

FUZZY MODELLING AND ROBUST CONTROL WITH APPLICATIONS TO ROBOTIC MANIPULATORS

FEI MEI

B. Sc., Nanjing University, P. R. China

M. Sc., Nanjing University, P. R. China

*A thesis submitted for the degree of Doctor of Philosophy
at The University of Tasmania*

School of Engineering
Faculty of Science and Engineering
The University of Tasmania
Hobart, Tasmania 7001
Australia

December, 1999

Authority of Access

This thesis may be made available for loan. Copying of any part of this thesis is prohibited for two years from the date this statement was signed; after that time limited copying is permitted in accordance with the Copyright Act 1968.

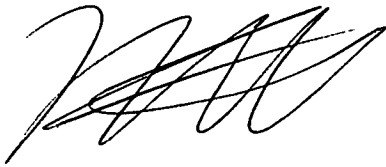
A stylized, cursive handwritten signature in black ink, appearing to read 'Fei Mei'.

Fei Mei

A handwritten date in black ink, written as '6/6/2000'.

Statement of Originality

The work contained in this PhD thesis is original, to the best of my knowledge and belief, except as acknowledged in the text. This thesis contains no material which has been accepted for the award of any other degree or diploma in any tertiary institution.

A handwritten signature in black ink, appearing to be 'Fei Mei', written in a cursive style.

Fei Mei

Acknowledgements

I wish to thank the many people who have helped me complete this thesis. For encouragement and support, I would especially like to thank my supervisor and friend, Dr. Man Zhihong. His enthusiasm and motivation has helped me greatly. I am particularly grateful to Prof. Thong Nguyen, my co-supervisor, for his valuable advice and consistent encouragement during my work. My grateful thanks also go to all the members and postgraduates in the School of Engineering for their kind reception and cooperation that I found during my PhD candidature at The University of Tasmania. For financial support, I am grateful to The University of Tasmania for providing me with Tasmania International Scholarships. Finally, I would like to thank my wife Bing Xu, my daughter, Lisa Jun Mei, my parents and my friends for their love, understanding and support during this period.

Abstract

In this thesis, fuzzy modelling of a class of nonlinear systems has been investigated based on fuzzy logic and linear feedback control theory, and a few robust variable structure control schemes for nonlinear systems have been developed. A number of robustness and convergence results with dramatically reduced control chattering are presented for variable structure control systems with applications to robotic manipulators in the presence of parameter variations and external disturbances. The major outcomes of the work described in this thesis are summarised as follows.

A robust tracking control scheme is proposed for a class of nonlinear systems with fuzzy model. It is shown that a nominal system model for a nonlinear system is established by fuzzy synthesis of a set of linearised local subsystems, where the conventional linear feedback control technique is used to design a feedback controller for the fuzzy nominal system. A variable structure compensator is then designed to eliminate the effects of the approximation error and system uncertainties. Strong robustness with respect to large system uncertainties and asymptotic convergence of the output tracking error are obtained.

A sliding mode control scheme using fuzzy logic and Lyapunov stability theory has been proposed. It is shown that a sliding mode is first designed to describe the desired system dynamics for the controlled system. A set of fuzzy rules are then used to adjust the controller's parameters based on the Lyapunov function and its time derivative. The desired system dynamics are then obtained in the sliding mode. The sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering of the control signals, compared with the conventional sliding mode controllers.

A robust continuous sliding mode control scheme for linear systems with uncertainties has been presented. The controller consists of three components: equivalent control, continuous reaching mode control and robust control. It retains the positive properties of sliding mode control but without the disadvantage of control chattering. The

proposed control scheme has been applied to the tracking control of a one-link robotic manipulator with fuzzy modelling of the nonlinear system.

A robust adaptive sliding mode control scheme with fuzzy tuning has been presented. It is shown that an adaptive sliding mode control is first designed to learn the system parameters with bounded system uncertainties and external disturbances. A set of fuzzy rules are then used to adjust the controller's uncertainty bound based on the Lyapunov function and its time derivative. The robust adaptive sliding mode controller with fuzzy tuning algorithm show the advantage of reducing the chattering and the amplitude of the control signals, compared with the adaptive sliding mode controller without fuzzy tuning. Experimental example for a five-bar robot arm is given in support of the proposed control scheme.

Finally, a new adaptive sliding mode controller has been developed for trajectory tracking in robotic manipulators. This controller is able to estimate the constant part of the system parameters as well as adaptively learn the uncertain part of the system parameters by the Gaussian neural network. It is shown that under a mild assumption, the proposed control law does not require measurement of acceleration signals. This new control law exhibits the good aspects of Slotine and Li's (1987) and keeps the chattering to a minimum level. An experiment of a five bar robotic system was done and the results have confirmed the effectiveness of the approach.

List of Publications

1. Fei Mei, Man Zhihong, Yu Xinghuo, Thong Nguyen, "A Robust Tracking Control Scheme for a Class of Nonlinear Systems with Fuzzy Nominal Model", *International Journal of Applied Mathematics and Computer Science*, vol.8, No.1, pp.145-158, 1998.
2. Fei Mei, Man Zhihong, Thong Nguyen, "Fuzzy Modelling and Tracking Control of Robotic Manipulators", to appear in *Mathematical and Computer Modelling*, 1999.
3. Fei Mei, Man Zhihong, Xinghuo Yu, "Robust adaptive sliding mode control of robots", submitted to *IEEE Transactions on Industrial Electronics*, 1999.
4. Xinghuo Yu, Man Zhihong, S S Cong, Fei Mei, "Robust adaptive sliding mode control of robotic manipulators", *International Journal of Robotics and Automation*, vol.14, no.2, pp. 54-60, 1999.
5. Fei Mei, Man Zhihong, "A Sliding Mode Control System with Fuzzy Logic Controller", *International Conference on Computational Intelligence and Multimedia Applications*, 9-11 February, 1998, Monash University, Australia.
6. Fei Mei, Man Zhihong, "Fuzzy Modelling and Tracking Control of Nonlinear Systems", *International Congress on Modelling and Simulation Proceedings*, pp. 902-906, 7-12 December, 1997, Hobart, Australia.
7. Fei Mei, Man Zhihong, Thong Nguyen, "Continuous sliding mode control with limited control input", *Proceedings of the Australian Universities Power Engineering Conference (AUPEC'98)*, Hobart, Australia, vol.1, pp212-216, September, 1998.
8. Fei Mei, Man Zhihong, Thong Nguyen, "The terminal controller design with application to robotic manipulators", *Proceedings of the Australian Universities Power Engineering Conference (AUPEC'98)*, Hobart, Australia, vol.2, pp532-536, September, 1998.
9. Fei Mei, Michael Negnevitsky, "A robust continuous sliding mode control scheme", *Proceedings of 6th International Conference on Fuzzy Theory and Technology*, Durham, USA, vol.1, pp292-294, October 23-28, 1998.

10. Fei Mei, "Sliding mode control signal analysis by discrete wavelet transform", *Proceedings of the 14th World Congress of International Federation of Automatic Control (IFAC'99)*, Beijing, China, vol. H, pp. 349-353, July 5-9, 1999.
11. Man Zhihong, Fei Mei, "A sliding mode control for nonlinear SISO systems with a new fuzzy model", *Proceedings of the 14th World Congress of International Federation of Automatic Control (IFAC'99)*, Beijing, China, vol. Q, pp. 359-362, July 5-9, 1999.
12. Fei Mei, Man Zhihong, Xinghuo Yu, Wei Lai, "RBF network based sliding mode control of robots", *Proceedings of the IEEE Hong Kong Symposium on Robotics and Control*, Hong Kong, vol. 1, pp.27-32, July 2-3, 1999.

Contents

Statement of Originality.....	ii
Acknowledgements	iii
Abstract	iv
List of Publications.....	vi
1 Introduction.....	1
1.1 MOTIVATION	1
1.2 SCOPE	4
1.3 THESIS OUTLINE	5
2 A survey of variable structure control theory	9
2.1 INTRODUCTION.....	9
2.2 BASIC VARIABLE STRUCTURE CONTROL THEORY.....	11
2.2.1 SYSTEM MODEL AND SLIDING MODE	11
2.2.2 EQUIVALENT CONTROL.....	13
2.2.3 ROBUSTNESS PROPERTY	16
2.2.4 TWO METHODS OF SLIDING MODE DESIGN.....	17
2.2.5 CONTROLLER DESIGNS.....	21
2.3 VARIABLE STRUCTURE CONTROL OF NONLINEAR SYSTEM	22
2.3.1 SYSTEM MODEL	22
2.3.2 SLIDING MODE AND EQUIVALENT CONTROL.....	23
2.3.3 CONTROLLER DESIGN	24
2.3.3.1 Diagonalisation method.....	24

2.3.3.2 Reaching law method	25
2.3.4 ROBUST CONTROL OF NONLINEAR SYSTEMS	27
2.4 APPLICATION TO ROBOT MANIPULATORS.....	29
2.4.1 DYNAMICS OF ROBOTIC MANIPULATORS.....	29
2.4.2 A ROBUST VSC CONTROLLER DESIGN	31
2.5 CONCLUDING REMARKS	33
3 Fuzzy logic and fuzzy logic controller.....	35
3.1 INTRODUCTION.....	35
3.2 FUZZY SET THEORY.....	37
3.2.1 FUZZY SETS	38
3.2.2 SET THEORETICAL OPERATORS	40
3.2.3 THE EXTENSION PRINCIPLE	42
3.2.4 FUZZY RELATIONS AND THEIR COMPOSITIONS	43
3.2.5 LINGUISTIC REPRESENTATION	45
3.3 FUZZY LOGIC AND FUZZY REASONING	48
3.4 FUZZY LOGIC CONTROL.....	54
3.5 CONCLUDING REMARKS	68
4 Fuzzy modelling and robust tracking control of nonlinear systems.....	70
4.1 INTRODUCTION.....	70
4.2 LINEARISATION OF NONLINEAR SYSTEMS.....	72
4.3 FUZZY MODELLING AND TRACKING CONTROL OF NONLINEAR SYSTEMS.....	74
4.3.1 FUZZY MODELLING AND TRACKING CONTROLLER DESIGN	74
4.3.2 A SIMULATION EXAMPLE.....	76
4.4 ROBUST TRACKING CONTROL WITH FUZZY NOMINAL MODEL	86
4.4.1 A ROBUST TRACKING CONTROL SCHEME.....	86

4.4.2	A SIMULATION EXAMPLE.....	90
4.5	CONCLUDING REMARKS	93
5	Fuzzy sliding mode control.....	98
5.1	INTRODUCTION.....	98
5.2	SLIDING MODE CONTROL OF NONLINEAR SYSTEMS.....	99
5.3	FUZZY TUNING OF THE SLIDING MODE CONTROLLER.....	101
5.4	AN ILLUSTRATIVE EXAMPLE.....	106
5.5	CONCLUDING REMARKS	111
6	Robust continuous sliding mode control.....	112
6.1	INTRODUCTION.....	112
6.2	A ROBUST CONTINUOUS SLIDING MODE CONTROL.....	114
6.3	A SIMULATION EXAMPLE	117
6.4	CONCLUDING REMARKS	120
7	Fuzzy adaptive sliding mode control	132
7.1	INTRODUCTION.....	132
7.2	ROBUST ADAPTIVE SMC WITH FUZZY TUNING	134
7.3	AN ILLUSTRATIVE EXAMPLE.....	142
7.4	CONCLUDING REMARKS	145
8	Robust adaptive sliding mode control of robots.....	153
8.1	INTRODUCTION.....	153
8.2	THE ROBUST ADAPTIVE SMC DESIGN	155
8.3	EXPERIMENTAL RESULTS	163
8.4	CONCLUDING REMARKS	167

9	Conclusions.....	172
9.1	SUMMARY	172
9.2	SUGGESTIONS FOR FUTHER WORK.....	175
	References.....	177
	Appendix.....	194
A	HARDWARE SETUP FOR A FIVE-BAR ROBOT ARM.....	194
B	C++ PROGRAMS FOR A FUZZY SLIDING MODE CONTROLLER.....	198
	FSMC.CPP.....	198
	FSMC DATA.TXT.....	208
	FSMC DATA.H.....	209
	COMPLEX.CPP.....	212
	CONTAINER.CPP.....	218
	LEAST SQUARES.CPP	222
	MISC.CPP	230
	MY_MATH.CPP	231
	MY_PROCESS.CPP	244
	PATH.CPP.....	248
	PC30 GIVEIO.CPP	250
	PCL833.CPP	254
	RUNGA KUTTA 4 TH ORDER.CPP	258
	VECTOR.CPP	260

Chapter 1

Introduction

1.1 Motivation

The basis for control system design and stability analysis is a dynamic mathematical model that captures prominent features of the system under consideration. However, in practical situations, such a requirement is not feasible because the controlled systems have high nonlinearities and uncertain dynamics, and simple linear or nonlinear differential equations cannot sufficiently represent the corresponding practical systems, and therefore, the designed controller based on such a model cannot guarantee the good performance such as stability and robustness.

During the last few years, fuzzy logic control has been suggested as an alternative to conventional control techniques for complex nonlinear systems due to the fact that fuzzy logic combines human heuristic reasoning and expert experience to approximate a certain desired behaviour function (Takagi and Sugeno, 1985; Cao et al., 1996; Wang et al., 1996). However, the asymptotic error convergence and stability of the closed-loop system may not be obtained due to the approximation error and uncertainties of the fuzzy model.

Many kinds of fuzzy models for control processes have been developed since Mamdani's (1974) paper was published. They can be classified into three kinds of models, Composition Rule of Inference (Zaheh, 1973), Approximate Reasoning Model (Nakanishi et al., 1993), Sugeno's Models such as Position type Model and Position-Gradient type Model (Sugeno and Yasukawa, 1993). Most of these models are expressed by a set of fuzzy linguistic propositions which are derived from the experience of skilled operators or by fuzzy implication which locally represent linear input-output relations of the system.

Most proposed conventional fuzzy models only consider the external behaviour of the system, and can be considered as a function approximation. It is very difficult to obtain a controller using those models. Even if the controller can be obtained by using some trial-and-error procedures the behaviour of the closed-loop system, for example, the stability of the system is still difficult to analyse. Also the number of rules increase very quickly when the system becomes complex because every local rule is only described by a constant. Therefore, the identification of these fuzzy models is still a difficult problem because there are too many parameters in the membership functions.

From a control point of view, system uncertainties can be classified as either structured or unstructured. Structured uncertainties are those dynamics that have a known functional form but unknown parameters, while unstructured uncertainties are simply those that are not structured. For example, system parameters and payload for a robotic system can be viewed as structured uncertainties; unstructured uncertainties include friction, disturbances, and unmodelled dynamics.

Two major conventional control methodologies have been developed for dealing with system uncertainties: *adaptive control* and *robust control*. Adaptive control is a control scheme in which so-called adaptation laws are constructed to learn explicitly unknown constant parameters of the system under control. For this reason, adaptive control is limited to those systems whose uncertainties are structured, although it is applicable to a wider range of uncertainties after employing robustness enhancement techniques. Robust control is a control of fixed structure that guarantees stability and performance for uncertain systems. Its design only requires some knowledge about bounding functions on the greatest possible value of the uncertainties. This implies that robust control is capable of compensating for both structured and unstructured uncertainties.

Variable structure control with sliding mode is a robust control technique with respect to system variations and external disturbances. Variable structure control was pioneered in the former Soviet Union in the 1960s by Emelyanov (1962, 1966) and then developed by many researchers (Utkin, 1971, 1977, 1978, 1983; Itks, 1976; Young, 1978, 1988; Slotine and Sastry, 1983; Gao and Hung, 1993). However, the control technique has not been widely accepted in the practical control engineering community, due mainly to the worry of chattering which is inherent in the variable structure control system.

This introduces the possibility of using conventional variable structure control method, and fuzzy logic technique to develop better control schemes for complex systems. In other words, “can fuzzy logic (with its powerful capabilities for modelling and control

of complex systems) and conventional linear or nonlinear control theory be combined to improve the system performance and control quality?”. The aim of this thesis is to show that by considering both these areas, superior system performance and control quality can be achieved.

1.2 Scope

The aim of this thesis is to present new robust control schemes by incorporating artificial intelligent techniques such as fuzzy logic and neural networks with conventional variable structure control system. In line with this, a review of basic variable structure control theory and discussion of recent research results on the robust variable structure control for a class of nonlinear systems with uncertain dynamics is given.

Fuzzy logic and fuzzy logic control are also reviewed to present a background to the methodology to be employed. Fuzzy sets, fuzzy reasoning, and fuzzy controller design are described.

The body of the thesis is devoted to fuzzy modelling of a class of nonlinear systems and developing robust variable structure control schemes by employing fuzzy logic, neural networks and adaptive control techniques. The rationale is explained more fully at the end of Chapter 2 and 3, followed by the robust control scheme development in the succeeding chapters. A review of the contents of the thesis is given in Section 1.3.

1.3 Thesis outline

The thesis is organised as follows.

In Chapter 1, the major thrust of the thesis, the motivation, scope and thesis outline are introduced.

In Chapter 2, the basic theory of variable structure control systems is briefly surveyed. Because the variable structure theory has many good features, it can be easily used to design controllers for linear or nonlinear systems. Although the robustness can be achieved without the exact knowledge of the control system, the system performance and control quality depend very much on the choosing of sliding mode parameters and the estimating of bounding functions of the system's unknown parts. In practice, excessive control input and severe control chattering which may excite unmodelled high-frequency dynamics are highly undesirable. In the following chapters of this thesis, several new and improved robust variable structure control schemes of nonlinear systems will be proposed by combining conventional methods and recently developed techniques, namely fuzzy logic and neural networks, and it will be shown to improve the system performance and enhance the control quality.

Chapter 3 provides a background of fuzzy logic and fuzzy logic control techniques to be applied in the later chapters. Fuzzy sets, fuzzy set operations, and fuzzy linguistic representation such as linguistic variables and linguistic modifiers (hedged) will be briefly outlined. Fuzzy reasoning or approximate reasoning is considered from the engineering viewpoint with IF-THEN fuzzy implications using Mamdani's minimum inference and Larsen's product inference. A fuzzy logic controller, mapping an input data vector into a scalar control output, normally comprises a rule base, fuzzifier and

defuzzifier. Commonly-used kinds of membership functions are described, focusing on control applications. The prominent advantage of the fuzzy logic controller is that it can effectively control complex ill-defined systems having nonlinearities, parameter variations and disturbances. However, there also exist some impediments in the design of the fuzzy logic controller. In general, fuzzy rules are obtained on the basis of intuition and experience, and membership functions are selected by trial and error procedure. Moreover, it is not easy to mathematically prove the system stability and robustness due to linguistic expression of the fuzzy rules. Therefore, a systematic design method of the fuzzy logic controller from which the stability and robustness can be clearly seen is to be explored (Lee, 1990). The succeeding chapters will employ fuzzy logic to establish system model of a class of nonlinear systems and enhance robustness and control quality of variable structure control systems.

In Chapter 4, a robust tracking control scheme is proposed for a class of nonlinear systems. The main contribution of this scheme is that a nominal system model for a nonlinear system is established by fuzzy synthesis of a set of linearised local subsystems, where the conventional linear feedback control technique is used to design a feedback controller for the fuzzy nominal system. A variable structure compensator is then designed to eliminate the effects of the approximation error and system uncertainties. Strong robustness with respect to large system uncertainties and asymptotic convergence of the output tracking error are obtained. A simulation example is given to support the proposed control scheme.

In Chapter 5, Lyapunov stability theory and fuzzy logic technique are combined together to design sliding mode control systems. It is shown that a sliding mode is

first designed to describe the desired system dynamics for the controlled system. A set of fuzzy rules are then used to adjust the controller's parameters based on the Lyapunov function and its time derivative. The desired system dynamics are then obtained in the sliding mode. The sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering of the control signals, compared with the conventional sliding mode controllers. The fuzzy tuning algorithm is also applied to the adaptive sliding mode control. Simulation and experimental examples are given in support of the proposed control scheme.

In Chapter 6, a robust continuous sliding mode control scheme for linear systems with uncertainties is developed. The controller consists of three components: equivalent control, continuous reaching mode control and robust control. It retains the positive properties of sliding mode control but reduces the disadvantage of control chattering. The proposed control scheme is applied to the tracking control of a one-link robotic manipulator by fuzzy modelling of the nonlinear system.

In Chapter 7, Lyapunov stability theory and fuzzy logic technique are combined together to design fuzzy adaptive sliding mode control systems. It is shown that an adaptive sliding mode control is first designed to learn the system parameters with bounded system uncertainties and external disturbances. A set of fuzzy rules are then used to adjust the controller's uncertainty bound based on the Lyapunov function and its time derivative. The robust adaptive sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering and the amplitude of the control signals, compared with the adaptive sliding mode controller without fuzzy

tuning. Experimental example for a five-bar robot arm is given in support of the proposed control scheme.

In Chapter 8, a new adaptive sliding mode controller is developed for trajectory tracking of robotic manipulators. This controller is able to estimate the constant part of the system parameters as well as adaptively learn the uncertain part of the system parameters by the Gaussian neural network. It is shown that under a mild assumption, the proposed control law does not require measurement of acceleration signals. This new control law exhibits the good properties as shown in Slotine and Li (1987) and keeps the chattering to a minimum level. An experiment for a five bar robotic system is carried out to confirm the effectiveness of the approach.

Chapter 9 summarises the results and draws conclusions. A brief review of each chapter is given, noting the important results. Topics and aspects for future work are suggested.

Appendix

The detailed hardware setup and C++ real time control programs for a five bar robotic manipulator are presented.

Chapter 2

A Survey of The Variable Structure Control Theory

2.1 Introduction

We have mentioned in chapter one that the variable structure control theory is a robust control with respect to system uncertainties and external disturbances. Generally speaking, variable structure control can be considered to be an extension of conventional feedback control in the sense that the structure of a state feedback regulator is allowed to change as its states cross discontinuity surfaces, which results in discontinuous feedback control input on one or more manifolds in the state space. From the point of the conventional feedback control theory, a variable structure control system can be treated as a combination of subsystems. Each subsystem has a fixed structure and operates in a specified region of the state space. The combination of these subsystems according to some prescribed rules results in a new system which is different from the individual subsystems and has the desired system response.

The main feature of a variable structure control system is the sliding motion. For the design of a variable structure controller, the first thing is to define a set of switching plane variables which are a function of the system states. The intersection of these

switching planes forms a sliding mode. The purpose of the variable structure controller is to drive the system states into the sliding mode on which the sliding motion occurs and the motion of the system is thus formally equivalent to a system of low order, called as equivalent system. Actually, the sliding motion on the sliding mode is the convergence motion of the system states from arbitrary initial values to the origin. The convergence rate depends on the design of sliding mode parameters. It is due to this feature that the variable structure control is also called *sliding mode control*.

Another feature of a variable structure system is that the transient response can be divided into two parts. First, the motion in which the variable structure controller drives the switching plane variables to reach the sliding mode. Second, the sliding motion in which the system states constrained on the sliding mode asymptotically converge to the origin. Usually, the sliding motion is determined only by the sliding mode parameters. However, the convergence of the switching plane variables are affected by the sliding mode parameters because the sliding mode parameters are involved in the controller gain matrices.

In this chapter, we will first review the basic variable structure control theory that has been useful in establishing robust variable structure control algorithms. In view of the focus of the thesis, we will then restrict our discussion to recent research results on the robust variable structure control for a class of nonlinear systems with uncertain dynamics.

In section 2.2 of this chapter, the basic variable structure control theory is briefly reviewed. The basic ideas and definitions, such as system model, the sliding mode, the condition for existence of sliding mode, robustness property, and an overview of four variable structure controllers, are discussed. In section 2.3, we deviate to address more complicated variable structure control for a class of nonlinear systems. In section

2.4, a robust variable structure controller design using reaching law method has been presented for robot manipulators.

2.2 Basic variable structure control theory

2.2.1 System model and sliding mode

Consider the following linear time invariant system

$$\dot{X}(t) = A X(t) + B u(t) \quad (2.1)$$

where $X \in R^n$ and $u \in R^m$ represent the state and control vectors, $A \in R^{n \times n}$ and $B \in R^{n \times m}$ are constant system matrices. It is assumed that $n > m$, B is of full rank m , and the pair (A, B) is completely controllable.

Define a set of switching plane variables s_i ($i = 1 \dots m$) passing through the state space origin

$$s_i = C_i X \quad i = 1 \dots m \quad (2.2-a)$$

$$\text{or} \quad S = C X \quad (2.2-b)$$

where $C_i \in R^n$ is a constant vector and

$$C = [C_1^T \dots C_m^T]^T \quad (2.2-c)$$

is an $n \times m$ constant matrix.

System (2.1) is said to attain a sliding mode when the state vector X reaches and remains on the intersection ($S = 0$) of the m switching plane variables

$$S_0 = \bigcap_{i=1}^m s_i = \{ X: C_i X = 0, i = 1 \dots m \} \quad (2.3)$$

The control input vector $u(t)$ in the variable structure control system usually has the following form (Utkin, 1977)

$$u(t) = K X + \psi X \quad (2.4)$$

where the first term in expression (2.4) is a linear feedback and the second term is a switching component. $\psi = [\psi_{ij}]$ and

$$\Psi_{ij} = \begin{cases} \alpha_{ij} & s_i x_j < 0 \\ \beta_{ij} & s_i x_j > 0 \end{cases} \quad (2.5)$$

The task of the control input $u(t)$ in expression (2.4) is to drive the switching plane variables to reach the sliding mode (2.3) by the suitable design of the controller gain matrices K and ψ . Thereafter, the system performance will be determined by the sliding motion on the sliding mode. Generally, the sliding mode is designed such that the system response restricted on the sliding mode has a desired behaviour such as asymptotic stability and prescribed transient response. Usually, the switching plane variables are designed as a linear functions of the system states. Many researches have shown that it is convenient for the linear sliding mode to be used in the design and analysis for a variable structure control system.

The next important problem is how to design controller parameters to guarantee the switching plane variables to reach the sliding mode, and then remain on the sliding mode. The work in Utkin (1977, 1978) and Young (1982) have shown that if the control input $u(t)$ is designed such that the tangent vector or time derivative of the switching plane variables always point toward the sliding mode surfaces, then the switching plane variables s_i ($i = 1, \dots, m$) asymptotically converge to zero, and the system states can remain on the sliding mode.

In fact, the condition for the switching plane variables to reach the sliding mode surfaces is a convergence problem. Therefore, the second Lyapunov method can be used to provide a natural setting for the analysis. Generally, the following Lyapunov function is often used in the variable structure controller design

$$V = \frac{1}{2} S^T S \quad (2.6)$$

In this case, the sufficient condition for the switching plane variables to reach the sliding mode surfaces can be expressed as follows

$$S^T \dot{S} < 0 \quad (2.7-a)$$

or

$$s_i \dot{s}_i < 0 \quad (i = 1, \dots, m) \quad (2.7-b)$$

It has been noted that most of the variable structure control algorithms are designed based on the sufficient condition in expression (2.7-a) or expression (2.7-b) (Utkin, 1978 and DeCarlo, 1988).

2.2.2 Equivalent control

On the sliding mode, $s_i = 0$ and $\dot{s}_i = 0$ ($i = 1, \dots, m$). Then, using expressions (2.1) and (2.2), we have

$$\begin{aligned} \dot{S} &= C \dot{X} \\ &= C A X + C B u_{eq} = 0 \end{aligned} \quad (2.8)$$

where u_{eq} is called as equivalent control.

If $|C B| \neq 0$, the equivalent control u_{eq} can be written as

$$\begin{aligned} u_{eq} &= -(C B)^{-1} C A X \\ &= -K X \end{aligned} \quad (2.9-a)$$

$$\text{where } K = (C B)^{-1} C A \quad (2.9-b)$$

The system response on the sliding mode can then be described by the following differential equation

$$\begin{aligned}\dot{X}(t) &= A X(t) - B(CB)^{-1}C A X(t) \\ &= [I - B(CB)^{-1}C] A X(t)\end{aligned}\quad (2.10)$$

System (2.10) is called equivalent system. The characteristics of the equivalent system (2.10) can be summarised as:

(1) The dynamical behaviour of the equivalent system is independent of the control input and depends only on the choice of the matrix C in expression (2.2-c). Therefore, the control input is just used to drive the system states into the sliding mode and thereafter to maintain it on the sliding mode. The determination of the matrix C may thus be completed with no prior knowledge of the form of the control input.

The reason for the equivalent system to have an independent motion from the control input is due to the fact that the matrix CB is nonsingular. In fact, the condition $|CB| \neq 0$ means that the null space of C and the range space B are complementary subspaces. Thus, when the sliding motion occurs on the sliding mode or within $N(C)$, the behaviour of the equivalent system is unaffected by the control input. If $|CB|=0$ as shown in Utkin (1977), the equivalent control is either not unique or does not exist. Therefore, sliding mode can not be reached.

(2) Equivalent system (2.10) is an $(n-m)$ th order system. The work in Darling and Zinober (1986) has shown that for the matrix B with full rank m , there exists an orthogonal $n \times n$ transformation matrix T such that

$$TB = \begin{bmatrix} 0 \\ B_2 \end{bmatrix} \quad (2.11)$$

where B_2 is an $m \times m$ nonsingular matrix.

Define a transformed state variable vector $Y = T X$, state equation (2.1) becomes

$$\dot{Y} = T A T^T Y + T B u \quad (2.12)$$

If Y is partitioned as

$$Y^T = \begin{bmatrix} Y_1^T & Y_2^T \end{bmatrix} \quad (2.13-a)$$

and matrices $T A T^T$ and $C T^T$ are partitioned as

$$T A T^T = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad (2.13-b)$$

$$C T^T = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \quad (2.13-c)$$

then, the system (2.12) can be written in the following form

$$\dot{Y}_1 = A_{11} Y_1 + A_{12} Y_2 \quad (2.14-a)$$

$$\dot{Y}_2 = A_{21} Y_1 + A_{22} Y_2 + B_2 u \quad (2.14-b)$$

On the sliding mode, we have

$$C_1 Y_1 + C_2 Y_2 = 0 \quad (2.15-a)$$

or

$$Y_2 = -F Y_1 \quad (2.15-b)$$

where

$$F = C_2^{-1} C_1 \quad (2.15-c)$$

The equivalent system can then be written in the following form

$$\dot{Y}_1 = (A_{11} - A_{12} F) Y_1 \quad (2.16)$$

Therefore, we can see from expression (2.16) that the equivalent system is (n-m)th order system, i.e., the system dynamics is simplified on the sliding mode.

2.2.3 Robustness property

Robustness property is an important feature of a variable structure control system. Suppose that system (2.1) has uncertainty in matrix A and external disturbance, then the system state equation can be written in the following form

$$\dot{X}(t) = (A_0 + \Delta A) X(t) + Bu + Df \quad (2.17)$$

where A_0 is the nominal system matrix, ΔA is the uncertainty, $f \in R^L$ is a bounded external disturbance vector, and matrix D is compatibly dimensioned. Without loss of generality, it can be assumed that matrices B and D are full rank and the uncertainty presented in the input distribution matrix B is incorporated in the system disturbance term. During the sliding motion, the state vector of the system satisfies the following equations

$$C X = 0 \quad (2.18-a)$$

$$C (A + \Delta A) X + C B u_{eq} + C D f = 0 \quad (2.18-b)$$

From expression (2.18-b), the equivalent control can be achieved as

$$u_{eq} = -(CB)^{-1} C (AX + \Delta AX + Df) \quad (2.19)$$

and the equivalent system equation is then given by

$$\dot{X} = [I - B(CB)^{-1}C](AX + \Delta AX + Df) \quad (2.20-a)$$

$$CX = 0 \quad (2.20-b)$$

Spurgeon (1991) has shown that the sliding mode system (2.20) is insensitive to parameter variations and the external disturbance if and only if the system uncertainty ΔA , matrices B and D satisfy the following rank relation

$$\text{rank} [B : D] = \text{rank} [B : \Delta AT] = \text{rank} [B] \quad (2.21)$$

where T is the matrix of the basis vectors of the reduced-order sliding subspace defined by expression (2.20).

Expression (2.21) is also called as the invariance condition. In addition, the robustness property for variable structure systems have been investigated by other researchers. For example, Gutman (1979) and Bormish and Leitmann (1983) have shown that if system uncertainties and disturbance satisfy the "matching condition", then the system is completely insensitive on the sliding mode, and the effect of disturbance and parameter variations can be minimised by minimising the time required to attain the sliding mode.

2.2.4 Two methods of sliding mode design

Equivalent system equation (2.10) shows that the system behaviour on the sliding mode depends only on the choice of the sliding mode parameter matrix C , and asymptotic stability and desired transient response can be obtained by the suitable design of matrix C . Usually, there are two methods for the sliding mode designs. They are (a) the quadratic minimisation method and (b) the eigenstructure assignment method.

The quadratic minimisation method for the sliding mode design was proposed by Utkin and Young (1978). In this method, the following cost function is defined

$$J(u) = \frac{1}{2} \int_{t_s}^{\infty} X(t)^T Q X(t) dt \quad (2.22)$$

where Q is a symmetric positive-definite matrix, and t_s denotes the time at which the sliding mode starts.

Partitioning the following matrix compatibly with Y (see subsection 2.2.2)

$$T^T Q T = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \quad (2.23)$$

where matrix T is defined in expression (2.11),

the cost function (2.22) can then be expressed in the following form

$$J(v) = \frac{1}{2} \int_{t_s}^{\infty} \{ Y_1^T Q^* Y_1 + v^T Q_{22} v \} dt \quad (2.24)$$

where

$$Q^* = Q_{11} - Q_{12} Q_{22}^{-1} Q_{21} \quad (2.25-a)$$

$$A^* = A_{11} - A_{12} Q_{22}^{-1} Q_{21} \quad (2.25-b)$$

$$v(t) = Y_2 + Q_{22}^{-1} Q_{21} Y_1 \quad (2.25-c)$$

$$\dot{Y}_1 = A^* Y_1 + A_{12} v(t) \quad (2.25-d)$$

Expression (2.24) is the form of standard linear quadratic optimal regulator problem.

By minimising expression (2.24), the optimal control $v(t)$ is given by

$$v(t) = -Q_{22}^{-1}A_{12}^T P Y_1 \quad (2.25-e)$$

Using expression (2.25-e) in expression (2.25-c), we have

$$Y_2 = -Q_{22}^{-1} [Q_{21} + A_{12}^T P] Y_1 = -F Y_1 \quad (2.26)$$

where the matrix P satisfies the following Riccati equation

$$PQ^* + A^*P - PA_{12}Q_{22}^{-1}A_{12}^T P + Q^* = 0 \quad (2.27-a)$$

and matrix

$$F = Q_{22}^{-1} [Q_{21} + A_{12}^T P] \quad (2.27-b)$$

can then be determined as required.

The eigenstructure assignment method is also very popular in the sliding mode designs and it was first used by Utkin and Young in 1978.

Suppose that the sliding mode has commenced on $N(C)$. Then the equivalent system can be written as

$$\dot{X}(t) = (A - BK)X(t) \quad (2.28)$$

where matrix K is given in expression (2.9-b).

During the sliding motion, the state variables must remain in $N(C)$ so that

$$C[A - BK] = 0 \Leftrightarrow R(A - BK) \subseteq N(C) \quad (2.29)$$

Let λ_i ($i = 1, \dots, n$) be the eigenvalues of $A - BK$ with corresponding eigenvectors v_i ,

then, from expression (2.29), we have

$$C[A - BK]v_i = \lambda_i C v_i = 0 \quad (2.30)$$

Expression (2.30) shows that either λ_i is zero or $v_i \in N(C)$. Since $A - BK = A_{eq}$ has m zero-valued eigenvalues, we can set $\{\lambda_i: i = 1, \dots, n-m\}$ be the nonzero eigenvalues and therefore, specifying the corresponding eigenvalues $\{v_i: i = 1, \dots, n-m\}$ fix the null space of C ($\dim[N(C)] = n - m$).

It is noted that C is not uniquely determined because the equation

$$CV = 0, \quad V = [v_1 \dots v_{n-m}] \quad (2.31)$$

has m^2 degree of freedom, which may be easily seen if we define

$$W = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} = Tv \quad (2.32)$$

where the partitioning of W is compatible with that of Y , then expression (2.31) becomes

$$0 = C^T T \cdot Tv = [C_1 \quad C_2] \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} = C_2 [F \quad I_m] \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} \quad (2.33)$$

Therefore, F can be determined from the following equation

$$FW_1 = -W_2 \quad (2.34)$$

The work of Dorling and Zinober (1986) has shown that this approach has the drawback that the eigenvectors of matrix $A-BK$ are not freely assignable, and at most m elements of an eigenvector may be assigned arbitrarily, after which the remaining $n-m$ elements are fully determined by the assigned elements. Thus one approach to eigenvector assignment is to select m elements according to some scheme and accept the remaining elements as determined. This may allow a degree of adjustment to be carried out by inspection.

Some other eigenvector assignment methods have also been proposed, and the details can be found in Moore (1976), Klein and Moore (1977) and Sinswat and Fallside (1977).

2.2.5 Controller designs

In most of the variable structure control schemes, the control law usually consists of a linear component u^L and a nonlinear component u^{NL} which are assumed to form control input u . The linear part is merely a state feedback

$$u^L = KX \quad (2.35)$$

While the nonlinear signal incorporates the discontinuous elements of the control. Some examples of possible types of nonlinearity are as given below.

