ADAPTIVE NEUROCONTROL AND ITS APPLICATION TO ROBOTS

by

FENG LIANG, B. Eng., M. Eng.

A Thesis

Submitted to the School of Graduate Studies

In Partial Fulfillment of the Requirements

For the Degree

Doctor of Philosophy

McMaster University

May 1995

ADAPTIVE NEUROCONTROL AND ITS APPLICATION TO ROBOTS

Dedicated To

My wife and son

DOCTOR OF PHILOSOPHY (1995)
(Mechanical Engineering)

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE:	Adaptive Neurocontrol and Its Application to Robots
AUTHOR:	Feng Liang B. Eng. (Beijing University of Aeronautics and Astronautics) M. Eng. (Beijing University of Aeronautics and Astronautics)
SUPERVISOR:	Dr. Hoda A. ElMaraghy Professor and Director of Flexible Manufacturing Centre (till June 30, 1994) Dean of Engineering, The University of Windsor (since July 1, 1994)

NUMBER OF PAGES: xiv, 234

ABSTRACT

This thesis is devoted to investigating adaptive neurocontrol of nonlinear systems with uncertain or unknown dynamic models. Novel theoretical synthesis and analysis of neurocontrol systems have been conducted, and applied to the control of flexible joint robots with experimental tests. The contributions of this thesis fall into the following three areas: (1) neural networks, (2) adaptive neurocontrol and (3) control of flexible joint robots.

The aim of my research in the neural network area is to search for fast and global convergent learning algorithms with reduced computation burden. The localized neural networks with competitive lateral inhibitory cells were introduced. The developed extended Kalman filtering algorithm with UD factorization can make the localized polynomial networks and localized pi–sigma networks possess fast learning convergence and less computation. The multi–step localized adaptive learning algorithm was derived for RBF networks which leads to about 10 fold improvement in the speed of learning convergence. New neural network models of nonlinear systems were introduced to facilitate neurocontroller design.

In the adaptive neurocontrol area, theoretical issues of the existing backprop–based adaptive neurocontrol schemes were first clarified. Then new direct and indirect adaptive neurocontrol schemes, with better performance, were proposed. It is noticed that the system stability of many existing neurocontrol schemes cannot be proved. In addition, few stability–based adaptive neurocontrol schemes are available and can only be applied to feedback linearizable nonlinear systems. The thesis provides two major contributions to the stability–based adaptive neurocontrol approach. The first contribution is extending the classical self–tuning control methodologies for linear systems to the self–tuning neurocontrol of nonlinear systems by using localized neural networks. This extension greatly enriches the neurocontrol algorithms with guaranteed system stability. The second contribution is proposing the variable index control approach, which is of great significance in the control field, and applying it to derive new stable robust adaptive neurocontrol schemes. Those new schemes possess inherent robustness to system model uncertainty, which is not required to satisfy any matching condition. They do not impose any growth condition and infinite differentiability assumption on the system nonlinearity. They can also be applied to nonlinear systems which are not feedback–linearizable.

As applications and extensions of the above theory, three different robust adaptive neurocontrol schemes for general flexible joint robots were derived with proven system stability. All three schemes are able to incorporate a priori information about the robot dynamics into the neurocontroller design to simplify the neural network design. No acceleration and jerk signals are required in these control laws. Moreover, arbitrary joint stiffness is allowed in the control algorithms.

To demonstrate the feasibility of the proposed learning algorithms and adaptive neurocontrol schemes, intensive computer simulations were conducted based on different nonlinear systems and functions. Different types of adaptive tracking problems and regulation problems were considered. Furthermore, the proposed adaptive neurocontrol schemes were experimentally tested using an existing experimental flexible joint robot. Both the simulation and experimental results confirm the practicability of the proposed schemes.

The thesis concludes that the neurocontrol approach, along with the development of neural computers and large scale parallel distributed processors, is capable of solving the complex control problem of nonlinear systems with uncertain or unknown dynamic models.

ACKNOWLEDGEMENTS

The author would like to express his deepest appreciation for the valuable support provided by his supervisor, Dr. H.A. ElMaraghy.

Special thanks are extended to systems analyst Todd Pfaff for his programming assistance. Many thanks are due to all the supervisory committee members. Many thanks are also due to friends and colleagues for their advice and encouragement.

The financial support from Dr. H.A. ElMaraghy's research funds, including IRIS and NSERC, from McMaster University, in the form of Cliffton Sherman Graduate Scholarship (1991–1993), and from Ontario Ministry of Education, in the form of Ontario Graduate Scholarship (1993–1994 and 1994–1995), are greatly appreciated.

This work is dedicated to my wife, whose confidence and full support made it all possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This chapter gives a brief introduction to neural networks and neurocontrol, and forms the motivation of the thesis. Section 1.1 presents a brief overview of neural networks and their applications. Section 1.2 presents the definition of neurocontrol and its applications to nonlinear systems, including robots. Section 1.3 presents the motivation of the thesis and tries to answer these questions: why neurocontrol? Why not others? Section 1.4 overviews the organization of the thesis.

1.1 NEURAL NETWORKS

Artificial neural networks (or, briefly, neural networks) are computational systems, either hardware or software, that mimic the computational abilities of biological neural network systems by using large numbers of simple, interconnected artificial neurons (fig. 1.1). Artificial neurons are simple emulations of biological neurons; they take in information from sensor(s) or other artificial neurons, perform very simple operations on these data, and pass the results on to other artificial neurons by synapses. Neural networks operate by having their massive, interconnected artificial neurons process data in this manner. They use both logical parallelism and serial operations.

Artificial neural network research was one of the early emerging branches in artificial intelligence in the 1950s. It has been developed into a multi–disciplinary science, involving neuroscience, biology, psychology, computer science, and engineering. The three main characteristics which describe a neural network, and which contribute to its functional abilities, are: structure, dynamics, and learning. In order to show the difference between biological and artificial neural networks, we use the terms units or nodes instead of neurons and connection weights instead of synapses in some contexts.

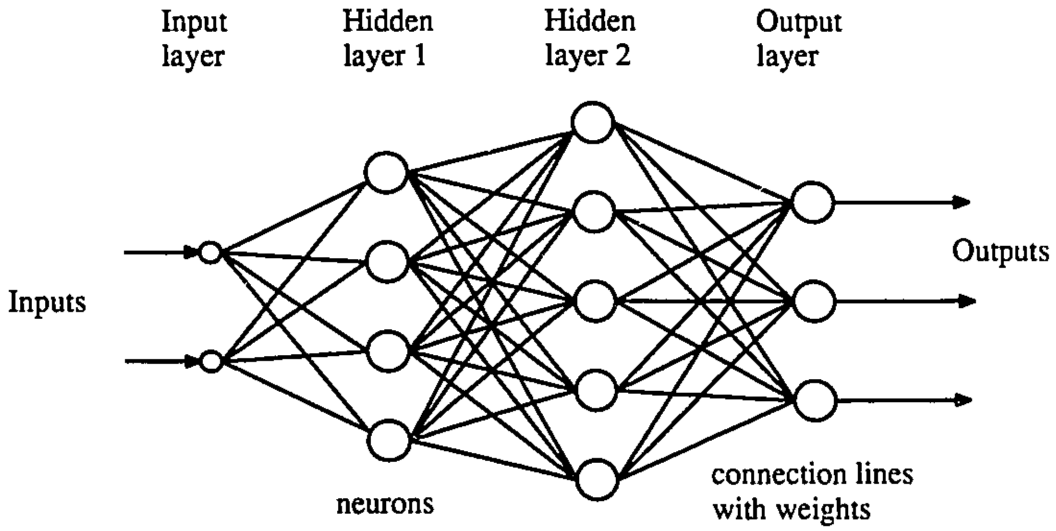


Fig. 1.1 An illustration of neural networks

Neural network research is often thought to be a recent development. However, this field was established before the advent of computers. Work in the field began in the 1940s (Hebb, 1949). Rosenblatt (1958) stirred considerable interest and activity in the field when he designed and developed the *Perceptron,* which was computationally precise and a true learning machine. The ADLINE (ADaptive LINear Element) network was developed shortly after the Perceptron, by Widrow and Hoff (1960). The Least–Mean–Squared learning rule, also known as the Delta rule, was introduced then. The research progressed for about ten years afterwards and declined in the late 1960s, due to a mistakenly generalized conclusion about multilayered systems. The research re–surged around 1986, attributed to the discovery of the well–known *Backpropagation* algorithm for multilayer perceptrons, which was independently developed by Werbos (1974), Parker (1985), LeCun(1986) and Rumelhart and McClelland (1986). Now, neural network research has become one active area of artificial intelligence (AI) and the results have been applied to many science and engineering disciplines (Haykin, 1994).

The distinguishing features of neural networks are: 1) content–addressed memory: they provide fast information retrieval / matching, and the storage of the information is distributed; 2) learning ability: they can learn to do tasks based on training data or initial experience, and can adapt to environmental changes; 3) parallel processing: they can perform fast information processing; 4) distributed processing: they are fault–tolerant with graceful degradation; 5) arbitrary function approximators: they can be applied to modeling and classification problems.

There are many different types of neural networks. The intensively studied types are:

- Feedforward multilayer perceptrons
- Associative memory networks (including auto–associative, hetero–associative)
- High order feedforward multilayer perceptrons (Sigma–Pi, Pi–Sigma and MARS)
- Radial basis function (RBF) networks (Gaussian, and other types)
- CMAC networks (BMAC, PCMAC)
- Recurrent neural networks (arbitrary and specified connections)
- Self–organized neural networks (single– and multi–layer)
- Fuzzy neural networks
- Hybrid and complex networks (hierarchical and hybrid)

Generally speaking, learning is a process of change in a system that enables that system to do similar tasks more efficiently and more effectively the next time. In other words, learning is the generalization of experience. There are two basic activities of learning: knowledge acquisition and skill refinement through practice. Human learning uses both of them. The types of learning used in neural networks are: 1) supervised learning, 2) unsupervised (including competitive learning and self–organized learning) and 3) reinforcement learning.

It is worth mentioning that there are different approaches and methodologies in neural network research, because of the multi–discipline characteristic of this field. There is no doubt that researchers who are interested in neural networks are attracted by their cognitive and computational properties. However, researchers from neuroscience, biology and psychology tend to favor experimental approaches, and draw their conclusions based on observation and generalization. On the other hand, researchers from computer science, physics, and engineering tend to favor logical and mathematical approaches, and specialize the functions of neural networks in their applications. Therefore, some conclusions about neural networks are generalized without theoretical proofs, and some neural networks with solid mathematical basis lack biological support. As research progresses, such problems will surely be solved.

1.2 NEUROCONTROL AND ROBOTIC CONTROL

Neurocontrol can be defined as functional emulation of the learning and adaptation mechanism, and the representation ability of biological neural networks into the design of control systems. The studied systems are mainly nonlinear, though neurocontrol can be applied to linear systems. The corresponding neurocontrollers can be implemented on computers or neural net chips. A neurocontroller and the controlled plant form a neurocontrol system. Neurocontrol is also named in the literature as neural network based control, neuromorphic control or neural control. It can be further classified into weight–fixed neurocontrol, adaptive neurocontrol and robust neurocontrol. A neurocontrol system is said to be adaptive if it can adjust its neurocontroller weights according to the changes in the system dynamics and environment. A neurocontrol system is said to be robust if it can tolerate the changes in the system dynamics and environment, and the neural network modeling errors that are confined by known upper and lower bounds. Due to the distinguishing features of neural networks, the neurocontrol approach is suitable for complex nonlinear systems with uncertain and time–varying factors, and can be further extended to autonomous control with environment recognition, decision making and skill acquisition. They belong to the scope of intelligent control in systems theory, and the related research work has grown rapidly in recent years, both in theory and applications.

Neurocontrol may also seem to be a new area started in recent years. However, its origin dates back to the 1960s and it is among the first few applications of neural networks. Widrow and Smith (1963) proposed a scheme of pattern recognizing control systems using ADLINE, which is the first known neurocontrol scheme. Albus worked a long time to set up a model to mimic the function of cerebella. He succeeded in building a cerebellar model articulation controller (CMAC) and applied it to the robotic control problem (Albus, 1975). Due to the overall decline of neural network research, these successes attracted little attention. After the revival of neural network research and the confirmation that neural networks are universal approximators, neurocontrol research grew rapidly into an active research area in control. A.G. Barto, S. Grossberg, M. Kuperstein, K.S. Narendra, D. Psaltis, R.S. Sutton, P.J. Werbos, and B. Widrow, to name a few, are among the people who started the recent application of neural networks to the control area.

Neurocontrol research belongs to the scope of intelligent control (Bavarian, 1988, Narendra and Mukhopadhyay, 1992). At present, neurocontrol theory is still in its infancy.

The research objective and studied systems are the same as intelligent control. Therefore, an overview of intelligent control is helpful to the understanding of neurocontrol.

Intelligent control systems are control systems that can functionally emulate some of human intelligence into their design and implementation. Intelligent control is mainly aimed at the nonlinear complex dynamic systems which contain uncertainty. By applying intelligent control, these systems are expected to function optimally according to the changes of task, environment and systems themselves without failure. There are numerous intelligent control schemes and different approaches at present. One can categorize intelligent control along many different dimensions. The following are some acceptable categorizations.

(a) categorization according to the type of intelligence:

(1) Self–learning intelligent control

(2) Self–optimizing intelligent control

(3) Self–adaptive intelligent control

(4) Self–reproducing intelligent control

(5) Self–organizing intelligent control

(6) Self–repairing intelligent control

(b) categorization according to the underlying paradigms:

(1) Expert control (1985)

(2) Fuzzy control (1978)

(3) Learning control (1983)

(4) Neurocontrol (1986)

(5) Autonomous control (1980).

(c) categorization according to the application systems:

(1) Intelligent process control systems

(2) Intelligent robotic control systems

(3) Intelligent weapon control systems

(4) Intelligent flight control systems

(5) Intelligent vehicle control systems

(d) categorization according to the structure of intelligent systems:

(1) Centralized; (2) Decentralized; (3) Hierarchical; (4) Multi–level

These categorizations can be combined together to give different names, as in the published papers. For example, self–organizing fuzzy control, or, adaptive neurocontrol. As Shirai and Tsujio (1985) stated, it appears that it is the fate of artificial intelligence that when techniques in a given field become established and are put into practice, they cease to be part of artificial intelligence. This is because the underlying principles of human intelligence have not yet been well understood, and most existing artificially intelligent systems only approximate part or a single aspect of human intelligence by means of conventional logic or mathematics. A similar case happens for intelligent control, including neurocontrol. Therefore, when we assess an AI system, the following questions should be answered: In which aspect(s) does the system emulate human intelligence? Is (are) this aspect(s) essential to the solution of the problem?

Robotics is another fast growing research field. Robots find more and more applications in manufacturing, hazard environments, aerospace, medical instruments and even domestic households. Robotics consists of many research areas. This thesis will deal with one of them, the robotic control area, especially the control of flexible–joint robots. The dynamics of a robotic system is highly nonlinear with uncertainty, and a robot may work under unknown and changing environments and execute different tasks. The control problem of robots is not solved completely. Neurocontrol may provide a new solution to this complex problem, since it is natural to attempt to give a robot a brain–like controller and decision maker. In fact, the human brain–eye–arm control loop is an excellent example to emulate for robot control.

According to the applications, robots can be classified into industrial robots, universal robots, and telerobots. According to their flexibility, robots can be classified into rigid–link rigid–joint robots, rigid–link flexible–joint robots, flexible–link rigid–joint robots, and flexible–link flexible–joint robots. Their degrees of freedom can be redundant or non–redundant. The bases of robots can be fixed or mobile. Their actuation strategies can be indirect or direct drive. The actuators can be electromagnetic, pneumatic, or hydraulic. The common sensors used to measure robotic states are encoders, strain gauges, sonar sensors, laser sensors, or even computer vision systems. Robotic controllers are implemented by either analog circuits or digital computers. The common control tasks are: position and orientation control, trajectory control, force/moment control, constrained motion control, and coordination control of multiple robots.

1.3 MOTIVATIONS FOR ADAPTIVE NEUROCONTROL

1.3.1 Challenges in Robotic Control

A general trend in designing robot manipulators is to make them lightweight relative to their loads, able to work at higher speed with high precision, adaptable to different tasks, cost–effective and efficient in applications. The dynamics of these advanced robot manipulators cannot be simply modeled as rigid bodies. The dynamic modeling errors due to neglecting the structural flexibility of various components in mechanical manipulators have a significant effect on the performance of robot systems, and sometimes even lead to system instability if they are not accounted for in the controller design. It has been observed that joint flexibility, which is more common in industrial robots than link flexibility, plays a significant role in determining the end deflection of robot arms. Joint flexibility results from power transmissions, such as shafts, gear trains, and belts. The actuators themselves also exhibit some electrical, hydraulic, or pneumatic "flexibility". Joint flexibility can be modeled as a lumped torsional spring, although stiffnesses may be nonlinear and difficult to determine exactly. The overall dynamic models of flexible joint robots can be described as a nonlinear matrix differential equation set, and their control problems are more difficult than rigid joint robot ones. Control designers often face a trade–off between the robustness of the control systems and the necessity of measuring joint accelerations and jerks.

A practical dynamic model of an n–link flexible joint robot manipulator usually contains structural and parametric uncertainty. Parametric uncertainty may arise from the irregular geometric shapes of the robotic components, non–uniform materials, non–symmetric motor or transmission installation, part worn-out and end–effector load changes. Structural uncertainty may result from neglected actuator dynamics, internal moving parts, friction and backlash, calibration errors and external disturbances. Sudden control action may excite unmodeled high frequency characteristics, such as link flexibility. Non–symmetric motor axes, for example, result in coupling between link dynamics and motor dynamics. There are different approaches to designing controllers for flexible–joint robots. They can be classified into: (1) exact model–based control, (2) robust control, (3) adaptive control, and (4) fuzzy control. Compared with these approaches, the neurocontrol approach requires the least a priori information about the robot dynamics.

Industrial robots were introduced to increase production flexibility and avoid machine tool and fixture redesign. Intelligent robots play an important role in modern flexible manufacturing systems. In addition to dynamic control, intelligent robots should be able to recognize their environmental and internal changes, make decisions, and perform task–planning, trajectory–planning, contact force–planning and coordination–planning based on external command and sensed information. The information could be qualitative, quantitative, fuzzy and uncertain. Artificial neural networks, along with some adaptive and learning mechanisms, may provide solutions to these problems.

1.3.2 Difficulties in Dealing with Nonlinearity and Uncertainty

With the rapid development of science and technology, and the strong requirement for automation, many large–scale and complex systems have come into use and require automatic control technology. The conventional control theory, which depends completely on the mathematical models of the controlled plants, increasingly exhibits limitations on handling such systems. Most actual dynamic systems are nonlinear, and often time–varying with uncertain factors (parametric, structural, dynamical), which make it very difficult to obtain accurate mathematical models for those systems. Even for nonlinear systems with exact mathematical models, general and effective control design tools are still under development.

In recent years, the adaptive control approach was widely applied to solve the parametric uncertainty problem. It can be applied to linear systems and some linearly parameterizable nonlinear systems. Variable structure control is a popular robust control scheme for nonlinear systems with uncertainty (Utkin, 1976, Slotine and Li, 1991). However, it is difficult to choose a suitable sliding surface for a nonlinear system or a general linear system, especially when the number of control inputs is much less than that of the system states. The feedback linearization approach (e.g., Isidori, 1989) represents a major breakthrough in nonlinear control field. Its adaptive and robust modifications were also proposed. Its main idea is to find a nonlinear diffeomorphism that transforms a nonlinear system into a linear system. Only a small class of nonlinear systems can be transformed into linear systems. For high order nonlinear systems, the outer–loop linear feedback gains, which become the coefficients of characteristic polynomials, are too large to realize if fast convergent error dynamics are required. This is because the canonical form of equivalent linear systems is in cascaded integrator form. Also, the inverse nonlinear transformation may not be global. The requirement that the nonlinearities in system dynamics be infinitely differentiable, or form the so–called "smooth vector field", excludes most of the practical nonlinear systems which contain dead–zone, saturation, hystereses, backlash, etc.

Moreover, the pure mathematical analytical structure of conventional control theory cannot handle qualitative information and make use of human empirical knowledge, skill and heuristic reasoning. Therefore, it is difficult to satisfy the design requirements of complex control systems using such theory (Astrom, 1991). In summary, the control problem of general nonlinear systems with uncertainty is still unsolved.

1.3.3 The Potential of the Neurocontrol Approach

As stated before, neural networks are universal approximators. Therefore, they can be applied to model nonlinearity in dynamic systems. Since they have the ability to learn, neural networks can be trained to emulate human skills, and qualitative control rules, and to adapt to environmental and system internal changes. Therefore, the main advantages of neurocontrol approach are that (a) quantitative system models are not required to design a neurocontroller; (b) it is easy to incorporate any a priori information about controlled systems into neurocontroller design; (c) neurocontrollers can adapt to environmental and system dynamic changes; (d) neurocontrollers can be developed by on–line or off–line learning; and (e) the neurocontrol approach is applicable to both linear and nonlinear systems (which are not necessarily linearly parameterizable).

Neurocontrol has been applied to many control problems at the simulation level. Among many reported applications are rigid and flexible joint robot control, inverse kinematics and dynamics, chemical process modeling and control, vehicle steering control, flight control, underwater vehicle control and nuclear reactor control. Werbos (1989), Narendra and Parthasarathy (1990), Widrow (1990), and Sanner and Slotine (1992) have discussed the basic theoretical issues of neurocontrol and system identification. The above preliminary results show the high potential for applying neural networks to control systems, and provide us with a hope of constructing universal models and controllers for general nonlinear systems. As Narendra and Mukhopadhyay (1992) stated, it has been demonstrated that neural networks are ideally suited to cope with the difficulties in control, i.e., complexity, nonlinearities, and uncertainty.

Compared with conventional approaches, the neurocontrol approach needs the least a priori information about the controlled system models. Therefore, it is more robust to modeling errors. Compared with the iterative learning control approach, neurocontrol does not require the desired trajectories to be repetitive. Compared with the fuzzy control approach, neurocontrol can be applied to more general nonlinear systems, and does not require a set of fuzzy control rules, which are difficult to obtain for general complex dynamic systems.

Adaptive neurocontrol is a short term for adaptive control using neural networks. Since the network weights can be obtained through on–line learning, this approach is more suitable for systems that are difficult to model and have time–varying factors. Based on the existing problems of nonlinear control, the potentials of neurocontrol, and incapability of the existing approaches, this thesis proposes to research adaptive neurocontrol and apply it to control flexible joint robots.

1.4 ORGANIZATION OF THE THESIS

The remaining chapters of this thesis are organized as follows. Chapter 2 presents literature reviews on neurocontrol and its application to robotic manipulators, with emphasis on the classifications, the state–of–the–art, and the existing problems of the current re-searches. Based on the review, the objectives and scope of the thesis are proposed and defined.

Chapter 3 presents some new learning algorithms for neural networks. Localized neural networks are introduced to speed up the learning and reduce computation. New neural network models for nonlinear systems are also presented.

Chapter 4 presents new extensions of the backprop–based adaptive neurocontrol schemes. Little knowledge on control theory is required to apply these schemes.

Chapter 5 extends self–tuning control theory for linear systems to the self–tuning neurocontrol of nonlinear systems. This contribution greatly enriches the stability–based neurocontrol schemes.

A variable index control theory is proposed in Chapter 6. It can be applied to solve the control problem of general nonlinear systems, while the currently available nonlinear control schemes can only be applied to feedback linearizable systems and some nonlinear systems with special structures. As a result, a global stable adaptive neurocontrol scheme is derived for unknown nonlinear systems.

In Chapter 7, two robust adaptive neurocontrol schemes are presented for the trajectory control of flexible joint robots. The global stability of these neurocontrol systems are proved theoretically.

Experimental tests were performed on an existing flexible joint robot to evaluate the real–time performance of the neurocontrollers. The results are presented in Chapter 8.

The thesis concludes with Chapter 9, where a summary of the achievements, discussions and suggestions for future work are given.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter reviews the major results and important work in neurocontrol and neurocontrol of robots that have been done in recent years. The thesis tries to classify the current research trends, and reviews some representative work in each direction and summarizes the problems in existing works. Section 2.2 presents the review on neurocontrol. Section 2.3 presents the review on neurocontrol of robots. The reviews show that neurocontrol theory is still in the early stage of development. Basic problems, such as the stability and optimality of general neurocontrol systems, the global convergence of the learning algorithms, the structure of neurocontrol systems and neurocontrollers, and neural network model validity tests, are still under investigation. Much work has yet to be done in the neurocontrol area. Based on the literature reviews, the objectives and scope of the thesis are defined in section 2.4.

2.2 LITERATURE REVIEW ON NEUROCONTROL

It is difficult to categorize one neurocontrol scheme into a specific class and cover all the existing neurocontrol approaches now, because the neurocontrol area is still not well developed and new ideas are still emerging. Based on the conventional control concepts and the major existing results in neurocontrol, the neurocontrol schemes are categorized into the following:

(1) Teacher–replacing neurocontrol scheme:

In these schemes, neural networks are trained to copy an existing "teacher controller" of a dynamic system, then replace the teacher and work alone to control the system (fig. 2.1). The teacher can be other controllers, human operators, or heuristic control rules. Some researchers call this kind of neurocontroller copying a teacher controller.

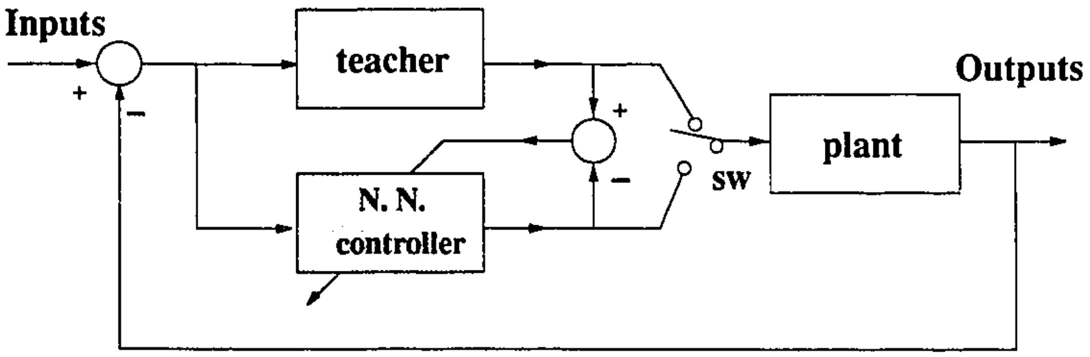


Fig. 2.1 Block diagram of teacher–replacing neurocontrol systems

(2) Inverse neurocontrol scheme:

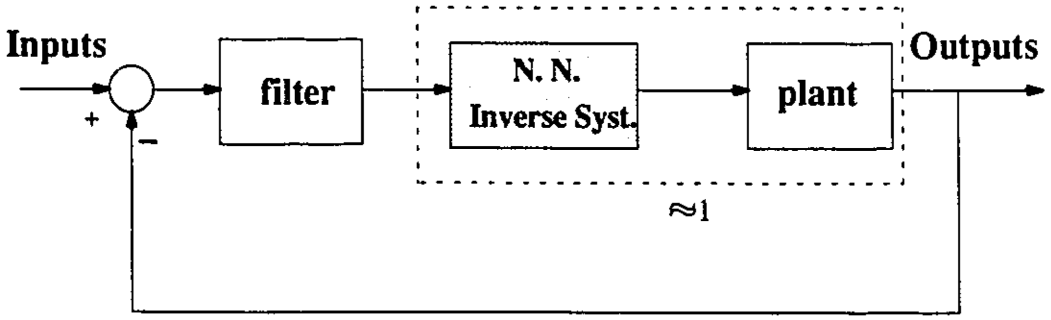


Fig. 2.2 Block diagram of inverse neurocontrol systems

This scheme is based on the inverse system theory. The neural networks are trained to model the inverse of the original systems, and then used to cancel the plant dynamics (fig.

2.2). The filter is used to limit the band–width of the systems. Here, the learning ability of neural networks is utilized for off–line design, not for on–line adaptation to system dynamic changes.

(3) Indirect adaptive neurocontrol scheme:

This scheme is one of the popular neurocontrol schemes and has been studied intensively. It consists of a neural network emulator of system dynamics and a feedforward neurocontroller (fig. 2.3). The neural network emulator is trained based on the modeling error e_M. The neurocontroller is trained by back–propagating the output tracking error e through the neural network emulator and the neurocontroller. It can perform on–line learning and adapts to system model errors. If the neural network structure and its initial weights are set properly, and the learning algorithms are efficient enough, a stable neurocontroller may be obtained, but the system stability is difficult to prove theoretically.

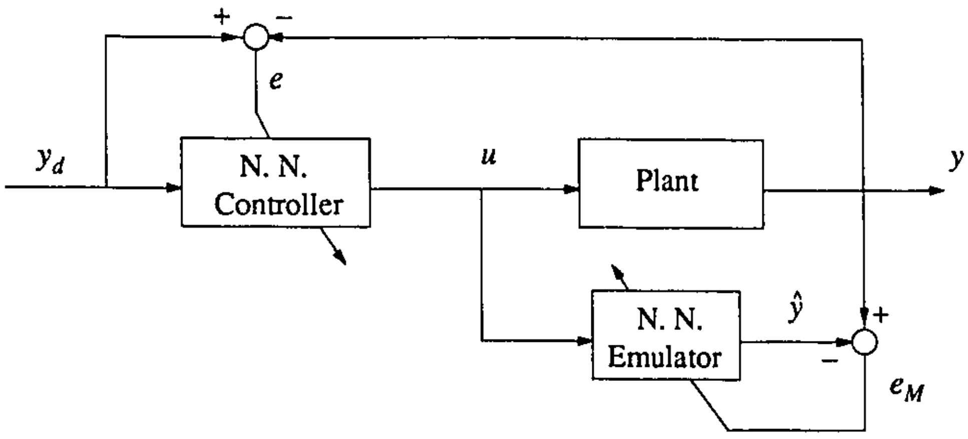


Fig. 2.3 Block diagram of indirect adaptive neurocontrol systems

(4) Direct adaptive neurocontrol scheme:

This is one of the neurocontrol schemes proposed at the beginning of neurocontrol research. There is only one neural controller, which is tuned using the output tracking error e (fig. 2.4). Since the desired value of u or the dynamics of the plant are unknown, the stan-

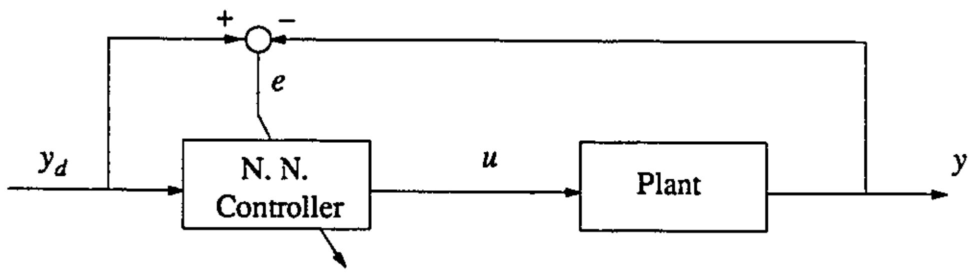


Fig. 2.4 Block diagram of direct adaptive neurocontrol systems

dard back–propagation strategy cannot be applied to this scheme directly. Many neurocontrol schemes derived from stability theorems belong to this approach.

(5) Robust neurocontrol schemes

This approach combines conventional robust control schemes with neural network models. The neural network models can be obtained by off–line or on–line training. The

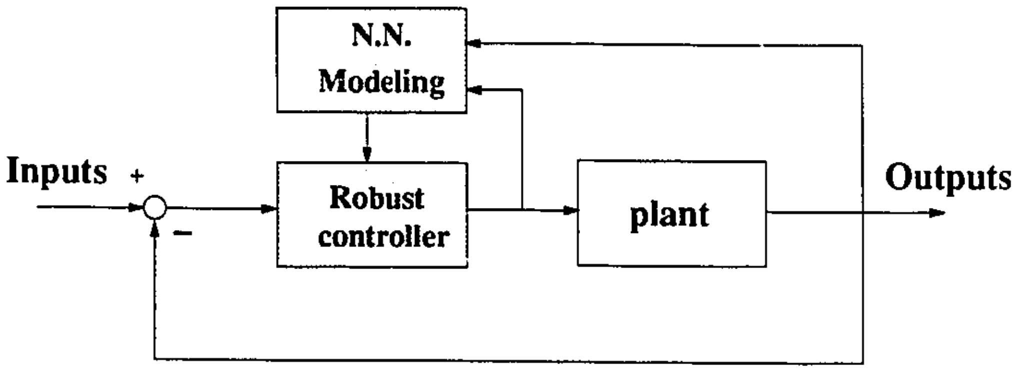


Fig. 2.5 The block diagram of robust neurocontrol scheme

robust neurocontrollers are designed based on the neural network models. Since there are usually modeling errors using finite numbers of neurons, robust neurocontrol is more attractive. Few robust control schemes can be applied to design the neurocontroller due to the complexity of neural networks.

There are also reinforcement neurocontrol, adaptive critic neurocontrol, optimal neurocontrol, fuzzy neurocontrol, fault tolerant neurocontrol, hybrid and hierarchical neurocontrol, etc. The structures of those neurocontrol systems are similar to the above schemes. However, their learning algorithms may have great differences. We will not discuss them since they are beyond the scope of this thesis.

The learning strategies of neurocontrol schemes can be classified into pure on–line learning, pure off–line learning, and hybrid on–line and off–line learning. Pure off–line trained neurocontrollers cannot adapt to the system dynamic changes, and the systems performances are dependent on the quality of generalization of the neurocontrollers. Pure on–line trained neurocontrollers have bad initial control performances. The system stability is difficult to guarantee. Hybrid on–line and off–line trained neurocontrollers share the merits and demerits of both.

Most existing neurocontrol schemes can also be classified into backprop–based neurocontrol and stability–based neurocontrol. Both of them can be in direct and indirect forms. Backprop–based neurocontrol schemes refer to those whose learning algorithms are based on the back–propagation strategy. Stability–based neurocontrol schemes refer to those whose learning algorithms are based on stability theorems. The former is conceptually simpler.

In the following, the existing important inverse neurocontrol schemes, indirect adaptive neurocontrol schemes and direct adaptive neurocontrol schemes are reviewed. Their improvement, extension and innovation form the achievements of the thesis.

2.2.1 Relevant Work and Problems

(1) Inverse neurocontrol schemes:

This approach (fig. 2.2) was studied and applied widely by researchers in robotics and process control areas. The general and preliminary idea of inverse neurocontrol schemes can be found in Psaltis, Sideris and Yamamura (1987a, b).

Gu (1990) gave a theoretical proof for the existence of inverse systems expressed by eqs. (2.2.1) and (2.2.2). Then an inverse neurocontroller was constructed and compensated by a feedback controller. According to the theorem, higher order derivatives of the outputs are needed to complete the inverse control.

$$\dot{x} = f(x) + B(x)u \tag{2.2.1}$$

$$y = h(x) \tag{2.2.2}$$

Levin et al. (1991) constructed an inverse neurocontroller with a Hamming net as the error–correcting output layer. Fukuda et al. (1991) constructed an inverse system neural feedforward controller compensated by output error constant gain feedback.

Although there is a class of nonlinear systems that are invertible, some problems exist for this scheme, such as the necessity of feedback compensation, instability of the inverse systems of the non–minimum phase nonlinear systems, and trainability of the neural network inverse system models without high order derivatives of the system outputs. Also, the approach allows little adaptability to model uncertainty. Most of the given schemes are suitable for off–line trained neurocontrollers only.

(2) Indirect adaptive neurocontrol systems:

The structure and basic principle of indirect adaptive neurocontrol can be found in Psaltis, Sideris and Yamamura (1987a and b), and Guez, Eilbert and Kam (1988). Jordan and Jacobs (1989) applied this approach to control an unstable system with successful simulation. Wu, Irwin and Hogg (1991) applied the indirect adaptive neurocontrol scheme to the regulation problem of a nonlinear system.

Hunt and Sbarbaro (1991) used the indirect adaptive neurocontrol system structure to implement the inverse model control (IMC) scheme for discrete nonlinear systems. The condition for invertibility of discrete nonlinear systems was given and Gaussian RBF networks were used as system models.

Kasparian and Batur (1992) gave a neurocontrol scheme with its structure similar to fig. 2.3. The difference is that the model based prediction control strategy (k–step) is applied to the feedforward neurocontroller, and the Daviden's least–squares minimization technique is used as the learning algorithm.

Narendra and Parthasarathy (1990) proposed four different neural network models for discrete nonlinear systems. They reformulated the neural networks with backpropagation algorithms from a control point of view, and indicated the relationship between neurocontrol and conventional adaptive control. Multilayer perceptrons are used in their indirect adaptive neurocontrol schemes. Thus, system stability may not be proved.

The indirect neurocontrol approach is closely related to the self–tuning adaptive control approach for linear systems. Just like the development of adaptive control for linear systems in the 1960s, adaptive neurocontrol is at the point where the stability of the control systems cannot be proved theoretically, though many papers have reported that it works. The convergence speed of the learning processes is very important to the control systems. Another question is how to determine the input variables to the neurocontrollers to make sure that the neurocontrol systems are stabilizable.

Direct adaptive neurocontrol scheme (fig. 2.4) is simpler in structure and its on–line computation is less compared with the indirect scheme. The existing direct adaptive neurocontrol schemes can be classified into backprop–based and stability–based.

(3) Backprop–based direct adaptive neurocontrol systems:

Psaltis, Sideris and Yamamura (1987a) tried to give a training algorithm for direct adaptive neurocontrol by using the Euler formula to estimate the partial derivatives of outputs with respect to inputs. For multi–input, multi–output systems, this scheme leads to erroneous results, because it is difficult to determine how much output change is caused by a specific input. Also, the change of a control input is not small enough to guarantee an accurate estimation.

Kuperstein and Rubinstein (1989) developed a sensor–based robot learning control scheme. Although the building of controllers is more like learning control through trial and error, the scheme works similar to the way direct adaptive neurocontrol systems do.

Bar–Kana and Guez (1990) suggested an adjustable linear feedback control law for the following system:

$$\dot{x} = A(x) + B(x)u \tag{2.2.3}$$

$$y = C(x)x + D(x)u \tag{2.2.4}$$

A Widrow–Hoff learning algorithm was used to adjust the gain matrix. Though the paper claimed that the scheme worked well, the stability of the control system and its training algorithm are questionable.

Cui and Shin (1991) tried to derive a training algorithm based on the qualitative information about the system input–output relations for direct adaptive neurocontrol systems. However, it only works for SISO systems whose input–output partial derivatives are of fixed sign.

Ho et al. (1991) derived a direct adaptive neurocontrol scheme for the following system:

$$x(k+1) = A(x(k)) + B(k)u(k) + w(k) \tag{2.2.5}$$

$$y(k) = C(k)x(k) + v(k) \tag{2.2.6}$$

where $w(k)$ and $v(k)$ are stochastic disturbances; $B(k)$ and $C(k)$ are known. The backpropagation–like learning algorithm is adopted. The scheme can only adapt to the change in $A(x(k))$, so it is partially adaptive. However, this scheme can only be applied to systems where $C(k)B(k)$ is not equal to zero matrix, which means that the minimum relative degree of the systems is one. It is difficult to extend this approach to continuous–time systems, because $\frac{\partial y(t)}{\partial u(t)}$ cannot be defined analytically (it is determined by differential equations).

Sadegh (1992) attempted to give an alternative solution of direct adaptive neurocontrol. To guarantee stability, some a priori information about the gradients of the input–state relationship of the plant is required. The analytical controllability theory of sampled–data systems is used to obtain this information based on a nominal model of the plant.

The main obstacle of backprop–based direct adaptive neurocontrol scheme is that the output errors cannot be *backpropagated* through the unknown nonlinear systems, and the learning algorithms are only locally convergent. Most reported schemes suffer from slow convergence. Similarly, it is not clear how to determine the input variables to the neurocontrollers to make sure that the control systems are stabilizable.

(4) Stability–based direct adaptive neurocontrol systems:

A representative example of stability–based direct adaptive neurocontrol systems was given by Sanner and Slotine (1992). They combined the sliding mode control methodology with neural network modeling to guarantee the stability and convergence of the training process, in spite of possible modeling errors. The result can be applied to eq. (2.2.7).

$$x^{(n)} + f(x,\dot{x}, \ldots ,x^{n-1}) = bu \tag{2.2.7}$$

Chen (1991) gave a stable neurocontrol scheme for an SISO system (2.2.8)

$$\begin{aligned} y_{k+1} = f(y_k, \ldots, y_{k-n+1}, u_{k-1}, \ldots, u_{k-m}) + \\ g(y_k, \ldots, y_{k-n+1}, u_{k-1}, \ldots, u_{k-m})u_k \end{aligned} \tag{2.2.8}$$

A dead–zone was introduced to guarantee the robustness to network approximation errors.

Chen (1991) used neural networks to identify the model of SISO linearizable nonlinear systems (as described in eqs. (2.2.1) and (2.2.2)), and formed a simple robust control law based on the identified dynamics. System stability is guaranteed.

Xu et al. (1991) considered the synthesis of feedback linearization and variable structure control for uncertain systems with a neural network as feedforward compensation. Feedback linearization provides a systematic approach such that an appropriate coordinate system can be found where that system can be linearized. The variable structure control is designed to produce a desired overall behavior in spite of system uncertainties. By combining the neural network as a generalized feedforward compensation, the high gain of variable structure control near equilibrium can be greatly reduced.

A robust error–feedback tracking controller is constructed with a neural network as the identifier of the inverse of the input–output Jacobian of the controlled plant (Tseng and Hwang, 1991). If the reference signal is slowly varying, system stability can be guaranteed.

Levin and Narendra (1992) tried to stabilize a dynamical system around an equilibrium point using neural networks. They applied the concept of controllability to stabilize system states to a set point. However, the result is local and the robustness of the controller depends on the accuracy of the neural network model.

The impressive aspect of the stability–based adaptive neurocontrol approach is that the stability of the control systems can be proved theoretically. It also shows the close relationship between adaptive neurocontrol and the modern adaptive control theory. Due to the limitations of existing nonlinear control theories, this approach can currently be applied only to feedback linearizable nonlinear systems and some systems with special dynamic structures.

2.2.2 Need for Research

Neurocontrol research restarted around 1986, developed steadily during 1986–1989, and boomed afterwards with many application tests and some theoretical analyses. The methodology was mainly experimental, that is, drawing conclusions based on one example.

As pointed out by Antsaklis (1992), there were many hopes, but few accomplishments. The consensus was reached around 1992 that neural networks can be used to model general nonlinear systems and there is a close relationship between adaptive control and neurocontrol. Now research has switched from intuitive applications of large neural nets to theoretical issues and applications to more specific problems.

In short, there are still many problems to be solved in neurocontrol. Among them are:

1) Some basic problems in the control field, such as stability, quantitative robustness evaluation, applicability to general nonlinear systems, constraints on system dynamics, are still open areas for research.

2) How to design the structure and input variables of a neural network to make the control systems stabilizable is also an open question.

3) The comprehensibility of the neurocontrol laws is still not clear. This is related to the analysis problem of control systems.

4) Most of the applications to date are confined to multi–layer feedforward neural networks with the back–propagation algorithm or its variations as the learning algorithms. In recent years, some new neural network architectures were proposed which are more suitable for system dynamics. Therefore, new extensions with better results are expected to be achieved.

5) The neural networks are mainly used for function approximation at present. The recognition ability, decision making ability and skill acquisition ability of neural networks should also be utilized to design control systems with higher level of intelligence.

6) Much work is required to test neurocontrol schemes experimentally and apply them to practical systems.

2.3 LITERATURE REVIEW ON NEUROCONTROL OF ROBOTS

Robotic control research has a longer history, compared with neural networks. It started in the middle of the 1970s, and developed fast after 1985. There are many control approaches for robotic manipulators, including: PID control, computed torque or inverse dynamics control, self–tuning, model reference or parametric adaptive control, variable structure control, feedback linearization control and Lyapunov–based control. Recently, learning control, fuzzy control, expert control, autonomous control and neural network based control have also been proposed and tested for some robots.

According to the type of control task, robotic control can be classified as: position /orientation control, trajectory control, force control, hybrid force/position control, constrained motion control, and multiple robot coordination control. There are many applications of neural networks in robotic control. Among them are: inverse kinematics mapping, inverse image data mapping, inverse dynamics, modeling uncertainty, teacher–replacing neurocontrol (learn from other controllers), force control, trajectory control, and hybrid intelligent robotic control. As indicated in Chapter 1, the control problem of flexible joint robots is more complex than that of rigid robots.

2.3.1 Relevant Work and Problems

Neurocontrol of robots started when the neurocontrol research began, although most work was theoretical. This is natural since the initial and final goal of building robots is to make them function like human beings. Possessing a brain–like controller will make this goal closer. The papers by Guez, Eilbert and Kam (1988), Sidelis, Yamamura and Psaltis (1987a), and Tsutsumi and Matsumoto (1987) are some of the early works on robotic neurocontrol.

(1) Inverse kinematics and dynamics

Guo and Cherkassky (1989) investigated the inverse kinematics representation using neural networks. Eckmiller et al. (1989) did similar work for a redundant robot arm. Guez and Ahmad (1989) trained a neural network to learn the inverse kinematics of PUMA 560.

Kosmatopoulos et al. (1991) identified a neural network model of a two–link rigid robotic system by using the dynamic backpropagation algorithm.

Herve, Sharma and Cucka (1991) noticed that a small calibration error in a vision system would result in large error in the complex inverse kinematic computation. Therefore, a neural network was used to approximate the mapping from the space of configurations of robots directly onto a space of measurable image parameters, and to realize hand/eye coordination without calibration.

In summary, these results confirm that neural networks can solve coordinate transformation problems and arbitrary input–output relation mapping problems.

(2) Modeling uncertainty

When there is partial uncertainty in a robotic system, a neural network can be applied to model it and used to compensate for such uncertainty. Leahy, Johnson and Rogers (1991) modified the inverse dynamics control scheme for robots working under unknown load conditions by using neural networks to estimate the dynamic parameters of the unknown load and to compensate for the uncertainty.

(3) Teacher–replacing neurocontroller

Asada and Liu (1991) proposed a teacher–replacing scheme to transfer the skill of human operators into a neural network and use the generalization ability of the network to build the skill–based controller. This scheme can find wide applications in industrial manufacturing.

Lee and ElMaraghy (1992) constructed a neurocontroller for a walking robot by learning a set of linear control laws at different working points. Here the neurocontroller generalizes the linear controllers into a nonlinear controller.

(4) Force control

Neurocontrol has also been tested by executing force control of robotic manipulators. For example, Fukuda et al. (1991) applied a direct neurocontrol scheme to the impact control of a one–DOF manipulator.

Pei, Leung and Zhou (1992) proposed a novel learning strategy which decomposes the whole task into several parts. The scheme is demonstrated with an example of robotic manipulator position/force learning control.

There are also neurocontrol schemes for position/force control of multiple robots (Tao and Luh, 1993).

(5) Trajectory control

Trajectory control using neural networks was studied by many researchers. Bassi and Bekey (1989) applied a neural network to identify the inverse dynamics of a robot, and then used the identified model to construct a feedforward and feedback controller for trajectory control.

Hosogi (1990) studied the trajectory control of rigid robots by emulating the function of the cerebellar Golgi cell system. A Hebb–type learning rule was used as the self–organized mechanism.

Fadali et al. (1990) gave a minimum–time control scheme of robotic manipulators using a backpropagation neural network.

Leung, Zhou and Pei (1992) applied a stable neurocontroller to a rigid robot. The controller combined variable structure control with neural network feedforward control to guarantee system stability.

(6) Hybrid intelligent robotic control

Handelman, Lane and Gelfand (1989) tried to integrate knowledge–based systems and neural networks for autonomous robots. Lane, Handelman and Gelfand (1990) discussed the possibility of robot intelligence and some schemes to realize it.

Rabelo and Avula (1991) proposed a hierarchical neurocontroller architecture consisting of two neural network systems for the manipulation of a robotic arm. The higher level neural system participated in the coordinates transformation and motion decision making. The lower one provided the control action sequence.

Fukuda and Shibata (1991) and gave a concept and strategy of hierarchical intelligent control for robots. Basically, they regarded the hierarchical intelligent control system as a hybrid system of neural networks, fuzzy logic, and AI, and divided the system into three levels: learning level, skill level and adaptation level. The adaptation level is realized by a neuromorphic controller in an uncertain environment. The skill level consists of trained fuzzy neural networks which emulate the human operation skills. The fuzzy logic works as the intermediate connecting neural networks and symbolic reasoning systems. The learning level recognizes the environment and the manipulated objects and makes decisions based on the sensed information. Detailed implementation is still under development.

2.3.2 Need for Research

The ultimate goal of intelligent robots is to possess high level intelligence and mimic human activities (Saridis, 1983, and Lane, Handelman and Gelfand, 1990). To achieve this goal, intelligent robots should have an artificial brain, a multiple redundant movement mechanism with efficient control systems, and a set of intelligent sensors that provide vision, hearing, contact force, touching and approaching information. Neural networks can play roles in all the three parts of intelligent robots. At present, PID control is still the main choice of most industrial robots, due partly to its simplicity and the unavailability of reliable substitutes.

The conventional robotic control schemes usually require mathematical models of the controlled robots, and are sensitive to model errors, disturbances and noises. When a robot's task is changed, control algorithm adjustment and related tedious planning are required to make the robot execute the new task. The decision making, planning, sensor information processing and control are separated into different subjects, which makes their integration difficult. The level of achieved robot autonomy is very primitive at present.

Currently, the theories of neural network based modeling and control are still under development. Their applications to robotic control were preliminary. The main problems are the slow learning convergence of neural networks, and intuition–based neurocontrol algorithms. How to determine an appropriate size (number of layers and neurons) of a neural network is another problem. Most existing adaptive neurocontrol schemes are difficult to implement in real–time due to their complexity. Solid theories which deal with the neurocontrol system structures and stability have yet to be developed.

2.4 OBJECTIVES AND SCOPE OF THE THESIS

The objectives of my thesis are: (1) to develop new adaptive neurocontrol schemes with better learning performance and guaranteed system stability for general nonlinear systems, and (2) to apply them to the trajectory control problem of flexible joint robots.

Based on the above objectives, the scope of the thesis is defined as follows.

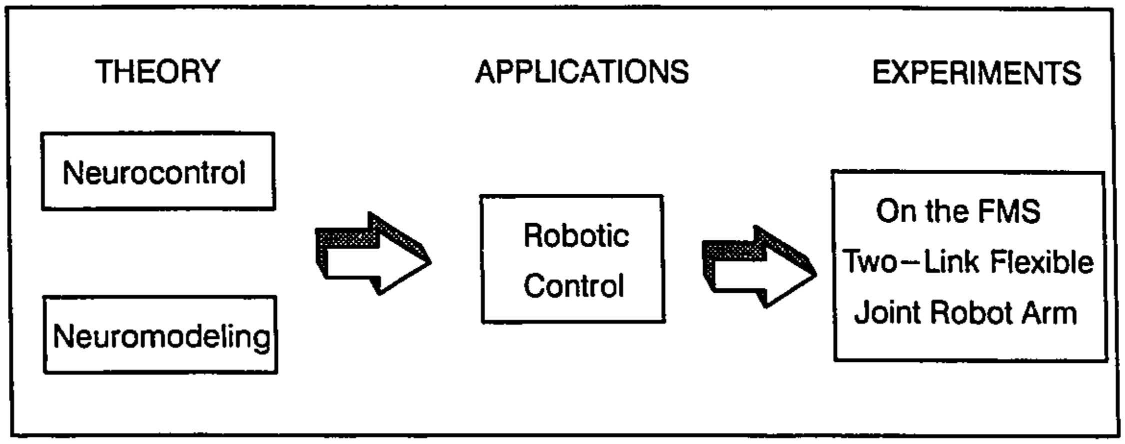


Fig. 2.6 The outline of the thesis objectives

1) to develop new backprop–based adaptive neurocontrol schemes using new learning algorithms and new neural networks

The convergence speed of the learning processes is very important to the stability and tracking errors of backprop–based adaptive neurocontrol systems. With new learning algorithms and new neural network structures introduced, better backprop–based adaptive neurocontrol schemes can be obtained. At the same time, the problem of determining the input variables to the neurocontrollers to make sure that the control systems are stabilizable will be addressed.

2) to develop self–tuning adaptive neurocontrol schemes

It is a general perception that neural network models of nonlinear systems are not useful for analytical controller design. This is still true for complex multilayer neural network models and some recurrent neural network models for nonlinear systems. With the introduction of new neural network models for nonlinear systems and the localized neural network concept, analytical design of neurocontrollers becomes possible. This research may greatly enrich the neurocontrol algorithms with guaranteed system stability, and establish a relationship between self–tuning control theory for linear systems and self–tuning neurocontrol for nonlinear control systems.

3) to develop new stability–based adaptive neurocontrol schemes for general nonlinear systems

There are only few stability–based adaptive neurocontrol schemes available currently, and they can only be applied to some feedback linearizable nonlinear systems. This is because there are few control schemes for nonlinear systems. The thesis proposes a new robust control theory for general nonlinear systems. Then a new robust adaptive neurocontrol scheme is derived with theoretically proved stability. Compared with the existing adaptive control schemes for nonlinear systems, the new neurocontrol scheme can be applied to nonlinear systems which are not necessarily linearly–parameterizable and/or feedback linearizable.

4) to develop robust adaptive neurocontrol schemes for flexible joint robots

Due to the existence of modeling errors, most conventional control schemes for flexible joint robots result in limited control precision. There is no stability–based adaptive neurocontrol scheme for flexible joint robots yet. Theoretically, all the neurocontrol schemes developed in the thesis may be applied to the control of flexible joint robots. Using a priori information about the dynamics of flexible joint robots, simpler neurocontrol schemes can be derived. These observations lead to the research reported in chapter 7 of the thesis.

5) to test the real–time features of neurocontrol schemes by an experiment using the available two–link flexible joint robot.

There are few experimental results on neurocontrol reported in the literature to date. To check the effects of real–time finite accuracy computation, discretization and measurement noises, an experiment was conducted using an available two–link flexible joint robot arm.

CHAPTER 3

NEW DEVELOPMENTS ON NEURAL NETWORKS

3.1 INTRODUCTION

There are two distinct approaches in neural network research. One is the biologically–motivated approach. The goal of this approach is to make the functions and behaviors of neural networks resemble those of the brain. Mathematics is secondary in this approach. The other is an application–oriented approach. The computation ability of neural networks is applied in different aspects according to different applications. Optimization techniques, approximation theory (Braess, 1986), and computational learning theory (Anthony and Biggs, 1992) are also applied to the development of neural networks. The thesis adopts the application–oriented approach.

The topological structures of neural networks can be classified into networks with one hidden layer, two hidden layers, *n* hidden layers, and irregular neural networks. The neurons can be fully connected or partially connected to each other, and the flow direction of signals can be feedforward only, or both feedforward and feedback (recurrent).

There are many different neural networks proposed in the literature. Their topological structures are the same as those classified above. The major difference among those neural networks is their activation functions. The common activation functions are: 1) sigmoidal functions (hyperbolic functions), 2) Gaussian functions, 3) thin–plate–spline functions, 4) multi–quadratic functions, 5) polynomial functions, 6) rational functions, 7) step functions, 8) spline functions, 9) trigonometric functions, 10) cosig functions, 11) wavelet functions, 12) orthogonal functions (Legendre, Chebychev, and Jacobi polynomials), 13) feature functions (box, CMAC), etc. These activation functions can be further classified into monotonous, semi–monotonous and non–monotonous, which may lead to global mapping, hybrid mapping, and local mapping networks at neuron level.

Cybenko (1989) proved that sigmoidal neural networks (i.e., multilayer perceptrons) are universal approximators. Poggio and Girosi (1990) proved that RBF networks are universal approximators. Cotter (1990) proposed several activation functions satisfying the Stone–Weierstrass theorem, and built the corresponding networks. The most exciting result was given by Leshno, Lin, Pinkus, and Schocken (1993). They proved that multilayer feedforward networks with a non–polynomial activation function can approximate any function. The differences in approximation performances using different activation functions are in their smoothness constraints (regularization theory). Also, different numbers of neurons are required for a given problem, if different activation functions are used. The selection principles of different activation functions are: (a) it fits the problem; (b) it is easy to realize by hardware.

The remainder of this chapter is organized as follows. Section 3.2 introduces the concept of localized neural networks to speed up the learning convergence and reduce the computation. An adaptive extended Kalman filtering algorithm is also presented to achieve fast learning. Section 3.3 introduces the multistep localized adaptive learning algorithm with optimal learning rate for Gaussian networks. Fast convergence and reduced computation are also achieved by using this algorithm. Section 3.4 presents some neural network models for nonlinear system, and discusses some issues about generalization and learnability. Section 3.5 gives a summary.

3.2 LOCALIZED NEURAL NETWORKS WITH CLI CELLS

It is difficult and time–consuming to train a large size artificial neural network, especially in a real–time application. This is due to the globally interconnected structures of artificial neural networks, which make the connection weights dependent on each other and make their updating very inefficient when the number of neurons reaches a certain level. Although biological neural networks do consist of a massive number of neurons, it is doubtful that every neuron can respond to all outside stimulations in the same manner, and all the synaptic weights are adjusted at the same time for a specific stimulation. Bioanatomy research proved that different areas of the cerebellar cortex correspond to different functions of the brain (microzones and microcomplexes), and that there are functionally different kinds of neurons, such as Purkinje cells, Golgi cells, granule cells and basket cells (Ito, 1984). These facts suggest that the brain possesses a functional localization property, and that neural networks can contain different types of neurons.

The localized feature function concept has been applied by many researchers. For example, Michie and Chambers (1968) built the "Boxes" system, which stored information in a lookup table form. The location of a piece of information was determined by its localized feature functions. The CMAC proposed by Albus (1975, 1979) is a more general lookup table, using feature functions that aggregate the values of underlying variables into overlapping regions instead of disjoint regions. Information is stored by spreading each item to be stored over neighboring table entries according to weighting profiles given by the feature functions. This approach is simple and guarantees fast learning convergence (Wong and Sideris, 1992), but the generalization ability of CMAC is limited by the fineness of the discretization. Moody and Darken (1988) investigated two–layer neural networks based on Albus' model of the cerebellum with radial basis function (Gaussian) activation units in the single hidden layer. The Gaussian functions possess a localized receptive field property. There were many extensions along this line. All the above reviewed work is limited, however, to the localization of the receptive field of a single neuron.

In the following, competitive lateral inhibitory (CLI) cells are introduced to localize the function of a neural network according to the receptive fields of its sub–networks. The CLI cells can be regarded as the functional emulation of the biological Golgi cells (Keeler, 1991). As a special case, the localized polynomial neural networks with competitive lateral inhibitory (CLI) cells are introduced. The advantages of the new networks include: 1) fast learning and global convergence property, which is essential for applications that need real–time or fast learning; 2) modular and additive structure, which makes the overall network structure easy to determine; 3) extendibility to new input space without affecting the learned relationship; and 4) graceful degradation.

This section is based on two papers by Liang and ElMaraghy (1993b and c).

