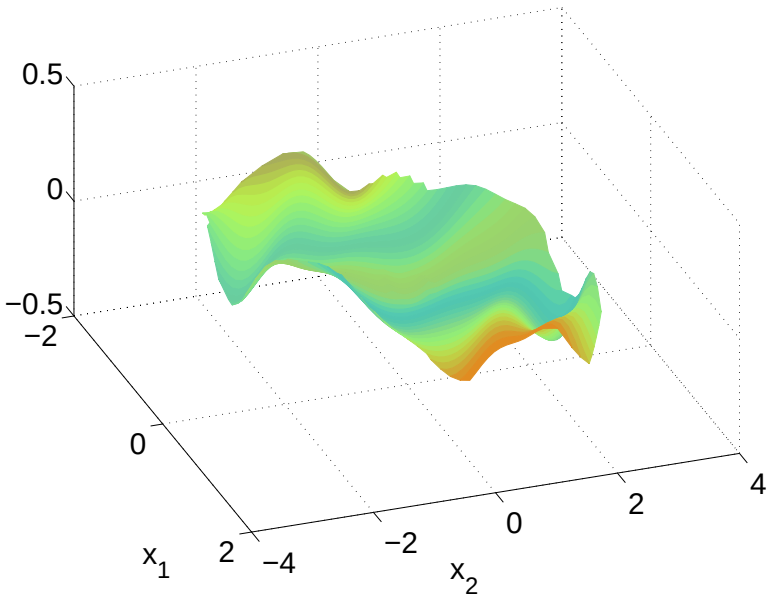

Fig. 4.7: Error surface for the embedded filtering procedure.

Chapter 5

Controller scheduling

5.1 Introduction

In this chapter, an approach for controller scheduling is proposed based on the results of Chapters 3 and 4.

§ 5.2 describes the problem and the proposed controller scheduler strategy. A theoretical analysis of the controller scheduling strategy is presented in § 5.3. Computational issues are considered in § 5.4 and results are summarized in § 5.5.

5.2 Problem description

We consider the problem of controlling a nonlinear system described by the state equations

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{x}_k \in S, &\qquad \mathbf{u}_k \in D \end{aligned} \tag{5.1}$$

where $\mathbf{x}_k \in \Re^n$ is the state vector and $\mathbf{u}_k \in \Re^m$ the control input vector. We assume the nonlinear function f is smooth, and the state vector $\mathbf{x}_k$ observable. The connected sets $S \subset \Re^n, D \subset \Re^m$

represent physical constraints on the system state and control, respectively. The control objective is to take the state to the origin.

We assume M state feedback controllers have been designed for this system, with the i^{th} control law given by $\mathbf{g}^i : \Re^n \rightarrow \Re^m$. We assume the origin is a stable equilibrium for each of the controllers in some (unknown) region. The objective of the switching strategy is to select one of the control outputs to apply the system at each control instant to achieve the largest possible region of stability for the closed-loop system with a good transient response in some sense.

The closed-loop system created by the application of each controller is characterized by a stability region and a performance index. The region of stability for controller i is defined as

$$R^i = \{ \mathbf{x}_o : \mathbf{x}_k^i(\mathbf{x}_o) \in S,\ \mathbf{g}^i(\mathbf{x}_k^i(\mathbf{x}_o)) \in D\ \forall k \geq k_o \quad \text{and} \quad \lim_{k\rightarrow\infty} \mathbf{x}_k^i(\mathbf{x}_o) = 0 \} \tag{5.2}$$

where $\mathbf{x}_k^i(\mathbf{x}_o)$ denotes the trajectory of the system (5.1) with the initial state $\mathbf{x}_o$ at $k = 0$ under control law $\mathbf{g}_k^i$. We consider performance indices for the controllers of the form

$$J_\delta^i(\mathbf{x}_o) = B^i + \sum_{k=0}^{\infty} \delta^k U^i(\mathbf{x}_k^i(\mathbf{x}_o)), \quad i = 1, \ldots, M \tag{5.3}$$

where $0 < \delta \leq 1$ is a discount factor, $U^i : \Re^n \rightarrow \Re$ is a positive definite *state cost function*, and B^i is the *bias coefficient* for the i^{th} controller. We assume (5.3) converges for $\delta = 1$.

The proposed approach for selecting the controller at each sampling instant is illustrated in Fig. 5.1. Neural networks are used to estimate the stability regions R^i and performance indices J_δ^i for the system under each control, denoted by $\hat{R}^i$ and $\hat{J}_\delta^i$, respectively. The index of the control input to be applied for the next period, denoted i_k, is then selected as

$$i_k \quad = \quad arg \min_{i \in I(\mathbf{x}_k)} \{\hat{J}^i(\mathbf{x}_k)\} \tag{5.4}$$

where

$$I(\mathbf{x}_k) \quad = \quad \{i | \mathbf{x}_k \in \hat{R}^i, \text{and if } i \neq i_{k-1}, \hat{J}^i(\mathbf{x}_k) < L_k^i\} \tag{5.5}$$

and for $i = 1, \ldots, M$

$$L_k^i = \begin{cases} \infty & \text{if } k = 0 \text{ or } \mathbf{x}_k \notin \hat{R}^{i_{k-1}} \\ L_{k-1}^i & \text{if } i \neq i_k \\ \min\{\hat{J}^i(\mathbf{x}_k), L_{k-1}^i\} & \text{if } i = i_k \end{cases}$$

In words, the scheduler selects the controller index from among the controllers for which the current state is in their estimated stability regions. The specific controller chosen corresponds to the minimum estimated performance index subject to the upper bounds L_k^i for the controllers that were not chosen for the previous period (i_{k-1}). The limits L_k^i guarantee a controller is not re-selected once it has been used until its performance index has decreased below the lowest value reached when it has been active. If the system leaves the stability region of the controller used during the previous period, all controllers become candidates again by setting $L_k^i = \infty$ for all $i = 1, \ldots, M$.

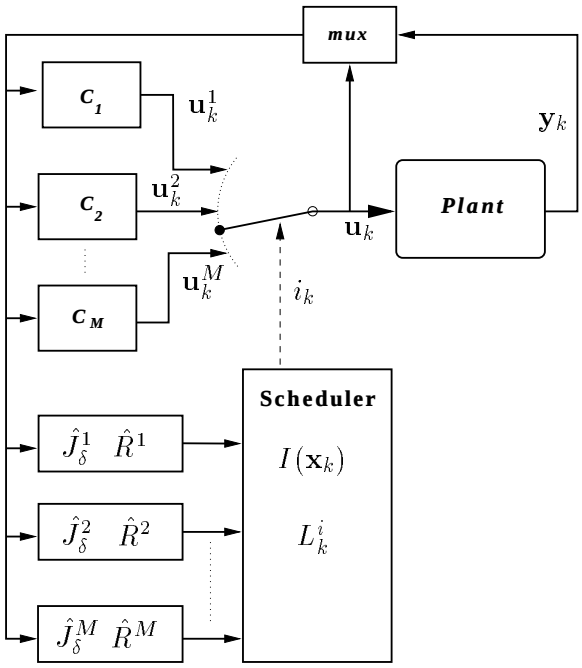

Fig. 5.1: Control Scheduling diagram.

This selection criterion is motivated by the *min-switching* strategy proposed in [MBA96] to switch among multiple controllers for a continuous time system for which there are known Lyapunov functions for the closed-loop system under each controller. The strategy proposed here replaces the Lyapunov functions with the neural network estimates of the performance measure, making it viable for systems for which the dynamics are not known precisely or Lyapunov functions cannot be found by analysis. The memory of the previous performance index values assures convergence when the neural network estimates are used (see § 5.3).

5.3 Analysis of closed-loop performance

In this section we consider the closed-loop performance of the min-switching strategy described in § 5.2. We first consider the system behavior in the perfect information case, that is, when the performance measures and stability regions are known for each controller. We then consider the effect of using the neural network estimators rather than the exact values.

The approach to analyze the closed-loop system follows the technique for the min-switching strategy suggested in [MBA96] for the continuous-time case. To restate the basic Lyapunov results for our discrete-time context, suppose for each control law g^i there is a known Lyapunov function $V^i(\mathbf{x})$ for the closed-loop system under that control law within the region of stability R^i. Moreover, suppose the control applied at each sample instant is chosen according to the min-switching strategy, that is, the control is selected which corresponds to a Lyapunov function with the minimum value among all the Lyapunov functions evaluated at the current state. The following theorem is the discrete-time version of the result in [MBA96].

Theorem 5.1. *If the system given by* (5.1) *is controlled by the min-switching strategy applied to a set of known Lyapunov functions, the origin is asymptotically stable in the region* $R = \bigcup R^i$.

Moreover, the function

$$W(\mathbf{x}) \quad = \quad \min_{i \in \{j \mid \mathbf{x} \in R^j\}} \{V^i(\mathbf{x})\} \tag{5.6}$$

is a Lyapunov function on R.

Proof. Follows from the continuous-time result in [MBA96], *mutatis mutandis*. □

In the case of discrete-time control, chattering between controllers does not introduce the technical difficulties in the stability proof that arises in the continuous-time case.

We now apply this result to the min-switching strategy considered in this paper by observing that if the performance indices J_1^i (5.3) converge ($\delta = 1$), they are in fact Lyapunov functions for the respective controllers. This follows from the the fact that

$$\Delta J_\delta^i(\mathbf{x}) \stackrel{\triangle}{=} J_\delta^i(F^i(\mathbf{x})) - J_\delta^i(\mathbf{x}) = (1-\delta)J_\delta^i(F^i(\mathbf{x})) - U(\mathbf{x}) \tag{5.7}$$

where $F^i(\mathbf{x}) = f(\mathbf{x}, g^i(\mathbf{x}))$. We have then the following result.

Theorem 5.2. *Given the system defined by* (5.1) *and a collection of control laws* $g^i, i = 1, \ldots, M$. *Suppose for each control law the origin is asymptotically stable for the closed-loop system*

$$\mathbf{x}_{k+1} = F^i(\mathbf{x}_k) = f(\mathbf{x}_k, g^i(\mathbf{x}_k))$$

in a connected region R^i, *and* J_δ^i *given by* (5.3) *converges for* $\delta = 1$. *Then there exists a scalar value* $\delta^* \in (0,1)$ *such that the origin is asymptotically stable in the region* $R = \bigcup R^i$ *for the closed-loop system controlled by the min-switching strategy* [MBA96] *for any* $\delta \in (\delta^*, 1]$. *Moreover, for a given* $\delta \in (\delta^*, 1]$ *the function*

$$J_\delta(\mathbf{x}) \quad = \quad \min_{i \in \{j \mid \mathbf{x} \in R^j\}} \{J_\delta^i(\mathbf{x})\}$$

is a Lyapunov function on R.

Proof. For each i, if J_δ^i converges, it follows from the definition and properties J_δ^i discussed in § 4.3 that it is continuous in δ and therefore there exists some $\delta_i^* \in (0,1)$ such that for all $\delta \in (\delta_i^*, 1]$ J_δ^i is a Lyapunov function for the closed-loop system under control law i on R^i. The theorem follows by letting $\delta^* = max(\delta_1^*, \ldots, \delta_M^*)$. □

We now turn to the min-switching strategy using the neural network estimators. In the following we assume the stability region estimators are all conservative, that is, for all $i = 1, \ldots, M$, $\hat{R}^i \subseteq R^i$. Moreover, we assume all the stability region estimates are nonempty and connected. These assumptions are reasonable given the properties of neural network classifiers and the ability to initiate the training for the stability region estimators based on a priori knowledge of the capabilities of the given controllers.