(a) A nonlinear component with constant gains

$$u_i^{NL} = M_i \text{sgn}(C_i X), \quad M_i > 0 \quad (2.36)$$

(b) A nonlinear component with state-dependent gains

$$u_i^{NL} = m_i(X) \text{sgn}(C_i X) \quad m_i(.) > 0 \quad (2.37)$$

(c) A linear feedback with switching gains

$$u^{NL} = \Psi X \quad (2.38-a)$$

where $\Psi = [\psi_{ij}]$ and

$$\Psi_{ij} = \begin{cases} \alpha_{ij} & s_i x_j < 0 \\ \beta_{ij} & s_i x_j > 0 \end{cases} \quad (2.38-b)$$

(d) A unit vector nonlinearity with scale factor

$$u^{NL} = \frac{NX}{\|MX\|} \quad (2.39)$$

where the null spaces of N , M and C are coincident.

The nonlinear control component is discontinuous on the individual hyperplane in cases (a) - (c). This may result in wasted control effort as the system state pierces one hyperplane, and is forced into another surface. In case (d), the individual controls are continuous, except on the intersection of the switching plane variables where all the nonlinear control elements become discontinuous together. The details of cases (a) - (d) are shown in Utkin (1978), Ryan (1983), Young (1977) and Dorling and Zinober (1983). Some special properties and behaviours of a system with control type (d) has been discussed in Surgeon (1991).

2.3 Variable structure control of nonlinear system

2.3.1 System model

In section (2.2) we have briefly reviewed the basic variable structure control theory of linear systems. Most of these ideas can be extended to the variable structure control of nonlinear systems. However, the complexity of the analysis and the controller designs may be increased due to the nonlinearity in the nonlinear system model. From the engineering point of view, the following nonlinear system is often considered (DeCarlo, et al., 1988).

$$\dot{X}(t) = f(t, X) + B(t, X)u(t) \quad (2.40)$$

where the state vector $X(t) \in \mathbb{R}^n$, the control input vector $u(t) \in \mathbb{R}^m$, $f(t, X) \in \mathbb{R}^n$ and $B(t, X) \in \mathbb{R}^{n \times m}$. Further, each entry in $f(t, X)$ and $B(t, X)$ is assumed to be continuous with continuous bounded derivative with respect to X .

Each entry $u_i(t)$ of the control input vector has the following form

$$u_i(t, X) = \begin{cases} u_i^+(t, X) & \text{with } \sigma_i(X) > 0 \\ u_i^-(t, X) & \text{with } \sigma_i(X) < 0 \end{cases} \quad i = 1 \dots m \quad (2.41)$$

where $\sigma_i(x)$ is the i th switching surface associated with the $(n-m)$ dimensional switching surfaces

$$\sigma(X) = [\sigma_1(X), \dots, \sigma_m(X)]^T \quad (2.42)$$

2.3.2 Sliding mode and equivalent control

Following the sliding mode design for linear systems in section 2.2.4, the method of equivalent control is a way to determine the system motion restricted to the sliding mode $\sigma(X) = 0$. Suppose that there exists a time $t_0 > 0$, and the state of the system reaches the sliding mode after $t \geq t_0$. On the sliding mode, the following two equations are satisfied

$$\sigma(X(t)) = 0 \quad t \geq t_0 \quad (2.43-a)$$

$$\dot{\sigma}(X(t)) = 0 \quad t \geq t_0 \quad (2.43-b)$$

Using system equation (2.40), expression (2.43-b) can be expressed as follows

$$\frac{\partial \sigma(X)}{\partial X} [f(t, X) + B(t, X)u_{eq}] = 0 \quad (2.44)$$

where u_{eq} is the so called equivalent control which can be obtained from expression (2.44) as follows

$$u_{eq} = - \left[\frac{\partial \sigma(X)}{\partial X} B(t, X) \right]^{-1} \frac{\partial \sigma(X)}{\partial X} f(t, X) \quad (2.45)$$

Using expression (2.45) in system model (2.40), the dynamics of the closed loop system on the sliding mode is given by

$$\dot{X} = \left[I - B(t, X) \left(\frac{\partial \sigma(X)}{\partial X} B(t, X) \right)^{-1} \frac{\partial \sigma(X)}{\partial X} \right] f(t, X) \quad (2.46)$$

Therefore, the problem of the sliding mode design is to choose the parameters in $\sigma(X) = 0$ such that the equivalent system (2.46) is stable. In most of variable structure control schemes for nonlinear systems, the linear sliding modes are often used. Therefore, some methods of sliding mode design in sections 2.2.4 can also be used.

2.3.3 Controller design

2.3.3.1 Diagonalisation method

In general, for nonlinear system equation (2.40), the control input is an m dimensional vector and each entry has the structure of the form

$$u_i = \begin{cases} u_i^+(t, X) & \text{for } \sigma_i(X) > 0 \\ u_i^-(t, X) & \text{for } \sigma_i(X) < 0 \end{cases} \quad (2.47)$$

To determine the switched feedback gains in control law (2.47), the following diagonalisation method is often used (DeCarlo et al., 1988).

First, a new control vector is considered in terms of a nonsingular transformation

$$u^*(t) = Q^{-1}(t, X) \left[\frac{\partial \sigma(X)}{\partial X} \right] B(t, X) u(t) \quad (2.48)$$

where $Q^{-1}(t, X) \left(\frac{\partial \sigma(X)}{\partial X} \right) B(t, X)$ is a nonsingular transformation, and $Q(t, X)$ is an arbitrary $m \times m$ diagonal matrix with elements $q_i(t, X)$ ($i = 1, \dots, m$) such that $\inf |q_i(t, X)| > 0$.

Using expression (2.48) in expression (2.40), the system dynamics becomes

$$\dot{X} = f(t, X) + B(t, X) \left[\frac{\partial \sigma(X)}{\partial X} B(t, X) \right]^{-1} Q(t, X) u^*(t) \quad (2.49)$$

If u_i^* is selected such that

$$q_i(t, X) u_i^{*+} < - \nabla \sigma_i(X) f(t, X) \quad \sigma_i(X) > 0 \quad (2.50-a)$$

$$q_i(t, X) u_i^{*-} > - \nabla \sigma_i(X) f(t, X) \quad \sigma_i(X) < 0 \quad (2.50-b)$$

then,

$$\sigma^T(X) \dot{\sigma}(X) < 0 \quad (2.51)$$

Expression (2.51) is the reaching condition for the system states to reach the sliding mode surfaces $\sigma(X) = 0$. On the sliding mode, the desired system dynamics can be obtained. Also, the control input $u(t)$ can be obtained from equation (2.48).

2.3.3.2 Reaching law method

In addition to the above diagonalisation method, the reaching law method (Gao and Cheng, 1989; Gao and Hung, 1993) has been developed and proved to be efficient in controller design. The *reaching law* is a differential equation which specifies the dynamics of a switching function $\sigma(X)$. The differential equation of an asymptotically

stable $\sigma(X)$ is itself a reaching condition, i.e. $\sigma^T(X)\dot{\sigma}(X) < 0$. In addition, by choice of the parameters in the differential equation, the dynamic quality of VSC system in the reaching mode can be controlled. A practical general form of the reaching law is

$$\dot{\sigma} = -Q \operatorname{sgn}(\sigma) - Kh(\sigma) \quad (2.52)$$

where

$$\begin{aligned} Q &= \operatorname{diag}[q_1, \dots, q_m], \quad q_i > 0 \\ \operatorname{sgn}(\sigma) &= [\operatorname{sgn}(\sigma_1), \dots, \operatorname{sgn}(\sigma_m)]^T \\ K &= \operatorname{diag}[k_1, \dots, k_m], \quad k_i > 0 \\ h(\sigma) &= [h_1(\sigma_1), \dots, h_m(\sigma_m)]^T \\ \sigma_i h_i(\sigma_i) &> 0, \quad h_i(0) = 0 \end{aligned}$$

Three practical special cases of (2.52) are given below.

1) Constant rate reaching

$$\dot{\sigma} = Q \operatorname{sgn}(\sigma) \quad (2.53)$$

This law forces the switching variable $\sigma(X)$ to reach the sliding mode surfaces $\sigma(X)=0$ at a constant rate $|\dot{\sigma}_i| = -q_i$. The merit of this reaching law is its simplicity. But, if q_i is too small, the reaching time will be too long. On the other hand, a q_i too large will cause severe chattering. In the following chapters of this thesis, a fuzzy tuning algorithm for q_i will be introduced to optimise the system performance.

2) Constant plus proportional rate reaching

$$\dot{\sigma} = -Q \operatorname{sgn}(\sigma) - K\sigma \quad (2.54)$$

Clearly, by adding the proportional rate term $-K\sigma$, the system state is forced to approach the sliding mode surfaces $\sigma(X)=0$ faster when σ is large. It can be shown that the reaching time for X to move from an initial state X_0 to the sliding surfaces is finite, and is given by

$$T_i = \frac{1}{k_i} \ln \frac{k_i |\sigma_{i0}| + q_i}{q_i}.$$

3) Power rate reaching

$$\dot{\sigma}_i = -k_i |\sigma_i|^\alpha \text{sgn}(\sigma_i), \quad 0 < \alpha < 1, i = 1, \dots, m \quad (2.55)$$

This power rate reaching law increases the reaching speed when the state is far away from the sliding mode surfaces $\sigma(X)=0$, but reduces the rate when the state is near the surfaces. The result is a fast reaching and low chattering reaching mode. Integrating (2.55) from $\sigma_i = \sigma_{i0}$ to $\sigma_i = 0$ yields

$$T_i = \frac{1}{(1-\alpha)k_i} |\sigma_{i0}|^{1-\alpha}, \quad i = 1, \dots, m \quad (2.56)$$

showing that the reaching time is finite.

The control law can be determined by the time derivative of σ and the reaching law (2.52), i.e.,

$$u = - \left[\frac{\partial \sigma}{\partial X} B(t, X) \right]^{-1} \left[\frac{\partial \sigma}{\partial X} f(t, X) + Q \text{sgn}(\sigma) + Kh(\sigma) \right] \quad (2.57)$$

where the matrix $\left[\frac{\partial \sigma}{\partial X} B(t, X) \right]^{-1}$ is nonsingular.

2.3.4 Robust control of nonlinear systems

In practical situations, the system dynamics of a nonlinear system is different from its nominal system model due to parameter uncertainties. To represent parameter uncertainties in the plant, the following state equation is considered (DeCarlo, 1988).

$$\dot{X} = [f(t, X) + \Delta f(t, X, r(t))] + [B(t, X) + \Delta B(t, X, r(t))]u(t) \quad (2.58-a)$$

where $r(t)$ is a vector function of uncertain parameters.

In most of researches (Corless and Leitmann, 1981; Gutman and Palmor, 1982; Peterson, 1985), the plant uncertainties Δf and ΔB are assumed to lie in the image of $B(t, X)$ for all variables t and X (this is called "matching condition"). Then dynamic equation (2.58-a) can be expressed as follows

$$\dot{X} = f(t, X) + B(t, X)u + B(t, X)e(t, X, r, u) \quad (2.58-b)$$

where $e(t, X, r, u)$ represents system uncertainties.

DeCarlo et al. (1988) shows that if $e(t, X, r, u)$ is bounded by a positive function $\rho(t)$

$$\|e(t, X, r, u)\|_2 \leq \rho(t) \quad (2.59)$$

and control input has the following form

$$u = u_{eq} + u_n \quad (2.60-a)$$

where

$$u_{eq} = - \left[\frac{\partial \sigma(X)}{\partial X} B(t, X) \right]^{-1} \left[\frac{\partial \sigma(X)}{\partial t} + \frac{\partial \sigma(X)}{\partial X} f(t, X) \right] \quad (2.60-b)$$

$$u_n = - \frac{B^T(t, X) \nabla_x V(t, X)}{\|B^T(t, X) \nabla_x V(t, X)\|^2} \hat{\rho}(t, X) \quad (2.60-c)$$

$$\hat{\rho}(t, X) = \alpha + \rho(t, X) \quad (2.60-d)$$

$$\nabla_x V(t, X) = \left[\frac{\partial \sigma(t, X)}{\partial X} \right]^T \sigma(t, X) \quad (2.60-e)$$

then system state can reach the sliding mode surfaces $\sigma(X) = 0$, and the desired system dynamics can be obtained by the suitable choice of the sliding mode parameters.

The results discussed in this section forms the foundation of the variable structure control theory for nonlinear systems. Although there are many classes of nonlinear systems, robustness and convergence of variable structure control systems may be established based on the results in this section (Utkin, 1978; Young, 1978 and DeCarlo et al., 1988).

2.4 Application to robot manipulators

A robotic manipulator is a typical nonlinear system. The investigations for the control of robotic manipulators have not only improved the robotic system performance, but also developed many new control techniques which have enhanced the modern control theory. Many control schemes such as feedback control and adaptive control have been developed for robotic manipulators. However, the variable structure control technique is one of the most powerful techniques due to the fact that the variable structure control can deal with systems with large uncertainties, bounded disturbances and nonlinearities. In recent years, the designs of robust variable structure control laws for rigid robotic manipulators that ensure robustness and asymptotic trajectory tracking have been investigated by many researchers. Many robustness and convergence results have been obtained by Young (1978, 1988), Morgan and Ozguner (1985), Slotine and Sastry (1983), Yeung (1988) and Leung et al (1991). In this section, the dynamics of the robotic manipulator and a robust variable structure controller design using the reaching law method (Gao and Hung, 1993) will be presented to highlight the main issues of the control scheme.

2.4.1 Dynamics of robotic manipulators

In the absence of friction and other disturbances, the joint space dynamics of an n-link robotic manipulator can be written via the so-called Euler-Langrange equations as (Spong and Vidyasagar, 1988):

$$\sum_j d_{kj}(q) \ddot{q}_j + \sum_{i,j} f_{ijk}(q) \dot{q}_i \dot{q}_j + \phi_k(q) = \tau_k \quad k = 1, \dots, n \quad (2.61)$$

where d_{kj} are the coefficients of the inertia matrix $D(q)$, $\phi_k(q)$ are the gravitational forces and τ_k are the input torques. The coefficients f_{ijk} of the coriolis and centrifugal terms are defined as

$$f_{ijk} = \frac{1}{2} \left\{ \frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} + \frac{\partial d_{ij}}{\partial q_k} \right\} \quad (2.62)$$

and f_{ijk} are known as Christoffel symbols.

It is common to write expression (2.62) in matrix form as

$$D(q) \ddot{q} + F(q, \dot{q}) \dot{q} + G(q) = \tau \quad (2.63)$$

where the k,j th element of the matrix F is defined as

$$f_{kj} = \sum_{i=1}^n \frac{1}{2} \left(\frac{\partial d_{ki}}{\partial q_j} + \frac{\partial d_{ji}}{\partial q_k} - \frac{\partial d_{ij}}{\partial q_k} \right) \dot{q}_i \quad (2.64)$$

and the component of $G(q)$ is ϕ_k .

Although the equation of motion (2.63) is complex and nonlinear for all but simple robotic manipulators, it has several fundamental properties which can be exploited to facilitate control system design (Ortega and Spong, 1989).

Property 1: The inertia matrix $D(q)$ is symmetric, positive-definite, and both $D(q)$ and $D(q)^{-1}$ are uniformly bounded as a function of q .

Property 2: There is an independent control input for each degree of freedom.

Property 3: The Euler-Lagrange equation for the robotic manipulator is linear in the unknown parameters. All the unknown parameters are constant (e.g. link masses, link lengths, moments of inertia, etc.) and appear as coefficients of known functions of the

generalised coordinates. By defining each coefficient or a linear combination of them as a separate parameter, a linear relationship results so that we may write equation (2.63) as

$$D(q)\ddot{q} + F(q, \dot{q})\dot{q} + G(q) = Y(q, \dot{q}, \ddot{q})\theta = \tau \quad (2.65)$$

where Y is an $n \times r$ matrix of known functions, known as the regressor, and q is a n dimensional vector of unknown parameters as shown in Spong and Vidyasagar (1989).

It can be seen later that the manipulator system (2.63) can also be expressed into the generalised form in expression (2.40). Therefore, the basic variable structure theory can be used to design robust controllers and the structural properties mentioned in the above can then be used to simplify controller designs.

2.4.2 A robust VSC controller design

Consider an n -link manipulator system with perturbations and disturbances described by

$$D(q)\ddot{q} + F(q, \dot{q})\dot{q} + G(q) = \tau + w(q, \dot{q}, p, t) \quad (2.66)$$

where p is an uncertain parameter vector, $w(q, \dot{q}, p, t)$ is the collection of all system perturbations and external disturbances. The state variable is defined as

$$X = [q^T \quad \dot{q}^T]^T \quad (2.67)$$

Then (2.66) can be put into the form

$$\dot{X} = f(t, X) + B(t, X)u + v(X, p, t) \quad (2.68)$$

where

$$u = \tau, \quad v(X, p, t) = \begin{bmatrix} 0 \\ D^{-1}(q)w(X, p, t) \end{bmatrix}$$

$$f(t, X) = \begin{bmatrix} \dot{q} \\ -D^{-1}(q)(F(q, \dot{q})\dot{q} + G(q)) \end{bmatrix}, \quad B(t, X) = \begin{bmatrix} 0 \\ D^{-1}(q) \end{bmatrix}.$$

The arm is to track a desired motion $q^d(t)$. The output tracking error is defined as

$$\epsilon = q^d - q, \quad e = \begin{bmatrix} \epsilon^T & \dot{\epsilon}^T \end{bmatrix}^T \quad (2.69)$$

The sliding mode surfaces are chosen as

$$\sigma(e) = Ce = \begin{bmatrix} \Lambda & I \end{bmatrix} \begin{bmatrix} \epsilon \\ \dot{\epsilon} \end{bmatrix} = \Lambda\epsilon + \dot{\epsilon} \quad (2.70)$$

Adopt the reaching law

$$\dot{\sigma} = -Q \operatorname{sgn}(\sigma) - K\sigma \quad (2.71)$$

Taking the time derivative of (2.70) gives

$$\begin{aligned} \dot{\sigma} &= \Lambda\dot{\epsilon} + \ddot{\epsilon} \\ &= \Lambda\dot{\epsilon} + \ddot{q}^d + D^{-1}(F\dot{q} + G - w - u) \end{aligned} \quad (2.72)$$

Equating (2.71) and (2.72) and solving for the control u yields

$$u = D\{Q \operatorname{sgn}(\sigma) + K\sigma + \Lambda\dot{\epsilon} + \ddot{q}^d\} + F\dot{q} + G - w \quad (2.73)$$

All quantities on the right-hand side of (2.73) are known except the disturbance w , which is unknown. To cope with this problem, define

$$w_1 = D^{-1}w \quad (2.74)$$

and assume that w_1 is bounded by

$$\|w_1\| \leq \bar{w} \quad (2.75)$$

Then replace w in (2.73) by $-D\bar{w} \operatorname{sgn}(\sigma)$ give following control law

$$u = D\{Q \operatorname{sgn}(\sigma) + K\sigma + \Lambda\dot{\epsilon} + \ddot{q}^d\} + F\dot{q} + G + D\bar{w} \operatorname{sgn}(\sigma) \quad (2.76)$$

Substituting (2.76) into (2.72) give

$$\dot{\sigma} = -Q \operatorname{sgn}(\sigma) - K\sigma - w_1 - \bar{w} \operatorname{sgn}(\sigma) \quad (2.77)$$

Thus, the reaching condition $\sigma^T \dot{\sigma} < 0$ is guaranteed.

This robust controller design is based on the bounding function estimation of the system perturbations and external disturbances, and the exact system knowledge is not required. However, the system performance and control quality depend very much on the choice of sliding mode parameter Q and the estimation of bounding function of the unknown parts. Excessive control torques and severe control chattering may be easily caused by over estimation of control parameters.

In order to overcome these shortcomings, in chapter 5 of this thesis, fuzzy logic technology will be used to dynamically adjust the sliding mode parameter Q , and control chattering will be reduced dramatically. In chapter 8, a robust adaptive control scheme for robots is established where neural network has been constructed to adaptively learn the bounding function of the unknown parts of the robot system.

2.5 Concluding remarks

The basic theory of variable structure control systems has been briefly surveyed in this chapter. Because the variable structure theory has many good features, it can be easily used to control linear or nonlinear systems. Although the robustness can be achieved without the exact knowledge of the control system, the system performance and control quality depend very much on the choice of sliding mode parameters and the estimation of bounding functions of the unknown parts. In practice, excessive control input and severe control chattering which may excite unmodelled high-frequency dynamics are highly undesirable. In the following chapters of this thesis, several new and improved robust variable structure control schemes of nonlinear systems will be proposed by combining conventional methods and recently developed

techniques, namely fuzzy logic and neural networks, to improve the system performance and enhance the control quality.

Chapter 3

Fuzzy Logic and Fuzzy Logic Control

3.1 Introduction

Zadeh (1965) published his first paper on a novel way of characterising nonprobabilistic uncertainties, which he called “fuzzy sets”. Fuzzy logic and fuzzy set theory has now evolved into a fruitful area containing various disciplines, such as calculus of fuzzy if-then rules, fuzzy graphs, fuzzy interpolation, fuzzy topology, fuzzy reasoning, fuzzy inference systems, and fuzzy modelling. The applications, which are multi-disciplinary in nature, include automatic control, consumer electronics, signal processing, time-series prediction, information retrieval, database management, computer vision, data classification, decision-making, and so on.

Within the diverse areas of fuzzy logic applications, control is the area to which fuzzy systems are most widely applied today. The genesis of fuzzy logic control was a paper by Chang and Zadeh (1972) outlining the basic approach. An important landmark in this development was the introduction of linguistic variables whose values are linguistic terms rather than numbers (Zadeh, 1973). Fuzzy technology, Zadeh

explained, is a means of computing with words for which the role model is the human ability to reason and make decisions without the use of numbers (Zaheh, 1996). The first implementation of these ideas was described in a paper by Mamdani (1974). The key principle underlying fuzzy logic control is based on a logical model which represents the thinking process that an operator might go through to control the system manually. The viability of fuzzy logic control has been demonstrated through widespread applications (Schwartz et al, 1994). Fuzzy logic control can be considered as one of the intelligent control techniques wherein engineering knowledge is reflected in the controller. It has been found that such controllers have definite advantage over the traditional PID controllers in that they are more robust with respect to structured or unstructured uncertainties. The linguistic characteristics of fuzzy control provide a very good approach to account for the sensor noise, unmodelled dynamics, parameter variations, disturbances and nonlinearities (Lee, 1990).

It is known that the classical PID controllers do not work very well for the case of nonlinear control and, even for linear control they typically have to be redesigned according to a new set of basic system parameters. Adaptive controllers have been proposed for these systems. By means of some tuning scheme, the controller parameters can be continuously adjusted to improve robustness against any changing environment. Auto-tuning (Astrom et al ,1992) or auto-calibration (Voda and Landau, 1995) of controllers for plants with uncertainties is useful although some knowledge of the process is needed. Tuning the controller parameters to achieve better performance can be accomplished by means of fuzzy logic (Tseng & Hwang, 1993). A smooth control signal can be achieved by fuzzy tuning of the discontinuous control component in the sliding mode control (Mei & Man et al, 1998), where control

chattering is an undesirable disadvantage inherent to conventional sliding mode control.

In recent years, fuzzy modelling of controlled systems has received much attention. A number of researchers (Feng and Cao et al, 1997; Tanaka et al, 1996) proposed fuzzy global systems with stability analysis based on Takagi-Sugeno fuzzy inference model. Mei & Man et al (1998) presented a fuzzy modelling and robust tracking control scheme for a class of nonlinear systems by fuzzifying over a number of operating points within the interested range.

Despite a large number of technical publications focusing on fuzzy systems, there exist some doubtful opinions on fuzzy controllers mainly due to some misunderstanding about fuzzy control. Attempts have been made to clear up this misunderstanding, e.g. Jager (1995). It is, therefore, of interest to start the notable part of this thesis, with a short primer on fuzzy set theory, fuzzy reasoning, and fuzzy logic control, using fuzzy tool in combination with other techniques.

The arrangement of this chapter is as follows. Section 3.2 gives the definitions of fuzzy sets, fuzzy set operations, fuzzy relation and compositions, and linguistic representations. Section 3.3 describes fuzzy logic and fuzzy reasoning. Section 3.4 introduces fuzzy logic control.

3.2 Fuzzy set theory

Due to the rapid growth of fuzzy logic literature, it is difficult to present a comprehensive survey of its wide applications. A lot of them are too theoretical to be

applied to engineering problems. The purpose of this section is to briefly summarise the basic concepts that are necessary in understanding fuzzy logic and fuzzy inference from a practical viewpoint.

3.2.1 Fuzzy sets

A *universe of discourse* (or *domain of definition*) U is the set of allowable values for a variable, denoted generically by $\{x\}$ which could be discrete or continuous, where x represents the *generic element of U* .

A *crisp set* A in a universe of discourse U can be defined as $A = \{x | \mu_A(x)\}$, where $\mu_A(x)$ is, in general, a condition by which $x \in A$. If we introduce a zero-one characteristic function such that $A \Rightarrow (\mu_A(x) = 1 \text{ if } x \in A \text{ and } \mu_A(x) = 0 \text{ if } x \notin A)$ then $\mu_A(x)$ is called a membership function for A .

A *fuzzy set* F in a universe of discourse U is characterised by a membership function $\mu_F(x)$ which takes on values in the interval $[0, 1]$. A fuzzy set may be viewed as a generalisation of the concept of an ordinary (i.e. crisp) set whose membership function only takes two values $\{0, 1\}$. A membership function for a fuzzy set F provides *a measure of the degree of similarity* of an element in U to F . A fuzzy set F in U may be represented as a set of ordered pairs of a generic element x and its grade of membership function: $F = \{(x, \mu_F(x)) | x \in U\}$. When U is continuous, F can be written concisely as $F = \int_U \mu_F(x) / x$, where the integral sign denotes the collection of all points $x \in U$ with associated membership function $\mu_F(x)$. When U is discrete, F is represented as $F = \sum_{x_i \in U} \mu_F(x_i) / x_i$, where the summation sign stands for the union of $(x, \mu_F(x))$ pairs. Similarly, the slash “/” in these expressions is only a marker and

does not imply division. Note that in fuzzy sets, an element can reside in more than one set to different degrees of similarity. This cannot occur in a crisp set theory.

The *support* of a fuzzy set F is the crisp set of all points $x \in U$ such that $\mu_F(x) > 0$. The element $x \in U$ at which $\mu_F(x) = 0.5$ is called the *crossover point*. For example, the formal domain for normal weight might be 40 to 100 kg. However, the fuzzy sets HEAVY and LIGHT can have non-zero membership grade at 50 to 90 kg with a crossover point at 70 kg, as shown in Fig. 3.1. A fuzzy set whose support is a single point x in U with $\mu_F(x) = 1$ is referred to as fuzzy *singleton*. In a singleton output space, fuzzy set membership functions are represented as single vertical points. Figure 3.2 illustrates how a variable SPEED can be composed on individual single points.

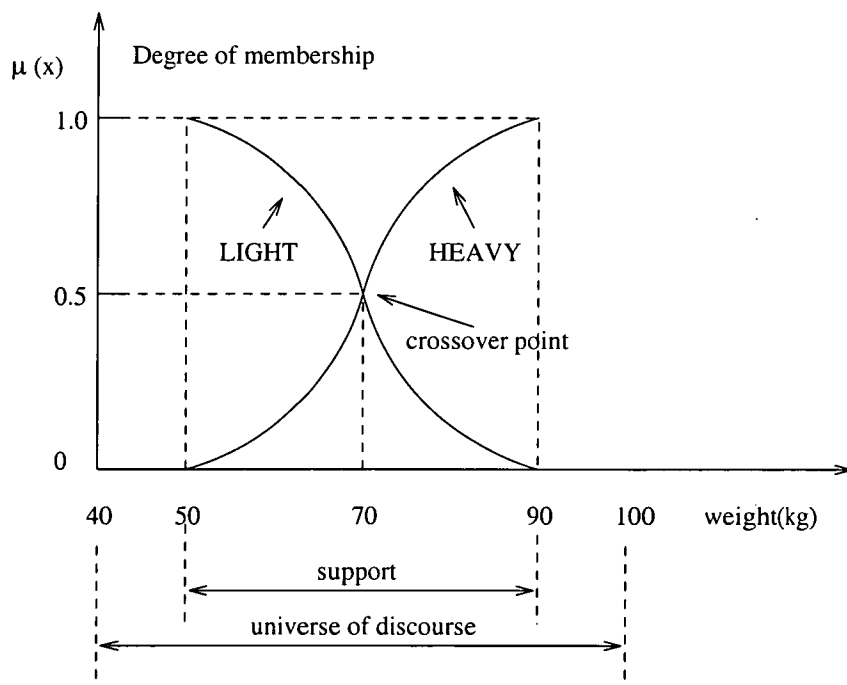


Fig. 3.1 Fuzzy variable "weight"

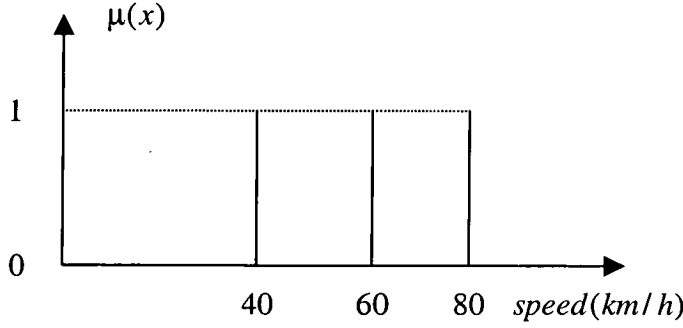


Fig. 3.2 Singleton representation.

3.2.2 Set theoretical operators (Lee, 1990)

Let fuzzy sets A and B in U be described by their membership functions $\mu_A(x)$ and $\mu_B(x)$. The set theoretic operations of union, intersection and complement for fuzzy sets are defined as following.

Union (disjunction): The membership function $\mu_{A \cup B}$ of the union $A \cup B$ (A OR B) is pointwise defined for all $x \in U$ by

$$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\} \quad (3.1)$$

Intersection (conjunction): The membership function $\mu_{A \cap B}$ of the intersection $A \cap B$ (A AND B) is pointwise defined for all $x \in U$ by

$$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\} \quad (3.2)$$

Complement (negation): The membership function $\mu_{\bar{A}}$ of the complement of a fuzzy set A , \bar{A} (NOT A), is pointwise defined for all $x \in U$ by

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (3.3)$$

Fig. 3.3 illustrates these three basic operations: a) illustrates two fuzzy sets A and B , b) is the complement of A , c) is the union of A and B , and d) is the intersection of A and B .

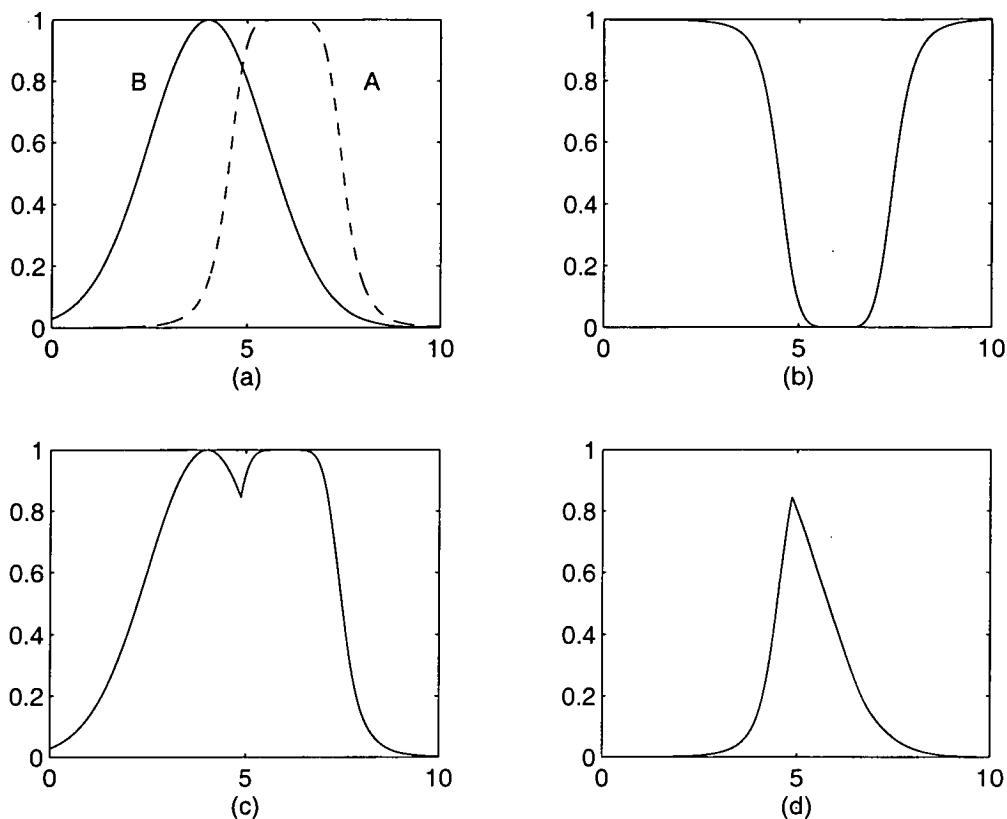


Fig. 3.3 Operations on fuzzy sets: (a) two fuzzy sets A and B ; (b) \bar{A} ; (c) $A \cup B$; (d) $A \cap B$.

Note that other consistent definitions for fuzzy AND and OR have been proposed in the literature under the names “T-norm” and “T-conorm” operators (Mendel, 1995), respectively.

T-norm: a t-norm, denoted by $*$, is a two-place function from $[0, 1] \times [0, 1]$ to $[0, 1]$, which includes fuzzy intersection, algebraic product, bounded product, and drastic product, defined as

$$x * y = \begin{cases} \min\{x, y\}, & \text{fuzzy intersection} \\ xy, & \text{algebraic product} \\ \max\{0, x + y - 1\}, & \text{bounded product} \\ \begin{cases} x \text{ if } y = 1 \\ y \text{ if } x = 1 \end{cases}, & \text{drastic product} \\ 0 \text{ if } x, y < 1 \end{cases}$$

where $x, y \in [0, 1]$.

T-conorm: A t-conorm, denoted by \oplus , is a two-place function from $[0, 1] \times [0, 1]$ to $[0, 1]$, which includes fuzzy union, algebraic sum, bounded sum, and drastic sum, defined as

$$x \oplus y = \begin{cases} \max\{x, y\}, & \text{fuzzy union} \\ x + y - xy, & \text{algebraic sum} \\ \min\{1, x + y\}, & \text{bounded sum} \\ \begin{cases} x \text{ if } y = 0 \\ y \text{ if } x = 0 \end{cases}, & \text{drastic sum} \\ 1 \text{ if } x, y > 0 \end{cases}$$

where $x, y \in [0, 1]$.

3.2.3 The extension principle

The extension principle is a tool for generalising crisp mathematical concepts to fuzzy sets. It has been extensively used in the fuzzy literature.

The extension principle: Let U and V be two universe of discourse and f be a mapping from U to V . For a fuzzy set A in U , the extension principle defines a fuzzy set B in V by

$$\mu_B(y) = \sup_{x \in f^{-1}(y)} [\mu_A(x)] \quad (3.4)$$

That is, $\mu_B(y)$ is the superium of $\mu_A(x)$ for all $x \in U$ such that $f(x)=y$, where $y \in V$ and we assume that $f^{-1}(y)$ is not empty. If $f^{-1}(y)$ is empty for some $y \in V$, define $\mu_B(y)=0$.

3.2.4 Fuzzy relations and their compositions

Cartesian product: If A_1, \dots, A_n are fuzzy sets in U_1, \dots, U_n , respectively, the Cartesian product of A_1, \dots, A_n is a fuzzy set in the product space $U_1 \times \dots \times U_n$ with the membership function

$$\mu_{A_1 \times \dots \times A_n}(x_1, \dots, x_n) = \min\{\mu_{A_1}(x_1), \dots, \mu_{A_n}(x_n)\} \quad (3.5a)$$

or

$$\mu_{A_1 \times \dots \times A_n}(x_1, \dots, x_n) = \mu_{A_1}(x_1) \dots \mu_{A_n}(x_n) \quad (3.5b)$$

Fuzzy relations: Fuzzy relations represent a degree of presence or absence of association, or interconnection between the elements of two or more fuzzy sets. An n -ary fuzzy relation is a fuzzy set in $U_1 \times \dots \times U_n$ and is expressed as

$$R_{U_1 \times \dots \times U_n} = \{((x_1, \dots, x_n), \mu_R((x_1, \dots, x_n)) | (x_1, \dots, x_n) \in U_1 \times \dots \times U_n\}. \quad (3.6)$$

Compositions: Because fuzzy relations are fuzzy sets in the product space, set theoretic and algebraic operations can be defined using operators for fuzzy union, intersection and complement. Let R and S be fuzzy relations in $U \times V$. The intersection and union of R and S , which are *compositions* of the two relations, are defined as

$$\mu_{R \cap S}(x, y) = \mu_R(x, y) * \mu_S(x, y) \quad (3.7a)$$

and

$$\mu_{R \cup S}(x, y) = \mu_R(x, y) \oplus \mu_S(x, y) \quad (3.7b)$$

respectively, where $*$ is any t -norm, and \oplus is any t -conorm.