3.2.1 Review of polynomial neural networks

Let $f(x_1, ..., x_N)$ be a real function continuous on the bounded region $\Omega \subset R^N$. Then the Weierstrass approximation theorem guarantees that there exists a real polynomial $P(x_1, ..., x_N)$ that can approximate $f(x_1, ..., x_N)$ with given precision, where:

$$P(x_1, \ldots, x_N) = a_0 + \sum_j a_j x_j + \sum_{j,k} a_{jk} x_j x_k + \sum_{j,k,l} a_{jkl} x_j x_k x_l + \cdots \qquad (3.2.1)$$

Define

$$x = [x_1, \ldots, x_N]^T \qquad (3.2.2)$$

$$\Phi(x) = [\phi_1(x) \ldots \phi_p(x)]^T = [1 \; x_1 \ldots x_N \; x_1^2 \; x_1 x_2 \ldots x_j x_k x_l \ldots]^T \qquad (3.2.3)$$

$$w = [w_1, \ldots, w_p]^T = [a_0 \; a_1 \ldots a_N \; a_{11} \; a_{12} \ldots a_{NN} \; a_{111} \cdots]^T \qquad (3.2.4)$$

where:

$$p = \sum_{i=0}^{L} \binom{N+i-1}{i} \qquad \text{and } L \text{ is the order of the polynomial} \qquad (3.2.5)$$

Then eq. (3.2.1) can be reformulated into the following form:

$$P(x_1, \ldots, x_N) = P(x) = w^T \Phi(x) = \sum_{l=1}^{p} w_l \, \phi_l(x) \qquad (3.2.6)$$

The polynomial neural networks are structured based on eq. (3.2.6). Assume that the inputs and outputs of a polynomial neural network are $x_1, \ldots, x_N$ and $y_1, \ldots, y_M$, respectively. Then its input–output relations are

$$\textit{Hidden layer:} \qquad z_l = \phi_l(x) \qquad l = 1, \ldots, p \qquad (3.2.7)$$

$$\textit{Output layer:} \qquad y_m = \sum_{l=1}^{p} w_{lm} z_l \qquad m = 1, \ldots, M \qquad (3.2.8)$$

where p determines the number of the hidden nonlinear neurons; z_l is the output of the l–th hidden nonlinear neuron.

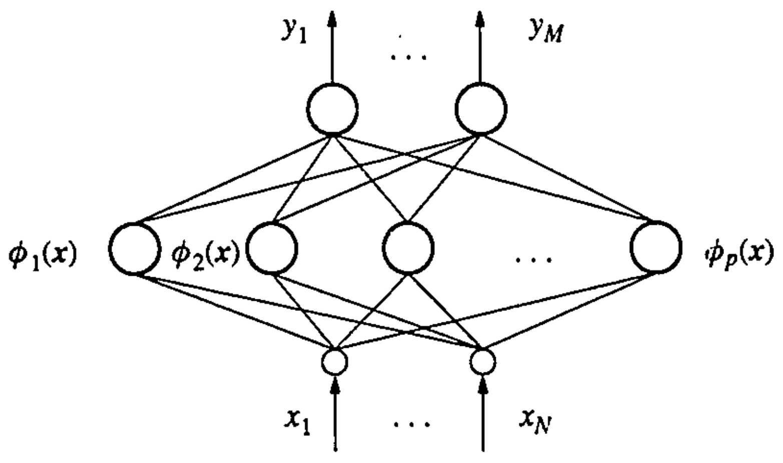


Fig. 3.1 The structure of polynomial neural networks

Fig. 3.1 shows the structure of the polynomial neural networks. Actually, this structure is valid for any function basis { $\phi_1(x), \ldots, \phi_p(x)$ }, for instance, the Chebyshev orthogonal polynomial basis, the Fourier trigonometric function basis, the Gaussian function basis and so on. Here we only discuss the case of polynomial neural networks defined by eqs. (3.2.7) and (3.2.8). The results in this paper can be directly applied to other cases by simply changing the definition of $\phi_1(x), \ldots, \phi_p(x)$.

Assume that there are unknown mappings between $x_1, \ldots, x_N$ and $y_1^d, \ldots, y_M^d$, which are expressed as follows:

$$y_m^d = f_m(x_1, \ldots, x_N) \qquad m = 1, \ldots, M \tag{3.2.9}$$

where the functions are continuously differentiable. If there are enough training data $\{ x_1(k), \ldots, x_N(k); \; y_1^d(k), \ldots, y_M^d(k) \mid k = 1, \ldots, K \}$ which can represent the mapping relations between $x_1, \ldots, x_N$ and $y_1^d, \ldots, y_M^d$, and the hidden neuron number p is unlimited, then the polynomial neural network shown in fig. 3.1 can approximate the unknown mappings (3.2.9) with arbitrary precision.

The backpropagation learning algorithm is commonly adopted to train the networks. Although the polynomial neural networks are more efficient in learning than conventional neural networks, such as multilayer feedforward perceptrons, the networks are more sensitive to the input noises and computational errors due to the direct product of inputs in the hidden layer, especially when the orders of the networks are high. Also, the efficiency of learning will be affected by massive weight coupling when the orders of the networks are high, which leads to a large scale optimization problem. In the following, the demerits of the polynomial neural networks are remedied by introducing the competitive lateral inhibitory cells and the localized representation.

3.2.2 Localized polynomial networks

For the function approximation problem stated in section 3.2.1, if the bounded working region $\Omega \subset R^N$ is divided into J smaller subregions, $\Omega_1, \ldots, \Omega_J$, then $f(x)$ can be represented in each subregion Ω_j by a much lower order polynomial with the same given precision.

$$f(x) \doteq P(x) = \sum_{l=1}^{p} w_l^j \, \phi_l(x) \equiv f_j(x) \qquad x \in \Omega_j \tag{3.2.10}$$

Define an input receptive field selection function for each subregion as follows:

$$s_j(x) = \begin{cases} 1 & x \in \Omega_j \\ 0 & x \notin \Omega_j \end{cases} \tag{3.2.11}$$

Then $f(x)$ can be expressed in the following localized representation:

$$f(x) = \sum_{j=1}^{J} s_j(x) \, (\sum_{l=1}^{p} w_l^j \, \phi_l(x) \,) = \sum_{j=1}^{J} s_j(x) f_j(x) \tag{3.2.12}$$

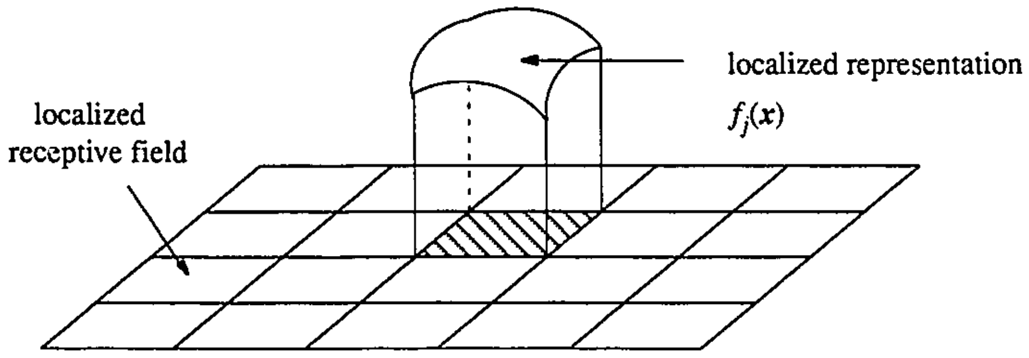


Fig. 3.2 Illustration of localized receptive fields and localized representation

Fig. 3.2 illustrates the localized receptive fields and localized representation of a given nonlinear function. It can be seen that although a nonlinear function can be very complex, in a small region, i.e., a localized receptive field, the function is still simple enough to be approximated by a low order polynomial, whose order can often be set to less than 3 in many cases. Such low order polynomial neural networks are very easy and fast to train. Therefore, if we can realize eq. (3.2.12), the localized representation of a given nonlinear function, then a new efficient neural network can be generated in which only a small size sub–neural network is needed to be trained and responsible for output at each step. Also, efficient forward computation and fast learning convergence are guaranteed. Since each localized representation can be approximated by a low order polynomial neural network, the remaining problem is how to realize the localized receptive field divisions and join them together. This is done by introducing a set of competitive lateral inhibitory (CLI) cells:

The localized receptive fields can be represented at least in two ways: 1) as a set of superboxes and 2) as a set of Gaussian potential functions. Such realizations define the function of the CLI cells, which resemble the function of biological Golgi cells in the cerebellum. Take the case of using Gaussian potential functions.

The inputs to the CLI cells are $\boldsymbol{x} = [x_1, \ ..., \ x_N]^T \in \Omega \subset R^N$, as shown in fig. 3.3. The CLI cells perform the following computation:

(1) Gaussian potential function computation to determine the potential of the current inputs in each localized receptive field:

$$p_j(\boldsymbol{x}) = \exp\{ - \sum_{n=1}^{N} \frac{(x_n - c_{jn})^2}{\sigma_{jn}^2} \} \qquad j = 1, \ldots, J \tag{3.2.13}$$

(2) Competitive lateral inhibition to select a unique excitatory receptive field:

$$s_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \ p_j(\boldsymbol{x}) = \max \ \{p_1(\boldsymbol{x}), \ \ldots, \ p_J(\boldsymbol{x})\} \\ 0 & \text{if } \ p_j(\boldsymbol{x}) < \max \ \{p_1(\boldsymbol{x}), \ \ldots, \ p_J(\boldsymbol{x})\} \end{cases} \tag{3.2.14}$$

Because only one $s_j(\boldsymbol{x})$ is allowed to be nonzero for each input vector $\boldsymbol{x}$, randomly select one $s_j(\boldsymbol{x})$ to be excitatory and set the other $s_j(\boldsymbol{x})$ to be zero when there is a tie. This reflects the lateral inhibition property.

The receptive field center of a CLI cell, $\boldsymbol{c}_j = [c_{j1}, \ ..., \ c_{jN}]^T$, can be determined by random or deterministic selection, unsupervised learning and statistical estimation.

It can be proved that if all the σ_{jn} are set to be equal, then the receptive fields $\Omega_1, \ ..., \ \Omega_J$ defined by the CLI cells are compact and disjoint except on the field boundaries, and their union forms the whole working region W. This is an important property of the Gaussian potential function.

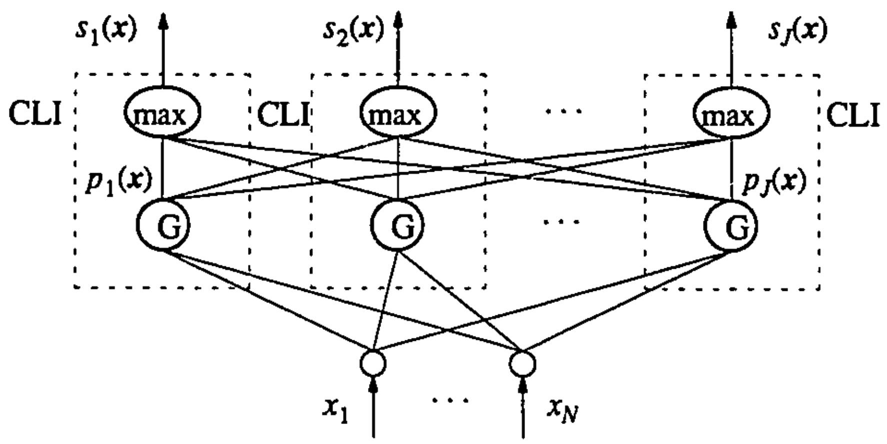


Fig. 3.3 Localized receptive field division using CLI cells

Based on the CLI cells, the structure of the localized polynomial neural networks with CLI cells is defined as shown in fig. 3.4 to realize the representation (3.2.12), where

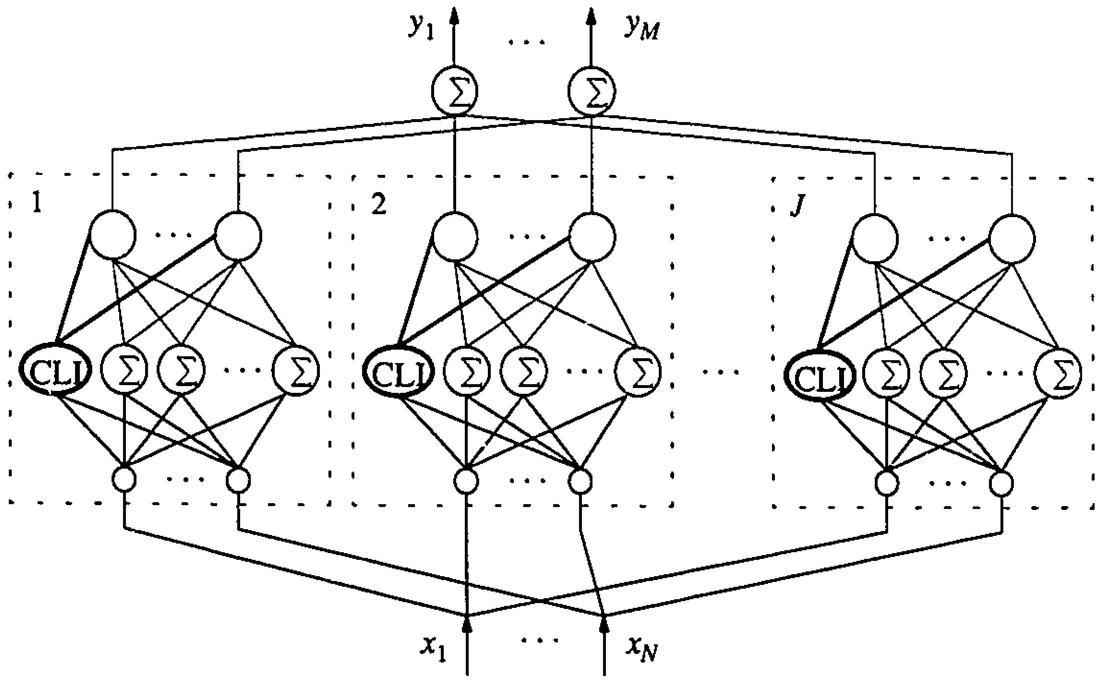


Fig. 3.4 The structure of localized polynomial networks with CLI cells

$f_j(x)$ is changed into $f_{jm}(x)$ in the multi–output case and realized by a low–order local polynomial neural network, and $s_j(x)$ is realized by the CLI cells. The interconnection lines between different CLI cells are omitted in the figure. The outputs of the overall multi–output network are:

$$y_m = \sum_{j=1}^{J} s_j(x) f_{jm}(x) \qquad m = 1, \ldots, M \tag{3.2.15}$$

The number of the CLI cells for each output, J, is related to the order of each sub–polynomial neural network, L. When J is large, L can be small. When the input space is not fixed, i.e., $\Omega \subset R^N$ is changeable, the following heuristic algorithm can determine the number of the CLI cells for each output during the training.

(1) If max $\{p_1(x), \ldots, p_J(x)\} < B_L$, add a new sub–polynomial neural network and let the center of its CLI cell locate at x.

(2) If there are several j such that $p_j(x) > B_U$, delete one sub–polynomial neural network that is redundant.

B_L and B_U are the lower and upper boundaries chosen by the designers. In the next section, the optimal Kalman filtering algorithm is applied to train the localized networks.

3.2.3 The optimal Kalman filtering learning algorithm

The backpropagation–like learning algorithm for the localized polynomial neural networks with CLI cells is easy to derive. However, it is slow in convergence. The extended Kalman filtering (EKF) algorithm is one of the fast training algorithms for neural networks in use (Singhal and Wu, 1989). Its drawbacks are the intensive computation, suboptimality and possible filtering divergence. In this paper, the trained network size is small at each training step due to the localized representation. Therefore, the EKF learning algorithm can be applied directly without intensive computation. Due to the special structure of the polynomial networks, the EKF learning algorithm is actually the optimal Kalman filtering algorithm, so global learning convergence of the network weights and thresholds can be achieved regardless of their initial values. Furthermore, the recursive UD factorization algorithm is applied to the optimal Kalman filtering algorithm to improve its stability of numerical computation. Thus, the proposed new networks with the new learning algorithm can remedy all three drawbacks stated above.

For the m–th output, if the j–th CLI cell is inhibitory, or $s_j(x) = 0$, then the j–th subnetwork weights and thresholds remain the same; if the j–th CLI cell is excitatory, or $s_j(x) = 1$, then the j–th subnetwork weights and thresholds can be updated using the following learning algorithm.

To apply the adaptive optimal Kalman filtering algorithm, the neural network input–output relations with unknown weight vectors are expressed in state space form as:

$$W^j_m(k+1) = W^j_m(k) \qquad m = 1, \ldots, M \qquad (3.2.16)$$

$$y^d_m(k) = y^j_m(k) + v^j_m(k) = \Phi(x)^T\, W^j_m(k) + v^j_m(k) \qquad m = 1, \ldots, M \qquad (3.2.17)$$

where $W^j_m(k)$ is the weight and threshold vector of the j–th sub–polynomial neural network for the m–th output; $y^d_m(k)$ is the desired m–th output; $y^j_m(k)$ is the actual m–th output of the overall neural network, which is equal to the output of the j–th sub–polynomial neural network; $v^j_m(k)$ is the approximation error between the desired m–th output and the actual m–th output. $\{ v^j_m(k)\}$ can be regarded as a white noise process, but not necessarily zero–mean. Let

$$E\{v^j_m(k)\} = r^j_m(k) \qquad Cov\{v^j_m(k),\ v^j_m(i)\} = R^j_m(k)\delta_{ki} \qquad (3.2.18)$$

In the following, an adaptive optimal Kalman filtering (or AOKF) algorithm (Liang, Zhang and Li, 1988) is applied to eqs. (3.2.16) and (3.2.17), which gives:

$$W^j_m(k+1) = W^j_m(k) + K^j_m(k)e^j_m(k) \qquad m = 1, \ldots, M \qquad (3.2.19)$$

$$e^j_m(k) = y^d_m(k) - \Phi(x(k))^T\, W^j_m(k) - \hat{r}^j_m(k) \qquad m = 1, \ldots, M \qquad (3.2.20)$$

$$K^j_m(k) = P^j_m(k)\Phi(x(k))/[\ \hat{R}^j_m(k) + \Phi(x(k))^T P^j_m(k)\Phi(x(k))\ ] \tag{3.2.21}$$

$$P^j_m(k+1) = P^j_m(k) - K^j_m(k)\Phi(x(k))^T P^j_m(k) \tag{3.2.22}$$

$$\hat{r}^j_m(k+1) = \hat{r}^j_m(k) + \lambda(k)e^j_m(k) \tag{3.2.23}$$

$$\hat{R}^j_m(k+1) = \hat{R}^j_m(k) + \lambda(k)\ [\ e^j_m(k)^2 - \hat{R}^j_m(k) - \Phi(x(k))^T P^j_m(k)\Phi(x(k))\ ] \tag{3.2.24}$$

where

$$\lambda(k) = (1-b)/(1-b^k) \qquad 0 < b < 1 \tag{3.2.25}$$

and the initial values of the above equations are

$$W^j_m(0) = W_0 \qquad \hat{r}_m(0) = 0 \tag{3.2.26}$$

$$P^j_m(0) = P_0 > 0 \qquad \hat{R}^j_m(0) = R_0 > 0 \tag{3.2.27}$$

$P^j_m(k)$ and $\hat{R}^j_m(k)$ are the parameter variance matrix and output error variance matrix respectively, and are symmetric positive definite. Due to computation error, $P^j_m(k)$ computed from eq. (3.2.22) can easily lose its positive definite property, which results in filtering divergence. Here, the recursive UD factorization is adopted to compute eqs. (3.2.21) and (3.2.22). Let

$$P^j_m(k) = U(k)D(k)U^T(k) \qquad U(k) = [\ U_1(k),\ U_2(k),\ .\ .\ .\ .\ ,U_p(k)\ ] \tag{3.2.28}$$

$$D(k) = diag\{\ d_1(k),\ d_2(k),\ .\ .\ .\ .\ ,d_p(k)\ \} \qquad 0 < C_L < d_i(k) < C_U \tag{3.2.29}$$

where p is the dimension of $W^j_m(k)$ and is defined by eq. (3.2.5); $U_i(k) \in R^p$; C_L and C_U are the lower and upper boundaries of $d_i(k)$ which can guarantee $P^j_m(k)$ to be positive definite and not to diverge. Let $U(0) = I$, and $D(0) = \mu I$ (μ>0). $U(k)$ is an upper triangular matrix with diagonal elements equal to one. Then the AOKF algorithm with recursive UD factorization is:

(1) Compute forward the excitatory sub–polynomial neural network;

(2) Compute scalar variables:

$$f_i = U_i^T(k)\Phi(x(k)) \qquad v_i = d_i(k)f_i \qquad i = 1, \ldots, p \tag{3.2.30}$$

(3) Let $\alpha_0 = \hat{R}_m^j(k)$, $L_0 = [0\ 0\ \ldots\ 0]^T$ and for i=1, . . . , p, compute:

$$\alpha_i = \alpha_{i-1} + f_i v_i \qquad d_i(k+1) = d_i(k)\alpha_{i-1}/\alpha_i \tag{3.2.31}$$

$$L_i = L_{i-1} + U_i(k)v_i \qquad \beta_i = -f_i/\alpha_{i-1} \tag{3.2.32}$$

$$U_i(k+1) = U_i(k) + \beta_i L_{i-1} \tag{3.2.33}$$

(4) Change the gain vector defined in eq. (3.2.21) into:

$$K_m^j(k) = L_p/\alpha_p \tag{3.2.34}$$

and change eq. (3.2.24) into:

$$\hat{R}_m^j(k+1) = \hat{R}_m^j(k) + \lambda(k)\ [\ e_m^j(k)^2 - \alpha_p\ ] \tag{3.2.35}$$

with eqs. (3.2.19), (3.2.20) and (3.2.23) remaining the same.

Repeat the above steps until $W_m^j(k)$ is convergent. The UD–AOKF algorithm needs the same computation amount (the order of complexity is $O(p^2)$) and storage as the standard Kalman filtering algorithm without UD factorization does, but possesses much better numerical convergence than the standard Kalman filtering algorithm.

The storage for the p_1 weights of a localized polynomial network is equal to $J \times p_1$ and might be larger than that for the p_2 weights of a global polynomial neural network. However, if both the localized and global polynomial neural networks are trained by the UD–AOKF algorithm, in which the storage for $P_m^j(k)$ in eq. (3.2.22) or for $U(k)$ and $D(k)$ in eq. (3.2.31) and (3.2.33) is also of the order of $O(p^2)$, then the overall storage required by the localized networks is equal to $J \times [\,p_1 + O(p_1^2)]$, which is no more than that of the global networks, $p_2 + O(p_2^2)$, since $p_1 << p_2$.

3.2.4 Extension to localized pi–sigma networks

Eq. (3.2.1) can be reformulated into the following form:

$$P(x_1, \ldots, x_N) = \prod_{l=1}^{\infty} \left(\sum_{n=1}^{N} w_{ln} x_n + \theta_l \right) \tag{3.2.36}$$

which determines the structure of pi–sigma networks, and clearly explains the term pi–sigma.

If the inputs and outputs of a pi–sigma network are $x_1, \ldots, x_N$ and $y_1, \ldots, y_M$, respectively, then the input–output relations of the single layer pi–sigma network are:

$$z_{lm} = \sum_{n=1}^{N} w_{lmn} x_n + \theta_{lm} \qquad \begin{array}{l} l = 1, \ldots, L \\ m = 1, \ldots, M \end{array} \tag{3.2.37}$$

$$y_m = \prod_{l=1}^{L} z_{lm} + \theta_m \qquad m = 1, \ldots, M \tag{3.2.38}$$

where L determines the order of the pi–sigma networks. $L = 1$ gives linear input–output relation. z_{lm} is the output of l–th summing unit for the m–th output. The threshold θ_m is added in eq. (3.2.38) to extend the representation ability of the pi–sigma networks to include the case of real polynomials with complex roots for low order pi–sigma networks.

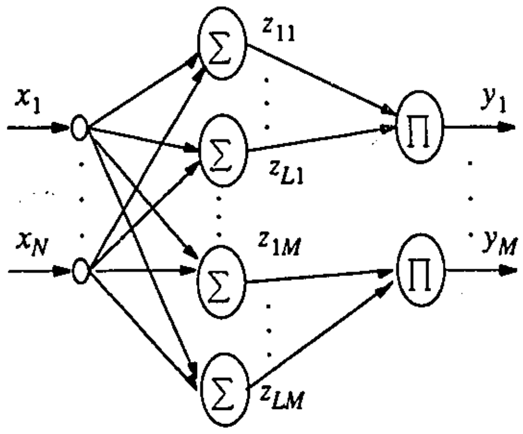


Fig. 3.5 Single layer pi–sigma networks

Fig. 3.5 shows the structure of a single layer pi–sigma network. Eq. (3.2.38) can also include *nonlinear activation functions* to avoid large amplitude outputs during training, which may lead to training divergence. The number of hidden neurons in a network is LM. Usually, we can make the number of weights to be learned, $[L(N+1)+1]M$, smaller than pM, where p is defined in eq. (3.2.5) for the same orders L, M, N.

Similarly, $f(x)$ can be represented in each subregion Ω_j by a polynomial with order L in pi–sigma format:

$$f(x) = \prod_{l=1}^{L} \left(\sum_{n=1}^{N} w_{ln}^{j} x_n + \theta_l^j \right) \equiv f_j(x) \qquad x \in \Omega_j \tag{3.2.39}$$

By using input receptive field selection functions (3.2.11), $f(x)$ can be expressed in the following localized representation:

$$f(x) = \sum_{j=1}^{J} s_j(x) \prod_{l=1}^{L} \left(\sum_{n=1}^{N} w_{ln}^{j} x_n + \theta_l^j \right) = \sum_{j=1}^{J} s_j(x) f_j(x) \tag{3.2.40}$$

Thus, localized pi–sigma networks can be constructed similarly.

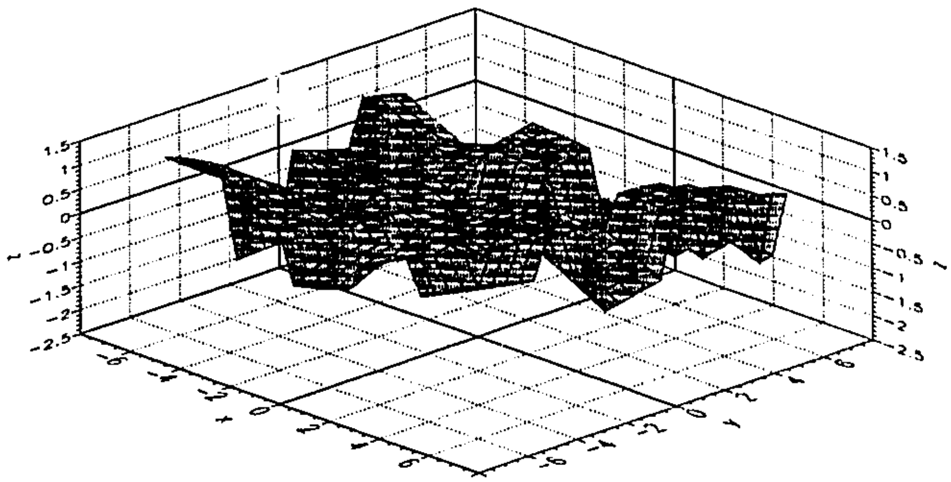
Compared with localized polynomial networks, the localized pi–sigma networks require much fewer weights to be learned for a problem with a large number of inputs. The cost is that the global convergence cannot be proved theoretically, although the locally convergent regions are fairly large for many problems if the UD–AOKF learning algorithm is used.

3.2.5 Simulation evaluation

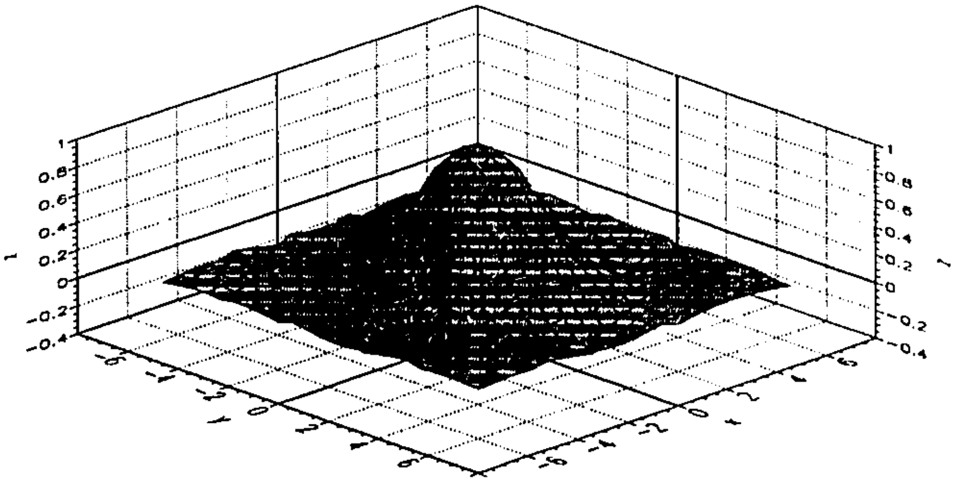
The performance of the localized polynomial neural network with CLI cells is tested by performing function approximation, because function approximation is one of the strictest applications of neural networks. The UD–AOKF algorithm is used as the learning algorithm to learn the following function:

$$z = \frac{\sin x \ \sin y}{xy} \qquad x,\ y \in [-2\pi,\ 2\pi] \qquad (3.2.41)$$

Let $L = 2, J = 25$. Then the number of hidden neurons in each sub–polynomial neural network is $p = 6$. Before training, the initial weights and thresholds are set randomly and the network output is quite different from the actual one (fig. 3.6 (a)). After 500 iterations of training



(a) before training



(b) after training

Fig. 3.6 Approximation of *z=sin(x) sin(y) /xy* using the new neural networks

using random data, the average error is 0.017; after 1000 iterations of training using random data, the average error is 0.01, and the result, shown in fig. 3.6 (b), is very close to the real function. The actual error mesh is shown in fig. 3.7. Similar results were achieved with dif-

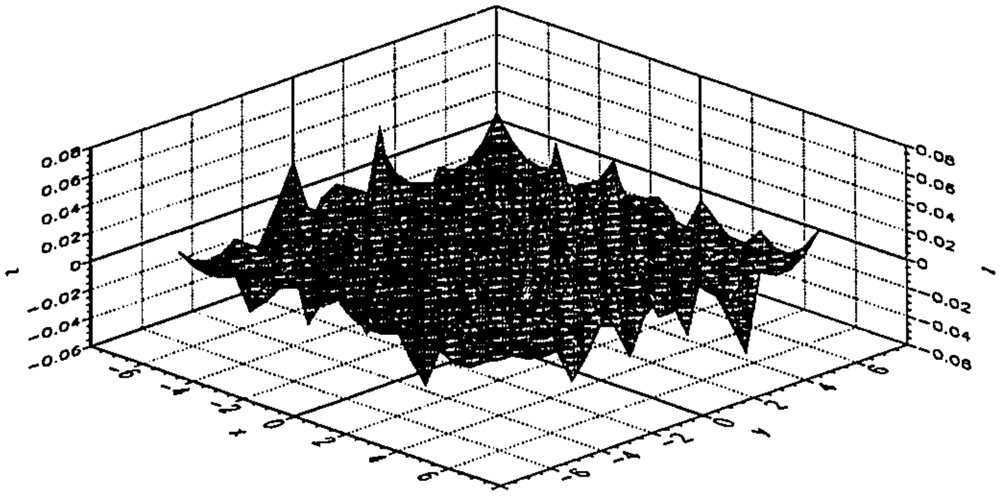


Fig. 3.7 Approximation error mesh of *z=sin(x) sin(y) /xy*

ferent initial values. This proves the fast and global convergence and insensitivity of the UD–AOKF algorithm to the initial values of the network weights, if J is properly determined. The result is also not sensitive to the location of the CLI cell centers as long as J is not too small. The same problem was solved using localized pi–sigma networks, and similar results were obtained.

If the same problem is solved using the global polynomial neural networks without CLI cells, the order of the network has to be larger than 7 to obtain a satisfactory approximation and the number of the hidden neurons has to be $p \geq 36$, which requires intensive computation, since the number of multiplications per iteration required to train the network is of the order of $O(p^2)$. Compared with the case of the localized polynomial neural network with CLI cells ($p = 6$), we can see that significant reduction in computation is achieved without losing the global optimality. The overall storage required by the localized networks is equal to $J \times [p_1 + O(p_1^2)] = 150 + 25 \times O(36)$, which is less than the overall storage, $p_2 + O(p_2^2) \geq 36 + O(36^2)$, required by the global networks.

More examples can be found in Liang and ElMaraghy (1993c). In section 3.4, the localized networks are also applied to nonlinear system modeling. The superior performance of localized networks is demonstrated again.

3.3 MULTISTEP LOCALIZED ADAPTIVE LEARNING OF RBF NETWORKS

Radial basis function (RBF) networks are becoming more popular as substitutes for multilayer perceptrons for function approximation and control applications due to their simplicity, faster convergence and well–developed mathematical background. There is a brief review of RBF networks, the existing problems and their applications in Liang and ElMaraghy (1993d). There are several radial basis functions, e.g., Gaussian, thin–plate–spline, and multi–quadratic, that can be used to construct RBF networks. Here, we only consider the Gaussian RBF networks.

In the following, a new learning algorithm is derived which can determine the Gaussian RBF centers and widths adaptively. Thus, off–line clustering of the RBF centers is not needed. The learning rate is determined by one–dimensional optimization algorithms. In addition, the multistep error index is introduced to speed up convergence and to remove the requirement of randomly feeding the training input data. The locality of Gaussian RBF networks is discussed and applied to reduce the training computation and lower the interference between different training data sets. It is also utilized to determine the number of RBF nodes for a given problem through adaptation (there was no on–line algorithm in the literature for such a purpose, to the best of this researcher's knowledge). The result is extended to cascaded RBF networks as well. The multistep error index is a general concept and can be applied to other classes of networks and other learning algorithms, for example, the conjugate gradient method and the variable metric method, to achieve the same benefit. The results in this section have been applied by several researchers in universities or research centers.

3.3.1 Adaptive learning algorithm with optimal learning rates

A Gaussian RBF network with N inputs, L RBF nodes, and M outputs is shown in fig. 3.8. The input–output relations of a Gaussian RBF network can be defined as follows:

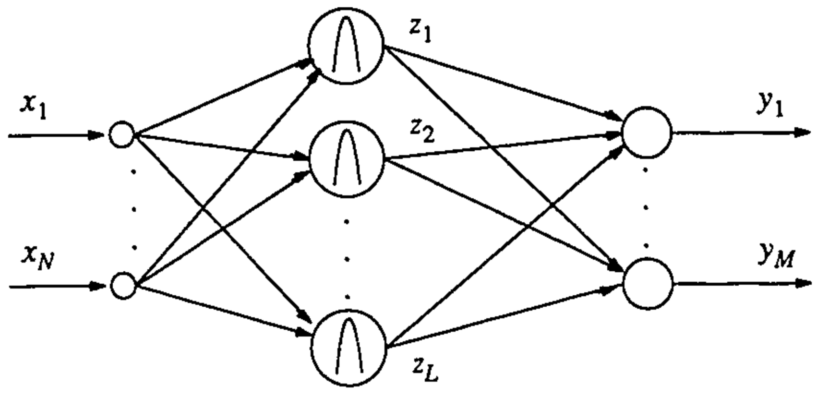


Fig. 3.8 Gaussian RBF networks

Output layer:

$$y_m = \sum_{l=1}^{L} w_{ml} z_l \qquad m = 1, \ldots, M \tag{3.3.1}$$

Hidden RBF layer:

$$z_l = \exp\{ - \sum_{n=1}^{N} \frac{(x_n - c_{ln})^2}{\sigma_{ln}^2} \} \qquad l = 1, \ldots, L \tag{3.3.2}$$

where x_n ($n = 1, \ldots, N$) is the n–th input variable; σ_{ln} is the standard deviation of the l–th RBF node for the n–th input; $[c_{l1} \ \ldots \ c_{lN}]^T$ defines the center vector of the l–th RBF node; z_l is the output of the l–th RBF node; y_m is the m–th output; and w_{ml} is the weight connecting the m–th output with the l–th RBF node.

Assume that there are unknown mappings between $x_1 \dots x_N$ and $y_1^d \dots y_M^d$, which are expressed as follows:

$$y_m = f_m(x_1, \dots, x_N) \qquad m = 1, \dots, M \qquad (3.3.3)$$

If there are enough training data $\{x_1(k) \dots x_N(k);\ y_1^d(k), \dots, y_M^d(k) \mid k = 1, \dots, K\}$ which can represent the relations between $x_1 \dots x_N$ and $y_1^d \dots y_M^d$, and L is unlimited, then the RBF network shown in fig. 3.8 can approximate the unknown mappings (3.3.3) with arbitrary precision (Moody and Darken, 1988).

The choices of the centers c_{ln} and the standard deviations σ_{ln} of the Gaussian functions are very important for achieving good approximation. There are two popular methods for their determination. Assume that $[x_1 \dots x_N]^T$ belongs to a compact subset X of R^N. The first method is to position the centers of the RBF functions evenly spaced out along all dimensions of the input space. The standard deviations are set to given constants. The method is simple, but suffers from the dimensionality. Moreover, it is difficult to determine the number of the RBF nodes which can guarantee the approximation precision. The second method is to use Kohonen's self–organized learning to cluster the training input data, and then position the centers of the RBF functions at the clustered centers of the self–organized network. The standard deviation is set to the shortest distance between the corresponding center and its neighboring centers. This method is very useful for classification problems. However, for function approximation problems, the input data are assumed to be evenly distributed over a given compact subset X. Therefore, the clustering result from self–organized learning is similar to that obtained from the subset division.

It has been observed that if the centers of the Gaussian functions are located at the extremum points and inflection points, and the standard deviations correspond to the absolute values of the derivatives or smoothness of the functions (3.3.3) around the centers, then a very high training precision approximation can be achieved with fewer RBF nodes and less training effort. Although these characteristics are difficult to acquire when the approximated functions are unknown, this observation tells us that evenly distributed centers may not be the best solution. In order to approximate the above characteristics and determine c_{ln} and σ_{ln} accordingly, we derive an adaptive learning algorithm for the RBF networks by optimizing the output errors with respect to w_{ml}, c_{ln} and σ_{ln}.

The instant output error index is defined as:

$$\varepsilon(k) = \frac{1}{2}\sum_{m=1}^{M} e_m^2(k) \qquad e_m(k) = y_m^d(k) - y_m(k) \tag{3.3.4}$$

Now the problem becomes determining a learning algorithm for w_{ml}, c_{ln} and σ_{ln} that minimizes $\varepsilon(k)$.

(1) The learning algorithm for w_{ml}

Using the gradient–descent method, we have:

$$w_{ml}(k+1) = w_{ml}(k) + \Delta w_{ml}(k) \tag{3.3.5}$$

where:

$$\Delta w_{ml}(k) = -\eta_1(k)\frac{\partial \varepsilon(k)}{\partial w_{ml}} \tag{3.3.6}$$

and

$$\frac{\partial \varepsilon(k)}{\partial w_{ml}} = e_m(k)\frac{\partial e_m(k)}{\partial y_m(k)}\frac{\partial y_m(k)}{\partial w_{ml}} = -e_m(k)z_l(k) \tag{3.3.7}$$

(2) The learning algorithm for c_{ln} and σ_{ln}

Similarly, for the RBF layer, we have

$$c_{\mathrm{ln}}(k+1) = c_{\mathrm{ln}}(k) + \Delta c_{\mathrm{ln}}(k) \tag{3.3.8}$$

$$\sigma_{ln}(k+1) = \sigma_{ln}(k) + \Delta\sigma_{ln}(k) \tag{3.3.9}$$

where

$$\Delta c_{\ln}(k) = -\eta_2(k)\frac{\partial \varepsilon(k)}{\partial c_{\ln}} \qquad \Delta \sigma_{\ln}(k) = -\eta_3(k)\frac{\partial \varepsilon(k)}{\partial \sigma_{\ln}} \tag{3.3.10}$$

and

$$\frac{\partial \varepsilon(k)}{\partial c_{\ln}} = \sum_{m=1}^{M} e_m(k)\frac{\partial e_m(k)}{\partial y_m(k)}\frac{\partial y_m(k)}{\partial z_l(k)}\frac{\partial z_l(k)}{\partial c_{\ln}}$$

$$= -2\sum_{m=1}^{M}\{e_m(k)w_{ml}(k)\}\ z_l(k)\frac{x_n(k) - c_{\ln}(k)}{\sigma_{\ln}(k)^2} \tag{3.3.11}$$

$$\frac{\partial \varepsilon(k)}{\partial \sigma_{\ln}} = \sum_{m=1}^{M} e_m(k)\frac{\partial e_m(k)}{\partial y_m(k)}\frac{\partial y_m(k)}{\partial z_l(k)}\frac{\partial z_l(k)}{\partial \sigma_{\ln}}$$

$$= -2\sum_{m=1}^{M}\{e_m(k)w_{ml}(k)\}\ z_l(k)\frac{[x_n(k) - c_{\ln}(k)]^2}{\sigma_{\ln}(k)^3} \tag{3.3.12}$$

Combining eqs. (3.3.5)–(3.3.12) gives the adaptive learning algorithm for RBF neural networks. Here, the momentum terms can be added to the learning algorithm, and the learning rates $\eta_i(k)$ (i=1, 2, 3) can be determined by a one dimensional optimization algorithm (e.g., Polak, 1971), or by some empirical formula.

3.3.2 Localized adaptive learning algorithm for Gaussian networks

Next, we explore the locality of the RBF functions. For simplicity, let us consider a simple Gaussian function:

$$z = \exp\{-\frac{(x-c)^2}{\sigma^2}\} \tag{3.3.13}$$

where $z, x, c, \sigma \in R^1$. It can be seen that

$$(1) \qquad \lim_{\sigma \to 0} \exp\{-\frac{(x-c)^2}{\sigma^2}\} = \delta(x-c) \tag{3.3.14}$$

where $\delta(x-c)$ is the Direc delta function. $\delta(0)$=1.

(2) $$\lim_{\sigma \to \infty} \exp\{ -\frac{(x-c)^2}{\sigma^2} \} = 1 \qquad (3.3.15)$$

(3) If $|x - c| > 3\sigma, \quad 0 < z < 1.23\%$ (3.3.16)

Property (1) shows that if some of the σ_{ln} (n=1, 2, . . . , N) are very small, the output of the l–th RBF function (3.3.2) is approximately zero for most inputs. This implies that if all of the σ_{ln}'s are too small, the RBF networks purely memorize the input data, and thus are of poor generalization ability. Property (2) indicates that if all of the σ_{ln} (n = 1, 2, . . . , N) are very large, the output of the l–th RBF function (3.3.2) is virtually equal to 1 for any input, which can be replaced by changing the thresholds of the output neurons. In this case, the RBF networks are of poor discrimination ability. To make RBF networks possess good generalization and discrimination ability, some upper and lower bounds on σ_{ln} should be set. Property (3) shows that the Gaussian RBF nodes are of local receptive fields, and respond to the inputs located in their local receptive fields only.

The weighted distance between an input vector $\mathbf{x} = [x_1 \ . \ . \ . \ x_N]^T$ and the l–th RBF center vector $\mathbf{c}_l = [c_{l1} \ . \ . \ . \ c_{lN}]^T$ is define as follows:

$$d(\mathbf{x}, \ \mathbf{c}_l) = \sqrt{\sum_{n=1}^{N} \frac{(x_n - c_{ln})^2}{\sigma_{ln}^2}} \qquad (3.3.17)$$

From property (3), we know that if $d(\mathbf{x}, \mathbf{c}_l) > 3$, the output of the l–th RBF function (3.3.2) is approximately zero. This reflects the locality of the Gaussian RBF nodes in RBF networks.

Given any input vector $\mathbf{x}$, if the output of the l–th RBF node, z_l, is greater than a given small constant δ, then the l–th RBF node is said to be active at the input excitation $\mathbf{x}$. δ can be chosen to be 2% or larger. Fig. 3.9 illustrates the locality property of RBF networks, where the solid lines and circles represent the active RBF nodes and connections.

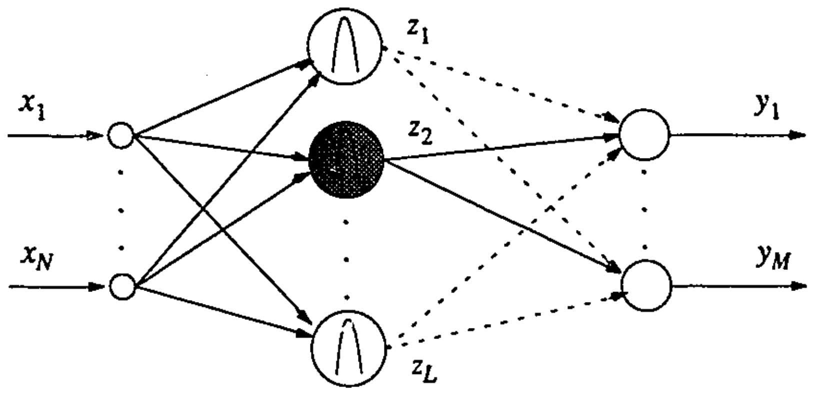


Fig. 3.9 Active RBF nodes and connections in localized adaptive RBF networks

For the l-th RBF node, when the input $\mathbf{x}$ satisfies $d(\mathbf{x},c_l) > 3$, the updating of w_{ml}, c_{ln} and σ_{ln} for corresponding l can hardly reduce the approximation errors of function (3.3.3). On the contrary, their adjustments may affect the learned input–output relations around the neighborhoods of other RBF center vectors. Also, a large amount of computation is required by this fruitless updating.

We present the following localized adaptive learning algorithm for RBF networks to synthesize the above analysis.

$$w_{ml}(k+1) = \begin{cases} w_{ml}(k) + \Delta w_{ml}(k) & \text{if the } l\text{-th RBF node is active} \\ w_{ml}(k) & \text{otherwise} \end{cases} \tag{3.3.18}$$

$$c_{ln}(k+1) = \begin{cases} c_{ln}(k) + \Delta c_{ln}(k) & \text{if the } l\text{-th RBF node is active} \\ c_{ln}(k) & \text{otherwise} \end{cases} \tag{3.3.19}$$

$$\sigma_{ln}(k+1) = \begin{cases} \sigma_{ln}(k) + \Delta \sigma_{ln}(k) & \text{if the } l\text{-th RBF node is active} \\ \sigma_{ln}(k) & \text{otherwise} \end{cases} \tag{3.3.20}$$

where the incremental terms in eqs. (3.3.18) through (3.3.20) are the same as those in eqs. (3.3.5), (3.3.8) and (3.3.9). The initial values of c_{ln} and σ_{ln} can be determined randomly, or by applying a priori information about the approximated functions. The learning rates

$\eta_i(k)$ ($i = 1, 2, 3$) are determined by a one–dimensional optimization search. The algorithm is not sensitive to inexact searches. It can be seen that the computation amount of the learning algorithm is mainly related to the approximation accuracy requirement, rather than the problem dimensions. This property makes the above new algorithm applicable to very large RBF networks.

In the aspect of weight updating, the localized learning algorithm is similar to CMAC networks (Albus, 1975), but the forward computation of localized RBF networks is still global and they possess much better generalization compared with CMAC networks.

Assume that the number of active RBF nodes for an input vector x is $L_a(\mathbf{x})$. The larger $L_a(\mathbf{x})$ is allowed to be, the more accurate the function approximation at x will be. Let the upper and lower boundaries on $L_a(\mathbf{x})$ for any x be $L_a^{\min}$ and $L_a^{\max}$, respectively. Then the number of the total RBF nodes, L, can be adjusted according to the following principles:

(a) If $L_a(\mathbf{x}) < L_a^{\min}$, then add $L_a^{\min} - L_a(\mathbf{x})$ RBF nodes to the network based on Heuristics 1, and let $L(k+1) = L(k) + L_a^{\min} - L_a(\mathbf{x})$, where $L(k)$ is the number of the total RBF nodes at iteration k.

(b) If $L_a(\mathbf{x}) > L_a^{\max}$, then delete $L_a(\mathbf{x}) - L_a^{\max}$ RBF nodes from the network based on Heuristics 2, and let $L(k+1) = L(k) + L_a^{\max} - L_a(\mathbf{x})$.

(c) If $L_a^{\min} \leq L_a(\mathbf{x}) \leq L_a^{\max}$, then $L(k+1) = L(k)$.

Heuristics 1: The centers of the added RBF nodes can be randomly distributed around vector x, or are simply set to x. Their standard deviations are set to the user–supplied default value. The weights connecting the new RBF nodes with the output nodes are set to small random numbers.

Heuristics 2: Among the active RBF nodes for input vector x, it is preferable to delete those whose standard deviations are very small.

If both the input and output data of the RBF networks are bounded, then the number of hidden RBF nodes, $L(k)$, determined from the above algorithm, will stabilize to a constant number, or a small range, depending on whether the given $L_a^{\min}$ and $L_a^{\max}$ are chosen properly or not. In either case, the determined RBF networks can approximate a given function with user–specified precision.

3.3.3 Multistep learning algorithms

The instant output error index (3.3.4) is a commonly used error index in neural network research. However, it is one of the major causes of inefficiency of all the back propagation–like learning algorithms. That the current output errors are minimized does not mean the global function approximation error is decreased. It is crucial to make sure that the already learned input–output relations, reflected by the current weight vector, are not affected by the continuing learning process. That is why small learning constants, momentum terms, and randomized input sequences are so important to the convergence of existing learning algorithms of neural networks. In formal optimization problems, the given function index totally determines the optimal parameters, but the instant output error index (3.3.4) does not. If we want to achieve a similar convergence property by applying the optimization strategy, we should use the following error index (3.3.21) instead, and also choose a better descent direction than the steepest descent, for example, the conjugate gradient. That is, E is related to all available training data and is a function of w_{ml}, c_{ln} and σ_{ln}.

$$E = \frac{1}{K}\sum_{k=1}^{K}\varepsilon(k) = \frac{1}{2K}\sum_{k=1}^{K}\sum_{m=1}^{M} e_m^2(k) \tag{3.3.21}$$

Therefore, to update w_{ml}, c_{ln} and σ_{ln}, we pass all the training data to the RBF network in order to calculate E and its gradients w.r.t. w_{ml}, c_{ln} and σ_{ln}. This is the so–called batch processing strategy. If there is little noise in the training data, fewer training data are required to obtain the same approximation precision compared with the ordinary backpropagation.

The batch processing strategy may make the learning processes very intensive in computation, thus is not practical for on–line applications such as system identification. By examining eq. (3.3.21), we observe that each training datum works as one constraint in defining an unknown function. If we have some a priori information about the smoothness of the unknown function, then fewer training data are needed to fully define the unknown function, which means that K in eq. (3.3.21) does not need to be the number of all the training data. For example, if we know that the unknown function is a straight line, then two training data can totally define the function. The reason why we need more training data is because the learning processes can not converge in one step using a constant learning rate, and the data may contain noise. This observation is further confirmed by the nonlinear sampling theory.

By making use of a priori information about the smoothness of the unknown function, we can change eq. (3.3.21) into the following multistep error index form:

$$E(k) = \frac{1}{\mu}\sum_{j=k-\mu}^{k} \varepsilon(j) = \frac{1}{2\mu}\sum_{j=k-\mu}^{k}\sum_{m=1}^{M} e_m^2(j) \tag{3.3.22}$$

where μ is the number of steps used to evaluate E at each time instant and is related to the smoothness of the approximated function or system.

Similarly, we can get

$$\frac{\partial E(k)}{\partial w_{ml}} = \frac{1}{\mu}\sum_{j=k-\mu}^{k} \frac{\partial \varepsilon(j)}{\partial w_{ml}} = -\frac{1}{\mu}\sum_{j=k-\mu}^{k} e_m(j) z_l(j) \tag{3.3.23}$$

$$\frac{\partial E(k)}{\partial c_{\text{ln}}} = \frac{1}{\mu}\sum_{j=k-\mu}^{k} \frac{\partial \varepsilon(j)}{\partial c_{\text{ln}}} = -\frac{2}{\mu}\sum_{j=k-\mu}^{k}\sum_{m=1}^{M} \{e_m(j) w_{ml}(j)\}\; z_l(j)\frac{x_n(j) - c_{\text{ln}}(j)}{\sigma_{\text{ln}}(j)^2} \tag{3.3.24}$$

$$\frac{\partial E(k)}{\partial \sigma_{\text{ln}}} = \frac{1}{\mu}\sum_{j=k-\mu}^{k} \frac{\partial \varepsilon(j)}{\partial \sigma_{\text{ln}}} = -\frac{2}{\mu}\sum_{j=k-\mu}^{k}\sum_{m=1}^{M} \{e_m(j) w_{ml}(j)\}\; z_l(j)\frac{[x_n(j) - c_{\text{ln}}(j)]^2}{\sigma_{\text{ln}}(j)^3} \tag{3.3.25}$$

Applying eqs. (3.3.23) through (3.3.25) to eqs. (3.3.18) through (3.3.20) provides the multistep localized adaptive learning algorithm for Gaussian RBF networks.

Compared with the existing learning algorithms for RBF networks, the new learning algorithm has the following advantages:

a. It is simple, and has faster convergence due to the multistep constraints, which make the approximated functions less uncertain, and localized learning, which eliminates much unnecessary computation.

b. It can determine adaptively the centers and standard deviations of the RBF nodes, which saves much tedious off–line work.

c. It can determine the number of the RBF nodes adaptively, for a given precision.

d. It converges even faster if the learning rates $\eta_i(k)$ ($i = 1, 2, 3$) are determined by a suitable one–dimensional optimization algorithm (e.g., Polak, 1975).

The multistep error index is a general concept and can also be applied to the other classes of networks and the other learning algorithms, for example, the conjugate gradient algorithm and the variable metric algorithm for the multilayer feedforward perceptrons, to achieve the same results.

3.3.4 Simulation evaluation

To compare the new multistep localized adaptive learning algorithm with the conventional one for RBF networks, $y = \sin x$ is approximated by one input, 20 RBF nodes and one output Gaussian RBF network for $x \in [-10,\ 10]$. Let $\mu = 7$, and $x(k+1) = x(k) + 0.5$ as the training input. μ is related to the sampling interval. 400 iterations are executed for the new learning algorithm. The result is shown in fig. 3.10, where the solid line represents the true function curve, and the dotted line represents the output of the RBF network. It can be seen that the approximation precision is high within the interval $[-10,\ 10]$ and not guaranteed outside $[-10,\ 10]$. This demonstrates the locality of RBF networks, which indicates that they possess good interpolation but bad extrapolation abilities, especially when the scale of the networks is limited. The computation effort is increased as μ increases. On the other hand, the conventional learning algorithm cannot converge for arbitrary numbers of iterations in this case, even with an RBF center and variance adaptation (without such adaptation, more RBF nodes are needed). Only when the input sequence $\{x(k)\}$ is randomized does the approximation converge, but after many more iterations. This proves the usefulness of the multistep error index. Also, the updating of all the weights causes more computation.

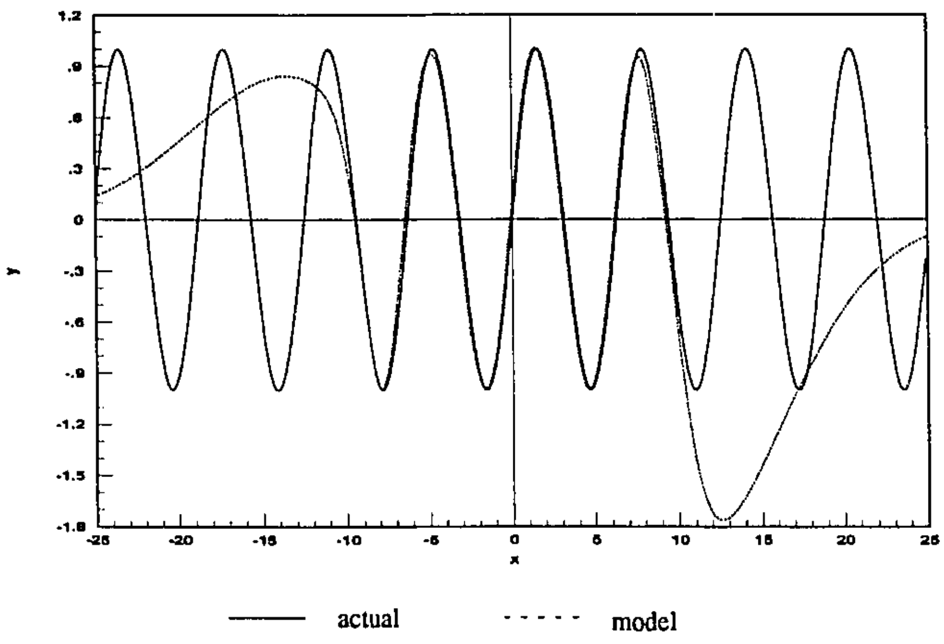


Fig. 3.10 Approximation of *sin x* using RBF networks with the new learning algorithm

The simulation results verify the correctness of the new learning algorithm and demonstrate its usefulness.

3.4 NEURAL NETWORK MODELS FOR NONLINEAR SYSTEMS

As indicated before, neural networks are universal approximators. By introducing dynamics into neural networks, they can also be applied to model nonlinear systems. According to the inputs and outputs of neural network models, the modeling problems can be categorized into (original) system identification (fig. 3.11) and inverse system identification (fig. 3.12).

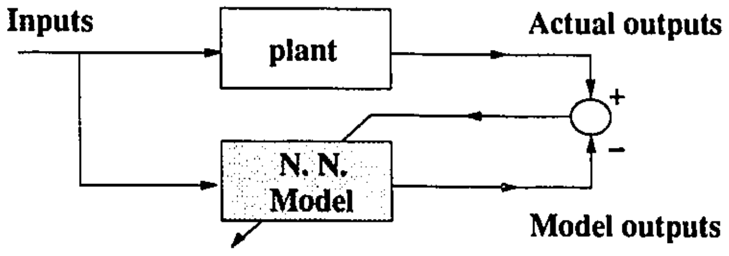


Fig. 3.11 The block diagram of system identification

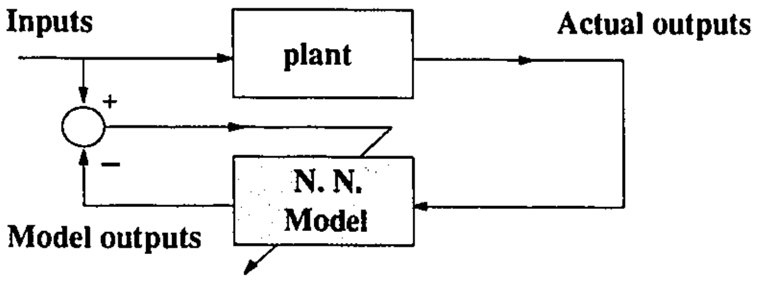


Fig. 3.12 The block diagram of inverse system identification

Most existing neural network models for nonlinear systems are based on recurrent network models and hybrid neural network models. The efficiency of neural network modeling depends on that of learning algorithms and the network structures. The commonly used learning algorithms for neural networks are: 1) gradient based learning algorithms, 2) extended Kalman filtering algorithms, 3) optimization based algorithms (CG and VM), 4) genetic learning algorithms, 5) statistical learning algorithms, and 6) fuzzy learning algorithms. These learning algorithms can be enhanced by pre–processing, post–processing and incorporating a priori information into the neural network models.

The theoretical problems of neural modeling include algorithm convergence, global optimality of the learning algorithms, determination of neural network structures for a given problem, optimal training data design, and model validity tests. Also, the identifiability issue should be addressed.

3.4.1 Four neural network models

Most existing dynamic systems are actually nonlinear. There are various representations of nonlinear systems. The four main distinct forms are summarized as follows:

(1) Continuous–time state–space form:

$$\dot{x} = f(x, u) + w(t) \tag{3.4.1}$$

$$y = h(x, u) + v(t) \tag{3.4.2}$$

where $x \in X \subset R^n$, $u \in U \subset R^m$ and $y \in Y \subset R^r$ are the system state vector, input vector and output vector, respectively. $f(.,.)$ and $h(.,.)$ are continuous real function vectors. $w(t)$ and $v(t)$ are input and output noises. Systems (3.4.1) and (3.4.2) are observable in the working region X and U. The system orders, n, m, and r, are known.

(2) Continuous–time input–output form:

$$y^{(q)} = g(y^{(q-1)}, \ . \ . \ . \ , \ \dot{y}, \ y; \ u^{(p)}, \ . \ . \ . \ , \ \dot{u}, \ u; e^{(q-1)}, \ . \ . \ . \ , \ \dot{e}, \ e) \tag{3.4.3}$$

where $u \in U \subset R^m$ and $y \in Y \subset R^r$ are the system input vector and output vector, respectively. $e \in R^r$ is the measurement noise. The system orders, q and p, are known.