Theorem 5.3. *Suppose the assumptions of Theorem 5.2 are satisfied and the performance estimates are computed using $\delta \in (\delta^*, 1]$ where δ^* is as specified in Theorem 5.2. Furthermore, suppose J_δ^i is continuous on R^i. If for some $\epsilon > 0$ the performance estimates satisfy*

$$|\hat{J}_\delta^i(\mathbf{x}) - J_\delta^i(\mathbf{x})| < \epsilon \quad \text{for all } \mathbf{x} \in \hat{R}^i,\ i = 1, \ldots, M \tag{5.8}$$

and the min-switching strategy is applied for some $\mathbf{x}_o \in \hat{R} = \bigcup \hat{R}^i$ resulting in a state trajectory such that there exists $K \in \mathcal{N}$ for which $\mathbf{x}_k \in \bigcap \hat{R}^i, \forall k > K$, then $\mathbf{x}_k \to X_\epsilon$ where

$$X_\epsilon \;=\; \bigcup_i X_\epsilon^i \;=\; \bigcup_i \{\mathbf{x} \mid J_\delta^i(\mathbf{x}) \leq \sup_{\tilde{\mathbf{x}} \in \chi_\epsilon^i} J_\delta^i(\tilde{\mathbf{x}})\} \tag{5.9}$$

and

$$\chi_\epsilon^i \;=\; \{\mathbf{x} \in R^i \mid -\Delta J_\delta^i(\mathbf{x}) \leq 2\epsilon\}.$$

with $\Delta J_\delta^i(\mathbf{x})$ given by (5.7).

Proof. See Appendix A. □

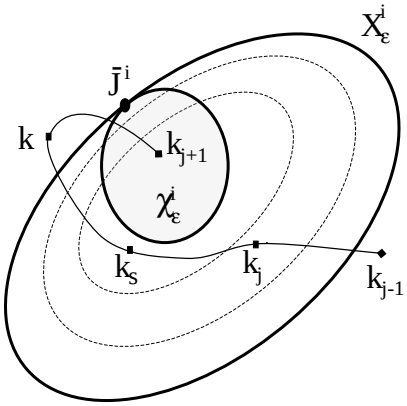

Fig. 5.2: Trajectory of the system illustrating convergence to X_ϵ.

Fig. 5.2 illustrates Theorem 5.3. The sets χ_ϵ^i and X_ϵ^i for the i^{th} controller are displayed. $\bar{J}^i$ is the supremum of the cost-to-go function J_δ^i on χ_ϵ^i. Some contour lines for smaller values of J_δ^i are also shown. A trajectory is displayed, where k_j represents the time steps where the scheduler switches to controller i. When the controller algorithm switches to another controller at k_s, the system may leave X_ϵ^i, represented by the step k, but must return to $X_\epsilon^i \subset X_\epsilon$ for controller i to be applied again. Definition (5.9) indicates that a similar figure applies for each of the controllers.

However, Theorem 5.3 does not guarantee the neighborhood X_ϵ converges to the origin. Moreover, the exact performance measures are not known in general, so the neighborhoods χ_ϵ^i could not be computed even if the bound on the estimation errors was known. Using the error bound ϵ it is possible to find a conservative estimate for X_ϵ. First, we can find an estimate for χ_ϵ if we write $\Delta J_\delta^i(\mathbf{x})$ as

$$\Delta J_\delta^i(\mathbf{x}) \stackrel{\triangle}{=} J_\delta^i(F^i(\mathbf{x})) - J_\delta^i(\mathbf{x}) = \frac{(1-\delta)}{\delta} J_\delta^i(\mathbf{x}) - \frac{1}{\delta} U(\mathbf{x})$$

Then, the condition for a point in the state space to belong to the set χ_ϵ^i can be written as

$$U(\mathbf{x}) - (1-\delta) J_\delta^i(\mathbf{x}) \quad \leq \quad 2\epsilon\delta \tag{5.10}$$

Even though (5.10) depends only on a particular state and not a trajectory, it is still not possible to compute since J_δ^i is not known exactly. An approximate expression can be obtained by using $\hat{J}_\delta^i$ instead.

$$\begin{aligned} U(\mathbf{x}) - (1-\delta)(\hat{J}_\delta^i(\mathbf{x}) + \epsilon) &\leq U(\mathbf{x}) - (1-\delta)J_\delta^i(\mathbf{x}) \leq 2\epsilon\delta \\ U(\mathbf{x}) - (1-\delta)\hat{J}_\delta^i(\mathbf{x}) &\leq \epsilon(1+\delta) \end{aligned}$$

Therefore, a conservative estimate for the set χ_ϵ^i is

$$\hat{\chi}_\epsilon^i = \{\mathbf{x} \in R^i | \quad U(\mathbf{x}) - (1-\delta)\hat{J}_\delta^i(\mathbf{x}) \leq \epsilon(1+\delta)\}. \tag{5.11}$$

A similar modification gives us the estimate of the convergence set X_ϵ as

$$\hat{X}_\epsilon = \bigcup_i \hat{X}_\epsilon^i = \bigcup_i \{\mathbf{x} \mid \hat{J}_\delta^i(\mathbf{x}) \leq \sup_{\tilde{\mathbf{x}} \in \hat{\chi}_\epsilon^i} \hat{J}_\delta^i(\tilde{\mathbf{x}}) + \epsilon\} \tag{5.12}$$

The estimation errors incurred by using (5.11) and (5.12) are eliminated when $\delta = 1$, however, since in this case we have $\Delta J_\delta^i(\mathbf{x}) = -U(\mathbf{x})$. In other words, when $\delta = 1$, the decrease in the exact cost at each stage is determined entirely by the state cost function. This observation leads to the following corollary.

Corollary 5.4. *Under the assumptions of Theorem 5.3 with $\delta = 1$, if the min-switching strategy is applied for some $x_o \in \hat{R} = \bigcup \hat{R}^i$ resulting in a state trajectory such that $x_k \to \bigcap \hat{R}^i$, then*

$$\mathbf{x}_k \to \{\mathbf{x} \in \Re^n \mid U(\mathbf{x}) \leq 2\epsilon\}$$

This corollary makes a very intriguing claim and allows us to have a direct relationship between the accuracy of the cost-to-go estimation with the size of the set to which the trajectories converge.

It is important also to note that Theorem 5.3 is a sort of worst-case result by establishing the set to which the system converges upon infinite changes with the control scheduling algorithm. The more likely case in which, after some time, a controller dominates over the rest will make the system

to inherit the performance properties of that particular controller. In particular, the system will be asymptotically stable if the switching rule stays with one of the controllers indefinitely.

The results of Chapter 4 on the estimation of performance indices by neural networks suggests that it is possible to improve the estimation of the performance indices for each controller while the system is running. Since only one controller is running at a time, the on-line training should be done only with the current controller. Applying the training algorithms described in Chapter 4, the error bound ϵ with respect to the true value of the cost-to-go function becomes a nonincreasing sequence. This change does not affect the proof in Theorem 5.3 and implies an improvement in the convergence result. An estimation for the bound of the error in the estimation of the cost-to-go functions for each controller can be computed using the confidence interval computation introduced in § 2.6.5.

5.4 Computational issues

There are several issues concerning the implementation of the controller scheduler algorithm. The implementation is structured in three parts: the estimation of the stability region for each controller, the estimation of the performance indices for each controller, and the scheduler algorithm given by (5.4) and (5.5). For the first two parts, we can differentiate two different stages, the learning and evaluation procedures. The learning stage is the most expensive one, in terms of memory and computation time, while the evaluation stage is much faster. As discussed in Chapters 3 and 4, the computation time and memory depends on the size of the networks and training patterns to use. The last part of the controller scheduler computation, the scheduler algorithm in itself, is the least expensive one because it requires only to keep memory of the previous bounds, the admissible controller set and previous controller applied, and a comparison of the performance indices to select the controller to use at each stage.

If we eliminate the on-line learning procedure for the neural network estimates, the whole controller scheduler process takes a computational effort that is roughly linear on the number of weights of the

networks used. If we keep the learning process on-line, that number grows roughly proportional to the square of the number of weights times the number of points retained for training in the window of on-line data.

In a real-time system it may not be possible to allocate all those computational resources to the controller scheduler process. Since it may be important to keep the learning procedures while in operation, they can be executed in another loop that run at a different rate and with different priority, and update its results to the on-line estimators.

5.5 Discussion

Looking back at Chapter 2 we can discuss the improvements and trade-offs gained by using the method explained in this chapter with respect to the controller scheduler methods described there.

The method developed by Morse [Mor96] for supervisory control presents all of its results on the basis of a family of linear systems while we can manage a framework for a class of nonlinear systems. With respect to the approach by Kolmanovsky and Gilbert [KG97] we present an estimation for the stability regions, which are invariant, and also an estimate of a performance for the controllers rather than using a possible arbitrary order of preference for them. Malmborg [MBA96] and Branicky [Bra94b] stated sufficient conditions for stability in the sense of Lyapunov but no further considerations with respect to performance are made.

In Chapters 6 and 7 two examples are worked out to show the properties of this approach to controller scheduling.

Chapter 6

Example: Inverted pendulum

6.1 Introduction

As an example of the methods developed in the previous chapters we consider the well known inverted pendulum (IP) problem. We use a physical IP that has served as a standard laboratory example for the SIMPLEX architecture described in § 2.7. In this chapter we present results of the neural network estimation of the stability region and performance indices, and controller scheduling experiments for three different controllers based on data from a simulation model.

6.2 Description

Fig. 6.1 shows a model scheme for a standard IP. The IP model equations used are:

$$\begin{aligned} J_t\ddot{x} + \tfrac{1}{2}\, ml\, cos\alpha\, \ddot{\alpha} + B_r\dot{x} - \tfrac{1}{2}\, ml\, sin\alpha\, \dot{\alpha}^2 &= F \\ \tfrac{1}{2}\, m\, cos\alpha\, \ddot{x} + \tfrac{1}{3} ml\ddot{\alpha} - \tfrac{1}{2}\, mg\, sin\alpha &= 0 \end{aligned} \tag{6.1}$$

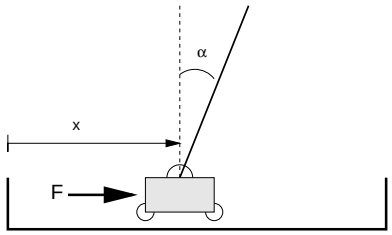

Fig. 6.1: Inverted pendulum model scheme.

with the following constraints:

$$\begin{aligned} |F| &\leq F_{max} \\ |x| &\leq x_{max} \\ |\dot{x}| &\leq v_{max} \end{aligned}$$

where α is the angle of the pendulum, x is the position of the pendulum base, m is the effective mass at the end of the pendulum, l is the pendulum length, J_t is the inertia of the base, B_r is the friction coefficient for the base, and g is the gravitational acceleration.

The parameters of the model were determined through identification experiments performed on the physical system situated at the SEI. The force F is implemented by an electrical motor. In these experiments, the dynamic of the motor is neglected since it has a time constant at least two orders of magnitude smaller than the mechanical time constant. Therefore, the approximation used is that F is proportional to the electrical current in the motor with constant K_i. The system parameters are given in Table 6.1.