Next, consider the composition of fuzzy relations from different product space that share a common set. If R and S are two fuzzy relations in $U \times V$ and $V \times W$, respectively, the *sup-star composition* of $R(U \times V)$ and $S(V \times W)$ is a fuzzy relation denoted by $R \circ S(U \times W)$ and is defined by

$$R \circ S = \{(x, z), \sup_{y \in V} [\mu_R(x, y) * \mu_S(y, z)], x \in U, y \in V, z \in W\}, \quad (3.8)$$

where $*$ could be any operator in the class of t -norm (the sup-product is most commonly used). When U , V and W are discrete universe of discourse, the sup operation is the maximum. Fig. 3.4 depicts a block diagram for the sup-star composition.

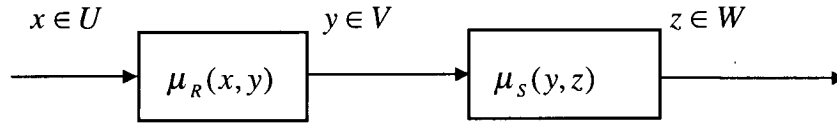
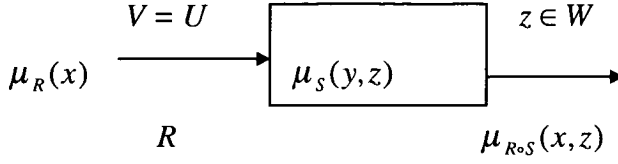


Fig. 3.4. Block diagram interpretation for the sup-star composition.

Suppose fuzzy relation R is just a fuzzy set, then $\mu_R(x, y)$ becomes $\mu_R(x)$ and $V=U$, consequently, $\sup_{y \in V} [\mu_R(x, y) * \mu_S(y, z)] = \sup_{x \in U} [\mu_R(x) * \mu_S(y, z)]$. Thus, when R is just a fuzzy set, the membership function for $R \circ S$ is

$$\mu_{R \circ S}(z) = \sup_{x \in U} [\mu_R(x) * \mu_S(y, z)] \quad (3.9)$$

Fig. 3.5 represents the block diagram interpretation for the sup-star composition when the first relation is just a fuzzy set.

Fig. 3.5. Sup-star composition when $V = U$.

3.2.5 Linguistic representation

A. Linguistic variables

The use of fuzzy sets provides a basis for a systematic way for the management of vague and imprecise concepts. According to Zadeh (1975), in retreating from precision in the face of overpowering complexity, it is natural to explore the use of what might be called *Linguistic variable*, i.e., variables whose values are not numbers but words or sentences in a natural or artificial language. Fuzzy sets can be employed to represent linguistic variable. Consider a real line as a continuous universe of discourse U . A *fuzzy number* F in U is a fuzzy sets which is normal and convex, i.e.,

$$\max_{x \in U} \mu_F(x) = 1 \quad (3.10a)$$

$$\mu_F(\lambda x_1 + (1 - \lambda)x_2) \geq \min(\mu_F(x_1), \mu_F(x_2)), x_1, x_2 \in U, \lambda \in [0, 1]. \quad (3.10b)$$

A linguistic variable can be regarded as a variable whose value is a fuzzy number or whose values are defined in linguistic terms or *labels*. A linguistic variable is characterised by its name u ; the term set of u , i.e., the set of terms $T(u)$ of linguistic values of u with each value being a fuzzy number defined on U ; a syntactic rule for generating the names of values of u ; and a semantic rule for associating with each value its meaning. For example, let SPEED (u) of a car be linguistic variable. It can be decomposed into the following set of terms $T(u) = \{\text{slow, medium, fast}\}$, where each term is characterised by a fuzzy set in the universe of discourse $U = [0 \text{ km/h, } 100$

km/h]. “Slow” speed might be interpreted as “a speed below about 40 km/h”, “medium” as “a speed close to 60 km/h”, and “fast” as “a speed above 80 km/h”. These terms can be characterised as fuzzy sets whose membership functions are shown in Fig.3.6. A vertical line from any measured value of speed intersects at most two membership functions. So, for example, a speed of 45 km/h resides in the fuzzy sets slow and medium to a similarity degree of 0.75 and 0.25, respectively.

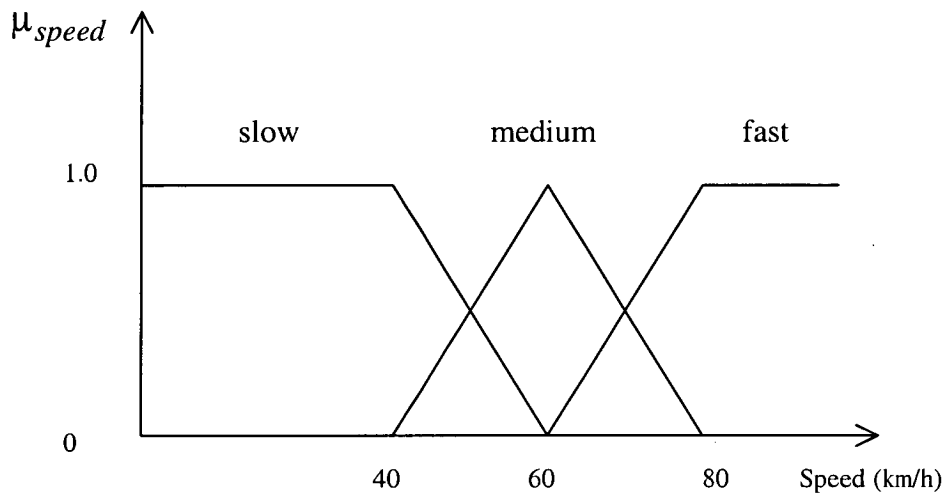


Fig. 3.6 Membership functions of three fuzzy sets, namely, “slow”, “medium”, “fast” for the speed of a car.

B. Hedges: Linguistic modifiers

A *Linguistic hedge* or modifier is an operation that modifies the meaning of a fuzzy set or a term. Hedges can be viewed as operators that act upon a fuzzy set membership function to modify it. Hedges play the same role in fuzzy modelling system as adverbs and adjectives do in a nature language. Accordingly, the number and order of hedges are significant, e.g. *not very large* and *very not large* relates to two different interpretations. A representative collection of commonly-used hedges with their effects is contained in Table 3.1 (Cox, 1994).

Hedge	Meaning
about, around, near, roughly	Approximate a scalar
above, more than	Restrict a fuzzy region
almost, definitely, positively	Contrast intensification
below, less than	Restrict a fuzzy region
vicinity of	Approximate broadly
generally, usually, normally	Contrast diffusion
neighbouring, close to	Approximate narrowly
not, not-so, not really	Negation or complement
quite, rather, somewhat, more-or-less	Dilute a fuzzy region
very, extremely	Intensify a fuzzy region

Table 3.1 Fuzzy linguistic hedge and their approximate meanings.

Power hedges (Zimmermann, 1985): The power hedges operate on grades of membership and represented by the general operator:

$$\mu_{\text{pow}(U)}(x) \equiv [\mu_U(x)]^p, \quad (3.11)$$

where p is a parameter specific to a certain linguistic modifier. The exponent p used in the hedge membership function is quite arbitrary, it can be changed depending on our interpretation of the hedge. If $p=1$, fuzzy set is not modified. Two cases are distinguished:

1) *Concentration*: $p > 1$, fuzzy set is concentrated. Because membership functions are assumed to be normalised, the operation of *concentration* leads to a membership function that lies within the membership function of original fuzzy set; both have the same support, and the same membership values where the value of the original membership function equals unity or zero.

The modifier *very* corresponds to $p=2$. For example, if *slow speed* is fuzzy set with membership function $\mu_s(x)$, then *very slow speed* and *very very slow speed* are fuzzy

sets with membership function $[\mu_s(x)]^2$, $[\mu_s(x)]^4$, respectively. An *artificial hedge* providing milder degree of concentration is the *plus*, whose membership function is $\mu_{plus(U)}(x) \equiv [\mu_U(x)]^{1.25}$.

2) *Dilation*: $p < 1$, fuzzy set is dilated. The operation of *dilation* leads to a membership function that lies outside of the membership function of the original fuzzy set; both have the same support, and the same membership values where the value of the original membership function equals unity or zero.

The modifier *more or less* corresponds to $p = 1/2$. For example, *more or less slow speed* is a fuzzy set with membership function $[\mu_s(x)]^{1/2}$. Another hedge quite useful is the *minus*, whose membership function is $\mu_{minus(U)}(x) \equiv [\mu_U(x)]^{0.75}$.

3.3 Fuzzy logic and fuzzy reasoning

Fuzzy logic is a logical system that is much closer in spirit to human thinking and natural language than traditional logic systems. As a calculus of compatibility, fuzzy logic is quite different from probability which is based on frequency distributions in a random population. Fuzzy logic describes the characteristic properties of continuously varying values by associating partitions of these values with a semantic label (Cox, 1994). Fuzziness is different from probability. The laws of contradiction and excluded middle are broken in fuzzy logic but are not broken in probability. Conditional probability, which must be defined in probability theory, can be derived from first principles using fuzzy logic (Kosko, 1992). Much of the description power of fuzzy logic comes from the fact that these semantic partitions can overlap (see Fig. 3.6) corresponding to the transition from one term to the next.

Fuzzy set serves as the basis for fuzzy logic. Fuzzy set theory starts with some basic concepts coming from crisp set theory, fuzzy logic also begins by borrowing some crisp logic notions. In crisp logic, a *proposition* is an ordinary statement involving terms already defined, and must be meaningful to call it “true” or “false”. A *tautology* is a proposition formed by combining other propositions p, q, r, \dots .

Let us start with the basic primitives of fuzzy logic: fuzzy propositions. A fuzzy proposition represents a statement like “speed (u) is SLOW”, where SLOW is a linguistic label, defined by a fuzzy set on the universe of discourse of speed. Fuzzy propositions connect variables with linguistic labels defined for those variables. As in classical logic, fuzzy propositions can be combined by using logical connectives *and* and *or*, which are implemented by t -norms and t -conorms, respectively.

Fuzzy logic provides a means of modelling human decision making within the conceptual framework of fuzzy logic and approximate reasoning (or fuzzy reasoning). In fact, reasoning with fuzzy logic is based on fuzzy rules which are expressed as logical implications, i.e. in the forms of IF-THEN statements. In fuzzy logic, the definition of a fuzzy implication may be expressed as a fuzzy implication function. Various implication functions can be classified into three main categories: *fuzzy conjunction*, *fuzzy disjunction*, and *fuzzy implication*. A fuzzy rule “IF u is A , THEN v is B ”, is represented by a fuzzy implication and is denoted by $A \rightarrow B$, where A and B are fuzzy sets in the universes U and V with membership functions μ_A and μ_B , respectively. Membership function of the fuzzy implication, denoted by $\mu_{A \rightarrow B}(x, y)$, measures the degree of truth of the implication between $x \in U$ and $y \in V$.

Fuzzy conjunction: The fuzzy conjunction is defined for all $x \in U$ and $y \in V$ by

$$A \rightarrow B = A * B, \quad (3.12)$$

where $*$ is an operator representing a t -norm.

Fuzzy disjunction: The fuzzy disjunction is defined for all $x \in U$ and $y \in V$ by

$$A \rightarrow B = A \oplus B \quad (3.13)$$

where \oplus is an operator representing a t -conorm.

Fuzzy implication: An extension is made by determining the fuzzy versions of some tautologies of $p \rightarrow q$ in crisp logic, e.g. $\sim p \vee q$, $\sim p \vee pq$, and $(\sim p \wedge \sim q) \vee q$. Thus, the fuzzy implication is associated with the corresponding families of fuzzy implication functions:

(1) Material implication:

$$A \rightarrow B = \text{not } A \oplus B \quad (3.14)$$

(2) Propositional calculus:

$$A \rightarrow B = \text{not } A \oplus (A * B) \quad (3.15)$$

(3) Extended propositional calculus:

$$A \rightarrow B = (\text{not } A * \text{not } B) \oplus B \quad (3.16)$$

In traditional propositional logic there are two very important inference rules, *Modus Ponens* (MP) and *Modus Tollens* (MT). Modus ponens is associated with the implication $[A \rightarrow B]$:

Premises	Consequence
Premise 1: x is A Premise 2: If x is A THEN y is B	y is B

In terms of propositions p and q , modus ponens is expressed as $(p \wedge (p \rightarrow q)) \rightarrow q$.

Modus tollens is associated with the implication $[(\text{not } A \rightarrow \text{not } B)]$:

Premises	Consequence
Premise 1: y is B Premise 2: If x is A THEN y is B	x is not A

In terms of proposition p and q , modus tollens is expressed as $(\sim q \wedge (p \rightarrow q)) \rightarrow \sim p$.

In fuzzy logic, modus ponens is extended to *Generalised Modus Ponens* (GMP):

Premises	Consequence
Premise 1: x is A^* Premise 2: If x is A THEN y is B	y is B^*

and modus tollens to *Generalised Modus Tonens* (GMT):

Premises	Consequence
Premise 1: y is B^* Premise 2: If x is A THEN y is B	x is A^*

where A, A^*, B, B^* are fuzzy sets introduced via linguistic variables x, y (Dubois and Prade, 1984). Two more families of fuzzy implication function obtained from GMP and GMT are:

(4) Generalisation of modus ponens:

$$A \rightarrow B = \sup \{c \in [0,1], A * c \leq B\} \quad (3.17)$$

(5) Generalisation of modus tollens:

$$A \rightarrow B = \inf \{t \in [0,1], B \oplus t \leq A\} \quad (3.18)$$

Based on the definition (3.12-3.18), many membership functions for fuzzy implication may be generated by employing appropriate triangular norms and conorms. For example, $\mu_{A \rightarrow B}(x, y)$ can be found from (3.14) as

$$\mu_{A \rightarrow B}(x, y) = \max[1 - \mu_A(x), \mu_B(y)], \quad (3.19a)$$

$$\mu_{A \rightarrow B}(x, y) = 1 - \min[\mu_A(x), 1 - \mu_B(y)], \quad (3.19b)$$

or

$$\mu_{A \rightarrow B}(x, y) = 1 - \mu_A(x)(1 - \mu_B(y)) \quad (3.19c)$$

with appropriate operators for the t -norm $*$ and t -conorm \oplus . From the fuzzy conjunction (3.12), membership function for *minimum implication*, proposed by Mamdani (1974):

$$\mu_{A \rightarrow B}(x, y) \equiv \min[\mu_A(x), \mu_B(y)] \quad (3.20)$$

and *product implication*, introduced by Larsen (1980):

$$\mu_{A \rightarrow B}(x, y) \equiv \mu_A(x) \mu_B(y) \quad (3.21)$$

can be obtained using minimum and algebraic product, respectively. It can be shown that membership functions (3.19a-c) agree with the accepted propositional logic definition of implication, demonstrated in Table 3.2 while neither Mamdani membership function (3.20) nor Larsen one (3.21) does. As shown in Table 3.2, minimum and product inferences preserve *cause and effect*, i.e. these implications are fired only when the antecedent and the consequent are both true.

$\mu_p(x)$	$\mu_q(x)$	$\mu_{p \rightarrow q}(x, y)$	$\min[\mu_A(x), \mu_B(y)]$	$\mu_A(x) \mu_B(y)$
1	1	1	1	1
1	0	1	0	0
0	1	0	0	0
0	0	1	0	0

Table 4.2 Implication with propositional logic, and min and product inference.

This is why minimum and product inferences are the most widely used inferences in engineering applications of fuzzy logic, and referred to as *engineering implications* (Mendel, 1995).

A comprehensive comparison between the available fuzzy implication rules is given in Lee (1990), based on some intuitive criteria for choosing a fuzzy implication that is as

close as possible to the input truth value function value. Although the minimum operation rule (3.20) and the product operation rule (3.21) of fuzzy implication do not have a well-defined logic structure, it is shown (Lee, 1990) that they are well suited for approximate reasoning, especially for the generalised modus ponens. The GMP, reducing to modus ponens when $A^* = A$, $B^* = B$, is closely related to the forward data-driven inference, particularly used in control system. The GMT, reducing to modus tollens when $A^* = \text{not } A$, $B^* = \text{not } B$, is closely related to the backward goal-driven inference, commonly used in expert systems.

Fig. 3.7 illustratively interprets that GMP is a fuzzy composition where the first fuzzy relation is merely a fuzzy set, A^* (see Figure 3.5). Consequently, $\mu_{B^*}(y)$ can be obtained from the sup-star composition (3.9):

$$\mu_{B^*}(y) = \sup_{x \in A^*} [\mu_{A^*}(x) * \mu_{A \rightarrow B}(x, y)] \quad (3.22)$$

where max-min or max-product formula can be stated for modus ponens. Note that in crisp logic, a rule is fired only if the first premise is exactly the same as the antecedent of the rule, and the result of such rule-firing is the rule's actual consequent. On the other hand, a fuzzy rule is fired so long as there is a non zero degree of similarity between the first premise and the antecedent of the rule, and, the result of such rule-firing is a consequent that has a non zero degree of similarity to the rule's consequent.

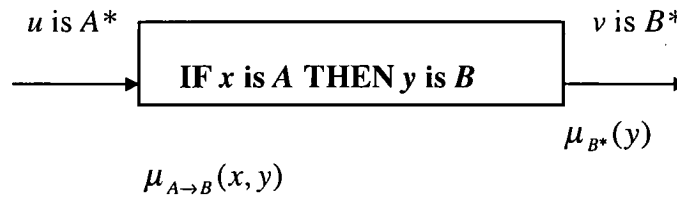


Fig. 3.7 Sup-star composition when $V = U$

3.4 Fuzzy logic control

Fuzzy logic control emerged in the 1970s following the work of E.H. Mamdani and colleagues in England, who developed the first fuzzy rule-based control system implemented on a laboratory-scale steam engine. The paradigmatic solution developed by Mamdani (1974) led to many successful applications in a broad range of areas, ranging from consumer products to industrial process control to automotive engineering. Using fuzzy logic as an effective means of capturing approximate, inexact nature of the real world, a fuzzy logic controller provides an algorithm which can convert the linguistic control strategy based on expert knowledge into an automatic control strategy. In particular, the methodology of fuzzy logic control has been proven through recent successful applications (Schwartz *et al.*, 1994) to be superior to conventional control when the processes are too complex, or when the available sources of information are interpreted imprecisely, or uncertainly. In this sense, fuzzy logic control may be viewed as a rapprochement between conventional control and human-like decision making techniques for improving robustness of control systems. This section gives a brief summary of the prolific literature on fuzzy logic control.

Fuzzy inference is the actual process of mapping from a given input to an output using fuzzy logic. A fuzzy logic controller (FLC) is a fuzzy inference system which maps an input data vector into a scalar control output. Fig. 3.8 depicts a fuzzy control system, where the elements of the fuzzy logic controller are rules, fuzzifier, inference engine, and defuzzifier.

Once the fuzzy rules have been chosen, an FLC can be considered as a nonlinear mapping from inputs (error e , change of error Δe) to outputs (control signal u). This mapping can be expressed quantitatively as $u=f(e, \Delta e)$. The design of an FLC is how to obtain explicitly this relation so that the control requirements are satisfied.

Rules may be provided by experts or can be extracted from numerical data. In either cases, engineering heuristic rules are expressed as a collection of IF-THEN statements. Each rule consists of linguistic variables, linguistic terms, logic connections, and implications. Each linguistic term is fuzzy set associated with its fuzzy membership function.

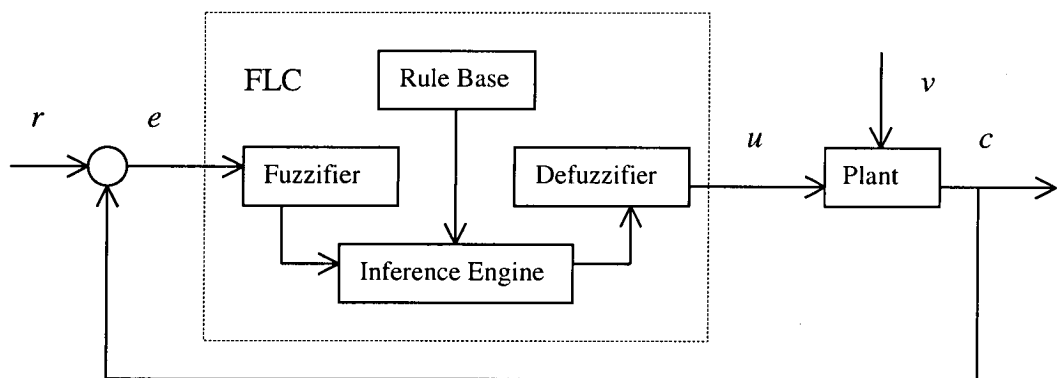


Fig. 3.8 Block diagram of a fuzzy logic control system.

The *fuzzifier* maps crisp number into fuzzy sets of a linguistic variable. The *inference engine* of an FLC maps fuzzy sets into fuzzy sets in the way fuzzy rules are combined. There are many different fuzzy logic inference procedures but only a few of them are

commonly used in engineering applications of fuzzy logic. The *defuzzifier* maps output sets into crisp numbers corresponding to the control action u .

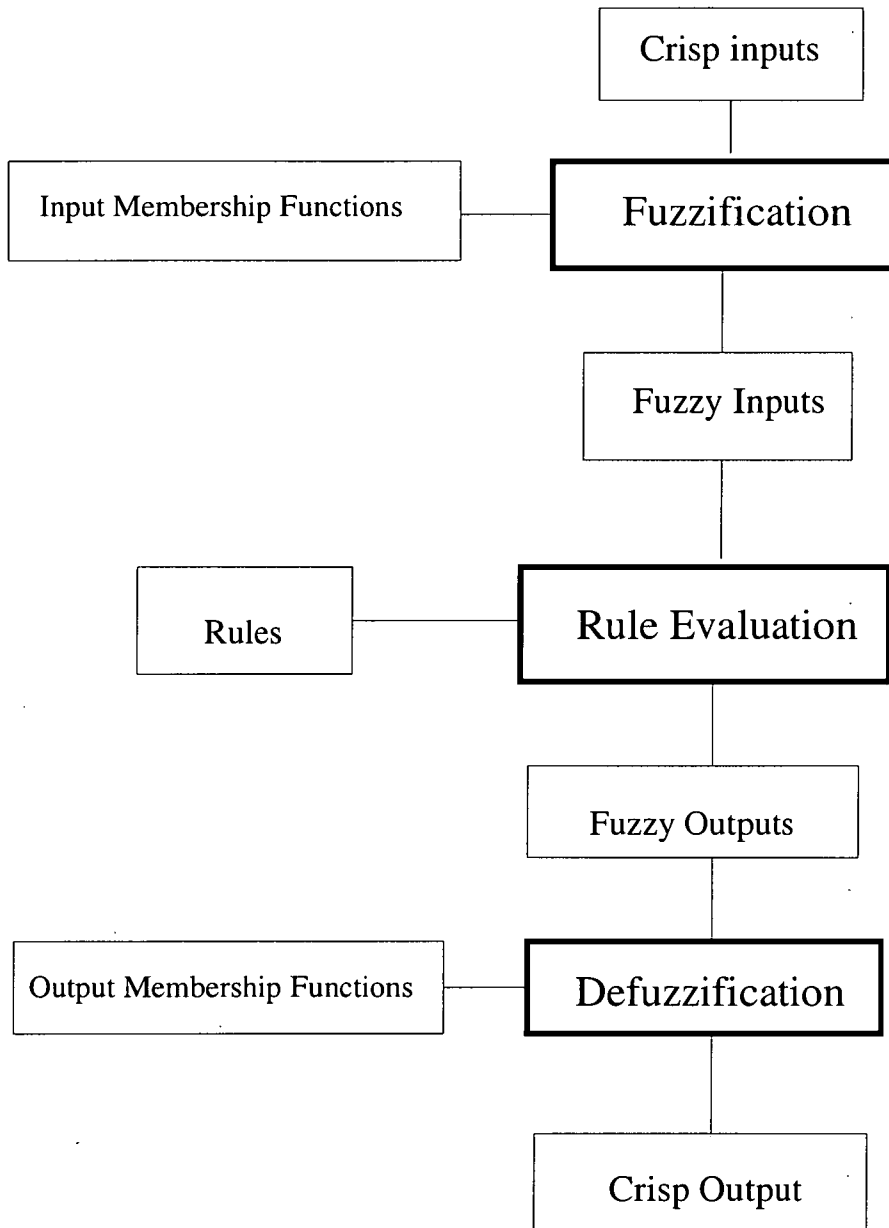


Fig. 3.9 Fuzzy inference process

A fuzzy logic controller can be viewed as a system employing a fuzzy inference process to produce crisp control output from fuzzy rules for given input values. The

inference process is the internal mechanism involving three steps: fuzzification, rule evaluation, and defuzzification, as shown in Fig. 3.9.

A. Fuzzification

In general, fuzzification is the process of translating real world values of an input variable (input values) to the grades of its input label (input grades). Fuzzification involves measuring the values of input variables, performing a scale mapping to transfer the range of values of input variables into corresponding universes of discourse, and converting input data into suitable linguistic values by means of labels (terms of a linguistic variable) of fuzzy sets. The input variables of an FLC, which can be a voltage, current, speed, or position in motion control systems, are usually the control error, e , or its change, Δe . Symbolically, the process of transforming crisp data into fuzzy sets can be represented by

$$x = \text{fuzzifier}(x_0) \quad (3.23)$$

where the x_0 is a crisp input value (e or Δe), x is a fuzzy set, and fuzzifier is a fuzzification operator. In order to develop useful rules, appropriate membership functions associated with each label should be well chosen. Therefore, it is necessary to have a good intuitive feel for the nature of the variables. The defining membership functions for all variables would be based on designer's intuitive assumptions, just as the rules themselves are. The fuzzification process also provides necessary conditions for defining linguistic control rules and fuzzy data manipulation in an FLC, including the discretisation/normalisation of the universe of discourse, fuzzy partition of the

input and output space, and choice of the appropriate membership function of a primary fuzzy set.

Discretisation of a universe of discourse is the process of quantising it into a certain number of segments (quantisation levels) with corresponding labels and thus, forming a discrete universe. A fuzzy set is then defined by assigning grade of membership values to each label. A scale mapping is needed to transform measured variables into values in the discretised universe. The choice of quantisation levels determines how fine a control can be obtained. Normalisation of a universe requires discretising it into a finite number of segments. Each segment is mapped into a suitable segment of the normalised universe which normally is the closed interval $[0, 1]$ or $[-1, 1]$.

In general, a linguistic variable is associated with a term set. Each term in the term set is defined on the same universe of discourse. A fuzzy partition, then, determines the number of terms in a term set. This number affects the granularity of the fuzzy control. Fig. 3.10 shows a fuzzy partition in the normalised universe $[-1, 1]$ with seven terms: NB, NM, NS, ZE, PS, PM, AND PB, which means negative big, negative medium, negative small, zero, positive small, positive medium, and positive big, respectively.

A membership function (MF), which defines how each point in the input space of a label is mapped to a degree of membership between 0 and 1, can principally take any nonlinear or piece-wise linear form. The latter is either a triangular or trapezoidal functions as shown, for example, in Fig. 3.6 and 3.10. They are the simplest and most commonly-used in engineering applications. The vertex of a triangular fuzzy label

usually corresponds to the mean value of a data set, while the base is twice the standard deviation of the data set.

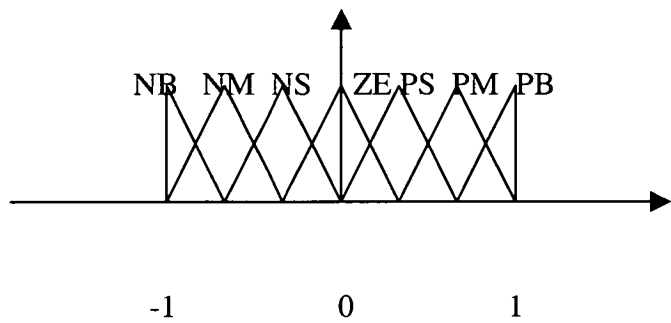


Fig. 3.10 A fuzzy partition representation

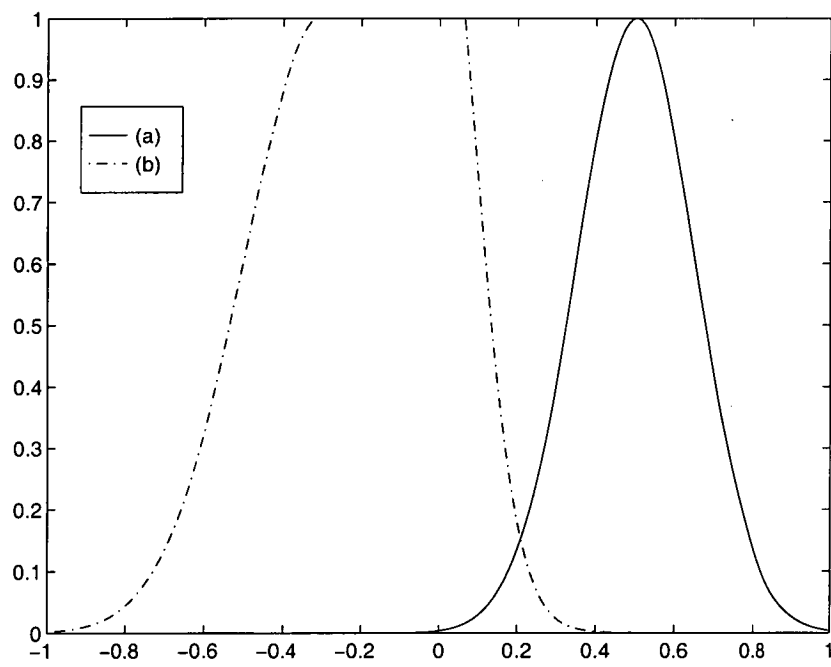


Fig. 3.11 Gaussian MFs: (a) $\sigma = 0.15, c = 0.5$;
(b) $\sigma_1 = 0.15, c_1 = 0.5, \sigma_1 = 0.15, c_1 = 0.5$

Crisp value can be converted into a fuzzy singleton (see Fig. 3.2) within a certain universe of discourse. This kind of membership function is widely used in fuzzy control applications because it is natural and easy to implement. The other commonly-used functions, with universe of discourse scaled to $[-1, 1]$, are described below. The Gaussian MF, determined by

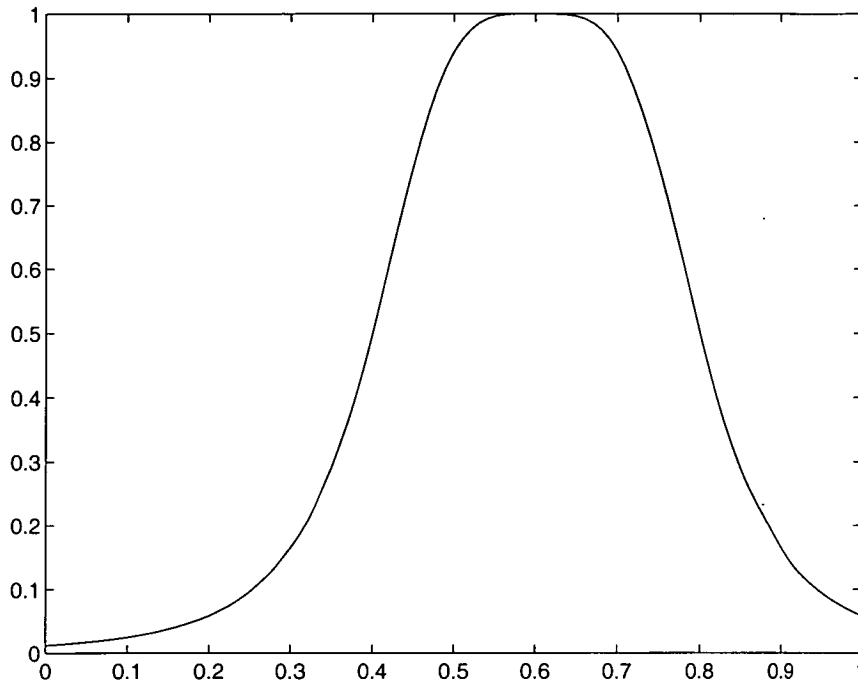
$$\mu(x, \sigma, c) = \exp\left(-\frac{(x - c)^2}{2\sigma^2}\right) \quad (3.24)$$

where σ and c are two parameters, has the forms of Fig. 3.11(a). A combination of two Gaussian membership functions gives the two-sided form of Fig.3.11 (b).

The generalised bell-shape MF with the form of Fig.3.12 is described by

$$\mu(x, a, b, c) = \frac{1}{1 + \left|\frac{x - c}{a}\right|^{2b}}, \quad (3.25)$$

where $a > 0$, $b > 0$, and $c > 0$. Both Gaussian and bell-shape membership functions have the advantage of being smooth and nonzero at all points, however they cannot specify asymmetry.

Fig.3.12 Bell MF: $a=0.2$, $b=2$, $c=0.6$

The sigmoid can be used to specify asymmetric membership functions, which are important in certain applications. Fig. 3.13 (a) plots the form of an open right sigmoidal MF defined by

$$\mu(x,a,c) = \frac{1}{1 + \exp(-a(x-c))}, \quad (3.26)$$

where $a > 0$ and $c > 0$. An asymmetric and closed (i.e. not open to the left or right) M can be synthesised by the difference of two sigmoidal functions as represented in Fig. 3.13(b).

For fuzzy control applications, triangular and trapezoidal functions seem to perform as well in practice as those based on some other forms of membership functions, and, in many case, are simpler to implement. It would be an interesting psychological

problem to investigate whether or not triangular membership functions are as accurate as some other forms in expressing intuitive concepts of fuzzy numbers (Schwartz et al., 1994).

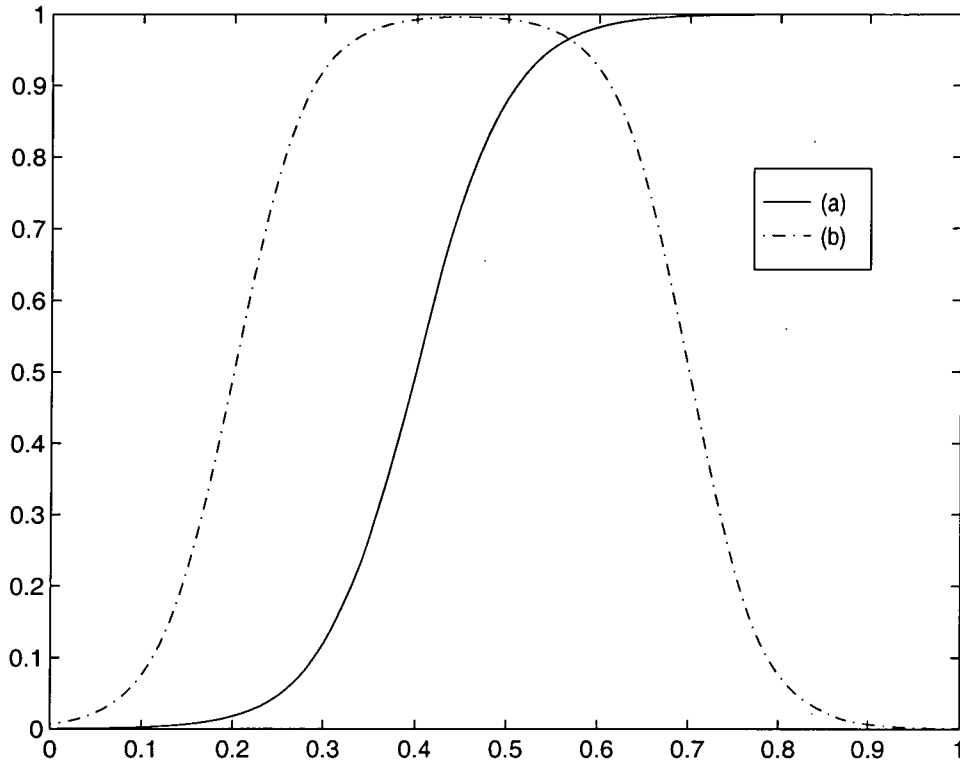


Fig.3.13 Sigmoidal MFs: (a) $a=20$, $c=0.4$; (b) $a_1=a_2=25$, $c_1=0.4$, $c_2=0.7$

B. Rule evaluation

Rule evaluation is the process of obtaining the grades of output labels (output grades) from input grades. The rule evaluation process is mainly important in the design of an FLC. It involves a fuzzy rule base and an inference unit. The rule base characterises the control goals and control strategy of the domain experts by means of a set of linguistic control rules. As the kernel of an FLC, the inference unit is capable of simulating human decision making based on fuzzy concepts and of inferring fuzzy

control actions employing fuzzy implication and rules of inference in fuzzy logic. Based on expert knowledge and fuzzy reasoning, the rule base of an FLC can be formatted in *fuzzy conditional statements* as follows:

$$R_i : \text{IF (set of conditions) THEN (set of consequences), } (i=1,2, \dots, n) \quad (3.27)$$

where the antecedents and the consequents of these rules are associated with linguistic values (labels). A *Fuzzy control rule* is a fuzzy conditional statement in which the antecedent is a condition determined in an application domain and the consequent is a control action for the plant. A rule base is normally formed by a set of fuzzy control rules combined by using the sentence connectives *and* and *also*. In most FLCs, the sentence connective *and* is usually implemented as a fuzzy conjunction in a Cartesian product space, in which the underlying variables take values in different universes of discourse. When the ordering of the fuzzy rules are immaterial, the sentence connective *also* is used with the associated properties of commutativity and associativity. From a practical point of view, the union operator is normally used for the connective *also* for constructing fuzzy models (Lee, 1990). The overall system behaviour can then be characterised by a single fuzzy relation R which is the combination of the fuzzy relations R_i . Symbolically,

$$R = \text{also}(R_1, R_2, \dots, R_i, \dots, R_n), \quad (3.28)$$

where *also* represents a sentence connective.