(3) Discrete–time state–space form:

$$x(k + 1) = f(x(k), u(k)) + w(k) \tag{3.4.4}$$

$$y(k) = h(x(k), u(k)) + v(k) \tag{3.4.5}$$

where $x \in X \subset R^{\overline{n}}$, $u \in U \subset R^{\overline{m}}$ and $y \in Y \subset R^{\overline{r}}$ are the system state vector, input vector and output vector, respectively. $w(k)$ and $v(k)$ are input and output noises. Systems (3.4.4) and (3.4.5) are observable in the working region X and U. The system orders, $\bar{n}$, $\bar{m}$, and $\bar{r}$, are known.

(4) Discrete–time input–output form:

$$y(k) = \bar{g}(y(k-1),\ldots,y(k-\bar{q});u(k),\ldots,u(k-\bar{p});e(k-1),\ldots,e(k-\bar{q})) + e(k) \qquad (3.4.6)$$

where $u \in U \subset R^m$ and $y \in Y \subset R^r$ are the system input vector and output vector, respectively. $e(k)$ is the measurement noise. The system orders, $\bar{q}$ and $\bar{p}$, are known.

For the continuous–time state–space form described by eqs. (3.4.1) and (3.4.2), $f(x,u)$ and $h(x,u)$ can be approximated by two cascaded neural networks as shown in fig. 3.13.

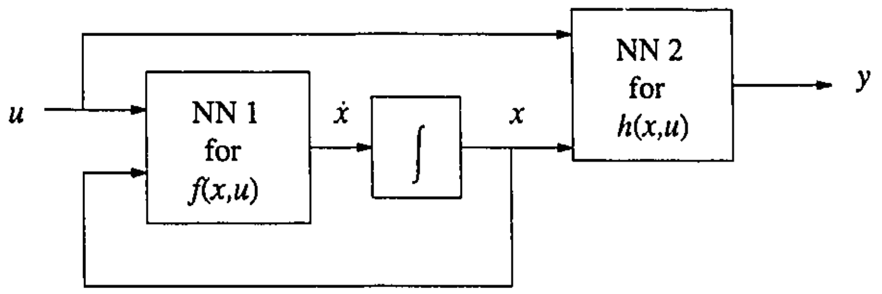


Fig. 3.13 Cascaded RBF networks for system model (3.4.1) and (3.4.2)

The overall input vectors to neural networks 1 and 2 are:

$$x^1 = x^2 = [x^T \; u^T]^T \qquad (3.4.7)$$

For instance, given sampling data of an unknown system, we can use the multistep localized adaptive learning algorithm of eqs. (3.3.18)–(3.3.20) to train the network shown in fig. 3.13.

For the discrete–time state–space form described by eqs. (3.4.4) and (3.4.5), if the integrators in fig. 3.13 are replaced by one–step time delays, then the identification procedure is similar to the continuous–time case.

For the continuous–time input–output form described by eq. (3.4.3), the systems are not directly identifiable if the high order derivatives of inputs and outputs are not available. Thus, we introduce the following filtered variables to circumvent such difficulty:

$$y_f(t) = y(t)/F_y(D) \qquad u_f(t) = u(t)/F_u(D) \qquad e_f(t) = e(t)/F_e(D) \tag{3.4.8}$$

where

$$F_y(D) = D^{n_y} + f_{y1}D^{n_y-1} + \ . \ . \ . \ + f_{n_y-1}D + f_{n_y} \tag{3.4.9}$$

$$F_u(D) = D^{n_u} + f_{u1}D^{n_u-1} + \ . \ . \ . \ + f_{n_u-1}D + f_{n_u} \tag{3.4.10}$$

$$F_e(D) = D^{n_e} + f_{e1}D^{n_e-1} + \ . \ . \ . \ + f_{n_e-1}D + f_{n_e} \tag{3.4.11}$$

where $D = \dfrac{d}{dt}$ is the differentiation operator; $n_y \geq q$, $n_u \geq p$, $n_e \geq q - 1$, and eqs. (3.4.9) – (3.4.11) are stable polynomials. Then the model (3.4.3) can be replaced by eq. (3.4.12).

$$y_f^{(q)} = g_f(y_f^{(q-1)}, \ . \ . \ . \ , \ \dot{y}_f, \ y_f; \ u_f^{(p)}, \ . \ . \ . \ , \ \dot{u}_f, \ u_f; e_f^{(q-1)}, \ . \ . \ . \ , \ \dot{e}_f, \ e_f) \tag{3.4.12}$$

After $g_f(\ \cdot \ )$ is identified, we can derive the $g(\ \cdot \ )$ in eq. (3.4.3) using the following relation:

$$\begin{aligned} y^{(q)} &= g(y^{(q-1)}, \ . \ . \ . \ , \ \dot{y}, \ y; \ u^{(p)}, \ . \ . \ . \ , \ \dot{u}, \ u; e^{(q-1)}, \ . \ . \ . \ , \ \dot{e}, \ e) \\ &= F_y(D)g_f(y^{(q-1)}/F_y(D), \ . \ . \ . \ , \ y/F_y(D); \ u^{(p)}/F_u(D), \ . \ . \ . \ , \ u/F_u(D); \\ &\qquad e^{(q-1)}/F_e(D), \ . \ . \ . \ , \ e/F_e(D)) \end{aligned} \tag{3.4.13}$$

For the discrete–time input–output form described by eq. (3.4.6), we assume that $\overline{q}$ and $\overline{p}$ in eq. (3.4.6) are known. Then only one simple neural network can approximate the system dynamics with the input vector to the network as:

$$x(k) = [\ y(k-1)^T, \ . \ . \ . \ , \ y(k-\overline{q})^T; \ u(k)^T, \ . \ . \ . \ , \ u(k-\overline{p})^T \ ]^T \tag{3.4.14}$$

and $N = r\overline{q} + m(\overline{p} + 1)$. If there is strong measurement noise, the input vector (3.4.14) has to be extended to include the prediction errors $\varepsilon(k)$, in order to obtain unbiased models

(Chen, Billings, and Grant, 1992).

$$x(k) = [y(k-1)^T, \ldots, y(k-\overline{q})^T; u(k)^T, \ldots, u(k-\overline{p})^T; \varepsilon(k-1)^T, \ldots, \varepsilon(k-\overline{q})^T]^T \tag{3.4.15}$$

In eq. (3.4.1), if all the state variables are measurable, then the nonlinear state equation (3.4.1) can be identified using single–hidden layer neural networks with a global convergent learning algorithm. First, eq. (3.4.1) is rewritten ($w(t)$ is neglected) as:

$$\dot{x} = f(x,u) = -\Lambda x + g(x,u) \tag{3.4.16}$$

where $\Lambda = diag\{\lambda_1, \ldots, \lambda_n\}$ is a positive definite matrix, and

$$g(x,u) = f(x,u) + \Lambda x \tag{3.4.17}$$

where $g(x,u)$ can be represented and learned using neural networks.

To guarantee the global learning convergence, we use single–hidden layer neural networks with linearly parameterizable output weights. Assume that the inputs to these neural networks consist of vectors x and u, the output vector of the hidden layer is denoted by $\Phi(x,u) \in R^p$, and $\Phi(x,u)$ is pre–determined and known. The output weight matrix is denoted by $W \in R^{p \times n}$. Then $g(x,u)$ can be approximated as:

$$g(x,u) = W^T \Phi(x,u) + d(t) \tag{3.4.18}$$

where $d(t)$ is the neural network representation error vector, due to finite number inclusion of neurons in the networks. $|d_i(t)| \leq d_0$ (i=1,2, . . ., n), where d_0 is a small positive constant and is reducible by proper network structure design. Let $\hat{x}$ stand for the estimation of the state vector x, and $\hat{W}$ stand for the estimation of W. Then the neural network model for the nonlinear system (3.4.1) is:

$$\dot{\hat{x}} = -\Lambda \hat{x} + \hat{W}^T \Phi(x,u) \tag{3.4.19}$$

Define $e_x = \hat{x} - x$, $\tilde{W} = \hat{W} - W$, and

$$e_\Delta = [e_{\Delta 1} \cdots e_{\Delta n}]^T \quad with \quad e_{\Delta i} = e_{xi} - \Delta_i sat(\frac{e_{xi}}{\Delta_i}) \tag{3.4.20}$$

where $sat(.)$ is the saturation function; $\Delta_i = d_0/\lambda_i$. e_Δ is continuous. Synthesizing eqs. (3.4.16), (3.4.18) and (3.4.19) gives:

$$\dot{e}_x = -\Lambda e_x + \tilde{W}^T \Phi(x,u) - d(t) \tag{3.4.21}$$

Let the learning algorithm of the neural network be

$$\dot{\hat{W}} = -\gamma \Phi(x,u) e_\Delta^T \tag{3.4.22}$$

where γ is the learning rate. Then e_Δ, the state estimation error vector with dead–zone Δ, converges to zero asymptotically. If $\Phi(x,u)$ satisfies the persistent excitation (PE) condition, then the neural network weights also converge to their optimal values ($\tilde{W} \to 0$).

This assertation is proved as follows. Let V be the Lyapunov function candidate:

$$V = \frac{1}{2}[e_\Delta^T e_\Delta + \gamma^{-1} tr(\tilde{W}^T \tilde{W})] \tag{3.4.23}$$

where $tr(.)$ is the matrix trace function. Differentiating V gives:

$$\dot{V} = e_\Delta^T \dot{e}_\Delta + \gamma^{-1} tr(\tilde{W}^T \dot{\tilde{W}}) = e_\Delta^T \dot{e}_x + \gamma^{-1} tr(\tilde{W}^T \dot{\hat{W}})$$

$$= e_\Delta^T[-\Lambda e_x + \tilde{W}^T \Phi(x,u) - d(t)] - tr(\tilde{W}^T \Phi(x,u) e_\Delta^T)$$

Notice that:

$$e_\Delta^T \tilde{W}^T \Phi(x,u) = tr[\tilde{W}^T \Phi(x,u) e_\Delta^T]$$

$$e_\Delta^T[-\Lambda e_x - d(t)] = \sum_{i=1}^{n} e_{\Delta i}[-\lambda_i(e_{\Delta i} + \Delta_i sat(e_{xi}/\Delta_i)) - d_i(t)]$$

$$= -e_\Delta^T \Lambda e_\Delta + \sum_{i=1}^{n}[-\lambda_i \Delta_i |e_{\Delta i}| - e_{\Delta i} d_i(t)]$$

$$\leq -e_\Delta^T \Lambda e_\Delta + \sum_{i=1}^{n}[-d_0 |e_{\Delta i}| + |e_{\Delta i}||d_i(t)|] \leq -e_\Delta^T \Lambda e_\Delta$$

$$\therefore \quad \dot{V} \leq -e_{\Delta}^{T} \Lambda e_{\Delta}$$

Since $\dot{e}_{\Delta}$ is bounded and continuous, we conclude that e_{Δ} converges to zero asymptotically.

In addition, if $\Phi(x, u)$ satisfies the PE condition, it can be proved using eq. (3.4.22) that the neural network weights also converge to their optimal values ($\tilde{W} \rightarrow 0$).

The identified neural network model (3.4.19) can be used for either prediction or control applications. If system (3.4.1) is unstable, then an outer–loop stabilizer is required to make the modeling possible.

In summary, neural networks can be applied to solve the nonlinear system identification problem, if the system order is known. In the following, we discuss some issues related to neural network modeling.

3.4.2 Generalization ability of neural networks

There is a general model for linear systems, because the structure of the linear system family is known. However, there is not a general model for nonlinear systems with a known unique structure. There are so many families in nonlinear systems that it is impossible to choose a correct system family in which only lumped parameters are unknown, without intensive deduction based on the underlying scientific laws. The model derived from the underlying scientific laws governing a nonlinear system is called the first–principle model. Because most actual dynamic systems are complex and affected by many unknown factors, it is difficult to build a first–principle model for many dynamic systems. Therefore, most nonlinear system identification algorithms are derived based on non–first–principle models. Usually, the first–principle models are minimally parameterized, and non–first–principle models are over–parameterized. Neural network based models for nonlinear systems are one type of non–first–principle models.

From the feature of machine learning (Kodrtoff and Michalski, 1990), we conclude that non–first–principle models are valid in the training set and the set that is represented or implied by the training set, but are uncertain beyond them. In other words, these models can interpolate well, but extrapolate with uncertainty, and badly in most cases. Neural network models are local. This reflects both the power and weakness of the non–first–principle models. Fig. 3.14 illustrates this characteristic, where X is the definition set of true mapping f; X_T and Y_T are input and output training sets; $\hat{f}$ is the neural network approximation of f. Then,

$$Y = \{ y \mid y = f(x),\ x \in X \} \qquad \hat{Y} = \{ y \mid y = \hat{f}(x),\ x \in X \} \tag{3.4.24}$$

$$f(X_T) = \hat{f}(X_T) = Y_T \tag{3.4.25}$$

Usually, $\hat{Y} \neq Y$.

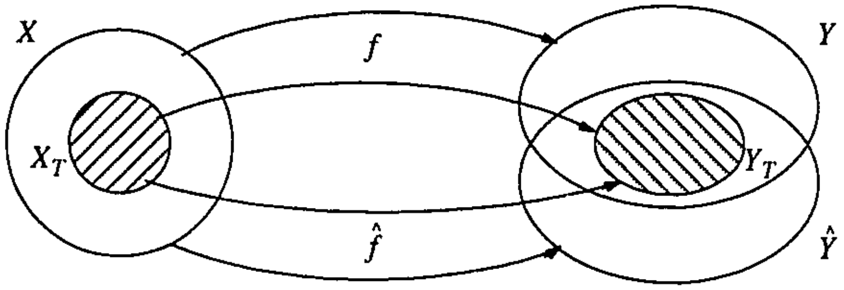


Fig. 3.14 Mapping relations between training sets and workable sets

Therefore, to make $\hat{Y} \doteq Y$, the training sets X_T and Y_T should be representative of the approximated functions or systems, and the working region X and U should be admissible. For sequential learning, the identified systems should also be excited persistently (Ljung and Soderstrom, 1983) to ensure the learning convergence and uniqueness. Otherwise, a neural network cannot approximate well on the working region X and U, even with an infinite number of neurons. Representativeness and persistent excitation are two necessary conditions to be satisfied for any nonlinear function approximation or nonlinear system identification problems using neural networks. This is different from linear cases, in which rep-resentativeness is not needed since the linear models are global.

Although many papers claim that neural networks are robust to input noise, no theoretical proof is available. The claimed robustness could be due to the sign function–like activation functions, the redundancy of the networks, and the average effect of massive training data. For RBF networks, replacing x_n with $x_n + v_n$ in eq. (3.3.2), where v_n is the input noise, gives

$$\hat{z}_l = \exp\{ -\sum_{n=1}^{N} \frac{(x_n + v_n - c_{\mathrm{ln}})^2}{\sigma_{\mathrm{ln}}^2} \} = a_l z_l \qquad (3.4.26)$$

where

$$a_l = \exp\{ -\sum_{n=1}^{N} \frac{v_n^2 + v_n(x_n - c_{\mathrm{ln}})}{\sigma_{\mathrm{ln}}^2} \} \qquad a_l > 0 \qquad (3.4.27)$$

Therefore, if no sign function–like activation function is introduced to the output layer of RBF networks, strong input noise does affect the approximation precision of RBF networks.

One effective way to attenuate the input noises to the neural networks is to introduce input signal filters for pre–processing, as shown in fig. 3.15, especially when some a priori information about the noise is available. Also, it can be seen from eq. (3.4.27) that increasing the overlapping (determined by σ_{ln}) of the receptive fields of the RBF functions can reduce the affects of the input noises. This means that for each input vector **x**, there are more active RBF nodes and the contribution or weight of each RBF node is lowered.

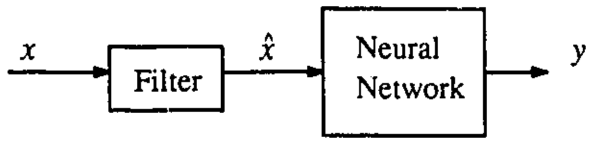


Fig. 3.15 Introduction of filters to neural networks

3.4.3 Simulation tests

The first example is modeling an input–output model of a discrete–time nonlinear system using Gaussian RBF networks and the multistep localized adaptive learning algorithm (3.3.18) through (3.3.20). For the following discrete–time nonlinear system,

$$y(k) = 0.8 * y(k-1) + \frac{u(k)}{1 + y(k-1)^2} \tag{3.4.28}$$

15 hidden RBF-nodes are used, and the multistep is chosen as $\mu = 8$. After 100 iterations of learning, the system (3.4.28) is identified using the training input $u(k) = sin(kT)$ and $T = 0.1$ *sec*. The test signal is $u(k) = sin(0.5kT) + 0.1sin(4*kT)$, and the test result is depicted in fig. 3.16. Here, the output time response of the RBF network model (dotted line) is approximately the same as the true output time response (solid line). The learning process converges very fast.

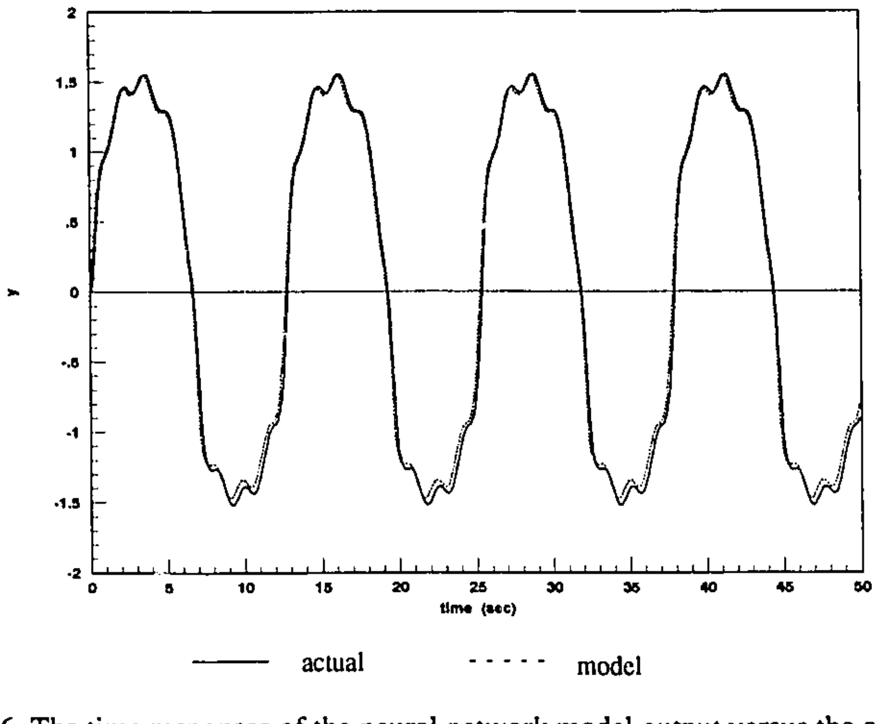


Fig. 3.16 The time responses of the neural network model output versus the actual one

The second example is modeling a state–space nonlinear system using localized polynomial networks. Identification speed is very important to the on–line system identification problems. At present, most training algorithms for neural networks are not fast enough for practical system identification applications. Here, a localized polynomial neural network with CLI cells is applied to identify the following discrete–time nonlinear system (Ungar and Narendra, 1992) in state space form:

$$x_1(k+1) = \left(\frac{x_1(k)}{1 + x_1^2(k)} + 1 \right) \sin\{x_2(k)\} \tag{3.4.29}$$

$$x_2(k+1) = x_2(k)\cos\{x_2(k)\} + x_1(k)\exp\left\{ - \frac{x_1^2(k) + x_2^2(k)}{8} \right\} + \frac{u^3(k)}{1 + u^2(k) + 0.5\cos\{x_1(k) + x_2(k)\}} \tag{3.4.30}$$

$$y(k) = \frac{x_1(k)}{1 + 0.5\sin\{x_2(k)\}} + \frac{x_2(k)}{1 + 0.5\sin\{x_1(k)\}} \tag{3.4.31}$$

where $x_1(k)$ and $x_2(k)$ are the state variables; $u(k)$ is the input variable; and $y(k)$ is the output variable. Now the problem is to make the following input–output model (3.4.6):

$$\hat{y}(k+1) = N[\ y(k),\ y(k-1),\ u(k),\ u(k-1)\ ] \tag{3.4.32}$$

represent the input–output relation of the actual system. Here, function $N[\ .\ ]$ is realized by a localized polynomial neural network with CLI cells, which has 4 inputs and one output.

Let the order of each sub–network be $L = 2$, and the number of the CLI cells be $J = 50$. The centers of the CLI cells are randomly distributed in the interval $[-2.8, 2.8]$. The number of hidden neurons in each sub–polynomial neural network is $p = 15$, which corresponds to modest computation at each iteration. Before training, the initial weights and thresholds are set randomly between –1 and 1. The training input is:

$$u(k) = \sin\left\{\frac{2\pi k}{10}\right\} + \sin\left\{\frac{2\pi k}{25}\right\} \tag{3.4.33}$$

Fig. 3.17 shows the training process of the neural network output compared with the

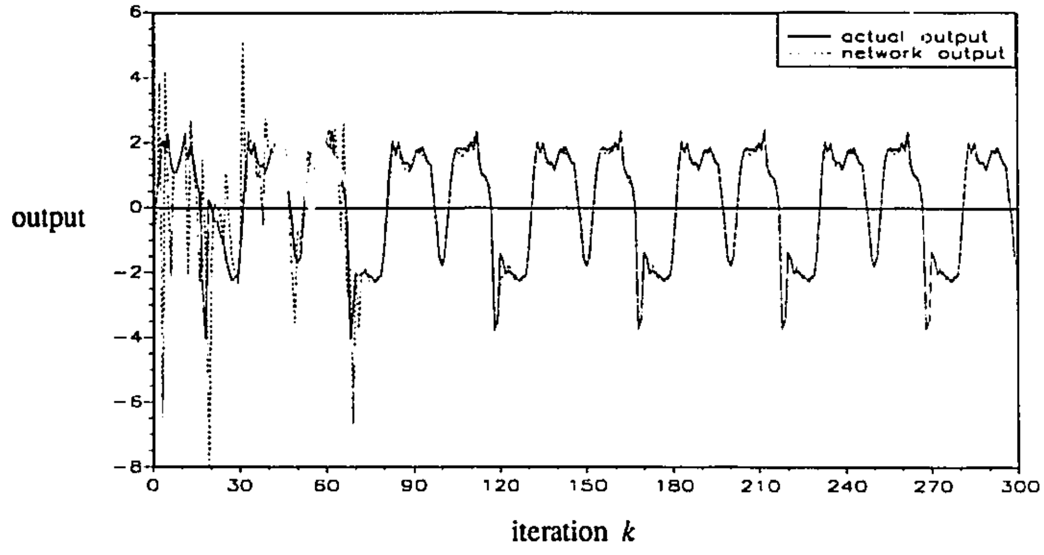


Fig. 3.17 Comparison of the actual and the network output during training

actual system output under the same training input (3.4.33). It can be seen that after 90 iterations of training, the neural network output is able to approximate the actual system output. In other words, after 90 iterations of training, the network model has learned the input–output relation of the actual system under the training input. Fig. 3.18 clearly shows that the

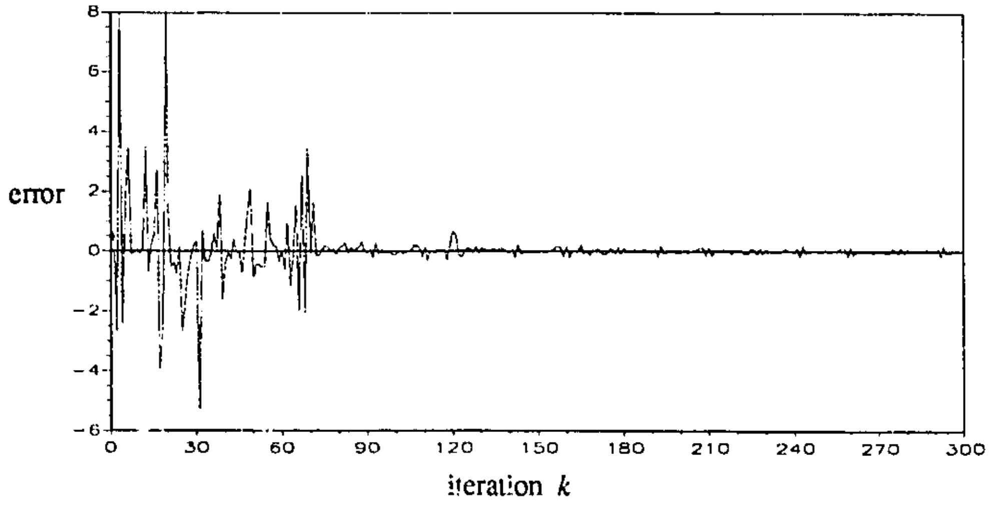


Fig. 3.18 Fast convergent training error curve vs. iteration (J=50)

training error converges and remains small and stable in spite of continuing training. Such a property is very useful for on–line system identification and output prediction, and self–tune adaptive control. This as well verifies the fast and global convergence of the UD–AOKF algorithm applied to the localized polynomial networks with CLI cells.

If system (3.4.29) – (3.4.31) is identified using a 4–20–10–1 multilayer feedforward perceptron trained by the backpropagation algorithm, then much more training iterations are needed to obtain similar precision. If the global polynomial neural network is adopted, intensive computation is unavoidable.

Just as in the function approximation case shown in section 3.2, the identification results are not sensitive to the number of the CLI cells as long as J is larger than 10. For example, fig. 3.19 shows the curve of training error vs. iteration when J=10, which is also fast

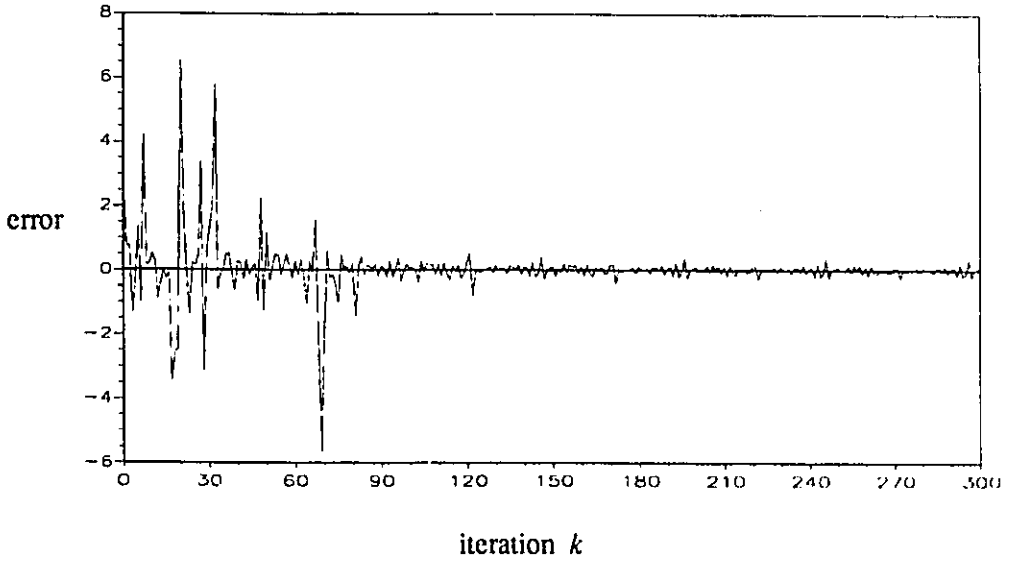


Fig. 3.19 Convergent training error curve vs. iteration (J=10)

convergent. The more there are CLI cells, the better the generalization of the trained network is. For example, for J=50, the network is trained by random inputs uniformly distributed in [–3, 3] for 500 iterations and then tested by input (3.4.33). A good prediction result was achieved. Certainly, the training inputs should be persistent excitation and represent the system characteristics.

3.5 SUMMARY

This chapter presents the localized neural network concept, the multistep localized adaptive algorithm for Gaussian RBF networks and the adaptive extended Kalman filtering algorithm for localized polynomial networks. For both localized polynomial networks and Gaussian RBF networks, on–line heuristic algorithms are given to determine the network structure adaptively. The localization concept can be extended to any neural networks. The multistep error index can also be extended to other learning algorithms to speed up the learning convergence. A brief discussion about the neural network based modeling is also provided.

The main problems in neural networks from an application point of view are the determination of the network size for a given problem and the global learning convergency. Some solutions to these problems for Gaussian RBF networks and polynomial networks are presented in this chapter. For other neural networks, there are some off–line learning algorithms for structure determination (constructive or destructive methods). Global convergence is difficult to obtain for many neural networks, due to their nonlinear parameterization and the difficulty in satisfying the representative persistent excitation condition. Since neural networks are not first–principle models, over–parameterization is one of their characteristics. There is redundancy in the weights, which produces multiple optimal solutions for a given problem. For neural networks with nonlinear parameterization, choosing good initial values of the weights and introducing some "shaking" mechanisms help the learning convergence. One feasible approximation of the representative persistent excitation is using random training data that cover the whole working region. Proper scaling of training data is another effective way of improving the speed of convergence.

The practical implementation of neural networks is another concern. Neural network chips are still being tested. Large scale neural networks are not suitable for some real–time applications when implemented using single digital processors.

It can be seen from the following chapters that linear parameterization of neural network weights is important to the global stability proofs of neurocontrol schemes. Liang and ElMaraghy (1994b) indicated that single–hidden layer sigmoidal neural networks can be formulated into linearly parameterizable networks, if the input–to–hidden weights are determined beforehand using approximation theory. The conclusion can be extended to single–hidden layer neural networks with any feasible activation functions.

CHAPTER 4

BACKPROP–BASED NEUROCONTROL SCHEMES

4.1 INTRODUCTION

Backprop–based neurocontrol systems refer to the systems whose neurocontrollers are updated by the back propagation principle, whether the learning algorithm is the delta–rule, or the conjugate gradient method. The typical structures of backprop–based neurocontrol systems are shown in figs. 2.2, 2.3 and 2.4, which represent the inverse (adaptive) neurocontrol systems, the indirect (adaptive) neurocontrol systems and direct (adaptive) neurocontrol systems respectively. They can be developed by on–line learning, off–line learning, or hybrid learning. The neurocontrol systems are called adaptive if they are developed by on–line learning. By introducing an on–line copying mechanism, the common inverse neurocontrol systems can also be made adaptive as shown in fig. 4.1.

Backprop–based neurocontrol differs from stability–based neurocontrol, reinforcement–based neurocontrol, and adaptive critic–based neurocontrol in the construction of learning algorithms.

The backprop–based neurocontrol schemes aim at solving general nonlinear system control problems, in both continuous–time domain and discrete–time domain. Recently, it was found (Gu, 1990, Liang and ElMaraghy, 1993a) that they could only be applied to invert-

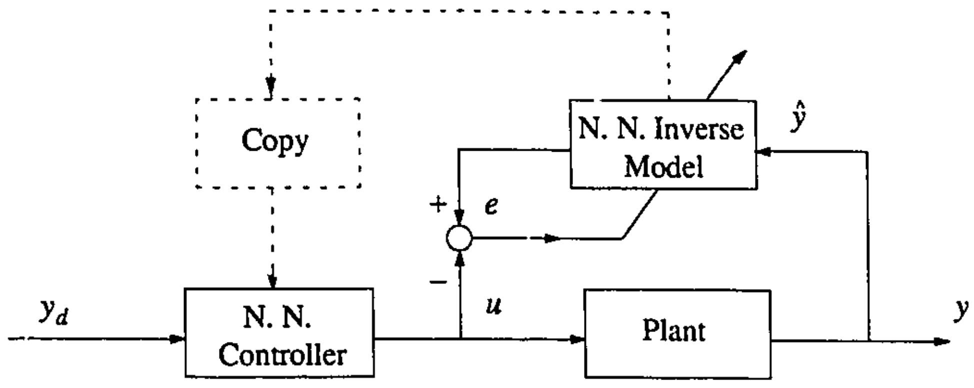


Fig. 4.1 The block diagram of the adaptive inverse system neurocontrol

ible minimum–phase nonlinear systems with known system orders. Controllability is another prerequisite.

The advantages of the backprop–based neurocontrol approach are that it is conceptually simple and that many different schemes have been tested for different applications. Little background in control theory is required to design such a neurocontroller. The disadvantage of this approach is the slow and local convergence of the learning processes, which makes the development of backprop–based neurocontroller time–consuming.

In this chapter, some theoretical issues regarding the backprop–based indirect and direct adaptive neurocontrol schemes are clarified. The existence conditions and the structure determination of backprop–based neurocontrollers are presented. New extensions are also provided. Section 4.2 discusses the backprop–based indirect adaptive neurocontrol schemes. The input–output representation of sampled–data nonlinear systems is established. It converts the system invertibility problem into the existence problem of implicit functions. Due to the usage of localized polynomial neural networks, fast learning convergence is achieved. Section 4.3 presents some new extensions of the backprop–based direct adaptive neurocontrol schemes with better performance. By introducing optimization algorithms without using derivatives, the output tracking errors can be back–propagated to the neurocontrollers without using neural network emulators. Section 4.4 gives summary and discussions.

4.2 INDIRECT ADAPTIVE NEUROCONTROL

Backprop–based indirect adaptive neurocontrol schemes are one of the widely–studied neurocontrol schemes (Miller, Sutton, and Werbos, 1990, and Hunt et al., 1992). Liang and ElMaraghy (1993a) gave a brief review of the related work. Fig. 4.2 shows the block

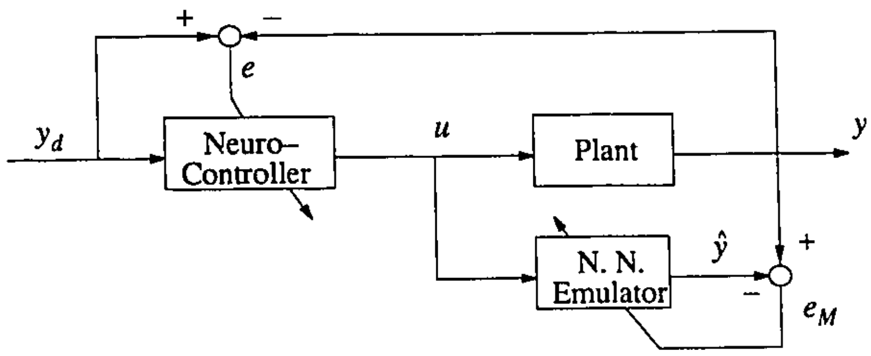


Fig. 4.2 The schematic diagram of the indirect adaptive neurocontrol systems

diagram of an indirect adaptive neurocontrol system. It consists of a neural network emulator modeling the controlled plant and a feedforward neural network as the controller. Its basic working principle is to use the modeling error e_M to tune the neural network emulator and back–propagate the tracking error e through the neural network emulator to tune the neurocontroller, in order to make both e_M and e tend to zero.

From the above principle, we know that the neurocontrollers aim at maintaining $y(t) = y_d(t)$ all the time. This can only happen if the equivalent dynamics of the neurocontroller and the plant is identity, or the plant dynamics is completely canceled out by the neurocontroller. Therefore, the neurocontroller must be equal to the inverse of the plant dynamics. This requirement is unrealizable sometimes, especially for systems with high relative degrees. Therefore, the backprop–based model reference indirect adaptive neurocontrol scheme as shown in fig. 4.3 is proposed. In this case, the equivalent dynamic of the neuro-

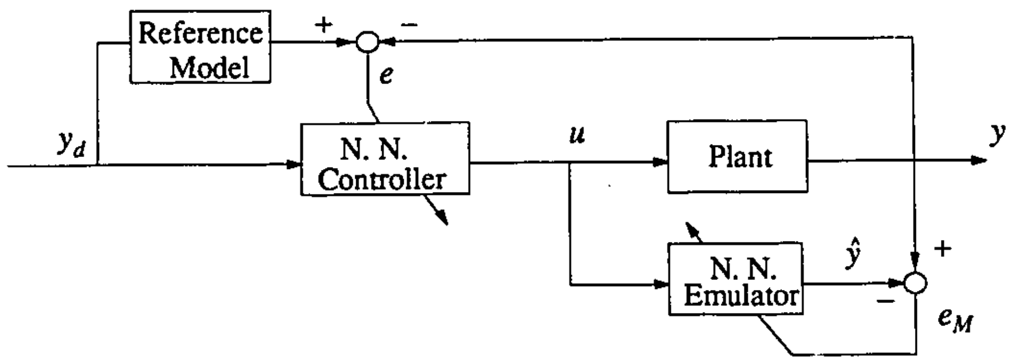


Fig. 4.3 The model reference indirect adaptive neurocontrol systems

controller and the plant is equal to the reference model, instead of identity. Thus, simpler neurocontrollers are required.

Most of the existing backprop–based indirect neurocontrol schemes are only feasible for off–line training. Fewer papers investigated the existence conditions for the indirect neurocontroller, and an efficient structure with defined input–output relationships. Moreover, the slow convergence of the widely–used backpropagation algorithm confined the efficiency of the indirect adaptive neurocontrollers. This section first presents the input–output representation of the sampled–data nonlinear systems and investigates the invertibility of continuous–time, discrete–time and sampled data NLS. Then the localized polynomial networks with the competitive lateral inhibitory (CLI) cells presented in section 3.2 are used to realize the neurocontrollers. Due to the localized networks, fast learning and on–line adaptation of both neural network models and neurocontrollers are possible. Therefore, the enhanced scheme possesses better adaptation ability to system changes. The ideas of combining a neurocontroller with a fuzzy controller and/or a conventional controller are also introduced.

4.2.1 Existence of indirect adaptive neurocontrollers

(1) Discretization and invertibility

Consider the continuous–time nonlinear system:

$$\dot{x} = f(x, u) \tag{4.2.1}$$

$$y = h(x, u) \tag{4.2.2}$$

where $x \in R^{n_s}$, $u \in R^{m_s}$ and $y \in R^{r_s}$ are the system state vector, input vector and output vector, respectively. $f(.,.)$ and $h(.,.)$ are unknown. The system (4.2.1) and (4.2.2) satisfies the following assumptions:

(1) it is controllable and observable in the working region Ω of x and u.

(2) $f(.,.)$ and $h(.,.)$ are continuously differentiable.

(3) the system orders, n_s, m_s, and r_s, are known.

The control problem for this unknown system is to design a controller that can make the system outputs track a given desired trajectory or remain at the zero state. The former is called the output tracking control problem. The latter is called the regulation problem, which can be regarded as a special case of the output tracking control problem.

To design an inverse controller, the system must be invertible. Gu (1990) presented a theorem that showed the existence of inverse systems of the affine–in–inputs nonlinear systems. According to the theorem, higher order of derivatives of the outputs are needed to complete the inverse control, which implies difficulty in practical applications. For the general system (4.2.1) and (4.2.2), it is still not easy to demonstrate its invertibility. If the state–space model of a nonlinear system can be transformed into an input–output model, then its invertibility is easy to determine.

Differentiating (4.2.2) gives:

$$\dot{y} = \left[\frac{\partial h(x,u)}{\partial x}\right]^T f(x,u) + \left[\frac{\partial h(x,u)}{\partial u}\right]^T \dot{u} \equiv g_1(x, u, \dot{u}) \tag{4.2.3}$$

$$\ddot{y} = g_2(x, u, \dot{u}, \ddot{u}) \tag{4.2.4}$$

. . . . . .

$$y^{(q)} = g_q(x, u, \dot{u}, \quad . . ., u^{(q)}) \tag{4.2.5}$$

Since the system is assumed to be observable, there exists a minimum number q so that equations (4.2.3)–(4.2.5) have solutions:

$$x = \chi(y, \dot{y}, . . ., y^{(q)}, u, \dot{u}, \ . . ., u^{(p)}) \qquad p \leq q \tag{4.2.6}$$

Substituting (4.2.6) into (4.2.2) gives the input–output representation of eqs. (4.2.1) and (4.2.2):

$$y = h(\chi, u) \tag{4.2.7}$$

or

$$\phi(y^{(q)}, . . ., \dot{y}, y, u^{(p)}, . . ., \dot{u}, u) = 0 \tag{4.2.8}$$

Eq. (4.2.8) is called the input–output model of a nonlinear system. The following are the two special cases of eq. (4.2.8):

$$y^{(q)} = \psi(y^{(q-1)}, . . ., \dot{y}, y, u^{(p)}, . . ., \dot{u}, u) \tag{4.2.9}$$

$$y^{(q)} + a(y^{(q-1)}, . . ., \dot{y}, y) = b(u^{(p)}, . . ., \dot{u}, u) \tag{4.2.10}$$

The above derivation is not strictly equivalent, since the observability of system (4.2.1) and (4.2.2) is not well defined, and eqs. (4.2.3)–(4.2.5) may have many solutions of x. In addition, eq. (4.2.8) cannot represent the internal dynamics which are not observable with respect to the selected outputs. Therefore, systems described by eq. (4.2.8) belong to a subset of those described by eq. (4.2.1) and (4.2.2). However, this subset is rich enough to cover many engineering systems. As for linear systems, the input–output (IO) model (4.2.8) is very useful for both analysis and design. If systems described by eqs. (4.2.1) and (4.2.2) are completely observable with respect to the selected outputs, then model (4.2.8) is their input–state representation without internal dynamics. If systems described by eqs. (4.2.1) and (4.2.2) are not observable with respect to the selected outputs, but their internal dynamics are bounded–input bounded–state (BIBS), then model (4.2.8) is their input–output representation and useful for control design. Similarly, the relative degrees can be defined.

Lemma 4.2.1: For continuous–time nonlinear systems represented by (4.2.8),

a) Their inverse systems have the same input–output representation (4.2.8) as the original systems do.

b) Assume that $\phi(\,\cdot\,)$ in eq. (4.2.8) satisfies the implicit function theorem. Then, if a nonlinear system can be represented by eq. (4.2.8), then the system is invertible; If a nonlinear system is invertible and minimum–phase, and the measurements of $y, \dot{y}, \ldots, y^{(q-1)}$ are available, then an inverse controller can be constructed.

Proof: it is straightforward.

In the backprop–based indirect adaptive neurocontrol scheme (shown in fig. 4.2), both the neurocontroller and the neural network emulator of the controlled plant are in discrete–time form, while, in practice, most controlled engineering plants are continuous in time. Therefore, the corresponding indirect adaptive neurocontrol systems are sampled–data control systems. To analyze the control systems, the input–output models are discretized. Take the case of eq. (4.2.9) for simplicity.

There are several methods to discretize a NLS (see section 4.3). Assume that $\dot{y}(t)$, $\ldots$, $y^{(q)}(t)$ and $\dot{u}(t), \ldots, u^{(p)}(t)$ at $t = kT$ can be estimated numerically, based on the sampled data $\{y(kT),\ y((k-1)T), \cdots,\ y((k-\mu)T)\}$ and $\{u(kT),\ u((k-1)T), \cdots,\ u((k-\nu)T)\}$, respectively. Here, T is the sampling period and determined by the system dynamics, the band widths of the actuators and the control performance requirement. μ and ν are determined by the orders q and p of eq. (4.2.9), and the estimation precision requirement.

Substituting the numerically estimated $\dot{y}(t), \ldots, y^{(q)}(t)$ and $\dot{u}(t), \ldots, u^{(p)}(t)$ at $t = kT$ into eq. (4.2.9) gives its discretized input–output form:

$$y(kT) = \overline{\psi}(y((k-1)T), \ldots, y((k-\mu)T);\ u(kT), \ldots, u((k-\nu)T)) \qquad (4.2.11)$$

where $y \in R^{r_r}$ and $u \in R^{m_r}$. The discretized input–output representation (4.2.11) possesses the same advantage as its continuous–time counterpart (4.2.9).

Due to the mechanical or electrical inertia, practical dynamic systems always possess some time delay from input action to output response, which leads to a weak connection between $y(kT)$ and $u(kT)$ in eq. (4.2.11). This observation suggests the use of the d–step ahead prediction input–output model, where the integer d is determined by the system dynamics. If the system dynamics are of finite order, then $\{ y((k+d-1)T),\ y((k+d-2)T), \cdots, y((k+1)T) \}$ can be represented by $\{y(kT),\ y((k-1)T), \cdots, y((k-\mu)T)\}$ and $\{u(kT),\ u((k-1)T), \cdots, u((k-\nu)T)\}$. This leads to the following d–step ahead prediction input–output model:

$$y((k+d)T) = \overline{\phi}(y(kT), \ldots, y((k-\mu)T);\ u(kT), \ldots, u((k-\nu)T)) \qquad (4.2.12)$$

Lemma 4.2.2: *If $\overline{\psi}(\ \cdot\ )$ in eq. (4.2.11) or $\overline{\phi}(\ \cdot\ )$ in eq. (4.2.12) satisfies the condition of the implicit function theorem, then the system represented by eq. (4.2.11) or eq. (4.2.12) is invertible. Since the discretized model (4.2.11) or (4.2.12) only contains the current and past sampled data, the inverse controller is always realizable if the system is also minimum–phase.*

Next, we will prove the existence of the indirect adaptive controllers and give a theoretical solution of the inverse controller for the sampled–data nonlinear system (4.2.11). This solution is very important for determining the inputs and structures of the neurocontroller and the neural network emulator shown in fig. 4.2, which were seldom discussed in the published literature.

(2) The theoretical solution of inverse controllers

Consider the nonlinear system (4.2.1) and (4.2.2) which can be discretized into the input–output model (4.2.11). It is a well known fact that the number of outputs able to track arbitrary given trajectories is limited by system structures, and orders m_s and r_s. This fact is clarified by the following definition.

Definition 4.2.1: *For nonlinear systems described by eq. (4.2.11), there exists a number $r_I \le \min\{r_s,\ m_s\}$ so that when $y_1 \in R^{r_I}$, selected from y, eq. (4.2.13) has bounded solution(s) of $u(kT)$ for arbitrary $\{\ y_1(kT),\ y_1((k-1)T), \cdots,\ y_1((k-\mu)T)\ \}$ and past–determined $\{\ u((k-1)T), \cdots,\ u((k-\nu)T)\ \}$; when $y_1 \in R^{r_I+1}$, eq. (4.2.13) does not. Then r_I is called the maximum number of the independently–controllable outputs.*

$$y_1(kT) = \overline{\psi}_1(y((k-1)T), \ldots,\ y((k-\mu)T);\ u(kT), \ldots,\ u((k-\nu)T)) \qquad (4.2.13)$$

Only r_I independently–controllable outputs can be designed to track arbitrary trajectories. If more system outputs are expected to be able to track arbitrary trajectories, one has to change system structures, and/or add more external inputs to the systems.

For example, $r_I = 1$ for system *S1* and *S2*, and $r_I = 2$ for system *S3*.

$$S1: \quad \begin{bmatrix} y_1(kT) \\ y_2(kT) \end{bmatrix} = \begin{bmatrix} 2y_1((k-1)T) + u(kT) \\ u(kT) \end{bmatrix} \qquad (4.2.14)$$

$$S2: \quad \begin{bmatrix} y_1(kT) \\ y_2(kT) \end{bmatrix} = \begin{bmatrix} u_1(kT) + u_2^2(kT) \\ u_1(kT) + u_2^2(kT) \end{bmatrix} \qquad (4.2.15)$$

$$S3: \quad \begin{bmatrix} y_1(kT) \\ y_2(kT) \end{bmatrix} = \begin{bmatrix} \sin\{y_1((k-1)T)y_2((k-2)T)\} + u_1(kT) + u_2^2(kT) \\ u_2(kT) + u_3((k-1)T) \end{bmatrix} \qquad (4.2.16)$$

Theorem 4.2.1: *For minimum–phase nonlinear systems described by eq. (4.2.11), if $\overline{\psi}(\ \cdot\ )$ in eq. (4.2.11) satisfies the condition of the implicit function theorem and the $(r_s - r_I)$ non–independently–controllable outputs are BIBS, then there exists a realizable* *inverse controller able to make the r_I independently–controllable outputs track arbitrary given trajectories asymptotically.*

Proof: Let $y = [y_1^T \; y_2^T]^T$, where $y_1 \in R^{r_I}$ is the independently–controllable output vector, and $y_{d1}(kT)$ is the given desired output vector of the independently–controllable output vector at $t=kT$. That the $(r_s - r_I)$ non–independently–controllable outputs are BIBO implies that for any bounded $\| y_1(kT) \|$ and $\| u(kT) \|$ (k=0, 1, 2, . . .), $\| y_2(kT) \|$ determined by eq. (4.2.17) is also bounded.

$$y_2(kT) = \bar{\psi}_2(y((k-1)T), \; . \; . \; . \; , \; y((k-\mu)T); \; u(kT), \; . \; . \; . \; , \; u((k-\nu)T) \;) \quad (4.2.17)$$

Define the error vectors $e_1((k-i)T) = y_{d1}((k-i)T) - y_1((k-i)T)$ and let

$$\begin{aligned}&\bar{\psi}_1(y((k-1)T), \; . \; . \; . \; , \; y((k-\mu)T); \; u(kT), \; . \; . \; . \; , \; u((k-\nu)T) \;) \\ &= y_{d1}(kT) - \lambda e_1((k-1)T) \qquad (4.2.18)\end{aligned}$$

where $|\lambda|$ is less than 1. Since $\bar{\psi}(\; \cdot \;)$ in eq. (4.2.11) satisfies the condition of the implicit function theorem, there always exists a solution $u(kT)$ which can be expressed as follows:

$$\begin{aligned}u(kT) = \mathcal{U}(&y_{d1}(kT) - \lambda e_1((k-1)T), \; y((k-1)T), \; . \; . \; . \; , \; y((k-\mu)T); \\ &u((k-1)T), \; . \; . \; . \; , \; u((k-\nu)T) \;) \qquad (4.2.19)\end{aligned}$$

Substituting (4.2.17) into (4.2.11) gives:

$$e_1(kT) - \lambda e_1((k-1)T) = 0 \qquad (4.2.20)$$

Due to $|\lambda|$ being less than 1, $\| e_1(kT) \| \rightarrow 0$ as $k \rightarrow \infty$. Since the system is minimum–phase, the control law (4.2.19) is stable and realizable. This proves the existence of an inverse controller for system (4.2.11), which can make the r_I independently–controllable outputs track arbitrary given trajectories asymptotically and the non–independently –controllable outputs are bounded.

Remark 4.2.1: when $r_I = m_s$, there is a unique solution of $u(kT)$; when $r_I < m_s$, there are multiple solutions of $u(kT)$ and one can define the best solution according to a given criterion. The solution(s) can be analytical or numerical.

Remark 4.2.2: The explicit analytical solution (4.2.19) is difficult to obtain for complex nonlinear systems. Therefore, the significance of this theorem is that it shows the existence of the asymptotical inverse controllers.

Remark 4.2.3: The right–hand side of eq. (4.2.18) can contain more terms or even nonlinear functions of $e_1((k - i)T)$ ($i = 1, 2, \ldots \tau$), as long as they can guarantee that the tracking error equations are asymptotically stable. The larger the number τ is, the more robust the controller is to the errors of model (4.2.11); at the same time, the more complex the controller and the tracking error equations are.

Remark 4.2.4: The control law (4.2.19) is dynamic. Therefore, when a system described by eqs. (4.2.1) and (4.2.2) is non–minimum–phase, control law (4.2.19) is unstable and not realizable. In fact, no asymptotic tracking can be achieved for non–minimum–phase systems (Slotine and Li, 1991).

In practice, accurate system models are usually not available, especially the accurate parameters. Therefore, the above inverse controllers have to be robustified when applied to unknown or uncertain systems, which is beyond the scope of this paper. Nevertheless, it can serve as a theoretical basis for the construction of the indirect adaptive neurocontrol systems. The structure of the indirect adaptive neurocontrol systems is defined in the following.

4.2.2 Structure of indirect adaptive neurocontrollers

In the following, we assume that 1) the nonlinear system model represented by eqs. (4.2.1) and (4.2.2) is not available except for its order, and is minimum–phase; 2) the nonlinear system model represented by eq. (4.2.1) and (4.2.2) can be approximated by a sampled–data input–output model (4.2.11), which implies that it is invertible; and 3) the asymptotical inverse control law (4.2.19) exists and is stable. Then two neural networks are adopted to approximate the model (4.2.11) and the control law (4.2.19). This determines the structure of the indirect adaptive neurocontrol systems, which is shown in fig. 4.4. Here, *MSD* represents the multi–step delay operations; *V* represents the vectorization of its inputs; *C* represents the matrix that selects the independently–controllable outputs; *A/D* represents the analog–to–digital converter; *D/A* represents the digital–to–analog converter. The input vector to the neurocontroller, determined by eq. (4.2.19), is

$$x_C(kT) = [\ [y_{d1}(kT) - \lambda e_1((k-1)T)]^T\ y^T((k-1)T) \\ \quad \ldots\ y^T((k-\mu)T)\ u^T((k-1)T)\ \ldots\ u^T((k-\nu)T)\ ]^T \qquad (4.2.21)$$

The input vector to the neural network emulator, determined by eq. (4.2.11), is

$$x_M(kT) = [\ y^T((k-1)T)\ \ldots\ y^T((k-\mu)T)\ u^T(kT)\ \ldots\ u^T((k-\nu)T)\ ]^T \qquad (4.2.22)$$

The two neural networks in fig. 4.4 can be any kind of efficient neural networks.

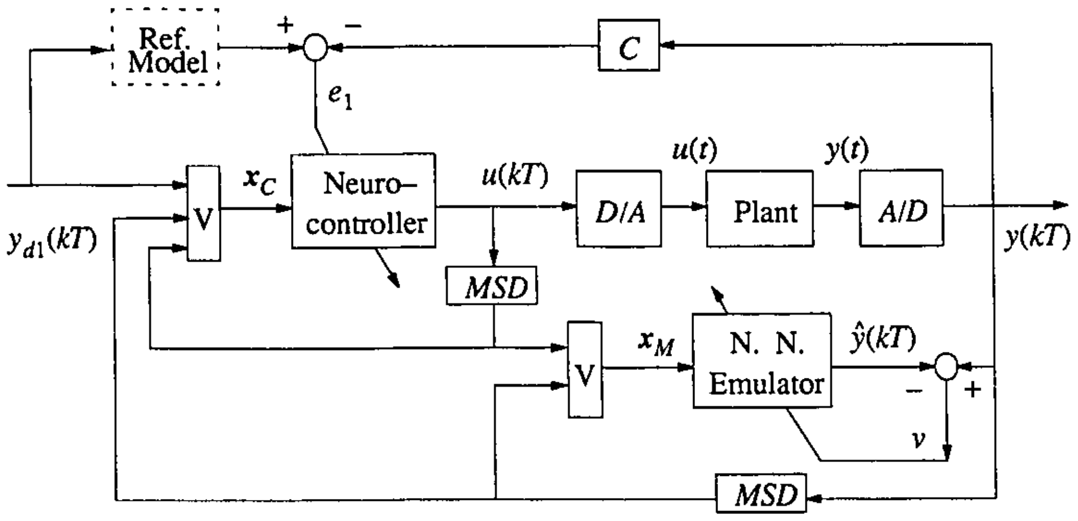


Fig. 4.4 The detailed structure of the indirect adaptive neurocontrol systems

Here, the localized polynomial networks with CLI cells, presented in section 3.2, are applied as the neurocontroller and the neural network emulator. When these networks are trained by the adaptive optimal Kalman filtering algorithm with the recursive UD factorization, their training processes are fast convergent.

To apply the adaptive optimal Kalman filtering algorithm with the recursive UD factorization, the input–weight–output relationships of the networks have to be set up. For the neural network emulator in fig. 4.4, we have

$$y_m(kT) = \hat{y}_m(kT) + v_m(kT) \qquad m = 1, \ldots, r_s \qquad (4.2.23)$$

$$\hat{y}_m(kT) = \Phi_M^T(x_M)\; W_{Mm}(kT) \qquad m = 1, \ldots, r_s \qquad (4.2.24)$$

where $y_m(kT)$ is the m–th output of the actual system; $\hat{y}_m(kT)$ is the m–th output of the neural network model; $\Phi_M(kT) \in R^{p_M}$ is defined similar to eq. (3.2.3) and p_M is determined by the dimension of x_M and the network order; $W_{Mm}(kT)$ is the weight and threshold vector of the neural network emulator for the m–th output; $v_m(kT)$ is the approximation error between $y_m(kT)$ and $\hat{y}_m(kT)$. $\{v_m(kT)\}$ can be regarded as a white noise process. Let

$$E\{v_m(kT)\} = r_{Mm}(kT) \qquad Cov\{v_m(kT),\; v_m(iT)\} = R_{Mm}(kT)\delta_{ki} \qquad (4.2.25)$$

For the neurocontroller in fig. 4.4, we have

$$y_{d1}(kT) = \hat{y}_1(kT) + e_1(kT) = h_1(x_C\,, W_C) + e_1(kT) \quad \in R^{r_I} \qquad (4.2.26)$$

$$W_C(kT) = [\; W_{C1}^T(kT) \;\ldots\; W_{Cr_I}^T(kT) \;]^T \qquad (4.2.27)$$

where $y_{d1}(kT)$ is the desired output vector of the system; $\hat{y}_1(kT)$ is the output vector of the neural network emulator and corresponds to the independently–controllable output vector $y_1(kT)$; $W_{Cm}(kT) \in R^{p_C}$ is the weight and threshold vector of the neurocontroller for the m–th control input $u_m(kT)$; p_C is determined by the dimension of x_C and the network order; $e_1(kT)$ is the approximation error between $y_{d1}(kT)$ and $\hat{y}_1(kT)$. $\{e_1(kT)\}$ can be regarded as a white noise process vector, and

$$E\{e_1(kT)\} = r_C(kT) \qquad Cov\{e_1(kT),\; e_1(iT)\} = R_C(kT)\delta_{ki} \qquad (4.2.28)$$

To apply the adaptive extended Kalman filtering algorithm, $H(kT)$, instead of $\Phi(kT)$ in eqs. (3.2.20) – (3.2.24), is required. It can be computed using the following formulae:

$$h_1(x_C, W_C) = [\; h_{11}(x_C, W_C) \;\ldots\; h_{1r_I}(x_C, W_C) \;]^T \qquad (4.2.29)$$

$$h_{1r}(x_C\ ,W_C) = \Phi_M^T(x_M)W_{Mr} \qquad r = 1,\ldots, r_I \tag{4.2.30}$$

$$u_m(kT) = \Phi_C^T(x_C)W_{Cm}(kT) \qquad m = 1,\ldots, m_s \tag{4.2.31}$$

$$\begin{aligned} H_{rm}(kT) &= \frac{\partial h_{1r}(x_C\ ,W_C)}{\partial W_{Cm}(kT)} = \frac{\partial h_{1r}(x_C\ ,W_C)}{\partial u_m(kT)}\frac{\partial u_m(kT)}{\partial W_{Cm}(kT)} \\ &= [\frac{\partial \Phi_M(x_M)}{\partial u_m(kT)}]^T W_{Mr}\Phi_C(x_C) \end{aligned} \tag{4.2.32}$$

$$H_m(kT) = [\ H_{1m}(kT)\ \ldots\ H_{r_I m}(kT)\ ] \qquad \in R^{p_C \times r_I} \tag{4.2.33}$$

$$H(kT) = [\ H_1^T(kT)\ \ldots\ H_{m_s}^T(kT)\ ]^T \qquad \in R^{m_s p_C \times r_I} \tag{4.2.34}$$

Then the neurocontroller and neural network emulator in fig. 4.4 can be trained by the adaptive extended Kalman filtering algorithm. Based on whether $W_{Cm}(kT)$ (m=1, 2, . . . , m_s) is updated as a whole or separately, the backprop–based indirect adaptive neurocontrol algorithm can be classified into the coupled one and the decoupled one. The latter is more efficient in computation and storage, but is suboptimal.