Controllers

The main purpose of this example is to demonstrate how the neural networks are used in the scheduling algorithm. Towards this end, three controllers were designed to achieve stabilization of the pendulum around the origin, each one designed to work differently and in such a way that from the

J_t	0.6650	Kg	g	9.8	m/s^2
m	0.21	Kg	F_{max}	2	N
l	0.61	m	x_{max}	2.0	m
K_i	4.47	N/Amp	v_{max}	3.0	m/s
B_r	0.1	Kg/s			

Table 6.1: IP model parameters.

combination of the controllers we expect to get better performance than just by the application of any one of them.

The first control algorithm applied is a standard LQR controller for the linearized model of the IP around the origin. The matrices Q and R selected are:

$$R = 1, \quad Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The feedback control law obtained is then

$$\begin{aligned} \mathbf{u} &= K\mathbf{x}, \\ K &= [10.0\ 12.60\ 48.33\ 9.09] \\ \mathbf{x} &= [x\ \dot{x}\ \alpha\ \dot{\alpha}]^T \end{aligned}$$

The LQR controller does not have a very large domain of stability since it is based on a linearization of the dynamics. Therefore, a sliding mode (SM) controller is designed to bring α to zero as fast as possible using a nonlinear switching rule and saturating control. The sliding surface is simply defined as

$$s(t) = \left(\frac{d}{dt} + \lambda\right)\alpha$$

Furthermore, to avoid chattering the saturation function of bandwidth ϕ is used instead of a sign function. The SM controller takes the form

$$\mathbf{u} = \hat{u}(\mathbf{x}, \lambda) + \gamma sat\left(\frac{s(t)}{\phi}\right)$$

where $\hat{u}$ is the nominal control action that satisfies the equation $\dot{s}(t) = 0$ and

$$\lambda = 5, \quad \phi = 0.1, \quad \gamma = 20$$

The SM controller is designed to bring the pendulum angle to zero. However, it may do so with a nonzero terminal base speed. Therefore, a third controller is designed to bring the base to rest, although not necessarily at $x = 0$. This controller is just another LQR feedback controller but with the gain for the base position set to zero, so we call it the *velocity feedback* (VF) controller. The control law is given by

$$\mathbf{u} = K\mathbf{x}, \quad K = [0 \quad 30.92 \quad 87.63 \quad 20.40].$$

6.3 Stability region estimation

In this section we present results of the neural network estimation of the stability region, for the LQR controller defined in the previous section, based on data from a simulation model. It should be noted that the SM controller only controls the angle θ, so its region of stability was determined without considering the track position and velocity. The VF controller is a stable controller but not asymptotically stable. Its region of stability was determined the same way as for the LQR controller.

First, we show results and comparison with the stability region determined experimentally on the physical system. Afterwards, a comparison with a LMI approach described in § 2.3.2 is presented based on simulation models.

The methods developed in Chapter 3 were used to train a neural network to estimate the region of stability for the IP system. Table 6.2 summarizes some parameters calculated for the network

and some results obtained on the simulator. Figures 6.2 and 6.3 show some cross-sections of the region found for the model described in § 6.2, compared to the best region based on constraints for individual state variables, calculated by maximizing the volume of the hypercube defined by the constraints. One motive for this comparison lies in the fact that a hypercube of this sort was used in the experimental model at the SEI to approximate the stability region. As expected, there is a large difference between the regions in favor of the neural network estimate.

Training samples:	1000
Testing samples:	300
Units in each layer:	4 - 12 - 6 - 2
Training set MSE:	0.01
Test set MSE:	0.05
θ, δ :	1.8 / 0.9

Table 6.2: Network estimation results

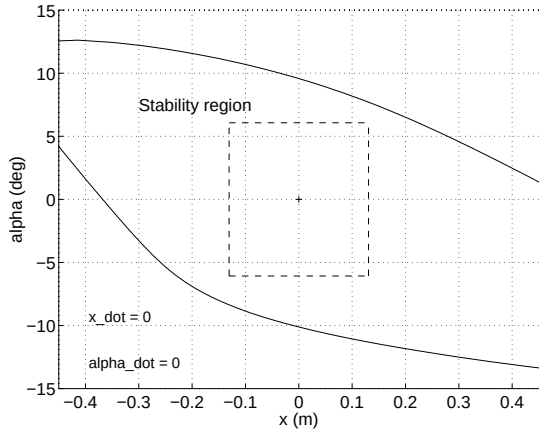

Fig. 6.2: State space cut showing the estimated stable region in the (x, α) plane for $\dot{x} = \dot{\alpha} = 0$.

solid	neural network estimation.
dashed:	best hypercube constraints.

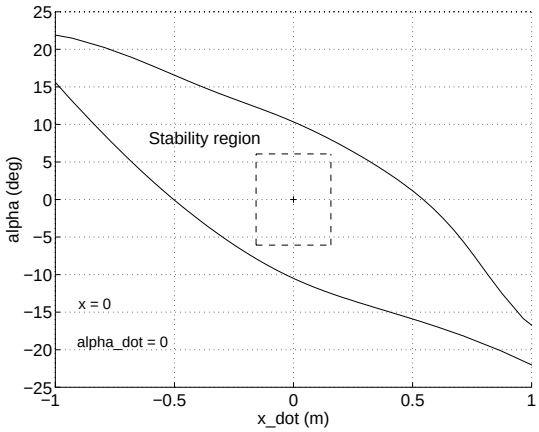

Fig. 6.3: State space cut showing the estimated stable region in the $(\dot{x}, \alpha)$ plane for $x = \dot{\alpha} = 0$.

solid	neural network estimation.
dashed:	best hypercube constraints.

6.3.1 Comparison with a LMI method

An LMI procedure has been recently implemented by Seto and Sha to look for a conservative stability region of an inverted pendulum and a double-inverted pendulum in development at the SEI. Based on the model equations (6.1), but with other parameter values coming from a different physical system, the LMI procedure described in § 2.3.2 is carried out. Figures 6.4 and 6.5 show two different two-dimensional cross-sections of the four-dimensional state space with a comparison of the ellipsoid generated by the LMI procedure and the neural network stability region estimate. As expected the LMI approach gives a more conservative region than the neural network, although the gap is greatly reduced with respect to the hypercube example shown in the first comparison. Fig. 6.6 gives a comparison in three dimensions. In this case we cut the four-dimensional space with the hyperplane $x = 0$ since we are looking at taking the system to the origin.

The stability region approximation shows that a neural network can have a better approximation than

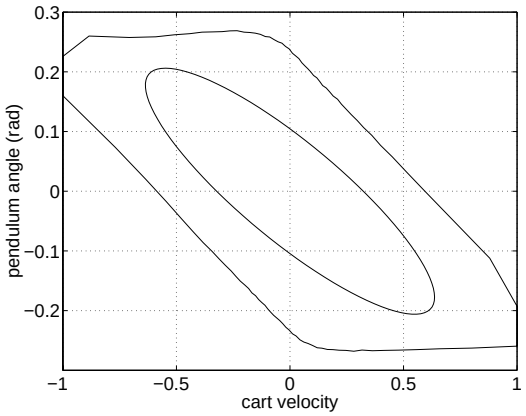

Fig. 6.4: Slice of the state space showing NN and LMI estimates.

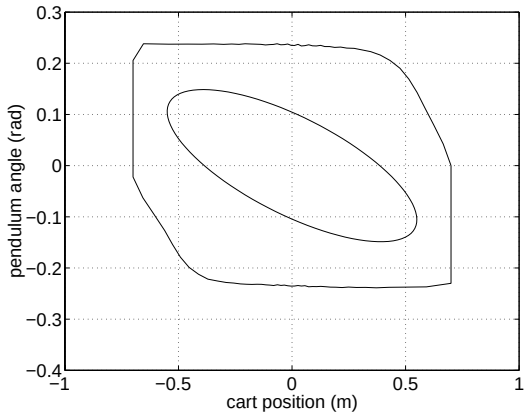

Fig. 6.5: Slice of the state space showing NN and LMI estimates.

the LMI estimation based on a linearization method. However, the LMI method is computationally very fast relative to the neural network training.

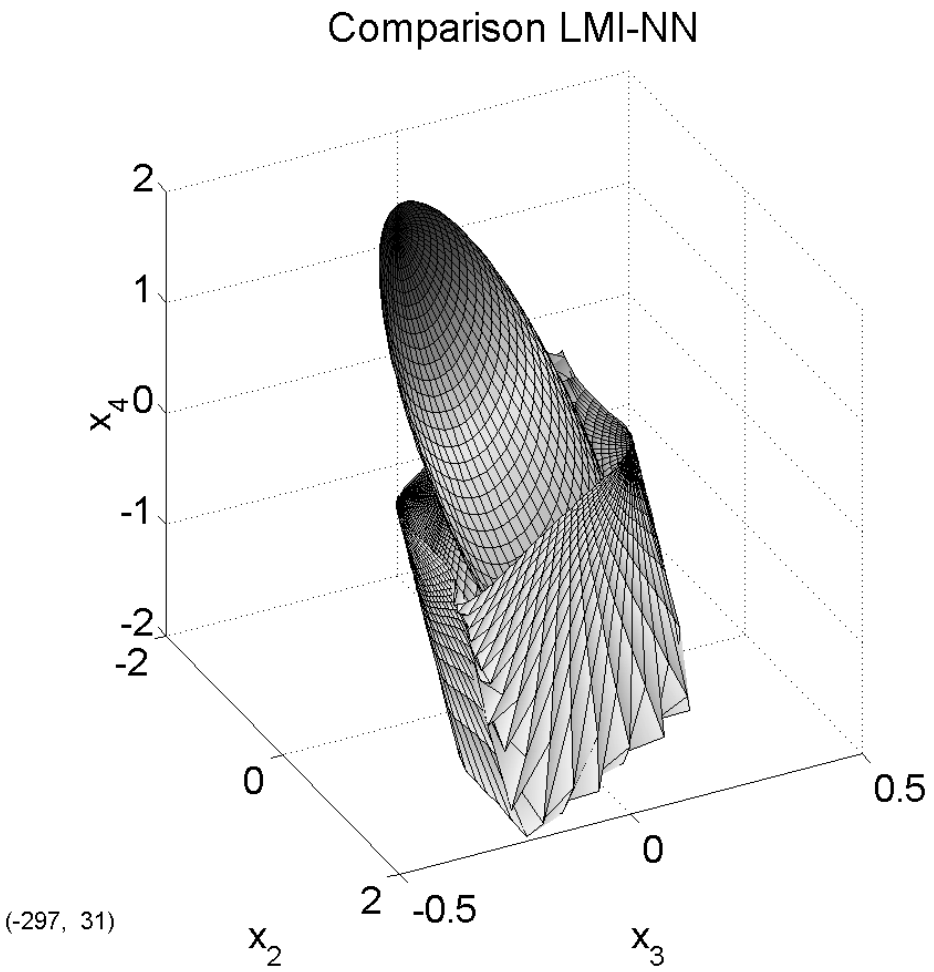

Fig. 6.6: Three dimensional plot (for $x = 0$) comparing NN and LMI estimates for the stability region.

6.4 Performance index estimation

Given the three controllers described in § 6.2, three neural networks were trained to approximate the cost-to-go function of each controller. Training was performed off-line in each case using 5000 random trajectories, followed by continued on-line during the simulated control experiments using HDP. The discount factor δ and the cost matrix P chosen were

$$\delta = 0.5, \qquad P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multilayer feedforward networks were used together with conjugate gradient optimization methods.

Figures 6.7, 6.8 and 6.9 show slices of the approximated performance measure in a 3D plot for each of the controllers with the angle rate and base position set to zero. Note that these relative index measures are intuitive in their shapes for each controller, and the relative values are also intuitive in that the SM controller obtains a better performance for larger values of α, but is not as good as the LQR controller for smaller angles.