In general, there are four modes of deriving fuzzy control rules: from expert experience and control engineering knowledge, from human operator's control actions, from the fuzzy model of a process, and from learning. A fuzzy control rule normally has the form of a fuzzy conditional statement relating the state variables in the antecedent and the control action in the consequent. Most of FLCs employ engineering knowledge and experience which are expressed in the language of fuzzy IF-THEN rules based on an introspective verbalisation of human expertise or an interrogation of experienced experts or operators. Many industrial control systems can be empirically controlled by skilled human operators without any quantitative models in mind. As pointed out by Sugeno (1985), in order to automate such processes, it is expedient to express the operator's control rules as fuzzy IF-THEN rules using linguistic variables. Fuzzy control rules can also be generated based on the linguistic description of the dynamic characteristics of the plant known as fuzzy model. Although this approach is somewhat more complicated, it yields better performance and reliability, and provides a more attractable structure for dealing theoretically with the FLC (see, e.g., Cao et al., 1994). Focusing on the ability to create fuzzy control rules and to modify them based on experience (learning), Procyk & Mamdani (1979) described the first self-organising controller with a hierarchical structure of two rule bases: the general rule base and the "meta-rules". The latter exhibit human-like learning ability to create and modify the general rule base according to the desired overall performance of the system. Self-organising control (Shao, 1988) or fuzzy learning systems with model reference (Kwong *et al.*, 1995) and neuro-fuzzy approach (Jang & Sun, 1995) have become increasingly interesting subjects for fuzzy control studies.

C. Defuzzification

The defuzzification process is the final stage in a fuzzy reasoning system involving the selection of a representative element based on the output fuzzy subset. Defuzzification is a mapping from a space of fuzzy control actions defined over an output universe of discourse into a space of real world (crisp) control actions. A defuzzification strategy is aimed at producing a non-fuzzy control action. Many strategies have been proposed in the literature; however their scientific bases are so inadequate that defuzzification is considered as an art rather than a science (Mendel, 1995).

The most frequently used defuzzification strategy is the *Centroid of Area*, which is defined as

$$y^* = \frac{\int_S y \mu_B(y) dy}{\int_S \mu_B(y) dy}, \quad (3.29a)$$

where $\mu_B(y)$ is the aggregated output membership function, and S denotes the support of $\mu_B(y)$. This formula is reminiscent of the calculation of expected values in probability distributions. For a discretised universe, summations replace integrations in (3.29a):

$$y^* = \frac{\sum_{j=1}^n \mu_B(y_j) y_j}{\sum_{j=1}^n \mu_B(y_j)}, \quad (3.29b)$$

where n is the number of quantisation levels of the output. Other defuzzification strategy arise for specific applications, which includes bisector of area, mean of maximum, largest of maximum, and smallest of maximum, and so on.

The method using singletons(single vertical points) instead of fuzzy set membership functions to describe output labels is referred to as *Takagi-Sugeno fuzzy model* while the inference method discussed thus far using fuzzy sets to describe the output labels is called *Mamdani fuzzy model*. In Takagi-Sugeno model, the aggregation step is no longer necessary and the centroid of area defuzzification method is slightly simplified. The centroid is calculated without a numerical integration of the entire fuzzy set surface as in (3.29a). Instead, the domain value at each singleton is multiplied by the height (truth membership value) of the singleton. This technique, in general, provides a centroid equivalent to the centroid found by taking the area across an entire fuzzy set. Because it is more compact and computationally efficient in representation than the Mamdani model, the Tagagi-Sugeno fuzzy model has widely been applied to fuzzy modelling and control of engineering systems (Feng et al., 1997; Mei and Man et al., 1998).

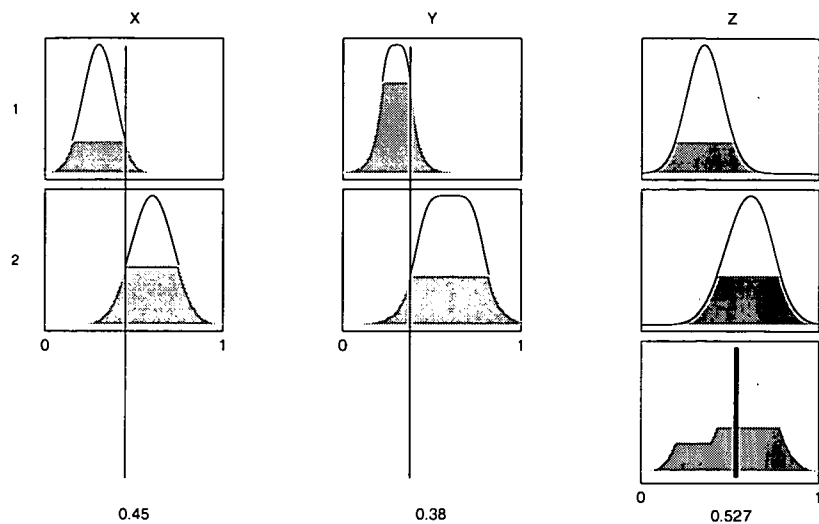


Fig. 3.14 The Mamdani fuzzy inference system using min and max for fuzzy AND and OR operators, respectively.

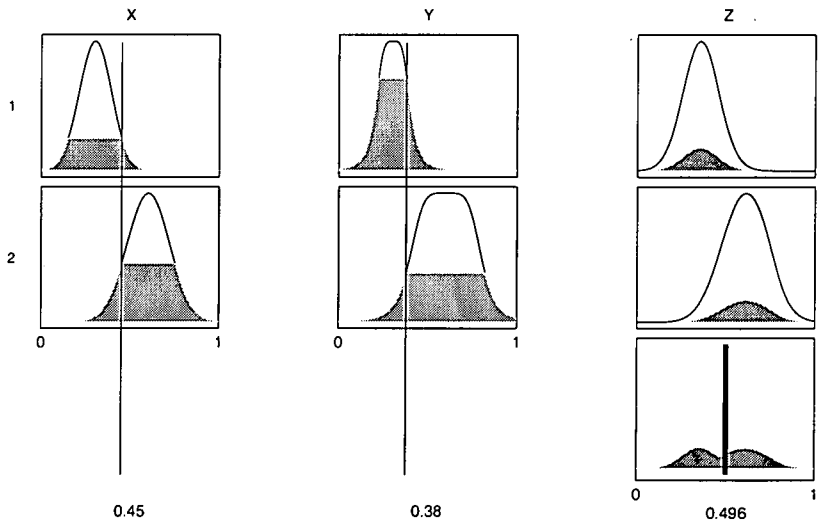


Fig. 3.15 The Mamdani fuzzy inference system using product and max for fuzzy AND and OR operators, respectively.

Fig. 3.14 and 3.15 show the Mamdani fuzzy inference process with centroid of area defuzzification method.

3.5 Concluding remarks

Fuzzy thinking has originated from a philosophical basis on indeterminism (McNeil and Freiburger, 1993). An inherent uncertainty and unpredictability, spontaneity and freedom, integrity and openness characterise the stunning complexity of our world and life. That is why the way fuzzy thinking has emerged in order to grasp real world complexity. As an introduction, this chapter provides an overview of fuzzy logic and fuzzy logic control. Fuzzy sets, fuzzy set operations, and fuzzy linguistic representation such as linguistic variables and linguistic modifiers (hedged) have been briefly outlined. Fuzzy reasoning or approximate reasoning is considered from the engineering viewpoint with IF-THEN fuzzy implications using Mamdani's minimum inference and Larsen's product inference. A fuzzy logic controller, mapping an input data vector into a scalar control output, normally comprises a rule base, fuzzifier and defuzzifier. Commonly-used kinds of membership functions are described, focusing on control applications. The prominent advantage of the fuzzy logic controller is that it can effectively control complex ill-defined systems having nonlinearities, parameter variations and disturbances. However, there also exist some impediments in the design of the fuzzy logic controller. In general, fuzzy rules are obtained on the basis of intuition and experience, and membership functions are selected by trial and error procedure. Moreover, it is not easy to mathematically prove the system stability and robustness due to linguistic expression of the fuzzy rules. Therefore, a systematic design method of the fuzzy logic controller from which the stability and robustness

can be clearly seen is to be explored (Lee, 1990). The following chapters will employ fuzzy logic to establish a mathematical model for a class of nonlinear systems and also enhance robustness and control quality of variable structure control systems.

Chapter 4

Fuzzy Modelling and Robust Tracking Control

4.1 Introduction

In the design of modern and classical control systems, the first step is to establish a suitable mathematical model to describe the behaviour of the controlled plant. However, in practical situations, such a requirement is not feasible because the controlled systems have high nonlinearities and uncertain dynamics, and simple linear or nonlinear differential equations cannot sufficiently represent the corresponding practical systems, and therefore, the designed controller based on such a model cannot guarantee the good performance such as stability and robustness. During the last few years, fuzzy logic control has been suggested as an alternative way to conventional control techniques for complex nonlinear systems due to the fact that fuzzy logic combines human heuristic reasoning and expert experience to approximate a certain desired behaviour function (see e.g. Takagi and Sugeno, 1985; Cao et al., 1996; Wang et al., 1996). However, the asymptotic error convergence and stability of the closed-

loop system may not be obtained due to the approximation error and uncertainties of the fuzzy model.

Many kinds of fuzzy models for control processes have been developed since Mamdani's (1974) paper was published. They can be classified into three kinds of models, Composition Rule of Inference (Zaheh, 1973), Approximate Reasoning Model (Nakanishi et al., 1993), Sugeno's Models such as Position type Model and Position-Gradient type Model (Sugeno and Yasukawa, 1993). Most of these models are expressed by a set of fuzzy linguistic propositions which are derived from the experience of skilled operators or by fuzzy implication which locally represent linear input-output relations of the system.

Most proposed conventional fuzzy models only consider the external behaviour of the system, and can be considered as a function approximation. It is very difficult to obtain a controller using those models. Even if the controller can be obtained by using some trial and error procedures the behaviour of the closed-loop system, for example, the stability of the system is still difficult to analyse. Also the number of rules increase very quickly when the system becomes complex because every local rule is only described by a constant. Therefore, the identification of these fuzzy models is still a difficult problem because there are too many parameters in the membership functions.

In order to overcome the disadvantages of conventional fuzzy models, we developed a fuzzy tracking dynamic model for nonlinear systems by linearising the system over a number of operating points and aggregating the linearised subsystems by fuzzy rules. It is shown that by choosing membership functions appropriately, the number of fuzzy

sets can be small but without loss of good tracking performance. The proposed fuzzy modelling and tracking control methodology for complex nonlinear systems is unique by combining the merits of fuzzy logic and conventional linear control theory. Here, fuzzy logic is used to formulate a system model by aggregating a set of linearised local subsystems which identify the nonlinear system approximately, and a fuzzy feedback controller is designed by use of conventional linear feedback theory and fuzzy reasoning.

The organisation of this chapter is as follows. In Section 4.2, a linearisation method for a class of nonlinear systems by Taylor's expansion is given. In Section 4.3, fuzzy modelling and tracking controller design for nonlinear systems are presented. A fuzzy system model is obtained by aggregating the linearised subsystems with fuzzy rules. In Section 4.4, a robust tracking control scheme is presented by using variable structure control theory based on the fuzzy nominal model. A simulation example using one-link rigid robotic manipulator is given in support of the proposed control scheme. Section 4.5 gives concluding remarks.

4.2 Linearisation of nonlinear systems

Consider the following nonlinear dynamic system

$$\dot{x}(t) = f(x, u) \quad (4.1a)$$

where $x = [x_1, x_2, \dots, x_n]^T$ is the state vector, $f(x, u)$ is a vector whose elements are nonlinear function, and $u = [u_1, u_2, \dots, u_m]^T$ is the control input vector. The control objective is to force the plant state vector x to follow a specified desired trajectory, x_d . Assuming $f(x, u)$ is differentiable with respect to x and u respectively, then

equation (4.1a) can be linearised at some point (x_i, u_i) by Taylor's expansion as follows

$$\begin{aligned}\dot{x} &= \dot{x}_i + \left. \frac{\partial f}{\partial x} \right|_i \tilde{x} + \left. \frac{\partial f}{\partial u} \right|_i \tilde{u} \\ \dot{x}_i &= f(x_i, u_i)\end{aligned}\tag{4.2}$$

where

$$\tilde{x} = x - x_i, \quad \tilde{u} = u - u_i$$

and u_i can be obtained from the following equilibrium condition (Palm and Rehfusess, 1997)

$$\dot{x}_i = 0$$

From expression (4.2), the following local linearised error dynamic equation can be obtained

$$\dot{\tilde{x}} = A_i \tilde{x} + B_i \tilde{u}\tag{4.3}$$

where

$$A_i = \left. \frac{\partial f}{\partial x} \right|_i \in R^{n \times n}, B_i = \left. \frac{\partial f}{\partial u} \right|_i \in R^{n \times m}\tag{4.4a}$$

Remark 4.1 If the controlled system is a single-input and single-output system, i.e., ,

$$x^{(n)}(t) = f(x) + b(x)u\tag{4.1b}$$

where

$$x = [x_1, x_2, \dots, x_n]^T = [x, \dot{x}, \dots, x^{(n-1)}]^T$$

then we have the following linearised system matrices,

$$A_i = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ a_0 & a_1 & \cdots & \cdots & a_{n-1} \end{bmatrix}, B_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b(x_i) \end{bmatrix} \quad (4.4b)$$

$$a_0 = \left. \frac{\partial f}{\partial x} \right|_{x_i} + u_i \left. \frac{\partial b}{\partial x} \right|_{x_i}, a_1 = \left. \frac{\partial f}{\partial \dot{x}} \right|_{x_i} + u_i \left. \frac{\partial b}{\partial \dot{x}} \right|_{x_i}, \dots$$

which is in a controllable form.

Remark 4.2: If equation (4.3) is in a controllable form, the feedback control law

$$\tilde{u} = -K_i \tilde{x} \quad (4.5)$$

can be designed by using conventional linear system theory (Ogata, 1990) so that the eigenvalues of $(A_i - B_i K_i)$ are the specified ones. The feedback gain K_i can be designed by using the Ackerman's formula in the case of single-input system as follows (Ogata, 1990)

$$K_i = [0, \dots, 0, 1] Q_i^{-1} \alpha(A_i) \quad (4.6)$$

where

$$\alpha(s) = s^n + \alpha_n s^{n-1} + \dots + \alpha_2 s + \alpha_1$$

is a desired stable polynomial, and

$$Q_i = [B_i \ A_i B_i \ A_i^2 B_i \ \dots \ A_i^{n-1} B_i]$$

4.3 Fuzzy modelling and tracking control of nonlinear systems

4.3.1 Fuzzy modelling and tracking controller design

Many Physical systems are very complex in practice so that it is very difficult to obtain their rigorous mathematical models. In recent years, fuzzy logic has been

applied to the field of system modelling and control engineering (Takagi and Sugeno, 1985; Wang and Mendel, 1992; Feng et al., 1997) by means of combining human heuristic reasoning and expert experience. In this section, the fuzzy model is established by the following fuzzy inference rules which include local linearised subsystems and feedback controllers.

$$\begin{aligned}
 R^i: \quad & \text{IF} \quad x_1 \text{ is } F_1^i \text{ AND } \dots x_n \text{ is } F_n^i \\
 & \text{THEN} \\
 & \dot{\tilde{x}} = A_i \tilde{x} + B_i \tilde{u} \\
 & \tilde{u} = u - u_i = -K_i \tilde{x} \\
 & i = 1, 2, \dots, l
 \end{aligned} \tag{4.7}$$

where R^i denotes the i -th fuzzy inference rule, l the number of inference rules, F_j^i ($j = 1, 2, \dots, n$) are fuzzy sets, $\tilde{x} = x - x_d$ is the tracking error of the system with desired trajectory x_d .

Let $\mu_i(x)$ be the normalized membership function of the inferred fuzzy set F^i where

$$F^i = \bigcap_{j=1}^n F_j^i \tag{4.8}$$

and

$$\sum_{i=1}^l \mu_i(x) = 1 \tag{4.9}$$

By using a standard fuzzy inference method, that is, using a singleton fuzzifier, product fuzzy inference and centre-average defuzzifier, the following global tracking error fuzzy model for the controlled nonlinear system can be obtained,

$$\dot{\tilde{x}} = A_0 \tilde{x} + B_0 \tilde{u} \quad (4.10)$$

$$\tilde{u} = u - u_d = -K\tilde{x} \quad (4.11)$$

where

$$\begin{aligned} A_0 &= \sum_{i=1}^l \mu_i A_i & B_0 &= \sum_{i=1}^l \mu_i B_i \\ K &= \sum_{i=1}^l \mu_i K_i & u_d &= \sum_{i=1}^l \mu_i u_i \end{aligned} \quad (4.12)$$

Remark 4.3: Here, we assume the fuzzy model is globally controllable, that is, (A_0, B_0) is a controllable pair.

4.3.2 A example of fuzzy modelling and control

Consider the following one-link robotic manipulator

$$ml^2\ddot{\theta} + d\dot{\theta} + mgl\cos(\theta) = u \quad (4.13)$$

with

- $m = 1\text{kg}$ - payload,
- $l = 1\text{m}$ - length of the link,
- $g = 9.81\text{m/s}^2$ - gravitational constant,
- $d = 1\text{kgm}^2/\text{s}$ - damping factor,
- u - control variable (kgm^2/s^2).

Assuming we are interested in the dynamics of the system in the range of $[-90^\circ, 90^\circ]$, then the fuzzy model can be obtained by linearising the nonlinear equation (4.13) over a number of points, such as $0^\circ, \pm 30^\circ, \pm 60^\circ, \pm 90^\circ$. The following fuzzy model is considered

$$R^1 : \text{IF } x_1 \text{ is about } -90^\circ, \text{ THEN } \dot{\tilde{x}} = A_1 \tilde{x} + B_1 \tilde{u}, \tilde{u} = u - u_1 = -K_1 \tilde{x}$$

$$R^2 : \text{IF } x_1 \text{ is about } -60^\circ, \text{ THEN } \dot{\tilde{x}} = A_2 \tilde{x} + B_2 \tilde{u}, \tilde{u} = u - u_2 = -K_2 \tilde{x}$$

$$R^3 : \text{IF } x_1 \text{ is about } -30^\circ, \text{ THEN } \dot{\tilde{x}} = A_3 \tilde{x} + B_3 \tilde{u}, \quad \tilde{u} = u - u_3 = -K_3 \tilde{x}$$

$$R^4 : \text{IF } x_1 \text{ is about } 0^\circ, \text{ THEN } \dot{\tilde{x}} = A_4 \tilde{x} + B_4 \tilde{u}, \quad \tilde{u} = u - u_4 = -K_4 \tilde{x}$$

$$R^5 : \text{IF } x_1 \text{ is about } 30^\circ, \text{ THEN } \dot{\tilde{x}} = A_5 \tilde{x} + B_5 \tilde{u}, \quad \tilde{u} = u - u_5 = -K_5 \tilde{x}$$

$$R^6 : \text{IF } x_1 \text{ is about } 60^\circ, \text{ THEN } \dot{\tilde{x}} = A_6 \tilde{x} + B_6 \tilde{u}, \quad \tilde{u} = u - u_6 = -K_6 \tilde{x}$$

$$R^7 : \text{IF } x_1 \text{ is about } 90^\circ, \text{ THEN } \dot{\tilde{x}} = A_7 \tilde{x} + B_7 \tilde{u}, \quad \tilde{u} = u - u_7 = -K_7 \tilde{x}$$

where

$$x_1 = \theta, \quad x_2 = \dot{\theta}, \quad \tilde{x} = [\tilde{x}_1, \tilde{x}_2]^T$$

$$\begin{aligned} A_1 &= \begin{bmatrix} 0 & 1 \\ -9.81 & -1 \end{bmatrix}, & B_1 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_1 &= 0, \\ A_2 &= \begin{bmatrix} 0 & 1 \\ -8.5 & -1 \end{bmatrix}, & B_2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_2 &= 4.9, \\ A_3 &= \begin{bmatrix} 0 & 1 \\ -4.9 & -1 \end{bmatrix}, & B_3 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_3 &= 8.5, \\ \\ A_4 &= \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & B_4 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_4 &= 9.81, \\ A_5 &= \begin{bmatrix} 0 & 1 \\ 4.9 & -1 \end{bmatrix}, & B_5 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_5 &= 8.5, \\ A_6 &= \begin{bmatrix} 0 & 1 \\ 8.5 & -1 \end{bmatrix}, & B_6 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_6 &= 4.9, \\ A_7 &= \begin{bmatrix} 0 & 1 \\ 9.81 & -1 \end{bmatrix}, & B_7 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_7 &= 0. \end{aligned}$$

The fuzzy sets for x_1 are chosen as in Fig.4.1.

The desired closed loop poles for each local model are chosen as $[-4, -3]$. Thus the feedback control gains are found by using of pole placement method as follows

$$K_1 = [2.2 \quad 6], \quad K_2 = [3.5 \quad 6], \quad K_3 = [7.1 \quad 6], \quad K_4 = [12 \quad 6],$$

$$K_5 = [16.9 \quad 6], \quad K_6 = [20.5 \quad 6], \quad K_7 = [21.8 \quad 6].$$

Therefore, the fuzzy control input can be written as

$$u = -\sum_{i=1}^7 \mu_i K_i \tilde{x} + \sum_{i=1}^7 \mu_i u_i \quad (4.14)$$

The control objective in this simulation is to force the one-link robotic manipulator to follow a desired trajectory which is generated by the following reference model

$$\begin{bmatrix} \dot{x}_d \\ \ddot{x}_d \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -25 & -10 \end{bmatrix} \begin{bmatrix} x_d \\ \dot{x}_d \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_s \quad (4.15)$$

where u_s is chosen as in Fig.4.2.

In this example, the initial values of x and x_d are selected as

$$[x_1(0), \dot{x}_1(0)] = [17.2^\circ, 0], \quad [x_d(0), \dot{x}_d(0)] = [0, 0].$$

Fig. 4.3 shows the output tracking using the fuzzy feedback controller. It can be seen that good tracking performance has been achieved by the proposed fuzzy model. Fig. 4.4 shows the control input.

In order to examine the robustness of the fuzzy model, we use five fuzzy inference rules instead of seven. If we choose the fuzzy sets as in Fig. 4.5 and the following fuzzy rules,

$$R^1 : \text{IF } x_1 \text{ is about } -90^\circ, \text{ THEN } \dot{\tilde{x}} = A_1 \tilde{x} + B_1 \tilde{u}, \quad \tilde{u} = u - u_1 = -K_1 \tilde{x}$$

$$R^2 : \text{IF } x_1 \text{ is about } -45^\circ, \text{ THEN } \dot{\tilde{x}} = A_2 \tilde{x} + B_2 \tilde{u}, \quad \tilde{u} = u - u_2 = -K_2 \tilde{x}$$

$$R^3 : \text{IF } x_1 \text{ is about } 0^\circ, \text{ THEN } \dot{\tilde{x}} = A_3 \tilde{x} + B_3 \tilde{u}, \quad \tilde{u} = u - u_3 = -K_3 \tilde{x}$$

$$R^4 : \text{IF } x_1 \text{ is about } 45^\circ, \text{ THEN } \dot{\tilde{x}} = A_4 \tilde{x} + B_4 \tilde{u}, \quad \tilde{u} = u - u_4 = -K_4 \tilde{x}$$

$$R^5 : \text{IF } x_1 \text{ is about } 90^\circ, \text{ THEN } \dot{\tilde{x}} = A_5 \tilde{x} + B_5 \tilde{u}, \quad \tilde{u} = u - u_5 = -K_5 \tilde{x}$$

with

$$\begin{aligned}
A_1 &= \begin{bmatrix} 0 & 1 \\ -9.81 & -1 \end{bmatrix}, & B_1 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_1 &= 0, \\
A_2 &= \begin{bmatrix} 0 & 1 \\ -6.94 & -1 \end{bmatrix}, & B_2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_2 &= 6.94, \\
A_3 &= \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & B_3 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_3 &= 9.81, \\
A_4 &= \begin{bmatrix} 0 & 1 \\ 6.94 & -1 \end{bmatrix}, & B_4 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_4 &= 6.94, \\
A_5 &= \begin{bmatrix} 0 & 1 \\ 9.81 & -1 \end{bmatrix}, & B_5 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_5 &= 0.
\end{aligned}$$

then the output tracking and fuzzy control input are shown in Fig4.6 and Fig.4.7, respectively. It can be seen that the output tracking error is increased in the vicinity of 60° in comparison with Fig.4.3. However, if we choose the fuzzy sets as in Fig. 4.8 and the following fuzzy rules,

$$R^1: \text{IF } x_1 \text{ is about } -90^\circ, \text{ THEN } \dot{\tilde{x}} = A_1 \tilde{x} + B_1 \tilde{u}, \quad \tilde{u} = u - u_1 = -K_1 \tilde{x}$$

$$R^2: \text{IF } x_1 \text{ is about } -60^\circ, \text{ THEN } \dot{\tilde{x}} = A_2 \tilde{x} + B_2 \tilde{u}, \quad \tilde{u} = u - u_2 = -K_2 \tilde{x}$$

$$R^3: \text{IF } x_1 \text{ is about } 0^\circ, \text{ THEN } \dot{\tilde{x}} = A_3 \tilde{x} + B_3 \tilde{u}, \quad \tilde{u} = u - u_3 = -K_3 \tilde{x}$$

$$R^4: \text{IF } x_1 \text{ is about } 60^\circ, \text{ THEN } \dot{\tilde{x}} = A_4 \tilde{x} + B_4 \tilde{u}, \quad \tilde{u} = u - u_4 = -K_4 \tilde{x}$$

$$R^5: \text{IF } x_1 \text{ is about } 90^\circ, \text{ THEN } \dot{\tilde{x}} = A_5 \tilde{x} + B_5 \tilde{u}, \quad \tilde{u} = u - u_5 = -K_5 \tilde{x}$$

with

$$\begin{aligned}
A_1 &= \begin{bmatrix} 0 & 1 \\ -9.81 & -1 \end{bmatrix}, & B_1 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_1 &= 0, \\
A_2 &= \begin{bmatrix} 0 & 1 \\ -8.5 & -1 \end{bmatrix}, & B_2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_2 &= 4.9, \\
A_3 &= \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & B_3 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_3 &= 9.81,
\end{aligned}$$

$$\begin{aligned} A_4 &= \begin{bmatrix} 0 & 1 \\ 8.5 & -1 \end{bmatrix}, & B_4 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_4 &= 4.9, \\ A_5 &= \begin{bmatrix} 0 & 1 \\ 9.81 & -1 \end{bmatrix}, & B_5 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_7 &= 0. \end{aligned}$$

then the output tracking and fuzzy control input are shown in Fig.4.9 and Fig.4.10, respectively. It is clearly shown that the output tracking error is diminished in the vicinity of 60° by comparing with Fig.4.6. Hence, it can be concluded that if the fuzzy rules are selected appropriately, good tracking performance can then be achieved even with less fuzzy rules.

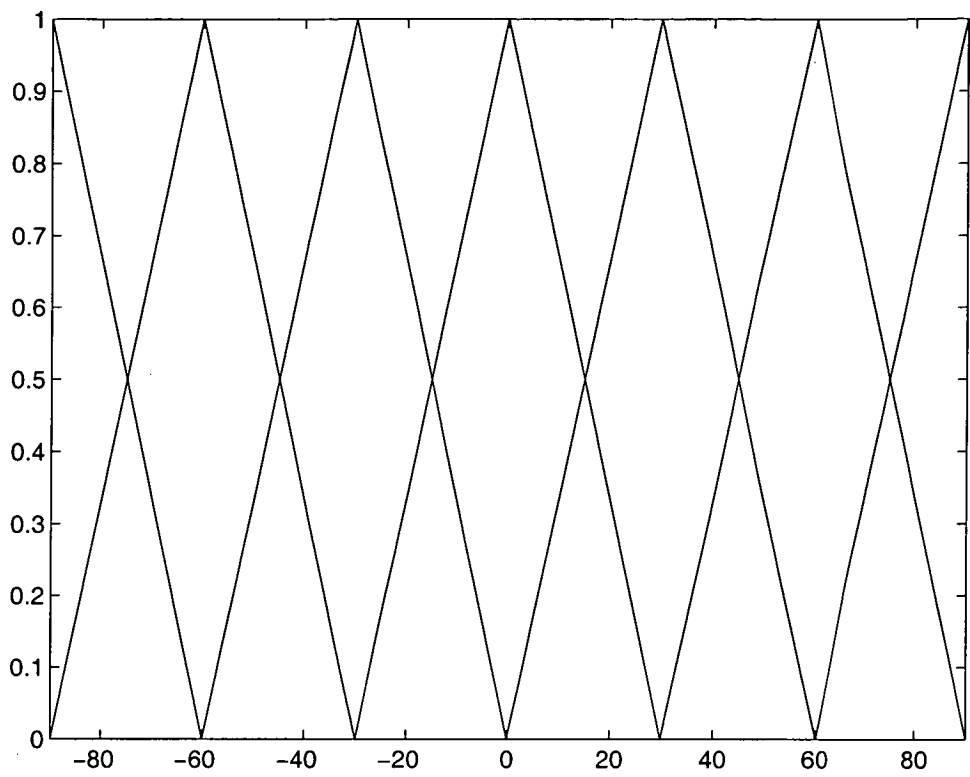


Fig. 4.1 Fuzzy sets of x_1

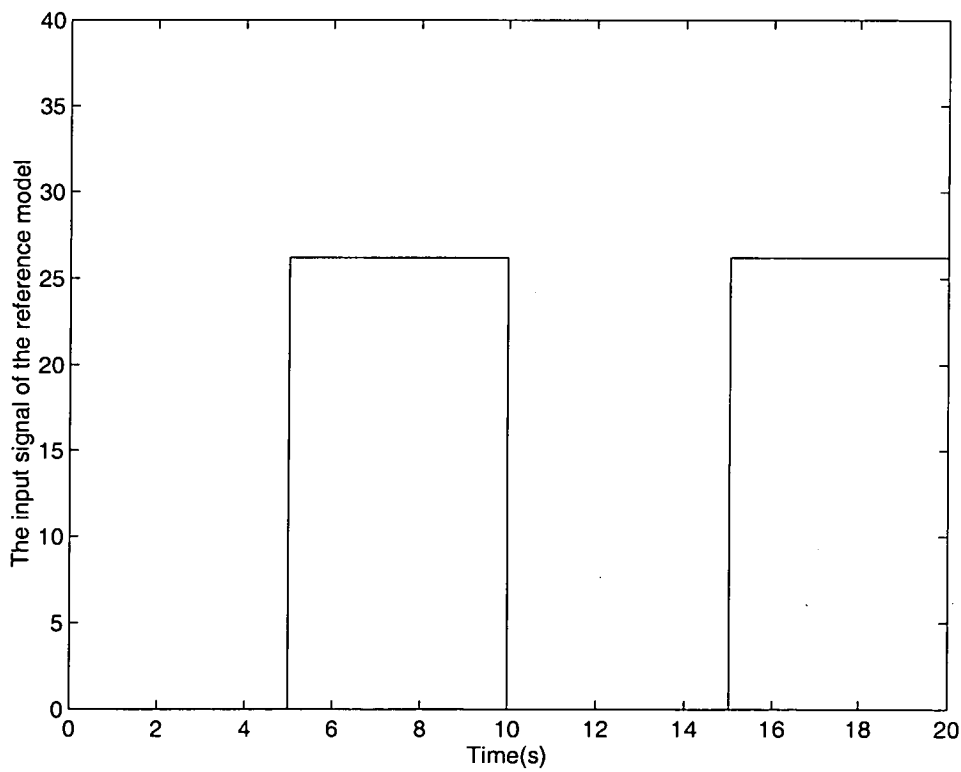


Fig.4.2 The control input of the reference model

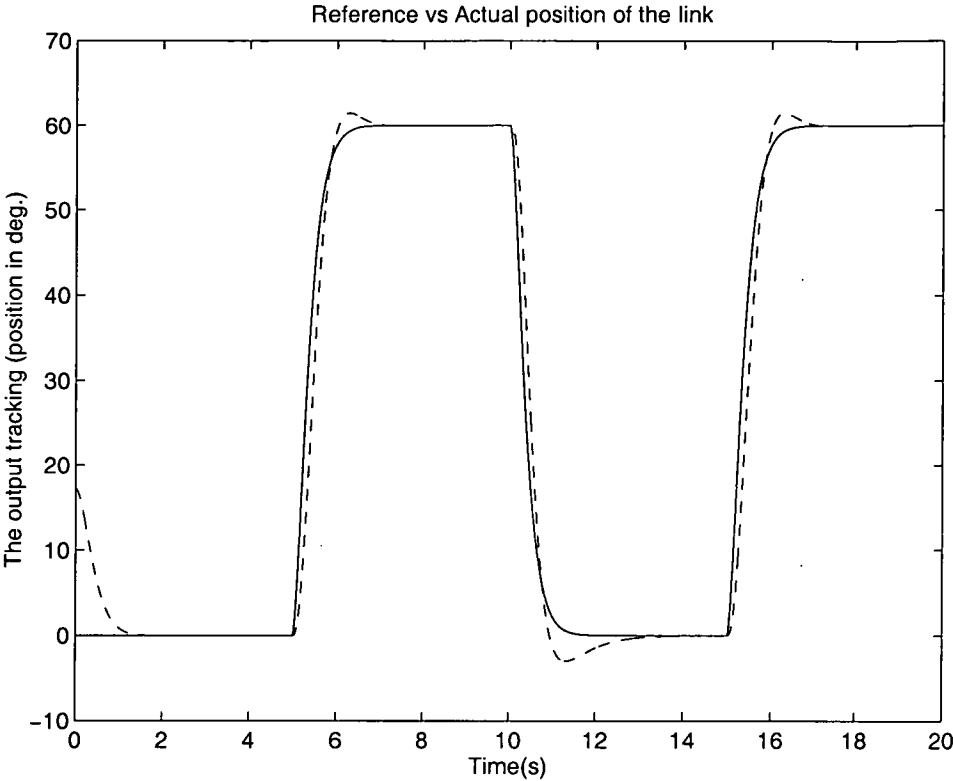


Fig.4.3 The output tracking of the link, where the solid line represents the desired trajectory.