The coupled backprop–based indirect adaptive neurocontrol algorithm can be stated as follows:

a. Set up the initial values: $W_C(0)$, $W_M(0)$, $x_C(0)$ and $x_M(0)$.

b. Perform the forward computation of the neurocontroller based on eq. (4.2.31).

c. Apply the control input vector $u(kT)$ to the plant and measure its output vector $y(kT)$.

d. Perform the forward computation of the neural network emulator based on eq. (4.2.24).

e. Backward update $W_{Mm}(kT)$ ($m = 1,\ldots, r_s$) based on the adaptive optimal Kalman filtering algorithm with the recursive UD factorization.

f. Backward update $W_C(kT)$ based on the adaptive extended Kalman filtering algorithm with $H(kT)$ defined in eq. (4.2.34).

g. Set $k = k+1$, and go to step b.

The decoupled indirect adaptive neurocontrol algorithm is the same as the coupled one except that step f is changed into the following:

f. for $m = 1, \ldots, m_s$, backward update $W_{Cm}(kT)$ based on the adaptive extended Kalman filtering algorithm with $H_m(kT)$ defined in eq. (4.2.33), and perform the forward computation of the neural network model based on eq. eq. (4.2.33) with newly updated $W_{Cm}((k + 1)T)$.

When the order of the localized polynomial networks with CLI cells is $L = 1$, the neurocontroller becomes a set of linear controllers integrated by the CLI cells. So does the neural network emulator. In this case, global convergence can be achieved under some conditions.

Assume that the controlled plant model (4.2.11) belongs to a class $\boldsymbol{M}$ of operators, and its corresponding inverse controller belongs to a class $\boldsymbol{C}$ of operators. The neural network model in fig. 4.4 defines an input–output function relation N_M and the neurocontroller in fig. 4.4 defines an input–output function relation N_C.

Theorem 2.2:

For a structure-and-parameter-unknown nonlinear system which is described by eqs. (4.2.1) and (4.2.2) and can be represented by the discretized input–output model (4.2.11), if

1) the controlled plant is BIBS, invertible and minimum-phase in the working region Ω;

2) $N_M \in \boldsymbol{M}$ and $N_C \in \boldsymbol{C}$.

3) the desired outputs $y_{d1,m}(kT)$ are persistently exciting during the training phase of the neurocontrol system;

Then the indirect adaptive neurocontrol algorithm above can make its r_I independently–controllable outputs track arbitrary given trajectories asymptotically for some initial values $W_C(0)$.

Proof: Since condition 1) meets the requirement of *Theorem 4.2.1,* there exists, theoretically, a realizable asymptotical inverse controller able to make the r_I independently–controllable outputs track arbitrary given trajectories asymptotically. Now the problem becomes whether there exists an indirect adaptive neurocontroller that functions as the asymptotical inverse controller.

Condition 2) guarantees that the neural network emulator can approximate the controlled plant, and the neurocontroller can approximate the asymptotical inverse controller, both with arbitrary precision. Then since the controlled plant described by eqs. (4.2.1) and (4.2.2) is time–invariant and the adaptive extended Kalman filtering algorithm is fast convergent, the indirect adaptive neurocontrol algorithm can guarantee that

$$\hat{y}_1(kT) \rightarrow y_1(kT) \quad \text{and} \quad \hat{y}_1(kT) \rightarrow y_{d1}(kT) \qquad \text{as } k \rightarrow \infty \tag{4.2.35}$$

for some initial values $W_C(0)$, and arbitrary initial values $W_M(0)$ and $y(0) \in \Omega$. Therefore, we have:

$$y_1(kT) \rightarrow y_{d1}(kT) \qquad \text{as } k \rightarrow \infty \tag{4.2.36}$$

which proves the theorem. The tracking error convergence speed is determined by the training algorithms of the two neural networks, in this case, the adaptive extended Kalman filtering algorithm. Therefore, the tracking errors are also fast convergent. Condition 3) guarantees that the indirect adaptive neurocontroller will be valid in the whole working region of the system.

The definition of persistent excitation can be found in Narendra and Annaswamy (1989). Assume that there is an input–output relation: $y(t) = \phi(u(t), t)\theta$, where $u(t)$ is the input vector; $y(t)$ is the output vector; $\phi(u(t), t)$ is a vector function; and θ is a constant parametric vector. Then $u(t)$ is called persistently exciting if the following inequality holds for any T and $t \in [0, \infty)$:

$$\int_{t}^{t+T} \phi(u(\tau), \tau)\phi(u(\tau), \tau)^T d\tau > kI$$

where k is a positive real number.

Remark 4.2.5: If the controlled plant is BIBS in the working region Ω, whether the learning processes are globally or locally convergent does not affect the system stability. Locally convergent learning may imply large tracking errors, but not instability. These tracking errors can be made uniformly bounded by proper neural network design. However, if the neural network weights are allowed to change freely, then the neural network emulator may lose its minimum phase, stabilizability/controllability during learning, just as conventional adaptive controllers. In those abnormal cases, the tracking errors may not be uniformly bounded. These phenomena have been observed in neurocontroller training. Therefore, constraints on the weights and control inputs have to be imposed. This is still an open area, even for adaptive control of linear systems.

Remark 4.2.6: *Theorem 4.2.2* assumes that the controlled plant described by eqs. (4.2.1) and (4.2.2) is time–invariant. If this is not true, but the structure or parameter change speed of the controlled plant is much slower than the learning speed of the neural network emulator and the neurocontroller shown in fig. 4.4, then *Theorem 2.2* is still valid. In fact, fast linear system model changes are quite possible because most linear system models are derived by linearizing nonlinear and/or time–varying systems. Fast system state changes can often lead to fast linear system model or parameter changes. However, most nonlinear system models can be regarded as time–invariant or slow–varying, although there do exist some sudden changes (which can be regarded as external impulse disturbances) when the systems come into contact with their environments. Therefore, the above assumption about slow–varying nonlinear systems can often be satisfied. The slow system changes along with fast neural network learning can guarantee system stability.

Remark 4.2.7: Condition 2) implies that the neural network emulator with specified structure can approximate the controlled plant model (4.2.11) and the neurocontroller with specified structure can also approximate the inverse controller (4.2.19). The more knowledge there is about the controlled plant model, the more confident we are in determining the structures of the neurocontrollers and the neural network emulators.

Remark 4.2.8: The training phase can be regarded as part of the system design phase. If the desired outputs $y_{d1,m}(kT)$ are not persistently exciting, then the indirect adaptive neurocontroller also works, but are only valid for the trained desired outputs. For new different desired output commands, the neurocontroller will learn to track them due to its continuous learning mechanism. If $u(kT)$ is persistently exciting, then the identified neural network emulator is valid in the whole working region Ω; If $u(kT)$ is not, then the identified neural network emulator is valid only in a local working region the control inputs $u(kT)$ have covered. In the indirect adaptive neurocontrol systems, $u(kT)$ cannot be specified to be persistently exciting. Fortunately, we do not care if the identified neural network model is global or local, as long as the r_l independently–controllable outputs can track given trajectories asymptotically.

Remark 4.2.9: Using different neural networks and/or different learning algorithms results in different neurocontrollers with different convergent speeds.

As for neural networks, it is known now that RBF networks, higher order polynomial networks and CMAC are more efficient in learning than the multilayer feedforward perceptrons (MFP), though the latter was more popular. It was proved in section 3.2 that localized networks, in which only a small size sub–network is trained and responsible for output at each iteration, are much more efficient in learning than global networks in which all the network weights are updated together.

The widely applied learning algorithms are the backpropagation (BP) algorithm, various modified BP algorithms (MBP), the simulated annealing algorithm (SAA), genetic algorithms (GA), the variable metric methods (VMM), the conjugate gradient methods (CGM), and the extended Kalman filtering (EKF) algorithm. BP and MBP algorithms are relatively simple. SAA and GA are designed to achieve global convergence. However, all these four are slow in convergence. VMM, CGM and EKF algorithms are among the best in the sense of learning speed. However, for global networks, the EKF algorithm suffers from intensive computation. When VMM and CGM utilize one–dimension–optimization procedures to determine step lengths (or learning constants), the amount of computation is large; When VMM and CGM adopt fixed or *ad hoc* step lengths, their convergent speed will be lowered.

The advantages of localized polynomial networks include: 1) fast learning and global convergence property, which is adequate for applications that need real–time or fast learning; 2) modular and additive structure, which makes the overall network structure easy to determine; 3) extendibility to new input space and graceful degradation.

In summary, the localized polynomial networks with CLI cells trained by the AEKF algorithm are some of the most efficient neural networks in the sense of learning speed and computation amount. Also, they are not sensitive to the initial values of the network weights. Therefore, the indirect adaptive neurocontrollers based on these networks are among the best. The system output tracking errors converge rapidly. If localized linear networks are used, then the learning processes of the neurocontrollers are globally convergent.

4.2.3 Extensions

(1) Incorporating a nominal model into the neural network emulator:

If it is available, a default model of a controlled plant with finite precision can be used to a) pre–train the neural network emulator and the neurocontroller to obtain better initial values $W_M(0)$ and $W_C(0)$ and to avoid large initial tracking errors; b) construct indirect adaptive neurocontrol systems with default models and neural network error (NNE) models (shown in fig. 4.5).

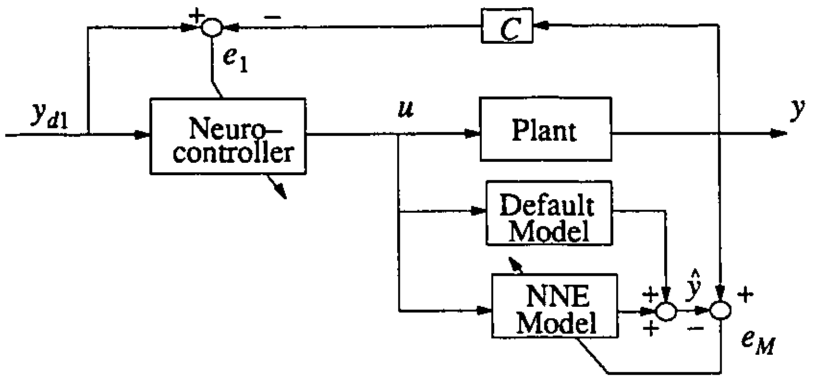


Fig. 4.5 The schematic diagram of the indirect adaptive neurocontrol system with a default plant model

(2) Incorporating a conventional controller into the neurocontroller

If there is a conventional controller for the plant, then the backprop–based indirect adaptive neurocontroller can be used to improve the control performance, as shown in fig. 4.6.

(3) Hybrid fuzzy and neurocontrol

If the controlled plant is not BIBO in the working region, then a supplementary controller is needed to stabilize the system within the working region Ω, and thus make the controlled plant BIBO. For example, a fuzzy controller can be constructed as a stabilizer (fig. 4.7). When the system outputs are outside the working region, the fuzzy controller works

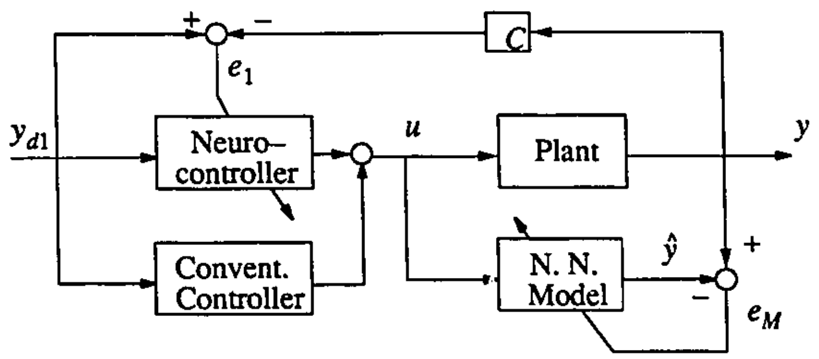


Fig. 4.6 Incorporating a conventional controller into the neurocontroller

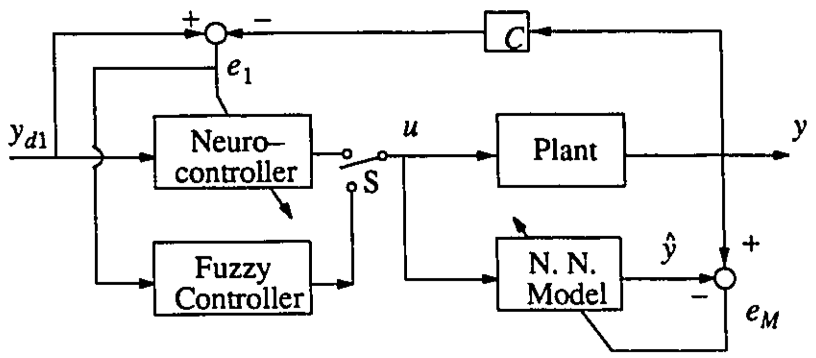


Fig. 4.7 The schematic diagram of hybrid neurocontrol and fuzzy control

and confines the system outputs to be in the working region Ω, within which the neurocontroller works.

4.2.4 Simulation tests

The indirect adaptive neurocontrol scheme was first tested by making a two–input two–output continuous–time linear system track sine signals. We let $T = 0.1$ sec. The order of the localized polynomial networks was set to 1. The initial values of the two networks were set randomly. Without any information about the system parameters and without pre–training of the neural network model, very good tracking control was achieved after only 60 iterations (see fig. 4.8 for one input and output). It was also tested by controlling a single–input nonlinear system. The fast convergence of the neurocontrol system was again demonstrated in this case. However, good initial values of the neurocontroller weights were re-

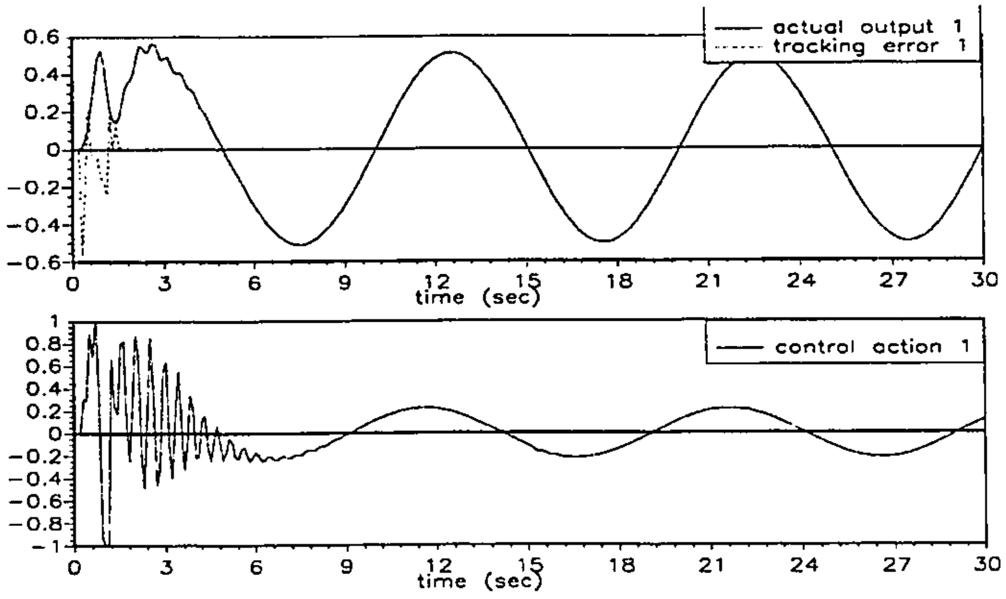


Fig. 4.8 Time response of a neurocontrol system

quired. This reflects the locally convergent property of the indirect adaptive neurocontrol scheme.

4.3 DIRECT ADAPTIVE NEUROCONTROL

Fig. 4.9 shows the structure of a backprop–based direct adaptive neurocontrol system. Its basic working principle is to use the tracking error vector *e* to tune the weights of the neurocontroller. Similar to the indirect adaptive neurocontrol, the direct adaptive neurocontroller is an inverse controller of the plant.

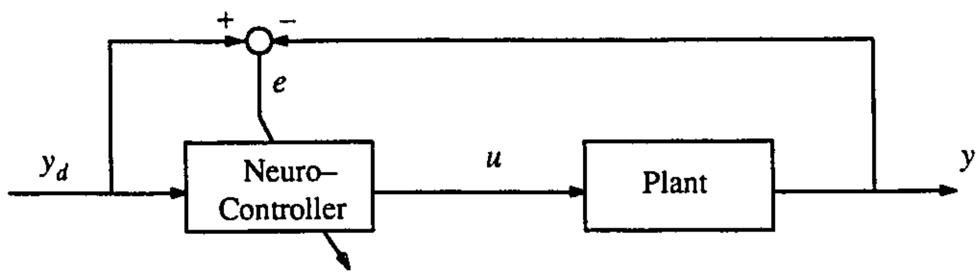


Fig. 4.9 Backprop–based direct adaptive neurocontrol scheme

The backprop–based model reference direct adaptive neurocontrol scheme can also be constructed as shown in fig. 4.10.

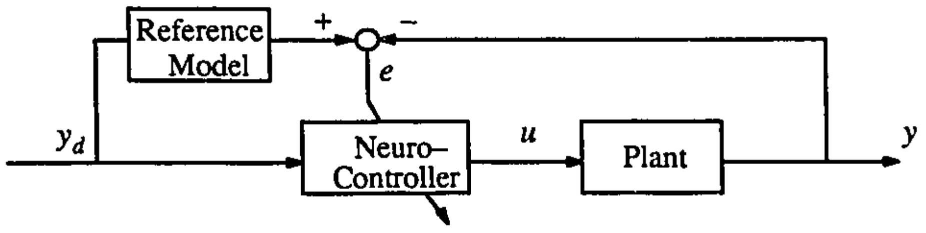


Fig. 4.10 The backprop–based model reference direct adaptive neurocontrol systems

The direct adaptive neurocontrol scheme is simpler, but more difficult to train, due to the unavailability of the partial derivatives of y w.r.t. u. The existing training algorithms for direct adaptive neurocontrol schemes are: 1) a backpropagation algorithm when the system models are known; 2) reinforcement algorithms; 3) a random search algorithm; and 4) a simple estimated gradient algorithm. When system models are unknown, approach 1) is not applicable. Approaches 2) and 3) are slow in learning convergence. The available method for approach 4) is too rough, since the Euler formula was used to estimate the gradient. Several representative direct adaptive neurocontrol schemes are reviewed in section 2.2.1.

In the following, two new extensions of the backprop–based direct adaptive neurocontrol are presented. The first one is based on accurate estimation of the input–output gradients. The second one is based on the optimization algorithm without using the derivatives, which gives fast learning convergence.

4.3.1 Existence of direct adaptive neurocontrollers

Since direct adaptive neurocontrollers (DANC) are inverse controllers, the existence of direct adaptive neurocontrollers is the same as that of indirect adaptive neurocontrollers. Therefore, all the conclusions and discussion in section 4.2.1 apply to backprop–based direct adaptive neurocontrol systems.

4.3.2 DANC with estimated gradients

Analyzing the learning algorithms of the indirect adaptive neurocontrol schemes, we can see that the function of the neural network emulator in fig. 4.2 is only to supply an estimation of the input–output Jacobian matrix of the controlled plant. Therefore, if an accurate estimation of the input–output Jacobian matrix of the controlled plant can be obtained through other approaches, the neural network emulator in fig. 4.2 is not necessary. When the neural network emulator in fig. 4.2 is removed, the indirect adaptive neurocontrol scheme becomes the direct one.

Psaltis, Sideris and Yamamura (1987a) tried to use the Euler formula to estimate the partial derivatives of system outputs with respect to inputs, which form the input–output Jacobian matrix. However, such an estimation can be quite erroneous, since the change of control inputs from the neurocontroller is not always small. This problem is solved here by introducing some numerical analysis techniques for estimating the input–output Jacobian matrix of the controlled plant with high precision.

The nonlinear systems considered here are described by eqs. (4.2.1) and (4.2.2). They can be approximated by a sampled–data input–output model (4.2.11). The independently–controllable output vector is y_1. The desired trajectory of y_1 is y_{d1}. All the assumptions about the systems are the same as those in section 4.2.1. The direct adaptive neurocontroller is designed to approximate eq. (4.2.19), and its input vector is defined by eq. (4.2.21).

Assume that the neurocontroller in fig. 4.9 is realized by the localized polynomial networks introduced in section 3.2. To apply the adaptive optimal Kalman filtering algorithm with the recursive UD factorization, the input–weight–output relationships of the neurocontrollers have to be set up. For the neurocontroller in fig. 4.9, we have

$$y_{d1}(kT) = y_1(kT) + e_1(kT) = h_1(x_C\ , W_C) + e_1(kT) \quad \in R^{r_1} \qquad (4.3.1)$$

$$W_C(kT) = [\ W_{C1}^T(kT)\ \ldots\ W_{Cr_l}^T(kT)\ ]^T \qquad (4.3.2)$$

where $y_{d1}(kT)$ is the desired output vector of the system; $y_1(kT)$ is the output vector of the plant; $W_{Cm}(kT) \in R^{p_C}$ is the weight and threshold vector of the neurocontroller for the m–th control input $u_m(kT)$; p_C is determined by the dimension of x_C and the network order; and $e_1(kT)$ is the measurement noise. Assume that $\{ e_1(kT) \}$ is a white noise process vector, and

$$E\{e_1(kT)\} = r_C(kT) \qquad Cov\{e_1(kT),\ e_1(iT)\} = R_C(kT)\delta_{ki} \qquad (4.3.3)$$

To apply the adaptive extended Kalman filtering algorithm, $H(kT)$ instead of $\Phi(kT)$ in eqs. (3.2.20) – (3.2.24) is required. It can be computed using the following formulae:

$$H(kT) = \frac{\partial h_1(x_C, W_C)}{\partial W_C} = \frac{\partial y_1(kT)}{\partial u(kT)} \frac{\partial u(kT)}{\partial W_C} \in R^{r_l \times m_* p_C} \qquad (4.3.4)$$

$$\frac{\partial u(kT)}{\partial W_C} = \Phi_C^T(x_C) \qquad (4.3.5)$$

where $\Phi_C(x_C)$ is known (defined by eq. 3.2.3). If the input–output Jacobian matrix $\frac{\partial y_1(kT)}{\partial u(kT)}$ can be estimated, then the adaptive optimal Kalman filtering algorithm with the recursive UD factorization can be used to train the direct adaptive neurocontrollers.

If the backpropagation algorithm is used for the neurocontrollers, it is easy to show that the training algorithm is always related to the input–output Jacobian matrix $\frac{\partial y_1(kT)}{\partial u(kT)}$. Thus, the values of the elements of $\frac{\partial y(k)}{\partial u(k)}$ do affect the descent directions of the training process, although small errors are tolerable.

If the system model equations (4.2.1) and (4.2.2) are known, then $\frac{\partial y_1(kT)}{\partial u(kT)}$ can be calculated as follows:

$$\frac{\partial y(t)}{\partial u(t)} = \frac{\partial h}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial h}{\partial u} \qquad t=kT \qquad (4.3.6)$$

$$\frac{d}{dt}\frac{\partial x}{\partial u} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial f}{\partial u} \qquad (4.3.7)$$

Therefore, the above neurocontroller can be easily applied to control invertible nonlinear systems with exact models.

However, $f(x,u)$ and $h(x,u)$ are unknown in most practical problems, so estimating $\frac{\partial y_1(kT)}{\partial u(kT)}$ from eqs. (4.3.6) and (4.3.7) is not a feasible solution. In the following, we apply numerical analysis techniques to give sufficiently accurate estimation of $\frac{\partial y_1(kT)}{\partial u(kT)}$.

The entries of $\frac{\partial y_1(kT)}{\partial u(kT)}$ are $\frac{\partial y_i}{\partial u_j}$ (i=1, 2, . . . , r_l and j=1, 2, . . . , m_s). The problem is how to estimate $\frac{\partial y_i}{\partial u_j}$ at t=kT with sufficient accuracy, for given sampled data :

$$\{u_j(kT),\ y_i(kT);\ u_j((k-1)T),\ y_i((k-1)T);\ \cdot\cdot\cdot;u_j((k-\mu),\ y_i((k-\mu)T)\} \qquad (4.3.8)$$

where μ is a given positive integer and is related to the precision of the approximation.

Since it is difficult to tell which control input causes the change of the outputs, the numerical estimation approach can only be applied to single–input nonlinear systems, or multiple–input nonlinear systems trained using one input at a time.

The simplest method is the backward Euler formula (Psaltis, Sideris and Yamamura, 1987a):

$$\left[\frac{\partial y_i}{\partial u_j}\right]_{t=kT} = \frac{y_i(kT) - y_i((k-1)T)}{u_j(kT) - u_j((k-1)T)} \qquad (4.3.9)$$

Because $|u_j(kT) - u_j((k-1)T)|$ is changing and not necessarily small, the accuracy of formula (4.3.9) cannot be high.

Noticing that u_j and y_i are both time functions, we have:

$$\left[\frac{\partial y_i}{\partial u_j}\right]_{t=kT} = \left[\frac{\dot{y}_i(t)}{\dot{u}_j(t)}\right]_{t=kT} \tag{4.3.10}$$

Therefore, if we can obtain high precision estimations of $\dot{u}_j(kT)$ ($j=1,2,\dots,m_s$) and $\dot{y}_i(kT)$ ($i=1,2,\dots,r_l$), the problem is solved. For simplicity, we consider the estimation of $\dot{y}(t)$ for $t \in [\ (k-\mu)T,\ kT\ ]$, based on sampled data $\{\ y(kT),\ y((k-1)T);\ \cdots,\ y((k-\mu)T)\ \}$. Then we can apply the formulas to $\dot{u}_j(kT)$ ($j=1,2,\dots,\overline{m}$) and $\dot{y}_i(kT)$ ($i=1, 2,\dots,r$).

Assume that $y(t) \in C^{\mu+1}$ and $T < 1$. Three different high precision methods for estimating the derivatives of $y(t)$ based on sampled data are given in the following.

Method 1 : Direct Multistep Approximation

The Taylor series expansion of $y(t-iT)$ at time instant t is

$$y(t-iT) = y(t) - iT\dot{y}(t) + \frac{(iT)^2}{2}\ddot{y}(t) + \cdots + \frac{(-iT)^{\mu}}{\mu!}y^{(\mu)}(t) + \varepsilon_{\mu}^{i}(t) \tag{4.3.11}$$

where $i = 1, 2, \dots, \mu$ and

$$\varepsilon_{\mu}^{i}(t) = \frac{(-iT)^{\mu+1}}{(\mu+1)!}y^{(\mu+1)}(\xi_i) \qquad \xi_i \in [\ t-iT,\ t\ ] \tag{4.3.12}$$

If μ+1 steps of $y(t-iT)$ are used, multiplying eq. (4.3.11) by b_i and summing them together gives:

$$\sum_{i=1}^{\mu} b_i y(t-iT) = \sum_{i=1}^{\mu} b_i\{y(t) - iT\dot{y}(t) + \frac{(iT)^2}{2}\ddot{y}(t) + \cdots + \frac{(-iT)^{\mu}}{\mu!}y^{(\mu)}(t) + \varepsilon_{\mu}^{i}(t)\} \tag{4.3.13}$$

Letting the coefficients of $\ddot{y}(t),\dots,y^{(\mu)}(t)$ be zeros, we obtain:

$$\sum_{i=1}^{\mu} b_i\frac{(-iT)^m}{m!} = 0 \qquad\qquad m = 2,\ 3,\ \cdots\ \mu \tag{4.3.14}$$

or $$\sum_{i=1}^{\mu} i^m b_i = 0 \qquad m = 2,\ 3,\ \cdots\ \mu \tag{4.3.15}$$

$$\begin{bmatrix} 2^2 & 3^2 & \cdots & \mu^2 \\ 2^3 & 3^3 & \cdots & \mu^3 \\ & & \cdots & \\ 2^\mu & 3^\mu & \cdots & \mu^\mu \end{bmatrix} \begin{bmatrix} b_2 \\ b_3 \\ \vdots \\ b_\mu \end{bmatrix} = - \begin{bmatrix} b_1 \\ b_1 \\ \vdots \\ b_1 \end{bmatrix} \tag{4.3.16}$$

Because the coefficient matrix in eq. (4.3.16) is always invertible, we can find b_i ($i = 1, 2, \ldots, \mu$) to make the approximation error of $\dot{y}(t)$ proportional to $\frac{T^{\mu+1}}{(\mu + 1)!}$. From eq. (4.3.13), we have

$$\dot{y}(t) = \{ \sum_{i=1}^{\mu} b_i\, y(t) - \sum_{i=1}^{\mu} b_i y(t - iT) \}/\{ T \sum_{i=1}^{\mu} ib_i \} + \bar{\varepsilon}_\mu(t) \tag{4.3.17}$$

and the approximation error is

$$\bar{\varepsilon}_\mu(t) = \{ \sum_{i=1}^{\mu} b_i \varepsilon_\mu^i(t) \}/\{ T \sum_{i=1}^{\mu} ib_i \} \propto O(T^\mu) \tag{4.3.18}$$

From eqs. (4.3.17) and (4.3.18), we can see that if $y(t)$ is a n–th order polynomial of t, then direct $(n+1)$–step approximation gives an exact estimation of $\dot{y}(t)$, regardless of the value of T.

For $\mu = 3$, we have

$$\dot{y}(t) \doteq \frac{1}{6T}[11y(t) - 18y(t - T) + 9y(t - 2T) - 2y(t - 3T)] \tag{4.3.19}$$

For $\mu = 4$, we have

$$\dot{y}(t) \doteq \frac{1}{12T}[25y(t) - 48y(t - T) + 36y(t - 2T) - 16y(t - 3T) + 3y(t - 4T)] \tag{4.3.20}$$

Example: let $y(t) = t^3$, $\dot{y}(t) = 3t^2$. Assume that we know the samples at t = 1, 2, 3, 4 ($T = 1$), which are $y(1) = 1$, $y(2) = 8$, $y(3) = 27$, $y(4) = 64$. Then applying formula (4.3.19) gives

$$\dot{y}(4) = \frac{1}{6}[11y(4) - 18y(3) + 9y(2) - 2y(1)] = 48 \qquad (4.3.21)$$

The estimated $\dot{y}(4)$ is equal to the exact value. This verifies our claim and the power of the formula.

Method 2 : Polynomial Interpolation

By applying the interpolation polynomial in the Lagrange form, we have, for $t \in [(k-\mu)T,\ kT]$,

$$y(t) = \sum_{i=0}^{\mu} y((k-i)T)\ l_i(t) + \varepsilon(t) \qquad (4.3.22)$$

where

$$l_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{\mu} \frac{t-(k-j)T}{(j-i)T} \qquad (4.3.23)$$

$$\varepsilon(t) = \frac{1}{(\mu+1)!} y^{(\mu+1)}(\xi_t) \prod_{i=0}^{\mu} [t-(k-i)T] \qquad \xi_t \in [(k-\mu)T,\ kT] \qquad (4.3.24)$$

where $\varepsilon(t)$ is the approximation error. If function $y(t)$ is smooth, then high precision can be achieved by increasing μ and decreasing T. From eq. (4.3.22), we can obtain the approximation of its derivative:

$$\dot{y}(t) \doteq \sum_{i=0}^{\mu} y((k-i)T)\ \dot{l}_i(t) \qquad (4.3.25)$$

$$\dot{l}_i(t) = \sum_{m=0}^{\mu} \prod_{\substack{j=0 \\ j \neq i,m}}^{\mu} \frac{t-(k-j)T}{(j-i)(m-i)T^2} \qquad (4.3.26)$$

Method 3 : Least–Squares Chebyshev Approximation

The Chebyshev polynomials are defined recursively as follows:

$$T_0(x) = 1 \qquad\qquad T_1(x) = x \qquad (4.3.27)$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{4.3.28}$$

and $\{\ T_0(x),\ T_1(x),\ \cdots, T_n(x)\ \}$ is an orthonormal system for $x \in [-1,\ 1]$ and any n.

Let

$$t = kT + \frac{\mu T}{2}(x-1) + T(x+1) \quad t \in [(k-\mu)T,\ kT + 2T] \tag{4.3.29}$$

$$y(t) \doteq \sum_{i=0}^{n} c_i T_i(x) \ = \sum_{i=0}^{n} c_i \hat{T}_i(t) \tag{4.3.30}$$

$$\begin{bmatrix} \hat{T}_0(kT) & \hat{T}_1(kT) & \cdots & \hat{T}_n(kT) \\ \hat{T}_0((k-1)T) & \hat{T}_1((k-1)T) & \cdots & \hat{T}_n((k-1)T) \\ \cdot & \cdot & \cdots & \cdot \\ \hat{T}_0((k-\mu)T) & \hat{T}_1((k-\mu)T) & \cdots & \hat{T}_n((k-\mu)T) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \cdot \\ c_n \end{bmatrix} = \begin{bmatrix} y(kT) \\ y((k-1)T) \\ \cdot \\ y((k-\mu)T) \end{bmatrix} \tag{4.3.31}$$

where $n \le \mu$. Therefore, the standard least–square method can be applied to eq. (4.3.31) to find the best approximation. After $c_i\,(i = 1, 2, \ldots, n)$ is computed, we can calculate $\dot{y}(t)$ using eq. (4.3.32).

$$\dot{y}(t) = \sum_{i=0}^{n} c_i \dot{\hat{T}}_i(t) \tag{4.3.32}$$

The approximation precision is related to n and μ.

Of these three methods, the direct multistep approximation method and the polynomial interpolation method are computationally more efficient. However, the least–squares Chebyshev approximation method is more robust to high measurement noise, especially when there are outliers.

Choosing any one of the above three methods, we can obtain a sufficiently accurate estimation of the input–output Jacobian matrix $\frac{\partial y_1(kT)}{\partial u(kT)}$. Therefore, the direct adaptive neurocontrol problem of unknown nonlinear systems is solved.

For multi–input systems, this scheme can only be used for off–line training since only one input is allowed to act on the systems at a time. These formulae are also useful in estimating velocity and acceleration signals with high precision.

4.3.3 DANC Using Optimization Without Using Derivatives

Besides the input–output Jacobian matrix estimation approach, there is another approach to backprop–based direct adaptive neurocontrol which does not require the input–output Jacobian matrix, i.e., the random searching scheme (Spall and Cristion, 1992). A similar approach is the reinforcement learning scheme (Sutton, Barto and Williams, 1992). The random searching and the reinforcement learning algorithms are simple, and no derivative information is required to train the neurocontrollers. However, they are slow in convergence, as is well known. In the following, the existing optimization algorithms without using derivatives are introduced to train the direct adaptive neurocontrollers and achieve fast learning convergence. The problem is formulated as follows.

The nonlinear systems considered here can be approximated by the sampled–data input–output model (4.2.11). All the assumptions about the nonlinear systems are the same as those in section 4.2.1. The direct adaptive neurocontroller is designed to approximate eq. (4.2.19), and its input vector is defined by eq. (4.2.21). For the direct adaptive neurocontroller shown in fig. 4.9 or 4.10, we define the following performance index:

$$E(t) = \frac{1}{2}\int_0^t [\ e(\tau)^T Q e(\tau) + u(\tau)^T R u(\tau)\ ]\ d\tau \qquad (4.3.33)$$

$$e(t) = y_d(t) - y(t) \qquad (4.3.34)$$

where $e(t)$ is the output tracking error vector; $y_d(t)$ is the desired output vector; $y(t)$ is the actual output vector, which is independently–controllable; $u(t)$ is the control input vector to the plant; Q and R are positive definite matrices.

If the neurocontrol algorithms are implemented on digital computers and based on sampled data, we can discretize eq. (4.3.33) into the following:

$$E(k) = \frac{T}{2}\sum_{i=0}^{k}[\ e(iT)^T Q e(iT) + u(iT)^T R u(iT)\ ] \tag{4.3.35}$$

where T is the sampling period. $e(iT)$ and $u(iT)$ are both the functions of the neurocontroller weights W_C. Similar to what was stated in section 4.2, the structure of the neurocontroller has to ensure the existence of the optimal weights for the given tracking control problem.

Then the modified Powell algorithm (Brent, 1973) can be applied to the index (4.3.35) to train the neurocontroller in fig. 4.9 or 4.10. Due to the fast convergence property of the modified Powell algorithm, better control performance can be achieved.

4.3.4 Enhancing Conventional Controllers

Similarly, the backprop–based direct adaptive neurocontrol schemes can be used to enhance the control performance of conventional controllers for both structural and parametric uncertainty in the controlled plant (fig. 4.11).

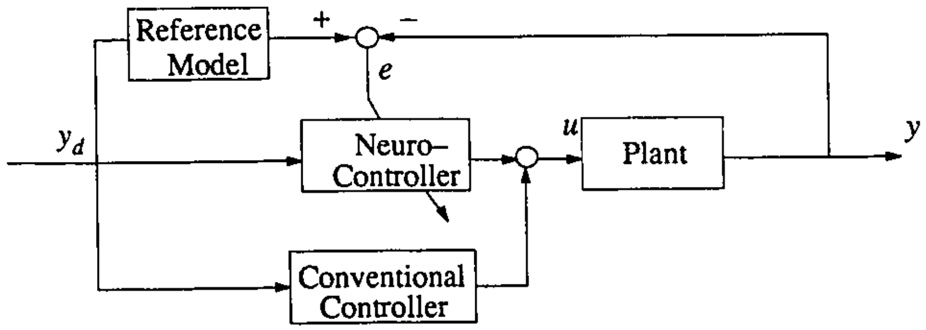


Fig. 4.11 Conventional controller enhanced by direct adaptive neurocontrol

4.4 SUMMARY

This chapter presents some new extensions and theoretical justifications of the backprop–based indirect and direct adaptive neurocontrol approaches. The invertibility of a nonlinear system is shown clearly by its input–output models. Based on the input–output models, the structure of the indirect and direct adaptive neurocontrol schemes is determined theoretically for the first time, and can ensure the stabilizability of the neurocontrol systems.

The localized polynomial networks are introduced to realize the neurocontrollers. Faster learning convergence is achieved, compared with existing schemes using sigmoidal neural networks and the backpropagation algorithms.

Three different numerical formulae are presented to estimate the input–output Jacobian matrix of a nonlinear system with high precision. Then a new backprop–based direct adaptive neurocontrol scheme is formulated based on the input–output Jacobian matrix. This scheme is a good and practical choice for single input nonlinear systems.

A new direct adaptive neurocontrol scheme based on optimization algorithms without using derivatives is also proposed. The new scheme possesses faster learning convergence compared with the random search scheme.

As stated at the beginning of the chapter, all the learning algorithms for backprop–based neurocontrol schemes are locally convergent, and it is difficult to prove their system stability. It was observed from the thesis research that the stability of neurocontrol systems is not directly related to the locality of the learning convergence, but to the speed of the learning convergence. If the learning convergence speed is faster than that of the divergence speed (dynamic response speed) of the controlled systems, then stable neurocontrol systems can be obtained. That is why the thesis is also devoted to developing fast learning algorithms and localized neural networks. The local learning convergence may lead to large control errors. Global learning convergence may imply small or zero control errors.

Similar to adaptive control, the loss of stabilizability/controllability and the minimum–phase property during the learning process may cause problems for the neurocontrol systems. However, due to their simplicity in concept and implementation, backprop–based neurocontrol schemes are attractive to practitioners and have found many applications in engineering.

There is a myth in neural learning control: that a neurocontrol system can work well during learning. This is like asking if a new worker can make high quality products when he/she has not acquired the skill. Both the neurocontrol system and the new worker have the learning ability, but this ability does not guarantee that they will not make mistakes. After a while of learning, both of them may work at the expert level. However, mistakes are unavoidable before this stage. In fact, the neurocontrol systems are designed to learn from mistakes (errors). Therefore, from a practical point of view, it is better to divide the usage of neurocontrol systems into a learning stage and an expert working stage. Only after it becomes an expert can a neurocontrol system complete quality work.

CHAPTER 5

SELF–TUNING NEUROCONTROL SCHEMES

5.1 INTRODUCTION

There are numerous neurocontrol schemes in the published literature (Miller III, Sutton and Werbos, 1990, Hunt, Sbarbaro, Zbikowski and Gawthrop, 1992). Their basic principles are all based on the universal approximation ability and learning ability of neural networks. According to their design modes and the adaptation ability, neurocontrollers may be classified into: (1) on–line trained neurocontrollers, (2) off–line trained neurocontrollers, and (3) hybrid off–line and on–line trained neurocontrollers, although they all have a learning stage and a working stage (section 4.4).

On–line trained neurocontrollers refer to those developed by pure on–line learning with little a priori knowledge about the controlled systems. Therefore, they are more attractive and more robust to system and environment changes, just like the adaptive controllers for linear systems. Global convergence of the adaptation laws, which are the learning algorithms of the neural networks, are crucial for the global stability and successful implementation of such systems. Off–line trained neurocontrollers refer to those developed off–line using known system default models. The neurocontrollers with fixed weights are applied to the actual systems, after training is completed. Therefore, global convergence of the learning algorithms for such neurocontrol systems is not necessary, since satisfactory neurocontrollers can often be found using available learning algorithms through trial and error. However, when the default models of the controlled systems possess high uncertainty, off–line trained neurocontrollers may not work well for actual systems. The off–line training approach becomes attractive when theoretical design methods are not available for such nonlinear systems.

Hybrid off–line and on–line trained neurocontrollers are the integrated results of the above two design methodologies for neurocontrollers. Off–line design is used to determine the structures and orders of the neurocontrollers, and test their validity. In addition, a priori knowledge about the controlled systems, e.g., the default models, can be incorporated into the neurocontrollers. When the neurocontrollers are applied to the actual systems, on–line learning mechanisms of the neurocontrollers are evoked to adapt the system uncertainty. The ultimate goal of neurocontrol is to remove all the off–line design work and to achieve intelligent control by pure on–line learning.

At present, most of the existing neurocontrol schemes are only suitable for off–line training of neurocontrollers, since the global convergence of the learning algorithms for neurocontrollers, and consequently, the global stability of neurocontrol systems, cannot be guaranteed. There are few stability–based neurocontrol schemes at present and they can only be applied to some feedback linearizable nonlinear systems. In this chapter, a novel self–tuning neurocontrol scheme (fig. 5.1) for nonlinear systems is proposed. It belongs to the paradigm of stability–based adaptive neurocontrol schemes, and can be indirect (explicit) and direct (implicit). It differs from the popular indirect adaptive neurocontrol schemes as shown in fig. 4.2, whose system stability cannot be proved theoretically at present. As shown in fig. 5.1, the self–tuning controllers are not represented by another neural network as in fig.4.2,

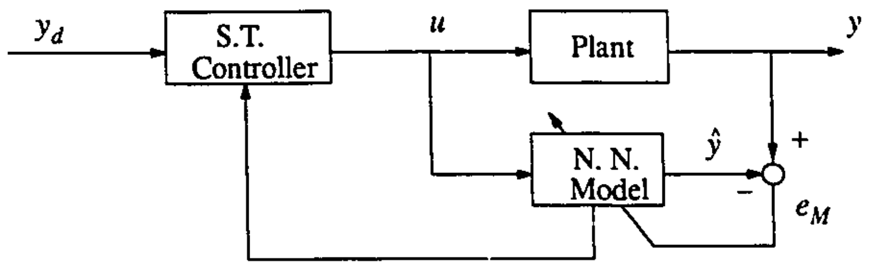


Fig. 5.1 A schematic diagram of the self–tuning neurocontrol systems

but are derived from the identified neural network models of the controlled systems. The localized polynomial networks with the competitive lateral inhibitory (CLI) cells presented in section 3.2 are used as the neural network model of the controlled system (plant). When the localized polynomial networks are trained by the adaptive extended Kalman filtering (AEKF) algorithm with recursive UD factorization, fast and global convergence of neural network models can be achieved, provided that the plant system order is given. If the localized linear networks are used, then the minimum variance (MV) control scheme, the generalized minimum variance (GMV) control scheme, the pole/zero placement (P/ZP) control scheme, the linear–quadratic–Gaussian (LQG) control scheme and the generalized predictive control (GPC) scheme in adaptive control theory (Åström and Wittenmark, 1989) can be applied to the identified neural network models with slight changes. Moreover, all the modifications developed in the past decade and the robust analysis techniques can be applied to the self–tuning neurocontrol systems. If localized higher–order polynomial networks are used, then the MV control scheme, the GMV control scheme and the GPC scheme can still be applied to the identified neural network models, by introducing the linear–in–control d–step–ahead input–output prediction models. Therefore, global stability of these self–tuning neurocontrol systems is guaranteed under the certainty equivalence principle, and is not sensitive to the initial setup of the networks.

5.2 STRUCTURE OF SELF–TUNING NEUROCONTROL SYSTEMS

Rewrite the nonlinear system represented by eqs. (4.2.1) and (4.2.2) in the following form:

$$\dot{x} = f(x,u) \tag{5.2.1}$$

$$y = h(x,u) \tag{5.2.2}$$

where $x \in R^{n_s}$, $u \in R^{m_s}$ and $y \in R^{r_s}$ are the system state vector, input vector and output vector, respectively, and $f(.,.)$ and $h(.,.)$ are unknown. The system expressed by eqs. (5.2.1) and (5.2.2) satisfies the following assumptions:

(1) it is controllable and observable in the working region Ω of x and u;

(2) $f(.,.)$ and $h(.,.)$ are continuously differentiable; and

(3) the system orders, n_s, m_s, and r_s, are known.

As shown in section 4.2, a class of finite order nonlinear systems (5.2.1) and (5.2.2) with sampled–data can be approximately discretized into the following d–step–ahead input–output prediction model:

$$y((k+d)T) = \psi(y(kT), \; . \; . \; . \; , \; y((k-\mu)T); \; u(kT), \; . \; . \; . \; , \; u((k-\nu)T) \;) \tag{5.2.3}$$

which is the same as eq. (4.2.12). Here, T is the sampling period determined by the system dynamics, the bandwidths of the actuators and control performance requirement. μ and ν are determined by the system orders, n_s, m_s, r_s, and the discretization precision requirement.

The r_I independently–controllable outputs are expressed as:

$$y_1((k+d)T) = \psi_1(y(kT), \; . \; . \; . \; , \; y((k-\mu)T); \; u(kT), \; . \; . \; . \; , \; u((k-\nu)T) \;) \tag{5.2.4}$$

Eq. (5.2.4) can be used for output tracking control problem. If measurement noise and modeling errors exist (denoted by $\varepsilon_1(kT)$), eq. (5.2.4) becomes:

$$y_1((k+d)T) = \psi_1(y(kT),. . .,y((k-\mu)T); u(kT),. . .,u((k-\nu)T)) + \varepsilon_1((k+d)T) \tag{5.2.5}$$

Assume that the $(r_s - r_l)$ non-independently-controllable outputs are BIBS (bounded input bounded state). If the system model represented by eq. (5.2.4) is unknown, but can be identified with a convergent identification algorithm, then various self-tuning controllers can be constructed based on the identified model (5.2.4) and the certainty equivalence principle. When the system model (5.2.4) is represented by a neural network, the derived controller is called the self-tuning neurocontroller (STNC).

The structure of the self-tuning neurocontrol scheme is shown in detail in fig. 5.2,

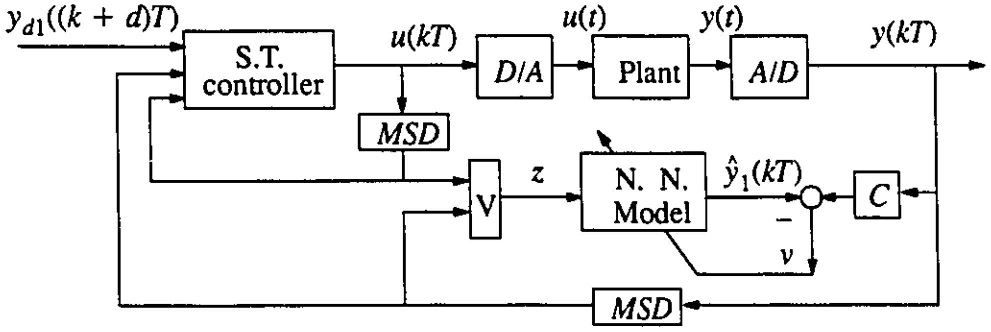


Fig. 5.2 The specified structure of self-tuning neurocontrol systems

where MSD represents the multi-step delay operations; V represents the vectorization of its inputs; C represents the constant matrix which selects the independently-controllable outputs; A/D represents the analog-to-digital converter; D/A represents the digital-to-analog converter; and $y_{d1}((k+d)T)$ is the desired output vector at $t = (k+d)T$. The input vector z to the neural network model, determined by eq. (5.2.4), is

$$z(kT) = [y^T((k-d)T). \, . \, .y^T((k-d-\mu)T) \; u^T((k-d)T). \, . \, .u^T((k-d-\nu)T)]^T \tag{5.2.6}$$

In the following, we will present solutions of the self-tuning neurocontroller, based on the localized polynomial networks with CLI cells. In the first case, localized linear networks are used to approximate the system models, and a set of linear self-tuning controllers for nonlinear systems can be easily derived based on the available adaptive control theory for linear systems. In the second case, localized higher-order polynomial networks are used to approximate the system models and to achieve similar results with fewer CLI cells and higher precision.

5.3 STNC USING LOCALIZED LINEAR NETWORKS

As shown in sections 3.2 and 3.4, eq. (5.2.4) can be approximated by the localized linear networks with sufficient number of CLI cells. Since the order of the polynomial networks is equal to one in this case, $\Phi(x)$ in eq. (3.2.3) becomes $[1 \;\; z(kT)]^T$. The whole working region Ω of the controlled system (5.2.1) and (5.2.2) is divided by the receptive fields of the CLI cells into sub–regions: $\Omega_1, \ldots, \Omega_J$. When the j–th CLI cell is excitatory, model (5.2.4) can be approximated by a biased linear system (BLS) corresponding to the j–th sub–linear network:

$$y_1((k+d)T) = \Theta_j + \sum_{i=0}^{\mu} A_{ji} y((k-i)T) + \sum_{i=0}^{\nu} B_{ji} u((k-i)T)\;) + \varepsilon_1((k+d)T) \quad (5.3.1)$$

$$= W_j^T \Phi(z(k+d)) + \varepsilon_1((k+d)T) \quad (5.3.2)$$

where $z(kT) \in \Omega_j$; Θ_j is the bias vector of the j–th sub–linear network and distinguishes the proposed scheme from the conventional linear system control; $A_{ji} \in R^{r_l \times r_s}$ and $B_{ji} \in R^{r_l \times m_s}$ are the coefficient matrices; and

$$W_j^T = [\Theta_j \; A_{j0} \ldots A_{j\mu} \; B_{j0} \ldots B_{j\nu}] \equiv [\theta_{j1}^T \ldots \theta_{jr_l}^T]^T \quad (5.3.3)$$

$\varepsilon_1((k+d)T)$ is the measurement noise and modeling error vector and is bounded, with zero means. To guarantee that the computed $u(kT)$ is bounded, B_{j0} is constrained to be of full row–rank.

Therefore, within a sub–region or receptive field Ω_j, the system models can be represented by biased linear systems, and the well–developed model reference adaptive control theory and self–tuning regulator theory can be applied to solve the adaptive control problem within each receptive field Ω_j. The overall controller consists of J sub–controllers, which have uniform structures and are integrated automatically by the CLI cells. This reflects the significance of the localized linear network models for nonlinear systems.

In the following, we only give the basic format of the GMV self–tuning neurocontrollers for nonlinear systems. Its modifications and other self–tuning control schemes can be easily derived by following the same procedure.

To determine a GMV controller, we select the following tracking error variance index:

$$I = E\{ \| P_j(q^{-1})y_1((k+d)T) - R_j(q^{-1})y_{d1}((k+d)T) \|^2 + \| Q'_j(q^{-1})u(kT) \|^2 \} \quad (5.3.4)$$

where $E\{.\}$ is the mathematical expectation operator; $\| x \|^2 = x^T x$; and

$$P_j(q^{-1}) = \sum_{i=0}^{\tau_p} P_{ji} q^{-i} \qquad R_j(q^{-1}) = \sum_{i=0}^{\tau_r} R_{ji} q^{-i} \qquad Q'_j(q^{-1}) = \sum_{i=0}^{\tau_r} Q'_{ji} q^{-i} \quad (5.3.5)$$

where $P_{ji} \in R^{r_l \times r_l}$, $R_{ji} \in R^{r_l \times r_l}$ and $Q_{ji} \in R^{m_s \times m_s}$ are constant matrices determined by the designers. q^{-1} is the backward shift operator. $P_{j0} = I$. Properly selecting P_{ji}, R_{ji}, Q_{ji} and the step numbers, $\tau_p \geq d-1$, τ_r, τ_q can make the self–tuning controller perform better and applicable to non–minimum phase nonlinear systems (Åström and Wittenmark, 1989).

To make the tracking error variance index I minimal, let

$$\frac{\partial I}{\partial u(kT)} = 2E\{B_{j0}^T[P_j(q^{-1})y_1((k+d)T) - R_j(q^{-1})y_{d1}((k+d)T)]\} + 2\, Q_{j0}^T Q'_j(q^{-1})u(kT)$$

$$= 2B_{j0}^T[P_j(q^{-1})\hat{y}_1((k+d)T) - R_j(q^{-1})y_{d1}((k+d)T)] + 2\, Q_{j0}^T Q'_j(q^{-1})u(kT) = 0$$

where $\hat{y}_1((k+d)T)$ is the optimal d–step ahead prediction of the biased linear system (3.7) and can be expressed as:

$$\hat{y}_1((k+d)T) = \Theta_j + \sum_{i=0}^{\mu} A_{ji} y((k-i)T) + \sum_{i=0}^{\nu} B_{ji} u((k-i)T) \;) \quad (5.3.6)$$

Define the following auxiliary vector:

$$\phi((k+d)T) = P_j(q^{-1})y_1((k+d)T) - R_j(q^{-1})y_{d1}((k+d)T) + Q_j(q^{-1})u(kT)$$
$$= \hat{\phi}((k+d)T) + \bar{\varepsilon}_1((k+d)T) \qquad (5.3.7)$$

where

$$Q_j(q^{-1}) = [B_{j0}B_{j0}^T]^{-1}B_{j0}Q_{j0}^TQ'_j(q^{-1}) \qquad (5.3.8)$$

$$\hat{\phi}((k+d)T) = P_j(q^{-1})\hat{y}_1((k+d)T) - R_j(q^{-1})y_{d1}((k+d)T) + Q_j(q^{-1})u(kT) \qquad (5.3.9)$$

$$\bar{\varepsilon}_1((k+d)T) = P_j(q^{-1})\varepsilon_1((k+d)T) \qquad (5.3.10)$$

Therefore, the minimization problem (5.3.4) can be solved by setting

$$\hat{\phi}((k+d)T) = 0 \qquad (5.3.11)$$

and the optimal control $u(kT)$ can be determined from eq. (5.3.11).

The unknown parameters in eq. (5.3.11) can be identified using either eq. (5.3.1) or eq. (5.3.7). The scheme derived using the parameters of identified model (5.3.1) is called the *indirect* self–tuning neurocontrol scheme. The scheme derived using the parameters of identified model (5.3.7) is called the *direct* self–tuning neurocontrol scheme.

For the indirect self–tuning neurocontrol scheme, when $z(kT) \in \Omega_j$, the parameter matrix W_j in eq. (5.3.2) can be identified using AEKF with recursive UD factorization, and used in eq. (5.3.11) to determine the control vector $u(kT)$. The resultant identification algorithm and control algorithm form the self–tuning neurocontrol algorithm. For the direct self–tuning neurocontrol scheme, $z(kT)$ has to be changed to account for the effect of $R_j(q^{-1})y_{d1}((k+d)T)$ in eq. (5.3.9). A similar self–tuning neurocontrol algorithm can be derived.

It can be seen that the main difference between the self–tuning neurocontrol scheme for general nonlinear systems and the self–tuning regulation schemes for linearly parametrizable systems is the number of sets of parameter matrices to be stored and their validity regions. For the linear regulation schemes, there is only one parameter matrix W, which is valid in the whole working region Ω. For the self–tuning neurocontrol scheme, W_j ($j = 1, 2, \ldots, J$) is valid only in its corresponding receptive field Ω_j (fig. 5.3). Thus W_j can only be estimated and used to determine the control vector $u(kT)$ when $z(kT) \in \Omega_j$. In the neurocontrol scheme, memory is introduced to save the learned input–output system dynamics. In conventional self–tuning regulation schemes, such information is lost as system states change from one sub–region to another if the system dynamics are not strictly linear.

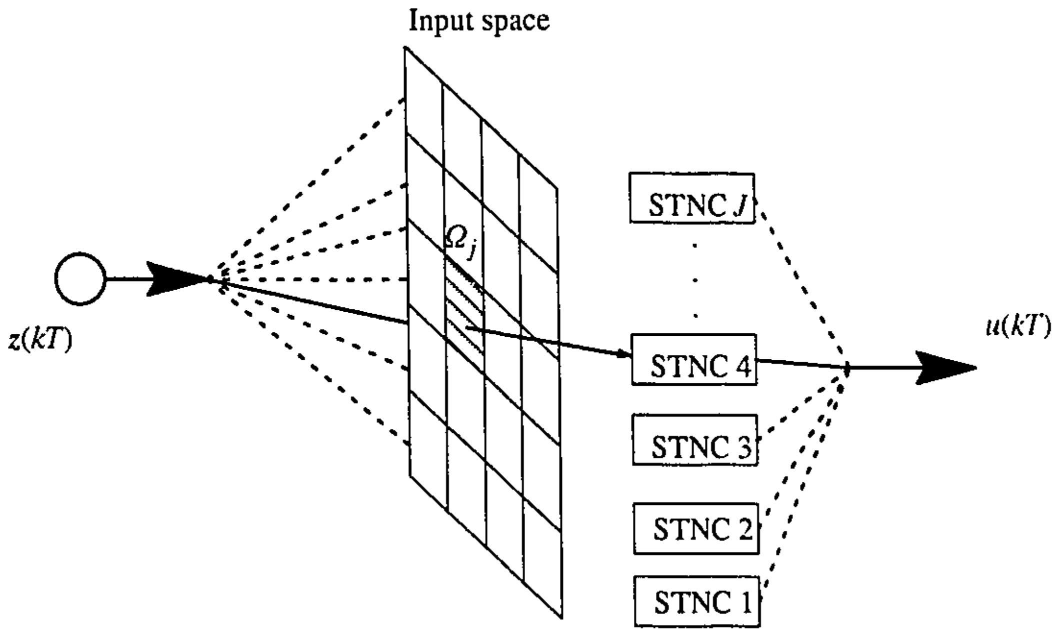


Fig. 5.3 A schematic diagram of the self–tuning neurocontrol systems

There are many available estimation algorithms to identify the parameter matrix W_j besides AEKF, such as the recursive least squares (RLS) algorithm in square–root form, the recursive maximum likelihood algorithm, the projection algorithm and the stochastic approximation algorithm. To guarantee the parameter matrix W_j to be convergent, $\Phi(z)$ in eq. (5.3.2) has to satisfy the persistent excitation condition. Moreover, to guarantee all the parameter matrices W_j ($j = 1, 2, \ldots, J$) to be convergent, the input–output data used to train the localized linear networks have to be representative of the system model (5.2.1) and (5.2.2). This requirement distinguishes the unstructured system identification from the structured one. To control a nonlinear system with time–varying parameters, forgetting–factors can be introduced into the above estimation/learning algorithms for the localized networks. When measurement noise, input noise, external disturbance, and neural network modeling errors exist, the normalized filtering, projection, leakage and dead–zone techniques (Ioannou and Sun, 1988) can be adopted to prevent the network parameter learning from drifting and bursting, and increase the robustness of the self–tuning neurocontrol systems.

Besides the GMV control scheme, the well–developed P/ZP control scheme, LQG control scheme and GPC scheme can also be adopted to achieve better control performance and better robustness to unmodeled dynamics, disturbance and noises. Detailed extensions are beyond the scope of this thesis.

In conclusion, the new self–tuning neurocontrol scheme with the localized linear networks makes the linear adaptive control theory applicable to general nonlinear systems. Thus the existing robust adaptive control schemes can be easily extended to the new scheme to improve its performance further. The global stability of the new self–tuning neurocontrol scheme can be proved similarly, as done by Ydstie (1991).