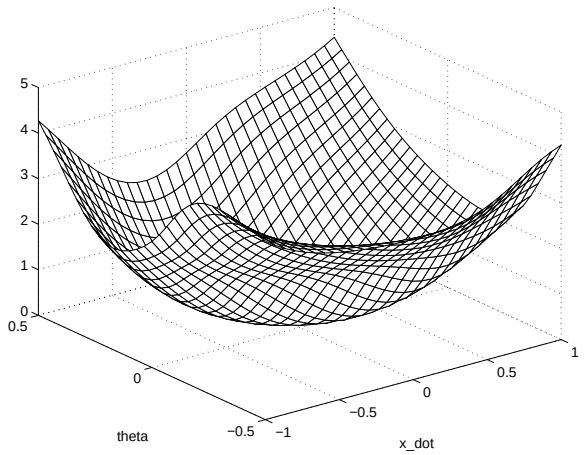

Fig. 6.7: Cut of the neural network estimation for the performance of the LQR controller (LQR).

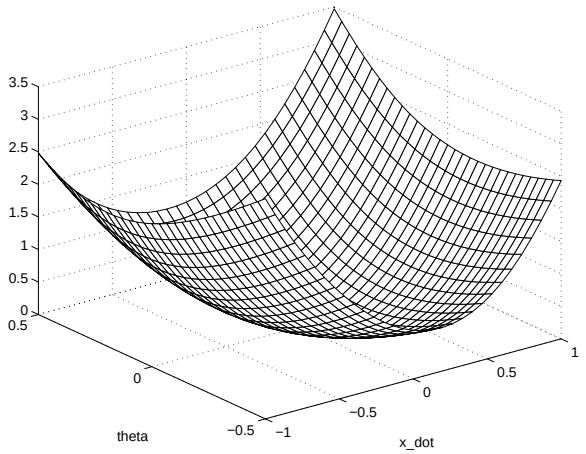

Fig. 6.8: Cut of the neural network estimation for the performance of the sliding controller (SM).

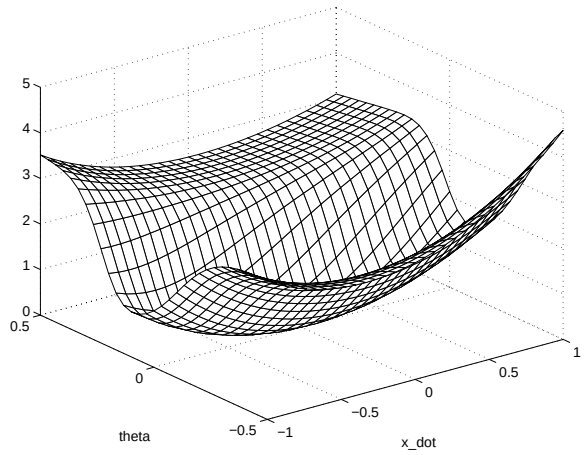

Fig. 6.9: Cut of the neural network estimation for the performance of the velocity feedback controller (VF).

6.5 Controller scheduling experiments

Having the estimates setup for the stability region and performance index for each controller implemented, the controller scheduling experiments using the strategy developed in Chapter 5 are shown.

Different types of simulation experiments were run on the system. In the first experiments the initial condition used in the trajectory simulation is $\alpha = 0.5$. All other state variables equal zero initially. As shown in Fig. 6.10, for this initial state: the LQR controller cannot stabilize the system; if the SM controller is applied alone, the angle goes to zero, but the final base speed is nonzero; and the VF controller will not drive the base position to zero, even if it can stabilize the pendulum and bring the base to rest.

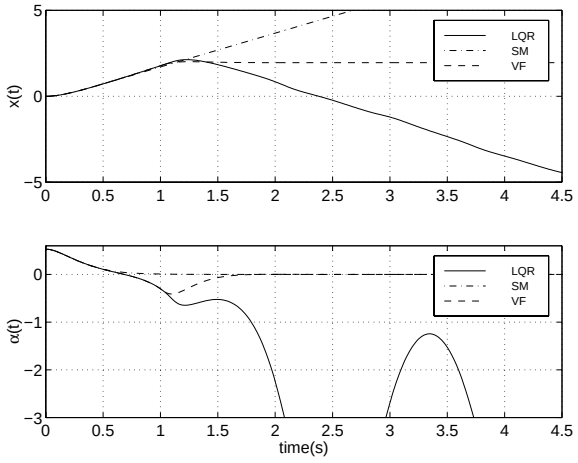

Fig. 6.10: Arbitrary trajectory showing the system being controlled by each designed controller by itself.

Fig. 6.11 shows the position trajectory from the same initial condition along with the corresponding switches between controllers when the minimum switching rule is applied to the system. In this experiment the scheduler simply chooses the controller corresponding to the minimum cost without applying the constraints L_k. The noise in the state measurements plus the neural estimation

and switching scheme applying the controller with the minimum performance estimation lead to the chattering observed in the controller selection. Note that the controller scheduling obtains an asymptotically stable behavior for a situation in which none of the controllers is acceptable by itself but the performance is very poor. Fig. 6.12 shows the result of the simulation when we use a

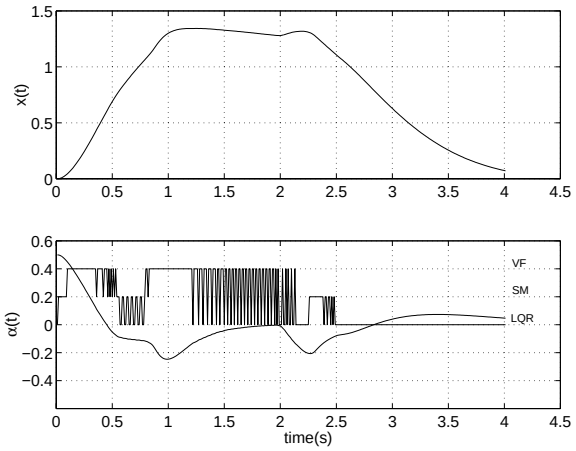

Fig. 6.11: Arbitrary trajectory showing the system behavior under the minimum switching rule control scheme.

dwell-time in the switching scheme to reduce chattering. The delay parameters are tuned to reduce the chattering. There are some trade-offs in the selection of these values. The parameter k_m looks into reducing the frequency of chattering while k_l tries to avoid switches related to estimation errors in the cost function approximation. Large values for the parameters may cause the system to have large oscillations around the slide surface or not to switch at all. The final values chosen are: $k_m = 4$, $k_l = 2$. We observe that the chatter has been reduced as expected. We also can see that the evolution of the combine cost-to-go estimate is not monotonically decreasing for all time.

In the next simulation we introduce the full controller scheduling rules developed in Chapter 5, including the L_k terms. Fig. 6.13 shows the position of the cart, the angle of the pendulum and the switches between the controllers for a simulation experiment when using the controller scheduling

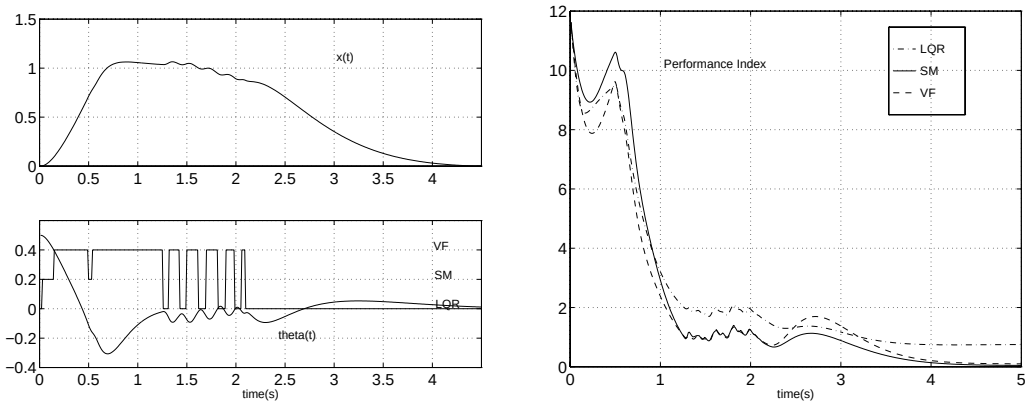

Fig. 6.12: Trajectory showing system behavior with inclusion of a chattering reduction mechanism and performance index for each controller. ($k_m = 4, k_l = 2$)

algorithm described in § 5.2. Fig. 6.14 shows the evolution of the performance index estimates for each controller in the run.

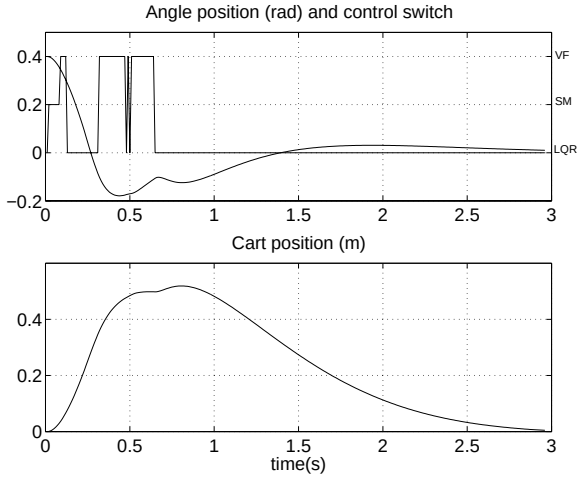

Fig. 6.13: Simulation using the neural network controller scheduling rule.

For the controller scheduling experiments, we compare the results of the proposed algorithm with

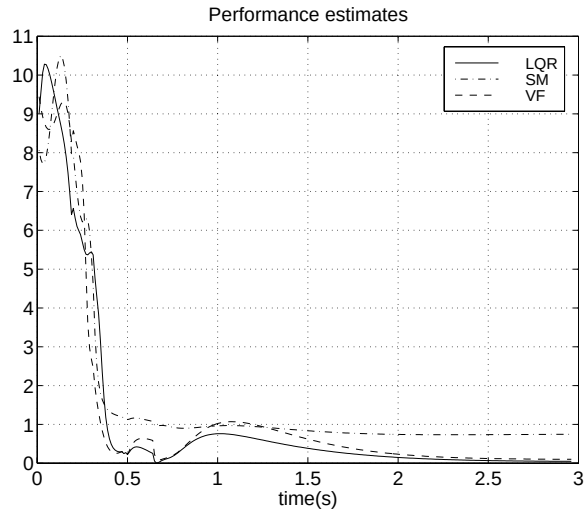

Fig. 6.14: Evolution of the performance indices for each controller for the neural network controller scheduling approach.

the *min-switching* rule of Malmborg et al. and even introducing a dwell-time to improve the chattering. Comparing the simulation experiments of the previous sections we observe that the new controller scheduling approach reduces the chattering problem in the switches. This advantage seems to come from the restriction to use a controller that is in the admissible controller set $I(\mathbf{x})$ defined in (5.5) at each time step. Computationally, there is no much overhead added to the controller code, just a memory to keep track of the previous decision and the admissible set of controllers at the current time step. In all experiments, the scheduler settles with the LQR controller so that the asymptotic behavior is that of the LQR controller.