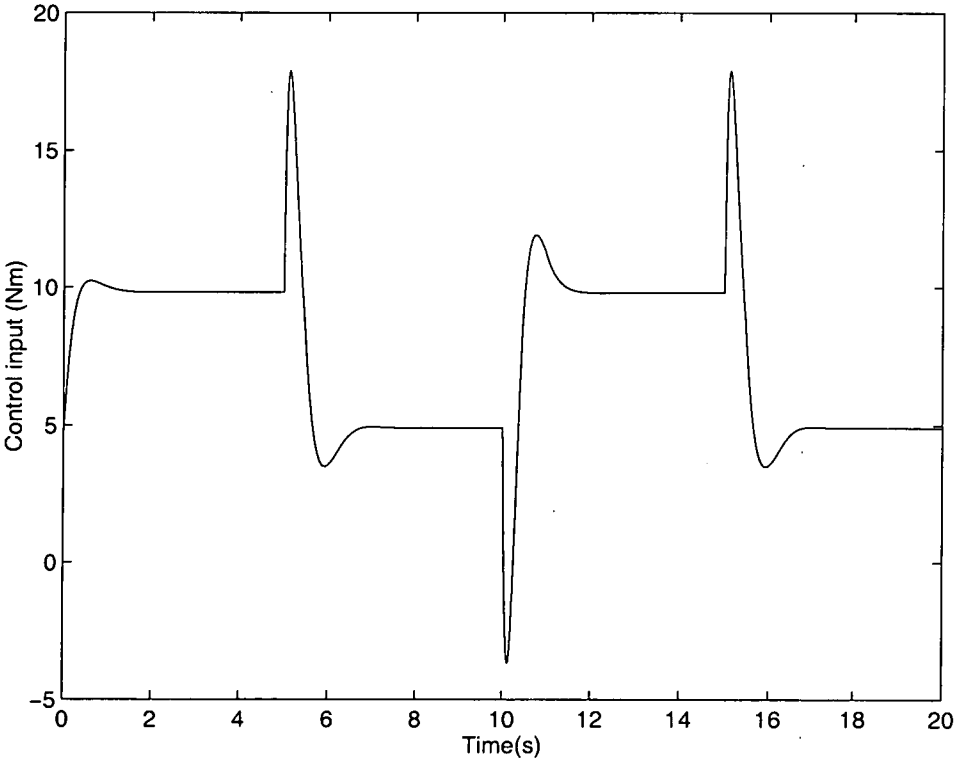


Fig 4.4 Fuzzy tracking control input

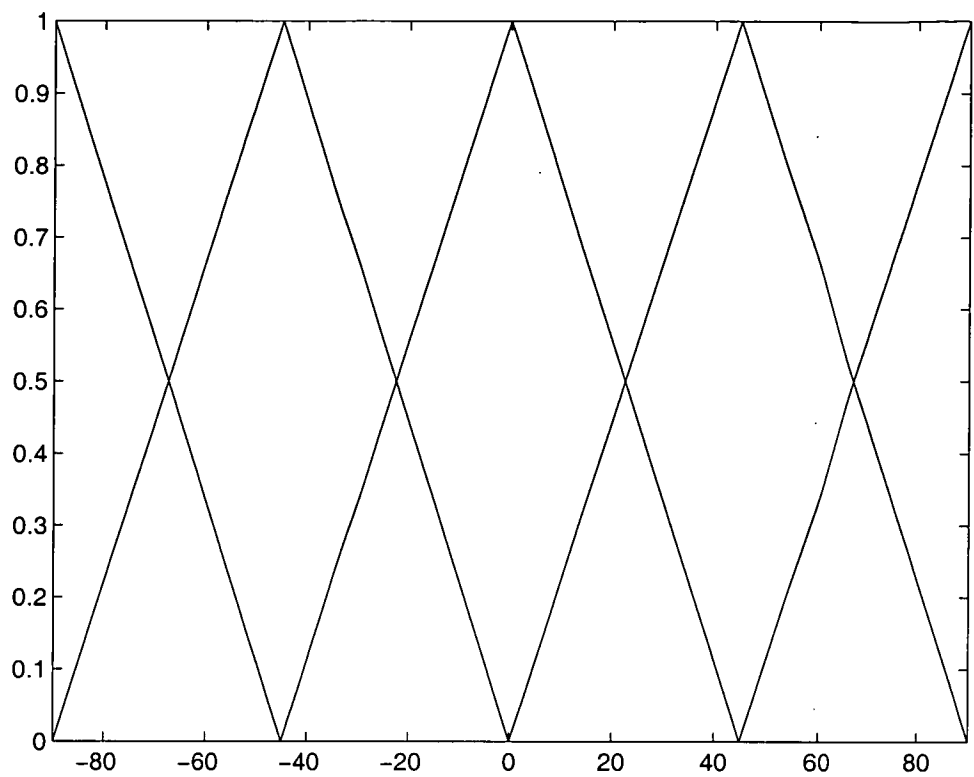


Fig.4.5 Fuzzy sets of x_1

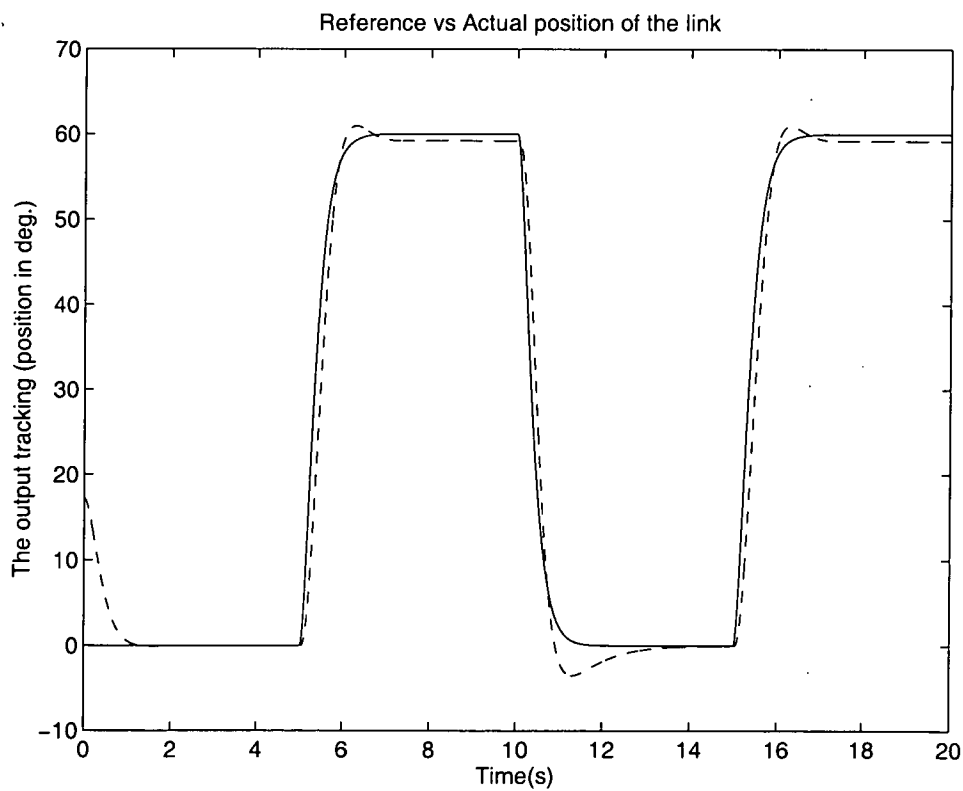


Fig.4.6 The output tracking of the link, where the solid line represents the desired trajectory.

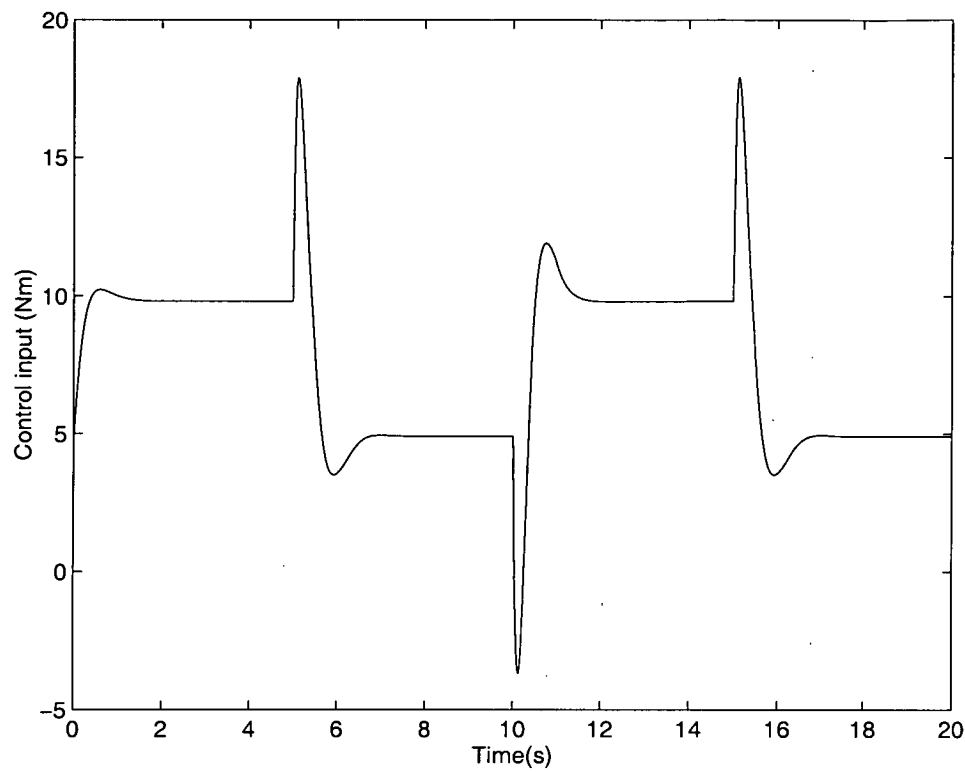


Fig.4.7 Fuzzy tracking control input

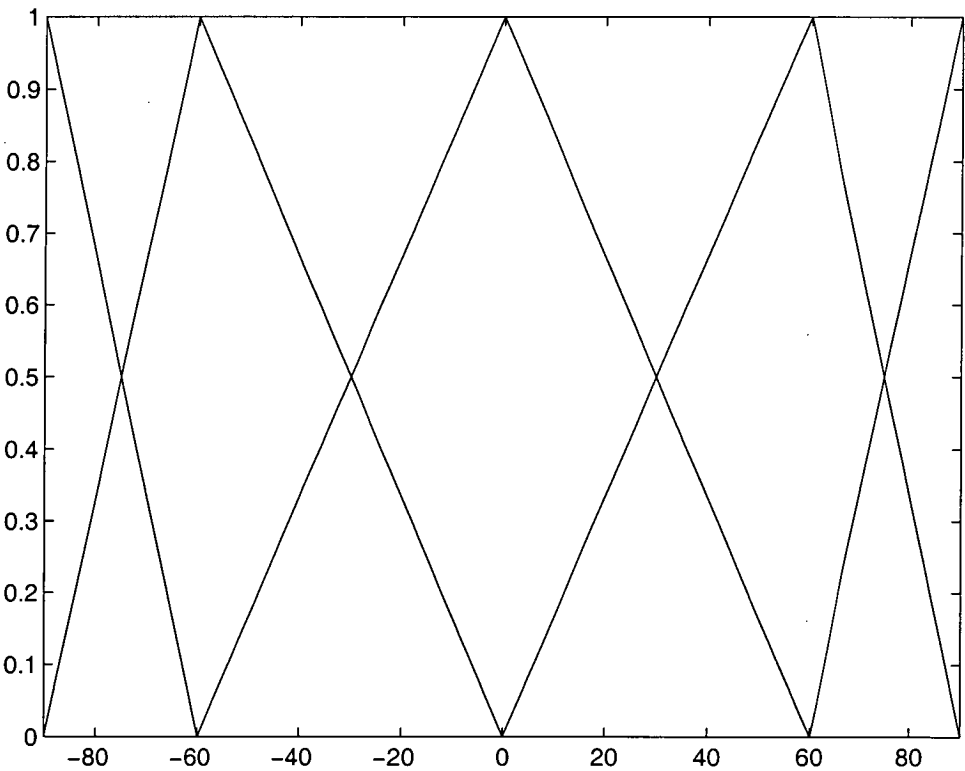


Fig.4.8 Fuzzy sets of x_1

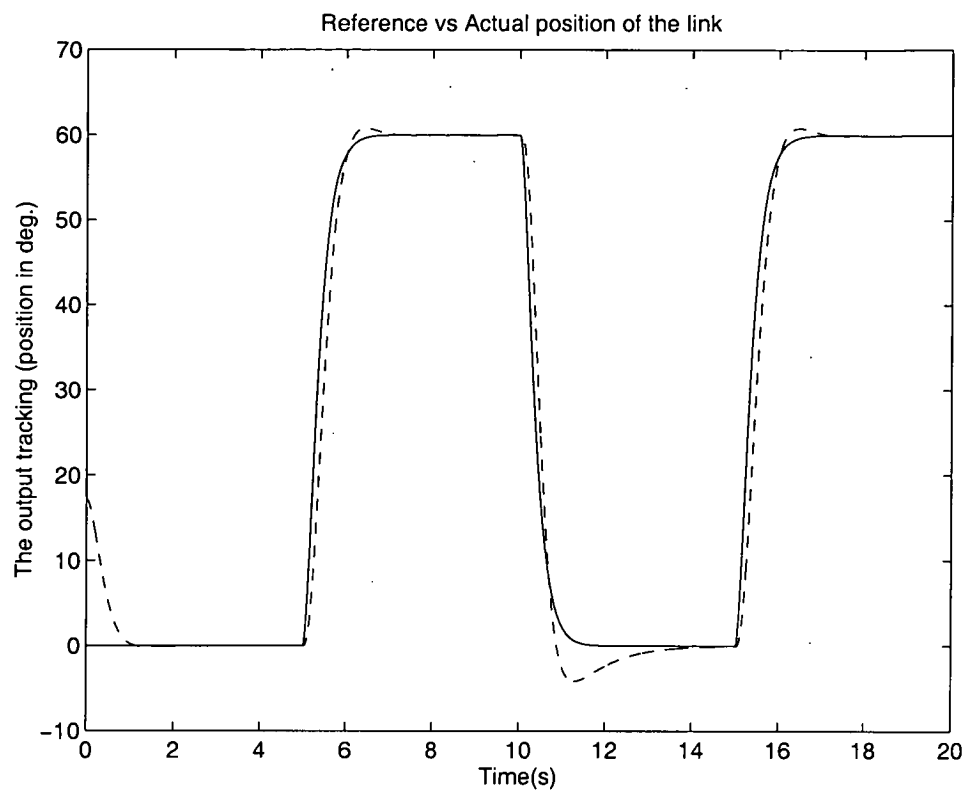


Fig.4.9 The output tracking of the link, where the solid line represents the desired trajectory.

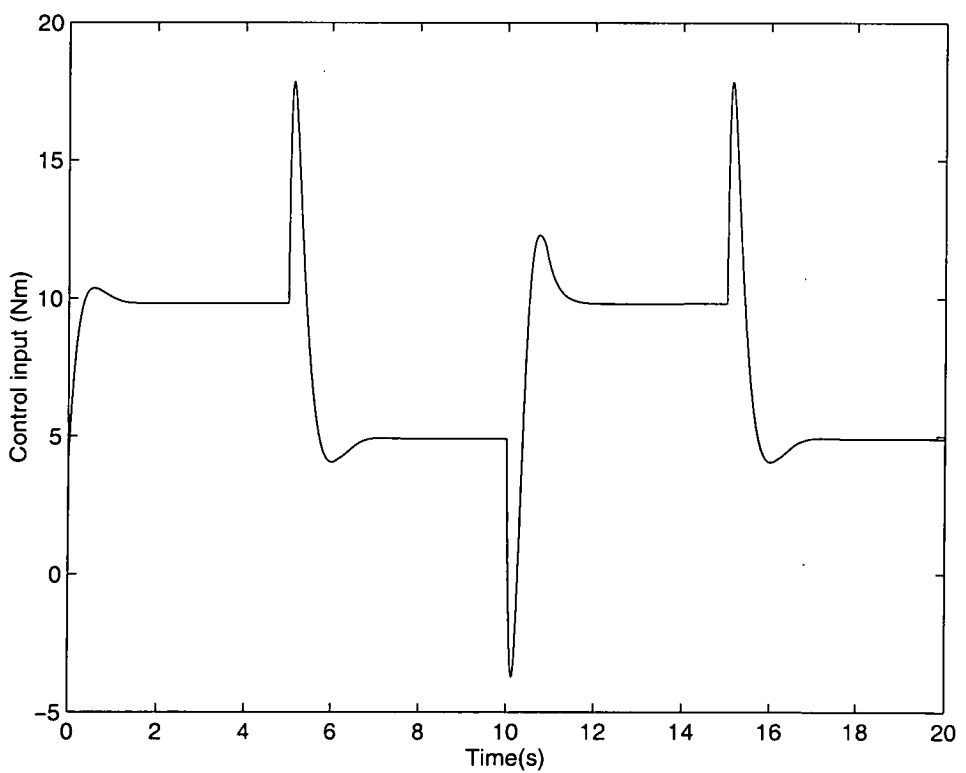


Fig.4.10 Fuzzy tracking control input

4.4 Robust tracking control with fuzzy nominal model

4.4.1 A robust tracking control scheme

The nonlinear system (4.1) can be expressed by the following error dynamic equation

$$\dot{\tilde{x}} = (A_0 + \Delta A)\tilde{x} + (B_0 + \Delta B)\tilde{u} + \Delta f \quad (4.16)$$

where $\Delta A, \Delta B$ represent the approximation error and system uncertainties and Δf denotes all the residual errors and disturbances which cannot be covered by $\Delta A, \Delta B$.

For further analysis, the following assumption is used in what follows (Man and Palaniswami, 1995)

Assumption 4.1: There exist matrix $H \in \mathbb{R}^{l \times n}$ and scalar E such that

$$\begin{aligned} \Delta A &= B_0 H \\ \Delta B &= B_0 E \\ 0 &< \|H\| < H_0 \\ 0 &< \|E\| < E_0 \\ 0 &< \|\Delta f\| < f_0 \end{aligned} \quad (4.17)$$

Remark 4.4: The above assumption is called uncertain matching conditions and have been used by several researchers (Man and Palaniswami, 1995; Shoureshi et al., 1990; Tarn et al., 1984). The upper bounds of the system uncertainties, namely H_0, E_0 and f_0 , may be obtained by experiments.

The objective of this section is to develop a robust tracking control scheme which ensures that the output tracking error \tilde{x} asymptotically converges to zero.

The controller design of the nonlinear system in this section is divided into two parts. First, a nominal feedback controller, as shown in (4.11), is designed based on fuzzy nominal system model, which guarantees that the tracking error \tilde{x} of the nominal system asymptotically converges to zero. Second, a variable structure compensator is designed based on an uncertain bound to eliminate the effects of the approximation error and uncertain dynamics so that the tracking error \tilde{x} of the closed loop system with uncertain dynamics asymptotically converges to zero.

The nominal feedback controller for the fuzzy nominal model is derived directly from (4.11), that is

$$u_0 = -K\tilde{x} + u_d \quad (4.18)$$

where

$$u_d = \sum_{i=1}^l \mu_i u^i \quad (4.19)$$

Next, we consider the variable structure compensator design for the uncertain system (4.16). Let the control input in system (4.1) have the following form

$$u = u_0 + u_1 \quad (4.20)$$

where u_0 is the nominal feedback control given in expression (4.18), and u_1 is a compensator to deal with the effects of system uncertainties.

Using expression (4.17) and (4.20) in (4.16), the error dynamics of the closed loop system with uncertainties is written as

$$\dot{\tilde{x}} = (A_0 - B_0 K)\tilde{x} + B_0 (H - EK)\tilde{x} + B_0 (I + E)u_1 + \Delta f \quad (4.21)$$

In order to use the variable structure theory to design the compensator u_1 , we define a switching hyperplane variable as follows

$$S = C\tilde{x} \quad (4.22)$$

where $C = [c_1, c_2, \dots, c_n]$ is chosen such that zeros of the polynomial $C\tilde{x} = 0$ are in the left half of the complex plane, and matrix CB_0 is non-singular (Man and Palaniswami, 1995).

For the design of the compensator u_1 and the stability analysis of tracking error dynamics (4.21), we give the following theorem.

Theorem 4.1: If the nominal feedback control u_0 is given by expression (4.18) and the compensator u_1 is designed such that

$$u_1 = \begin{cases} \frac{(SCB_0)}{\|SCB_0\|^2} [-SC(A_0 - B_0K)\tilde{x} + \rho] & S \neq 0 \\ 0 & S = 0 \end{cases} \quad (4.23)$$

where

$$\rho = \frac{1}{1-E_0} [-\|SC(A_0 - B_0K)\tilde{x}\| - \|SCB_0\|(H_0\|\tilde{x}\| + E_0\|K\tilde{x}\|) - \|S\|\|C\|f_0] \quad (4.24)$$

then, the tracking error \tilde{x} asymptotically converges to zero as time tends to infinity.

Proof: Defining a Lyapunov function

$$v = \frac{1}{2} S^2 \quad (4.25)$$

and differentiating v with respect to time, we get

$$\begin{aligned}
\dot{v} &= S\dot{S} \\
&= SC[(A_0 - B_0 K)\tilde{x} + B_0(H - EK)\tilde{x} + B_0(1 + E)u_1 + \Delta f] \\
&= -ESC(A_0 - B_0 K)\tilde{x} + SCB_0(H - EK)\tilde{x} + SC\Delta f + (1 - E)\rho \\
&\leq \|E\| \|SC(A_0 - B_0 K)\tilde{x}\| + \|SCB_0\| (\|H\| \|\tilde{x}\| + \|E\| \|K\tilde{x}\|) + \|S\| \|C\| \|\Delta f\| \\
&\quad + \frac{1 - E}{1 - E_0} [-\|SC(A_0 - B_0 K)\tilde{x}\| - \|SCB_0\| (H_0 \|\tilde{x}\| + E_0 \|K\tilde{x}\|) - \|S\| \|C\| \|f_0\|] \\
&\leq -(1 - \|E\|) \|SC(A_0 - B_0 K)\tilde{x}\| - \|S\| \|C\| (f_0 - \|\Delta f\|) \\
&\quad - \|SCB_0\| [(H_0 - \|H\|) \|\tilde{x}\| + (E_0 - \|E\|) \|K\tilde{x}\|] \\
&< 0, \quad \text{for } S \neq 0
\end{aligned} \tag{4.26}$$

Expression (4.26) is the sufficient condition for the switching hyperplane variable S to reach the sliding mode

$$S = C\tilde{x} = 0 \tag{4.27}$$

In the sliding mode, the tracking error \tilde{x} asymptotically converges to zero.

Remark 4.5: The robustness property of the proposed control scheme in this section is obvious. First, the effects of the approximation error and uncertain dynamics can be eliminated by using the variable structure compensator. Second, the closed loop system is completely insensitive to uncertainties after the system error dynamics reach the sliding mode.

Remark 4.6: To eliminate the chattering in the control input, the following boundary layer compensator (Man and Palaniswami, 1995) can be used in place of expression (4.23).

$$u_1 = \begin{cases} \frac{(SCB_0)}{\|SCB_0\|^2} [-SC(A_0 - B_0 K)\tilde{x} + \rho] & \|SCB_0\| \geq \delta_1 \\ \frac{(SCB_0)}{\delta_1^2} [-SC(A_0 - B_0 K)\tilde{x} + \rho] & \|SCB_0\| < \delta_1 \end{cases} \tag{4.28}$$

where $\delta_1 > 0$.

The above boundary layer compensator can force switching plane variable to move towards the sliding mode surface and then the control signal can be smoothed inside a boundary layer. This will achieve optimal trade-off between control bandwidth and tracking precision. Therefore, the chattering and sensitivity of the controller to system uncertainties can be eliminated (Man and Palaniswami, 1995).

4.4.2 A simulation example

To illustrate the proposed robust tracking control scheme in this section, a simulation example is carried out for a one-link robotic manipulator. The dynamic equation of the one-link robotic manipulator is given by

$$ml^2\ddot{\theta} + d\dot{\theta} + mgl \cos(\theta) = u \quad (4.29)$$

with

- $m = 1\text{kg}$ - payload,
- $l = 1\text{m}$ - length of the link,
- $g = 9.81\text{m/s}^2$ - gravitational constant,
- $d = 1\text{kgm}^2/\text{s}$ - damping factor,
- u - control variable (kgm^2/s^2).

Assuming we are interested in the dynamics of the system in the range of $[-90^\circ, 90^\circ]$, then the fuzzy nominal model can be obtained by linearising the nonlinear equation (4.29) over a number of points, such as $0^\circ, \pm 45^\circ, \pm 90^\circ$. The following fuzzy nominal model has been obtained.

$$\begin{aligned} R^1 : \text{IF } x_1 \text{ is about } 0^\circ \\ \text{THEN } \dot{\tilde{x}} = A_1 \tilde{x} + B_1 \tilde{u} \end{aligned}$$

$$\begin{aligned} R^2 : \text{IF } x_1 \text{ is about } -45^\circ \\ \text{THEN } \dot{\tilde{x}} = A_2 \tilde{x} + B_2 \tilde{u} \end{aligned}$$

$$\begin{aligned} R^3 : \text{IF } x_1 \text{ is about } +45^\circ \\ \text{THEN } \dot{\tilde{x}} = A_3 \tilde{x} + B_3 \tilde{u} \end{aligned}$$

$$\begin{aligned} R^4 : \text{IF } x_1 \text{ is about } -90^\circ \\ \text{THEN } \dot{\tilde{x}} = A_4 \tilde{x} + B_4 \tilde{u} \end{aligned}$$

$$\begin{aligned} R^5 : \text{IF } x_1 \text{ is about } +90^\circ \\ \text{THEN } \dot{\tilde{x}} = A_5 \tilde{x} + B_5 \tilde{u} \end{aligned}$$

where

$$\begin{aligned} x_1 = \theta, x_2 = \dot{\theta}, \tilde{x} = [\tilde{x}_1, \tilde{x}_2]^T, u_i = mgl \cos(\theta_i) \\ A_1 = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ A_2 = \begin{bmatrix} 0 & 1 \\ -6.94 & -1 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ A_3 = \begin{bmatrix} 0 & 1 \\ 6.94 & -1 \end{bmatrix}, \quad B_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ A_4 = \begin{bmatrix} 0 & 1 \\ -9.81 & -1 \end{bmatrix}, \quad B_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ A_5 = \begin{bmatrix} 0 & 1 \\ 9.81 & -1 \end{bmatrix}, \quad B_5 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned}$$

The fuzzy sets for x_1 are chosen as in Fig.4.11.

The desired closed loop poles for each local model are chosen as $[-4, -3]$. Thus the feedback control gains are found by using of pole placement method as follows

$$\begin{aligned}
K_1 &= [12 \ 6], \\
K_2 &= [5.1 \ 6], \\
K_3 &= [18.9 \ 6], \\
K_4 &= [2.2 \ 6], \\
K_5 &= [21.8 \ 6].
\end{aligned}$$

The control objective in this simulation is to force the one-link robotic manipulator to follow a desired trajectory which is generated by the following reference model

$$\begin{bmatrix} \dot{x}_d \\ \ddot{x}_d \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -25 & -10 \end{bmatrix} \begin{bmatrix} x_d \\ \dot{x}_d \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_s \quad (4.30)$$

where u_s is chosen as in Fig. 4.12.

In this example, the initial values of x and x_d are selected as

$$[x_1(0), \dot{x}_1(0)] = [17.2^\circ, 0], [x_d(0), \dot{x}_d(0)] = [0, 0].$$

Sliding mode is prescribed as

$$S = [10, 1] \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix}.$$

Uncertain bounds are selected as

$$E_0 = 0.1, H_0 = 2, f_0 = 1.$$

Fig. 4.13 shows the output tracking using only a fuzzy nominal feedback controller. It can be seen that due to large system uncertainties, the output cannot track the desired trajectory closely. Fig. 4.14 shows the output tracking and control input using a fuzzy nominal feedback controller with a variable structure compensator. Obviously, the effects of the system uncertainties are eliminated and a good tracking performance is achieved. But there exists chattering in the control input. Fig. 4.15 shows good performance of the closed loop system in which a fuzzy nominal feedback controller with a boundary layer compensator is used to eliminate control chattering.

4.5 Concluding remarks

A robust tracking control scheme has been proposed in this chapter for a class of nonlinear systems. The main contribution of this scheme is that a nominal system model for a nonlinear system is established by fuzzy synthesis of a set of linearised local subsystems, where the conventional linear feedback control technique is used to design a feedback controller for the fuzzy nominal system. A variable structure compensator is then designed to eliminate the effects of the approximation error and system uncertainties. Strong robustness with respect to large system uncertainties and asymptotic convergence of the output tracking error are obtained. A simulation example has been given to support the proposed control scheme.

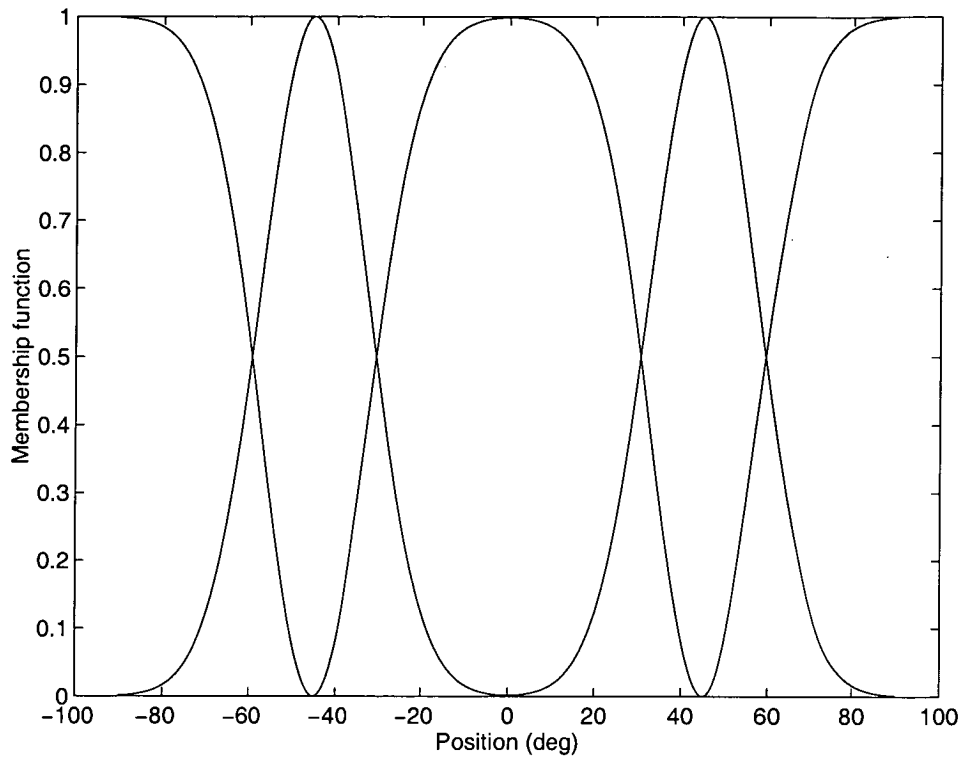


Fig.4.11 Fuzzy sets of state x_1

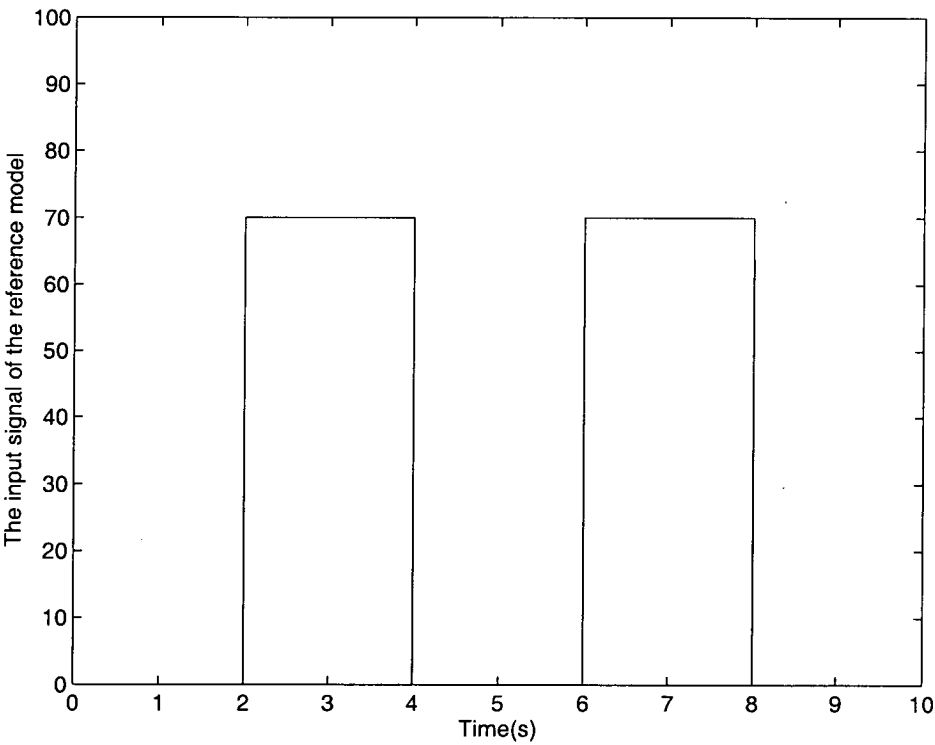


Fig. 4.12 The input signal u_s of the reference model

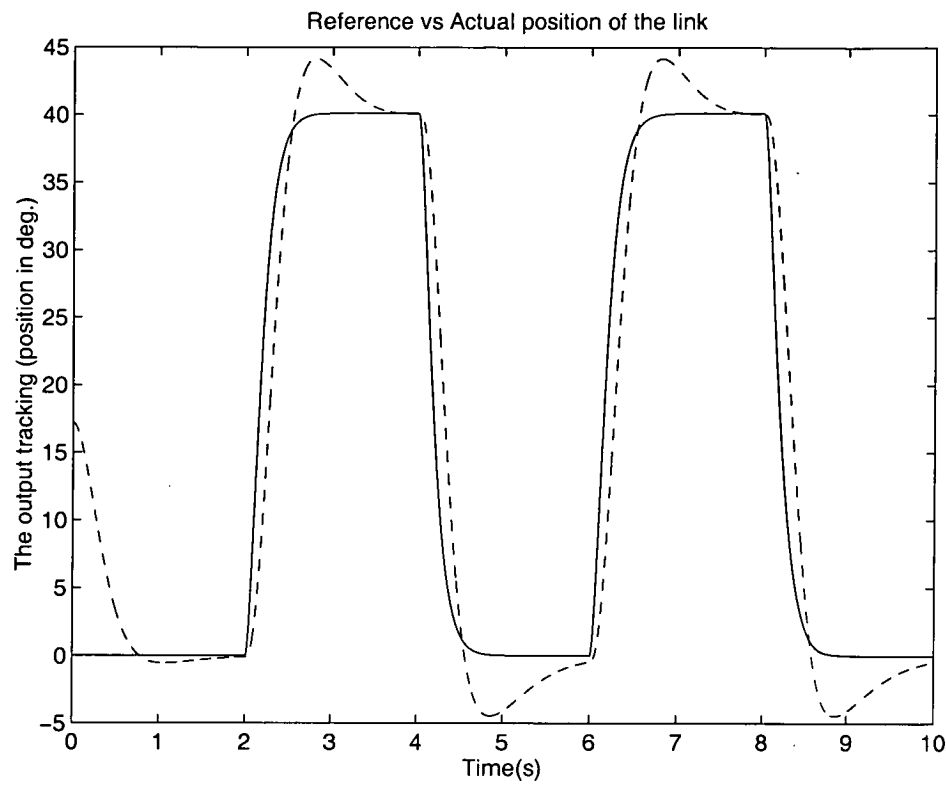


Fig.4.13-a The output tracking of the link using a fuzzy nominal feedback controller. Solid line represents the desired trajectory

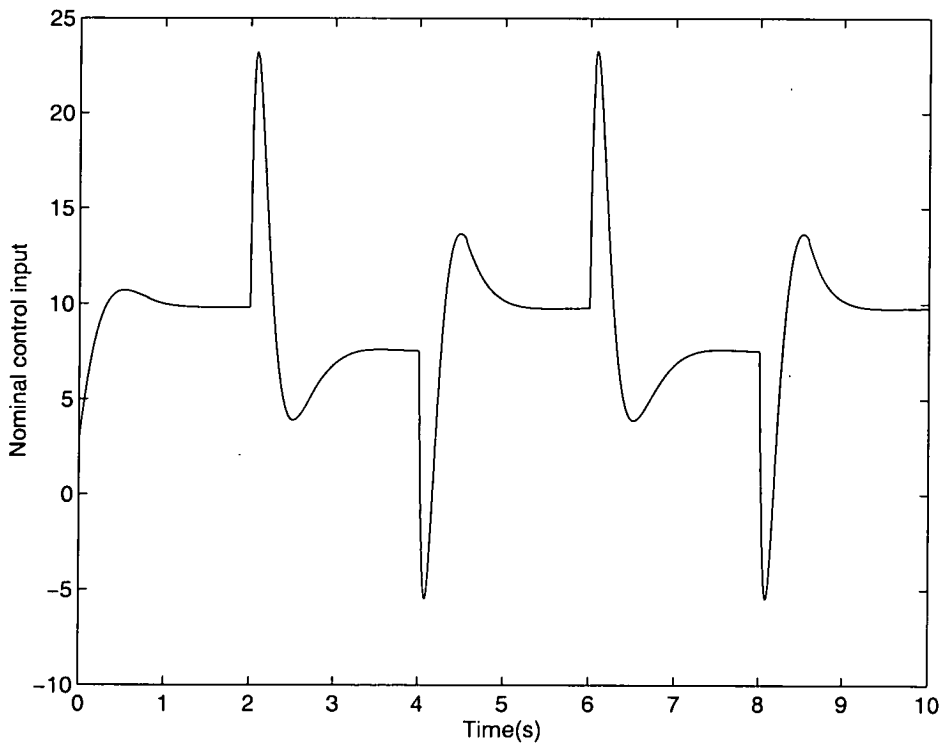


Fig. 4.13-b Nominal feedback control input

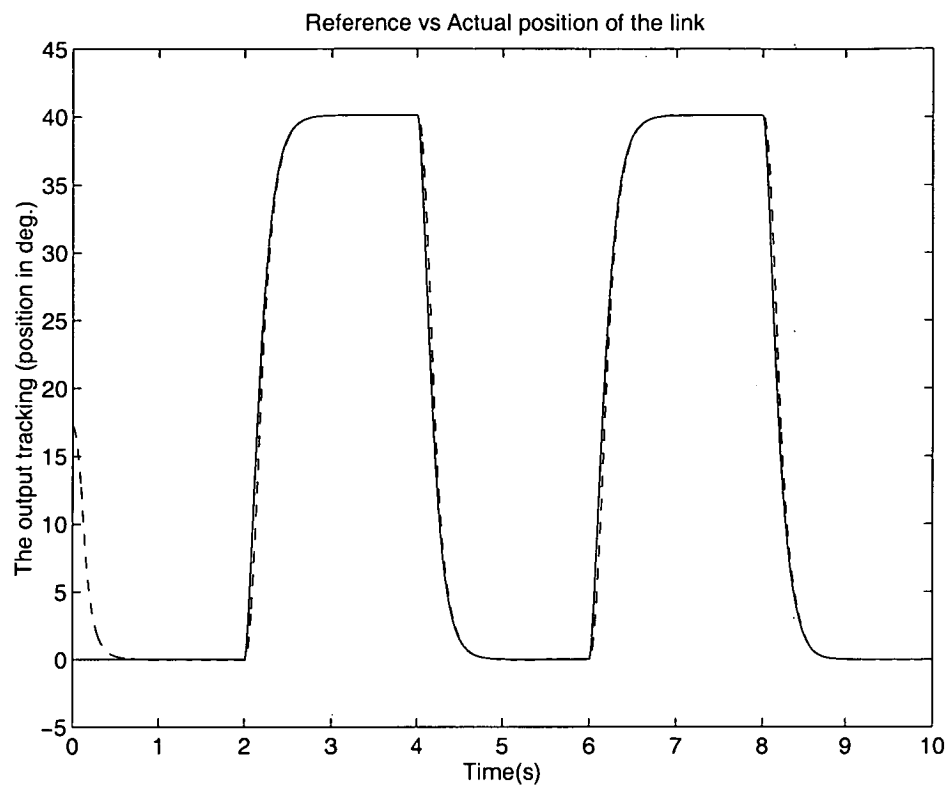


Fig. 4.14-a The output tracking with variable structure compensator

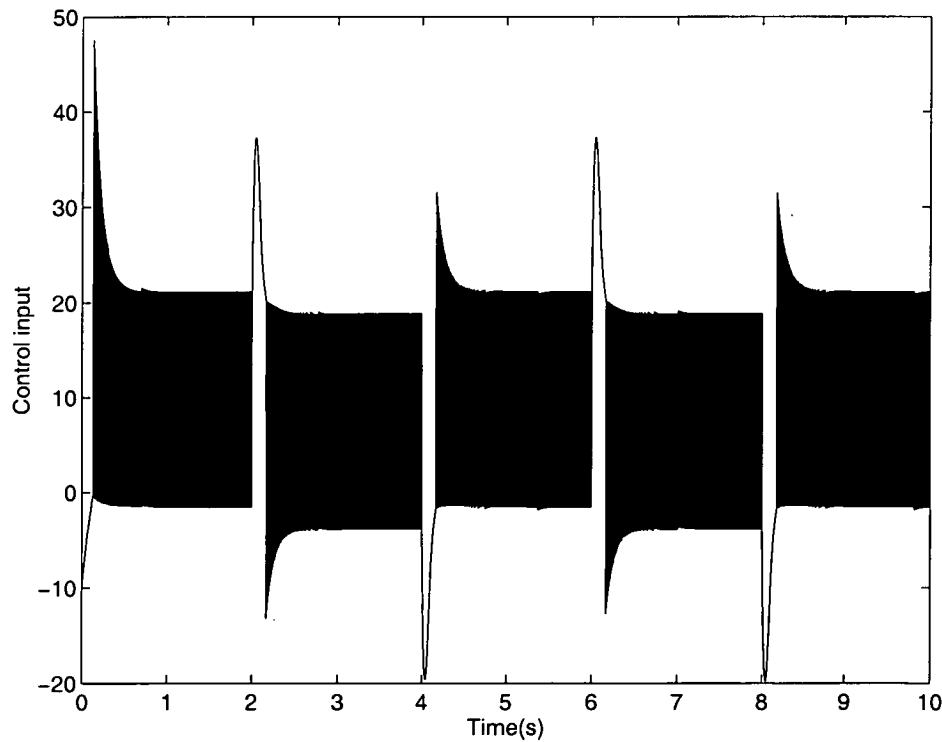


Fig. 4.14-b Control input with compensator

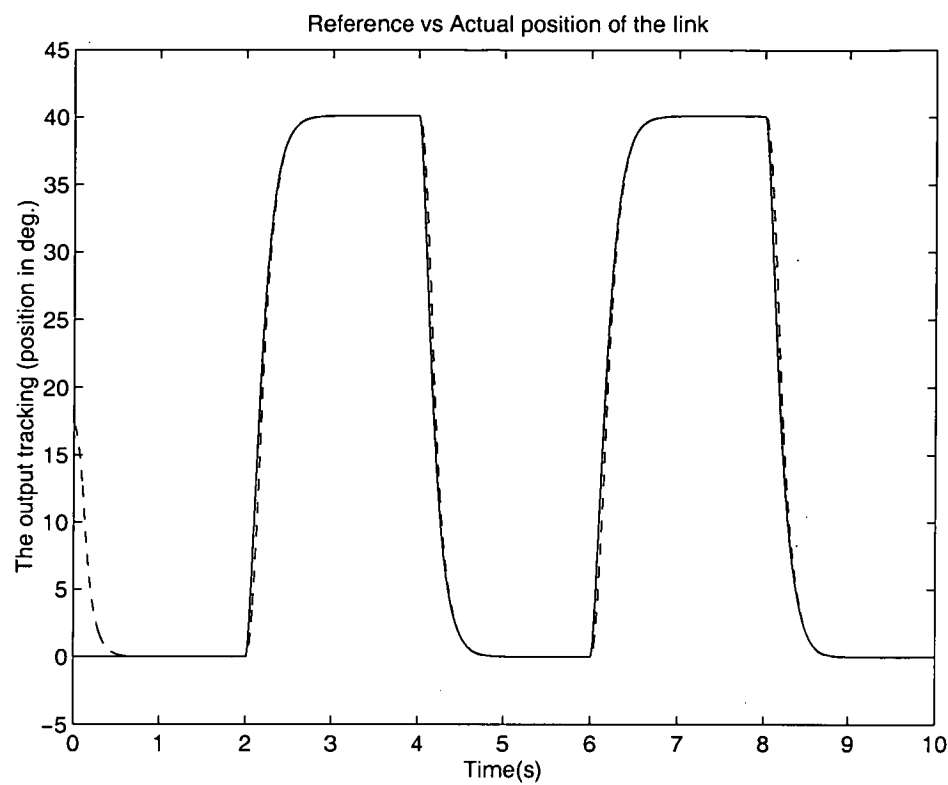


Fig. 4.15-a The output tracking using boundary layer compensator ($\delta_1 = 0.04$).