5.4 STNC USING LOCALIZED HIGHER–ORDER POLYNOMIAL NETWORKS

The localized linear networks approximate well and the receptive field Ω_j of the j–th sub–linear network can be extended by introducing forgetting factors into the estimation algorithms which make the network estimators able to track time–varying parameter changes and therefore enlarge the validity region of the BLS (5.3.1). However, many CLI cells, corresponding to the division number of the working region Ω, are still needed to guarantee high approximation precision. If the localized higher-order polynomial networks are used, then the number of CLI cells can be reduced under the same high approximation precision. The higher the order of the localized polynomial networks is, the fewer CLI cells are needed.

Due to the nonlinearity of the localized higher-order polynomial networks, P/ZP and LQG control schemes are not applicable. Nevertheless, MV, GMV, and GPC control schemes are still valid. Similarly, we present the basic format of the GMV self-tuning neuro-controllers using localized higher-order polynomial networks to represent nonlinear systems. $\Phi(x)$ in eq. (3.2.3) is defined according to the network order L. When $z((k+d)T) \in \Omega_j$, the network model can be expressed as:

$$y_1((k+d)T) = \psi_{1j}(y(kT),\ldots,y((k-\mu)T); u(kT),\ldots,u((k-\nu)T)) + \varepsilon_1((k+d)T)$$

$$= \overline{W}_j^T \Phi(z(k+d)) + \varepsilon_1((k+d)T) \qquad (5.4.1)$$

where $\overline{W}_j$ contains more terms than W_j, defined by eq. (5.3.3), does. If eq. (5.4.1) is used directly to derive GMV (or MV) control schemes, then some nonlinear equations have to be solved on-line to determine $u(kT)$, which may result in multi-solutions and increase on-line computation.

If system (5.2.1) and (5.2.2) can be approximated by localized linear networks, it should be approximated by the following modified localized higher-order polynomial networks representing the linear-in-control d-step-ahead input-output prediction model:

$$y_1((k+d)T) = B_{j0}u(kT) + \overline{\psi}_{1j}(y(kT),\ldots,y((k-\mu)T);\; u((k-1)T),$$

$$\ldots, u((k-\nu)T)) + \varepsilon_1((k+d)T) \qquad (5.4.2)$$

$$= \underline{W}_j^T \Phi(z(k+d)) + \varepsilon_1((k+d)T) \qquad z((k+d)T) \in \Omega_j \qquad (5.4.3)$$

where $\overline{\psi}_{1j}(\cdot)$ is a function to be determined, and $B_{j0} \in R^{r_l \times m_l}$ is a constant matrix with its rank $= r_j$. $\varepsilon_1((k + d)T)$ contains the representation error and the output measurement noise.

It is worth mentioning that the modified localized higher–order polynomial networks are still universal approximators. Due to their locality, eq. (5.4.2) is valid only in the local receptive field Ω_j, which can be adjusted by controlling the number of the CLI cells. However, if global instead of localized networks are adopted, using model (5.4.2) does mean losing of their representation ability. For example, $u^2(kT)$ cannot be well approximated by $au(kT)$ for $u(kT) \in [-100, 100]$, but can be for $u(kT)$ belonging to any small interval (a is a parameter to be determined).

The optimal d–step ahead prediction $\hat{y}_1((k + d)T)$ of system (5.4.2) can be expressed as:

$$\hat{y}_1((k + d)T) = B_{j0}u(kT) + \overline{\psi}_{1j}(y(kT), \ldots, y((k - \mu)T); u((k - 1)T), \ldots, u((k - \nu)T))$$

$$= \underline{W}_j^T \Phi(z(k + d)) \tag{5.4.4}$$

Define:

$$q^{-i}\overline{\psi}_{1j}(y(kT), \ldots, y((k - \mu)T);\ u((k - 1)T), \ldots, u((k - \nu)T)) =$$

$$\overline{\psi}_{1j}(y((k - i)T), \ldots, y((k - \mu - i)T);\ u((k - 1 - i)T), \ldots, u((k - \nu - i)T))$$

Then using the same tracking error variance index (5.3.4) and the same auxiliary vector $\phi((k + d)T)$ as defined in eq. (5.3.7), the optimal control $u(kT)$ can be determined in a similar manner from eq. (5.3.11). The previous discussion about the estimation/learning algorithms is also applicable here.

If the localized neural network model with the specified structure can approximate the controlled plant model (5.2.1) and (5.2.2), the stability of the self–tuning neurocontrol systems can be guaranteed and the new scheme works for non–minimum–phase NLS. The more knowledge there is about the controlled plant model, the more confident we are in determining the structures of the localized polynomial network models.

As a special case, let $P_j(q^{-1}) = R_j(q^{-1}) = I$, and $Q'_j(q^{-1}) = 0$. Then we can obtain the MV control scheme by determining $u(kT)$ from eq. (5.4.5).

$$\hat{y}_1((k+d)T) - y_{d1}((k+d)T) = 0 \tag{5.4.5}$$

5.5 SELF–TUNING SLIDING NEUROCONTROL SCHEME

In the previous section, we have applied the existing adaptive control schemes for linear systems to self–tuning neurocontrol of nonlinear systems. In this section, we present a self–tuning sliding neurocontrol scheme using localized polynomial networks, since its controller parameters are easy to determine and the system stability is easy to analyze.

For the given desired output trajectory $y_{d1}(k)$ $(k = 1, 2, \dots)$, define the tracking error vector of the r_I independently–controllable outputs as follows:

$$e_1(k) = y_{d1}(k) - y_1(k) \tag{5.5.1}$$

When $z(k) \in \Omega_j$, the tracking error model of a nonlinear system represented by eq. (5.3.2) or (5.4.4) is:

$$e_1(k) = y_{d1}(k) - \underline{W}_j^T \Phi(z(k)) - \varepsilon_1(k) \tag{5.5.2}$$

The sliding surface for the tracking error model (5.5.2) can be defined as:

$$s(k) = e_1(k) - Ce_1(k-d) \tag{5.5.3}$$

or

$$s(k) = e_1(k) - \sum_{i=d}^{\mu+d} C_i e_1(k-i) \tag{5.5.4}$$

where C or C_i are chosen to satisfy the following two conditions: 1) making $s(k) = 0$ an asymptotically stable motion; and 2) ensuring that the derived sliding control laws are realizable, even when system (5.2.1) and (5.2.2) is non–minimum phase. Theoretically, both eqs. (5.5.3) and (5.5.4) are feasible. However, eq. (5.5.3) corresponds to a reduced–order system response and requires more control effort to achieve such a result. Eq. (5.5.4) corresponds to the system response with the same order as the system has and requires less control effort to achieve such effect, but it is complex.

If eqs. (5.3.2) or (5.4.4), realized by localized polynomial networks, can approximate system (5.2.1) and (5.2.2) with bounded small errors, then the approaching law of the sliding control scheme can be simply chosen as:

$$s(k + d) = K\ s(k) \tag{5.5.5}$$

and $K = diag\{K_1, \ . \ . \ . \ , \ K_{r_l}\}$ with $|K_i| < 1$. Obviously, the approaching law itself is asymptotically stable.

From eqs. (5.4.4) and (5.5.4), we have:

$$s(k + d) = e_1(k + d) - \sum_{i=0}^{\mu} C_i e_1(k - i)$$

$$= y_{d1}(k + d) - B_{j0}u(k) - \overline{\psi}_{1j}(z(k)) - \sum_{i=0}^{\mu} C_i e_1(k - i) - \varepsilon_1(k + d) \tag{5.5.6}$$

Substituting eq. (5.5.6) into eq. (5.5.5) and solving for $u(k)$, we have:

$$u(k) = B_{j0}^T[B_{j0}B_{j0}^T]^{-1}\{y_{d1}(k + d) - \overline{\psi}_{1j}(z(k)) - \sum_{i=0}^{\mu} C_i e_1(k - i) - K\ s(k) - \hat{\varepsilon}_1(k + d)\} \tag{5.5.7}$$

where $\hat{\varepsilon}_1(k + d)$ is the estimation of the mean values of $\varepsilon_1(k + d)$. If no a priori information about $\varepsilon_1(k + d)$ is given, simply set $\hat{\varepsilon}_1(k + d) = 0$.

By properly designing the localized polynomial network structure, it is easy to make $\| \varepsilon_1(k + d) \|$ less than a specified small constant M_0. Therefore, it can be proved that the neurocontrol law (5.5.7) makes the tracking error vector, $e_1(k)$, of the r_l independently–controllable outputs globally converge to the neighborhood of zero. The "static" tracking errors are proportional to M_0, and can, therefore, be reduced to a specified level. Due to the modular and additive structure of localized polynomial networks, the self–tuning neurocontrollers can be extended easily to new input space without affecting the learned input–output relations, and they degrade gracefully when there is a sub–network failure.

5.6 SIMULATION TEST

In this section, the preceding theoretical development is applied to the following system (Mukhopadhyay and Narendra, 1993) as an example:

$$y(k+1) = \frac{\cos\{y(k)\}}{1+\sin^2\{y(k)\}} + \sin\{y(k)\}\exp\{-y^2(k)\} + \frac{y(k)}{1+y^2(k)} + u(k) \quad (5.6.1)$$

where $u(k)$ is the input variable, and $y(k)$ is the output variable. Assume that we do not know the system model (5.6.1) except for its order and delay steps. To simplify the simulation, we assume that the linear–in–control structure of the system is known. Now the problem is to find the following input–output model to represent the input–output relation of the actual system:

$$\hat{y}(k+1) = N[\ y(k),\ u(k)\ ] = \overline{N}[\ y(k)] + bu(k) \quad (5.6.2)$$

where $N[\ .\ ]$ is realized by a localized polynomial neural network with CLI cells.

Let the order of each sub–network be $L = 2$, and the number of the CLI cells be $J = 5$. The centers of the CLI cells are randomly distributed in the interval $[-4, 4]$. The number of hidden neurons in each sub–polynomial neural network is $p = 4$, which corresponds to a modest computation at each iteration. Before training, the initial weights and thresholds are set randomly between –1 and 1. The desired output trajectory is

$$y_d(k) = 0.25(\sin\{\frac{2\pi k}{10}\} + \sin\{\frac{2\pi k}{25}\}) \quad (5.6.3)$$

Fig. 5.4 shows the transient time response profile of the self–tuning neurocontrol system. Off–line training of the localized network was not needed. It can be seen that after 100 iterations of on–line learning, the tracking error $e(k) = y_d(k) - y(k)$ converges to the neighborhood of zero. Increasing the number of CLI cells can further decrease the size of this neighborhood. Setting different initial values of $y(k)$, weights and thresholds give similar results. This verifies the fast and global convergence of the new self–tuning neurocontrol scheme.

If the same control task of system (5.6.1) is performed using the popular indirect adaptive neurocontrol scheme as shown in fig. 4.2, and sigmoidal neural networks or localized higher–order polynomial networks are used as the neurocontroller and system model, then only local stability of the control system can be achieved.

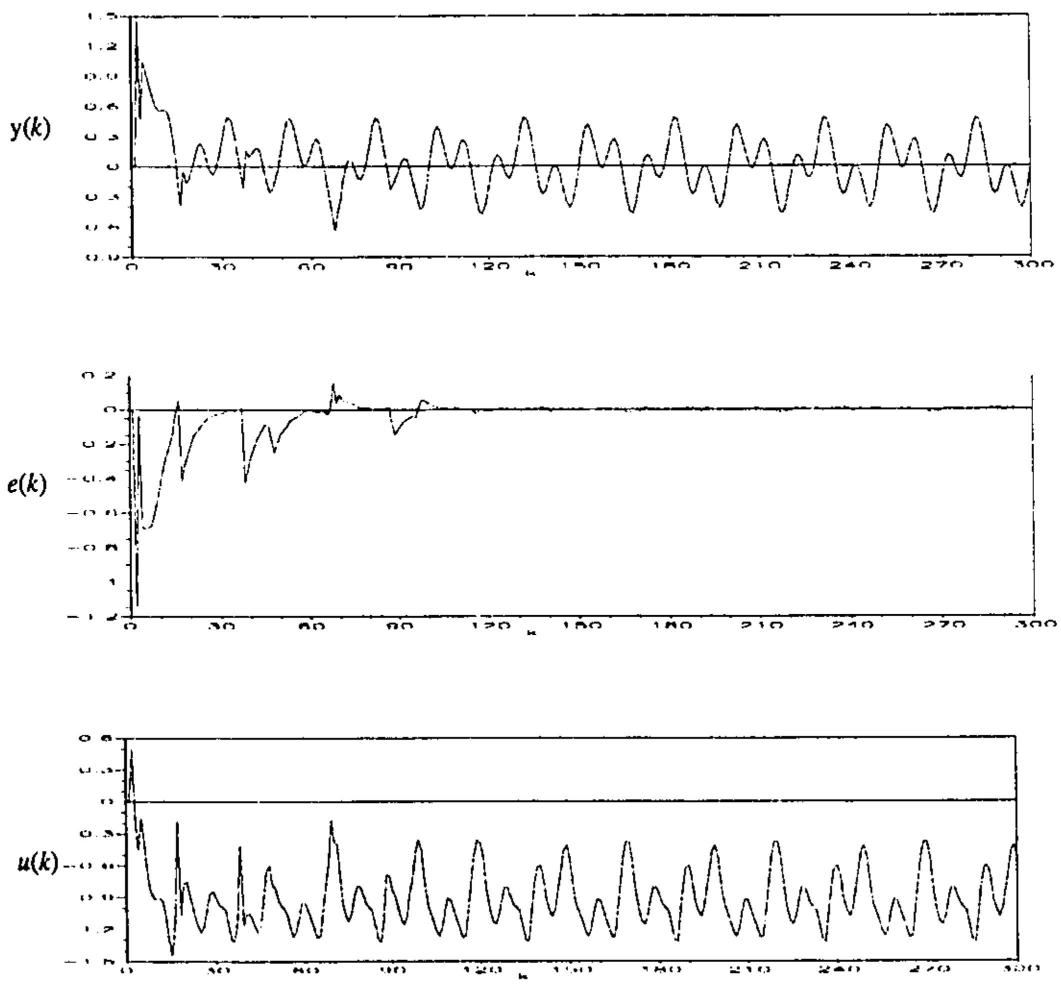


Fig. 5.4 The control system time responses under trajectory (5.6.3)

Let the conditions set for the previous simulation remain the same, except that the desired output trajectory is changed to:

$$y_d(k) = 2(\sin\{\frac{2\pi k}{10}\} + \sin\{\frac{2\pi k}{25}\}) \tag{5.6.4}$$

which means a larger working region for the control system. Similar good results were also obtained, as shown in fig. 5.5. This clearly shows that the self–tuning neurocontrollers can be extended easily to new input space without affecting the learned input–output relations, due to the modular and additive structure of the localized polynomial networks. Therefore, with the network structure determination algorithm given in section 3.2, it should not be a concern whether the system states go beyond the pre–defined working region or not, since the new self–tuning neurocontrol scheme with localized polynomial networks also possesses self–organizing ability.

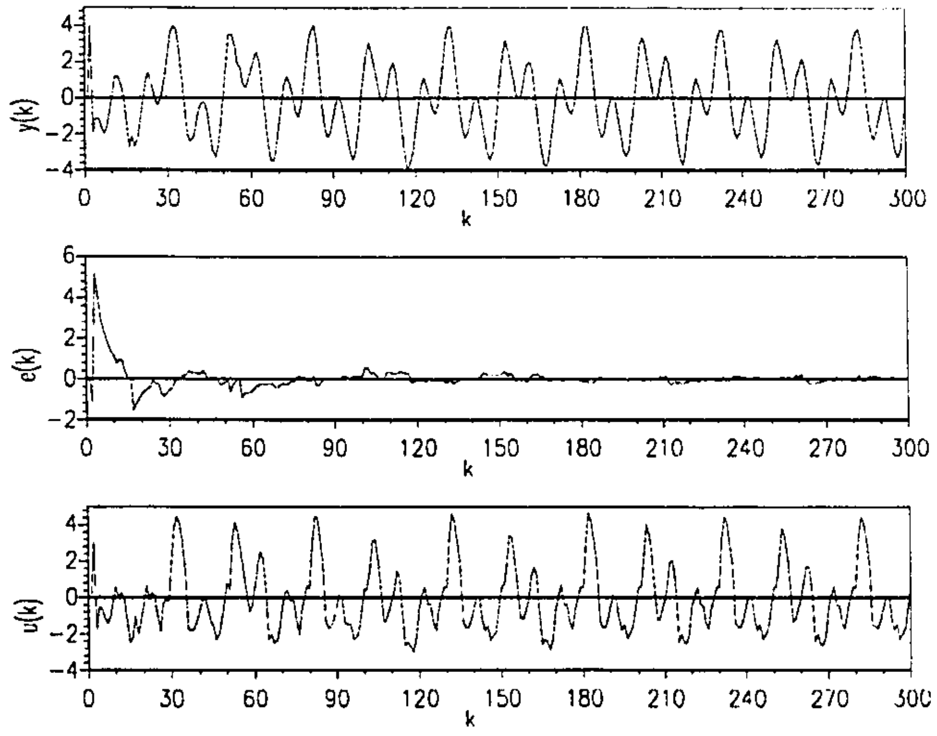


Fig. 5.5 The control system time responses under trajectory (5.6.4)

5.7 SUMMARY

This chapter presents a novel self–tuning neurocontrol scheme for nonlinear systems. It makes the self–tuning neurocontrol of nonlinear systems closely related to the maturing adaptive control theory for linear systems. If the localized linear networks are used to model the controlled nonlinear systems, then the well–known MV, GMV, LQG, P/ZP and GPC schemes can be directly applied to the design of self–tuning neurocontrollers. If the localized higher–order polynomial networks are used to model the controlled nonlinear systems, then the MV, GMV, and GPC schemes can still be applied to designing self–tuning neurocontrollers. A sliding neurocontroller version of this scheme is also presented. Due to the modular and additive structure of localized polynomial networks, the self–tuning neurocontrollers can be extended easily to new input space without affecting the learned input–output relations, and they degrade gracefully when there is a sub–network failure. Due to the unknown structures of the controlled systems and the neural networks with finite neurons, there still exist some small bounded unmodeled dynamics in the system models. Therefore, the adaptive neurocontrol systems must be robust to such model errors and all the available modifications of the adaptive control laws developed in the last decade can be used here to further improve the performance of the self–tuning neurocontrol systems. Simulation results confirm the above theory.

CHAPTER 6

GLOBALLY STABLE ADAPTIVE NEURO–CONTROL

6.1 INTRODUCTION

As indicated in previous sections, there are few stability–based adaptive neurocontrol schemes currently available. The existing schemes can only be applied to some feedback linearizable nonlinear systems (e.g., Chen, 1991, Sanner and Slotine, 1992). The reason is that there are few available control schemes for nonlinear systems.

Feedback linearization (e.g., Isidori, 1989) represents a major approach in the nonlinear control field. Its main idea is to find a nonlinear diffeomorphism that transforms a nonlinear system into a linear system. Recently, the idea of feedback linearization has been extended to transform a nonlinear system into another nonlinear system (the so–called canonical form) that is solvable in control design. Also, their robust control (e.g., Spong and Vidyasagar, 1989) and adaptive control modifications (e.g., Sastry and Isidori, 1989, Marino, Kanellakopoulos, and Kokotovic, 1989) have been proposed to account for modeling errors and to improve control performance. Only a small class of nonlinear systems can be transformed into linear systems. Those systems are called feedback linearizable nonlinear systems. For high-order nonlinear systems, the outer-loop linear feedback gains, which become the coefficients of system characteristic polynomials, are too large to realize if fast convergent error dynamics are required. This is because the canonical form of equivalent linear systems is in cascaded integrator form. Also, the inverse nonlinear transformation may not be global. The requirement that nonlinearities in system dynamics must be infinitely differentiable, or form the so-called "smooth vector field", excludes most of the practical nonlinear systems which contain dead-zones, saturation, hystereses, backlash, etc.

In this chapter, the variable index control approach for general nonlinear systems is proposed. It guarantees the global stability of the control systems for both stabilization and tracking problems. It possesses inherent robustness to system model uncertainty. The robustness is due to the worst-case approaching laws, not sign functions of the sliding functions as in variable structure systems. It does not impose any growth conditions and infinite differentiability assumptions on the system nonlinearity. It can be applied to nonlinear systems that are not feedback-linearizable, and uncertain linear systems with time-varying and state-dependent parameters. The control actions are chatter-free. The variable index control approach can be used to derive stability-based adaptive neurocontrol schemes.

Compared with variable structure control (VSC) (Utkin, 1976, 1977, Asada and Slotine, 1986), the variable index control approach does not suffer from: a) possible chattering during sliding motion, or b) the difficulty in determining a feasible sliding surface for general nonlinear multivariable systems that guarantees global stability of the closed-loop systems. In fact, these two problems of VSC motivated our research on variable index control. Compared with the backstepping approach (Kanellakopoulos, Kokotovic and Morse, 1991), the variable index control approach can solve more general problems that cannot be solved by backstepping. The Lyapunov-based control approach can provide a nonlinear control law with proven system stability. For a given nonlinear system, constructing a feasible Lyapunov function is challenging work.

The rest of this chapter is organized as follows. Section 6.2 presents the variable index control theory. The global stability of the variable index control systems is proved. Examples and simulations are presented to demonstrate the design procedure. Section 6.3 extends it to general nonlinear systems. Section 6.4 derives a global stable adaptive neurocontrol scheme for general nonlinear systems. Section 6.5 summarizes the contributions of this chapter.

6.2 VARIABLE INDEX CONTROL THEORY

6.2.1 Preliminaries

Consider a general affine nonlinear system

$$\dot{x}(t) = A(x(t)) + B(x(t))\ u(t) \qquad (6.2.1)$$

where $x(.) \in R^n$ is the system state vector; $u(.) \in R^m$ is the control vector; and $B(x) \in R^{nxm}$; n and m are known. The definitions of $A(x)$ and $B(x)$ are assumed to guarantee the existence of solutions of the differential equations (6.2.1).

The assumptions about the nonlinear system (6.2.1) are:

a) It is stabilizable/controllable (Isidori, 1989);

b) The state–variable measurements are available;

c) $A(0)=0$, and $B(x) \neq 0$ for $\forall x \in R^n$;

d) $B(x) = B\sigma(x)$ is known, where B is an $n \times m$ constant matrix, and $\sigma(x)$ is an $m \times m$ invertible matrix for $\forall x \in R^n$; and

e) The system model (6.2.1) is of bounded uncertainty in $A(x)$:

$$a_{Li}(x) \leq a_i(x) \leq a_{Ui}(x) \qquad \text{for } i = 1, \ldots, n \qquad (6.2.2)$$

where $a_i(x)$ is the i–th entry of $A(x)$, and $a_{Li}(x)$ and $a_{Ui}(x)$ are known and bounded functions as long as $\| x \|$ is bounded.

Most of the nonlinear control approaches, including feedback linearization, require the first three assumptions in addition to other assumptions. Assumption d) represents a restriction on $B(x)$. However, it will be shown in section 6.3 that general nonlinear system $\dot{x} = f(x, u)$ can be transformed into the form with $B(x)=B$. Assumption e) gives the definition of the system uncertainty, which is not required to satisfy any matching condition. In section 6.3, $B(x)$ is also allowed to be uncertain. Thus, the proposed variable index control imposes the least strict assumptions about the controlled nonlinear systems, so it can be applied to more general nonlinear systems.

Define a function set $\mathcal{K}$ as follows:

$\mathcal{K}=\{ a(.) \mid a : \Re^+ \mapsto \Re^+$, *continuous, strictly increasing with* $a(0)=0$ $\}$

Lemma 6.2.1:

Given vectors $a=[a_1, \ldots, a_n]^T$ *and* $b(p)= [b_1(p), \ldots, b_n(p)]^T$, *where* $b_i(p)$ *is a function of vector p and* a_i *is not. The entries of b(p) satisfy the following constraints for any possible p:*

$$b_i^L \leq b_i(p) \leq b_i^U \qquad \text{for } \; i = 1, \ldots, n \tag{6.2.3}$$

Then the upper bound of a^Tb *as a function of p is* $a^T\bar{b}$. *That is,*

$$\max_p a^Tb \leq a^T\bar{b} = \sum_{i=1}^{n}[(b_i^U + b_i^L)a_i + |a_i|(b_i^U - b_i^L)]/2 \tag{6.2.4}$$

where $\bar{b} = [\bar{b}_1, \ldots, \bar{b}_n]^T$ *and:*

$$\bar{b}_i = [b_i^U + b_i^L + sgn(a_i)(b_i^U - b_i^L)]/2 = \begin{cases} b_i^U & \text{if } \; a_i \geq 0 \\ b_i^L & \text{if } \; a_i < 0 \end{cases} \quad i = 1, \ldots, n \tag{6.2.5}$$

***Proof*:** the proof is straightforward.

Lemma 6.2.1 is important in deriving the variable index control laws.

6.2.2 The worst–case approaching control strategy

It is well known that (1) variable structure control laws are robust to bounded uncertainty, and (2) the robustness is attributed to the sign function of a sliding function. There is a sliding motion after the system states remain on sliding surfaces. However, generating sliding motion is not the only way to achieve robustness. Actually, due to the introduction of the sliding motion, chattering is unavoidable when pure sign function is used. In the following we introduce the worst–case approaching control strategy. The robustness of the control systems using the worst–case approaching control strategy comes from the upper bound estimation of the time derivative of error index functions. The stabilization problem is considered first.

Let $\Delta = \{a_i(x),\ i = 1,2,\ldots,n\}$ represent the uncertainty of the system model (6.2.1). Let the error index $s(x)$ be a continuously differentiable positive definite function. $s(x) \geq a(\| x \|)$, where $a(.) \in \mathcal{K}$. For $s(x)$, we have:

$$\dot{s}(x) = \left[\frac{\partial s}{\partial x}\right]^T \dot{x} = \left[\frac{\partial s}{\partial x}\right]^T A(x) + \left[\frac{\partial s}{\partial x}\right]^T B(x)\ u \qquad (6.2.6)$$

The problem now is to determine a control law $u = u(x,s)$ under system uncertainty Δ which makes $\dot{s} < 0$ for $\forall x \in R^n$ and $x \neq 0$. If such a control law exists, then we can prove the system global stability easily, by choosing $V = s$ as the Lyapunov candidate.

To account for the system uncertainty, we propose the following worst–case approaching control law for the chosen error index:

$$\max_{\Delta} \dot{s} \leq -f(s) < 0 \qquad (6.2.7)$$

where $f(s)$ determines the convergent speed of s. Following are some feasible choices of $f(s)$:

$$a.\ \ f(s) = k \qquad k > 0\ \ is\ a\ constant \qquad (6.2.8)$$

b. $f(s) = ks \qquad k > 0$ *is a constant or a function of s* (6.2.9)

c. $f(s) = k_1 s + k_2 \int s(x)\, dt \qquad k_1,\ k_2 > 0$ (6.2.10)

From eqs. (6.2.2), (6.2.6) and lemma 6.2.1 we know that:

$$\max_{\Delta} \dot{s}(x) \le \left[\frac{\partial s}{\partial x}\right]^T [\bar{A}(x) + B(x)u] \tag{6.2.11}$$

where:

$$\bar{A}(x) = [\bar{a}_1(x)\ .\ .\ .\ \bar{a}_n(x)]^T \tag{6.2.12}$$

$$\bar{a}_j(x) = \{a_{Uj}(x) + a_{Lj}(x) + sgn\{\frac{\partial s}{\partial x_j}\}[a_{Uj}(x) - a_{Lj}(x)]\}/2 \tag{6.2.13}$$

and $sgn(.)$ is the sign function. It is worth pointing out that although eq. (6.2.13) contains $sgn\{\frac{\partial s}{\partial x_j}\}$, the right hand side of eq. (6.2.11) does not, due to the multiplication of $\frac{\partial s}{\partial x}$ and $\bar{A}(x)$. As a special case, if $a_{Lj}(x) = -a_{Uj}(x)$ in eq. (6.2.2), eq. (6.2.13) becomes:

$$\bar{a}_j(x) = sgn\{\frac{\partial s}{\partial x_j}\}a_{Uj}(x) \qquad and \qquad |a_j(x)| \le a_{Uj}(x) \tag{6.2.14}$$

Let:

$$\left[\frac{\partial s}{\partial x}\right]^T \bar{A}(x) + \left[\frac{\partial s}{\partial x}\right]^T B(x)\ u = -f(s) \tag{6.2.15}$$

or

$$Q(x)u = R(x,s) \tag{6.2.16}$$

where

$$Q(x) := \left[\frac{\partial s}{\partial x}\right]^T B(x) \qquad R(x,s) := -\left[\frac{\partial s}{\partial x}\right]^T \bar{A}(x) - f(s) \tag{6.2.17}$$

Notice that $Q(x) \in R^{1\times m}$. When $m > 1$, eq. (6.2.16) has multiple solutions of u. Therefore, a suitable solution of u can always be derived from eq. (6.2.16), especially when the designers know the dynamics of the systems well. In the following, a least square solution of eq. (6.2.16) is given. Define $\| Q(x) \| = \sqrt{Q(x)Q(x)^T}$. When $\|Q(x)\| > 0$, we can obtain:

$$u = Q^T(x)\ [Q(x)\ Q^T(x)]^{-1}\ R(x,s) \qquad (6.2.18)$$

Eq. (6.2.18) is called the least–squares variable index control law. It is robust to both large system structural and parametric uncertainty.

Since $Q(x)$ is a function of $x \in R^n$ which is changing all the time, there may exist a set in R^n such that $\|Q(x)\| = 0$. Define this set as:

$$\Omega = \{\ x \mid \|Q(x)\| = 0,\ x \in R^n\ \} \qquad (6.2.19)$$

Ω is called the singularity set of $Q(x)$. When $x \in \Omega$, u cannot be determined by eq. (6.2.18). This problem will be solved later by introducing feasible multiple error indices.

As long as $x \notin \Omega$, we can prove that the least–squares variable index control law (6.2.18) makes $\| x \|$ decrease uniformly under uncertainty Δ, by choosing $V = s(x)$ as the Lyapunov candidate.

Property 6.2.1:

Let $f(s)$ be a continuous function of s. If $a_{Li}(x)$ and $a_{Ui}(x)$, the upper and lower bounds of $a_i(x)$, are continuous functions of x, then $R(x,s)$ is a continuous vector function of x. Control law (6.2.18) is also a continuous vector function for $x \notin \Omega$.

Proof:

$R(x,s)$ is a scalar function. From eq. (6.2.17), we know:

$$R(x,s) = -\sum_{j=1}^{n} \left[\frac{\partial s}{\partial x_j}\right] \bar{a}_j(x) - f(s)$$

where $\bar{a}_j(x)$ is defined by eq. (6.2.13). Thus,

$$R(x,s) = -\sum_{j=1}^{n} \{ \frac{\partial s}{\partial x_j}[a_{Uj}(x) + a_{Lj}(x)] + \left|\frac{\partial s}{\partial x_j}\right|[a_{Uj}(x) - a_{Lj}(x)]\}/2 - f(s)$$

Since $s(x)$ is continuously differentiable, $\frac{\partial s}{\partial x_j}$ and its absolute value are both continuous functions of x. Therefore, $R(x,s)$ is a continuous function of x, using the given assumptions. Since $Q(x)$ is a continuous function matrix of x, control law (6.2.18) is also a continuous vector function for $x \notin \Omega$.

The continuity of control inputs is a desirable feature, since it ensures smooth control.

6.2.3 Existence of feasible multiple error indices:

As indicated above, the variable index control law (6.2.18) is not defined when $x \in \Omega$. Since Ω could be a large set in R^n, $\| Q(x) \| = 0$ does not imply that $x = 0$ or x is in the neighborhood of the origin. This means that control law (6.2.18) with one error index may not stabilize system (6.2.1) if $x \in \Omega$. To eliminate the singularity set Ω, the concept of feasible multiple error indices is introduced. It will be proved in the following that there always exists a set of feasible multiple error indices for $B(x) = B\sigma(x)$ which makes the variable index control law (6.2.18) well defined in the whole space R^n.

Let the error indices $s_1(x), \ . \ . \ . \ , \ s_\mu(x)$ be a set of continuously differentiable positive definite functions. $s_i(x) \geq a(\| x \|)$, where $a(.) \in \mathcal{K}$. The corresponding $Q(x,s_i)$ and $R(x,s_i)$ $(i = 1, \ldots, \mu)$ are defined similar to eq. (6.2.17), and are rewritten in the following for convenience:

$$Q(x,s_i) = \left[\frac{\partial s_i}{\partial x}\right]^T B(x) \qquad R(x,s_i) = -\left\{\left[\frac{\partial s_i}{\partial x}\right]^T \overline{A}_i(x) + f_i(s_i)\right\} \qquad (6.2.20)$$

where $\overline{A}_i(x)$ is defined similar to eq. (6.2.12), and $f_i(s_i)$ can be chosen from eqs. (6.2.8)-(6.2.10). The singularity set is:

$$\Omega_i = \{ \ x \mid \| Q(x,s_i) \| = 0, \quad x \in R^n \ \} \qquad (6.2.21)$$

The corresponding variable index control algorithms are:

$$u = Q(x,s_i)^T R(x,s_i)/[Q(x,s_i)Q(x,s_i)^T] \qquad \text{if } x \notin \Omega_i \tag{6.2.22}$$

As indicated by Property 6.2.1, u is a continuous vector function of x for each index. When $\| Q(x,s_i) \| > 0$ for any $i = 1, \ldots, \mu$, control law (6.2.22) can be applied to the system (6.2.1) and make $\| x \|$ decrease all the time. When $\| Q(x,s_i) \| = 0$ for all $i = 1, \ldots, \mu$, μ constraints on state vector x are acquired as follows:

$$\| Q(x,s_1) \| = 0 \ , \ \ldots , \ \| Q(x,s_\mu) \| = 0 \tag{6.2.23}$$

Eq. (6.2.23) is equivalent to the following:

$$Q_M(x) = [Q(x,s_1) \ldots Q(x,s_\mu)]^T = 0 \in R^{\mu m \times 1} \tag{6.2.24}$$

For a given number μ, if $x = 0$ is the unique solution of eq. (6.2.24), then $s_1(x), \ldots, s_\mu(x)$ are called a set of *feasible error indices with respect to* $B(x) = B\sigma(x)$. If μ is the minimum number for $s_1(x), \ldots, s_\mu(x)$ to be feasible, then $s_1(x), \ldots, s_\mu(x)$ are also called *independent.*

The minimum number of feasible multiple error indices required for a given system is dependent on $B(x)$, the number of control inputs and the number of state variables. It is worth pointing out that the choice of feasible multiple error indices is not relevant to $A(x)$. In the following, we will show the existence of feasible multiple error indices for nonlinear systems (6.2.1), and prove that $s_i(x) = x^T P_i x$ $(i = 1, 2, \ldots, \mu)$ are feasible multiple error indices for $B(x) = B\sigma(x)$, and that μ can be determined analytically. Here P_i is a symmetric positive definite matrix.

Substituting $s_i(x) = x^T P_i x$ and $B(x) = B\sigma(x)$ into eq. (6.2.20) gives:

$$Q(x,s_i) = \left[\frac{\partial s_i(x)}{\partial x} \right]^T B(x) = 2x^T P_i B\sigma(x) \tag{6.2.25}$$

To make a set of error indices feasible, we have to ensure that $x = 0$ is the unique solution of eq. (6.2.24), i.e., the following equations:

$$\| Q(x, s_i) \| = 0 \quad or \quad Q(x, s_i) = 0 \quad for\ all\ i = 1, 2, \ldots, \mu \tag{6.2.26}$$

Since

$$Q(x, s_i) = 2x^T P_i B\sigma(x) = 0 \quad \Leftrightarrow \quad x^T P_i B = 0 \tag{6.2.27}$$

we obtain:

$$\left\{ \begin{array}{l} x^T P_1 B = 0 \\ \quad \cdots \\ x^T P_\mu B = 0 \end{array} \right. \quad or \quad \left[P_1 B \ \ldots \ P_\mu B \right]^T x \equiv Q_c x = 0 \tag{6.2.28}$$

It can be seen that if $rank(Q_c) = n$, then $x = 0$ is the unique solution of eq. (6.2.28), thus of eq. (6.2.24). There are many ways to make $rank(Q_c) = n$, because we can choose P_i ($i = 1, 2, \ldots, \mu$) freely. For example, let G be a symmetric positive definite matrix, and $\{G, B\}$ be controllable. Let:

$$P_1 = I, \ P_2 = G, \ \ldots, P_\mu = G^{\mu - 1} \tag{6.2.29}$$

where μ is equal to the controllability index. It is easy to prove that $rank(Q_c) = n$ for the above choice. If the structure information about matrix B is utilized, then even better choices of P_i ($i = 1, 2, \ldots, \mu$) can be obtained.

In summary, $s_i(x) = x^T P_i x$ ($i = 1, 2, \ldots, \mu$) are feasible multiple error indices for nonlinear system (6.2.1) with $B(x) = B\sigma(x)$. If $x \neq 0$, there always exists an index i such that $\| Q(x, s_i) \| > 0$, which guarantees that the control vector defined by eq. (6.2.22) exists in the whole space R^n except where $x = 0$. Also, $\Omega_1 \cap \Omega_2 \cap \cdots \cap \Omega_\mu = \{0\}$.

6.2.4 Stability theorem of variable index control systems

Since we can always find a set of feasible error indices for nonlinear system (6.2.1), the proposed variable index control approach can be applied to system (6.2.1). The overall result is summarized in the following theorem.

Theorem 6.2.1:

Under the given assumptions a) – e) about the nonlinear system (6.2.1), the variable index control law (6.2.22) with a selected sequence of indexes from i = $1, 2, \ldots, \mu$ guarantees that: 1) the control signals are uniformly bounded, 2) x = 0 is a globally and asymptotically stable motion of system (6.2.1), and 3) the control system is robust to model uncertainty.

Proof:

There always exists a set of feasible error indices $s_i(x) = x^T P_i x$ $(i = 1, 2, \ldots, \mu)$ for nonlinear systems with $B(x) = B\sigma(x)$. If $x \neq 0$, there always exists an index i such that $\| Q(x, s_i) \| > 0$ for $x \notin \Omega_i$, which guarantees that the control vector defined by eq. (6.2.22) exists and is bounded for $x \notin \Omega_i$. Let the Lyapunov function candidate be $V = s_i(x) = x^T P_i x \geq \alpha_i \| x \|^2$, where α_i is a positive constant. Therefore, V is globally positive definite. The time derivative of V is

$$\dot{V} = \dot{s}_i(x) = \left[\frac{\partial s_i}{\partial x}\right]^T [A(x) + B(x)u]$$

$$= \left[\frac{\partial s_i}{\partial x}\right]^T [A(x) + B(x)Q(x,s_i)^T R(x,s_i)/[Q(x,s_i)Q(x,s_i)^T]$$

$$= \left[\frac{\partial s_i}{\partial x}\right]^T A(x) - Q(x,s_i)Q(x,s_i)^T/[Q(x,s_i)Q(x,s_i)^T]\{\left[\frac{\partial s_i}{\partial x}\right]^T \overline{A}_i(x) + f_i(s_i)\}$$

$$= \left[\frac{\partial s_i}{\partial x}\right]^T [A(x) - \overline{A}_i(x)] - f_i(s_i)$$

It is easy to prove that:

$$\left[\frac{\partial s_i}{\partial x}\right]^T [A(x) - \overline{A}_i(x)] \leq 0 \qquad \textit{if} \quad x \neq 0$$

using Lemma 6.2.1. Therefore,

$$\dot{V} = \left[\frac{\partial s_i}{\partial x}\right]^T [A(x) - \bar{A}_i(x)] - f_i(s_i) \leq -f_i(s_i) = -f_i(V) < 0 \qquad \text{if} \quad x \neq 0$$

From the Lyapunov theorem, we know that the variable index control law (6.2.22) with index i can make V, thus $\| x \|$, decrease for $x \notin \Omega_i$, as long as the index i is selected. Also, $\| u \|$ is uniformly bounded as long as $\| x \|$ is bounded and $x \neq 0$. Therefore, by selecting a sequence of indices from $i = 1, 2, \ldots, \mu$, we can guarantee that $\| x \|$ decreases for $x \notin \Omega_1 \cap \Omega_2 \cap \cdots \cap \Omega_\mu = \{0\}$. This implies that $\| x \|$ decreases globally as long as $x \neq 0$, for any given initial state vector $x_0 \in R^n$ and model uncertainty.

Since there is switching in the variable index control law (6.2.22) due to the index selection, the control action smoothness and system stability during switching have to be investigated. They are both relevant to the definitions of $s_i(x) = x^T P_i x$ $(i = 1, 2, \ldots, \mu)$ and the switching frequency.

The control actions are smooth if there is infrequent switching, and the values of $s_1(x), \ldots, s_\mu(x)$ are of the same order of magnitude for a given x. The variable index control systems are stable during switching if there are no limit cycles and frequent switching, i.e., no chattering. Limit cycles may occur when control law switching does not result in a decrease of $\| x \|$, even if the switching is not frequent. The limit cycles can be eliminated if the decreasing rate of $\| x \|$ is not too slow. The frequent switching can be eliminated by choosing the next index i in eq. (6.2.22) such that the distance between the current system state vector x and the ith singularity set Ω_i is furthest in the measurement of Euclidian distance, and keeping this index as long as $\| Q(x, s_i) \| > 0$. In summary, the control action smoothness and system stability during switching can be guaranteed by making the values of $s_1(x), \ldots, s_\mu(x)$ in the same order of magnitude for a given x, and selecting a proper index sequence and $f_i(s_i)$ $(i = 1, 2, \ldots, \mu)$.

Therefore, the variable index control law (6.2.22) with a selected sequence of indices from $i = 1, 2, \ldots, \mu$ makes $x=0$ a global asymptotical stable motion of system (6.2.1), and the control system robust to model uncertainty.

The boundedness of control u is proved in the following. For $\| x \| > 0$, $\overline{A}(x)$ and $B(x)$ are bounded for bounded $\| x \|$. From eqs. (6.2.20) and (6.2.25), we know $\| Q(x, s_i) \|$ and $\| R(x, s_i) \|$ are also bounded for bounded $\| x \|$. Thus, $\| u \|$ is always bounded if $\| x \| > 0$. The boundedness of $\| u \|$ for $x=0$ can be proved using its limit.

Notice that $\overline{A}_i(0)=0$, and $B(0) \neq 0$. Eqs. (6.2.20) and (6.2.25) ensure that when x is in the small neighborhood of zero, the following inequalities hold:

$$\| \overline{A}_i(x) \| \leq a_0 \| x \| \qquad \| R(x, s_i) \| \leq a_1 \| x \|^2 \tag{6.2.30}$$

$$a_2 \| x \| \leq \| Q(x, s_i) \| \leq a_3 \| x \| \qquad [Q(x, s_i) Q(x, s_i)^T]^{-1} = \| Q(x, s_i) \|^{-2} \tag{6.2.31}$$

where a_j is a positive constant ($j=0, 1, 2, 3$). Therefore,

$$\begin{aligned}\lim_{x \to 0} \| u \| &= \lim_{x \to 0} \| Q(x, s_i)^T R(x, s_i) / [Q(x, s_i) Q(x, s_i)^T] \| \\ &\leq \lim_{x \to 0} \| Q(x, s_i)^T \| \; \| Q(x, s_i) \|^{-2} \; \| R(x, s_i) \| \\ &\leq \lim_{x \to 0} a_3 \| x \| a_2^{-1} \| x \|^{-2} a_1 \| x \|^2 = 0\end{aligned}$$

When $x \neq 0$, $\| u \|$ is always bounded. From the continuity of u defined by eq. (6.2.22) for a given index i and the above limit of u, we can set $u=0$ when $x=0$. For tracking problems, $\overline{A}_i(0, t) \neq 0$, and a_j ($j=0, 1$) are positive time functions. Similarly, we can prove that $\| u \|$ is bounded. Therefore, $\| u \|$ is uniformly bounded as long as $\| x \|$ is bounded.

Since the system stability is proved under the given model uncertainty (6.2.2), the control system is also robust. This proves the theorem.

Remark 6.2.1: Although different error indices are needed for different systems, there always exists a set of feasible error indices for nonlinear system (6.2.1) with $B(x) = B\sigma(x)$. Therefore, the variable index control approach can solve the control problem of system (6.2.1). This characteristic remedies the problem facing variable structure control, that is, how to choose a feasible sliding surface to guarantee the stability of sliding motions for general nonlinear systems.

Remark 6.2.2: The variable index control law (6.2.22) is highly robust to system model uncertainty. The robustness is due to $\overset{\max}{\Delta}\, \dot{s}(x)$, not $sgn(s(x))$. This is a major difference between the variable index control and variable structure control. Since we do not need to drive system states to any specified surfaces except $x = 0$, there is no sliding motion. Since we do not need to maintain the system states on any specified surfaces and infrequent switching can be guaranteed, there is no chattering. This result shows that generating sliding motion is not the only way to achieve robustness.

Remark 6.2.3: The variable index control law (6.2.22) is given in state regulation form. Let $e = x - x_d$, where x_d is the desired coordinated trajectory of the state vector x. Let $s_1(e), \ldots, s_\mu(e)$ be a set of feasible error indices based on e. Then control law (6.2.22) can be extended directly to the tracking control problems with a few notation changes. It is worth pointing out that the number of states which can be designed to track arbitrary trajectories is less than or equal to the number of control inputs. The rest states have to track coordinated trajectories derived based the independent trajectories.

Remark 6.2.4: Selecting different sequences of indices may lead to different control performances. This is an interesting topic which requires further elaboration. In the following, we introduce a small constant δ (say, 10^{-7}) into eq. (6.2.22), and propose a simple index selection algorithm.

$$u = Q(x,s_i)^T R(x,s_i)/[Q(x,s_i)Q(x,s_i)^T + \delta] \qquad if\ x \notin \Omega_i \qquad (6.2.32)$$

where i is determined by the following index selection algorithm:

For a given error constant ε,

a. if $\| Q(x,s_1) \| > \varepsilon$, *then* $i = 1$;
b. if $\| Q(x,s_k) \| \le \varepsilon$ for $k < j < \mu$ and $\| Q(x,s_j) \| > \varepsilon$, *then* $i = j$;
c. if $\| Q(x,s_k) \| \le \varepsilon$ for all $k = 1, \ldots, \mu$, then decrease ε and go to step *a*.

Example 6.2.1: Consider the following third-order nonlinear system:

$$\dot{x}_1 = x_2 + \theta_1\phi_1(x_1,x_2,x_3) + \phi_2(x_1,x_2,x_3)$$
$$\dot{x}_2 = x_3$$
$$\dot{x}_3 = \sigma(x_1,x_2,x_3)u + \theta_3\phi_3(x_1,x_2,x_3)$$

where $\phi_i(x_1,x_2,x_3)$ $(i=1,3)$ and $\sigma(x_1,x_2,x_3) \neq 0$ are known. θ_i $(i=1,3)$ and $\phi_2(x_1,x_2,x_3)$ are unknown. However, $\underline{\theta}_i \le \theta_i \le \overline{\theta}_i$ $(i=1,3)$, and $\sum_{i=1}^{3} \underline{a}_i x_i \le \phi_2(x_1,x_2,x_3) \le \sum_{i=1}^{3} \overline{a}_i x_i$, where the bounds and the parameters are known. $\phi_1(x_1,x_2,x_3)$ and $\phi_2(x_1,x_2,x_3)$ must be defined to ensure global controllability of the system. $\phi_i(0)=0$ $(i=1,2,3)$.

In summary, the system is of bounded parametric and structural uncertainties. It is difficult to apply the popular backstepping method to this system, since it is not in either the strict-feedback form or the pure-feedback form (Kanellakopoulos, and Kokotovic, 1989). The generality of ϕ_i $(i=1,2,3)$ and σ may make the system not feedback linearizable. Only the stabilization problem is considered.

Let $x = [x_1\ x_2\ x_3]^T$, $\underline{a} = [\underline{a}_1\ \underline{a}_2\ \underline{a}_3]^T$ and $\overline{a} = [\overline{a}_1\ \overline{a}_2\ \overline{a}_3]^T$. Comparing eq. (6.2.1), we know that $B(x) = [0\ 0\ \sigma]^T$. The bounds of $A(x)$ in eq. (6.2.2) are:

$$a_1(x) = x_2 + \theta_1\phi_1 + \phi_2 \quad a_{L1}(x) = x_2 + [(\underline{\theta}_1 + \overline{\theta}_1)\phi_1 - (\overline{\theta}_1 - \underline{\theta}_1)|\phi_1|]/2 + \underline{a}^T x$$

$$a_{U1}(x) = x_2 + [(\underline{\theta}_1 + \overline{\theta}_1)\phi_1 + (\overline{\theta}_1 - \underline{\theta}_1)|\phi_1|]/2 + \overline{a}^T x$$

$$a_2(x) = a_{L2}(x) = a_{U2}(x) = x_3$$

$$a_3(x) = \theta_3\phi_3 \quad a_{L3}(x) = [\underline{\theta}_3 + \overline{\theta}_3]\phi_3 - (\overline{\theta}_3 - \underline{\theta}_3)|\phi_3|]/2$$

$$a_{U3}(x) = [(\underline{\theta}_3 + \overline{\theta}_3)\phi_3 + (\overline{\theta}_3 - \underline{\theta}_3)|\phi_3|]/2$$

Unlike other robust control approaches, the bounds are easy to estimate, especially when one knows the physical interpretation of the terms.

The multiple error indices are chosen as $s_i(x) = x^T P_i x$ (i=1, 2, 3), where:

$$P_1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} \qquad P_3 = \begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

to make them feasible. Let $f_i(s_i) = ks_i$, $k > 0$. Therefore,

$$Q(x, s_i) = 2x^T P_i B(x) = 2x^T P_{i3}\sigma(x) \qquad i = 1,\ 2,\ 3$$

$$P_{13} = [0\ 0\ 2]^T \quad P_{23} = [0\ 1\ 2]^T \quad P_{33} = [1\ 1\ 2]^T$$

Therefore, $\| Q(x, s_i) \| = 0$ for all $i = 1, 2, 3$ if and only if $\| x \| = 0$. Similarly,

$$R(x, s_i) = -2x^T P_i \overline{A}_i(x) - ks_i \qquad i = 1,\ 2,\ 3$$

where $\overline{A}_i(x)$ is determined as follows:

$$\overline{A}_i(x) = [\overline{a}^i_1(x)\ \overline{a}^i_2(x)\ \overline{a}^i_3(x)]^T \qquad i = 1,\ 2,\ 3$$

$$\overline{a}^i_j(x) = \begin{cases} a_{Uj}(x) & if\ \ x^T P_{ij} \geq 0 \\ a_{Lj}(x) & if\ \ x^T P_{ij} < 0 \end{cases} \qquad j = 1,\ 2,\ 3$$

$$u = -[2x^T P_i \overline{A}_i(x) + ks_i]/[2x^T P_{i3}\sigma(x)] \tag{6.2.33}$$

Thus the variable index control law (6.2.33) with the index selection algorithm can make the system globally stable at $x = 0$ in spite of the existence of model uncertainty.

Example 6.2.2: It is difficult to define a feasible conventional sliding surface for general linear systems using variable structure control approach. However, there always exist feasible multiple error indices for uncertain linear systems. Therefore, the robust control problem of general uncertain linear systems can be solved without chattering using the proposed variable index control approach. To make the statement simple and explain the result graphically, we design a variable index controller for a second order system. Extension to higher order systems is trivial.

The equations of a general second order linear system can be expressed as follows:

$$\begin{cases} \dot{x}_1 = a_{11}x_1 + a_{12}x_2 + b_1 u \\ \dot{x}_2 = a_{21}x_1 + a_{22}x_2 + b_2 u \end{cases} \tag{6.2.34}$$

where $b_1 \neq 0$ and $b_2 \neq 0$ are known, and a_{ij} is nonlinear, time–varying and uncertain. Their bounds are defined as follows:

$$a_{ij}^{\min} \leq a_{ij}(x,t) \leq a_{ij}^{\max}\ , \quad \text{for } all\ x \in R^n \text{ and } t \in [t_0, \infty] \tag{6.2.35}$$

where $a_{ij}^{\min}$ and a_{ij}^{max} are known constants or functions of x. The system is assumed to be controllable.

Since $n=2$ and $m=1$, two error indices are required to ensure global system stability. Let $b_1=1$ and $b_2=2$, and $s_i(x) = x^T P_i x$ $(i=1,2)$, where

$$P_1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

Let $f_i(s_i) = 2s_i$. Therefore,

$$Q(x,s_1) = 4(x_1 + x_2) \qquad Q(x,s_2) = 2(-x_1 + 3x_2) \tag{6.2.36}$$

Therefore, $\| Q(x,s_i) \| = 0$ for both $i=1,2$ if and only if $\| x \| = 0$. Similarly,

$$R(x,s_1) = -2(2x_1\bar{a}_1^1(x) + x_2\bar{a}_2^1(x)) - 2s_1 \tag{6.2.37}$$

$$R(x,s_2) = -2[(x_1 - x_2)\bar{a}_1^2(x) + (x_1 - 2x_2)\bar{a}_2^2(x)] - 2s_2 \tag{6.2.38}$$

where $\bar{a}_j^i$ $(i,j=1,2)$ can be determined according to the bounds of a_{ij}. The variable index control law is:

$$u = \begin{cases} R(x,s_1)/Q(x,s_1) & \text{if } |Q(x,s_1)| = 4|x_1 + x_2| > 0 \\ R(x,s_2)/Q(x,s_2) & \text{if } |Q(x,s_2)| = 2|-x_1 + 3x_2| > 0 \end{cases} \tag{6.2.39}$$

The singularity sets are defined by $\Omega_i = \{ x \mid \| Q(x,s_i) \| = 0,\ x \in R^2 \}$, and $\Omega_1 \cap \Omega_2 = \{0\}$. This can be seen clearly from fig. 6.1. The first option in eq. (6.2.39) is not defined on $x_1 + x_2 = 0$ and the second option of eq. (6.2.39) is not defined on $-x_1 + 3x_2 = 0$. However, there is always one well-defined option for $x \neq 0$, and $u = 0$ for $x = 0$. By switching between these two options, the variable index control law (6.2.39) can make the system states converge to the origin.

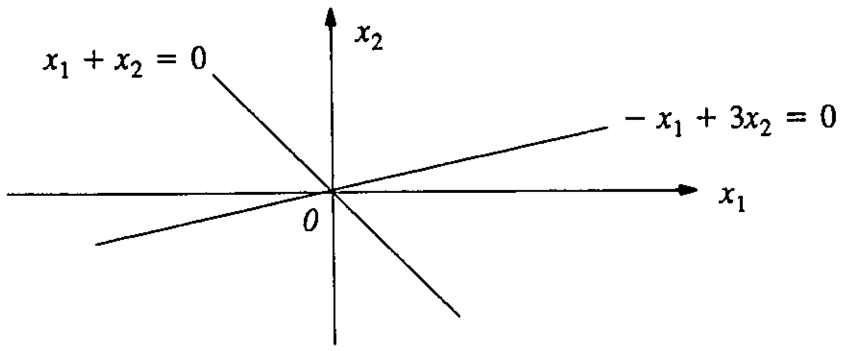


Fig. 6.1 Illustration of the singularity sets of example 3

For sampled-data control systems, the probability of the sampled system states falling on the singularity sets is almost zero. Therefore, there are actually few switches when $\| x \|$ is large. When $\| x \|$ is very small, frequent switches may occur, but they have little influence on $\| u \|$. These switches can be eliminated by introducing a small dead-zone, or using the hybrid variable index and linear control scheme (Liang and ElMaraghy, 1994a).

6.2.5 Simulation example

In order to test the variable index control laws by simulation, a two-link planar flexible joint manipulator is considered. Its dynamical equations are:

$$(A_1 + A_2 + 2A_3c_2)\ddot{q}_1 + (A_2 + A_3c_2)\ddot{q}_2 - A_3\dot{q}_2(2\dot{q}_1 + \dot{q}_2)s_2 + K_1(q_1 - p_1) = 0 \quad (6.2.40)$$
$$(A_2 + A_3c_2)\ddot{q}_1 + A_2\ddot{q}_2 + A_3\dot{q}_1^2\, s_2 + K_2(q_2 - p_2) = 0 \quad (6.2.41)$$
$$I_{m1}\ddot{p}_1 + B_{m1}\dot{p}_1 + F_{m1}\, sat(\dot{p}_1) - K_1(q_1 - p_1) = u_1 \quad (6.2.42)$$
$$I_{m2}\ddot{p}_2 + B_{m2}\dot{p}_2 + F_{m2}\, sat(\dot{p}_2) - K_2(q_2 - p_2) = u_2 \quad (6.2.43)$$

where $s_i = \sin q_i$, $c_i = \cos q_i$ and $sat(.)$ is the saturation function; q_i and p_i are the i–th link angle and motor angle.

$$A_1 = 5.57\ kgm^2 \qquad A_2 = 0.85\ kgm^2 \qquad A_3 = 1.05\ kgm^2$$

$$I_{m1} = 1.94\ kgm^2\ \ B_{m1} = 1.10\ Nms/rad\ \ F_{m1} = 0.2\ Nm\ \ K_1 = 790\ Nm/rad$$

$$I_{m2} = 0.25\ kgm^2\ \ B_{m2} = 0.22\ Nms/rad\ \ F_{m2} = 0.2\ Nm\ \ K_2 = 610\ Nm/rad$$

The multiple error indices are chosen as $s_i(x) = x^TP_ix$, where P_i $(i = 1, \ldots, 4)$ are chosen as follows:

$$P_i = \begin{bmatrix} P_{11} & 0 & 0 & P^i_{14} \\ 0 & P_{22} & 0 & P^i_{24} \\ 0 & 0 & P_{33} & P^i_{34} \\ P^{iT}_{14} & P^{iT}_{24} & P^{iT}_{34} & P_{44} \end{bmatrix} \qquad \begin{array}{l} P_{11} = P_{33} = 4I \\ P_{22} = P_{44} = 2I \end{array} \qquad I \in R^2$$

and

$$i{=}1, \qquad P^1_{14} = P^1_{24} = P^1_{34} = I$$

$$i{=}2, \qquad P^2_{14} = 0\ ,\ P^2_{24} = P^2_{34} = I$$

$$i{=}3, \qquad P^3_{14} = P^3_{24} = 0\ ,\ P^3_{34} = I$$

$$i{=}4, \qquad P^4_{14} = P^4_{24} = P^4_{34} = 0$$

It can be seen that P_i is always positive definite.

Simulation 1: This simulation was designed to test the robustness of the variable index control under parametric uncertainty: joint stiffness uncertainty. Assume that:

$$K_i^{\min} \le K_i \le K_i^{\max} \quad \text{for } i = 1,\ 2$$

where

$$K_i^{\min} = 500Nm/rad \quad K_i^{\max} = 800Nm/rad$$

All other parameters assume their nominal values given above. Assume that set point control is required with $q_{d1} = 1.5rad$ and $q_{d2}=-0.5rad$. $f_i(s_i) = 0.8s_i$.