Chapter 7

Example: Submerged vessel

7.1 Description

A second example to demonstrate the abilities of the proposed controller scheduling approach is taken from another laboratory system at the SEI. Fig. 7.1 shows a schematic of the Bubble-system designed at the SEI. Fig. 7.2 shows a diagram of the bubble system including the main variables involved in the system. The goal is to stabilize the diver to an arbitrary position inside the water tank by modifying the size of the bubble in the diver using a piston. One characteristic of this system is that any equilibrium position is open-loop unstable since the bubble inside the diver tends to grow when the diver is going up by differential pressure and vice versa, creating a positive feedback. There is also a significant time delay in the response due to the long tube connecting the piston with the bubble, which acts as a transmission line. The long tube connecting the diver with the piston also introduces a variable reacting force depending on its shape during any particular run. Another setup characteristic is the friction between the diver and the central bar that maintains the diver in a vertical movement. This static friction causes oscillations around the setpoints.

The model for such system is,

$$\begin{aligned} \ddot{y} &= -\tfrac{1}{h_e}|\dot{y}|\dot{y} - \tfrac{g}{h_e}h - f_r(y,\dot{y}) \\ \dot{h} &= -\alpha(y,h) - \beta(y,h)u(t-\tau) \end{aligned} \tag{7.1}$$

with

$$\begin{aligned} f_r(\dot{y}) &= f_1\dot{y} + f_2\ sgn(\dot{y})\ e^{-f_3|\dot{y}|} \\ \alpha(y,h) &= \tfrac{p_{at}\rho g V_o}{p_{at}\rho g V_o(1+\frac{A_c}{A_t})+p_a^2} \\ \beta(y,h) &= \tfrac{A_p p_a^2}{p_{at}\rho g V_o(1+\frac{A_c}{A_t})+p_a^2} \\ h_e &= \tfrac{m}{\rho A_c} \\ p_a &= p_{at} + \rho g(y + h_e - z_e + H_c + h(1+\tfrac{A_c}{A_t}) \end{aligned} \tag{7.2}$$

where y is the position of the diver with respect to the water level and h is the height of the bubble.

Table 7.1 explains the rest of the parameters in (7.1) and (7.2). The parameters of the model (7.1) were determined through experimentation or direct measurement. The estimated values in SI units are shown in Table 7.1. For the first equation governing y, the position of the diver, the first term of the right hand side is the water resistance, the second term reflects Archimedes' principle, and the last term is the Newtonian static and dynamic friction. The input is the velocity of the piston that pushes the air between the diver and the cylinder.

This equipment is controlled using the real-time SIMPLEX architecture described in § 2.7. The current configuration of the SIMPLEX has heuristic rules to solve the switching problem between controllers which are very difficult to create and maintain, even for small systems.

Controllers

In order to control the position of the diver in the tank, two state feedback controllers were designed. It is very difficult to come up with one single controller that works well in the whole range of operation. Because of that, one controller was designed to work well under small perturbations

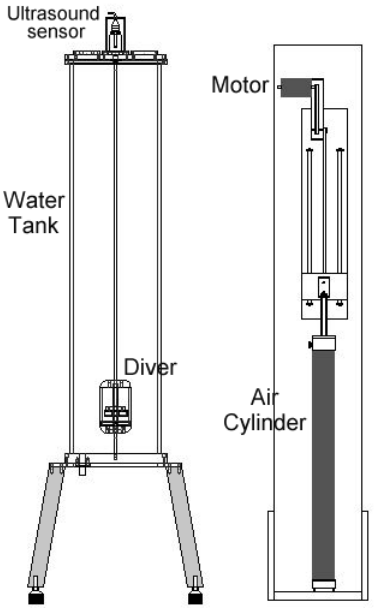

Fig. 7.1: Sketch of the components of the submerged vessel system.

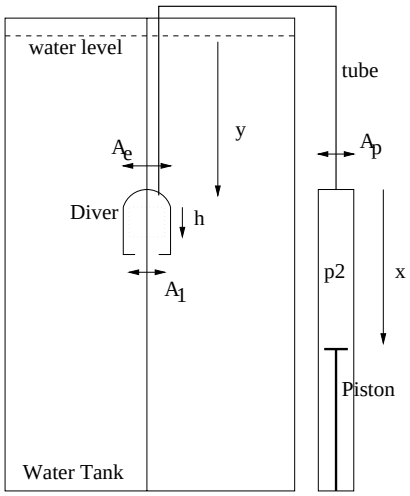

Fig. 7.2: Physical parameters for the submerged vessel system.

of the system from the setpoint while the other controller should handle large deviations from the setpoint.

Both controllers are linear state feedback controllers used in the original SIMPLEX architecture implementation. One controller, which we called $\mathbf{u}^1$, has an acceptable performance close to the set point while the second controller $\mathbf{u}^2$ performs better in a larger operating region using a bang-bang action, but with unacceptable oscillations near the set point.

Both controllers are designed by linearizing the model (7.1) around the setpoint value y_{ref} and use

h_e	0.018	m	equilibrium bubble size
τ	0.3	s	time delay
m	0.108	Kg	diver cup net weight
ρ	1000	$\mathrm{Kg}\ m^{-3}$	water density
ρ_p	1190	$\mathrm{Kg}\ m^{-3}$	plexiglass density
p_{at}	101250	Pa	atmospheric pressure
g	9.8	m/s^2	gravity acceleration
A_c	0.00597	m^2	diver cup cross-section
A_t	0.0670	m^2	water tank cross-section
z_e	0.0127	m	water level with respect to tank
H_c	0.127	m	diver cup height
f_1	0.1	Ns/m	dynamic friction coefficient
f_2	0.5	N	static friction coefficient
f_3	20	s/m	static friction coefficient

Table 7.1: Bubble system model parameters.

LQR design. The controllers can be expressed by

$$\begin{aligned}
\mathbf{u}^i &= sat(K^i\mathbf{x}, u_m), \qquad i = 1, 2 \\
x &= [y\ \dot{y}\ h\ ]^T \\
K^1 &= [0.1 \quad 0.12278 \quad -8.3964]^T \\
K^2 &= [0.81623 \quad 0.41854 \quad -30.62843]^T \\
u_m &= 1.25
\end{aligned}$$

where $sat()$ is the linear function with saturation at $\pm u_m$. The larger gains of controller $\mathbf{u}^2$ cause the control to saturate faster than controller $\mathbf{u}^1$, turning into a bang-bang control not far away from the setpoint.

7.2 Stability region estimation

For each controller, the stability region was estimated using the methods developed in Chapter 3. The trajectories were generated based on the model described in § 7.1. The setpoint value for the diver cup $y_{\rm ref}$ was added as an additional input to the neural network to be able to handle different setpoints.

Table 7.2 summarizes the results of the training procedures. Fig. 7.3 shows a section of the stability region for a setpoint value $y_{\rm ref} = 25\ in$, for controller 1, designed to work on a smaller region of the state space. Fig. 7.4 shows the stability region for controller 2. The same setpoint and slice of the state space is selected for both figures.

Controller:	Ctrl 1	Ctrl 2
Training samples:	2000	2000
Testing samples:	400	400
Units in each layer:	4/10/10/2	4/10/10/2
Training set MSE:	0.01	0.01
Test set MSE:	0.071	0.048
θ, δ :	1.7 / 9.0	1.6 / 5.6
σ_v :	5 %	5 %

Table 7.2: Network estimation results for the stability region.

Both estimators were trained to a similar accuracy. The estimates of the stability regions are more conservative than in the simple examples of Chapter 3 because the confidence intervals are generated with larger values for the variances of the errors in the states of the bubble system. The value for the variance σ_v for the states of the bubble is computed from trajectory data from the physical system.

We note that controller 1 presents a larger stability region than controller 2. It can be explained as

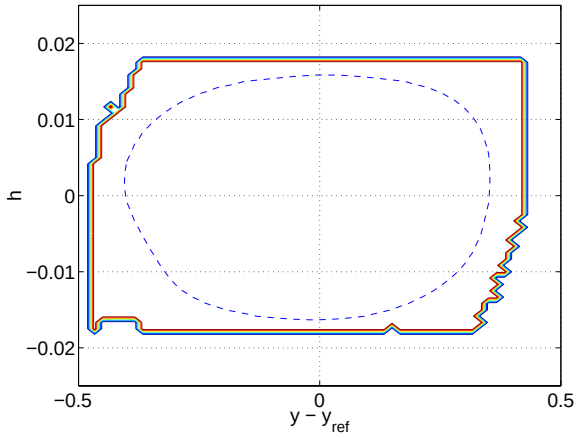

Fig. 7.3: State space cut showing the estimated stable region in the (y, h) plane for $\dot{y} = 0$ and $y_{\text{ref}} = 25\ in$ when using controller 1.

solid:	stability region from model.
dashed:	neural network estimation.

an effect of the delay over the closed-loop behavior. The delay may cause the larger control values to overshoot the system and reducing its applicability on the edges of the stability region.

On the other hand, the larger gains may lead to a faster controller and then, preferable to use, when possible, to close on the setpoint faster.

7.3 Controller scheduling approach

The experiments were set up using a computer controlled system under the SIMPLEX architecture that is running under Lynx, a real-time operating system. The base sample time for control was 100 ms although a faster rate was used (10 times) to obtain more accurate measurements and to compute the velocity of the diver. A set point of $y_{\text{ref}} = 25$ *in.* was selected. The two controllers described in § 7.1 were used to test the switching strategy.

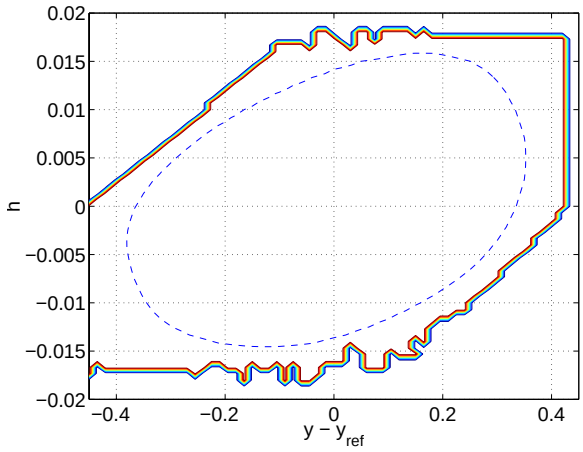

Fig. 7.4: State space cut showing the estimated stable region in the (y, h) plane for $\dot{y} = 0$ and $y_{\text{ref}} = 25\ in$ when using controller 2.

solid:	stability region from model.
dashed:	neural network estimation.

Data was collected from the system to be used in the off-line neural network training procedures. In this example we work inside the regions of stability of both controllers so that it is not necessary to use the stability region estimators. Estimation of the performance indices for $\delta = 0.5$ by the methods explained in Chapter 4 was carried out and used to manage the switching between these two controllers.

The analytical model was used for initial training of the neural network. Experimental data from twenty different runs were used to adapt the parameters of the neural networks to estimate the performance indices of both controllers. Fig. 7.5 shows a typical data profile to estimate the performance index of one of the controllers and Fig. 7.6 shows a slice of the resulting performance estimate.

During the experiments the neural networks were kept fixed. This was due primarily to the fact that the computer setup did not have enough power to do training on-line. It was noted that the SIMPLEX code in itself, which handles the communication between processes and with the physical

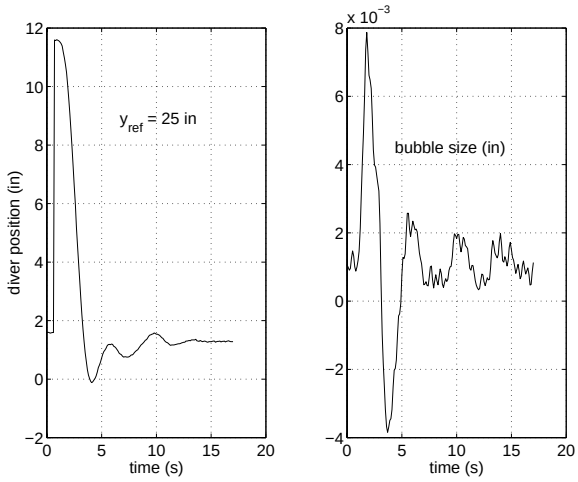

Fig. 7.5: Data obtained from the physical system under controller 1.