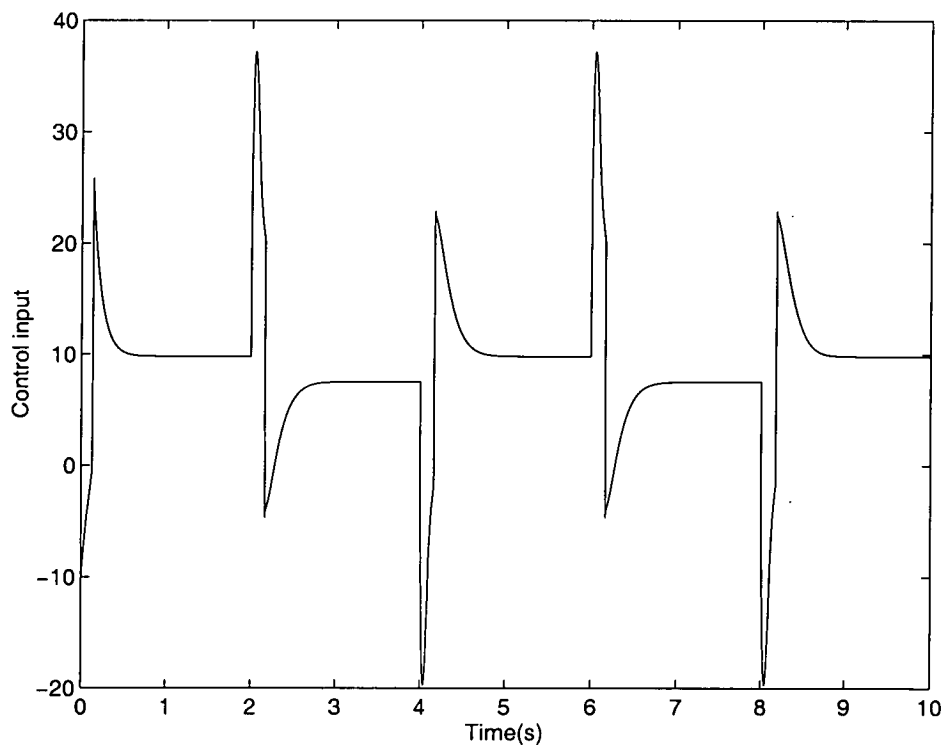


Fig. 4.15-b Control input with boundary layer compensator ($\delta_1 = 0.04$).

Chapter 5

Fuzzy Sliding Mode Control

5.1 Introduction

Fuzzy logic technique based on approximate reasoning has been widely used for the design of control systems (Mamdani and Assilian, 1974; Tagaki and Sugeno, 1985; Cao et al., 1996). The early fuzzy control algorithms proposed by Mamdani et al. have the following characteristics (Yager and Filev, 1994): (i) The detailed mathematical description of the controlled system is not required, (ii) the human experience and understanding about the system are used to design a fuzzy controller which allows us to implement heuristic strategies, which are defined by linguistically described statements, to formulate the control algorithm.

In order to improve the performance of fuzzy control systems, the fuzzy control algorithms with the Mamdani model have been further investigated and developed in recent years. It is shown that (Yager and Filev, 1994), if the structure of the fuzzy

control networks can be identified and the membership functions of the fuzzy sets can be estimated by the use of some optimisation techniques, a satisfactory tracking performance of the closed loop fuzzy systems can be obtained.

In this chapter, Lyapunov stability theory and fuzzy logic technique are combined together to design sliding mode controller. It is shown that a sliding mode is first designed to describe the desired system dynamics for the controlled system. A set of fuzzy rules are then used to adjust the parameters of the controller based on the Lyapunov function and its derivative. The sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering of the control signals, which makes the fuzzy sliding mode control more practical. A simulation example is given in support of the proposed control scheme.

The chapter is organised as follows. In section 5.2, a class of nonlinear systems to be considered, Lyapunov function and the sliding mode controllers are formulated. In section 5.3, the fuzzy rules used to adjust the parameters of the controller are described in detail. In section 5.4, a simulation for a one-link rigid robotic manipulator is performed in support of the proposed control scheme. Section 5.5 gives concluding remarks.

5.2 Sliding mode control of nonlinear systems

In this paper, we focus on the design of sliding mode controllers for a class of nonlinear systems whose dynamic equations can be expressed in the following form (Gao and Hung, 1993):

$$\dot{x} = A(x) + B(x)u \quad (5.1)$$

where $\dim-x = n$, $\dim-u = m$, $n > m$, and $A(x)$ and $B(x)$ are of appropriate dimensions. The switching surface is defined by

$$s(x) = c^T x = 0 \quad (5.2)$$

where $c^T = [c_1 \ c_2 \ \dots \ c_n]$ is chosen such that the zeros of the polynomial $c^T x = 0$ are in the left half-plane and $c^T B(x) \neq 0$. Eqn. (5.2) represents a desired system dynamics with a lower order than the given plant. The sliding mode controller u is designed such that any state x outside the switching surface is driven to reach the surface. On the switching surface, the sliding mode takes place, following the desired system dynamics.

Theorem 5.1: If the control law is designed as follows

$$u = -[c^T B(x)]^{-1} [c^T A(x) + Q \operatorname{sgn}(s) + Kh(s)] \quad (5.3)$$

where

$$\begin{aligned} Q &= \operatorname{diag}[q_1, \dots, q_m], \quad q_i > 0 \\ \operatorname{sgn}(s) &= [\operatorname{sgn}(s_1), \dots, \operatorname{sgn}(s_m)]^T \\ K &= \operatorname{diag}[k_1, \dots, k_m], \quad k_i > 0 \\ h(s) &= [h_1(s_1), \dots, h_m(s_m)]^T \\ s_i h_i(s_i) &> 0, \quad h_i(0) = 0 \end{aligned}$$

then the system state x reaches origin as time tends to infinity.

Proof: Consider the following Lyapunov function

$$V = \frac{1}{2} s^T s \quad (5.4)$$

The derivative of V is

$$\begin{aligned} \dot{V} &= s^T \dot{s} \\ &= s^T c^T \dot{x} \\ &= s^T c^T [A(x) + B(x)u] \\ &= s^T c^T [A(x) - B(x)[c^T B]^{-1} [c^T A(x) + Q \operatorname{sgn}(s) + Kh(s)]] \\ &= -s^T [Q \operatorname{sgn}(s) + Kh(s)] \\ &< 0 \quad s \neq 0 \quad \text{QED.} \end{aligned} \quad (5.5)$$

From (5.5), we have the following reaching law,

$$\dot{s} = -Q \operatorname{sgn}(s) - Kh(s) \quad (5.6)$$

It is clearly known that, if Q is too small, the reaching time will be too long. On the other hand, a large Q will cause severe chattering. In next section, a fuzzy tuning algorithm for Q is introduced to achieve a better trade-off between the settling time and chattering phenomena.

5.3 Fuzzy tuning of the sliding mode controller

The Mamdani's fuzzy inference model (Mamdani and Assilian, 1974) is used to adjust the controller's parameter matrix Q . The Lyapunov function defined in (5.4) can be considered as the distance between the desired and actual states. The control objective

is that the distance should be decreased as soon as possible. We divide the Lyapunov function $\frac{1}{2}s_i^2$ (V) into three fuzzy subsets, such as 'big', 'medium' and 'small'. The convergent rate or the derivative of Lyapunov function $s_i\dot{s}_i$ (\dot{V}) is divided as the fuzzy subsets 'positive', 'zero' and 'negative'. The fuzzy control system consists of two linguistic input variables V and \dot{V} and one linguistic output variable Q . The two fuzzy input variables are defined as follows

$$V=\{B, M, S\} \quad (5.7)$$

$$\dot{V}=\{P, Z, N\} \quad (5.8)$$

where B = big, M = medium, S = small, P = positive, Z = zero, N = negative. Fuzzy output variable Q is defined as follows

$$Q=\{B, MB, M, MS, S\} \quad (5.9)$$

where MB = medium big, MS = medium small. The input-output relation of the fuzzy controller with fuzzy variables (5.7), (5.8) and (5.9) is written by

$$V, \dot{V} \rightarrow Q \quad (5.10)$$

The fuzzy tuning algorithm for Q is described in fuzzy rules as follows and also represented in "look-up" Table 5.1.

1. If (V is B) and (\dot{V} is P) then (Q is B)

- 2. If (V is B) and (\dot{V} is Z) then (Q is MB)
- 3. If (V is B) and (\dot{V} is N) then (Q is M)
- 4. If (V is M) and (\dot{V} is P) then (Q is MB)
- 5. If (V is M) and (\dot{V} is Z) then (Q is M)
- 6. If (V is M) and (\dot{V} is N) then (Q is MS)
- 7. If (V is S) and (\dot{V} is P) then (Q is M)
- 8. If (V is S) and (\dot{V} is Z) then (Q is MS)
- 9. If (V is S) and (\dot{V} is N) then (Q is S)

Remark 5.1: The above fuzzy rules carry the control principle: when V is ‘big’ and \dot{V} is ‘positive’ (i.e., V is increasing), then chose ‘big’ control input Q to force the system state to converge to the desired state faster; when V is ‘small’ and \dot{V} is ‘negative’ (i.e., V is decreasing), then chose ‘small’ control input Q to decrease control chattering.

Q		\dot{V}		
		P	Z	N
V	B	B	MB	M
	M	MB	M	MS
	S	M	MS	S

Table 5.1. Fuzzy inference rules table

The membership functions for fuzzy variables V , \dot{V} , and Q with normalised universe of discourses are chosen as in Fig. 5.1 (a)-(c). The fuzzy inference surface is shown in Fig. 5.1 (d).

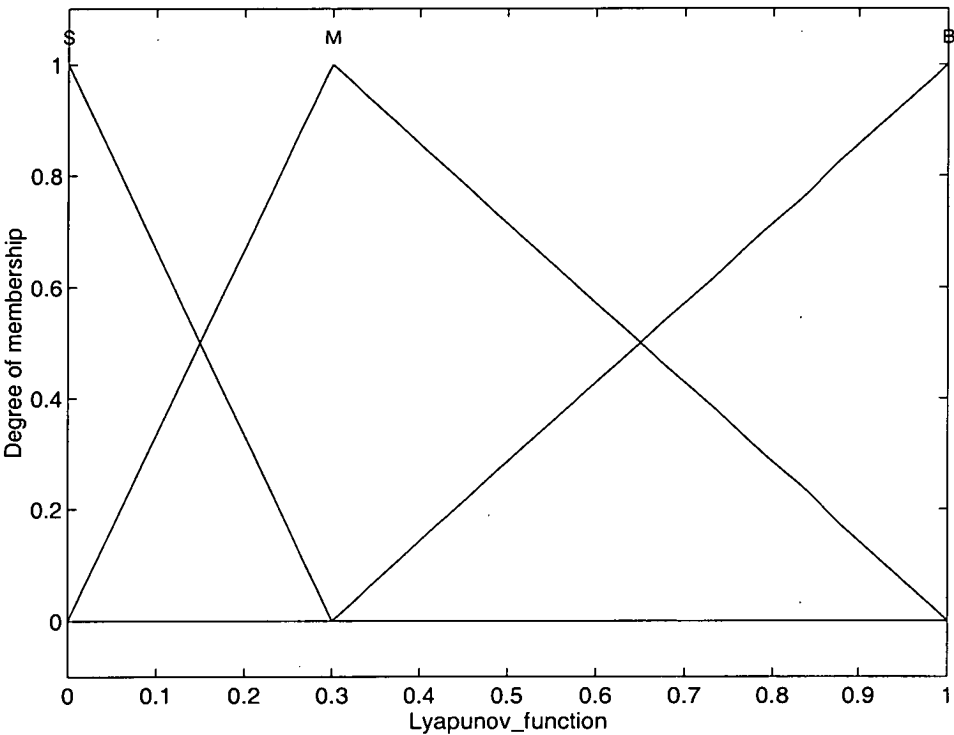


Fig. 5.1 (a) Fuzzy subsets of V

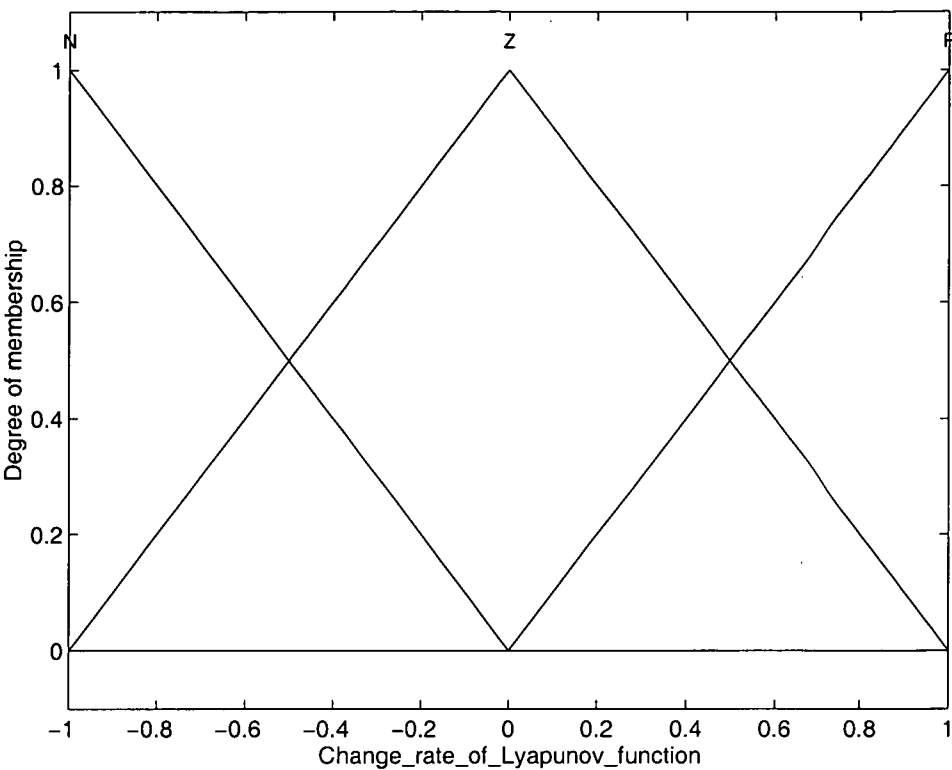


Fig. 5.1 (b) Fuzzy subsets of \dot{V}

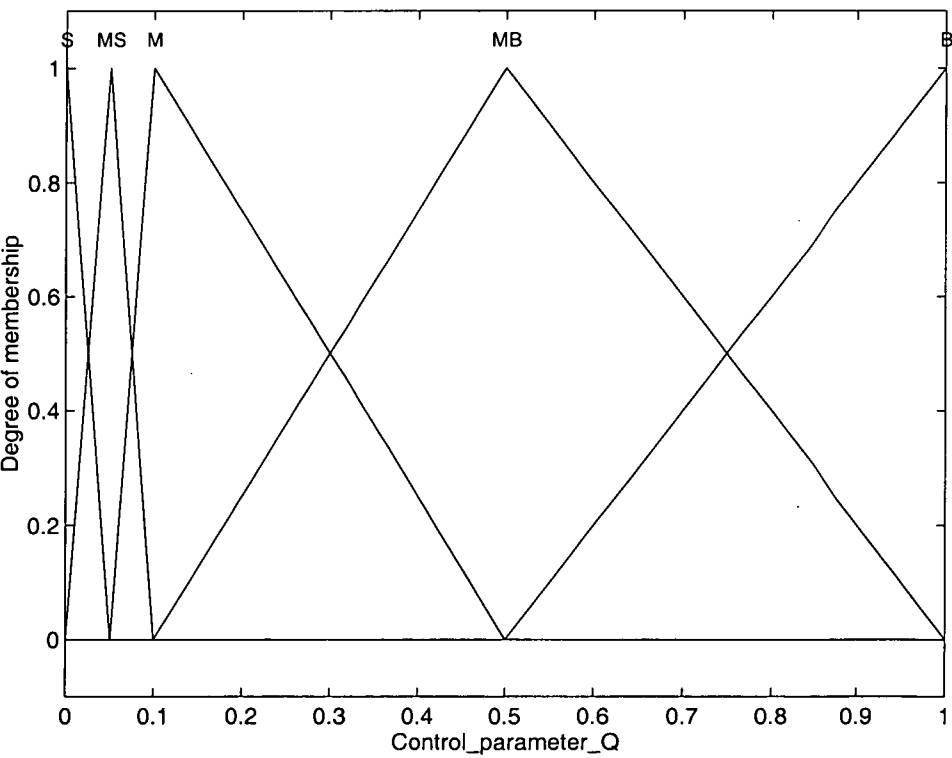
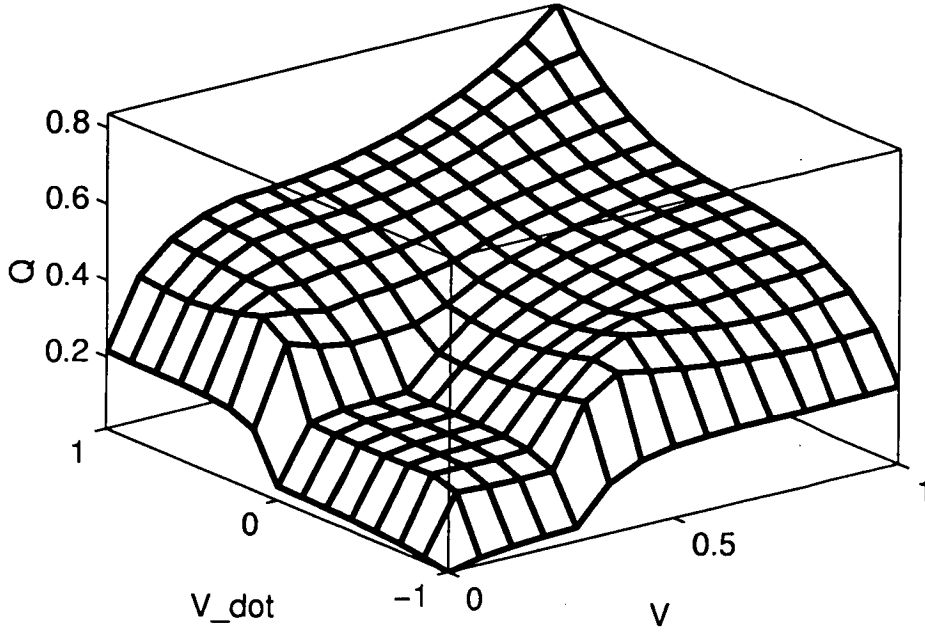


Fig. 5.1 (c) Fuzzy subsets of Q

Fig. 5.1 (d) Surface of $Q(V, \dot{V})$

5.4 An illustrative example

To illustrate the proposed fuzzy tuning sliding mode control scheme in this chapter, a simulation example is carried out for a one-link rigid robotic manipulator. The dynamic equation of motion of the manipulator is given by

$$ml^2\ddot{\theta} + d\dot{\theta} + mgl\cos(\theta) = u + u_d \quad (5.11)$$

with $m = 1\text{kg}$, $l = 1\text{m}$, $g = 9.81\text{m/s}^2$, $d = 1\text{kgm}^2/\text{s}$, and $u_d = 0.3\sin(10t)$ is the external disturbance.

Eqn. (5.11) can be rewritten in matrix form as follows

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_2 - 9.81\cos(x_1) + 0.3\sin(10t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \quad (5.12)$$

where x_1 represents the angle position θ . The control objective in this simulation is to force the one-link robotic manipulator to follow a desired trajectory which is generated by the following reference model

$$\begin{bmatrix} \dot{x}_d \\ \ddot{x}_d \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -25 & -10 \end{bmatrix} \begin{bmatrix} x_d \\ \dot{x}_d \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_s \quad (5.13)$$

where u_s is chosen as in Fig. 5.2.

Define an error vector

$$e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} x_1 - x_d \\ \dot{x}_1 - \dot{x}_d \end{bmatrix} \quad (5.14)$$

Sliding mode is prescribed as

$$s = c^T e = [\Lambda, I] \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \Lambda e_1 + \dot{e}_1 \quad (5.15)$$

Adopt the reaching law

$$\dot{s} = -Q \operatorname{sgn}(s) \quad (5.16)$$

Taking the time derivative of (5.14) gives

$$\dot{s} = \Lambda \dot{e}_1 - \ddot{x}_d - \dot{x}_2 - 9.81 \cos(x_1) + 0.3 \sin(10t) + u \quad (5.17)$$

Equating (5.16) and (5.17) and solving for the control u yields

$$u = -Q \operatorname{sgn}(s) - \Lambda \dot{e}_1 + \ddot{x}_d + \dot{x}_2 + 9.81 \cos(x_1) - 0.3 \sin(10t) \quad (5.18)$$

Fig. 5.3 shows the output tracking using the conventional sliding mode controller (5.18) with $Q=5$ and $\Lambda=6$, which is shown in Fig. 5.4. Fig. 5.5 shows the output tracking using a sliding mode controller (5.18) with fuzzy tuning for Q , the control input is shown in Fig. 5.6. The scaling factor for fuzzy variables $\frac{1}{2}s^2$, ss and Q are chosen as 1, 6 and 5, respectively. It can be seen that the control chattering is greatly reduced with fuzzy tuning.

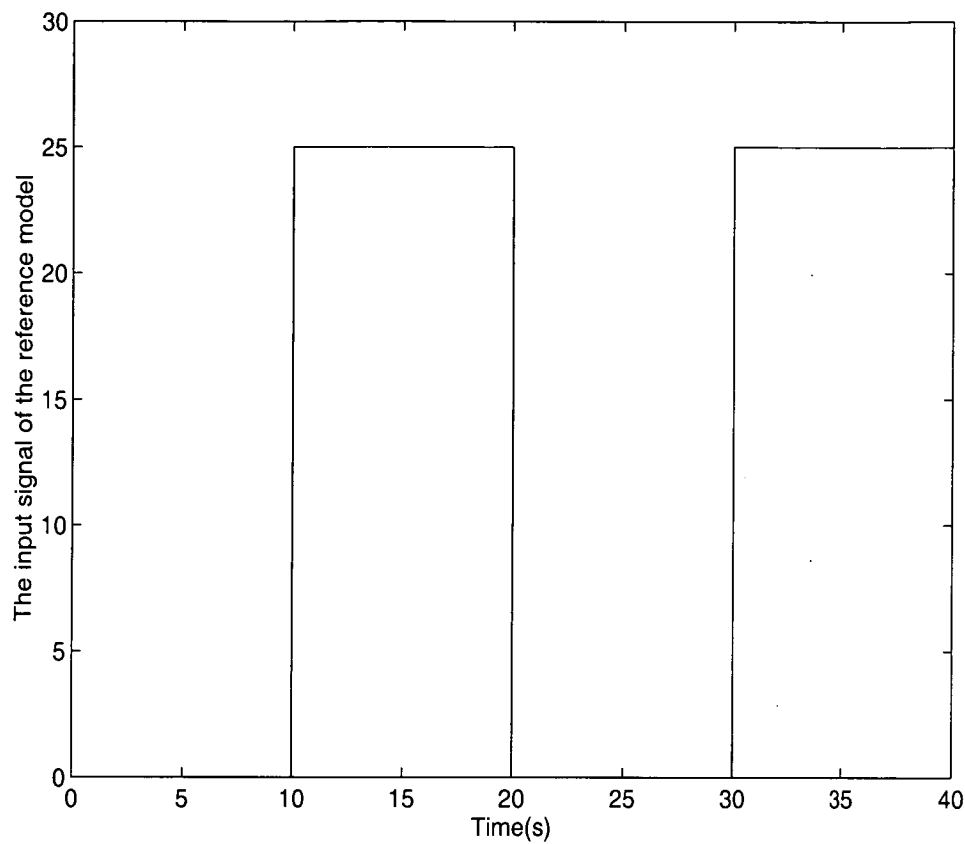


Fig. 5.2 The input signal of the reference model

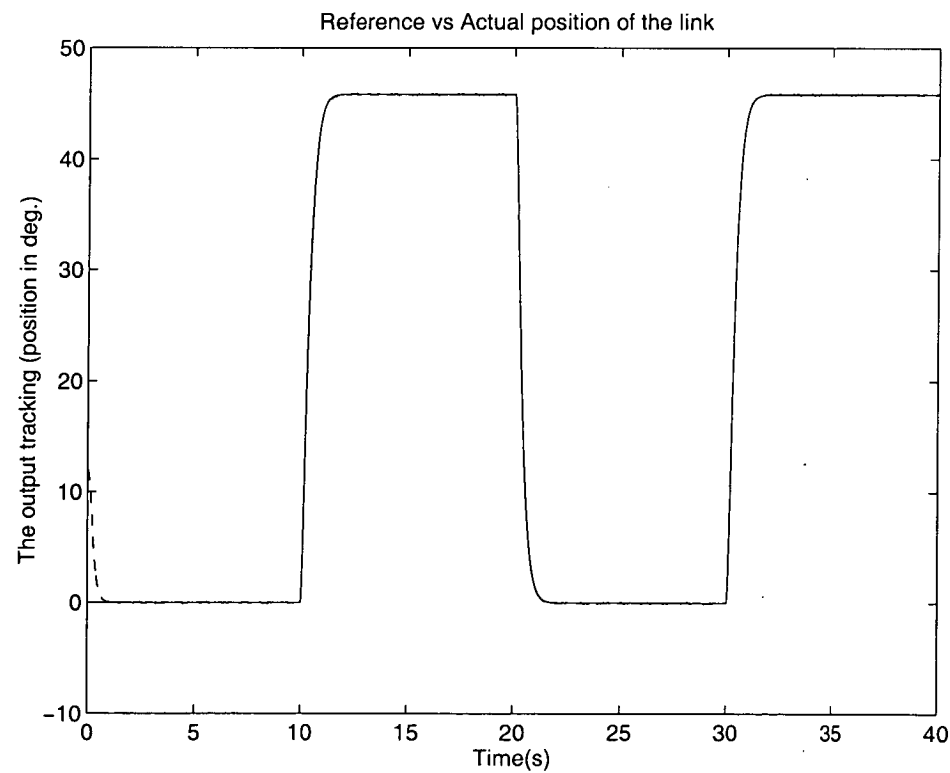


Fig. 5.3 The output tracking with conventional sliding mode controller

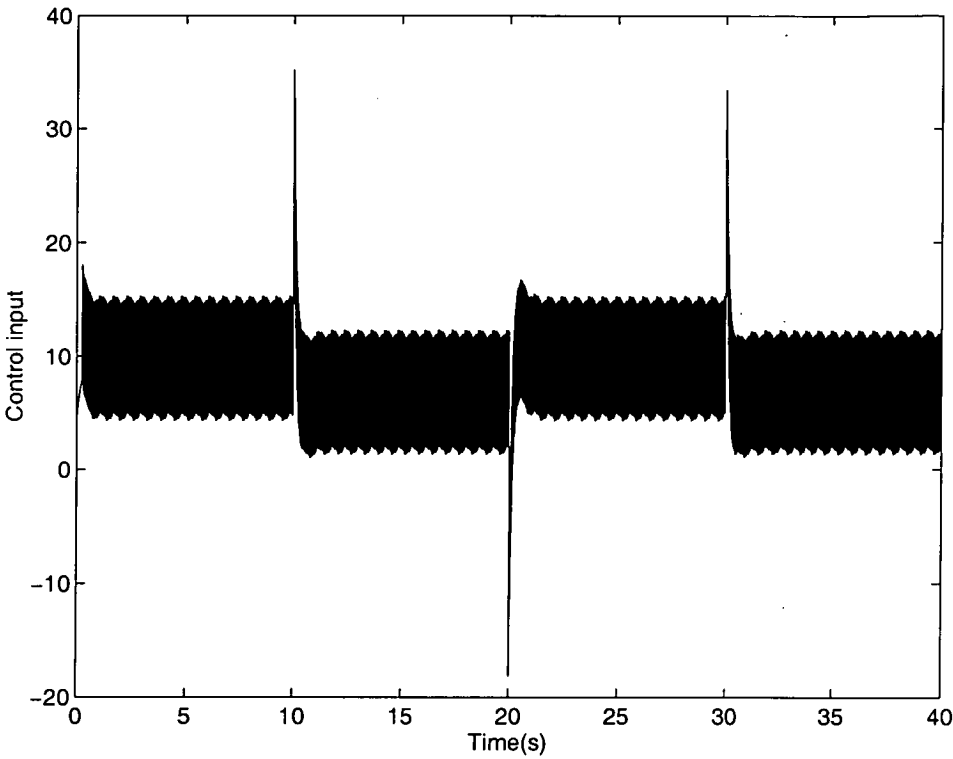


Fig.5. 4 The sliding mode control input without fuzzy tuning

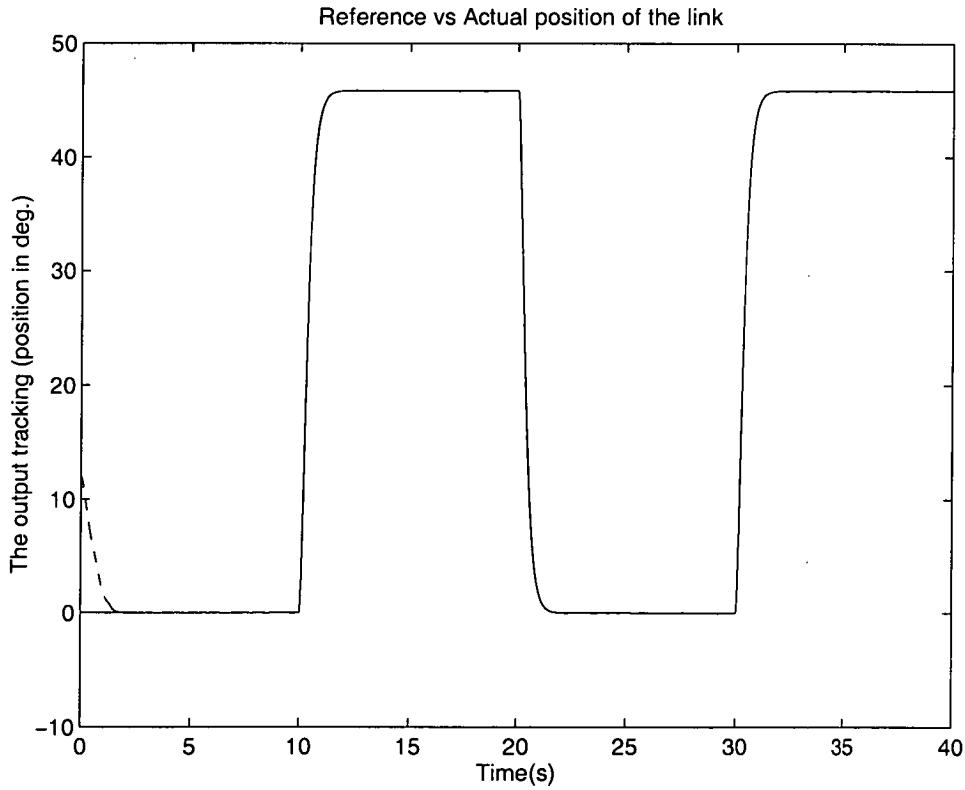


Fig.5. 5 The output tracking with fuzzy tuning

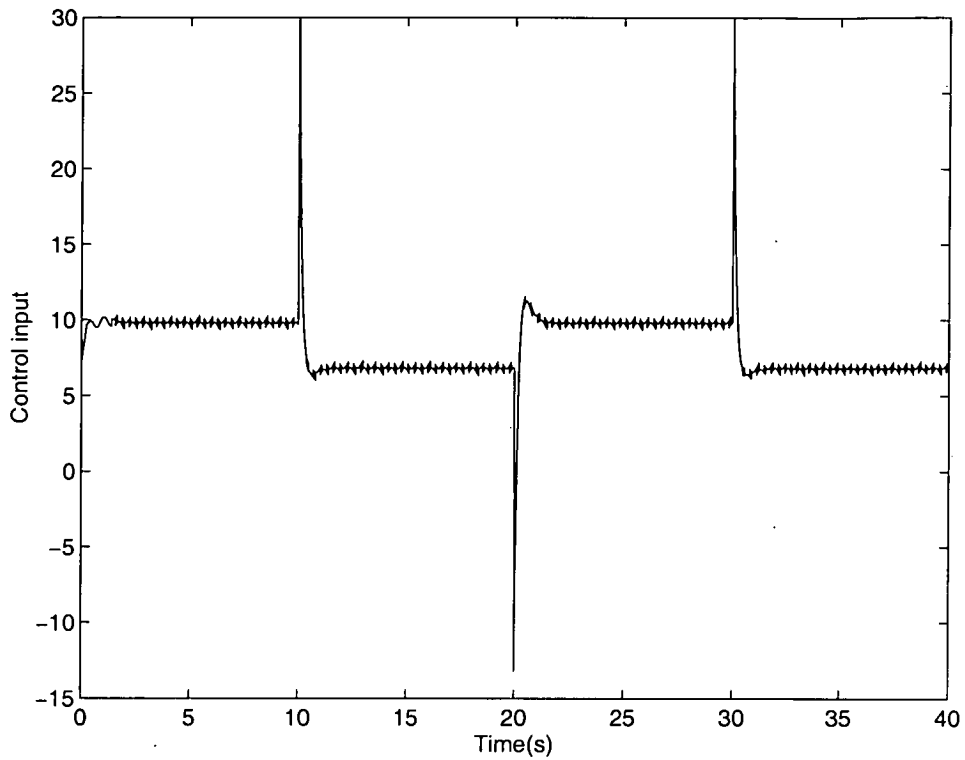


Fig. 5.6 The control input with fuzzy tuning sliding mode controller

5.5 Concluding remarks

In this chapter, Lyapunov stability theory and fuzzy logic technique are combined together to design sliding mode control systems. It is shown that a sliding mode is first designed to describe the desired system dynamics for the controlled system. A set of fuzzy rules are then used to adjust the parameters of the controller based on the Lyapunov function and its derivative. The sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering of the control signals, compared with the conventional sliding mode controllers. A simulation example using fuzzy sliding mode control for a one link robotic manipulator is given in support of the proposed control scheme.