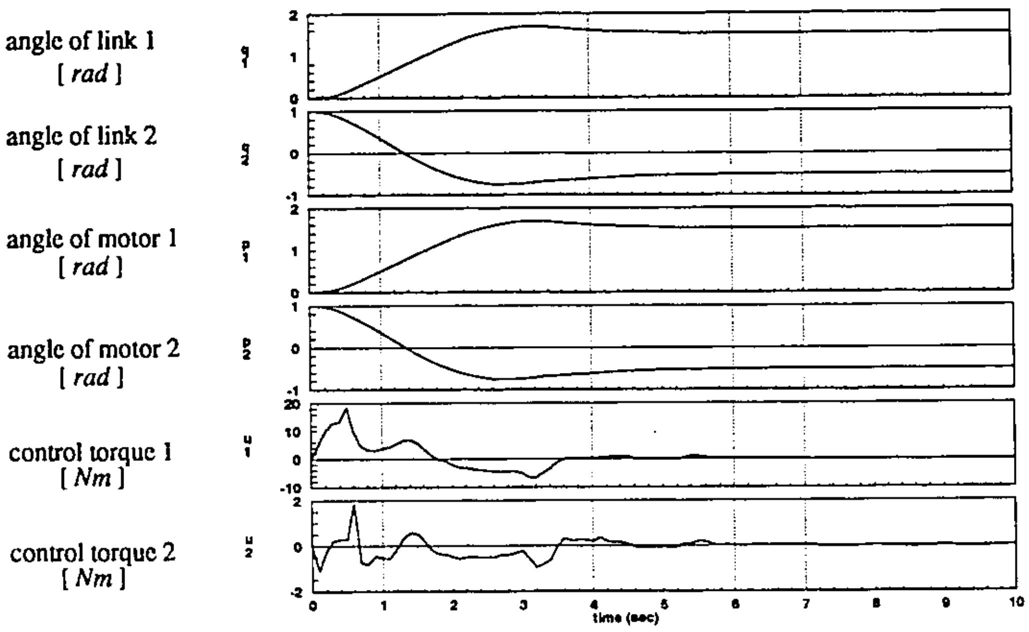


Fig. 6.2 Time responses of the two–link robot under joint stiffness uncertainty

Fig. 6.2 shows the simulation results with sampling period $T = 0.01$ s. The control performance is satisfactory. This verifies the robustness of the variable index control law to parametric uncertainty.

The effects of unknown load on the robot control system can also be examined. For example, let $0 \leq m_p \leq 10kg$. Results similar to those shown in fig. 6.2 can be obtained.

Simulation 2: This simulation was designed to test the robustness of the variable index control under structural uncertainty: link dynamic uncertainty. Assume that the structures and parameters of eqs. (6.2.40) and (6.2.41) are unknown, but eqs. (6.2.42) and (6.2.43) are completely known. The maximum angular accelerations of link 1 and 2 are all estimated as $\ddot{q}_{imax}$ =2 rad/sec^2. The results are shown in fig. 6.3. They are similar to those shown in

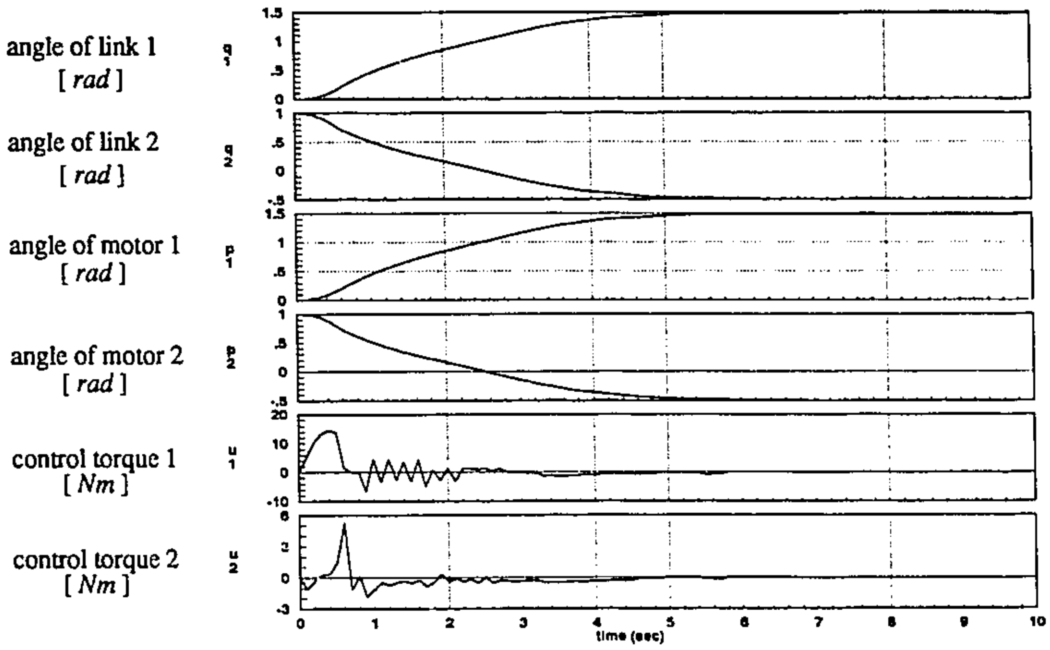


Fig. 6.3 Time responses of the two–link robot under link dynamic uncertainty

fig.6.2. This verifies the robustness of variable index control law to structural uncertainty.

Simulation 3: This simulation was designed to perform trajectory tracking control. Fig. 6.4 shows the time history of the tracking response, where $q_{d1} = sin(\pi t/2.5)$, $q_{d2} = 1 - q_{d1}$, and sampling period $T = 0.01\ s$. The control results are satisfactory. This verifies that the variable index control law is applicable to tracking control problems, if the desired trajectories are given.

These simulations verify the correctness of the theory. Detailed design of the variable index control laws for flexible joint robots can be found in Liang and ElMaraghy (1994a).

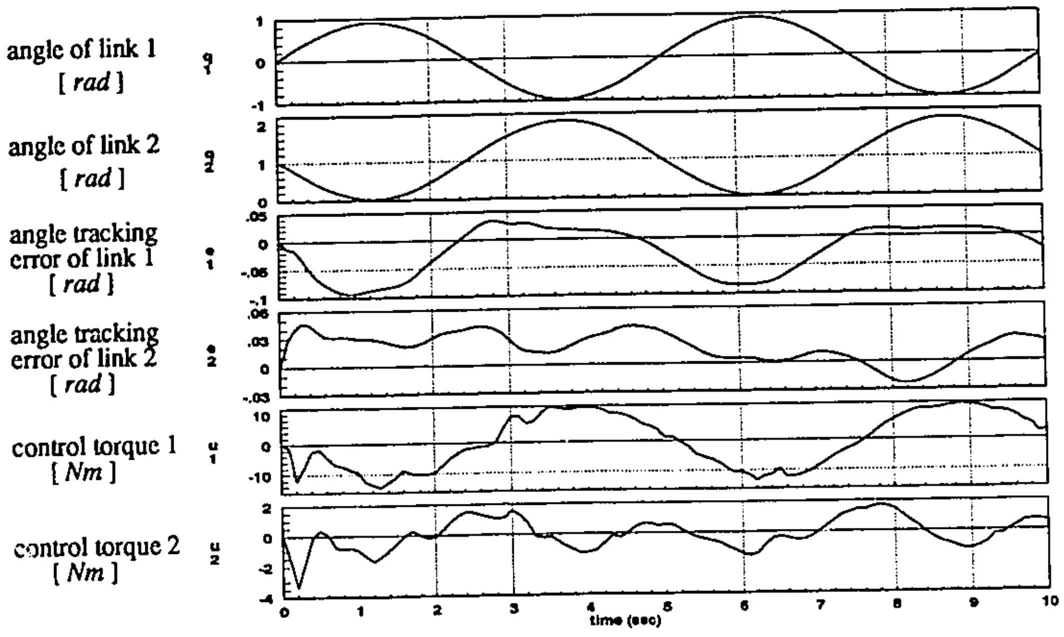


Fig. 6.4 Trajectory tracking responses of the two–link robot

6.3 EXTENSION TO GENERAL NONLINEAR SYSTEMS

Consider the general nonlinear systems described by:

$$\dot{x} = f(x, u) \tag{6.3.1}$$

where $f(0,0) = 0$, x and u have the same definitions as before. $f(x,u)$ is assumed to guarantee the existence of solutions of the differential equations (6.3.1). Nonlinear system (6.3.1) is assumed to be stabilizable/controllable. The system model (6.3.1) is of bounded uncertainty in $f(x,u)$:

$$f_{Li}(x, u) \leq f_i(x, u) \leq f_{Ui}(x, u) \qquad \text{for } i = 1, \ldots, n \tag{6.3.2}$$

where $f_i(x, u)$ is the i–th entry of $f(x,u)$, and $f_{Li}(x, u)$ and $f_{Ui}(x, u)$ are known and bounded functions as long as $\| x \|$ and $\| u \|$ are bounded. Obviously, system (6.3.1) includes eq. (6.2.1) as a special case, even with general $B(x)$.

There are two approaches to show the existence of feasible multiple error indices for general nonlinear system (6.3.1). The first approach is by introducing dynamic compensa-

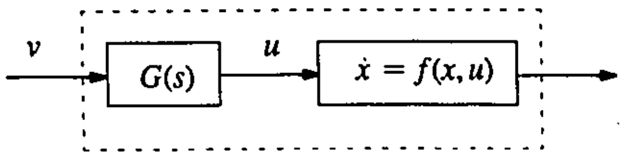


Fig. 6.5 General nonlinear systems with stable dynamic compensators

tors (see fig. 6.5) to system (6.3.1). That is, let:

$$u = G(s)v \quad or \quad \dot{u} = Eu + Fv \tag{6.3.3}$$

where $v \in R^m$ is the new control input vector to be designed, and $G(s) = (sI - E)^{-1}F$ is a designer–chosen stable and minimum–phase transfer function matrix. Then system (6.3.1) can be transformed into the following:

$$\dot{z} = \bar{f}(z) + \bar{B}v \tag{6.3.4}$$

where $z = [x^T \ \ u^T]^T \in R^{n+m}$, and

$$\bar{f}(z) = \begin{bmatrix} f(x,u) \\ Eu \end{bmatrix} \quad \bar{B} = \begin{bmatrix} 0 \\ F \end{bmatrix} \tag{6.3.5}$$

Since eq. (6.3.4) fits the assumptions for eq. (6.2.1), feasible multiple error indices exist for system (6.3.4), i.e., the compensated nonlinear system (6.3.1). Thus, the variable index control law (6.2.22) with an index selection algorithm can make $x=0$ a global asymptotically stable motion of the nonlinear system (6.3.1), in spite of the existence of the uncertainty (6.3.2).

Introducing filter–like dynamic compensators to system dynamics is desirable for many applications, since smoother control actions can be achieved. A useful choice of the dynamic compensators is that $G(s) = aI/(s+a)$, i.e., $E = -aI$, $F = aI$, where $a > 0$.

The second approach is by using nonlinear diffeomorphism to transform eq. (6.3.1) into the following:

$$\dot{z} = \bar{f}(z) + B\sigma(z)u \tag{6.3.6}$$

where $z = T(x)$ is the new system coordinates. $T(x)$ can be determined from a nominal model of system (6.3.1). The uncertainty in eq. (6.3.1) can be regarded as the uncertainty of $\bar{f}(z)$ if $\| u \|$ is uniformly bounded by a constant (this is true in practical implementation).

The second approach is useful when increasing system order is not desirable. Although not all of the systems represented by eq. (6.3.1) can be transformed into eq. (6.3.6), the nonlinear systems that can be transformed into eq. (6.3.6) belong to a larger class of nonlinear systems than the class of feedback linearizable nonlinear systems do . We do not require a nonlinear system be transformed into a linear system. Instead, we transform a nonlinear system into another nonlinear system that is solvable. Therefore, this class of nonlinear systems includes those that may not be feedback linearizable.

Remark 6.2.5: Assume that the uncertainty in $B(x)$ of eq. (6.2.1) can be expressed as:

$$b_{Lij}(x) \leq b_{ij}(x) \leq b_{Uij}(x) \qquad \text{for } i = 1, \ldots, n, \; j = 1, \ldots, m \tag{6.3.7}$$

where $b_{ij}(x)$ is the ij–th entry of $B(x)$, and $b_{Lij}(x)$ and $b_{Uij}(x)$ are known and bounded functions as long as $\| x \|$ is bounded. Then the above shows that the variable index control law (6.2.22) also works by introducing dynamic compensators. If $\| u \|$ is uniformly bounded by a constant (this is true in practical implementation), variable index control law (6.2.22) can ensure the robustness of the control system under the uncertainty in $A(x)$ and $B(x)$.

In conclusion, the variable index control approach can be applied to general nonlinear system (6.3.1), since a set of feasible error indices for compensated nonlinear system (6.3.1) can always be found.

6.4 ADAPTIVE NEUROCONTROL OF GENERAL SYSTEMS

In this section, a new global stable adaptive neurocontrol scheme for general nonlinear system (6.3.1) is derived based on the variable index control approach. It greatly extends the current stability–based neurocontrol approach to more general control problems. Compared with the conventional approaches, the new adaptive neurocontrol scheme does not require the controlled nonlinear systems to be feedback linearizable, and the uncertain parameters to be linearly parameterizable.

6.4.1 Neural network representation

As indicated in section 3.4, the nonlinear system (6.3.1) can be represented by a neural network. In the following, single–hidden layer neural networks are used to represent the nonlinear system (6.3.1). These neural networks can be RBF networks, sigmoidal networks (Liang and ElMaraghy, 1994b), or polynomial networks. The structural design of these neural networks should ensure that a given nonlinear system is well approximated. In other words, the span of function bases $\{g_1(x,u),\ g_2(x,u),\ldots,g_L(x,u)\}$ defined by the hidden neurons should contain $f(x,u)$. If this is true, the following approximation can be obtained:

$$f(x,u) = W^T\Phi(x,u) + d(t) \tag{6.4.1}$$

where $d(t)$ stands for the representation error vector of the neural network; $W \in R^{L\times n}$ is the unknown hidden–output connection weight matrix of the neural network; and $\Phi(x,u)$ is the output vector of the hidden neurons and is known for given $z = [x^T \ \ u^T]^T \in R^{n+m}$. $\Phi(x,u)$ is defined as:

$$\Phi(x,u) = [g_1(x,u),\ g_2(x,u),\ \ldots,\ g_L(x,u)]^T \tag{6.4.2}$$

Assume that the nonlinear system (6.3.1) is time–invariant or slowly time–varying. Then $W \in R^{L\times n}$ can be regarded as an unknown constant weight matrix to be learned. Let the estimated W be $\hat{W}$, and the estimation error vector be $\tilde{W} = \hat{W} - W$. By proper neural network structural design, the entries of $d(t)$ can be bounded by any specified small constant

c.

$$|d_i(t)| < c \qquad i = 1, \ldots, n \tag{6.4.3}$$

6.4.2 The new adaptive neurocontrol algorithm

The adaptive regulation problem is considered. The state measurement $x(t)$ is assumed to be available. If the desired coordinated state trajectory of nonlinear system (6.3.1) is known, the result is easily extended to the tracking control problem. For convenience, equations (6.3.4) and (6.3.5) are rewritten in the following form:

$$\dot{z} = \bar{f}(z) + \bar{B}v \tag{6.4.4}$$

$$\bar{f}(z) = \begin{bmatrix} f(x,u) \\ Eu \end{bmatrix} \quad \bar{B} = \begin{bmatrix} 0 \\ F \end{bmatrix} \tag{6.4.5}$$

where E and F are known matrices.

Let $s_i(z) = z^T P_i z/2$ $(i = 1, \ldots, \mu)$ be a set of feasible error indices, where P_i is a symmetric positive definite matrix, and

$$P_i = [P_{i1} \quad P_{i2}] \in R^{(n+m)\times(n+m)} \quad P_{i1} \in R^{(n+m)\times n} \quad P_{i2} \in R^{(n+m)\times m} \tag{6.4.6}$$

For $i = 1, 2, \ldots, \mu$, define:

$$Q(z, s_i) = z^T P_{i2} F \tag{6.4.7}$$

$$\hat{R}(z, s_i) = -z^T P_{i1}[\bar{d}(t) + \hat{W}^T \Phi(x,u)] - z^T P_{i2} Eu - k s_i(z) \tag{6.4.8}$$

where

$$\bar{d}(t) = [\bar{d}_1(t) \ldots \bar{d}_n(t)]^T \qquad \bar{d}_j(t) = c \cdot sgn\{z^T P_{i1}\} \tag{6.4.9}$$

The weight matrices $\hat{W}_1$ and $\hat{W}_2$ are computed from the following learning algorithms:

$$\dot{\hat{W}} = \Gamma \Phi(x,u) z^T P_{i1} \tag{6.4.10}$$

where $\Gamma \in R^{L\times L}$ is a chosen positive definite matrix that determines the learning rate.

Applying the variable index control approach given in section 6.2, we can obtain:

$$v = Q^T(z, s_i)\hat{R}(z, s_i)/[Q(z, s_i)\ Q^T(z, s_i)] \quad \text{if} \quad \|Q(z, s_i)\| > 0 \tag{6.4.11}$$

Thus, the control input vector can be determined as follows:

$$u = G(s)v \qquad or \qquad \dot{u} = Eu + Fv \tag{6.4.12}$$

where $G(s) = (sI - E)^{-1}F$ is a stable and minimum–phase transfer function matrix.

Then the robust adaptive neurocontrol law (6.4.12) and (6.4.11) with eq. (6.4.10) as the adaptation/learning algorithm can make all the signals in the adaptive neurocontrol system uniformly bounded, and the system state vector x of eq. (6.3.1) will asymptotically converge to zero.

6.4.3 Global stability proof

To prove the global stability of the adaptive neurocontrol system, let us consider the following Lyapunov function candidate:

$$V(t) = s_i(z) + \frac{1}{2}tr(\tilde{W}^T\Gamma^{-1}\tilde{W}) \tag{6.4.13}$$

for a selected error index $s_i(z)$ with $\| Q(\varepsilon, s_i)\| > 0$. Therefore,

$$\begin{aligned}
\dot{V}(t) &= \dot{s}_i(z) + tr(\tilde{W}^T\Gamma^{-1}\dot{\tilde{W}}) = z^TP\dot{z} + tr(\tilde{W}^T\Gamma^{-1}\dot{\hat{W}}) \\
&= z^TP_{i1}[W^T\Phi(x,u) + d(t)] + z^TP_{i2}(Eu + Fv) + tr[\tilde{W}^T\Phi(x,u)z^TP_{i1}] \\
&\le z^TP_{i1}[W^T\Phi(x,u) + \bar{d}(t)] + z^TP_{i2}Eu + Q(z,s_i)v + z^TP_{i1}\tilde{W}^T\Phi(x,u) \\
&= z^TP_{i1}[W^T\Phi(x,u) + \bar{d}(t)] + z^TP_{i2}Eu + \hat{R}(z,s_i) + z^TP_{i1}\tilde{W}^T\Phi(x,u) \\
&= -ks_i(z) \le 0 \qquad \text{(using eq. (6.4.8))}
\end{aligned} \tag{6.4.14}$$

Because eq. (6.4.14) is valid for all $t \ge 0$, $z = [x^T \ \ u^T]^T$ and $\tilde{W}$ are bounded uniformly if their initial values are bounded. The uniform boundedness of $x(t)$ and $u(t)$ ensures that $\dot{x}(t)$ and $\Phi(x,u)$ are also uniformly bounded if $x(0)$ and $u(0)$ are bounded. Thus, $\dot{\hat{W}}(t)$

is uniformly bounded. The network structure design guarantees that the entries of the optimal weight matrix W is bounded, and $\tilde{W} = \hat{W} - W$. Therefore, $\hat{W}(t)$ is also uniformly bounded. As proved in Theorem 6.2.1, $v(t)$ is uniformly bounded, because $Q(z, s_i)$ and $\hat{R}(z, s_i)$ are uniformly bounded, and there always exists an index i such that $\| Q(z, s_i) \| > 0$. This proves that all the signals in the adaptive neurocontrol system are uniformly bounded if their initial values are bounded.

To prove the asymptotic convergence of x, we apply Barbalat's lemma to the following continuous nonnegative function:

$$V_1(t) = V(t) - \int_0^t [\dot{V}(\tau) + ks_i(z)]d\tau$$

with $\dot{V}_1(t) = -ks_i(z)$ and $\ddot{V}_1(t) = -kz^TP\dot{z}$. Therefore, $\dot{V}_1(t) \to 0$, thus $x(t) \to 0$ as $t \to \infty$. The same arguments given in the proof of theorem 6.2.1 regarding the switching of V among diffenent $s_i(z)$ apply to this proof, as well. Therefore, the system state vector x of eq. (6.3.1) asymptotically converges to zero. This concludes the global stability proof.

Remark 6.4. 1: The index selection algorithm introduced in section 6.2 can be used in the adaptive neurocontrol algorithm.

Remark 6.4. 2: The learning algorithm (6.4.10) can be changed into the following least–squares learning algorithm to achieve better learning performance:

$$\dot{\hat{W}} = \Gamma(t)\Phi(x, u)z^TP_{i1} \tag{6.4.15}$$

$$\dot{\Gamma}(t) = -\Gamma(t)\Phi(x, u)\Phi^T(x, u)\Gamma(t) \tag{6.4.16}$$

If eq. (6.3.1) is of some time–varying factors, then eq. (6.4.16) can be modified into:

$$\dot{\Gamma}(t) = -\lambda(t)\Gamma(t) - \Gamma(t)\Phi(x, u)\Phi^T(x, u)\Gamma(t) \tag{6.4.17}$$

where $\lambda(t) > 0$ is the forgetting factor. If $\Phi(x,u)$ satisfies the persistent excitation condition, then $\hat{W}(t) \rightarrow W$ as $t \rightarrow \infty$, i.e., the neurocontroller converges to the best performance it can achieve under the given structure. Otherwise, the neural network weights converge to constants that are feasible for the given control inputs.

Remark 6.4. 3: It is worth indicating that in conventional robust adaptive control, the unmodeled system dynamics determine the lower bounds of the control precision. These bounds are irreducible, unless better system models are used, or internal models are included in the control design. In the new robust adaptive neurocontrol, however, if the unmodeled system dynamics are observable through system outputs, the modeling error bounds can always be reduced by proper neural network structure design, owing to the universal approximation ability of neural networks.

Remark 6.4. 4: In practice, there is always some a priori information about the nonlinear system (6.3.1), e.g., the partial structure of $f(x, u)$, or a nominal model of eq. (6.3.1). It is easy to incorporate this into the adaptive neurocontrol system design, to reduce on–line computation. For instance, if $f(x, u)$ can be expressed as:

$$f(x,u) = f_1(x,u) + f_2(x,u) \tag{6.4.18}$$

where $f_1(x, u)$ is completely known, $f_2(x, u)$ is unknown. Then $f_1(x, u)$ can be used directly in eqs. (6.4.8) and (6.4.11). Only $f_2(x, u)$ is required to be approximated using neural networks, so the neural network design is simplified.

Liang and ElMaraghy (1994d) have applied the above theory to the trajectory control of flexible joint robots with success.

6.5 SUMMARY

In this chapter, the variable index control approach is proposed to solve the robust control problem of general nonlinear systems with bounded parametric or structural uncertainties. It is applied to derive stability–based adaptive neurocontrol schemes for unknown nonlinear systems. The variable index control approach guarantees the global stability of the control systems. It possesses inherent robustness to system model uncertainty, which is not required to satisfy any matching condition. The robustness of variable index control laws is due to the worst–case approaching control strategy. It does not impose any growth conditions and infinite differentiability assumptions on the system nonlinearity. It can be applied to nonlinear systems that are not feedback–linearizable, and uncertain linear systems with time–varying and state–dependent parameters. As there is no sliding motion and the system is stable during switching, the control actions are chatter–free. The approaches presented in this chapter represent a breakthrough in both robust control of nonlinear systems and the stability–based adaptive neurocontrol.

The theory has been applied to trajectory control of flexible joint robots, and the results are also useful in the control of rigid robots when actuator dynamics are considered. Simulation shows that the theory is feasible and the resultant control systems are robust to bounded model errors.

It is assumed throughout this chapter that the system states are measurable. If this is not true, system state observers have to be constructed. If the variable index controllers are implemented using digital computers, the measurement noise and sampling errors must also be considered. It is better to implement a large scale neurocontroller using neural network hardware; otherwise, intensive computation is unavoidable.

CHAPTER 7

ROBUST ADAPTIVE NEUROCONTROL OF FLEXIBLE–JOINT ROBOTS

7.1 INTRODUCTION

By neglecting link flexibility, joint stiffness nonlinearity, external disturbance and coupling between link dynamics and motor dynamics, an n–link flexible–joint robot manipulator with a free end–effector can be approximately modeled (Spong, 1989) as follows:

$$M(q)\,\ddot{q} + H(q,\dot{q}) + K_s\,(q-p) = 0 \tag{7.1.1}$$

$$I_m\,\ddot{p} + B_m\,\dot{p} + F_m - K_s\,(q-p) = u \tag{7.1.2}$$

where $q = [q_1\ \dots\ q_n]^T$ is the vector of the joint angles; $M(q)$ is the positive definite inertia matrix; $H(q,\dot{q})$ is the vector of centrifugal, Coriolis and gravitational terms; $p = [p_1,\dots,p_n]^T$ is the vector of rotational angles of actuator rotors; $I_m = diag[I_1,\dots,I_n]$, where I_i is the moment of inertia of the ith rotor; $B_m = diag[B_{m1},\dots,B_{mn}]$, where B_{mi} is the viscous friction coefficient of the ith joint; $F_m = [F_{m1},\dots,F_{mn}]^T$, where F_{mi} is the friction torque acting on the ith rotor; $K_s = diag[K_{s1}\ ,\ \dots\ ,\ K_{sn}]$, where K_{si} is the elastic constant of the ith joint; and u is the vector of driven torques applied to the actuator rotors.

As indicated in section 1.3.1, joint flexibility of robots is a practical problem in many industrial and aerospace robots. Due to the existence of modeling errors, most conventional control schemes for flexible–joint robots have limited control precisions. The control problem of flexible–joint robots (FJR) with uncertain dynamic models attracted great interest from both academia and industry. Different control approaches have been proposed for flexible–joint robots in the past decade. They can be classified into: 1) the exact model–based control approach, 2) robust control approach, 3) adaptive control approach, 4) iterative learning control approach, 5) fuzzy control approach and 6) neurocontrol approach.

The exact model–based control approach, which dates back to the early 1980s, includes the paradigms of the singular perturbation schemes (e.g., Ficola, Marino and Nicosia,1983), feedback linearization schemes and inverse dynamics scheme (e.g., DeLuca, Isidori and Nicolo, 1985; Bortoff and Spong, 1987; Jankowski and ElMaraghy, 1991), invariant manifold scheme (e.g., Khorasani and Spong, 1985), etc. Since these schemes require exact knowledge of the robot parameters, studying the robustness of these control systems is necessary. The robust control approach allows the robotic system models to possess some bounded structural and parametric uncertainty (e.g., Canudas and Lys, 1988; Lin, 1991; Liang and ElMaraghy, 1993f; Qu, 1993). This approach can usually guarantee global stability of robotic systems and does not need measurement of accelerations and jerks, although it often results in non–optimal or conservative solutions, especially when the uncertainty is large. For ordinary variable structure controllers, it is difficult to choose a proper sliding surface to guarantee the existence of asymptotically stable sliding motion, if acceleration and jerk signals are not available.

The adaptive control approach allows the existence of small structural and large parametric uncertainty in system models. The schemes given by Ghorbel, Spong and Hung (1989), Spong (1989), and Khorasani (1991) are based on the use of the singular perturbation technique, and linear parameterization techniques. They can be applied to robots with high joint stiffness. Lozano and Brogliato (1992), Mrad and Ahmad (1992) and Han (1992) reported new adaptive trajectory control schemes for arbitrary joint stiffness and without the use of link accelerations and jerks. Most of these schemes deal with parametric uncertainty only. Some of them further assume that the actuator parameters and joint stiffnesses are known. Although there may be large transient responses before parameter adaptation converges, this approach is promising in solving some practical problems. There are also iterative learning control schemes (e.g. Miyazaki et al, 1986) that deal with the flexible–joint robot control problem.

Recently, the fuzzy control approach and neurocontrol approach have been attracting more interest due to their potential in dealing with large structural and large parametric uncertainty. Compared with the exact model–based control approach, robust control approach and adaptive control approach, the neurocontrol approach needs the least a priori information about the controlled flexible–joint robotic systems, and is therefore more robust to modeling errors (e.g., friction and link flexibility). It is also easy to incorporate a priori information about the robotic systems into neurocontroller design, if it is available. In general, the iterative learning control approach, fuzzy control approach and neurocontrol approach tolerate modeling errors better than the exact model–based control approach, robust control approach and adaptive control approach. Compared with the iterative learning control approach, the neurocontrol approach does not need the desired trajectories to be repetitive. Unlike the fuzzy control approach, the neurocontrol approach does not need a set of fuzzy control rules, obtained by either summarizing previous operating experience or performing fuzzy model identification, and can be applied to more general nonlinear systems.

However, the neurocontrol approach is relatively new, and there are fewer theoretical results and less practical implementation experience available. The main existing problems in the neurocontrol approach are: (1) how to ensure the global convergence of the learning processes, and (2) how to determine the neural network structure for a given problem. At present, most existing backprop–based neurocontrol schemes are only suitable for off–line training of neurocontrollers (Psaltis, et al., 1987a, Narendra and Parthasarathy, 1990, Liang and ElMaraghy, 1993a), since there are several methods and heuristics to solve the above mentioned problems by off–line training. The existing stability–based neurocontrol schemes (e.g., Sanner and Slotine,1992) can be used for on–line training of neurocontrollers, but are difficult to apply to flexible–joint robots. Few published papers deal with the neuro-control problem of flexible–joint robots, due to their higher order dynamics. The scheme presented by Zeman, et al. (1989) was among the first attempts to deal with this problem, and it belongs to the backprop–based neurocontroller catalog. To the best of our knowledge, there is no stability–based adaptive neurocontrol scheme for flexible–joint robots in the published literature.

Theoretically, all the neurocontrol schemes proposed in the previous chapters can be applied to the control of flexible–joint robots. However, using a priori information about the dynamics of flexible–joint robots, simpler neurocontrol schemes can be derived.

This chapter aims at developing stable robust adaptive neurocontrollers for general flexible–joint robots. The feedback signals are the joint and motor angular positions and velocities. Measurements of accelerations or jerks are not required. It is proved that all the signals in the closed–loop adaptive neurocontrol systems can be made bounded and the output tracking errors can be guaranteed to be globally convergent to zero. Since the new adaptive neurocontrol systems are robust to neural network representation errors, the structure design of the neural networks can be simplified. Unlike the conventional adaptive controllers, the proposed adaptive neurocontrol schemes allow the flexible–joint robots to possess a general structure and unknown disturbances, and need the least a priori information about the dynamics. Any observable dynamics, some of which were regarded as unmodeled in conventional approaches, can be modeled by neural networks. The modeling errors are reducible by proper network structure design, which guarantees that the output tracking errors satisfy the control requirement.

The rest of this chapter is organized as follows. Section 7.2 presents a direct adaptive neurocontrol scheme for flexible–joint robots with proved stability. Section 7.3 presents a new model reference adaptive neurocontrol scheme for general flexible–joint robots. Section 7.4 gives the summary and discussions of the contributions.

7.2 THE DIRECT ADAPTIVE NEUROCONTROL SCHEME

7.2.1 The neurocontrol algorithm

The practical dynamic model given by eqs. (7.1.1) and (7.1.2) for an n–link flexible–joint robot manipulator usually contains structural and parametric uncertainty. The parametric uncertainty may come from load changes, irregular geometric shapes of the robotic components, non–uniform materials, an unsymmetrical motor or transmission installation, or part worn-out.. The structural uncertainty may come from the internal moving parts, friction, backlash, and unsymmetrical or unparalleled motor axes and joints. Sudden control action may excite the unmodeled high frequency characteristics, such as link flexibility. The unsymmetrical motor axes result in coupling between link dynamics and motor dynamics. Based on these facts, the following general model for an n–link flexible–joint robot manipulator is proposed:

$$M(q)\ \ddot{q} + f_1(q,\dot{q}) = K(q-p)\ p \tag{7.2.1}$$

$$I_m\ \ddot{p} + f_2(q,p,\dot{p}) = u \tag{7.2.2}$$

where the variable definitions are the same as in eqs. (7.1.1) and (7.1.2); $M(q)$ and I_m are the symmetric positive definite inertia matrices with unknown structures and parameters;

$f_1(q,\dot{q})$ and $f_2(q,p,\dot{p})$ are unknown function vectors; and $K(q{-}p)$ is the unknown symmetric positive definite joint stiffness matrix and is nonlinear in $q{-}p$, and $\underline{K}_i \leq K_i \leq \overline{K}_i$ with the boundary matrices known.

Obviously, the control problem for the unknown robotic system (7.2.1) and (7.2.2) is more difficult than that for the known robotic system (7.1.1) and (7.1.2). Assume that the desired trajectory of the end–effector of a robot manipulator is expressed in joint coordinates as $q_d \in C^4$, and the corresponding desired trajectory of the rotational angles of actuator rotors is expressed as p_d, $\dot{p}_d$, $\ddot{p}_d$, which are to be determined. Only the measurements of $q(t)$, $\dot{q}(t)$, $p(t)$, and $\dot{p}(t)$ are available.

Define the tracking error vectors as:

$$e = q - q_d \qquad \delta = p - p_d \tag{7.2.3}$$

Let $s_1 = \dot{e} + \Lambda_1 e$ and $s_2 = \dot{\delta} + \Lambda_2 \delta$. Therefore,

$$s_1 = \dot{q} - \dot{q}_r \qquad \text{with} \qquad \dot{q}_r = \dot{q}_d - \Lambda_1 e \tag{7.2.4}$$

$$s_2 = \dot{p} - \dot{p}_r \qquad \text{with} \qquad \dot{p}_r = \dot{p}_d - \Lambda_2 \delta \tag{7.2.5}$$

where q_r and p_r are called the reference trajectories and are computable from measured signals; Λ_1 and Λ_2 are diagonal positive definite matrices. With the new notions, the generalized error equations of the flexible–joint robotic systems can be expressed as follows:

$$M(q)\dot{s}_1 + [M(q)\ddot{q}_r + f_1(q,\dot{q})] = Kp \tag{7.2.6}$$

$$I_m\dot{s}_2 + [I_m\ddot{p}_r + f_2(q,p,\dot{p})] = u \tag{7.2.7}$$

where $M(q)$, I_m and K and the terms in the square brackets are unknown.

Define new supplementary vectors of s_i with dead–zone Δ_i as follows:

$$s_{\Delta i} = [s_{\Delta i1} \ \cdots \ s_{\Delta in}]^T \qquad i{=}1,2 \tag{7.2.8}$$

with $$s_{\Delta ij} = s_{ij} - \Delta_i sat(\frac{s_{ij}}{\Delta_i}) \qquad j=1,2,\ldots,n \tag{7.2.9}$$

where *sat*(.) is the saturation function.

Let

$$g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = [M(q)\ddot{q}_r + f_1(q,\dot{q})] - \frac{1}{2}\dot{M}(q)s_{\Delta 1} \tag{7.2.10}$$

$$g_2(q,p,\dot{p},\ddot{p}_r) = I_m\ddot{p}_r + f_2(q,p,\dot{p}) \tag{7.2.11}$$

$$g_3(q-p) = [K_{11} \cdots K_{1n}; \cdots ; K_{n1} \cdots K_{nn}]^T \tag{7.2.12}$$

where K_{ij} is the *ij*–th entry of $K(q–p)$. Then eqs. (7.2.6) and (7.2.7) become:

$$M(q)\dot{s}_1 + \frac{1}{2}\dot{M}(q)s_{\Delta 1} = Kp - g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) \tag{7.2.13}$$

$$I_m\dot{s}_2 = u - g_2(q,p,\dot{p},\ddot{p}_r) \tag{7.2.14}$$

Although $g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})$, $g_2(q,p,\dot{p},\ddot{p}_r)$ and $g_3(q-p)$ are unknown, neural networks can be used to learn and approximate them. Assume that the corresponding approximation error vectors of the neural networks with finite number of neurons are $d_1(t)$, $d_2(t)$ and $d_3(t)$. Then $g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})$, $g_2(q,p,\dot{p},\ddot{p}_r)$ and $g_3(q-p)$ can be expressed as:

$$g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = g_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) + d_1(t) \tag{7.2.15}$$

$$g_2(q,p,\dot{p},\ddot{p}_r) = g_{N2}(q,p,\dot{p},\ddot{p}_r) + d_2(t) \tag{7.2.16}$$

$$g_3(q-p) = g_{N3}(q-p) + d_3(t) \tag{7.2.17}$$

$$K(q-p) = K_N(q-p) + K_d(t) \tag{7.2.18}$$

where $g_{Ni}(\cdot)$ $(i=1,2,3)$ are realized by neural networks; $K_N(q-p)$ and $K_d(t)$ are formulated from $g_{N3}(\cdot)$ and $d_3(t)$ according to eq. (7.2.12). By proper neural network design, $d_1(t)$, $d_2(t)$ and $d_3(t)$ are reducible and can be confined to be bounded by any specified small constant ε_{ij}.

$$|d_{ij}(t)| < \varepsilon_{ij} \qquad i=1,2,\; j=1,\ldots,n \tag{7.2.19}$$

$$|d_{3j}(t)| < \varepsilon_{3j} \qquad j=1,2,\ldots,n^2 \tag{7.2.20}$$

A priori information about the structure of functions $g_1(q,\dot{q},\ddot{q}_r,p)$ and $g_2(q,p,\dot{p},\ddot{p}_r)$ can be used to simplify the neural network structure. For example, $g_1(q,\dot{q},\ddot{q}_r,p)$ defined by eq. (7.2.10) can be approximated by three sub–neural networks as follows:

$$g_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = M_N(q)\ddot{q}_r + f_{N1}(q,\dot{q}) - g_{N11}(q,\dot{q},s_{\Delta 1}) \tag{7.2.21}$$

with

$$g_{N11}(q,\dot{q},s_{\Delta 1}) = \frac{1}{2}\dot{M}_N(q)s_{\Delta 1} \tag{7.2.22}$$

In principle, all the existing neural networks, such as sigmoidal neural networks, *RBF* networks, *CMAC*, polynomial networks, and wavelet networks, can be adopted to represent $g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})$, $g_2(q,p,\dot{p},\ddot{p}_r)$ and $g_3(q-p)$. However, using different neural networks will result in different approximation precisions and computation complexity. Here, two–layer neural networks, e.g., the localized polynomial networks with CLI cells, are adopted due to the possibility of deriving global convergent learning algorithms. Using linear parameterization, we can express $g_{Ni}(\cdot)$ $(i = 1, 2, 3)$ as follows:

$$g_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = \Phi_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\theta_1 \tag{7.2.23}$$

$$g_{N2}(q,p,\dot{p},\ddot{p}_r) = \Phi_2(q,p,\dot{p},\ddot{p}_r)^T\theta_2 \tag{7.2.24}$$

$$g_{N3}(q-p) = \Phi_3(q-p)^T\theta_3 \tag{7.2.25}$$

where θ_i $(i = 1, 2, 3)$ is the unknown output weight vector of the networks; and $\Phi_i(\cdot)$ is defined according to the structure of the used neural networks. Assume that the flexible–joint robotic system (7.2.1) and (7.2.2) are time–invariant or slowly time–varying. Then θ_i $(i = 1, 2, 3)$ can be regarded as unknown true constant parameter vectors to be learned. Let the estimated θ_i be $\hat{\theta}_i$, and the estimation error vector be $\tilde{\theta}_i = \hat{\theta}_i - \theta_i$ $(i{=}1, 2, 3)$. The outputs of the neural networks at time instant t can be expressed as:

$$\hat{g}_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = \Phi_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\hat{\theta}_1(t) \tag{7.2.26}$$

$$\hat{g}_{N2}(q,p,\dot{p},\ddot{p}_r) = \Phi_2(q,p,\dot{p},\ddot{p}_r)^T\hat{\theta}_2(t) \tag{7.2.27}$$

$$\hat{g}_{N3}(q-p) = \Phi_3(q-p)^T\hat{\theta}_3(t) \tag{7.2.28}$$

Thus, the following direct adaptive neurocontrol algorithm is proposed:

$$u = -D_2 s_2 + \hat{g}_{N2}(q,p,\dot{p},\ddot{p}_r) \tag{7.2.29}$$

$$p_d = \hat{K}_N^{-1}[-D_1 s_1 + \hat{g}_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1})] \tag{7.2.30}$$

where D_1 and D_2 are diagonal positive definite matrices; $\hat{K}_N$ is derived from eqs. (7.2.12) and (7.2.28). Define:

$$\overline{d}_1(t) = K_d(t)p - d_1(t) \tag{7.2.31}$$

By the property of the control task, $|p_i| < c_i$. Therefore,

$$|\overline{d}_{1i}(t)| = |\sum_{j=1}^{n}(K_{dij}(t)p_j - d_{1i}(t)| \leq \sum_{j=1}^{n}|K_{dij}(t)||p_j| + |d_{1i}(t)| \leq \sum_{j=1}^{n}\varepsilon_{3ij}c_i + \varepsilon_{1i} \equiv \overline{\varepsilon}_{1i} \tag{7.2.32}$$

Let:

$$\Delta_1 = \max_j\{[\overline{\varepsilon}_{1j} + \varepsilon_{12j}]/D_{1jj}\} \quad \Delta_2 = \max_j\{\varepsilon_{2j}/D_{2jj}\} \qquad j=1,\ldots,n \tag{7.2.33}$$

where

$$\varepsilon_{12j} = \Delta_2\sum_{i=1}^{n}|\hat{K}_{Nji}|/\lambda_{2i} \qquad \Delta_{12} = \max_j\{\varepsilon_{12j}/D_{1jj}\} \tag{7.2.34}$$

and $\hat{K}_{Nji}$ is the ji–th entry of the estimated stiffness matrix $\hat{K}_N$ and their upper boundaries can be estimated from the robot design, since $\underline{K}_i \leq K_i \leq \overline{K}_i$ with the boundary matrices known; λ_{2i} is the i–th diagonal entry of matrix Λ_2. If the network weights are updated as

$$\dot{\hat{\theta}}_1 = -\Gamma_1\Phi_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})s_{\Delta 1} \tag{7.2.35}$$

$$\dot{\hat{\theta}}_2 = -\Gamma_2\Phi_2(q,p,\dot{p},\ddot{p}_r)s_{\Delta 2} \tag{7.2.36}$$

$$\dot{\hat{\theta}}_3 = \Gamma_3\Phi_3(q-p)s_3(s_{\Delta 1},p) \tag{7.2.37}$$

where $\Gamma_i\,(i=1,2,3)$ are symmetric positive definite matrices; and $s_3(s_{\Delta 1},p)$ is defined from re–ordering the right–hand side of the following equation:

$$g_3(q-p)^T s_3(s_{\Delta 1},p) = s_{\Delta 1}^T K(q-p)p \tag{7.2.38}$$

For example, when $n = 2$, $s_3(s_{\Delta 1},p)$ can be easily determined as follows:

$$s_{\Delta 1}^T K(q-p)p = [s_{\Delta 11} \quad s_{\Delta 12}]\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = g_3(q-p)^T s_3(s_{\Delta 1},p) \tag{7.2.39}$$

with

$$g_3(q-p) = [K_{11} K_{12} K_{22}]^T \tag{7.2.40}$$

$$s_3(s_{\Delta 1},p) = [s_{\Delta 11}p_1 \quad s_{\Delta 12}p_1 + s_{\Delta 11}p_2 \quad s_{\Delta 12}p_2]^T \tag{7.2.41}$$

Here, the symmetry of $K(q-p)$ is utilized to reduce the dimension of $s_3(s_{\Delta 1},p)$.

Theorem 7.2.1:

Assume that Φ_1 and Φ_3 in eqs. (7.2.35) and (7.2.37) satisfy the persistent excitation (PE) condition. The neurocontrol law (7.2.29) and (7.2.30) with eqs. (7.2.35) – (7.2.37) as the adaptation/learning algorithm can make all the signals in the adaptive neurocontrol system uniformly bounded, and the tracking errors will asymptotically converge to a small neighborhood of zero.

Proof: The proof can be found in appendix A.1.

***Remark** 7.2.1*: To compute control vector $u(t)$ by eq. (7.2.29), we need $\dot{p}_d$ and $\ddot{p}_d$, which are related to s_2, $\dot{p}_r$ and $\ddot{p}_r$. Differentiating eq. (7.2.30), we have:

$$\dot{p}_d = \hat{K}_N^{-1}[\dot{\hat{g}}_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) - D_1\dot{s}_1 - \dot{\hat{K}}_N p_d] \tag{7.2.42}$$

$$\ddot{p}_d = \hat{K}_N^{-1}[\ddot{\hat{g}}_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) - D_1\ddot{s}_1 - 2\dot{\hat{K}}_N\dot{p}_d - \ddot{\hat{K}}_N p_d] \tag{7.2.43}$$

where $\dot{\hat{K}}_N$ can be directly determined from eqs. (7.2.12), (7.2.25) and (7.2.37), while $\ddot{\hat{K}}_N$ can be determined from differentiating eq. (7.2.12) which is related to eq. (7.2.37) through eqs. (7.2.17), (7.2.25) and (7.2.28). Thus, $\ddot{\hat{K}}_N$ is a function matrix of $q, p, s_{\Delta 1}, p_d$ and their derivatives. Also,

$$\dot{\hat{g}}_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = \dot{\Phi}_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\hat{\theta}_1(t) + \Phi_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\dot{\hat{\theta}}_1(t) \tag{7.2.44}$$

$$\ddot{\hat{g}}_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = \ddot{\Phi}_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\hat{\theta}_1(t) + 2\dot{\Phi}_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\dot{\hat{\theta}}_1(t) + \Phi_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\ddot{\hat{\theta}}_1(t) \tag{7.2.45}$$

where $\dot{\hat{\theta}}_1(t)$ is determined by eq. (7.2.35); $\ddot{\hat{\theta}}_1(t)$ can be determined from differentiating eq. (7.2.35); $\dot{\Phi}_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})$ and $\ddot{\Phi}_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})$ are function matrices of $q,\dot{q},\ddot{q}_r,s_{\Delta 1}$ and their first and second order derivatives; $\dot{s}_1$ and $\ddot{s}_1$, thus $\dot{s}_{\Delta 1}$ and $\ddot{s}_{\Delta 1}$, are related to $\ddot{q}$ and $q^{(3)}$, which can be estimated as follows, by solving and differentiating eqs. (7.2.13) and (7.2.21):

$$\dot{s}_1 = \hat{M}_N(q)^{-1}[\hat{K}_N p - \hat{h}_{N1}(q,\dot{q},\ddot{q}_r) + \varepsilon_1(t)] \tag{7.2.46}$$

$$\ddot{s}_1 = \hat{M}_N(q)^{-1}[-\dot{\hat{M}}_N(q)\dot{s}_1 + \hat{K}_N\dot{p} + \dot{\hat{K}}_N p - \dot{\hat{h}}_{N1}(q,\dot{q},\ddot{q}_r) + \dot{\varepsilon}_1(t)] \tag{7.2.47}$$

where

$$h_1(q,\dot{q},\ddot{q}_r) = g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) + \frac{1}{2}\dot{M}(q)s_{\Delta 1} = M(q)\ddot{q}_r + f_1(q,\dot{q}) \equiv h_{N1}(q,\dot{q},\ddot{q}_r) + h_d(t) \tag{7.2.48}$$

$$\varepsilon_1(t) = [\tilde{M}_N(q) - M_d(t)]\dot{s}_1 + [K_d(t) - \tilde{K}_N]p + \tilde{h}_{N1}(q,\dot{q},\ddot{q}_r) - h_d(t) \tag{7.2.49}$$

and $\varepsilon_1(t)$ and $\dot{\varepsilon}_1(t)$ reflect the estimation errors. $M_d(t)$, $K_d(t)$, and $h_d(t)$ are the neural network representation errors of their corresponding terms, which can be constrained to be uniformly bounded by any small constants. Since Φ_1 and Φ_3 in eqs. (7.2.35) and (7.2.37) are assumed to satisfy the persistent excitation (PE) condition, then $\tilde{\theta}_i(t)$, and thus the terms with ~ in eq. (7.2.49), will tend to zero. Therefore, $\varepsilon_1(t)$ and $\dot{\varepsilon}_1(t)$ can be confined to have small norms. Finally, $\ddot{q}$ and $q^{(3)}$ can be computed as:

$$\ddot{q} = \dot{s}_1 + \ddot{q}_d - \Lambda_1\dot{e} \tag{7.2.50}$$

$$q^{(3)} = \ddot{s}_1 + q_d^{(3)} - \Lambda_1(\ddot{q} - \ddot{q}_d) \tag{7.2.51}$$

In conclusion, by using eqs. (7.2.42) – (7.2.51), $\dot{p}_d$ and $\ddot{p}_d$ can be computed with the measurements of q, $\dot{q}$, p, and $\dot{p}$ only, and the desired trajectory $q_d(t) \in C^4$. Their computation errors come from the unknown $\varepsilon_1(t)$ and $\dot{\varepsilon}_1(t)$ only. Moreover, it is easy to prove that the estimation errors of $\dot{p}_d$ and $\ddot{p}_d$ affect the output tracking precision, but do not affect the system stability, as long as the estimation errors (regarded as a kind of modeling errors) are uniformly bounded. The dead–zone Δ_i ($i = 1, 2$) is enlarged accordingly.

Remark 7.2.2: From eq. (7.2.31), we know that a good approximation to the stiffness matrix $K(q-p)$ is more important in minimizing asymptotic tracking errors. This should be reflected in the network design. The larger λ_{1i} and λ_{2i} are, the smaller the tracking errors are, and the larger the control actions are. Here, the estimations of the tracking error boundaries, as shown in eqs. (A.1.9) and (A.1.11), are conservative and simple to use.

Remark 7.2.3: In the proof of the theorem, we observed that the motor angle tracking error vector $\delta(t)$ converges to a neighborhood of zero before the link angle tracking error vector $e(t)$ does. This is the inherent feature of flexible–joint robotic dynamics, or cascaded dynamic systems. Only after $p(t)$ produces desired torsional torques can the links be forced to follow desired angle trajectories. However, arbitrary joint stiffness is allowed in this neurocontrol scheme, which is different from the singular perturbation approach (e.g., Ghorbel, Spong and Hung, 1989).

Remark 7.2.4: The e_1–modification (leakage) or parameter projection techniques (Narendra and Annaswamy, 1989) can be applied to eqs. (7.2.35) – (7.2.37) to further prevent parameter adaptation from divergence.

Remark 7.2.5: It is worth indicating that in conventional robust adaptive control, the unmodeled system dynamics determine the lower boundaries of the tracking control precision. This boundaries are irreducible, unless better system models are used, or internal models are included in the control design. In adaptive neurocontrol, however, if the unmodeled system dynamics are observable through system outputs, the modeling error boundaries can always be reduced by proper neural network structure design, owing to the universal approximation ability of neural networks.

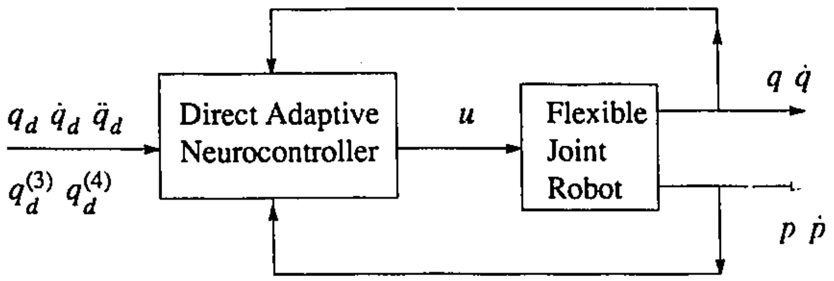


Fig. 7.1 Block diagram of the direct adaptive neurocontrol system

The new direct adaptive neurocontrol system for flexible–joint robots can be illustrated by fig. 7.1. The neurocontrol algorithm consists of the following steps: (1) error signal computing; (2) neural network weight updating; (3) desired motor trajectory estimation; and (4) control torque computing.

7.2.2 Simulation tests

In order to evaluate the proposed control algorithms, simulations were performed based on the dynamic model of the FMS flexible–joint robot. The terms in eqs. (7.1.1) and (7.1.2) are determined as follows:

$$M(q) = \begin{bmatrix} (a_1 + 2a_3c_2) & (a_2 + a_3c_2) \\ (a_2 + a_3c_2) & a_2 \end{bmatrix} \tag{7.2.52}$$

$$h(q, \dot{q}) = \begin{bmatrix} -\ a_3\dot{q}_2(2\dot{q}_1 + \dot{q}_2)s_2 + B_{j1}\dot{q}_1 \\ a_3\dot{q}_1^2\, s_2 + B_{j2}\dot{q}_2 \end{bmatrix} \tag{7.2.53}$$

A set of nominal values of the parameters (Massoud and ElMaraghy, 1993) are

$$a_1 = 2.087 \qquad a_2 = 0.084 \qquad a_3 = 0.216$$

$$I_{m1} = 0.1224\ kgm^2 \quad B_{m1} = 1.254\ Nms/rad \quad K_1 = 125.56\ Nm/rad$$

$$I_{m2} = 0.0168\ kgm^2 \quad B_{m2} = 0.119\ Nms/rad \quad K_2 = 31.27\ Nm/rad$$

$$B_{j1} = 2.041\ Nms/rad \quad B_{j2} = 0.242\ Nms/rad$$

$$l_1 = 0.40\ m \quad l_2 = 0.35\ m \quad d_w = 0.018\ m$$

$$F_{m1} = 3.5\ sgn(\dot{p}_1)\ Nm \quad F_{m2} = 1.2\ sgn(\dot{p}_2)\ Nm$$

The relation between the joint angles and the end–effector position is:

$$x = l_1 cos(q_1) + l_2 cos(q_1 + q_2) \tag{7.2.54}$$

$$y = l_1 sin(q_1) + l_2 sin(q_1 + q_2) \tag{7.2.55}$$

In the following, we assume that the model and the above parameters of the flexible–joint robot are unknown. The adaptive neurocontrol laws (7.2.29) and (7.2.30) with eqs. (7.2.35) – (7.2.37) as the learning algorithms are applied to control the end–effector trajectory. Due to their simplicity, the Gaussian RBF networks are used to model the robot dynamics. Some a priori information about the structure of functions $g_1(q, \dot{q}, \ddot{q}_r, p)$ and $g_2(q, p, \dot{p}, \ddot{p}_r)$ is used to simplify the neural network structure design. To ensure the approximation precision and task space coverage, 2000 neurons are used in the neurocontroller.

As indicated before, $\dot{p}_d$ and $\ddot{p}_d$ are required to compute control vector $u(t)$ by eq. (7.2.29). To estimate $\dot{p}_d$ and $\ddot{p}_d$ using eqs. (7.2.42) – (7.2.51), $\Phi_i(\cdot)$ $(i = 1, 2, 3)$ has to satisfy the persistently exciting conditions. However, it is difficult to verify if $\Phi_i(\cdot)$ $(i = 1, 2, 3)$ is persistent excitation or not for a given desired trajectory. Notice that the estimation errors of $\dot{p}_d$ and $\ddot{p}_d$ affect the output tracking precision, but do not affect the system stability as long as the estimation errors are uniformly bounded (ElMaraghy and Liang, 1993). Thus, we use $\ddot{q}_d$ to replace $\ddot{q}$ and $q_d^{(3)}$ to replace $q^{(3)}$, in order to eliminate the PE requirement. With this modification, the neurocontrol law is greatly simplified, but at the cost of tracking precision.

Simulation 7.2.1: The first simulation is conducted based on the following straight line trajectory:

$$x_d = 0.6\ m \tag{7.2.56}$$

$$y_d = 0.3sin(\omega t - \pi/2)\ \ m \tag{7.2.57}$$

The corresponding $q_d \in C^4$ can be calculated off-line using the inverse kinematics formula. The sampling period is 2 ms. The actual end-effector trajectory of the robot during the learning stage is plotted in fig. 7.2. After about 1000 steps of learning, the tracking error becomes

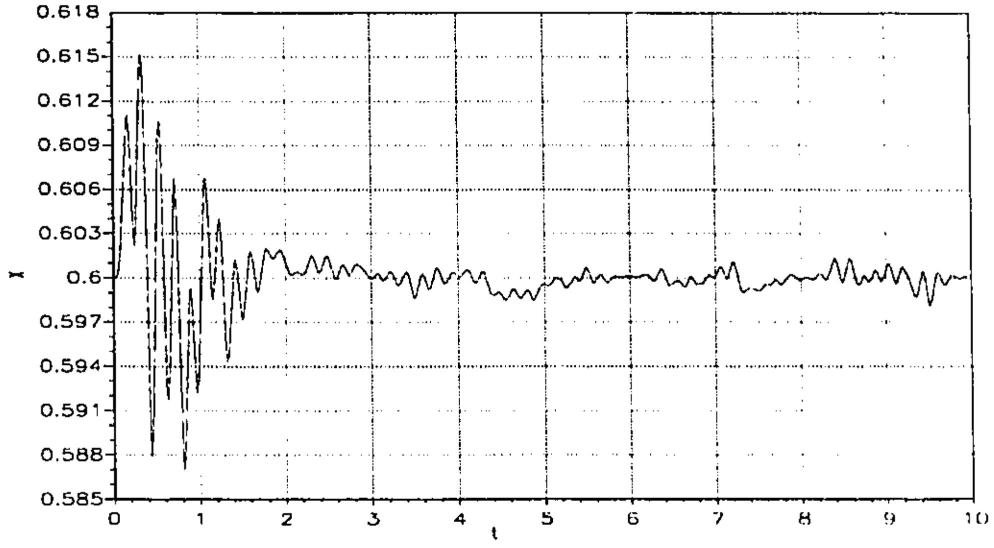


Fig. 7.2 Straight line trajectory tracking: learning stage

less than 1.5 mm. After the learning converges, the tracking error remains small as shown in fig. 7.3.

Simulation 7.2.2: The second simulation is conducted to track the following circle trajectory:

$$x_d = 0.64 + 0.8cos(\omega t)\ m \tag{7.2.58}$$

$$y_d = 0.8sin(\omega t)\ \ m \tag{7.2.59}$$

The sampling period is 2 ms. The actual end-effector trajectory of the robot during the learn-

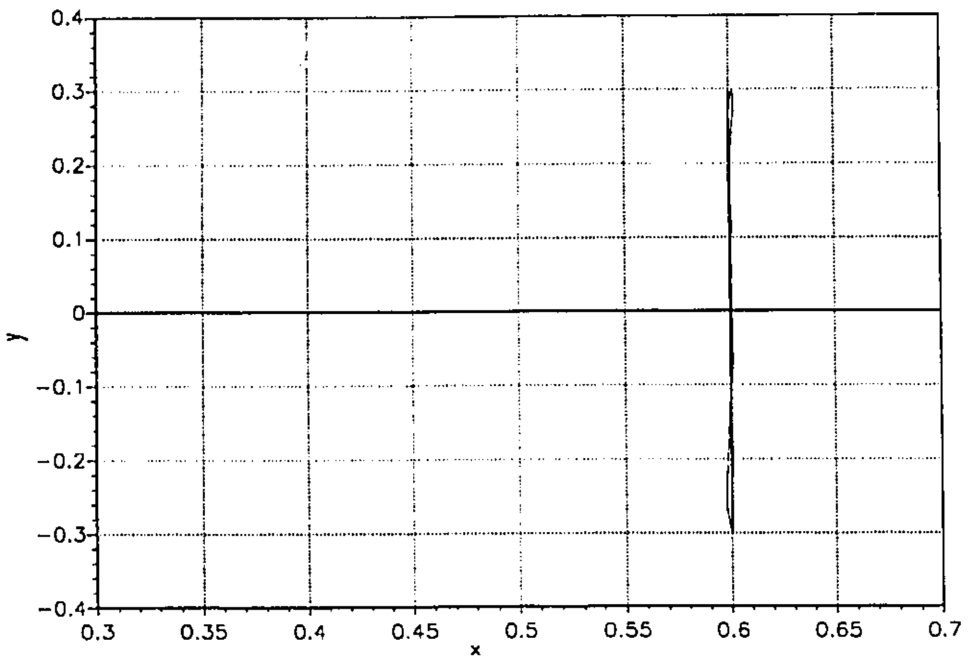


Fig. 7.3 Straight line trajectory tracking: working stage

ing stage is plotted in fig.7.4. After about 1400 steps of learning, the tracking error becomes less than 3 mm. After the learning converges, the tracking error remains small as shown in fig. 7.5.

Simulation 7.2.3: The third simulation is conducted to track the rectangle trajectory with its four vertices as (0.7,0.2), (0.55,0.2), (0.55,–0.2), (0.7, –0.2). The sampling period is 2 ms. The actual end–effector trajectory of the robot during the learning stage is plotted in fig.7.6. After about 2000 steps of learning, the tracking error becomes less than 2 mm. After the learning converges, the tracking error remains small as shown in fig. 7.7.