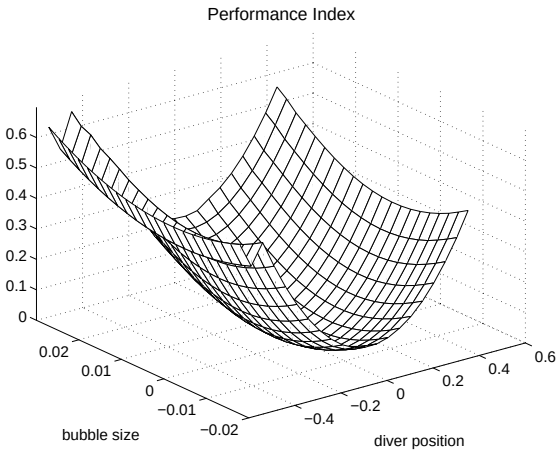

Fig. 7.6: Performance index estimate for controller 1.

system already takes a big part of the cpu time making a little more difficult to do long computations inside one process that does not have high priority among all the processes involve in the SIMPLEX architecture.

Figures 7.7 and 7.8 show a switching experiment for a step change in the setpoint value for y_{st} from 13 to 25 inches. Fig. 7.9 shows the estimated performance indices during the run. From

the figure we observe that controller 2 is preferred for larger values of y. After approximately 5.5 seconds, when the system is getting close to the desired setpoint, $\hat{J}_\delta^1$ becomes smaller than $\hat{J}_\delta^2$ and the scheduler switches to controller 1.

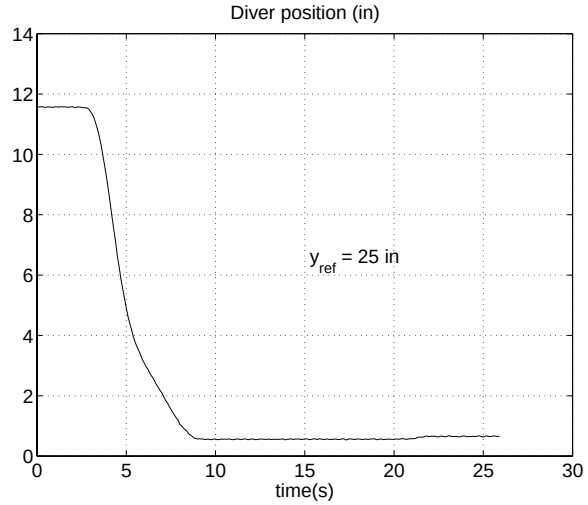

Fig. 7.7: Level position of the vessel during a switching experiment.

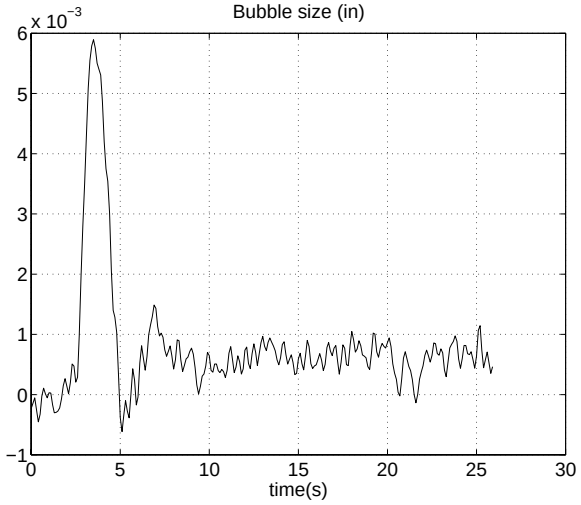

Fig. 7.8: Bubble size during a switching experiment.

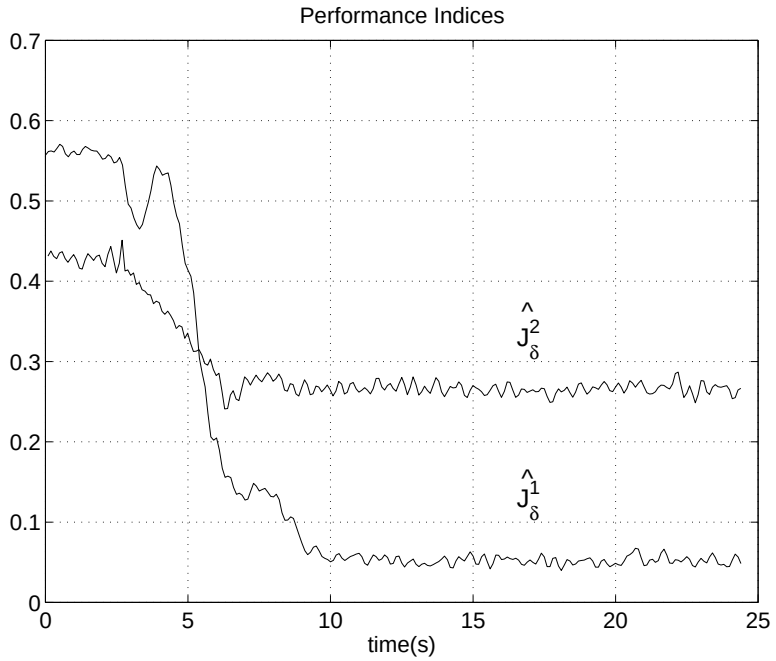

Fig. 7.9: Performance indices estimates for controllers for the submerged vessel during a switching experiment.

The control scheduling experiment gives a very satisfactory result. Based on the estimates for the controller's stability region and cost-to-go performance, the controller scheduler algorithm makes its decision on which one to apply to the system in the next sample. The decision taken by the scheduler in the sample experiment is intuitive. It selects the fastest controller to get closer to the setpoint and then switches to a slower but more effective controller around the setpoint.

Chapter 8

Conclusions

8.1 Summary of contributions

In summary, the principal contributions of this thesis are:

- Development of a generic non-model-based method to find regions of stability for dynamical systems using neural networks plus specific decision rules to make a conservative estimation.
- A general approach to estimate performance indices for dynamical systems. Neural networks are used as estimators with neurodynamic programming training techniques.
- A new application of neural networks for controller scheduling with guaranteed stability and performance. It combines the good properties of neural networks with Lyapunov analysis to generate a suitable framework for controller scheduling of complex nonlinear dynamical systems.
- The first implementation of an autonomous evolvable system using the SIMPLEX architecture that learns the switching rules. This was a strong motivation for this work and a reasonable scenario for a neural network type of architecture.

It should be noticed that the SIMPLEX architecture also addresses other issues that are not being taken into consideration here. The controller scheduling algorithm described in Chapter 5 provides a method to select the controllers on a stability and performance basis which the current version of SIMPLEX lacks. Moreover, the stability region estimation method developed in Chapter 3 and used in the controller scheduling algorithm can be applied by itself to prevent any break downs by computing the switching surface to go back to the *safety* controller. However, the SIMPLEX architecture also considers taking measures to guarantee software and communications failure recovery.

8.2 Recommendations for future work

The suggestions on future research lines is divided in sections directly related to each major topics addressed in this thesis.

8.2.1 On the stability region analysis

Many lines of research are still open for the estimation of the region of attraction of a nonlinear system. In many cases the key issue in finding the right region of attraction lies in the constraints impose on the states or control inputs of the dynamical system being studied. A method like the one described in § 3.3.1 puts all the burden in the designer to find the regions A,B and C. Moreover, since the final classification only indicates whether the system is inside or outside the region of attraction, any modification in the constraints, would involve retraining the network again and again. Therefore, it would make sense to try to estimate the decision variable or variables that set up the regions A, B and C.

In the case in which the constraints on the states are the ones which determine the region of attraction, one possible option is to try to estimate how far away the system would get from the equilibrium state and use that information for the decision. This could be done with the function

$d : \Re^n \rightarrow \Re^n$ such that

$$d(\mathbf{x}_o) = \sup_{k \in \mathcal{Z}^+} |\mathbf{x}_{ik} - \mathbf{x}_{i_{eq}}|, \quad i = 1, \ldots, n \tag{8.1}$$

where $\mathbf{x}_{ik}$ represents the i^{th} component of the state vector at time step k and $\mathbf{x}_{i_{eq}}$ stands for the i^{th} component of the equilibrium state. This definition is easily generalizable to use measures other than the absolute value. For example, any other function of the state vector that might be desirable could be used. This type of information would be better to handle trajectory tracking problems.

Another possible approach to estimate stability regions is to use recurrent neural networks. A recurrent neural network architecture is a dynamic system in itself and not just a static mapping, suggesting more basic questions of approximating a region of attraction of one nonlinear system by another nonlinear dynamic system and how close and conservative the approximation can be. These two dynamic systems may not be similar in the overall behavior but should have similar stability regions.

8.2.2 On the performance estimates

There are several ways to improve the estimation procedures. Presently, we can handle theoretical results in the case in which we can express the estimate as a linear combination of basis functions. Bertsekas and Tsitsiklis shown that convergence is not guaranteed with nonlinear estimates, but there are no conclusive results concerning neural network estimators yet [BT96]. Obviously, as long as we only adapt the output layer, the convergence results we obtained can be applied to neural networks. A direction to go with linear estimators would be the use of wavelets as a way to optimize the number and shape of the necessary basis functions.

From the comparisons between the different estimation procedures in Chapter 4 one can think of combining the higher-order method with the embedded filtering method to gain in accuracy and also regularization. Research efforts should be directed in the direction of improving the embedded filtering algorithm to make possible its combination with the higher-order method.

As mentioned in § 8.2.1, the use of recurrent neural networks should also be analyzed for this type of estimators. Recurrent networks have a potential to store values as attractors to be exploited in solution of the recursive equation.

8.2.3 On the controller scheduler strategy

One limitation of the proposed algorithms is that the on-line training techniques can be applied only to the actual closed-loop dynamics, i.e. at any point in time we are updating one and only one stability region and performance index estimate. A way to get around this is by having a model of the system. A model of the system would allow us to predict the future values for states and generate the training patterns for the neural networks associated to the controllers not in use at that moment.

Another requirement for the strategy to work is that the controllers must stabilize the plant. The example in Chapter 6 shows that the requirement could be relaxed if there is way to estimate if there exists a region in which the application of one controller may lead the system to the stability region of another controller.

The controller scheduling approach designed in Chapter 5 assumes the controllers are given. A further step in the process would involve the introduction of a controller design based on the information collected by the estimators. One possible way of doing this would be to use a Q-learning approach, i.e. to approximate a Q-function from the performance estimates gathered from each controller, setting up a new controller that would be the best combination of the controllers already specified for the system. That approach would ultimately generate an optimal controller.

In this thesis we considered the full information case, i.e. we assume it is possible to measure all state variables. The inclusion of an observer into the algorithm is the next step to take. It is necessary to investigate how this extra dynamics affects the estimation and scheduling process. The control scheduling problem solved in Chapter 5 involves only the stabilization of a plant around an

equilibrium point. The next step is to extend this approach to consider also a tracking problem.