Chapter 6

Robust Continuous Sliding Mode Control

6.1 Introduction

Sliding mode control (SMC), based on the theory of variable structure systems, first proposed by Emelyanov in 1967, has attracted a lot of research on control systems for the last two decades. Sliding mode control is a robust control technique that has been successfully used for the control of linear and nonlinear systems. In order to design sliding mode control systems, a switching surface or a sliding mode is defined first, then a sliding mode controller is designed to drive the system state variables to the switching surface. If the system state is above or below the switching surface, then the controller will drive the system state back to the surface. Once the system state reaches the switching surface, the system state will slide along the switching surface, and the dynamics of the controlled system is pre-determined by the sliding mode parameters and remains insensitive to variations of system parameters and external disturbances.

The main drawback of sliding mode control is the associated undesirable chattering problem. It not only increases wear of the final control element but also excites high frequency unmodelled plant dynamics (Slotine and Sastry, 1983). To reduce this unwanted control chattering, Slotine and Sastry (1983) proposed a 'boundary layer' approach which approximates the ideal relay characteristics used by sliding mode control by linear saturated amplifier characteristics. This boundary layer sliding mode control only guarantees that the state will converge to a band (boundary layer) centered on the sliding mode. It provides no guarantee that the state will converge exactly to the sliding mode, although the state can be brought arbitrarily close to the sliding mode by narrowing the boundary layer at the cost of higher control authority.

As an alternative, Zhou and Fisher (1992) proposed a continuous approach using the concept of boundary layer equivalence. The main disadvantage of this technique is that it tends to produce conservative designs. Because their control law is equivalent to a cubic feedback control, therefore, the control input may be very large and exceeds control element's physical limits.

Recently, Ha and Negnevitsky (1996) introduced a fuzzy tuning sliding mode control, it is shown that the control performance can be improved by fuzzy tuning.

In this chapter, we present a robust continuous sliding mode control scheme based on Lyapunov stability theory. The proposed controller consists of three components: equivalent control, continuous reaching mode control and robust control. It retains the positive properties of sliding mode control but without the disadvantage of control

chattering. As an example, the proposed control scheme is applied to the design of a robust tracking controller for a one-link robotic manipulator.

6.2 A robust continuous sliding mode control

Consider a linearised dynamic system described by

$$\dot{x} = (A + \Delta A)x + (B + \Delta B)u \quad (6.1)$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}$ is the control signal, $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^n$ are the nominal system matrices and in controllable form, ΔA and ΔB represent uncertainties. The following matching condition is used for further discussion (Man and Palaniswami, 1995):

$$\begin{aligned} \Delta A &= BH \\ \Delta B &= BE \end{aligned} \quad (6.2)$$

with

$$\begin{aligned} 0 &< \|H\| < H_0 \\ 0 &< |E| < E_0 < 1 \end{aligned}$$

Remark 6.1: Because (A, B) is in controllable form, therefore E is a scalar.

Sliding variable is defined by,

$$S = C^T x \quad (6.3)$$

where $C^T = [c_1 \ c_2 \ \dots \ c_n]$ is chosen such that the zeros of the polynomial $C^T x = 0$ are in the left half-plane and $C^T B \neq 0$.

A equivalent control is given by $\dot{S} = 0$ or:

$$C^T \dot{x} = C^T (Ax + Bu_{eq}) = 0 \quad (6.4)$$

that is

$$u_{eq} = -K_{eq} x = -(C^T B)^{-1} C^T A x \quad (6.5)$$

A continuous reaching mode controller can be designed by following reaching law:

$$\dot{S} = -Q \operatorname{erf}\left(\frac{S}{\delta}\right) \quad (6.6)$$

where Q is a positive reaching law parameter, δ is a positive number, and $\operatorname{erf}(x)$ is a standard error function. The continuous sliding mode controller can be easily derived as

$$u_c = -(C^T B)^{-1} Q \operatorname{erf}\left(\frac{S}{\delta}\right) \quad (6.7)$$

In order to deal with system uncertainties, a robust controller is designed as follows:

$$u_r = -\frac{SC^T B}{\|SC^T B\|} \frac{1}{1 - E_0} (H_0 \|x\| + E_0 \|K_{eq} x\|) \quad (6.8)$$

Hence, the robust continuous sliding mode controller is represented by

$$u = u_{eq} + u_c + u_r \quad (6.9)$$

Theorem 6.1: *If the controller is given by (6.9), then the system state x reaches origin as time tends to infinity.*

Proof: Define a Lyapunov function

$$V = \frac{1}{2} S^2 \quad (6.10)$$

Then

$$\begin{aligned} \dot{V} &= S\dot{S} \\ &= SC^T \dot{x} \\ &= SC^T [(A + \Delta A)x + (B + \Delta B)u] \\ &= SC^T [Ax + BHx + B(1 + E)u] \\ &= SC^T Ax + SC^T BHx \\ &\quad + (1 + E)[-SC^T Ax - SQ\text{erf}(\frac{S}{\delta}) - \|SC^T B\| \frac{1}{1 - E_0} (H_0 \|x\| + E_0 \|K_{eq} x\|)] \\ &< -ESC^T Ax + SC^T BHx - \|SC^T B\| \frac{1 - |E|}{1 - E_0} (H_0 \|x\| + E_0 \|K_{eq} x\|) \\ &= -ESC^T BK_{eq} x + SC^T BHx - \|SC^T B\| \frac{1 - |E|}{1 - E_0} (H_0 \|x\| + E_0 \|K_{eq} x\|) \\ &< 0 \quad \text{if } S \neq 0 \end{aligned}$$

QED

6.3 A simulation example

To illustrate the proposed robust continuous sliding mode control scheme in this chapter, a simulation example is carried out for a one-link robotic manipulator. The dynamic equation of the one-link robotic manipulator is given by

$$ml^2\ddot{\theta} + d\dot{\theta} + mgl \cos(\theta) = u \quad (6.11)$$

with

$$\begin{aligned} m &= 1\text{kg} - \text{payload,} \\ l &= 1\text{m} - \text{length of the link,} \\ g &= 9.81\text{m/s}^2 - \text{gravitational constant,} \\ d &= 1\text{kgm}^2/\text{s} - \text{damping factor,} \\ u &= \text{control variable (kgm}^2/\text{s}^2\text{).} \end{aligned}$$

In Chapter 4, a fuzzy modelling of a nonlinear system is obtained by linearising procedures and the following fuzzy inference rules,

$$R^i: \quad \text{IF} \quad x_1 \text{ is } F_1^i \text{ AND } \dots x_n \text{ is } F_n^i$$

THEN

$$\tilde{\dot{x}} = A_i \tilde{x} + B_i \tilde{u}$$

$$\tilde{u} = u - u_i \quad (6.12)$$

$$i = 1, 2, \dots, l$$

where R^i denotes the i -th fuzzy inference rule, l the number of inference rules,

F_j^i ($j = 1, 2, \dots, n$) are fuzzy sets, $\tilde{x} = x - x_d$ is the tracking error of the system with

desired trajectory x_d , and $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$.

Let $\mu_i(x)$ be the normalized membership function of the inferred fuzzy set F^i where

$$F^i = \bigcap_{j=1}^n F_j^i \quad (6.13)$$

and

$$\sum_{i=1}^l \mu_i(x) = 1 \quad (6.14)$$

By using a standard fuzzy inference method, that is, using a singleton fuzzifier, product fuzzy inference and centre-average defuzzifier, the following global tracking error fuzzy model for the controlled nonlinear system can be obtained,

$$\dot{\tilde{x}} = (A + \Delta A)\tilde{x} + (B + \Delta B)\tilde{u} \quad (6.15)$$

$$\tilde{u} = u - u_d \quad (6.16)$$

where

$$A = \sum_{i=1}^l \mu_i A_i \quad B = \sum_{i=1}^l \mu_i B_i \quad u_d = \sum_{i=1}^l \mu_i u_i$$

ΔA and ΔB represent system uncertainties.

Assuming we are interested in the dynamics of the system in the range of $[-90^\circ, 90^\circ]$, then the fuzzy model can be obtained by linearising the nonlinear equation (6.11) over a number of points, such as $0^\circ, \pm 60^\circ, \pm 90^\circ$. The following fuzzy model has been obtained.

R^1 : IF x_1 is about -90° ,
 THEN $\dot{\tilde{x}} = A_1 \tilde{x} + B_1 \tilde{u}$, $\tilde{u} = u - u_1$

R^2 : IF x_1 is about -60° ,
 THEN $\dot{\tilde{x}} = A_2 \tilde{x} + B_2 \tilde{u}$, $\tilde{u} = u - u_2$

R^3 : IF x_1 is about 0° ,
 THEN $\dot{\tilde{x}} = A_3 \tilde{x} + B_3 \tilde{u}$, $\tilde{u} = u - u_3$

R^4 : IF x_1 is about 60° ,
 THEN $\dot{\tilde{x}} = A_4 \tilde{x} + B_4 \tilde{u}$, $\tilde{u} = u - u_4$

R^5 : IF x_1 is about 90° ,
 THEN $\dot{\tilde{x}} = A_5 \tilde{x} + B_5 \tilde{u}$, $\tilde{u} = u - u_5$

with

$$\begin{aligned} A_1 &= \begin{bmatrix} 0 & 1 \\ -9.81 & -1 \end{bmatrix}, & B_1 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_1 &= 0, \\ A_2 &= \begin{bmatrix} 0 & 1 \\ -8.5 & -1 \end{bmatrix}, & B_2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_2 &= 4.9, \\ A_3 &= \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & B_3 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_3 &= 9.81, \\ A_4 &= \begin{bmatrix} 0 & 1 \\ 8.5 & -1 \end{bmatrix}, & B_4 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_4 &= 4.9, \\ A_5 &= \begin{bmatrix} 0 & 1 \\ 9.81 & -1 \end{bmatrix}, & B_5 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & u_5 &= 0. \end{aligned}$$

Fig. 6.1 shows the membership functions. Choosing $Q = 10, \delta = 0.3, H_0 = 2$, the simulation results are shown in Fig. 6.2 –6.8. Fig. 6.2 shows the output tracking. Fig. 6.3 shows the control signal, it is observed that no chattering occurred with the proposed robust control scheme. Fig. 6.4 shows the tracking error. Fig. 6.5-6.7 show the three control components: equivalent control, continuous reaching mode control, and robust control, respectively. Fig. 6.8 shows the phase portrait.

Choosing $Q = 10$, $\delta = 1$, $H_0 = 2$, the simulation results are shown in Fig. 6.9 –6.15. Fig. 6.9 shows the output tracking. Fig. 6.10 shows the control signal, it is observed that no chattering occurred with the proposed robust control scheme. Fig. 6.11 shows the tracking error. Fig. 6.12-6.14 show the three control components: equivalent control, continuous reaching mode control, and robust control, respectively. Fig. 6.15 shows the phase portrait.

Choosing $Q = 10$, $H_0 = 2$, and using discontinuous control $u_d = -Q \operatorname{sgn}(S)$ instead of u_c , then the simulation results are shown in Fig. 6.16 –6.22. Fig. 6.16 shows the output tracking. Fig. 6.17 shows the control signal, it is observed that severe chattering occurred with the discontinuous control scheme. Fig. 6.18 shows the tracking error. Fig. 6.19-6.21 show the three control components: equivalent control, discontinuous control, and robust control, respectively. Fig. 6.22 shows the phase portrait.

6.4 Concluding remarks

In this chapter we developed a robust continuous sliding mode control scheme for linear systems with uncertainties. The controller consists of three components: equivalent control, continuous reaching mode control and robust control. It retains the positive properties of sliding mode control but without the disadvantage of control chattering. The proposed control scheme has been applied to the tracking control of a one-link robotic manipulator by fuzzy modelling of the nonlinear system.

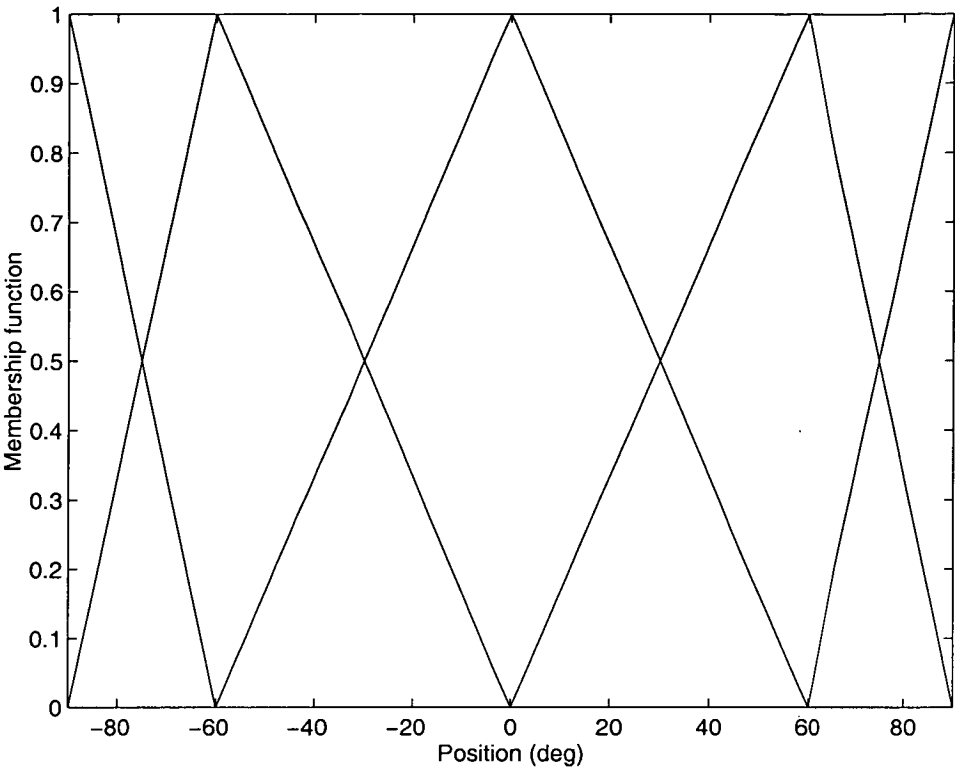


Fig.6.1 Membership functions for positions

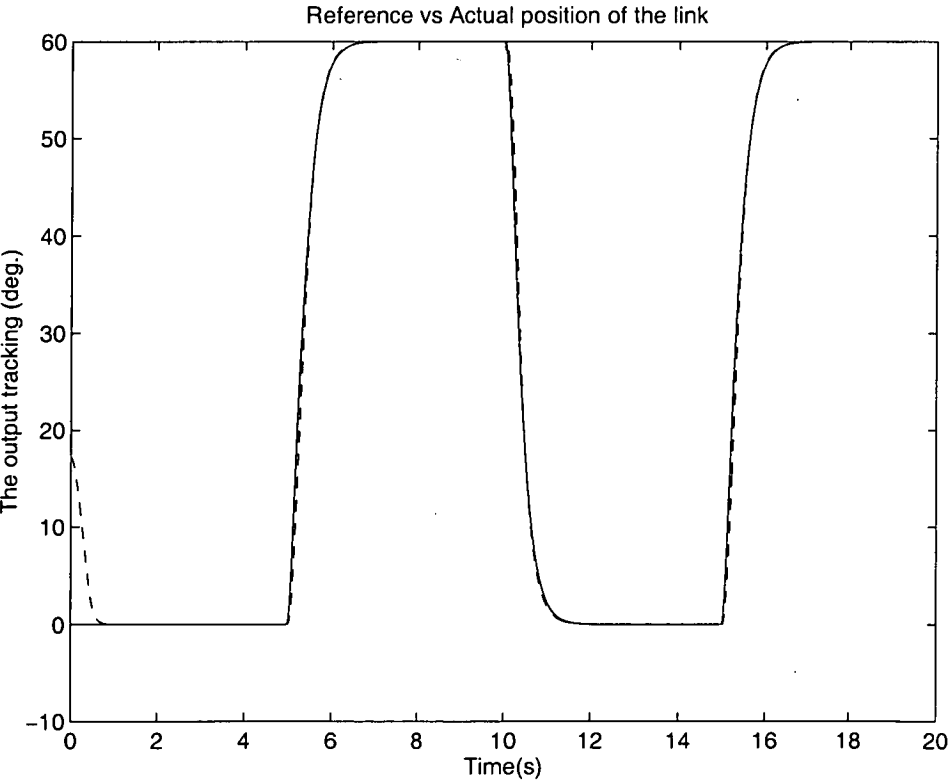


Fig.6.2 Output tracking performance ($\delta = 0.3$)

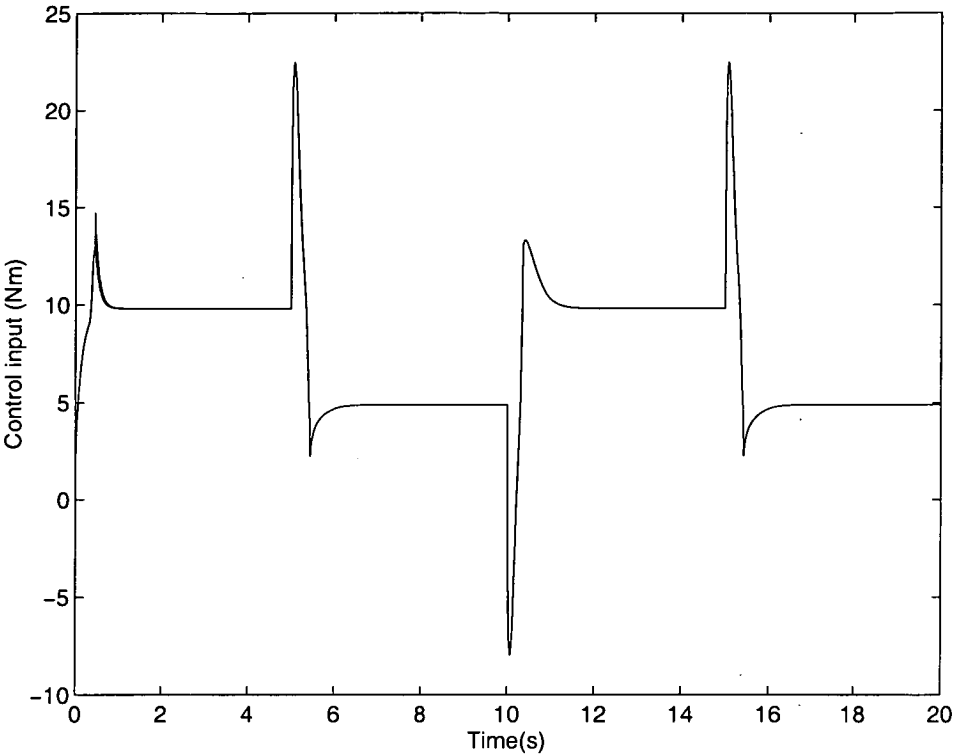


Fig.6.3 Control input ($\delta = 0.3$)

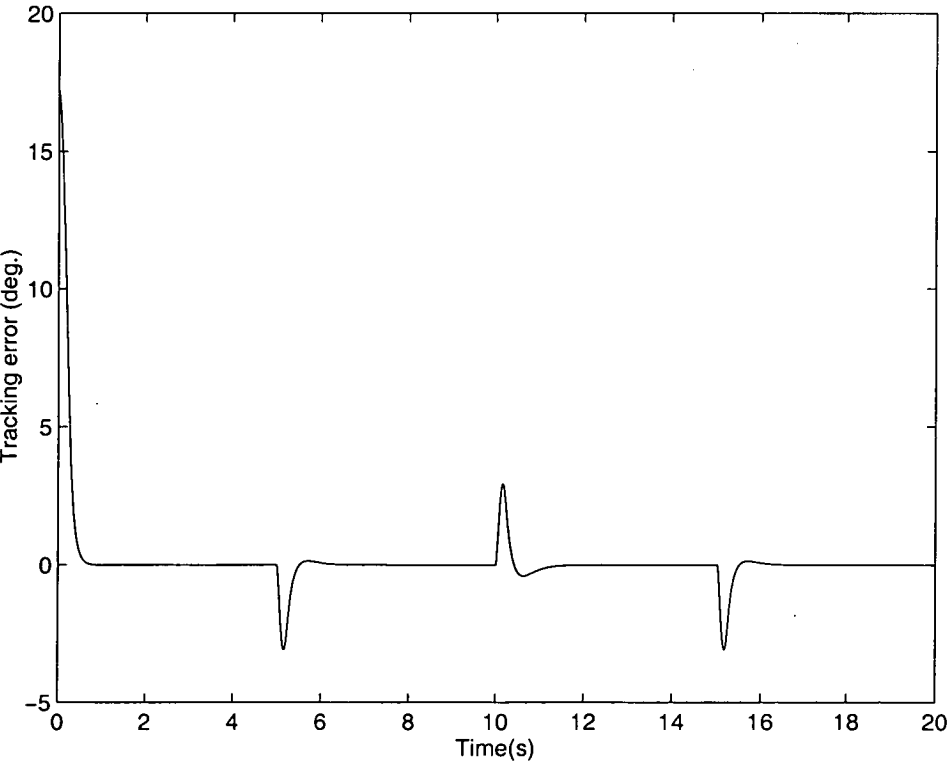


Fig.6.4 Tracking error ($\delta = 0.3$)

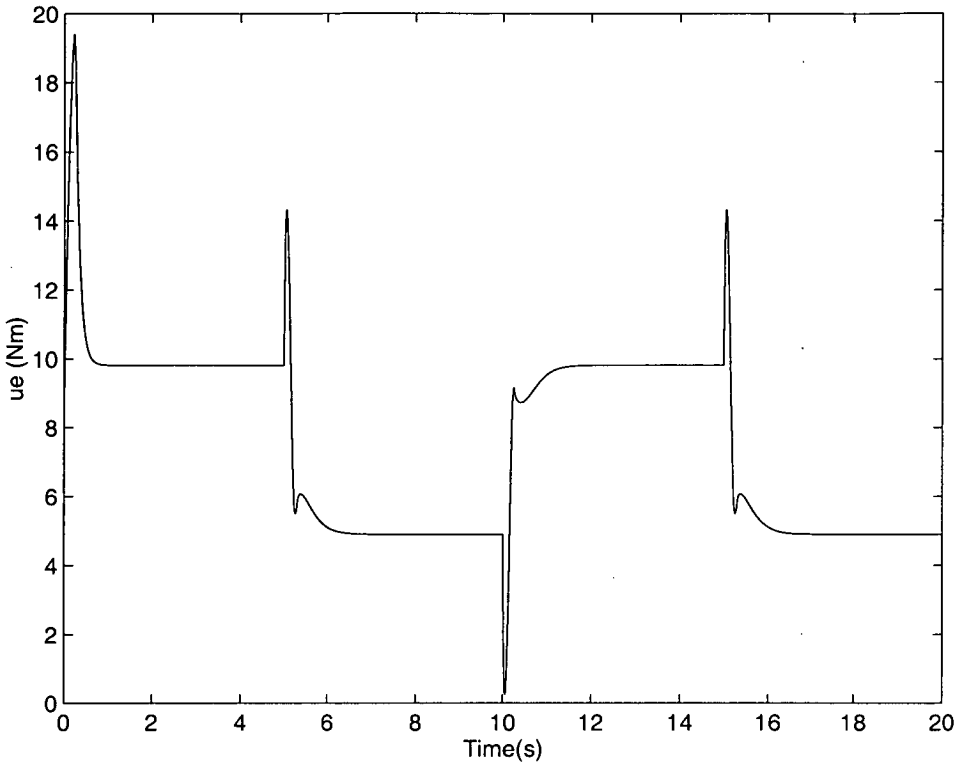


Fig. 6.5 Control component u_e ($\delta = 0.3$)

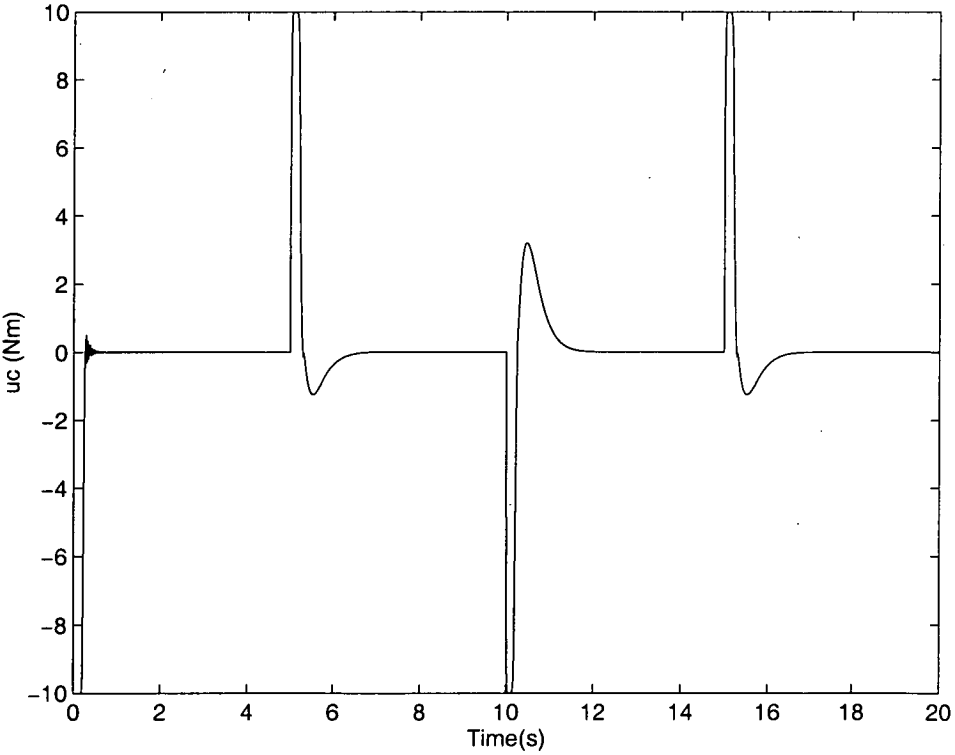


Fig. 6.6 Control component u_c ($\delta = 0.3$)

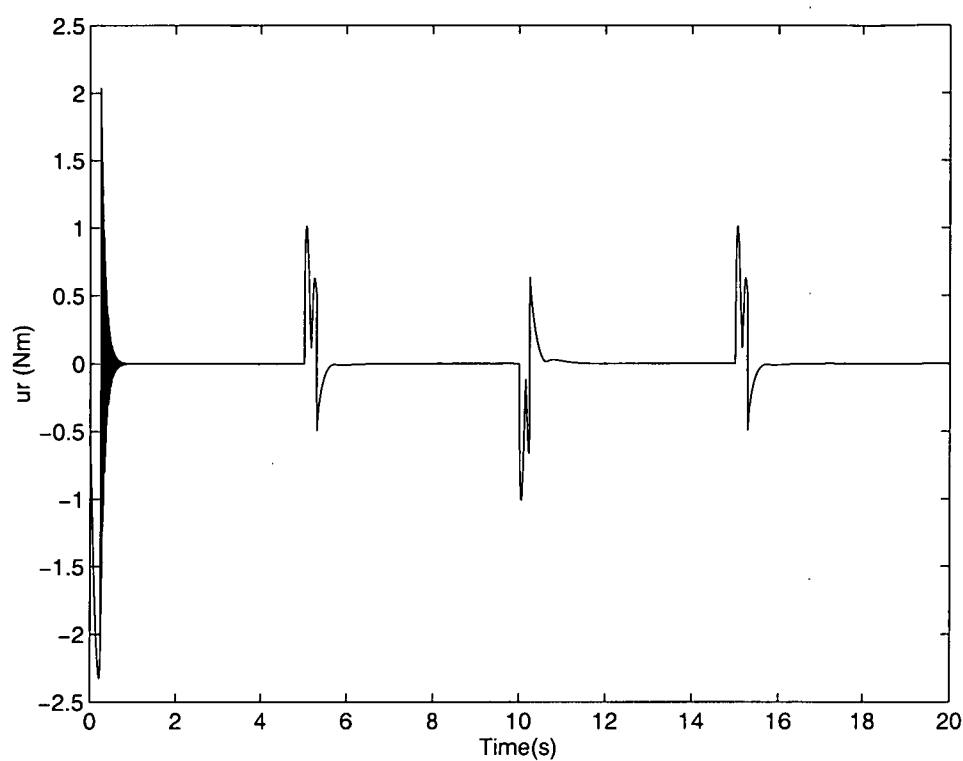


Fig. 6.7 Control component u_r ($\delta = 0.3$)

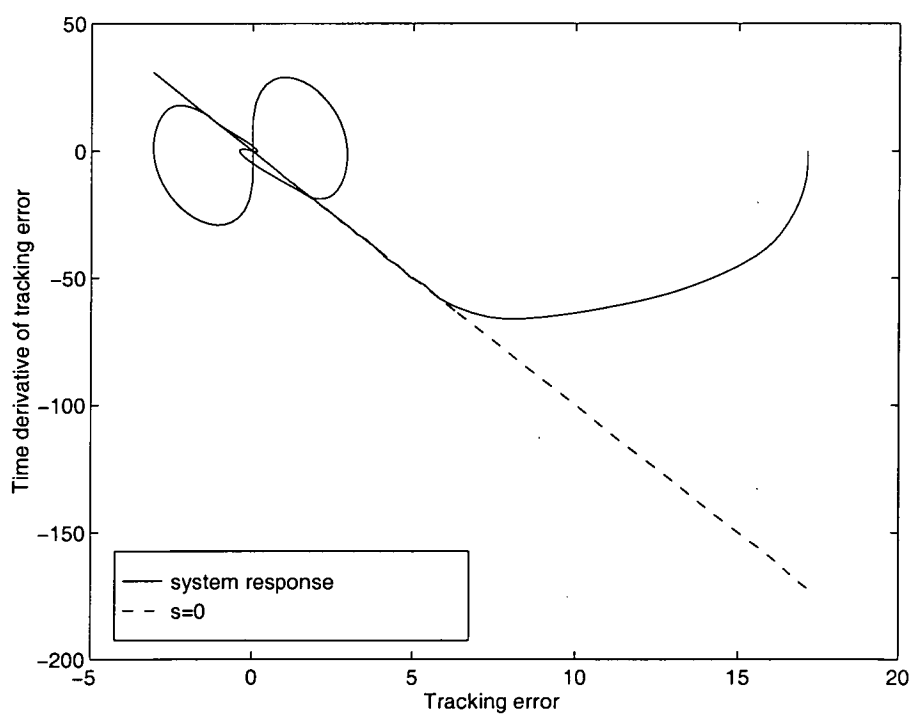


Fig. 6.8 Phase portrait ($\delta = 0.3$)

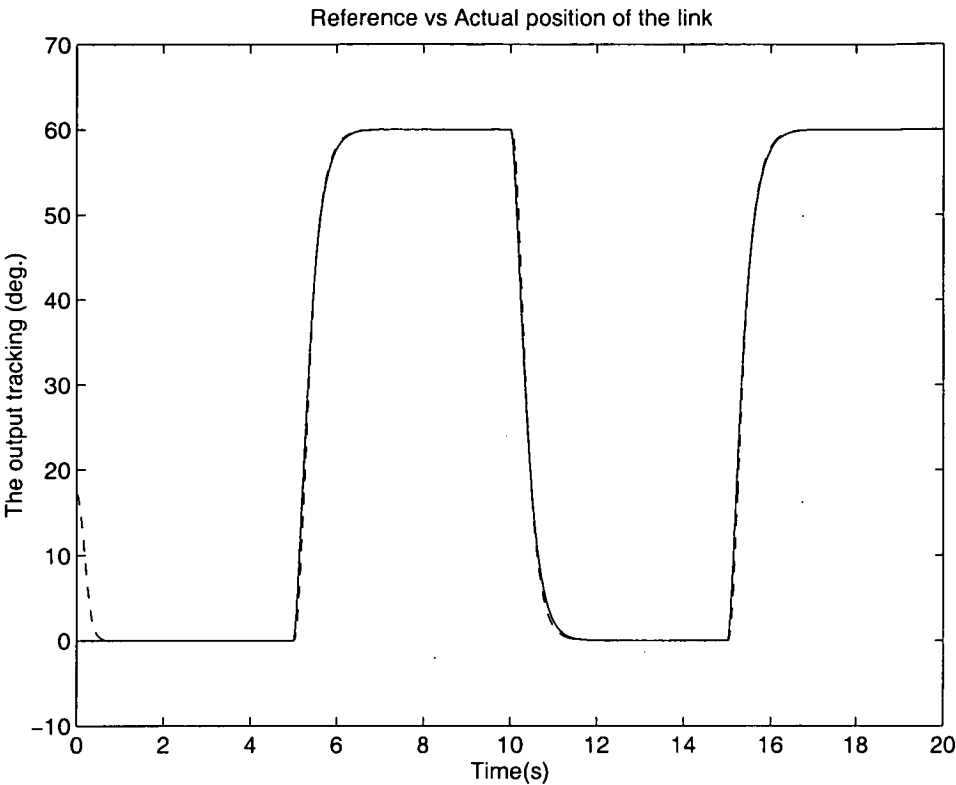


Fig. 6.9 Output tracking ($\delta = 1$)

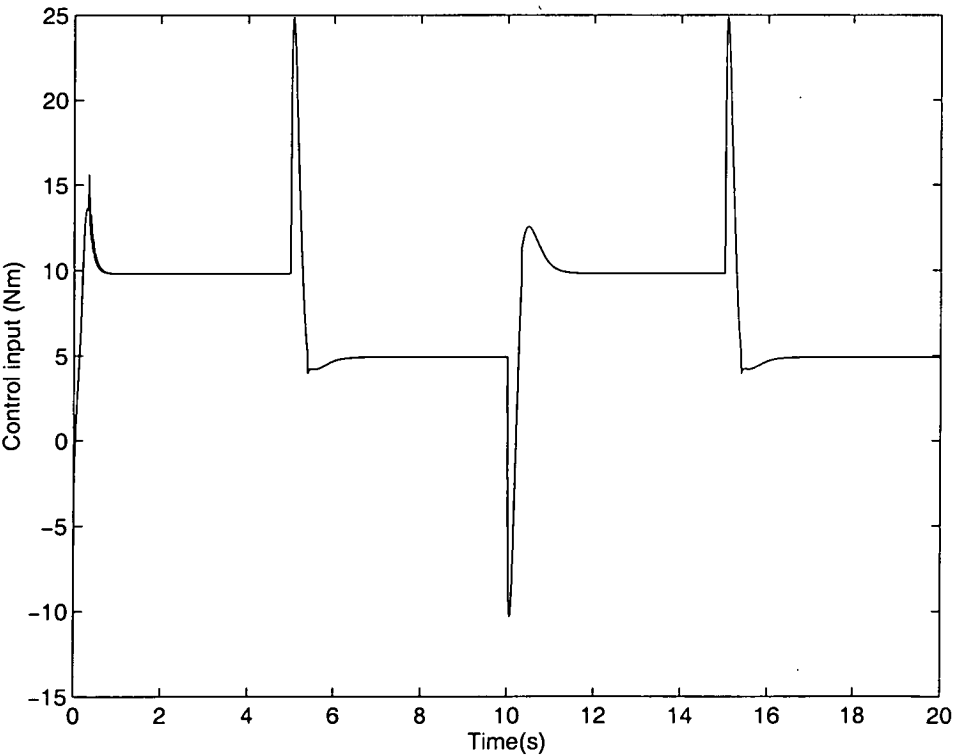


Fig. 6.10 Control input ($\delta = 1$)