The simulation results show that the direct adaptive neurocontrol scheme is feasible and robust to the neural network representation errors. In the simulation, the number of the RBF neurons is fixed. The adaptation algorithm proposed in section 3.3 can be applied here to determine the optimal number for the given problem.

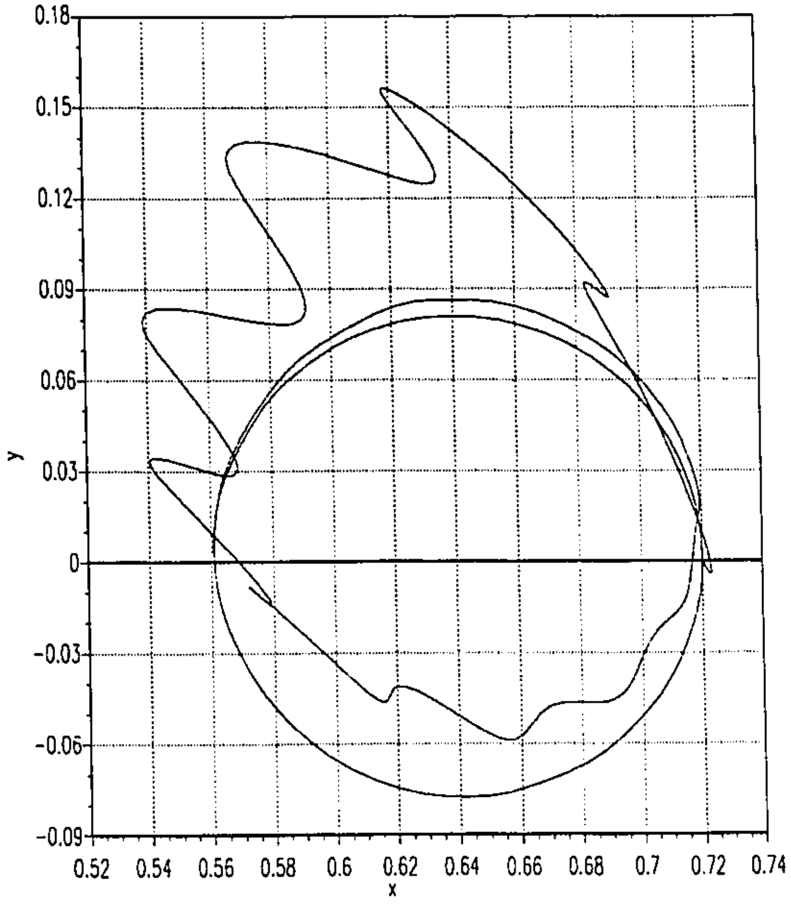


Fig. 7.4 The circle trajectory tracking: learning stage

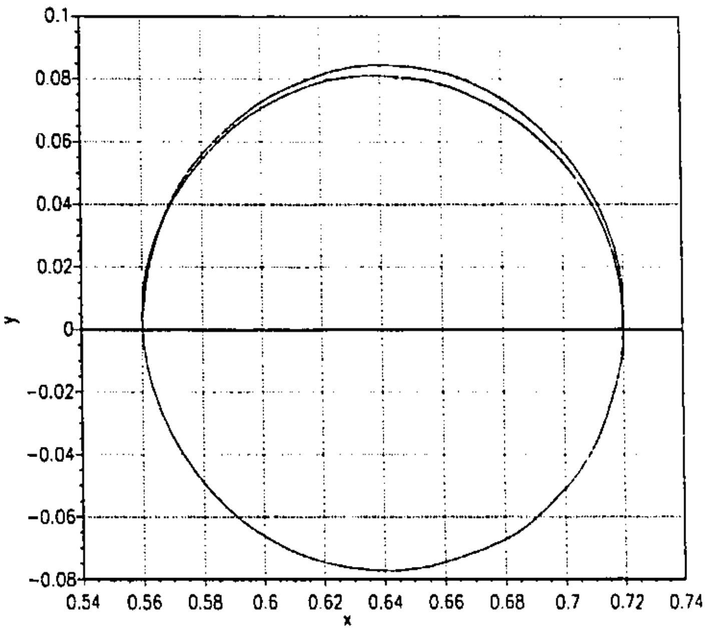


Fig. 7.5 The circle trajectory tracking: working stage

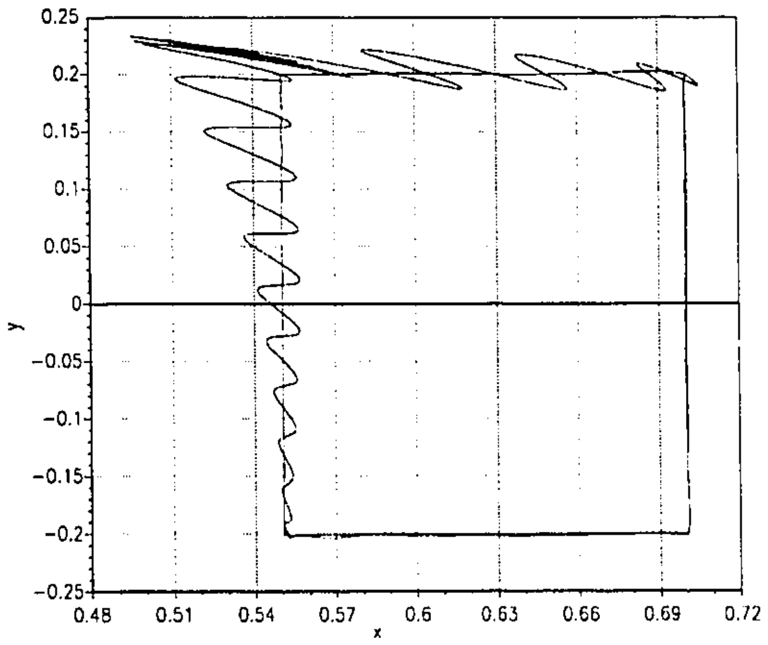


Fig. 7.6 The rectangle trajectory tracking: learning stage

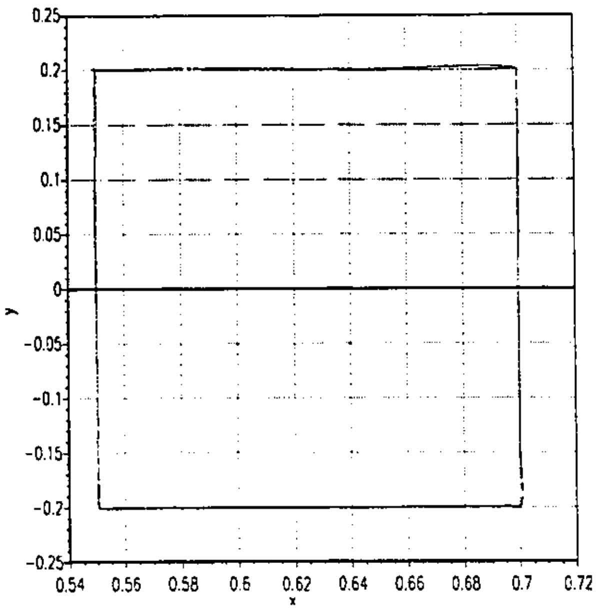


Fig. 7.7 The rectangle trajectory tracking: working stage

7.3 MODEL REFERENCE ADAPTIVE NEUROCONTROL SCHEME

7.3.1 Re–formulating the dynamic model of flexible–joint robots

A widely studied model of an n–link flexible–joint robot manipulator is given by eqs. (7.1.1) and (7.1.2). In order to obtain the direct input–output relation between the joint angle vector q and control vector u, we differentiate eq. (7.1.1) and have:

$$M(q)q^{(3)} + \dot{M}(q)\ddot{q} + \dot{H}(q,\dot{q}) + K(\dot{q} - \dot{p}) = 0 \qquad (7.3.1)$$

$$M(q)q^{(4)} + 2\dot{M}(q)q^{(3)} + \ddot{M}(q)\ddot{q} + \ddot{H}(q,\dot{q}) + K(\ddot{q} - \ddot{p}) = 0 \qquad (7.3.2)$$

From eqs. (7.1.1), (7.1.2) and (7.3.1), we know that:

$$\ddot{q} = -M(q)^{-1}[H(q,\dot{q}) + K(q - p)] \equiv -f_1(q,\dot{q},p) \qquad (7.3.3)$$

$$\ddot{p} = I_m^{-1}[u - B_m\dot{p} + K(q - p)] \equiv I_m^{-1}u - f_2(q,p,\dot{p}) \qquad (7.3.4)$$

$$q^{(3)} = -M(q)^{-1}[\dot{M}(q)\ddot{q} + \dot{H}(q,\dot{q}) + K(\dot{q} - \dot{p})] = -f_3(q,\dot{q},p,\dot{p}) \quad (7.3.5)$$

Substituting eqs. (7.3.3)–(7.3.5) into eq. (7.3.2) gives:

$$M(q)q^{(4)} + h_m(q,\dot{q},p,\dot{p}) = R_m u \quad (7.3.6)$$

or

$$q^{(4)} + h(q,\dot{q},p,\dot{p}) = R(q)u \quad (7.3.7)$$

where $R(q) = M(q)^{-1}R_m = M(q)^{-1}KI_m^{-1}$ is invertible, but not necessarily positive definite. Also, $\| R(q) \| < R_0$, where R_0 is a positive constant. Eq. (7.3.7) is called the input–output model of a flexible–joint robot, with eq. (7.1.2) or (7.3.4) as its internal dynamics. Assume that q, $\dot{q}$, p, and $\dot{p}$ are measurable. It can be seen that if $h(q,\dot{q},p,\dot{p}) \in R^n$ and $R(q) \in R^{n\times n}$ are known, then eq. (7.3.7) is state feedback linearizable.

In practice, the dynamic model of an n–link flexible–joint robot manipulator, described by eqs. (7.1.1) and (7.1.2), usually contains structural and parametric uncertainty, as stated in section 1.3.1. The terms $M(q)$, $H(q,\dot{q})$, K, I_m, and B_m in eqs. (7.1.1) and (7.1.2) may be determined using common identification procedures. However, these procedures imply a lot of off–line work and the validity of the model is to be tested. For multi–link robots, they represent a great amount of computation, which may be implemented using neural network hardware. Therefore, from both the modeling and computation viewpoints, the neurocontrol approach has some advantage. In the following, we assume that $h(q,\dot{q},p,\dot{p})$ and $R(q)$ are unknown, and are approximated using single–hidden layer neural networks, such as sigmoidal neural networks and *RBF* networks. We have

$$h(q,\dot{q},p,\dot{p}) = W_h^T\Phi_h(q,\dot{q},p,\dot{p}) + d_h(t) \quad (7.3.8)$$

$$R(q) = [R_1(q) \ . \ . \ . \ R_n(q)] \qquad with \quad R_i(q) = W_i^T\Phi_i(q) + d_i(t) \quad (7.3.9)$$

where $W_h \in R^{L\times n}$ and $W_i \in R^{L_i\times n}$ are the unknown neural network weight matrices; $\Phi_h(q,\dot{q},p,\dot{p}) \in R^L$ and $\Phi_i(q) \in R^{L_i}$ are the output vectors of the hidden layer neurons of the corresponding neural networks; L and L_i are the numbers of the hidden neurons; and $d_h(t)$ and $d_i(t)$ are the neural network representation error vectors.

In the following, the input–output model (7.3.7) with unknown $h(q,\dot{q},p,\dot{p})$ and $R(q)$ is used to derive a model reference adaptive neurocontrol scheme.

7.3.2 Model reference adaptive neurocontrol algorithm

For the input–output model (7.3.7) with relative degree 4, we define the following reference model:

$$A_m(D)q_r = r \tag{7.3.10}$$

where $D = \frac{d}{dt}$ is the differential operator; q_r is the reference output vector of the joint angle vector q; r is the external input vector; and $A_m(D)$ is the desired monic Hurwitz polynomial chosen by designers, and can be in the following forms:

$$A_m(D) = D^4 + a_{m1}D^3 + a_{m2}D^2 + a_{m3}D + a_{m4} \equiv (D + \lambda_0)A_{m0}(D) \tag{7.3.11}$$

$$or \quad A_m(D) = (D + \lambda_0)^4 = (D + \lambda_0)A_{m0}(D) \qquad \lambda_0 > 1/2 \tag{7.3.12}$$

Assume that the desired trajectory of the end–effector of a robot manipulator is expressed in the joint coordinates as $q_d \in C^4 \cap L_\infty$. Then

$$r(t) = A_m(D)q_d(t) = q_d^{(4)}(t) + a_{m1}q_d^{(3)}(t) + a_{m2}\ddot{q}_d(t) + a_{m3}\dot{q}_d(t) + a_{m4}q_d(t) \tag{7.3.13}$$

Set $q_r^{(i)}(0) = q_d^{(i)}(0)$ for i=0, . . . , 4. Then the reference model (7.3.10) guarantees that $q_r^{(i)}(t) = q_d^{(i)}(t)$ for i=0, . . . , 4.

Let (c, A_0, b) be a minimal realization of $A_{m0}(D)^{-1}$ in state space, i.e.:

$$c(sI - A_0)^{-1}b = A_{m0}(s)^{-1} \tag{7.3.14}$$

Let $E(D)$ be a given monic Hurwitz polynomial of degree 3. Then $F(D)$, a monic polynomial of degree 3, and $G(D)$, a polynomial of degree 3, can be determined uniquely from the following Bezout equation with chosen $E(D)$ and $A_m(D)$:

$$F(D)D^4 + G(D) = E(D)A_m(D) \tag{7.3.15}$$

Let $(c_1,\ A_1,\ b_1)$ be a minimal realization of $[F(D) - E(D)]/E(D)$. Notice that $[F(D)-E(D)]$ is a polynomial of degree 2.

Define the following error vectors:

$$e = q - q_r = q - q_d \tag{7.3.16}$$

$$\bar{e} = e + \eta_0 \tag{7.3.17}$$

$$\bar{e}_\Delta = [\bar{e}_{\Delta 1} \cdot \cdot \cdot \bar{e}_{\Delta n}]^T \qquad with \qquad \bar{e}_{\Delta i} = \bar{e}_i - \Delta sat(\bar{e}_i/\Delta) \tag{7.3.18}$$

where Δ stands for the dead–zone, or the upper boundary of the tracking errors $\bar{e}_i$ or e_i. It is determined by the approximation precision of neural networks. Let $\hat{W}_h$ be the estimation of W_h, and $\hat{W}_i$ be the estimation of W_i. Their estimation error matrices are $\tilde{W}_h$ and $\tilde{W}_i$, respectively. Thus,

$$\tilde{W}_h = \hat{W}_h - W_h \qquad \tilde{W}_i = \hat{W}_i - W_i \quad (i = 1,\ \ldots,\ n) \tag{7.3.19}$$

$$W = [W_h^T\ W_R^T]^T \quad with \quad W_R^T \equiv [W_1^T \cdot \cdot \cdot W_n^T\ ] \tag{7.3.20}$$

$$\tilde{W} = \hat{W} - W \tag{7.3.21}$$

For $h(q,\dot{q},p,\dot{p})$ and $R(q)$ represented by eqs. (7.3.8) and (7.3.9), we have

$$\hat{h}(q,\dot{q},p,\dot{p}) = \hat{W}_h^T \Phi_h(q,\dot{q},p,\dot{p}) \tag{7.3.22}$$

$$\hat{R}(q) = [\hat{R}_1(q)\ .\ .\ .\ \hat{R}_n(q)] \quad with \quad \hat{R}_i(q) = \hat{W}_i{}^T \Phi_i(q) \tag{7.3.23}$$

$$or \qquad \hat{R}(q) = \hat{W}_R{}^T \Phi_R \quad with \quad \Phi_R = diag\{\Phi_1(q) .\ .\ .\ \Phi_n(q)\} \tag{7.3.24}$$

we give the following model reference adaptive neurocontrol law:

$$u = \hat{R}(q)^{-1}[r - \frac{G(D)}{E(D)}q + \hat{W}_h{}^T \overline{\Phi}_h - \hat{W}_R{}^T (c_1 U)^T] \tag{7.3.25}$$

where

$$\overline{\Phi}_h = \frac{F(D)}{E(D)} \Phi_h(q,\dot{q},p,\dot{p}) \tag{7.3.26}$$

$$\dot{U} = A_1 U + b_1 u^T \Phi_R^T \tag{7.3.27}$$

The learning algorithm of weight matrix W is:

$$\dot{\hat{W}} = -\gamma \psi \bar{e}_\Delta^T \tag{7.3.28}$$

η_0 in eq. (7.3.17) and ψ are determined by the following filters:

$$\dot{Z} = A_0 Z + b\overline{\Phi}(q,\dot{q},p,\dot{p},u)^T \tag{7.3.29}$$

$$\psi^T = cZ \tag{7.3.30}$$

$$\overline{\Phi}(q,\dot{q},p,\dot{p},u) = [\ \overline{\Phi}_h^T\ ,\ \ -c_1 U - u^T \Phi_R^T\ ]^T \tag{7.3.31}$$

$$\dot{X} = A_0 X - Z\dot{\hat{W}} \tag{7.3.32}$$

$$\dot{\eta}_0 = -\lambda_0 \eta_0 + (cX)^T \tag{7.3.33}$$

Theorem 7.3.1:

The model reference adaptive neurocontrol law (7.3.25) with eq. (7.3.28) as the learning algorithm of the weight matrix W ensures that all the signals in the adaptive neurocontrol system are uniformly bounded, provided that their initial values are bounded. Moreover, we have:

$$\lim_{t\to\infty} \bar{e}_\Delta(t) = 0 \qquad \lim_{t\to\infty} \eta_0(t) = 0 \qquad \lim_{t\to\infty} X(t) = 0 \qquad \lim_{t\to\infty} |e_i(t)| < \Delta \tag{7.3.34}$$

This implies that the end–effector of the controlled FJR with unknown dynamics can track any given trajectory with user–specified precision. p and $\dot{p}$ are uniformly bounded. The controller is also robust to the representation errors of the neural networks and bounded disturbances.

Proof: The proof of theorem 7.3.1 can be found in Appendix A.2.

Remark 7.3.1: Since there is some a priori information about the structures and parameters of $h(q,\dot{q},p,\dot{p})$ and $R(q)$, it can be incorporated into the neurocontroller design to achieve better generalization, as indicated before.

Remark 7.3.2: The tracking error e and the neural network representation error $d(t)$ are correlated through eq. (A.2.6), where the filtered error vector $\bar{d}(t) = A_{m0}(D)^{-1}d(t)$ is much smoother than $d(t)$. Thus, $|\bar{d}_i(t)| < \lambda_0 \Delta$ imposes milder conditions on $d(t)$ due to the filtering. For the same precision requirement of neural network approximation, the larger λ_0 is, the smaller Δ is, and the smaller the tracking errors are. λ_0 is upper–limited by the response speed of actuators and the lowest elastic mode of the links.

Remark 7.3.3: A flexible–joint robot can also be represented by the following task space model:

$$x^{(4)} + f(x, \dot{x}, p, \dot{p}) = Q(x)u \tag{7.3.35}$$

where $x = [x_1, ..., x_m]^T$ stands for the position vector of the robot end–effector in the task/world space ($1 \le m \le 6$). The definitions of p and u remain the same. Then the model reference adaptive neurocontrol approach presented here can be applied directly to eq. (7.3.35) with some notation changes only.

Remark 7.3.4: Unlike the direct adaptive neurocontrol law presented in section 7.2, the persistent excitation condition is not necessary for the model reference adaptive neurocontrol law (7.3.25) with eq. (7.3.28) as the learning algorithm.

7.4 SUMMARY

This chapter presents a novel direct adaptive neurocontrol scheme and a model reference adaptive neurocontrol scheme for general flexible–joint robots. The former requires the PE condition, while the latter does not. Both of them require little a priori information about the dynamics of the controlled flexible–joint robots, and are applicable to robots with arbitrary joint flexibility and the measurements of link and motor angles and velocities only. Unlike many existing neurocontrol schemes, the presented adaptive neurocontrol schemes can guarantee the global stability of the robotic control systems. The neurocontrollers are robust to the representation errors of the neural networks with a finite number of neurons and bounded additive external disturbances. These schemes show the close relationship between the matured adaptive control theory and the emerging neurocontrol theory.

Liang and ElMarghy (1994d) introduced another stability–based adaptive neurocontrol scheme using the variable index control approach, which can be regarded as the application of the theory presented in section 6.4. To the best of our knowledge, the neurocontrol schemes presented in this chapter are the first ones to produce globally stable adaptive neurocontrol systems for flexible–joint robots.

As indicated before, stability–based neurocontrollers can be developed using on–line learning. The backprop–based neurocontrollers are usually developed using off–line learning. However, for both of them, the usage of the neurocontrollers can be divided into a learning stage and a working stage. The working stage is defined as the time when the neurocontrollers have performed the control tasks well, and is always on–line. The control performance of neurocontrollers are usually unacceptable during the learning stage, whether on–line or off–line. Only when the neurocontrollers reach the working stage can they really accomplish the desired control tasks.

CHAPTER 8

EXPERIMENTAL TESTS

8.1 INTRODUCTION

The purpose of conducting experiments is to test the real–time features, and numerical characteristics of the proposed neurocontrol schemes. Also, there are more uncertain factors in physical setups than found in simulation models, which often present challenges to many model–based control approaches.

There are some experimental results of neurocontrol schemes (e.g., Yegerlehner and Meckl, 1993) in the published literatures. However, we are not aware of any experimental results on neurocontrol of flexible–joint robots.

The general procedure of control systems development involves the following steps: (a) control task and precision definition; (b) control system hardware setup; (c) modeling, parameter identification and model verification; (d) system dynamic analysis; (e) control scheme selection and simulation test; (f) real–time control code programming and implementation; and (g) control system testing and modification.

Since the FMS two–link flexible–joint robot has been set up already and the new neurocontrol schemes have been developed and tested by simulation, only steps (f) and (g) remain to be done. For the neurocontrol approach, it is not necessary to conduct step (c), because a first–principle dynamic model of the controlled system is not required.

The rest of this chapter is organized as follows. Section 8.2 gives a brief description of the experimental robot, including its structure, dynamic model and the system precision of the experimental setup. Section 8.3 presents the neurocontrol algorithms used in the experiments. Section 8.4 presents the experimental results and discussions.

8.2 EXPERIMENTAL ROBOT DESCRIPTION

8.2.1 Structure of the experimental robot

The FMS experimental robot is shown in fig. 8.1. It has two links which are driven

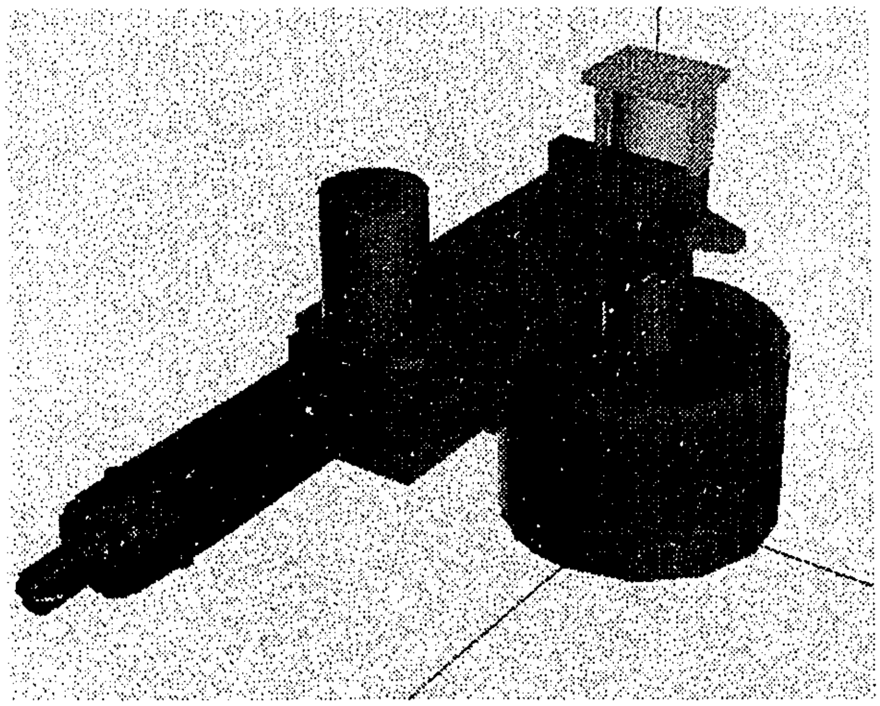


Fig. 8.1 The overall structure of the experimental robot

by two direct drive motors. The links and the motors are connected through torsional springs to introduce low joint stiffness, i.e., high joint flexibility. The two–link robot forms a four–degrees–of–freedom system—two for the link positions and two for the motor rotations (fig.

8.2 and 8.3). The measured signals are the two link angles from incremental optical encod-

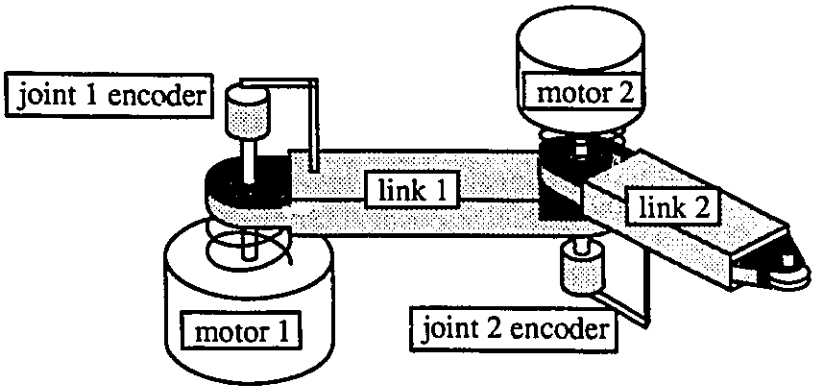


Fig. 8.2 The mechanical composition of the experimental robot

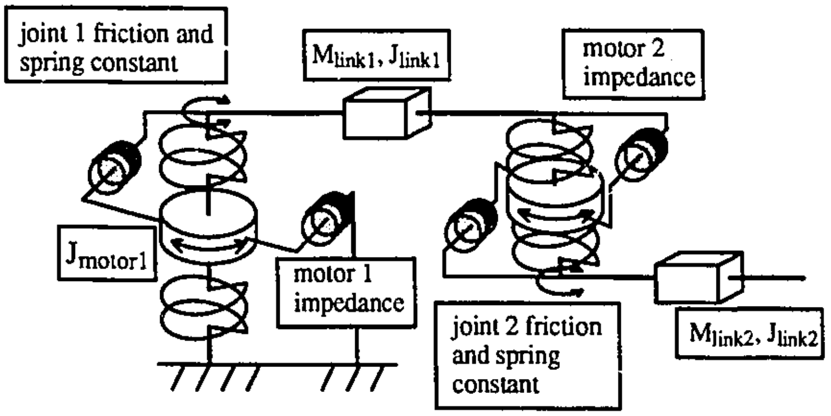


Fig. 8.3 The mechanical relationship of the experimental robot

ers, two motor rotor angles from the encoders, and the two motor rotor angular velocities from the motor servo–systems. The motor encoders are built in, while the link encoders are mounted on the stator shafts. The control algorithm is executed on the dSpace DSP processor. The outputs of the controller and the inputs to the motors are the control voltages, which are transformed into control torques by the direct drive motor servo–systems. The overall control system setup of the experimental robot is illustrated in fig. 8.4.

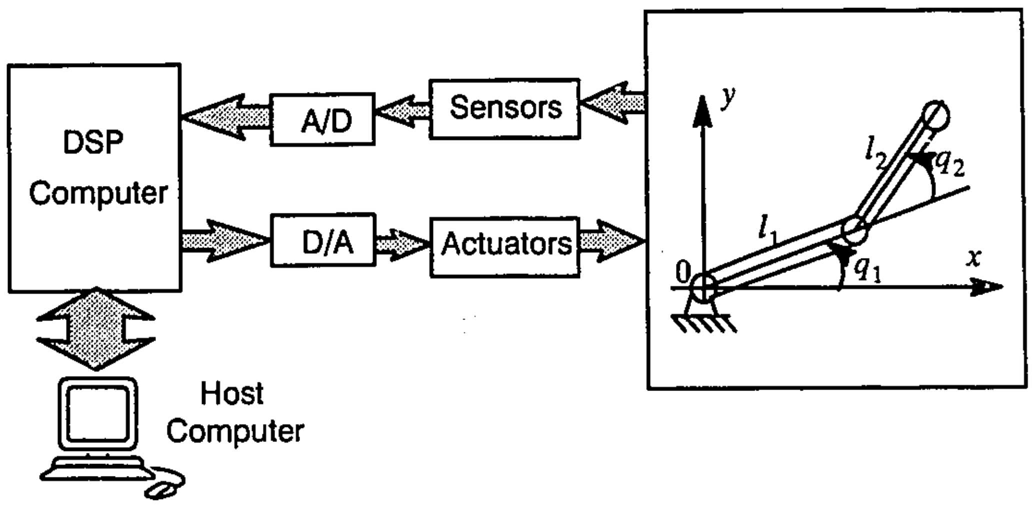


Fig. 8.4 The control system setup of the experimental robot

8.2.2 Experimental system modeling

The dynamic model of the experimental robot is the same as defined by eqs. (7.1.1) and (7.1.2). The nominal parameters of the robots are given in section 7.2.2.

There are both parametric and structural uncertainties in the model. The nominal parameters were obtained through both parameter identification using MATRIX$_X$ and estimation using I-DEAS solid modeling. There are estimation errors in the parameters. The structural uncertainty comes from: (1) neglecting the servo–motor dynamics, (2) non–ideal alignment of the joints due to the torsional spring connection, (3) inaccurate modeling of friction, and (4) pulling torque from the power and signal cables.

The recommended transfer functions of the motor dynamics from the producer are nonlinear voltage–to–torque gains. In the experiments, the gains are regarded as constants.

The friction torques affect both link dynamics and motor dynamics. Assume that $f_j(\dot{q})$ is the link friction torque vector, and is simply modeled as:

$$f_j(\dot{q}) = B_j\dot{q} \tag{8.3.1}$$

Let $B_m = diag[B_{m1}\ B_{m2}]$, where B_{mi} is the viscous friction coefficient of the ith joint. Let α_{0i} be Coulomb friction torque magnitude, $(\alpha_{0i}+\alpha_{1i})$ be the break-away friction torque magnitude, and $\dot{p}_{0i}$ be a contact surface related constant. Then the friction torque exerted on the i-th motor rotor can be approximated as:

$$F_{mi} = (\alpha_{0i} + \alpha_{1i}e^{-(\dot{p}_i/\dot{p}_{0i})^2})sgn(\dot{p}_i) + B_{mi}\dot{p}_i \qquad (8.3.2)$$

The sign function can be approximated by the following:

$$sgn(\dot{x}) \doteq \frac{\dot{x}}{|\dot{x}| + \varepsilon} \qquad \varepsilon \text{ is a small positive constant} \qquad (8.3.3)$$

The relation between the joint angles and the end-effector position (fig. 8.4) is:

$$x = l_1cos(q_1) + l_2cos(q_1 + q_2) \qquad (8.3.4)$$

$$y = l_1sin(q_1) + l_2sin(q_1 + q_2) \qquad (8.3.5)$$

and the Jacobian matrix is:

$$J(q) = \begin{bmatrix} -l_1s_1 - l_2s_{12} & -l_2s_{12} \\ l_1c_1 + l_2c_{12} & l_2c_{12} \end{bmatrix} \qquad (8.3.6)$$

Therefore,

$$det(J(q)) = -(l_1s_1 + l_2s_{12})l_2c_{12} + l_2s_{12}(l_1c_1 + l_2c_{12}) = l_1l_2s_2 \qquad (8.3.7)$$

It can be seen that the Jacobian matrix is invertible except when $sin(q_2)=0$, or $q_2=k\pi$ (k can be any integer).

The desired trajectory of an end-effector $x_d \in C^4$ is usually given in the task space. To find the desired trajectory in the joint angular coordinates, the inverse kinematics relations are required. The desired joint angles can be determined from the desired end-effector position as follows:

$$q_1 = arcsin[(x^2 + y^2 + l_1^2 - l_2^2)/(2l_1\sqrt{x^2 + y^2})] - \phi \qquad (8.3.8)$$

$$q_2 = arccos[(x^2 + y^2 - l_1^2 - l_2^2)/(2l_1l_2)] \qquad (8.3.9)$$

$$\phi = arctan(x/y) \qquad (8.3.10)$$

which gives one configuration to be used in the experiments. The continuity of q_i is used to determine the actual angle ϕ in the above equations.

The desired joint angle derivatives can be determined as follows:

$$\dot{q}_d = J(q_d)^{-1}\dot{x}_d \tag{8.3.11}$$

$$\ddot{q}_d = J(q_d)^{-1}[\ddot{x}_d - \dot{J}(q_d)\dot{q}_d] \tag{8.3.12}$$

$$q_d^{(3)} = J(q_d)^{-1}[x_d^{(3)} - 2\dot{J}(q_d)\ddot{q}_d - \ddot{J}(q_d)\dot{q}_d] \tag{8.3.13}$$

$$q_d^{(4)} = J(q_d)^{-1}[x_d^{(4)} - 3\dot{J}(q_d)q_d^{(3)} - 3\ddot{J}(q_d)\ddot{q}_d - J^{(3)}(q_d)\dot{q}_d] \tag{8.3.14}$$

8.2.3 System precision of the experimental setup

For any control system, the overall control precision which can be achieved is always confined by the precision of their components. In the experimental robots, the precision of the angular measurements from the encoders is very high. The resolution of the encoder attached to motor 1 is 1,024,000 pulses/rev; the resolution of the encoder attached to motor 2 is 655,360 pulses/rev; the resolutions of the encoders attached to both link 1 and 2 are 32,000 pulses/rev. Therefore, the maximum angular measurement error is ± 0.01225 deg. Since the encoder readings are directly counted, there is no other source of error in the angular measurements.

The angular velocities of two motor rotors are fed into the dSpace board through 16–bit A/D converter. The velocity signals are very noisy. The A/D conversion error for the velocity of motor 1 is $\pm 1.92 \times 10^{-4}$ rad/s or ± 0.011 deg/s, and the A/D conversion error for the velocity of motor 2 is $\pm 3.84 \times 10^{-4}$ rad/s or ± 0.022 deg/s. The controller outputs computed in the DSP board are converted into control voltage by the 12–bit D/A converter. The D/A conversion error of control input 1 is $\pm 100/2^{11} = \pm 0.0488$ (Nm), and the D/A conversion error of control input 2 is $\pm 30/2^{11} = \pm 0.0146$ (Nm). The steady state joint angle control errors due to the limited word length of the D/A converter are $\pm 0.0488/K_1 = \pm 3.89 \times 10^{-4}$ rad for link 1, and $\pm 0.0146/K_2 = \pm 4.67 \times 10^{-4}$ rad for link 2, without considering the friction effect.

Calibration of flexible–joint robots is more difficult than that of rigid ones, because the initial motor angles are not easy to set, and there may be joint pre–deformation. The encoders provide only relative angle measurements, and there is also task space coordinate set-up error. These factors all contribute to calibration errors in the control system. In addition to affecting the actual trajectory control precision, calibration errors also introduce structural uncertainty of the robot dynamics.

These errors will limit the control precision of the experimental robots for any control algorithms. The maximum velocity of motor 1 is 6.28 rad/sec. This means that this motor can easily become overloaded.

8.3 CONTROL ALGORITHMS

In the experiments, the direct adaptive neurocontrol scheme derived in section 7.2 is adopted in this chapter to conduct the experimental test. The feasibility of the neurocontrol algorithm was demonstrated by simulation in section 7.2.2.

The direct adaptive neurocontrol scheme is summarized in the following:

$$u = -D_2 s_2 + \hat{g}_{N2}(q, p, \dot{p}, \ddot{p}_r) \tag{8.3.1}$$

$$p_d = \hat{K}_N^{-1}[-D_1 s_1 + \hat{g}_{N1}(q, \dot{q}, \ddot{q}_r, s_{\Delta 1})] \tag{8.3.2}$$

$$\dot{p}_d = \hat{K}_N^{-1}[\dot{\hat{g}}_{N1}(q, \dot{q}, \ddot{q}_r, s_{\Delta 1}) - D_1 \dot{s}_1 - \dot{\hat{K}}_N p_d] \tag{8.3.3}$$

$$\ddot{p}_d = \hat{K}_N^{-1}[\ddot{\hat{g}}_{N1}(q, \dot{q}, \ddot{q}_r, s_{\Delta 1}) - D_1 \ddot{s}_1 - 2\dot{\hat{K}}_N \dot{p}_d - \ddot{\hat{K}}_N p_d] \tag{8.3.4}$$

with eqs. (7.2.35)–(7.2.37) as the learning algorithm.

To satisfy the persistent excitation condition, we either use the random trajectory during the training stage, or use $\ddot{q}_d$ to replace $\ddot{q}$ and $q_d^{(3)}$ to replace $q^{(3)}$ in eqs. (8.3.3) and (8.3.4) which eliminate the PE requirement.

Eqs. (8.3.1) and (8.3.2) can be enhanced into the following:

$$u = -D_2 s_2 - A_2 sgn(s_2) + \hat{g}_{N2}(q, p, \dot{p}, \ddot{p}_r) \tag{8.3.5}$$

$$p_d = \hat{K}_N^{-1}[-D_1 s_1 - A_1 sgn(s_1) + \hat{g}_{N1}(q, \dot{q}, \ddot{q}_r, s_{\Delta 1})] \tag{8.3.6}$$

where A_1 and A_2 are positive definite diagonal matrices; $sgn(.)$ means applying the sign function to every entry of vectors s_1 and s_2. This modification makes the neurocontrol algorithm robust to the neural network modeling errors.

Similarly, Gaussian RBF networks are used to realize the neurocontroller. The tested desired trajectories are (1) a straight line, (2) a circle and (3) a rectangle, as defined in section 7.2.2.

The real–time control code was programmed in C language. The code was compiled to obtain the object modules. These modules were then downloaded to the DSP board. A brief description of the experimental procedure can be found in Joseph (1992) or Massoud (1993).

8.4 EXPERIMENTAL RESULTS

The following different experimental strategies were tested by applying the direct adaptive neurocontrol law summarized in section 8.3: (a) using pure neurocontroller directly to control the robot without any off–line training; (b) using pure neurocontroller with off–line training to determine good initial values of the weights; and (c) using the robust neurocontroller with off–line trained initial weights and structure information about the motor dynamics.

For strategy (a), it was difficult to complete the learning processes of the neurocontroller due to the poor initial control responses and the motor servo–system overloading. The link 1 angle was confined within (–80°, 80°). The link 2 angle was confined within (–90°, 90°). The robot cannot work if the angles are out of the range. The frequent source of trouble was the motor 1 servo–system overloading. Sometimes, the experiment just could not be started due to the high initial angular velocity response.

For strategy (b), the neurocontroller could complete the learning processes. However, there was a computation time and control precision problem. In the simulation results presented in section 7.2.2, 2000 RBF neurons were used to guarantee high approximation precision. If the same number of neurons are used in the real–time implementation, then about 0.2 *sec* of computation time is required in each updating step. This means that the sampling period has to be larger than 0.2 *sec*. Since the direct adaptive neurocontrol law (8.3.1)–(8.3.6) with learning algorithm (7.2.35) – (7.2.37) is presented in the continuous time domain, a sampling period that is too large greatly reduces the control precision, and may even make the system unstable. If fewer neurons are used, the computation time can be decreased, but the neural network approximation precision, and thus the control precision, is lowered.

This observation shows that for real–time applications, large scale neural networks have to be implemented on neural net hardware to utilize their parallelism fully for fast computing. Otherwise, the so–called advantage of massively parallel processing cannot be obtained using conventional digital computers.

Strategy (c) was conducted with success. Since the structure of the motor dynamics was assumed to be known, $\hat{g}_{N2}(q, p, \dot{p}, \ddot{p}_r)$ in eq. (8.3.5) assumed a very simple form. The structure information about $g_1(q, \dot{q}, \ddot{q}_r, s_{\Delta 1})$ defined in eq. (7.2.10) was further used to simplify the neural network design for $g_1(q, \dot{q}, \ddot{q}_r, s_{\Delta 1})$. Finally, 200 neurons were used to approximate $g_1(q, \dot{q}, \ddot{q}_r, s_{\Delta 1})$ with the computation time being about 5.6 ms per step. The sampling period was set to 6 ms.

The neurocontroller was first trained using the nominal model of the experimental robot given in section 7.2.2. Then the off–line learned weights were used in the robust neurocontrol algorithm (8.3.3) – (8.3.6) to control the robot. Three different trajectories were tested: (1) a straight line, (2) a circle and (3) a rectangle. All the trajectories were repetitive so that the experiment can continue for any time period. One block in " $\subset$ " shape (with its mass about 1 kg) could be attached anywhere on the second link (with its mass about 1.8 kg) of the robot. It was used to change the dynamics of the experimental robot and test the robustness of the neurocontrol law.

The following experimental results are based on strategy (c). The two–layer Gaussian RBF networks are used to represent the neurocontroller. There are 200 neurons in the hidden layer.

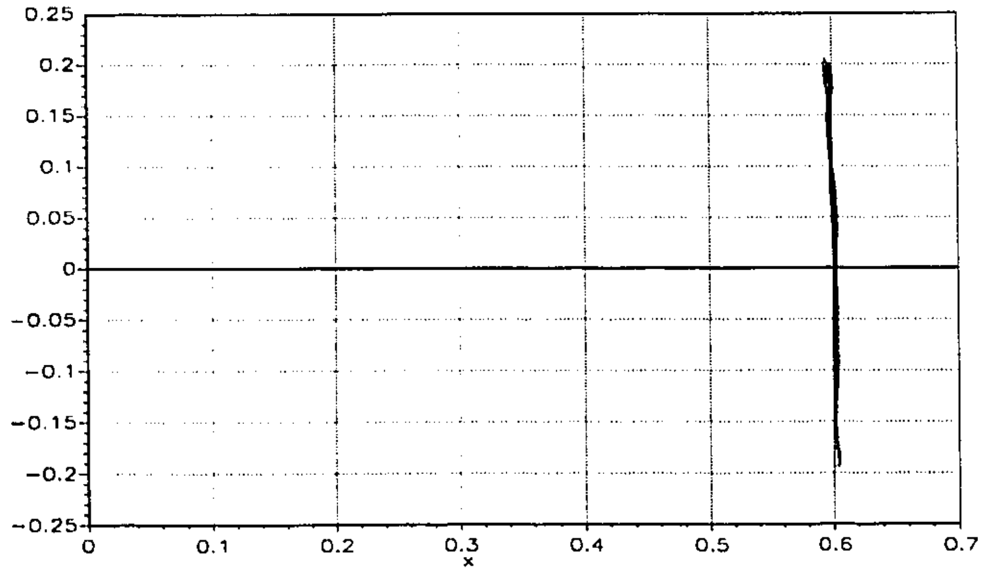


Fig. 8.5 The actual tip trajectory of the experimental robot (line)

Experiment 1: A straight line trajectory tracking control was conducted. Fig. 8.5 shows the actual tip trajectory of the experimental robot under the robust direct adaptive neurocontrol law. Due to the off–line training and the robustness, the control performance is consistent, in spite of the continuing learning. The maximum tracking error of the desired straight line is 5 mm or 0.0061 rad in joint angles, and the average tracking error is 0.78 mm. The average tip velocity is 400 mm/s, which was faster than those in other reported experiments.

When the ⊂ block was attached to the second link at any position, the same control precision was obtained (fig. 8.6). The neurocontrol system was always stable. This shows

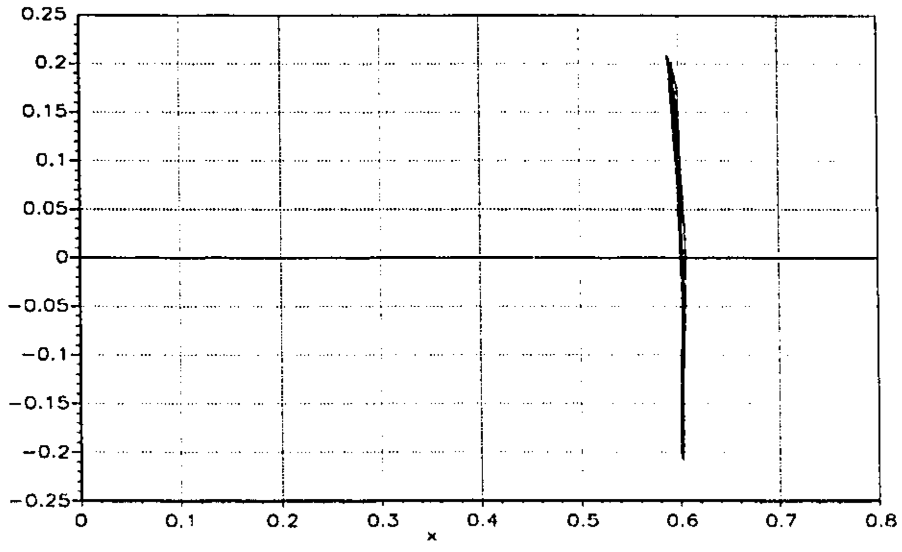


Fig. 8.6 The actual tip trajectory of the experimental robot with ⊂ block (line)

that the neurocontrol law is robust to dynamic or load changes of the robot. The load change is dramatic in this experiment, since the experimental robot is light and the mass of the ⊂ shape block (about 1 kg) is approximately half of that of the second link (about 1.8 kg) of the robot. The performance of robots which use conventional controllers deteriorates greatly when subjected to such a load change.

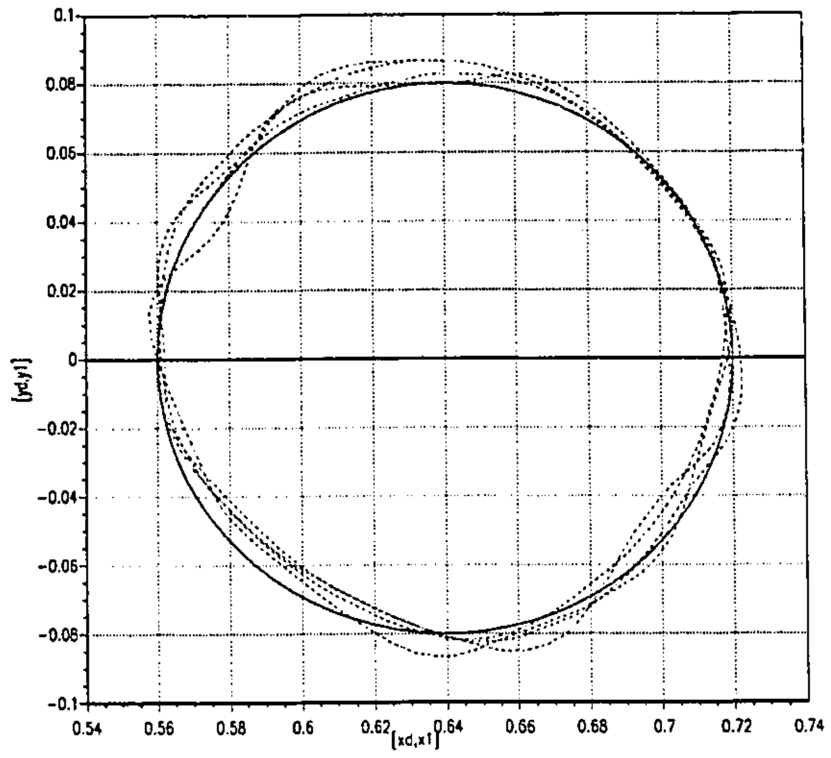


Fig. 8.7 The actual tip trajectory of the experimental robot (circle)

Experiment 2: A circle trajectory tracking control was conducted. Fig. 8.7 shows the actual tip trajectory of the experimental robot under the robust direct adaptive neurocontrol law. The maximum tracking error of the desired circle is 15 mm or 0.018 rad in joint angle, and the average tracking error is 1.34 mm or 0.016 rad in joint angle. The average tip velocity is 252 mm/s.

When the ⊏ block was attached to the second link at any position, the same control precision was obtained (see fig. 8.8) and the actual tip trajectory looks similar to fig. 8.7. The initial tracking error was larger compared with fig. 8.7 due to the attachment of the ⊏ block and then converged. The neurocontrol system was always stable. This shows again that the neurocontrol law is robust to dynamic or load changes of the robot.

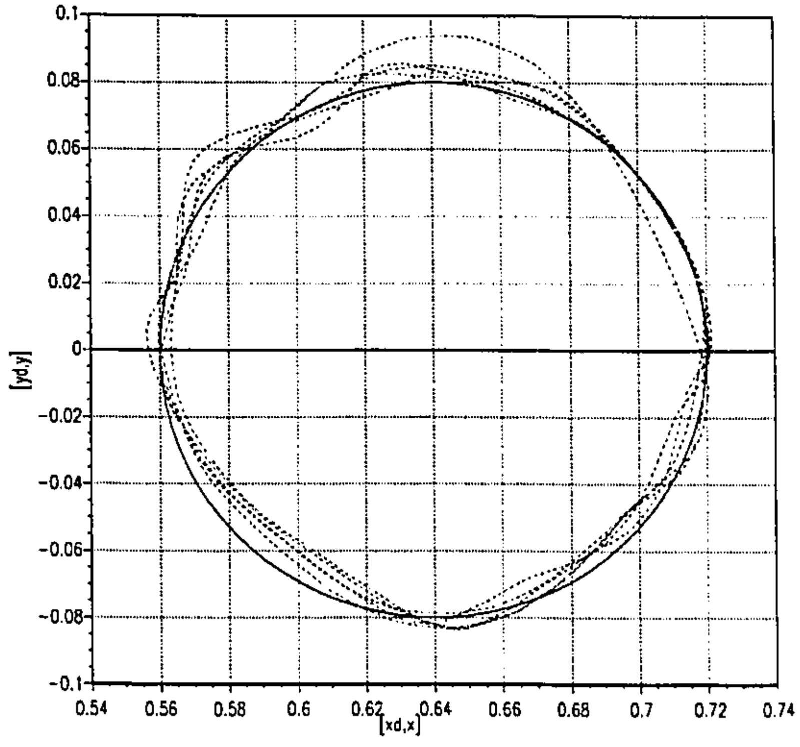


Fig. 8.8 The actual tip trajectory of the experimental robot with ⊂ block (circle)

Experiment 3: A rectangle trajectory tracking control was conducted. Fig. 8.9 shows the actual tip trajectory of the experimental robot under the robust direct adaptive neurocontrol law. The maximum tracking error of the desired rectangle is 15 mm or 0.021 rad in joint angle, and the average tracking error is 1.12 mm. The average tip velocity is 200 mm/s. When the ⊂ block was attached to the second link at any position, the same control precision was obtained (see fig. 8.10) and the actual tip trajectory looks similar to fig. 8.9. The initial tracking error was larger compared with fig. 8.9 due to the attachment of the ⊂ block and then converged. The robustness of the neurocontrol law to dynamic or load changes of the robot is demonstrated once more.

The control precision and average tip moving speed of all the three experimental results are summarized in table 8.4.1.

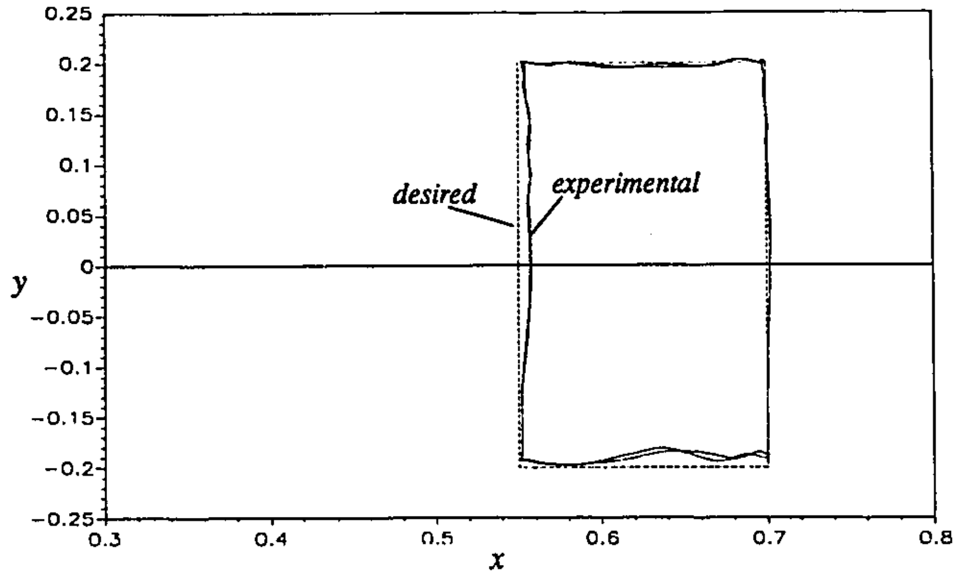


Fig. 8.9 The actual tip trajectory of the experimental robot (rectangle)

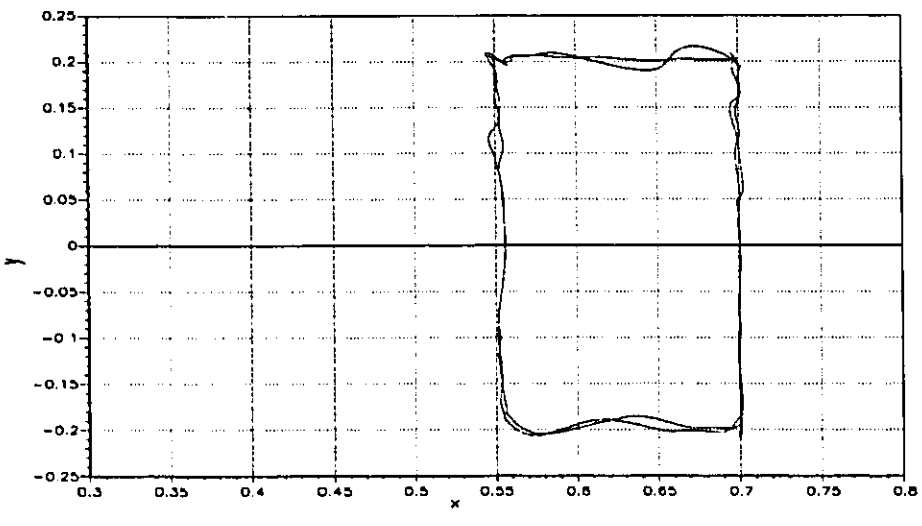


Fig. 8.10 The actual tip trajectory of the experimental robot with ⊂ block (rectangle)

The experimental results are compared with the simulation results presented in Section 7.2, and summarized in Table 8.4.2. It can be seen that the trajectory tracking precision of the experimental results is lower than that of the simulation results. The major reason is because the number of neurons is greatly limited by the real–time computation requirement

Table 8.4.1 Summary of the experimental results

Trajectory	Max. Position Error	Aver. tip speed
———	5 mm / 0.35 deg	400 mm/s
○	15 mm / 1.05 deg	252 mm/s
▭	15 mm / 1.22 deg	200 mm/s

Table 8.4.2 Comparison of the experimental and simulation results

Trajectory	Maximal Tracking Error	
	Experiment	Simulation
———	5 mm	1.5 mm
○	15 mm	3.0 mm
▭	15 mm	2.0 mm

for the actual experiments. The signal measurement noise and larger sampling period also lower the tracking precision.

The experimental results presented here are compared with other available results in the following. Since experimental results for trajectory control of flexible joint robots identical to our setup was not found, similar work is compared here. Carusone, Buchan and D'Eleuterio (1993) presented their experimental results for the square and circle tracking control of a rigid–joint flexible–link robot. Linear control law was developed for the trajectory control, therefore the robot could only move slowly. The experimental results are summarized in Table 8.4.3. It can be seen that the results obtained in this research are better regarding precision with faster moving speed. The control of a rigid–joint flexible–link robot is more difficult than that of a rigid–link flexible–joint robot (Spong and Vidyasagar, 1989).

Table 8.4.3 Experimental results from Carusone, et al.

Trajectory	Max. Tracking Error	Max. tip speed	Tracking Error at Corner/start
□	39.5 mm	125 mm/s	69.7 mm
○	35.3 mm	157 mm/s	51.8 mm

The next comparison is based on the experimental results presented in Tarn et al. (1991). A PUMA 560 was used and a circle trajectory was tested. The PUMA 560 can be regarded as a rigid–link rigid–joint robot, at low working speed. A feedback linearization method was used to design the trajectory control law. The experimental results are summarized in Table 8.4.4. They compared the results obtained by considering the motor dynamics

Table 8.4.4 Experimental results from Tarn, et al.

○	Max. Pos. Error	Max. tip speed
2nd order model	30 mm	170 mm/s
3rd order model	7 mm	170 mm/s

in the controller design (3rd order model) with those obtained by neglecting the motor dynamics in the controller design (2nd order model). The results in this thesis correspond to the case of neglecting the motor dynamics in the controller design (2nd order model). It can be seen that this thesis results are better than those obtained by Tarn et al. using the 2nd order model and worse than those obtained using 3rd order model, and faster in moving speed. The control of a rigid–joint rigid–link robot is easier than that of a rigid link flexible joint robot.

The third comparison is based on the experimental results presented in An et al. (1989). The MIT serial link direct drive arm, which can be regarded as a rigid–link rigid–joint robot, was used at a low working speed. The desired trajectory was a smooth position change. Computed torque control law was applied to control the robot. The experimental results are summarized in Table 8.4.5. An et al. investigated the effect of digital computing

Table 8.4.5 Experimental results from An, et al.

	Max. Angle Errors	Ours
Analog servo	0.33, 0.64, 1.4 deg.	N/A
Digital servo	2.20, 2.00, 4.0 deg.	0.35—1.22deg

on the control precision, using both an analog servo controller and a digital servo controller. The experimental results in this thesis are comparable with theirs.

The last comparison is based on the experiments conducted by S. Yu of department of mechanical engineering at McMaster University in 1994. A flexible–joint flexible–link experimental robot was built and used in the experiments. A model reference adaptive control law based on the linear model of the robot was tested for end–effector position control, which is much easier than trajectory control. Trajectory control was not achieved in his work. The results of this thesis were applied to both trajectory and position control with success.

In summary, the control precision of the experiments is acceptable and the motion of the robot is faster compared with other reported experiments. Since the sampling period is larger and the number of neurons used is smaller, the control precision in the experimental test is lower than that in the numerical simulation shown in section 7.2.2. This problem can be solved by using neural net hardware. The robust modification expressed by eqs. (8.3.5) and (8.3.6) can ensure good transient responses and require fewer neurons.

The experimental results show that incorporating a priori information about the system dynamics can simplify neurocontroller design. They also show that it is possible to make the robots light, move fast and accurately, and carry heavier loads compared to their own weight.

8.5 DISCUSSIONS

Through experimental testing, it was found that the neurocontrol approach can be used to control a system without detailed modeling and parameter identification of the system dynamics. Compared with the exact model–based control (EMC) schemes, the adaptive neurocontrol (ANC) schemes require the least a priori information about the system dynamics. If an accurate model of a system were available, EMC schemes would give the best performance. ANC schemes are able to improve their performance and approach the best results. If there are uncertainties and/or time–varying factors (which are difficult to model), EMC schemes usually deteriorate.

There is always a lower bound of control precision which can be achieved using model–based control schemes due to the existence of uncertainty as analyzed before. For neurocontrol schemes, this bound can be made very small through proper neural network structure design. The generality of neurocontrol schemes is acquired at the cost of computation complexity. Using neural network hardware can resolve this computation complexity problem.

If the model reference adaptive neurocontrol scheme presented in section 7.3 is applied experimentally, better results can be expected, since the persistent excitation signals are not required. As stated in Chapter 3, the generalization is not guaranteed outside the learning space. Therefore, the input space of a neurocontroller should cover the whole working space of the robot.