8.2.4 Algorithm improvements

It is a very important to study and improve the numerical algorithms for this approach to be considered for a practical application. The main problem appears when on-line training is needed. In this case, the computation increases by a factor proportional to the number of parameters of the network with respect to the off-line learning together with on-line evaluation only. Algorithms for training are already well optimized, so it does not seem likely we can reduce the order of the computations in this case. One way to decreased the necessary computations to estimate those functions on-line, would be to optimize the parameters of the networks off-line as much as possible.

As usual, computational speed can be improved by orders of magnitudes with the advances in microprocessor speed or the use of neural network chips instead of neural network software.

8.2.5 Application environments

Time constraints allowed the application of this algorithm only to a couple of example problems that can be considered low-order, slow systems. This approach though has the potential to be used for high-order and faster systems given enough computation power.

Chemical processes may in general be a suitable type of system to use this approach. They are usually high order, highly nonlinear systems, for which an accurate model is very difficult to obtain. Usually, the dynamics of such processes are slow, so that computation time is not the most important issue.

8.2.6 Stochastic approach

The motivation to look for safety issues on the application part was one of the motivations to choose deterministic rather than stochastic models. As mentioned in the background chapter, there are some important results in the stochastic domain. It has an advantage that the uncertainty on the system is introduced in a systematic way into the approach at the expense of the more involved mathematics. It would be valuable to translate some of our approaches to the stochastic framework to combine the results.

Appendices

Appendix A

Theorem proofs

Theorem 4.5

Proof. We define a Lyapunov function for the neural parameters as,

$$V(\tilde{W}) = \|\tilde{W}\|^2 = \tilde{W}^T\,\tilde{W}, \quad \tilde{W} = W - \hat{W} \tag{A.1}$$

Using equations (4.7), (4.10) and (4.11) and rearranging terms the adaptive algorithm for the weights $\hat{W}$ is

$$\begin{aligned}
\hat{W}_{k+1} &= \hat{W}_k - \eta(U(\mathbf{x}_k) + \hat{W}_k^T(\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k))) \cdot (\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)) \\
\tilde{W}_{k+1} &= \tilde{W}_k + \eta(U(\mathbf{x}_k) + \hat{W}_k^T(\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k))) \cdot (\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)) \\
&= \tilde{W}_k - \eta\tilde{W}_k^T(\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)) \cdot (\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)) \\
&= \tilde{W}_k + \Sigma_k
\end{aligned}$$

Computing the difference of the Lyapunov function (A.1)

$$\begin{aligned}
\Delta V_k &= \tilde{W}_{k+1}^T\,\tilde{W}_{k+1} - \tilde{W}_k^T\,\tilde{W}_k \\
&= 2\tilde{W}_k^T\Sigma_k + \Sigma_k^T\,\Sigma_k
\end{aligned}$$

Substituting for Σ_k and collecting terms we ended up with

$$\begin{aligned}\Delta V_k &= \eta\,(\tilde{W}_k^T(\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)))^2\,(\eta\,\|\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)\|^2 - 2)\\ &\leq \eta\,(1+\delta)^2\,\|\tilde{W}\|^2\,\Psi_M^2\,(\eta\,(1+\delta^2)\Psi_M^2 - 2)\end{aligned}$$

Using condition (4.12) we conclude that

$$\Delta V_k \begin{cases} = 0 & \text{if } \tilde{W}_k^T(\delta\Psi(\mathbf{x}_{k+1}) - \Psi(\mathbf{x}_k)) = 0 \\ < 0 & \text{otherwise} \end{cases}$$

□

Theorem 4.6

Proof. To simplify notation, the superscript i will be dropped in the understanding that we are dealing with a closed loop system for a given controller. Using the definitions for λ_k, μ_k and φ_k from equations (4.17) and (4.18) the error term to adapt the network parameters is given by

$$\begin{aligned}\varepsilon_k &= \hat{J}_\delta(\mathbf{x}_{k+1}) - \varphi_{k+1}\\ &= \hat{J}_\delta(\mathbf{x}_{k+1}) - \lambda_k\hat{J}_\delta(\mathbf{x}_k) - \mu_k\varphi_k + \gamma U(\mathbf{x}_k)\\ \varphi_{k+1} &= \lambda_k\hat{J}_\delta(\mathbf{x}_k) + \mu_k\varphi_k - \gamma U(\mathbf{x}_k)\end{aligned}$$

with $\gamma = 1/\delta$. Substituting $\hat{J}_\delta(\mathbf{x}_k)$ by its linear estimates (4.11)

$$\begin{aligned}\varepsilon_k &= \hat{W}_k^T(\Psi(\mathbf{x}_{k+1}) - \lambda_k\Psi(\mathbf{x}_k)) - \mu_k\varphi_k + \gamma U(\mathbf{x}_k)\\ &= \hat{W}_k^T\Theta_\lambda - \mu_k\varphi_k + \gamma U(\mathbf{x}_k) \qquad \text{(A.2)}\end{aligned}$$

To apply (4.9) with the new error ε_k defined we need to compute $\nabla_{\hat{W}}\,\varepsilon_k$

$$\nabla_{\hat{W}}\,\varepsilon_k = \Theta_\lambda - \mu_k\nabla_{\hat{W}}\,\varphi_k$$

If we denote $\Phi_k = \nabla_{\hat{W}} \varphi_k$ then

$$\nabla_{\hat{W}} \varepsilon_k = \Theta_\lambda - \mu_k \Phi_k \tag{A.3}$$

$$\Phi_{k+1} = \mu_k \Phi_k + \lambda_k \Psi(\mathbf{x}_k) \tag{A.4}$$

Combining (4.9), (A.2), (A.3) and (A.4) we obtain equations (4.19).

To analyze the convergence of the algorithm we introduce the true weight vector

$$J_\delta(\mathbf{x}) = W^T \Psi(\mathbf{x}) \tag{4.10}$$

and the new training state variables

$$\tilde{W}_k = W - \hat{W}_k \tag{A.5}$$

$$\tilde{\varphi}_k = W^T \Psi(\mathbf{x}_k) - \varphi_k \tag{A.6}$$

$$\tilde{\Phi}_k = \Psi(\mathbf{x}_k) - \Phi_k \tag{A.7}$$

into the training equations (4.19) leading to

$$\begin{aligned}
\tilde{W}_{k+1} &= (I - \eta_k(\Theta_\gamma - \mu_k \tilde{\Phi}_k)\Theta_\lambda^T)\tilde{W}_k + \eta_k \mu_k (\Theta_\gamma - \mu_k \tilde{\Phi}_k)\tilde{\varphi}_k \\
\tilde{\varphi}_{k+1} &= \lambda_k \tilde{W}_k^T \Psi(\mathbf{x}_k) + \mu_k \tilde{\varphi}_k \\
\tilde{\Phi}_{k+1} &= \mu_k \tilde{\Phi}_k - \Theta_\gamma
\end{aligned}$$

with $\Theta_\gamma = \Psi(\mathbf{x}_{k+1}) - \gamma \Psi(\mathbf{x})$. In matrix notation it can be expressed as

$$Z_{k+1} = A_k(Z_k)\, Z_k + B_k \tag{A.8}$$

$$Z_k^T = [\tilde{W}_k^T \;\; \tilde{\varphi}_k \;\; \tilde{\Phi}_k^T] \tag{A.9}$$

$$A_k = \begin{bmatrix} I - \eta_k(\Theta_\gamma - \mu_k \tilde{\Phi}_k)\Theta_\lambda^T & \eta_k \mu_k (\Theta_\gamma - \mu_k \tilde{\Phi}_k) & 0 \\ \lambda_k \Psi(\mathbf{x}_k)^T & \mu_k & 0 \\ 0 & 0 & \mu_k I \end{bmatrix}$$

$$B_k^T = [0 \quad 0 \quad -\Theta_\gamma^T]$$

If we choose $\mu_k < 1$ the sequence $\tilde{\Phi}_k$ is bounded. A sufficient condition to have convergence for $\tilde{W}_k$ and $\tilde{\varphi}_k$ is to impose A_k to be contractive, i.e. $\|A_k\| < 1$. However, because of the structure of A_k, and in particular the upper-left A_{11} submatrix, it is not possible to make A_k contractive independently of the trajectory $\mathbf{x}_k$ being followed. In fact, it is easy to see that A_k has always a singular value greater or equal to 1. If we choose a vector $\mathbf{v}$ orthogonal to Θ_λ then

$$A_k \begin{bmatrix} \mathbf{v} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \lambda_k \Psi(\mathbf{x}_k)^T \mathbf{v} \\ 0 \end{bmatrix}$$

Therefore, at each time step k the values of η_k, μ_k and λ_k have to be optimized to give a contractive mapping in the direction of the trajectory of the state vector Z_k.

□

Theorem 4.8

Proof. From (4.10) and (4.11) and the definition of $\Gamma^i_{\delta r,k}$ we get,

$$\begin{aligned} \Gamma^i_{\delta r,k} &= \Lambda^T(\mathbf{x}^i_k) W_k \\ \hat{\Gamma}^i_{\delta r,k} &= \Lambda^T(\mathbf{x}^i_k) \hat{W}_k \end{aligned}$$

with $\Lambda^T_k = \Lambda^T(\mathbf{x}^i_k) = [\ \Psi(\mathbf{x}^i_k)\ \Psi(\mathbf{x}^i_{k+1})\ \ldots\ \Psi(\mathbf{x}^i_{k+r-1})\ ]^T$. Using the expressions for the higher order HDP given by (4.26)-(4.28) and following the same line of reasoning as in Theorem 4.5 it is straight forward to find that the expression for the error term ϵ_k is

$$\epsilon_k = (\Lambda^T_k - A_{\delta r} \Lambda_{k+1}) \tilde{W}_k \tag{A.10}$$

where $\tilde{W}_k = W - \hat{W}_k$ is the error in the weight parameters. The rest of the proof is analogous to the proof in Theorem 4.5 noting that

$$\begin{aligned} \|\Lambda_k\| &\leq r\Psi_M \\ \|A_{\delta r}\Lambda_{k+1}^T\| &\leq (r-1)\Psi_M + \left(\sum_{i=1}^{r} \alpha_i^2 \delta^{2i}\right)^{1/2} \Psi_M \end{aligned}$$

□

Theorem 5.3

Proof. For a given $\mathbf{x}_o \in \hat{R}$, let i_k be the sequence of controllers selected by the min-switching rule. If there exists some K and $l \in \{1, \ldots, M\}$ such that $i_k = l$ for all $k > K$, the theorem is true since the origin is a stable equilibrium for controller l. On the other hand, if the controller switches infinitely often, there must be one controller l which is selected infinitely often. Let the sequence of time indices $0 < k_1 < k_2 \ldots$ be an infinite sequence of sampling instants when controller l is selected with $\mathbf{x}_k \in \bigcap \hat{R}^i$, $\forall\, k > k_1$. The min-switching rule implies

$$\hat{J}_\delta^l(\mathbf{x}_{k_{j+1}}) < \hat{J}_\delta^l(\mathbf{x}_{k_j}), \quad \forall j$$

because of the limits L_{k_j}. Since the $\hat{J}_\delta^l(\mathbf{x}_{k_j})$ are bounded from below, the sequence $\hat{J}_\delta^l(\mathbf{x}_{k_j})$ converges to some constant C.