CHAPTER 9

CONCLUSIONS

In chapter 2, we reviewed the state–of–the–art of neural networks in control and its existing problems. Throughout this thesis, new contributions have been made to the neurocontrol field. These contributions are summarized in section 9.1. Section 9.2 presents discussions of the achievements. Some important issues which are still to be solved in future research are outlined in section 9.3.

9.1 SUMMARY OF CONTRIBUTIONS

Almost all dynamic systems in reality are nonlinear. It is often difficult, if not impossible, to obtain accurate dynamic system models. Flexible–joint robot systems belong to highly coupled nonlinear dynamic systems, and using linear control approaches usually produce poor control performance. There are few nonlinear control design approaches available to date, most of which can only be applied to feedback linearizable nonlinear systems. The neurocontrol approach opens a new area that may present general solutions to nonlinear control systems. Based on the preliminary level of development of the neurocontrol approach, the following research was conducted:

(a) a search for better learning algorithms and neural network architectures suitable for nonlinear system modeling and neurocontroller formulation;

(b) the construction of new adaptive neurocontrol schemes with better learning performance and guaranteed system stability;

(c) the derivation of adaptive neurocontrol schemes for general flexible–joint robots to achieve better performance under large modeling errors.

Three major contributions in the neural network area, four major contributions in the neurocontrol area and three major contributions in the neurocontrol of flexible–joint robots were achieved in this thesis. These are:

In the neural network area:

(1) A localized neural network concept was proposed and tested. The localization can speed up the learning processes and reduce computation.

(2) A new multistep localized adaptive learning algorithm for RBF networks with about ten times faster learning convergence and reduced computation requirement was derived and verified.

(3) New neural network models of nonlinear systems with different representations, with and without measurement noises, were formulated and tested by numerical simulation.

In the neurocontrol area:

(4) Theoretical issues related to the popular backprop–based indirect and direct adaptive neurocontrol schemes were clarified for the first time from the viewpoint of the control theory.

(5) A new backprop–based indirect adaptive neurocontrol algorithm with faster convergence was derived, as a result of using localized polynomial networks. Also, a global

asymptotically stable indirect adaptive neurocontrol scheme was proposed and tested by numerical simulation, using localized linear networks.

(6) New direct adaptive neurocontrol schemes with faster learning convergence were proposed. They are based on input–output Jacobian matrix estimation and the optimization algorithms without using derivative information.

(7) New self–tuning neurocontrol schemes were proposed, mathematically proved and tested by numerical simulation. They extend the adaptive control theory for linear systems to nonlinear systems.

(8) The variable index control theory was introduced, mathematically proved and tested by numerical simulation.

(9) A stable robust adaptive neurocontrol scheme based on the variable index control theory was derived. Its robustness and stability were mathematically proved. It can be applied to general nonlinear systems, which are not necessarily feedback–linearizable.

In the neurocontrol of flexible–joint robots:

(10) A new direct adaptive neurocontrol scheme with proven stability for general flexible joint robots was derived, theoretically proved and tested by both numerical simulation and experiments.

(11) A model reference adaptive neurocontrol scheme was derived for general flexible joint robots. The global stability was proved theoretically.

(12) A robust and stable adaptive neurocontrol scheme for general flexible joint robots was derived based on the variable index control theory. The system global stability was proved theoretically.

Based on these contributions, the following conclusions were reached:

1. The localized neural networks can be applied to neurocontrol with faster learning convergence and less computation burden. The benefit is tremendous for complex high order nonlinear systems, especially for real–time control applications.
2. The adaptive extended Kalman filtering learning algorithm with UD factorization provides good learning performance for many neural networks.
3. The four new neural network models of nonlinear systems, introduced in this thesis, are useful for neurocontroller design.
4. The input–output determination, for stabilizability, of the backprop–based adaptive neurocontrollers was solved.
5. The new backprop–based adaptive neurocontrol schemes are about ten times faster in learning convergence than the existing ones published previously.
6. Compared with the backprop–based adaptive neurocontrol schemes, the stability–based adaptive neurocontrol schemes perform much better in learning and stability. For the latter, it is easy to set the initial values of the weights.
7. The self–tuning neurocontrol schemes extend the stability–based neurocontrol schemes to discrete time and/or sampled–data nonlinear systems. This makes fast and globally convergent learning possible.
8. The variable index control approach and the corresponding stable and robust adaptive neurocontrol scheme can be applied to general continuous–time nonlinear systems. These nonlinear systems are not necessarily feedback linearizable and the nonlinearities are not necessarily differentiable. For the adaptive neurocontrol scheme, linear parameterization of unknown parameters, the common assumption in conventional adaptive control of nonlinear systems, is not required. These results are significant in the control field.

9. The self–tuning neurocontrol schemes and the other stability–based adaptive neurocontrol schemes are significant, because they show close resemblance to the maturing linear adaptive control theory. The dynamic behavior differences between the linear systems and nonlinear systems were also clearly revealed in this research.

10. All three adaptive neurocontrol schemes for general flexible joint robots are innovative. They can solve the control problem of general flexible joint robots with large modeling errors. The proposed methodologies are also applicable and useful to non–neurocontrol approaches.

11. Incorporating a priori information about the controlled systems can simplify the neural network design and reduce on–line computation.

12. The experimental tests show that the neurocontrol schemes can produce high control precision. The neural networks can be used to compensate the modeling errors.

9.2 DISCUSSIONS

9.2.1 Utilizing a priori information

In most existing neurocontrol schemes, it is assumed that the models of the controlled dynamic systems are unknown. This assumption shows the power of the neurocontrol approach. However, in practice, there is always some a priori information about the controlled dynamic systems, especially for those which have been studied for many years. For example, simplified dynamic models and kinematics of robotic systems have been investigated for more than 20 years. The a priori information includes the dynamic order, the structure, the nominal model, and even existing conventional controllers of a controlled system. The thesis shows that all of these can be incorporated into the neurocontroller design (sections 4.2, 4.3, 6.4, 7.2 and 7.3). The more we know about a controlled system, the better the neurocontroller with the information incorporated will be.

There may be problems in neurocontrol approach with pure neural network models. First, the structure of a neural network for a given control system is difficult to determine uniquely. Secondly, there are many weights to be tuned at the learning stage, which, if the neural networks start learning from scratch, makes the initial dynamic responses unpredictable and with large control errors. Thirdly, neurocontrol algorithms require intensive computation for complex multivariable control systems. These problems can be partially solved by incorporating a priori information about the controlled systems into the neurocontroller design, as shown in the thesis. Also, it helps to use localized neural networks to reduce the number of weights to be updated, and to pre–train the neurocontrollers off–line. Interesting applications of neurocontrol approach are the hybrid neural and conventional control, or using neural networks as internal models of the dynamic systems.

9.2.2 Neurocontroller implementation

At present, the generality of neurocontrollers is achieved at the cost of computing complexity. Massive neurons performing simple and uniform parallel computation were claimed to be an advantage of neural networks. If the neural networks are implemented on conventional digital computers, the parallelism cannot be utilized. It is difficult to achieve fast computation of large–scale neural networks using a sequential processing computer. For real–time control applications, the number of neurons that can be used is greatly limited by the sampling frequency requirement and the computer speed. This was observed in the experimental test.

Historically, any new control theory is accompanied with new hardware innovation. As we know, classical controllers were implemented by analog circuits. Adaptive controllers are realized by digital processors. Therefore, it can be predicted that the real power of neurocontrol theory will be fully realized when neural network hardware comes into use.

9.2.3 Neurocontroller structure determination

For single–input single–output control problems, most neurocontrol schemes work well. For large scale nonlinear systems, proper structure design of neurocontrollers is crucial to successful neurocontrol.

There are two structures to be determined in a neurocontrol system: the overall neurocontrol system structure and the neurocontroller structure. The former is completely determined if one neurocontrol scheme is adopted. It also distinguishes one neurocontrol scheme from another and determines the system stability. The latter determines the modeling precision and control precision, and is related to the structure design of neural networks.

In general, it is difficult to determine an optimal neural network structure for a given control problem, especially in real time. However, it is reasonable in practice to determine a feasible neural network structure, through trial and error, for a given problem that is not very complex. If the number of used neurons is less than the optimal one, an approximation error may exist. If the number of used neurons is larger than the optimal one, a generalization error may also exist. In reality, if the number of used neurons is not far away from the optimal one, acceptable performance in engineering can often be obtained.

The structure determination algorithms proposed in sections 3.2 and 3.3 are useful in finding a suboptimal number for a given problem in real–time, due to their simplicity. Locality is an inherent feature of biological neural networks, and can be applied to artificial neural networks to simplify the structure design.

For complex control problems with a high dimension of inputs to the neurocontrollers, the number of neurons required for universal approximation ability increases dramatically. This is the so–called curse of dimensionality. However, if we drop the requirement of universal approximation ability, much fewer neurons are required for many practical problems, especially when some a priori information about the smoothness of the approximated nonlinearity and the working region are known.

9.2.4 Neural network model vs. first–principle model

As indicated in section 3.4, neural network models for nonlinear systems are not structured first–principle models. The first–principle model of a nonlinear system is the best for either analysis, or prediction and control. Its structure and parameters are unique. However, first–principle models are difficult to obtain for many problems.

Neural network models represent an unstructured approach. The models are usually over–parameterized and the neurons and weights are redundant, which gives the neural network models some universal approximation ability. At the same time, the over–parameterization and the redundancy cause problems of over–generalization and multiple solutions.

To avoid such problems, the training data have to be representative and persistent excitation (section 3.4). Similarly, incorporating a priori information about a given problem will reduce the over–parameterization and the redundancy.

Although the functional structures of artificial neural networks resembles some biological neural networks, the scales, connectivity and learning mechanisms are quite different from each other at present. More research work with collaboration among different disciplines has to be done in the future. Sooner or later, people will not regard neural networks as a magic, but rather as efficient AI and mathematical tools that can be applied to solve many cognitions, complex mappings and nonlinear problems in many fields.

9.2.5 Neurocontrol and adaptive control

As indicated in section 7.4, the usage of neurocontrollers can be divided into a learning stage and a working stage. The control performance of neurocontrollers are usually unacceptable during the learning stage, whether on–line or off–line. The same claim applies to conventional adaptive control, too. This is because both of them are based on the learn–

by–mistake (error) principle. If there were no control errors, the neurocontrollers and adaptive controllers could not learn or adapt.

This analysis shows the importance of classification of the learning stage and the working stage. The learning stage belongs to the continuing design of the controllers. Unawareness of this fact may lead to problems in practical systems.

Another issue is about the necessity of the persistent excitation (PE) requirement. For both neurocontrol and adaptive control, some schemes require PE to guarantee system stability, while some schemes do not. However, if the PE requirement is not satisfied, the parameters will certainly not converge to their optimal values in both cases. This raises a question: if the parameters have not converged, what is the effect of continuing learning?

As discussed previously, continuing learning or adaptation occurs only if there are control and/or modeling errors. If the parameters have not converged, these parameters may work only for some desired control tasks. Therefore, when there is a control task change, the current parameters may not fit and control and/or modeling errors may appear. Continuing learning or adaptation switches the controlled system from a working stage to a new learning stage, and deteriorated control performance may appear during this period.

In the thesis, some robust control schemes are combined with the neurocontrol schemes to alleviate this problem and to achieve more consistent control performance.

9.3 FUTURE WORK

Most of the results in the thesis apply to deterministic dynamic systems. Further research should consider stochastic dynamic systems, since there is always measurement noise in the feedback signals.

For the backprop–based neurocontrol schemes and the self–tuning neurocontrol schemes proposed in the thesis, the feedback signals are the measurable outputs of the controlled systems. For the rest of stability–based neurocontrol schemes presented in the thesis, the feedback signals are the system states. Since system states may not be available in many practical systems, constructing neuro–observers is an interesting topic for further research.

Most stability–based neurocontrol schemes are given in continuous time domain. They have to be implemented on digital computers. Therefore, the sampling effects of these control algorithms have to be further investigated. As shown in the experiments, larger sampling periods lead to control precision reduction.

Similar to linear adaptive control, the issues related to possible loss of stabilizability and controllability of adaptive neurocontrol schemes during the learning stage are yet to be solved. It is desirable to extend the thesis results to the case when dynamic order uncertainty exists in system dynamics.

In the thesis, most of the carried out work relates to the trajectory control of robots. Further extensions to force control and constrained motion control is another interesting topic for research.

Due to the powerful mapping ability of neural networks, the proposed neurocontrol schemes for robots can be enhanced with neural networks performing computer vision–to–task space position and orientation mapping, desired trajectory formulation, and collision avoidance.

BIBLIOGRAPHY

[1] Al–Ashoor, R.A., Patel, R.V. and Khorasani, K. (1993), "Trajectory following robust adaptive control of flexible–joint manipulators", *IEEE Trans. Syst., Man, and Cybern.*, Vol. 23, pp. 589–602.

[2] Albus, J.S. (1979), "Mechanisms of planning and problem solving in the brain", *Mathematical Biosciences*, Vol. 45, pp. 247–293.

[3] Albus, J. (1975), "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)", *J. Dynamic Systems, Measurement, and Control*, Vol. 97, Sept., pp. 220–227.

[4] An, C.H., Atkeson, C.G., Griffiths, J.D. and Hollerbach, J.M. (1989), "Experimental evaluation of feedforward and computed torque control", *IEEE Trans. on Robotics and Automation*, Vol. 5, No. 3, pp. 368–373.

[5] Anthony, M. and Biggs, N.L. (1992), *Computational Learning Theory: An Introduction*, Cambridge University Press, Cambridge, UK.

[6] Antsaklis, P.J. (1992), "Neural networks for control systems", *IEEE Control Systems Magazine*, Vol. 12, April, pp. 8–10.

[7] Asada, H. and Liu, S. (1991), "Transfer of human skills to neural net robot controller", *Proc. IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 2442–2448.

[8] Asada, H. and Slotine, J.–J.E. (1986), *Robot Analysis and Control*, John Wiley and Sons, New York.

[9] Åström, K.J. (1991), "Where is the intelligence in intelligent control", *IEEE Control Systems Magazine*, Vol. 11, No.1, pp. 37–39.

[10] Åström, K.J. and Wittenmark, B. (1989), *Adaptive Control*, Addison–Wesley.

[11] Bar–Kana, I. and Guez, A. (1990), "Neuromorphic computing architecture for adaptive control", *Proc. IJCNN'90*, Washington, D.C., pp. II: 323–326.

[12] Bassi, D.F. and Bekey, G.A. (1989), "High precision position control by cartesian trajectory feedback and connectionist inverse dynamics feedforward", *Proc. IJCNN'89*, Washington, D.C., pp. II: 325–331.

[13] Bavarian, B. (1988), "Introduction to neural networks for intelligent control", *IEEE Control Systems Magazine*, Vol. 8, No. 3, Apr., pp. 3–7.

[14] Bortoff, S.A. and Spong, M.W. (1987), "Feedback linearization of flexible joint manipulators", *Proc. 1987 IEEE Conf. on Decision and Control*, pp. 1357–1362.

[15] Braess, D. (1986), *Nonlinear Approximation Theory*, Springer–Verlag.

[16] Brent, R. P. (1973), *Algorithm for Minimization without Derivatives*, Prentice–Hall, Inc.

[17] Canudas De Wit, C. and Lys, O. (1988), "Robust control and parameter estimation of robots with flexible joints", *Proc. 1988 IEEE Int. Conf. on Robotics and Automation*, Philadelphia, PA, pp. 324–329.

[18] Carusone, J., Buchan, K.S. and D'Eleuterio, G.M.T. (1993), "Experiments in end–effector tracking control for structurally flexible space manipulators", *IEEE Trans. on Robotics and Automation*, Vol. 9, No. 5, pp. 553–560.

[19] Chen, F.–C. (1991), "A dead–zone approach in nonlinear adaptive control using neural networks", *Proc. IEEE Conf. on Decision and Control*, Brighton, UK, pp. 156–161.

[20] Chen, S. Billings, S.A. and Grant, P.M. (1992), "Recursive hybrid algorithm for nonlinear system identification using radial basis function networks", *Int. J. of Control*, Vol. 55, No. 5, pp. 1051–1070.

[21] Cotter, N.E. (1990), "The Stone–Weierstrass theorem and its application to neural networks", *IEEE Trans. on Neural Networks*, Vol. 1, No. 4, pp. 290–295.

[22] Cui, X. and Shin, K.G. (1991), "Design of an industrial process controller using neural networks", *Proc. ACC'90*, pp. 2466–2471.

[23] Cybenko, G. (1989), "Approximation by superpositions of a sigmoidal function", *Math. Control, Signals, and Systems*, Vol. 2, pp. 303–314.

[24] Duffy, J. (1990), "The fallacy of modern hybrid control theory", *Int. J. Robotics Systems*, Vol. 7, No. 2, pp. 139–144.

[25] Eckmiller, R. Beckmann, J., Werntges, H. and Lades, M. (1989), "Neural kinematics net for a redundant robot arm", *Proc. IJCNN'89*, Washington, D.C., pp. II: 333–339.

[26] ElMaraghy, H.A. and Liang, F. (1993), "Integrated mechanical and control design for well–designed flexible joint robots", *Proc. 1993 DND Workshop on Advanced Technologies in Knowledge Based Systems and Robotics*, Ottawa, Nov. 1993.

[27] Fadali, M.S., Aguirre, F.J. and Egbert, D.D. and Tacker, E.C. (1990), "Minimum–time control of robotic manipulators using a back propagation neural network", Proc. ACC, San Diego, CA, pp. 2997–3000.

[28] Fukuta, T., Shibata, T., Kosuga, K., Arai, F., Tokita, M. and Mitsuoka, T. (1991), "Neuromorphic sensing and control – application to position, force and impact control for robotic manipulators", *Proc. IEEE CDC*, Brighton, UK, pp. 162–167.

[29] Fukuta, T. and Shibata, T. (1991), "Adaptation and learning for hierarchical

intelligent control", Proc. IJCNN'91, Baltimore, MD, pp. I: 269–274.

[30] Ficola, A., Marino, R. and Nicosia, S. (1983), "A singular perturbation approach to the control of elastic joints", *Proc. 21st Ann. Allerton Conf. on Communications, Control and Computing*, Monticello, IL, pp. 220–225.

[31] Ghorbel, F., Spong, M. and Hung, J. (1989), "Adaptive control of flexible joint manipulators", *Proc. 1989 IEEE Int. Conf. on Robotics and Automation*, Phoenix, AZ, pp. 1188–1193.

[32] Gu, Y.–L. (1990), "On nonlinear system invertibility and learning approaches by neural networks", *Proc. 1990 American Control Conference*, San Diego, CA, pp. 3013–3018.

[33] Guez, A., and Ahmad, Z. (1989), "Accelerated convergence in the inverse kinematics via multilayer feedforward networks", *Proc. IJCNN'89, Washington, D.C.*, pp. II: 341–344.

[34] Guez, A., Eilbert, J. and Kam, M. (1988), "Neural network architecture for control", *IEEE Control Systems Magazine*, Vol. 8, No.3, pp. 22–25.

[35] Guo, J. and Cherkassky, V. (1989), "A solution to the inverse kinematic problem in robotics using neural network processing", *Proc. IJCNN'89, Washington, D.C.*, pp. II: 299–304.

[36] Han, Y. (1992), *Adaptive Tracking Control Of Feedback Linearizable Nonlinear Systems*, Ph.D thesis, McMaster University, Hamilton, ON.

[37] Handelman, D.A., Lane, S.H. and Gelfand, J.J. (1989), "Integration of knowledge–based system and neural network techniques for autonomous learning machines", *Proc. IJCNN'89*, Washington, D.C., pp. I: 683–688.

[38] Haykin, S. (1994), *Neural Networks, A Comprehensive Foundation*, Macmillan

College Publishing Company, Inc.

[39] Hebb, D.O. (1949), *The Organization of Behavior,* Wiley, New York.

[40] Herve, J.–Y., Sharma, R. and Cucka, P. (1991), "Toward robust vision–based control: hand/eye coordination without calibration", *Proc. IEEE Symp. Intell. Contr.*, Arlington, VA, pp. 457–462.

[41] Ho, T.T., Ho, H.T., Bialasiewicz, J.T. and Wall, E.T. (1991), "Stochastic neural direct adaptive control", *Proc. IEEE Int. Symp. Intell. Contr.,* Arlinton, VA, pp. 176–179.

[42] Hosogi, S. (1990), "Manipulator control using layered neural network model with self–organized mechanism", *Proc. IJCNN'90,* Washinton, D.C., pp. II: 217–220.

[43] Hunt, K.J., Sbarbaro, D., R. Zbikowski and Gawthrop, P.J. (1992), "Neural networks for control systems – a survey", *Automatica*, Vol. 28, No. 6, pp. 1083–1112.

[44] Hunt, K.J., Sbarbaro, D. (1991), "Neural networks for nonlinear internal model control", *IEE Proc. D,* Vol. 138, No. 5, pp. 431–438.

[45] Ioannou, P. and Sun, J. (1988), "Theory and design of robust direct and indirect adaptive control schemes", *Int. J. Control,* vol. 47, pp.775–813.

[46] Isidori, A. (1989), *Nonlinear control systems*, Springer–Verlag, New York.

[47] Ito, M. (1984), *The Cerebellum and Neural Control*, Raven Press, New York.

[48] Jankowski, K.P. and ElMaraghy, H.A. (1991), "Dynamic decoupling for hybrid control of rigid/ flexible joint robots interacting with the environment", *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 1226 – 1231.

[49] Joseph, S. (1992), "Experimental robot dSpace hardware/software", *FMRD 19–01–92*, McMaster University, Hamilton, ON.

[50] Jordan, M.I. and Jocobs, R.A. (1989), "Learning to control an unstable system with forward modeling", in *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann Publishers, Inc., pp. 29–39.

[51] Kanellakopoulos, I., Kokotovic', P.V. and Morse, A.S. (1991), "Systematic design of adaptive controllers for feedback linearizable systems", *IEEE Trans. Automatic Control.*, Vol. AC–36, pp. 1241–1253.

[52] Kasparian, V. and Batur, C. (1992), "Neural network structure for process control using direct and inverse process model", *Proc. 1992 American Contr. Conf.*, Chicago, IL, pp. 562–566.

[53] Keeler, J.D. (1991), "A dynamical system view of cerebellar function", in S. Forrest (ed.), *Emergent Computation*, MIT Press, Cambridge, MA, pp. 396–410.

[54] Khorasani, K. (1991), "Adaptive control of flexible joint robots", *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 2127–2134.

[55] Khorasani, K. and Spong, M.W. (1985), "Invariant manifolds and their application to robot manipulators with flexible joints", *Proc. 1985 IEEE Int. Conf. Robotics and Automation*, St. Louis, MO, 1985.

[56] Kodrtoff, Y. and Michalski, R. (1990), *Machine Learning: An Artificial Intelligence Approach*, Vol. 3, Morgan Kaufmann Publishers, Inc.

[57] Kosmatopoulos, E.B., Chassiakos, A.K. and Christodoulou, M.A. (1991), "Robot identification using dynamic neural networks", *Proc. IEEE CDC*, Brighton, UK, pp. 2934–2935.

[58] Kuperstein, M. and Rubinstein, S. (1989), "Implementation of an adaptive neural controller for sensor–motor coordination", *Proc. IJCNN'89*, Washington, D.C., pp. II: 305–316.

[59] Lane, S.H., Handelman, D.A. and Gelfand, J.J. (1990), "Can robot learning like people do?", Proc. SPIE, Vol. 1294, pp. 296–309.

[60] Leahy, Jr. M.B., Johnson, M.A. and Rogers, S.K. (1991), "Neural network payload estimation for adaptive robot control", *IEEE Trans. Neural Networks*, Vol. 2, No. 1, pp. 91–100.

[61] LeCun, Y. (1986), "Learning processes in an asymmetric threshold network", *Disordered Systems and Biological Organization*, E. Beinestock, Fogelman Souli, and G. Weisbuch (eds.), Springer, Berlin.

[62] Lee, D.M.A. and ElMaraghy, W.H. (1992), "A neural network solution for bipedal gait synthesis", *Proc. IJCNN'92*, Baltimore, Maryland, pp. II: 763–768.

[63] Leung, T.P., Zhou, Q.–J. and Pei, H.–L. (1992), "A robust neural network controller", *Proc. ACC'92*, Chicago, IL, pp. 983–986.

[64] Leshno, M., Lin, V.Ya., Pinkus, A. and Schocken, S. (1993), "Multilayer feedforward networks with a non–polynomial activation function can approximate any function", *Neural Networks*, Vol. 5, pp. 1248–1256.

[65] Levin, E.G. and Inbar, G.E. (1991), "Neural network architecture for adaptive system modeling and control", *Neural Networks*, Vol. 4, pp. 185–191.

[66] Levin, A.U. and Narendra, K. (1992), "Stabilization of nonlinear dynamical systems using neural networks", Proc. IJCNN'92, Baltimore, MD, Vol. I, pp. 275–280.

[67] Liang, F. and ElMaraghy, H.A. (1995), "Robust and global stabilization of nonlinear systems", to appear in *Proc. 1995 American Control Conference*, Seattle, WA.

[68] Liang, F. and ElMaraghy, H.A. (1994a), "Pseudo–sliding control and its

application to robots", accepted by *Trans. ASME Journal of Dynamic Systems, Measurement and Control.*

[69] Liang, F. and ElMaraghy, H.A. (1994b), "Model reference adaptive neurocontrol of flexible joint robots", *Proc. 1994 IEEE Int. Conf. on Neural Networks*, Orlando, FL, pp. 2765–2770.

[70] Liang, F. and ElMaraghy, H.A. (1994c), "Direct adaptive neurocontrol of flexible joint robots using localized pclynomial networks", *Proc. 1994 IEEE Int. Conf. on Robotics and Automation*, San Diego, CA, pp. 3186–3191.

[71] Liang, F. and ElMaraghy, H.A. (1994d), "Globally stable adaptive neurocontrol of flexible joint robots", *Proc. 1994 Int. Symp. Robotics and Manufacturing*, Maui, HI, Aug.

[72] Liang, F. and ElMaraghy, H.A. (1994e), "Self-tuning neurocontrol using localized polynomial networks with CLI cells", *Proc. 1994 American Control Conference*, Baltimore, MD, June, pp. 2148–2152.

[73] Liang, F. and ElMaraghy, H.A. (1993a), "Indirect adaptive neurocontrol using localized polynomial networks with CLI cells", *Proc. 1993 Int. Joint Conf. on Neural Networks*, Nagoya, Japan, Oct. 1993, pp. 657–660.

[74] Liang, F. and ElMaraghy, H.A. (1993b), "Localized polynomial neural networks with fast global learning convergence", *Proc. 1993 World Congress on Neural Networks*, Portland, OR, July 1993.

[75] Liang, F. and ElMaraghy, H.A. (1993c), "Localized pi–sigma networks with competitive lateral inhibitory cells", in G. Rzevski, J. Pastor and R.A. Adey (eds.), *Application of Artificial Intelligence in Engineering VIII*, Vol. 1: *Design, Methods and Techniques*, Computational Mechanics Publications and Elsevier Applied

Science, July, pp. 407–421.

[76] Liang, F. and ElMaraghy, H.A. (1993d), "Multistep localized adaptive learning RBF networks for nonlinear system identification", *Proc. of 1993 European Control Conference*, Groningen, the Netherlands, June, pp. 111–116.

[77] Liang, F. and ElMaraghy, H.A. (1993e), "Integrated mechanical and control design for well–designed flexible joint robots", *Proc. 1993 DND Workshop on Advanced Technologies in Knowledge Based Systems and Robotics*, Ottawa, ON, Nov. 1993.

[78] Liang, F. and ElMaraghy, H.A. (1993f), "Robust control of flexible joint robots", *Proc. 1993 Canadian Conf. on Electrical and Computer Engineering*, Vancouver, BC, Sept. 1993, pp. 127–130.

[79] Liang, F. and ElMaraghy, H.A. (1991a), "Sliding mode control without chattering and its application to robotic manipulators", *Flexible Manufacturing Centre Report FMRD 18–01–1991*, McMaster University, Hamilton, ON, May 1991.

[80] Liang, F. and ElMaraghy, H.A. (1991a), "Discrete sliding mode control of robots", *Flexible Manufacturing Centre Report FMRD 18–02–1991*, McMaster University, Hamilton, ON, May 1991.

[81] Liang, F., Zhang, L. and Li, C. (1988), "The Equivalent System Method for Evaluating Flight Qualities of Aircraft", *Acta Aeronautica et Astronautica Sinica*, Vol.9, No.11, Nov.

[82] Lin, L.–C. (1991), "State feedback H_∞ control of manipulators with flexible joints and links", *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 218–223.

[83] Lozano, R. and Brogliato, B. (1992), "Adaptive control of robot manipulators with flexible joints", *IEEE Trans. on Automatic Control*, Vol. AC–37, pp. 174–181.

[84] Ljung, L. and Soderstrom, T. (1983), *Theory and Practice of Recursive Identification*, MIT Press.

[85] Marino, R., Kanellakopoulos, I. and KoKotovic', P.V. (1989), "Adaptive tracking for feedback linearizable SISO systems", *Proc. 28th IEEE Conf. on Decision and Control*, Tampa, FL, pp. 1002–1007.

[86] Massoud, A. and ElMaraghy, H.A. (1993), "Design, dynamics, and identification of a flexible joint robot manipulator", *The IASTED Int. Conf. on Robotics and Manufacturing*, Oxford, UK, pp. 72–75.

[87] Massoud, A. (1994), "User's manual for the experimental robot real–time control system", *FMRD Report*, McMaster University, Hamilton, ON.

[88] Michie, D. and Chambers, R.A. (1968), "BOXES: An experiment in adaptive control", in E. Dale and D. Michie (eds.), *Machine Learning 2*, Oliver and Boyd, London, pp. 137–152.

[89] Miller III, W.T., Sutton, R.S. and Werbos, P.J. (1990), *Neural Networks for Control*, M.I.T. Press.

[90] Miyazaki, F., Kawamura, S., Matsumori, M. and Arimoto, S. (1986), "Learning control scheme for a class of robot systems with elasticity", *Proc. 25th CDC*, Athens, Greece, pp.74–79.

[91] Moody, J. and Darken, C. (1988), "Learning with localized receptive fields", in D. Touretzky, G. Hinton, and T. Sejnowski (eds.), *Proc. of the 1988 Connectionist Summer School*, Morgan Kaufmann, San Mateo, CA, pp. 133–143.

[92] Mrad, F.T. and Ahmad, S. (1992), "Adaptive control of flexible joint robots using position and velocity feedback", *Int. J. Control*, Vol. 55, No. 5, pp. 1255–1277.

[93] Mukhopadhyay, S. and Narendra, K.S. (1993), "Disturbance rejection in nonlinear

systems using neural networks", *IEEE Trans. on Neural Networks*, Vol. 4, No. 1, pp. 63–72.

[94] Narendra, K.S. and Annaswamy, A. (1989), *Stable Adaptive Systems*, Prentice Hall, Englewood Cliffs.

[95] Narendra, K.S. and Mukhopadhyay, S. (1992), "Intelligent control using neural networks", *IEEE Control Systems Magazine*, Vol. 12, No. 3, pp. 11–18.

[96] Narendra, K.S. and Parthasarathy, K. (1990), "Identification and control of dynamical systems using neural networks", *IEEE Trans. Neural Networks*, Vol. 1, No. 1, March, pp. 4–27.

[97] Narendra, K.S. and Parthasarathy, K. (1988), "Neural networks in dynamical systems", *IEEE Control Systems Magazine*, Vol. 8, No. 3, pp. 230–241.

[98] Osburn, P.V., Whitaker, H.P. and Kezer, A. (1961), "New developments in the design of model reference adaptive control systems", *Proc. of the IAS 29th Annual Meeting*, New York.

[99] Parker, D. (1985), "Learning logic", TR–87, Center for Computational Research in Economics and Management Science, MIT Press, Cambridge, MA.

[100] Pei, H.–L., Leung, T.P., and Zhou, Q.–J. (1992), "Backward construction — a decomposed learning method for robot force/position control", Proc. IJCNN'92, Baltimore, MD, Vol. I, pp. 293–298.

[101] Poggio, T. and Girosi, F. (1990), "Networks for approximation and learning", *Proc. of the IEEE*, Vol. 78, pp. 1481–1497.

[102] Polak, E. (1971), *Computational Methods in Optimization*, Academic Press, New York.

[103] Psaltis, D., Sideris, A. and Yamamura, A.A. (1987a), "A multilayered neural

network controller", *IEEE Int. Conf. on Neural Networks*, San Diego, CA, June, reprinted in *IEEE Control Systems Magazine*, Vol. 8, Apr. 1988, pp. 17–21.

[104] Psaltis, D., Sideris, A. and Yamamura, A.A. (1987b), "Neural controllers", *IEEE Int. Conf. on Neural Networks*, San Diego, CA, June.

[105] Qu, Z. (1993), "Input–output robust control of flexible joint robots", *Proc. 1993 IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, Vol. 3, pp. 1004–1010.

[106] Rabelo, L.C. and Avula, X.J.R. (1991), "Hierarchical neurocontroller architecture for intelligent robotic manipulation", *Proc. IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 2656–2661.

[107] Readman, M.C. and Belanger, P.R. (1992), "Stabilization of the fast modes of a flexible joint robot", Int. J. Rob. Res., Vol. 11, April.

[108] Rosenblatt, F. (1958), "The perceptron: a probabilistic model for information storage and organization in the bain", *Psych. Review*, Vol. 65, pp. 386–408.

[109] Rumelhart, D.E. and McClelland, J.L. and the PDP Research Group (1986), *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol. 1, MIT Press.

[110] Sanner, R.M. and Slotine, J.–J. E. (1992), "Gaussian networks for direct adaptive control", *IEEE Trans. on Neural Networks*, Vol. 3, No. 6, pp. 837–863.

[111] Saridis, G.N. (1983), "Intelligent robotic control", *IEEE Trans. on Automatic Control*, Vol. 28, No. 5, pp. 547–557.

[112] Sastry, S.S. and Isidori, A. (1989), "Adaptive control of linearizable systems", *IEEE Trans. Automat. Control*, Vol. AC–34, No. 11, pp. 1123–1131.

[113] Shirai, Y. and Tsujio, J. (1985), *Artificial Intelligence: Concepts, Techniques and Applications*, John Wiley & Sons.

[114] Singhal, S. and Wu, L. (1989), "Training multilayer perceptrons with the extended Kalman filter algorithm", in *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann Publishers, Inc., pp. 133–140.

[115] Slotine, J.–J.E. and Li, W. (1991), *Applied Nolinear Control Systems*, Prentice–Hall.

[116] Sontag, E.D. (1992), "Feedback stabilization using two–hidden–layer nets", *IEEE Trans. on Neural Networks*, Vol. 3, pp. 981–990.

[117] Sontag, E.D., (1990), *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, Pringer, New York.

[118] Sontag, E.D. and Sussmann, H.J. (1991), "Backpropagation seperates where perceptrons do", *Neural Networks*, Vol. 4, pp. 243–249.

[119] Spall, J.C. and Cristion, J.A. (1992), "Direct adaptive control of nonlinear stochastic systems using neural networks and stochastic approximation", *Proc. of the 31st IEEE Conf. on Decision and Control*, Tucson, AZ, pp. 878–883.

[120] Spong, M.W. (1989), "Adaptive control of flexible joint manipulators", *Systems and Control Letters*, Vol. 13, pp. 15–21.

[121] Spong, M.W. and Vidyasagar, M. (1989), *Robot Dynamics and Control*, John Wiley & Sons, New York, 1989.

[122] Sutton, R.S., Barto, A.G. and Williams, R.J. (1992), "Reinforcement learning is direct adaptive optimal control", *IEEE Control Systems Magazine*, Vol. 12, Apr., pp. 19–22.

[123] Tao, J.M. and Luh, J.Y.S. (1993), "Application of neural network with real–time training to robust position/force control of multiple robots", *Proc. 1993 IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, Vol. 1, pp. 142–148.

[124] Tarn, T.–J., Bejczy, A.K., Yun, X. and Li, Z. (1991), "Effect of motor dynamics on nonlinear feedback robot arm control", *IEEE Trans. on Robotics and Automation*, Vol. 7, No. 1, pp. 114–122.

[125] Tseng, H.C. and Hwang, V.H. (1991), "Neural networks for nonlinear servomechanism", *Proc. IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 2414–2417.

[126] Tsutsumi, K. and Matsumoto, H. (1987), "Neural computation and learning strategy for manipulator position control", *IEEE Int. Conf. on Neural Networks*, San Diego, CA, June.

[127] Ungar, L.H. and Narendra, K.S. (1992), *Neural Networks in Control Systems*, lecture notes of Workshop No. 8, American Control Conf., Chicago, IL.

[128] Utkin, V.I. (1976), *Control systems of variable structure*, Wiley, New York.

[129] Utkin, V.I. (1977), "Variable structure systems with sliding modes", *IEEE Trans. Automat. Contr.*, Vol. AC–22, No. 2, pp. 212–222.

[130] Werbos, P.J. (1989), "Neural networks for control and system identification", *Proc. 28th CDC*, Tampa, FL, pp. 260–265.

[131] Werbos, P.J. (1974), *Beyond regression: new tools for prediction and analysis in the behavioral science*, Ph.D dissertation, Harvard University, Cambridge, MA.

[132] White, D.A. and Sofge, D.A. (1992), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, New York.

[133] Widrow, B. (1990), "Adaptive inverse control", *Proc. SPIE*, Vol. 1294, pp. 12–21.

[134] Widrow, B. and Hoff, M.E. (1960), "Adaptive Switching Circuits", *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Board*, Part 4, pp. 96–104.

[135] Widrow, B. and Smith, F. (1963), "Pattern recognizing control systems", *Proc. Comput. Inform. Sci. (COINS) Symp.*, Spartan Brooks, Washington, DC.

[136] Wong, Y.–F. and Sideris, A. (1992), "Learning convergence in the cerebellar model articulation controller", *IEEE Trans. on Neural Networks*, Vol. 3, No. 1, pp. 115–121.

[137] Wu, Q.H., Irwin, G.W. and Hogg, B.W. (1991), "A neural network regulator", *Proc. IEE Int. Conf. on Control*, Edinburgh, UK, pp. 145–150.

[138] Xu, J.–X., Donne, J. and Ozguner, U. (1991), "Synthesis of feedback linearization and variable structure control with neural net compensation", *Proc. IEE Int. Symp. on Intelligent Control*, Arlington, VA, pp. 184–189.

[139] Yegerlehner, J.D. and Meckl, P.H. (1993), "Experimental implementation of neural network controller for robot undergoing large payload changes", *Proc. 1993 IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, Vol. 2, pp. 744–749.

[140] Ydstie, B.E. (1991), "Stability of the direct self–tuning regulator", in Kokotovic', P.V. (ed.), *Foundations of adaptive control*, Springer–Verlag, pp. 201–238.

[141] Zeman, V., R.V. Patel, and K. Khorasani (1989), "A neural network based control strategy for flexible–joint manipulators", *Proc. 28th IEEE Conf. on Decision and Control*, Tampa, FL, pp. 3025–3027.

APPENDIX A

THEOREM PROOFS

A.1 THE PROOF OF THEOREM 7.2.1

To prove theorem 7.2.1, consider the following Lyapunov function candidate:

$$V = \frac{1}{2}[s_{\Delta 1}^T M(q) s_{\Delta 1} + \tilde{\theta}_1^T \Gamma_1^{-1} \tilde{\theta}_1 + \tilde{\theta}_3^T \Gamma_3^{-1} \tilde{\theta}_3 + s_{\Delta 2}^T I_m s_{\Delta 2} + \tilde{\theta}_2^T \Gamma_2^{-1} \tilde{\theta}_2] \equiv V_1 + V_2 \tag{A.1.1}$$

where

$$V_1 = \frac{1}{2}[s_{\Delta 1}^T M(q) s_{\Delta 1} + \tilde{\theta}_1^T \Gamma_1^{-1} \tilde{\theta}_1 + \tilde{\theta}_3^T \Gamma_3^{-1} \tilde{\theta}_3] \tag{A.1.2}$$

$$V_2 = \frac{1}{2}[s_{\Delta 2}^T I_m s_{\Delta 2} + \tilde{\theta}_2^T \Gamma_2^{-1} \tilde{\theta}_2] \tag{A.1.3}$$

Eqs. (7.2.13) and (7.2.14) can be regarded as a large scale nonlinear system with two interconnected sub–systems. V_1 and V_2 are their corresponding Lyapunov function candidates.

Differentiating eq. (A.1.1) and using eqs. (7.2.13) and (7.2.14) gives:

$$\begin{aligned}\dot{V} &= s_{\Delta 1}^T M(q) \dot{s}_1 + \frac{1}{2} s_{\Delta 1}^T \dot{M}(q) s_{\Delta 1} + \tilde{\theta}_1^T \Gamma_1^{-1} \dot{\tilde{\theta}}_1 + \tilde{\theta}_3^T \Gamma_3^{-1} \dot{\tilde{\theta}}_3 + s_{\Delta 2}^T I_m \dot{s}_2 + \tilde{\theta}_2^T \Gamma_2^{-1} \dot{\tilde{\theta}}_2 \\ &= s_{\Delta 1}^T [Kp - g_1(q, \dot{q}, \ddot{q}_r, s_{\Delta 1})] + \tilde{\theta}_1^T \Gamma_1^{-1} \dot{\hat{\theta}}_1 + \tilde{\theta}_3^T \Gamma_3^{-1} \dot{\hat{\theta}}_3 \\ &\quad + s_{\Delta 2}^T [u - g_2(q, p, \dot{p}, \ddot{p}_r)] + \tilde{\theta}_2^T \Gamma_2^{-1} \dot{\hat{\theta}}_2\end{aligned}$$

Since

$$Kp - g_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) = [\hat{K}_N - \tilde{K} + K_d(t)]p - g_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) - d_1(t)$$

$$= \hat{K}_N\delta + \hat{K}_N p_d - \tilde{K}p - g_{N1}(q,\dot{q},\ddot{q}_r,s_{\Delta 1}) + \bar{d}_1(t)$$

$$= \hat{K}_N\delta - D_1 s_1 - \tilde{K}p + \Phi_1(q,\dot{q},\ddot{q}_r,s_{\Delta 1})^T\tilde{\theta}_1 + \bar{d}_1(t)$$

$$u - g_2(q,p,\dot{p},\ddot{p}_r) = -D_2 s_2 + \Phi_2(q,p,\dot{p},\ddot{p}_r)^T\tilde{\theta}_2 - d_2(t)$$

Using adaptation laws (7.2.35)–(7.2.37), we have:

$$\dot{V} = s_{\Delta 1}^T[\hat{K}_N\delta - D_1 s_1 - \tilde{K}p + \bar{d}_1(t)] - s_{\Delta 2}^T[D_2 s_2 + d_2(t)] + \tilde{\theta}_3^T\Gamma_3^{-1}\dot{\hat{\theta}}_3$$

and

$$-s_{\Delta 1}^T\tilde{K}p + \tilde{\theta}_3^T\Gamma_3^{-1}\dot{\hat{\theta}}_3 = -\tilde{\theta}_3^T\Phi_3(q-p)s_3(s_{\Delta 1},p) + \tilde{\theta}_3^T\Phi_3(q-p)s_3(s_{\Delta 1},p) = 0$$

$$\therefore\ \dot{V} = -s_{\Delta 1}^T[D_1 s_1 - \bar{d}_1(t)] + s_{\Delta 1}^T\hat{K}_N\delta - s_{\Delta 2}^T[D_2 s_2 + d_2(t)]$$

$$= -s_{\Delta 1}^T D_1 s_{\Delta 1} - \sum_{j=1}^{n}[D_{1jj}\Delta_1|s_{\Delta 1j}| - s_{\Delta 1j}\bar{d}_{1j}(t)] + s_{\Delta 1}^T\hat{K}_N\delta$$

$$- s_{\Delta 2}^T D_2 s_{\Delta 2} - \sum_{j=1}^{n}[D_{2jj}\Delta_2|s_{\Delta 2j}| + s_{\Delta 2j}d_{2j}(t)]$$

$$\le -s_{\Delta 1}^T D_1 s_{\Delta 1} + \sum_{j=1}^{n}|s_{\Delta 1j}|\ [\ |\bar{d}_{1j}(t)| - D_{1jj}\Delta_1] + s_{\Delta 1}^T\hat{K}_N\delta$$

$$- s_{\Delta 2}^T D_2 s_{\Delta 2} + \sum_{j=1}^{n}|s_{\Delta 2j}|\ [\ |d_{2j}(t)| - D_{2jj}\Delta_2]$$

$$\le -s_{\Delta 1}^T D_1 s_{\Delta 1} + \sum_{j=1}^{n}|s_{\Delta 1j}|\ [\sum_{i=1}^{n}|\hat{K}_{Nji}\delta_i| - D_{1jj}\Delta_{12}] - s_{\Delta 2}^T D_2 s_{\Delta 2}$$

by using the relations (7.2.19), (7.2.32), (7.2.33), and (7.2.34). Therefore,

$$\dot{V}_1 \le -s_{\Delta 1}^T D_1 s_{\Delta 1} + \sum_{j=1}^{n}|s_{\Delta 1j}|\ [\sum_{i=1}^{n}|\hat{K}_{Nji}\delta_i| - D_{1jj}\Delta_{12}] \qquad \text{(A.1.4)}$$

$$\dot{V}_2 \le -s_{\Delta 2}^T D_2 s_{\Delta 2} \qquad \text{(A.1.5)}$$

Since eq. (7.2.14) is decoupled from eq. (7.2.13) under neurocontrol algorithm (7.2.29) and (7.2.30), we can examine the convergence of signal $s_{\Delta 2}$ first. Under neurocontrol algorithm (7.2.29), eq. (7.2.14) becomes:

$$I_m \dot{s}_2 + D_2 s_2 = \Phi_2(q, p, \dot{p}, \ddot{p}_r)^T \tilde{\theta}_2 - d_2(t) \tag{A.1.6}$$

Because eq. (A.1.5) is valid for all $t \geq 0$, $s_{\Delta 2}$ and $\tilde{\theta}_2$ are bounded for all $t \geq 0$ if $s_{\Delta 2}(0)$ and $\tilde{\theta}_2(0)$ are bounded. Since s_2 is uniformly bounded and $s_2 = \dot{\delta} + \Lambda_2 \delta$, $\delta(t)$ is also bounded for all $t \geq 0$ if $\delta(0)$ is bounded. To prove the asymptotic convergence of $s_{\Delta 2}$, we apply Barbalat's lemma to the following continuous nonnegative function:

$$V_{2'}(t) = V_2(t) - \int_0^t [\dot{V}_2(\tau) + s_{\Delta 2}(\tau)^T D_2 s_{\Delta 2}(\tau)] d\tau \tag{A.1.7}$$

with $\quad \dot{V}_{2'}(t) = -s_{\Delta 2}^T D_2 s_{\Delta 2} \qquad \ddot{V}_{2'}(t) = -2 s_{\Delta 2}^T D_2 \dot{s}_{\Delta 2}$ (A.1.8)

By definition, $\dot{s}_{\Delta 2}$ is either 0 or $\dot{s}_2$. By neural network design, $\Phi_2(q, p, \dot{p}, \ddot{p}_r)$ and $d_2(t)$ are uniformly bounded. Therefore, $\dot{s}_2$, given by eq. (A.1.6), is bounded for all $t \geq 0$, which proves that $\ddot{V}_{2'}(t)$ is uniformly bounded. Hence, $\dot{V}_{2'}(t)$ is a uniformly continuous function of time. Using Barbalat's lemma proves that $\dot{V}_{2'}(t) \to 0$, thus $s_{\Delta 2}(t) \to 0$ as $t \to \infty$. This means that the inequality of $|s_{2i}(t)| \leq \Delta_2$ is obtained asymptotically, and the tracking error $\delta(t)$ is asymptotically bounded as:

$$|\delta_i(t)| \leq \Delta_2 / \lambda_{2i} \tag{A.1.9}$$

where λ_{2i} is the i–th diagonal entry of Λ_2.

Before $\delta(t)$ converges, $s_{\Delta 1}$ may converge to or diverge from zero temporarily, depending on the sign of $\dot{V}_1$. This features the characteristics of interconnected dynamics. After $\delta(t)$ converges to a neighborhood of zero, as shown in eq. (A.1.9), we have

$$\begin{aligned}\dot{V}_1 &\le -s_{\Delta 1}^T D_1 s_{\Delta 1} + \sum_{j=1}^{n} |s_{\Delta 1j}| \, [\sum_{i=1}^{n} |\hat{K}_{Nji}\delta_i| - D_{1jj}\Delta_{12}] \\ &\le -s_{\Delta 1}^T D_1 s_{\Delta 1} + \sum_{j=1}^{n} |s_{\Delta 1j}| \, [\sum_{i=1}^{n} |\hat{K}_{Nji}|\Delta_2/\lambda_{2i} - D_{1jj}\Delta_{12}] \\ &\le -s_{\Delta 1}^T D_1 s_{\Delta 1} \qquad \text{(A.1.10)}\end{aligned}$$

Similarly, we can prove that $s_{\Delta 1}$, $e(t)$, $\tilde{\theta}_1$ and $\tilde{\theta}_3$ are bounded for all $t \ge 0$ if their initial values are bounded. By design, $q_d(t) \in C^4$ is uniformly bounded, and θ_1 and θ_3 are constants, hence $q(t)$, $\dot{q}(t)$, $\hat{\theta}_1$ and $\hat{\theta}_3$ are uniformly bounded. Eq. (7.2.30) confirms that $p_d(t)$ is uniformly bounded. Thus $p(t)$ is bounded due to eq. (A.1.9). $u(t)$ is also uniformly bounded. In conclusion, all the signals in the adaptive system are uniformly bounded if their initial values are bounded.

Noticing that $\dot{s}_1$, given by eq. (A.1.12), is bounded. We conclude that the tracking error $e(t)$ is asymptotically bounded as:

$$|e_i(t)| \le \Delta_1/\lambda_{1i} \qquad \text{(A.1.11)}$$

where λ_{1i} is the i–th diagonal entry of Λ_1.

$$M(q)\dot{s}_1 + D_1 s_1 + \frac{1}{2}\dot{M}(q)s_{\Delta 1} = K\delta - \tilde{K}_N p_d + \Phi_1(q,\dot{q},\ddot{q}_r,p,s_{\Delta 1})^T\tilde{\theta}_1 + \bar{d}_1(t) \quad \text{(A.1.12)}$$

The theorem 7.2.1 is proved.

A.2 THE PROOF OF THEOREM 7.3.1

Theorem 7.3.1 is proved in the following. Due to the inherent global boundedness of the sigmoidal basis functions and Gaussian basis functions, the certainty–equivalence approach can be applied to derive the control law. As usual, the first step of the certainty–equivalence design is to find a dynamic feedback control law that guarantees the specified stability and tracking performances when the weight matrix W is known. Then the unknown W is replaced with its estimate $\hat{W}$ and the so called "certainty–equivalence" control law is implemented. The boundedness of the signals in the system and the tracking convergence are proved using the Lyapunov method.

Using the stable filter $F(D)/E(D)$ to filter eq. (7.3.7) gives:

$$\frac{F(D)D^4}{E(D)}q = -\frac{F(D)}{E(D)}h(q,\dot{q},p,\dot{p}) + \frac{F(D)}{E(D)}R(q)u \tag{A.2.1}$$

Substituting eqs. (7.3.15), (7.3.8) and (7.3.9) into eq. (A.2.1), and regarding W_h and W_i ($i = 1, \ldots, n$) as constant matrices, we have:

$$A_m(D)q = \frac{G(D)}{E(D)}q - W_h^T\frac{F(D)}{E(D)}\Phi_h + W_R^T\Phi_R u + W_R^T\frac{F(D)-E(D)}{E(D)}(\Phi_R u) + d(t)$$

$$= \frac{G(D)}{E(D)}q - W_h^T\overline{\Phi}_h + W_R^T\Phi_R u + W_R^T(c_1 U)^T + d(t) \tag{A.2.2}$$

where

$$d(t) = \frac{F(D)}{E(D)}\{\ [d_1(t)u_1 \ \cdot\ \cdot\ \cdot\ d_n(t)u_n] - d_h(t)\} \tag{A.2.3}$$

and $\| d(t) \|$ can be uniformly bounded by any user–specified small constant, if $\| u(t) \|$ is uniformly bounded. Substituting eq. (7.3.25) into eq. (A.2.2) generates:

$$A_m(D)q = r + \tilde{W}^T\overline{\Phi} + d(t) \tag{A.2.4}$$

Noticing the reference model (7.3.10), we have:

$$A_m(D)e = \tilde{W}^T\overline{\Phi} + d(t) \tag{A.2.5}$$

or

$$e = A_m(D)^{-1}[\tilde{W}^T\overline{\Phi}] + (D+\lambda_0)^{-1}[\overline{d}(t) + \varepsilon(t)] \tag{A.2.6}$$

where $\varepsilon(t)$ stands for the exponentially decaying tracking error caused by the mismatch of the initial values of $q_r^{(i)}(0)$ and $q^{(i)}(0)$; $\overline{d}(t) = A_{m0}(D)^{-1}d(t)$.

Next, we apply the well–known swapping technique to transform the tracking error equation (A.2.6) into a strictly positive real form. Let

$$e = \frac{1}{D+\lambda_0}\{\tilde{W}^T[A_{m0}(D)^{-1}\overline{\Phi}] + \overline{d}(t) + \varepsilon(t)\}$$

$$+ \frac{1}{D+\lambda_0}\{\ A_{m0}(D)^{-1}[\tilde{W}^T\overline{\Phi}] - \tilde{W}^T[A_{m0}(D)^{-1}\overline{\Phi}]\ \} \tag{A.2.7}$$

$$= \frac{1}{D+\lambda_0}\{\tilde{W}^T\psi + \overline{d}(t) + \varepsilon(t)\} - \eta_0 \tag{A.2.8}$$

where η_0, defined by eq. (7.3.33), is obtained from eq. (A.2.7) using the swapping lemma.

From eqs. (7.3.17) and (A.2.8), we know that

$$\overline{e} = \frac{1}{D+\lambda_0}\{\tilde{W}^T\psi + \overline{d}(t) + \varepsilon(t)\} \tag{A.2.9}$$

Assume that the i–th entry of $\overline{d}(t)$ satisfies $|\overline{d}_i(t)| < \lambda_0\Delta$. Consider the following Lyapunov function candidate:

$$V = \frac{1}{2}[\overline{e}_\Delta^T\overline{e}_\Delta + \gamma^{-1}tr(\tilde{W}^T\tilde{W}) + \int_t^\infty \varepsilon(\tau)^T\varepsilon(\tau)d\tau] \tag{A.2.10}$$

We have:

$$\dot{V} = \overline{e}_\Delta^T\dot{\overline{e}}_\Delta + \gamma^{-1}tr(\tilde{W}^T\dot{\tilde{W}}) - \frac{1}{2}\varepsilon(t)^T\varepsilon(t)$$

$$= \overline{e}_\Delta^T\dot{\overline{e}} + \gamma^{-1}tr(\tilde{W}^T\dot{\hat{W}}) - \frac{1}{2}\varepsilon(t)^T\varepsilon(t)$$

$$= \overline{e}_\Delta^T[-\lambda_0\overline{e} + \tilde{W}^T\psi + \overline{d}(t) + \varepsilon(t)] - tr(\tilde{W}^T\psi\overline{e}_\Delta^T) - \frac{1}{2}\varepsilon(t)^T\varepsilon(t)$$

$$= \sum_{i=1}^{n}\overline{e}_{\Delta i}[-\lambda_0\overline{e}_i + \overline{d}_i(t)] + \overline{e}_\Delta^T\varepsilon(t) - \frac{1}{2}\varepsilon(t)^T\varepsilon(t)$$

$$= \sum_{i=1}^{n}[-\lambda_0\overline{e}_{\Delta i}^2 - \lambda_0\Delta|\overline{e}_{\Delta i}| + \overline{e}_{\Delta i}\overline{d}_i(t)] + \frac{1}{2}\overline{e}_\Delta^T\overline{e}_\Delta - \frac{1}{2}[\varepsilon(t) - \overline{e}_\Delta(t)]^T[\varepsilon(t) - \overline{e}_\Delta(t)]$$

$$< -(\lambda_0 - \frac{1}{2})\overline{e}_\Delta^T\overline{e}_\Delta - \frac{1}{2}[\varepsilon(t) - \overline{e}_\Delta(t)]^T[\varepsilon(t) - \overline{e}_\Delta(t)] \tag{A.2.11}$$

It can be seen that if $\lambda_0 > 1/2$, or after $\varepsilon(t) \to 0$, $\dot{V} \le 0$.

Assume that the initial values of all the signals in the neurocontrol system are bounded. Eqs. (A.2.10) and (A.2.11) ensure that $\bar{e}_{\Delta}$, thus $\bar{e}$, $\tilde{W}$ are uniformly bounded. Therefore, $\hat{W}$, $\hat{W}_h$, $\hat{W}_i$, and $\hat{W}_R$ are all uniformly bounded, because their true values exist and are bounded. Since $q^{(i)}(0)$ is bounded, $q^{(i)}(t)$ (i=0, 1, . . .,3) is bounded at least in the finite time interval (0, t_f), due to their continuity. $\Phi_h(q,\dot{q},p,\dot{p})$ and $\Phi_R(q)$ are related to the outputs of the hidden layer neurons, and are uniformly bounded for any variables, which reflects the advantage of neural network approach. If necessary, special techniques such as pseudo–inverse, or adding μI to $\hat{R}(q)$ (μ is a small positive constant), can be adopted to ensure that $\hat{R}(q)^{-1}$ is always bounded for any estimate $\hat{W}_R$. Eqs. (7.3.25) and (7.3.27) make sure that matrix U is bounded in the finite time interval (0, t_f). All of these guarantee that control vector u is bounded at least in (0, t_f). In practice, hard limits on the control magnitude can guarantee that u is uniformly bounded.

As long as $u(t)$ is bounded, eq. (7.3.31) guarantees that $\overline{\Phi}(q,\dot{q},p,\dot{p},u)$ is bounded. Since the filter defined by eqs. (7.3.29) and (7.3.30) is BIBO, Z and ψ are also bounded. Noticing that $\bar{d}(t)$, $\varepsilon(t)$ and $\dot{\varepsilon}(t)$ are always bounded, we conclude from eq. (A.2.9) that $\bar{e}(t)$ and $\dot{\bar{e}}(t)$ are bounded. Similarly, the boundedness of $\dot{\hat{W}}, X$, and η_0 can be consequently concluded from eqs. (7.3.28), (7.3.32) and (7.3.33). Therefore, $e^{(i)}(t)$ is bounded for $t \in (0,\ t_f)$ and $i = 0, 1, . . .,3$, derived from eqs. (7.3.17) and (A.2.6). Because $q_d \in C^4 \cap L_\infty$, $q^{(i)}(t_f)$ is bounded ($i = 0, 1, . . .,3$), and the interval (0, t_f) can be extended to (0, $2t_f$) in which $q^{(i)}(t)$ ($i = 0, 1, . . .,3$) is bounded. Recursively, the time interval is extended to (0, ∞).

The internal states p and $\dot{p}$ are determined by eq. (7.1.2) or (7.3.4). Because $u(t)$ and $q(t)$ are uniformly bounded and eq. (7.1.2) or (7.3.4) is BIBO, p, $\dot{p}$, and $\ddot{p}$ are also uniformly bounded. We can conclude that all the above signals in the adaptive system are uniformly bounded, in spite of the existence of representation errors of neural networks.

Using Barbalat's lemma gives that $\lim_{t\to\infty} \bar{e}_{\Delta}(t) = 0$. Hence, $\lim_{t\to\infty} \dot{\hat{W}}(t) = 0$, $\lim_{t\to\infty} X(t) = 0$, and $\lim_{t\to\infty} \eta_0(t) = 0$. $\lim_{t\to\infty} |e_i(t)| < \Delta$ (i=0, 1, . . ., n).

Unlike conventional adaptive control schemes, the tracking error boundary Δ can be reduced to any small number by proper neural network structure design. Thus, the end–effector of the controlled FJR with unknown dynamics can track any given trajectory with user–specified precision. This concludes the proof.