For each $i = 1, \ldots, M$ let $\eta^i(\mathbf{x})$ denote the error in the i^{th} performance estimate at state $\mathbf{x}$ where it is assumed $|\eta^i(\mathbf{x})| \leq \epsilon$ for some $\epsilon > 0$. This implies that when the i^{th} controller is applied at a state $\mathbf{x} \in \hat{R}^i$,

$$\begin{aligned} \Delta\hat{J}_\delta^i(\mathbf{x}) &\triangleq \hat{J}_\delta^i(F^i(\mathbf{x})) - \hat{J}_\delta^i(\mathbf{x}) \\ &= \Delta J_\delta^i(\mathbf{x}) + \eta^i(F^i(\mathbf{x})) - \eta^i(\mathbf{x}) \\ &\leq \Delta J_\delta^i(\mathbf{x}) + 2\epsilon. \end{aligned}$$

Since J_δ^i is a Lyapunov function for the system under controller i, $\Delta J_\delta^i(\mathbf{x}) \leq 0$. Returning to the specific controller l, suppose that there are an infinite number of the x_{k_j} that remain a finite distance from the set

$$\chi_\epsilon^l = \{\mathbf{x} \in R^l \mid -\Delta J_\delta^l(\mathbf{x}) \leq 2\epsilon\}.$$

This would imply the sequence $\Delta \hat{J}_\delta^l(\mathbf{x}_{k_j})$ is negative and bounded away from zero infinitely often, contradicting $\hat{J}_\delta^l(\mathbf{x}_{k_j}) \to C$. Therefore, $\mathbf{x}_{k_j} \to X_\epsilon^l$. More precisely, given any $\tilde{\epsilon} > 0$, there exists some $K_{\tilde{\epsilon}}^l$ such that the distance $d(\mathbf{x}_{k_j}, \chi_\epsilon^l) < \tilde{\epsilon}$ for all $k_j > K_{\tilde{\epsilon}}^l$. This is illustrated in Fig. 5.2.

While controller l is applied, $J_\delta^l(\mathbf{x}_k)$ is monotone non-increasing since J_δ^l is a Lyapunov function for the system. Therefore, $J_\delta^l(\mathbf{x}_k) \leq J_\delta^l(\mathbf{x}_{k_j})$, for $k \geq k_j$ until another controller becomes active (see Fig. A.1). Define $I_l = \{k \in \mathcal{N} | i_k = l\}$ and $\bar{J}_\delta^l = \sup_{\tilde{\mathbf{x}} \in \chi_\epsilon^l} J_\delta^l(\tilde{\mathbf{x}})$. Then, for any given $\beta > 0$ we

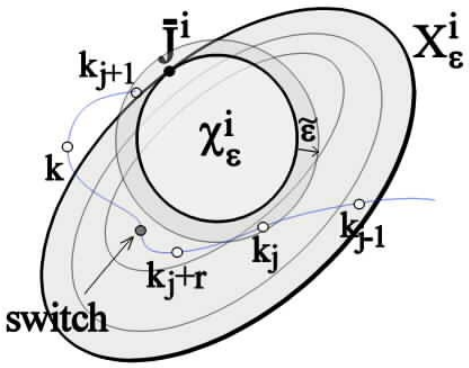

Fig. A.1: Trajectory of the system illustrating convergence to X_ϵ.

have

$$J_\delta^l(\mathbf{x}_k) \leq \sup_{k_j} J_\delta^l(\mathbf{x}_{k_j}) \leq \bar{J}_\delta^l + \beta \qquad \forall k \in I_l, k > K_{\tilde{\epsilon}}^l$$

because of the continuity assumption on $J_\delta^l(\mathbf{x})$. Since this has to be true for any l, after a finite K in

which all the controllers that are not used infinite number of times do not become active any more, the sequence $\mathbf{x}_k$ is arbitrarily close to X_ϵ.

□

References

[Aga92] Ravi P. Agarwal. *Difference equations and inequalities : theory, methods, and applications*. M. Dekker, New York, 1992.

[Bar74] Y. Bard. *Nonlinear parameter estimation*. Academic Press, New York - London, 1974.

[BGFB94] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalitites in System and Control Theory*. SIAM, Philadelphia, 1994.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, Great Britain, 1995.

[BN95a] J. Balakrishnan and K.S. Narendra. Adaptation and learning using multiple models, switching, and tuning. *IEEE Control Systems Magazine*, 15(3):37–51, Jun 1995.

[BN95b] J. Balakrishnan and K.S. Narendra. Intelligent control using fixed and adaptive models. In *Proceedings of 1995 American Control Conference*, volume 1, pages 597–601, Evanston, IL, USA, Jun 1995.

[Bra94a] Michael S. Branicky. Analysis of continuous switching systems: Theory and examples. In *Proc. American Control Conference*, volume 3, pages 3110–4, Baltimore, MD, Jun 1994.

[Bra94b] Michael S. Branicky. Stability of switched and hybrid systems. In *Proc. 33rd IEEE Conf. Decision Control*, volume 4, pages 3498–3503, Lake Buena Vista, FL, Dec 1994.

[BT96] Dimitri P. Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[BYB94] S.J. Bradtke, B.E. Ydstie, and A.G. Barto. Adaptive linear quadratic control using policy iteration. In *Proc. American Control Conference*, volume 3, pages 3475–9, Baltimore, MD, Jun 1994.

[CHW88] H-D. Chiang, M.W. Hirsch, and F. Wu. Stability regions of nonlinear autonomous dynamical systems. *IEEE Trans. on Automatic Control*, 33(1):16–27, Jan 1988.

[Cla83] F. Clarke. *Optimization and Nonsmooth Analysis*. Wiley, New York, 1983.

[CLR96] G Chryssolouris, M. Lee, and A. Ramsey. Confidence interval prediction for neural network models. *IEEE Trans. on Neural Networks*, 7(1):229–32, Jan 1996.

[CO96] P.E. Caines and R. Ortega. The semilattice of piecewise constant controls for nonlinear systems: a possible foundation for fuzzy control. In *Postprint volume from IFAC Nonlinear Control System Design Symposium*, volume 2, pages 843–8, Tahoe City, CA, Jun 1996. Pergamon.

[Day92] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362, 1992.

[DK71] E.J. Davison and E.M. Kurak. A computational method for determining quadratic lyapunov functions for nonlinear systems. *Automatica*, 7:627–636, 1971.

[HKP91] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*, volume I. Addison-Wesley, 1991. Lectures Notes.

[HS93] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: optical brain surgeon. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Infor-*

mation Processing Systems, volume 5, pages 164–71. Morgan Kaufmann, san Mateo, CA, 1993.

[JR96] M. Johansson and A. Rantzer. Computation of piecewise quadratic lyapunov functions for hybrid systems. Isrn lutfd2/tfrt–7549-se, Lund Institute of Technology, Lund, Sweden, Jun 1996.

[KB60] R.E. Kalman and J.E. Bertram. Control system analysis and design via the second method of lyapunov I: Continuous-time systems. *Trans. ASME Journal of Basic Engineering*, pages 371–393, Jun 1960.

[KG97] I. Kolmanovsky and E.G. Gilbert. Multimode regulators for systems with states and control constraints and input disturbances. In A. Stephen Morse, editor, *Control using logic-based switching*, Lecture Notes in Control and Information Sciences 222, pages 104–17. Springer-Verlag, New York, NY, 1997.

[Koh89] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, 3rd. ed. edition, 1989.

[Lev44] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quaterly Journal of Applied Mathematics II*, 2:164–68, Feb 1944.

[Lip95] S.B. Lippman. *C++ Primer*. Addison Wesley, 2nd edition, Feb 1995.

[LM94] Derong Liu and Anthony N. Michel. *Dynamical Systems with saturation Nonlinearities*. Lectures Notes in Control and Information Sciences. Springer-Verlag, 1994.

[LN96] A. Levin and K.S. Narendra. Control of nonlinear dynamical systems using neural networks – part ii: Observability, identification and control. *IEEE Trans. on Neural Networks*, 7(1):30–42, Jan 1996.

[MADF97] M.W. McConley, B.D. Appleby, M.A. Dahleh, and E. Feron. A control lyapunov function approach to robust stabilization of nonlinear systems. In *Proc. American Control Conference*, Alburquerque, NM, Jun 1997. AACC.

[Mar63] D.W. Marquardt. An algorithm for least squares estimation of nonlinear parameters. *SIAM Journal*, 11(2):431–41, Feb 1963.

[Mar91] D.R. Marpaka. Recognition of stability domains of nonlinear dynamical systems using multilayer feed-forward artificial neural networks. In *XXIII Southeastern Symp. on Syst. Theory*, pages 212–217, Columbia, SC, USA, Mar 1991. IEEE.

[MBA96] J. Malmborg, B. Berhardsson, and K.J. Aström. A stabilizing switching scheme for multi-controller systems. In *Proc. of the IFAC World Congress*, volume F, pages 229–234, San Francisco, California, USA, 1996. IFAC'96, Elsevier Science.

[Min86] Michel Minoux. *Mathematical Programming. Theory and Algorithms*. John Wiley & Sons Ltd., 1986.

[Mor95] A.S. Morse. Supervisory control of families of linear set-point controllers - Part II: Robustness. In *34th IEEE Conference on Decision and Control*, volume 2, pages 1750–5, New York, USA, Dec 1995.

[Mor96] A.S. Morse. Supervisory control of families of linear set-point controllers - Part I: Exact matching. *IEEE Trans. on Automatic Control*, 41(10):1413–31, Oct 1996.

[MZT96] P.M. Mills, A.Y. Zomaya, and M.O. Tade. *Neuro-Adaptive Process Control*. John Wiley & Sons Ltd., 1996.

[NB97] K.S. Narendra and J. Balakrishnan. Adaptive control using multiple models. *IEEE Trans. on Automatic Control*, 42(2):171–187, Feb 1997.

[NL95] E. Noldus and M. Loccufier. A new trajectory reversing method for the estimation of asymptotic stability regions. *Int. J. Control*, 61(4):917–932, 1995.

[NP90] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, pages 4–27, 1990.

[PL96] S. Pettersson and B. Lennartson. Stability and robustness for hybrid systems. In *35th IEEE Conference on Decision and Control*, volume 2, pages 1202–7, Dec 1996.

[Sha95] L. Sha. A software architecture for dependable and evolvable industrial computing systems. In *Proc. IPC'95*, pages 145–156, Detroit, MI, May 1995.

[SL91] Jean-Jacques. E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall Inc., 1991.

[SS92] R.M. Sanner and J-J. E. Slotine. Gaussian networks for direct adaptive control. *IEEE Trans. on Neural Networks*, 3(6):837–863, 1992.

[Sut88] R.S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Ter96] Tamás Terlaky. *Interior point methods in mathematical programming*. Kluwer, 1996.

[TR97] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Trans. on Automatic Control*, 42(5):674–690, May 1997.

[Tre68] H.L. Van Trees. *Detection, Estimation and Modulation Theory*. John Wiley and Sons, New York, 1968. Part I.

[VV85] A. Vannelli and M. Vidyasagar. Maximal lyapunov functions and domains of attraction for autonomous nonlinear systems. *Automatica*, 21:69–80, Jan 1985.

[Wat89] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[Wer91] P. Werbos. A menu of designs in reinforcement learning over time. In W.T. Miller III, R.S. Sutton, and P. Werbos, editors, *Neural Networks for control*, chapter 3. MIT Press, 2nd edition, 1991.

[ZHZL88] J. Zaborsky, G. Huang, B. Zheng, and T-C. Leung. On the phase portrait of a class of large nonlinear dynamic systems such as the power system. *IEEE Trans. on Automatic Control*, 33(1):4–15, Jan 1988.

[Zub64] V.I. Zubov. *Methods of A. M. Lyapunov and their application.* P.Noordhoff Ltd., 1964.