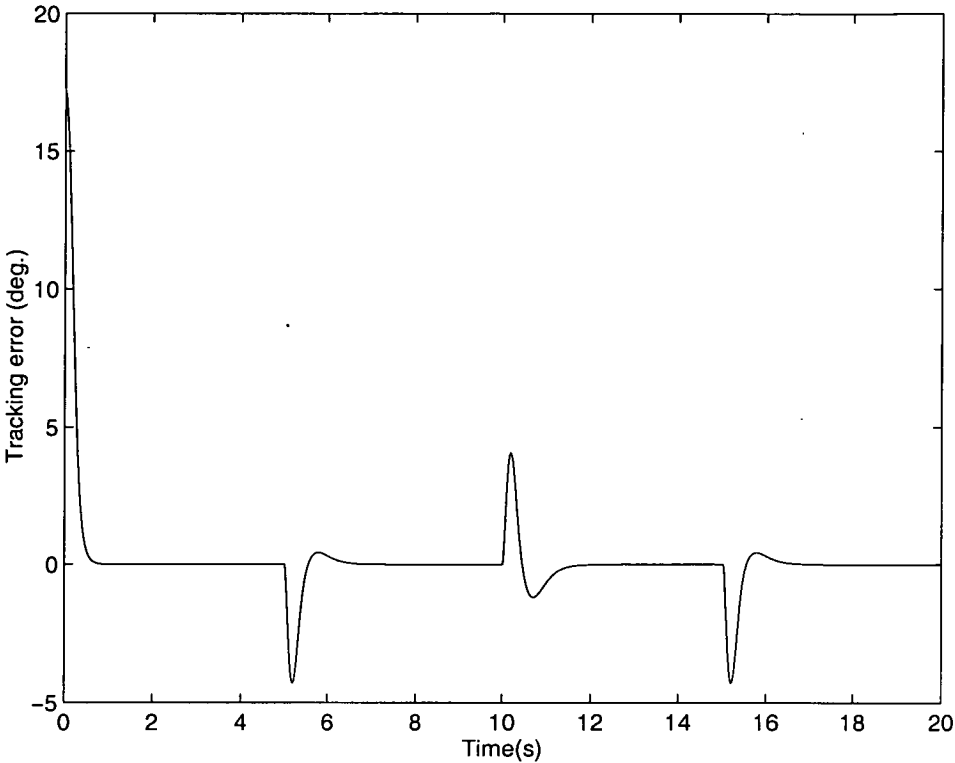


Fig. 6.11 Tracking error ($\delta = 1$)

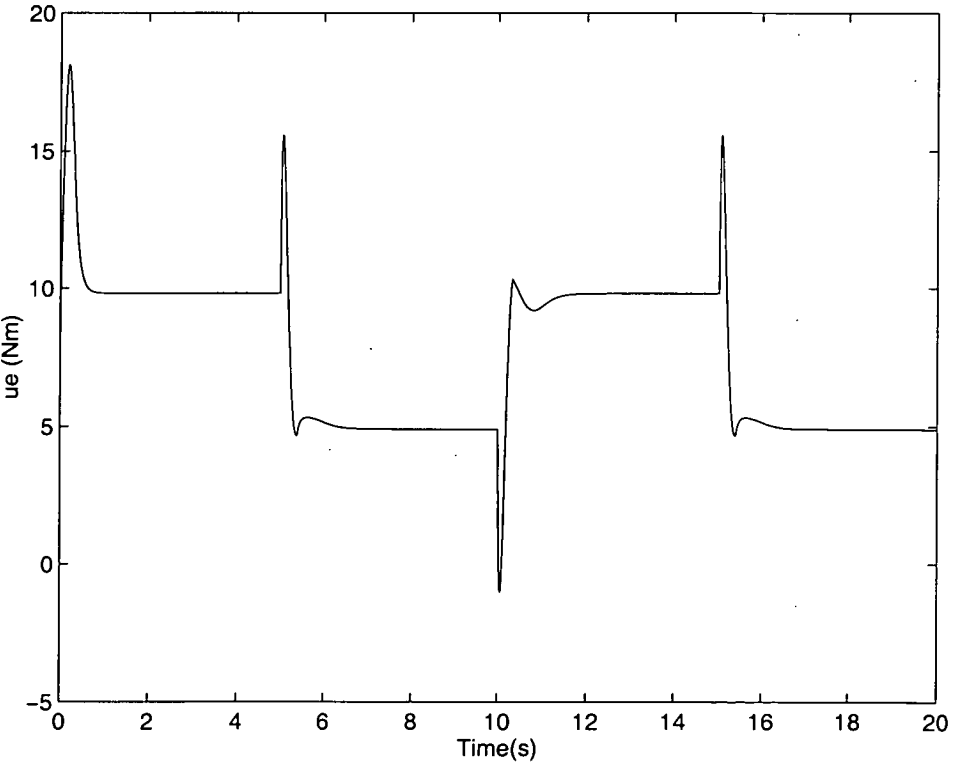
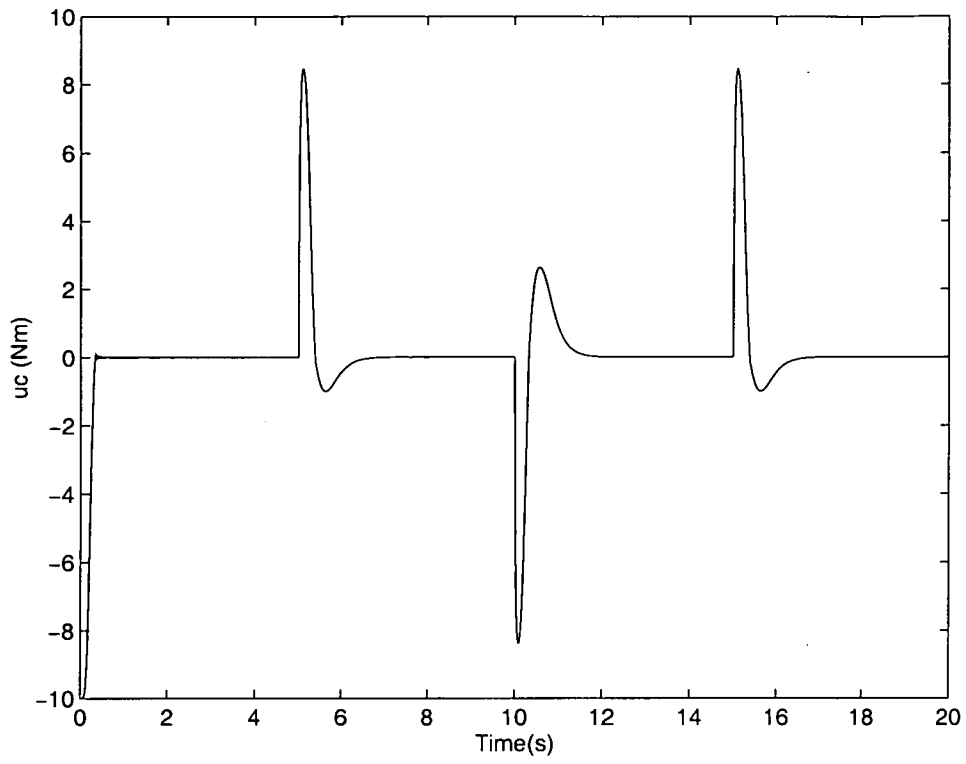
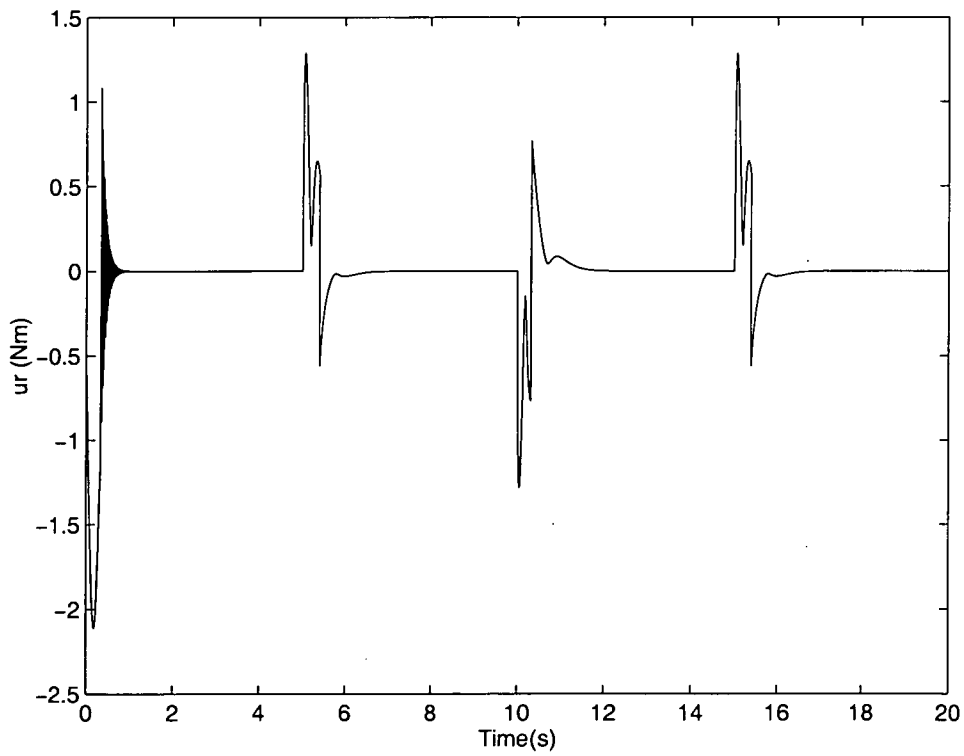


Fig. 6.12 Control component u_e ($\delta = 1$)

Fig. 6.13 Control component u_c ($\delta = 1$)Fig. 6.14 Control component u_r ($\delta = 1$)

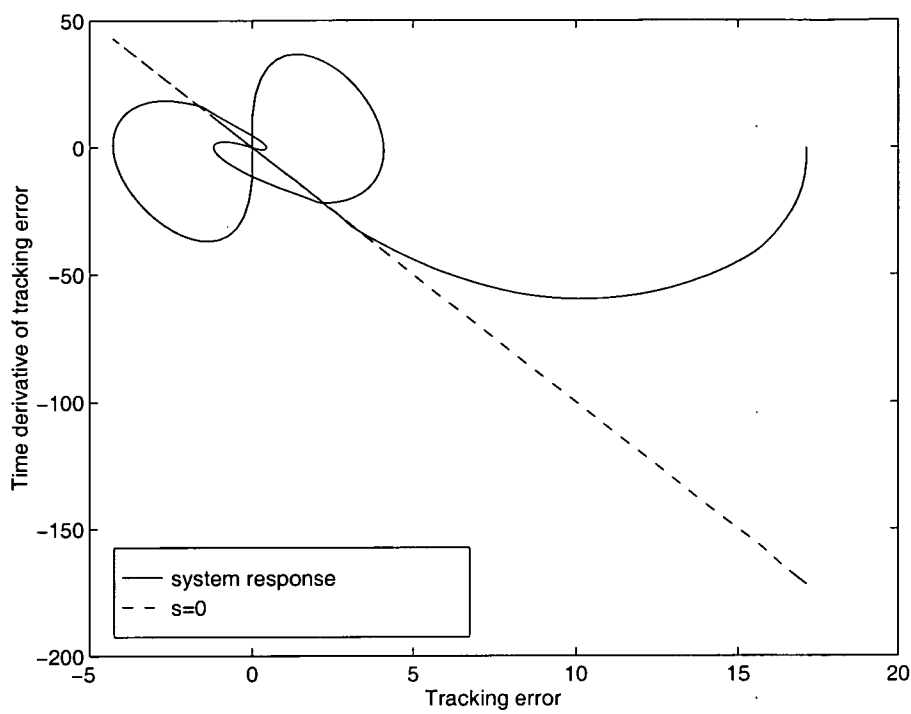


Fig. 6.15 Phase portrait ($\delta = 1$)

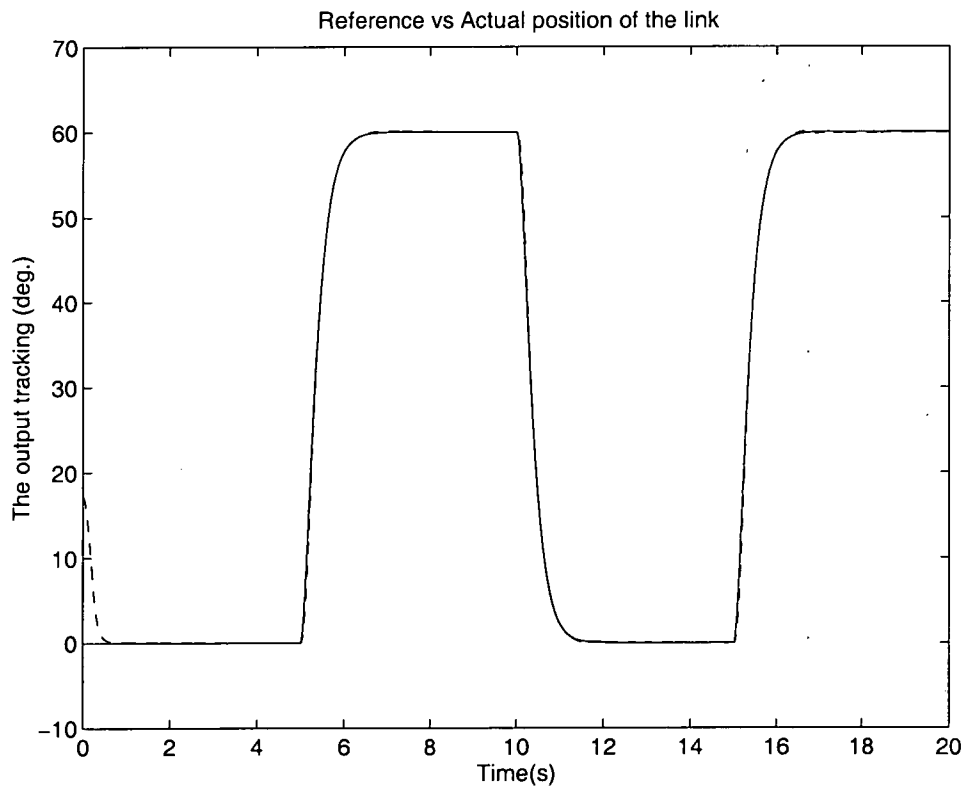


Fig. 6.16 Output tracking (sgn(s))

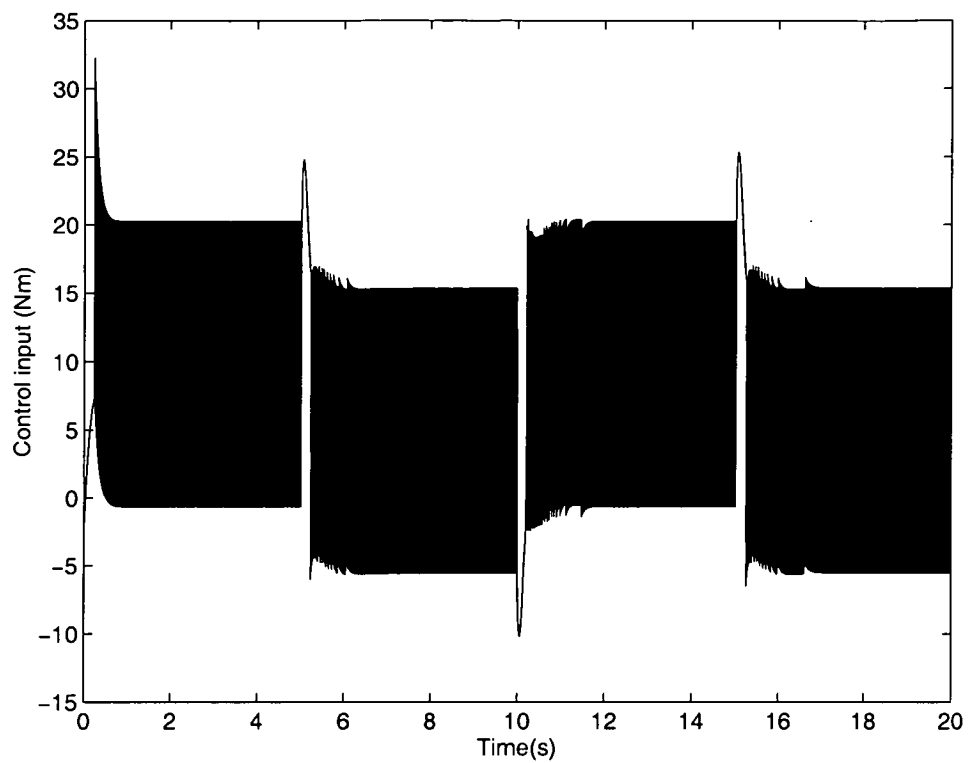


Fig. 6.17 Control input (sgn(s))

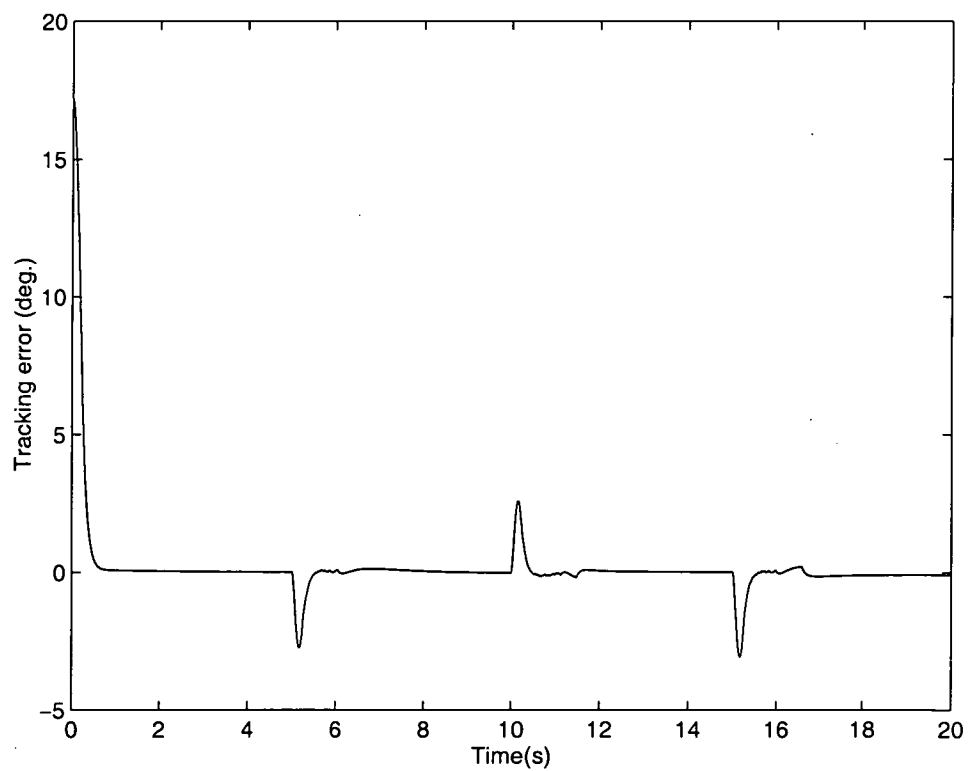


Fig. 6.18 Tracking error (sgn(s))

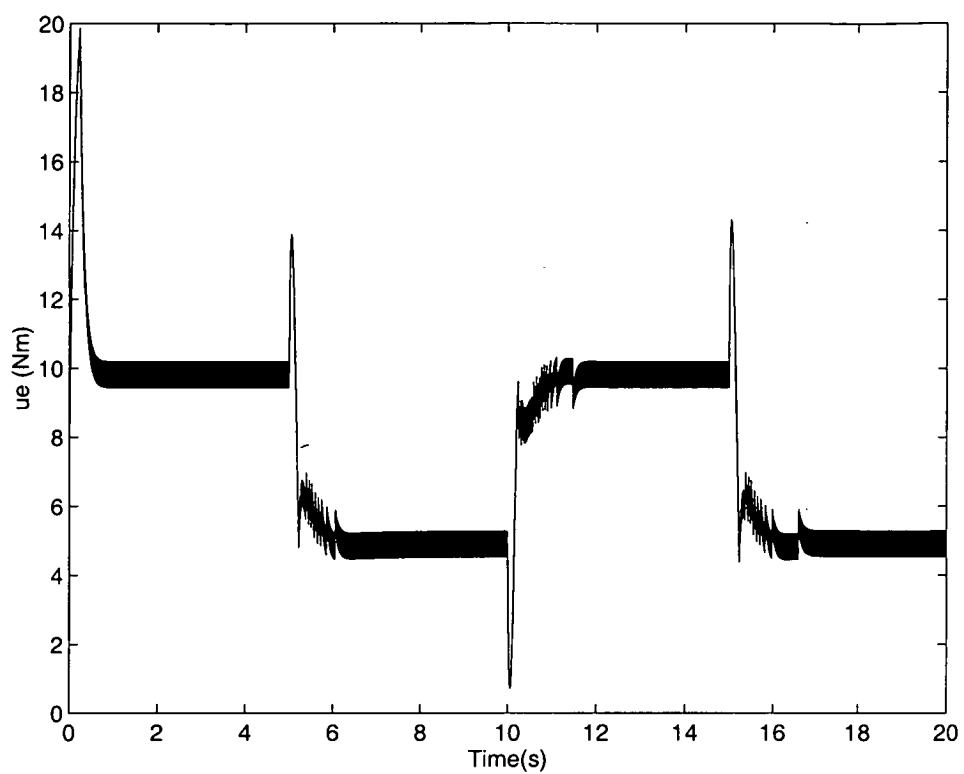


Fig. 6.19 Control component u_e (sgn(s))

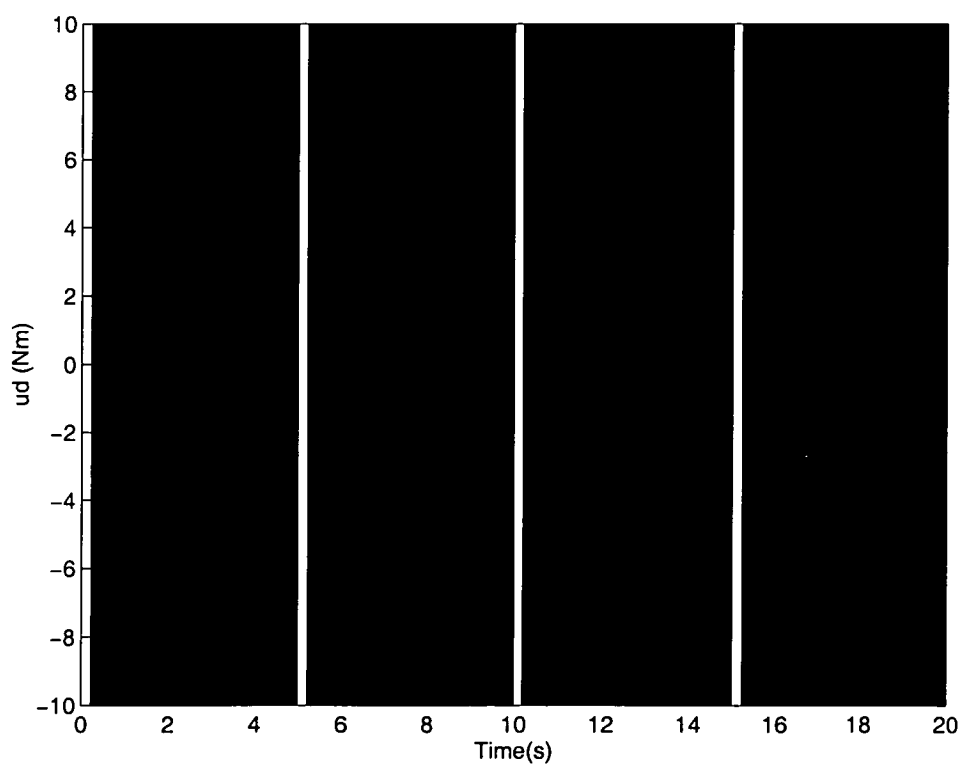


Fig. 6.20 Control component u_d (sgn(s))

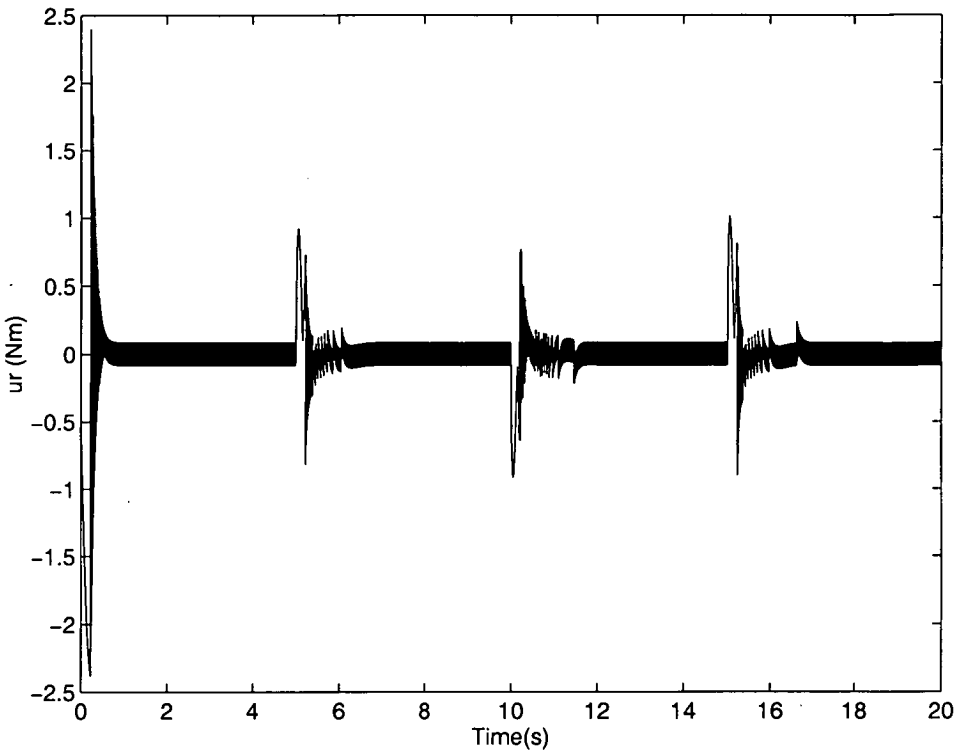


Fig. 6.21 Control component u_r (sgn(s))

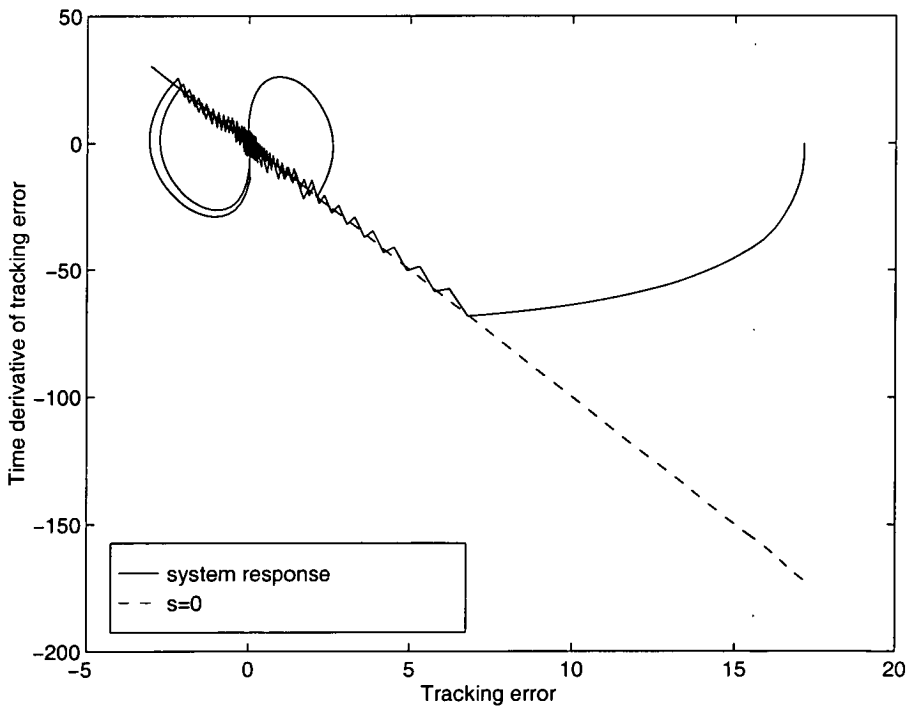


Fig. 6.22 Phase portrait (sgn(s))

Chapter 7

Fuzzy Adaptive Sliding Mode Control

7.1 Introduction

Adaptive control of mechanical manipulators has been studied extensively by many researchers using various methodologies (Craig and Sastry, 1986; Slotine and Sastry, 1984; Balestrino et al., 1979; Young, 1978), such as hyperstability theory, Lyapunov stability method, self-tuning regulator control and sliding mode control (SMC). Among the various methodologies, the SMC demonstrates its good robustness and low computational cost. SMC is a discontinuous feedback control that switches the system control structure during the evolution of the system state so that the system states remain in a prescribed subspace. A SMC controller for robot manipulators suggests that each manipulator link matches a first order sliding motion such that when the system state reaches the sliding mode, it chatters along the sliding mode and exhibits sensitivity to the system parameter variations and external disturbances (Slotine and Li, 1987).

One of the well known results in SMC design for trajectory tracking of robotic manipulators was obtained by Slotine and Li (1987, 1991). Their adaptive SMC is based on the linear parametrization approach for the robotic systems which do not involve any uncertainties and disturbances. The validity of the control algorithm is based on the assumption that the system parameters must be constant and the desired trajectory should be sufficiently “rich” to guarantee parameter convergence. To overcome the robustness problem of Slotine and Li's controller, Su and Leung (Su and Leung, 1993) have proposed a modification that employs the bound estimation algorithms in the controller to give rise to a robust adaptive SMC controller without *a priori* knowledge of upper bounds of the system parameters. This controller is able to handle large and time varying variations in the parameters. However, the problem is that their control torques designed incur very large chattering which is not desirable in practice, especially in mechanical systems.

In chapter 5, a fuzzy tuning algorithm was employed for conventional sliding mode control to diminish undesired control chattering. In this chapter, a fuzzy tuning adaptive sliding mode control is proposed for trajectory tracking of rigid robotic manipulators with uncertainties to achieve robustness as well as desired qualities such as minimal chattering and small control torques.

This chapter is organised as follows: In Section 7.2, a fuzzy tuning adaptive sliding mode controller is presented. Section 7.3 describes the experimental results for confirmation of the theoretical results. Section 7.4 gives concluding remarks.

7.2 Robust adaptive SMC with fuzzy tuning

Consider the dynamics of the n -joint robotic manipulators represented by the following second order nonlinear vector differential equation

$$H(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau(t) + \varepsilon(t) \quad (7.1)$$

where $q(t)$ is the $nx1$ vector of joint angular positions, $H(q)$ is the nxn symmetric positive definite inertia matrix, $C(q, \dot{q})\dot{q}$ is the $nx1$ vector containing Coriolis, centrifugal forces, $g(q)$ represents the derivation of the manipulators potential energy, $\tau(t)$ is the $nx1$ vector of applied joint torques which are actually control inputs, and $\varepsilon(t)$ is the $nx1$ bounded input disturbances vector.

We assume that the robotic manipulators have uncertainties, i.e.

$$H = H_0 + \Delta H \quad (7.2)$$

$$C = C_0 + \Delta C \quad (7.3)$$

$$g = g_0 + \Delta g \quad (7.4)$$

where H_0, C_0, g_0 are the nominal part and $\Delta H, \Delta C, \Delta g$ are the uncertain part. The uncertain components represent bounded small possible time varying, unmodelled dynamics or load changes. Substituting the expressions (7.2)-(7.4) into the dynamics (7.1) gives rise to

$$H_0(q)\ddot{q} + C_0(q, \dot{q})\dot{q} + g_0(q) = \tau(t) + \rho(t) \quad (7.5)$$

with

$$\rho(t) = \varepsilon(t) - \Delta H\ddot{q} - \Delta C(q, \dot{q})\dot{q} - \Delta g(q) \quad (7.6)$$

representing all the uncertain terms. When $\rho(t)=0$, we call the dynamics as the “nominal dynamics” of the robotic manipulators.

Let the parameters to be estimated be represented by,

$$a = a_0 + \Delta a \quad (7.7)$$

The sliding function is

$$s = \ddot{\tilde{q}} + \Lambda \tilde{q} = \dot{q} - \dot{q}_r \quad (7.8)$$

where $\dot{q}_r = \dot{q}_d - \Lambda \tilde{q}$, and q_d is the desired trajectory while Λ is a positive constant matrix. We have the following result:

Theorem 5.1: *The control torques and the adaptive learning laws given below, ensure convergence of the sliding function s to zero:*

$$\tau = \tau_1 + \tau_2 \quad (7.9-a)$$

$$\tau_1 = Y\hat{a} - Ks, \quad (7.9-b)$$

$$\tau_2 = -Q \operatorname{sgn}(s) \quad (7.9-c)$$

$$\dot{\hat{a}} = -\Gamma Y^T s, \quad (7.10)$$

where Y is the regressor matrix satisfying

$$H_0 \ddot{q}_r + C_0 \dot{q}_r + g_0 = Y a_0$$

K is a positive definite constant matrix, \hat{a} the estimated parameter, $\tilde{a} = \hat{a} - a_0$, Q the upper bound matrix for $\rho(t)$, and Γ a constant positive definite matrix.

Proof: To analyse the convergence and stability, we consider the Lyapunov function

$$V(t) = \frac{1}{2} (s^T H_0 s + \tilde{a}^T \Gamma^{-1} \tilde{a}) \quad (7.11)$$

The derivative of $V(t)$ is

$$\dot{V}(t) = s^T H_0 \dot{s} + \frac{1}{2} s^T \dot{H}_0 s + \tilde{a}^T \Gamma^{-1} \dot{\tilde{a}} \quad (7.12)$$

Substituting $\dot{s} = \ddot{q} - \ddot{q}_r$ and $H_0 = H - \Delta H$ yields

$$\dot{V}(t) = s^T (H\ddot{q} - \Delta H\ddot{q} - H_0 \ddot{q}_r) + \frac{1}{2} s^T \dot{H}_0 s + \tilde{a}^T \Gamma^{-1} \dot{\tilde{a}} \quad (7.13)$$

Since $H\ddot{q} + C\dot{q} + g = \tau + \varepsilon$. therefore

$$H\ddot{q} = \tau + \varepsilon - (C_0 + \Delta C)\dot{q} - (g_0 + \Delta g) \quad (7.14)$$

So, we have

$$\dot{V}(t) = s^T (\tau + \varepsilon - \Delta H\ddot{q} - \Delta C\dot{q} - \Delta g - H_0\ddot{q}_r - C_0\dot{q} - g_0) + \frac{1}{2}s^T \dot{H}_0 s + \dot{\hat{a}}^T \Gamma^{-1} \tilde{a} \quad (7.15)$$

Now, using $s = \dot{q} - \dot{q}_r$ and $\frac{1}{2}s^T (\dot{H}_0 - 2C_0)s = 0$ (Slotine and Sastry, 1987, 1991) we

have

$$s^T (-C_0\dot{q}) + \frac{1}{2}s^T \dot{H}_0 s = -s^T C_0 s + \frac{1}{2}s^T \dot{H}_0 s - s^T C_0 \dot{q}_r = -s^T C_0 \dot{q}_r,$$

Substituting $\tau = \tau_1 + \tau_2$ we get

$$\dot{V}(t) = s^T (\tau_1 - H_0\ddot{q}_r - C_0\dot{q}_r - g_0) + s^T (\tau_2 + \rho) + \dot{\hat{a}}^T \Gamma^{-1} \tilde{a} \quad (7.16)$$

Since $H_0\ddot{q}_r + C_0\dot{q}_r + g_0 = Ya_0$ and (7.9)-(7.10), therefore

$$\dot{V}(t) = s^T (Y\hat{a} - Ks - Ya_0) + \dot{\hat{a}}^T \Gamma^{-1} \tilde{a} + s^T (\tau_2 + \rho). \quad (7.17)$$

But $s^T \text{sgn}(s) = \|s\|$, Choosing $\dot{\hat{a}}^T = (-\Gamma Y^T s)^T = -s^T Y \Gamma$, we have

$$\hat{a}^T \Gamma^{-1} \tilde{a} = -s^T Y \Gamma \Gamma^{-1} (\hat{a} - a_0). \quad (7.18)$$

Therefore

$$\begin{aligned} \dot{V}(t) &= -s^T K s - s^T \rho + s^T \tau_2 \\ &= -s^T K s - s^T \rho - s^T Q \operatorname{sgn}(s) \\ &\leq 0 \end{aligned} \quad (7.19)$$

Tracking convergence is then proved, i.e. $s \rightarrow 0$.

QED.

Remark 7.1: From (7-9) one can see that the control torque contains two parts: τ_1 is identical to the control law of Slotine and Li (1987; 1991) which is used to estimate the constant part of the system parameters; the other part τ_2 is the switching control characterised by upper bound matrix Q for system uncertainties and external disturbances.

Remark 7.2: In order to achieve the good quality of the above adaptive sliding mode control, we apply the same fuzzy tuning mechanism as described in Section 5.3 to adjust switching control parameter Q according to Lyapunov function and its time derivative.

The fuzzy inference system consists of two linguistic input variables V and \dot{V} and one linguistic output variable Q . The two fuzzy input variables are defined as follows

$$V=\{B, M, S\} \quad (7.20)$$

$$\dot{V}=\{P, Z, N\} \quad (7.21)$$

where B = big, M = medium, S = small, P = positive, Z = zero, N = negative. Fuzzy output variable Q is defined as follows

$$Q=\{B, MB, M, MS, S\} \quad (7.22)$$

where MB = medium big, MS = medium small. The input-output relation of the fuzzy controller with fuzzy variables (7.20), (7.21) and (7.22) is written by

$$V, \dot{V} \rightarrow Q \quad (7.23)$$

The fuzzy tuning algorithm for Q is described in following fuzzy rules,

1. If (V is B) and (\dot{V} is P) then (Q is B)
2. If (V is B) and (\dot{V} is Z) then (Q is MB)
3. If (V is B) and (\dot{V} is N) then (Q is M)
4. If (V is M) and (\dot{V} is P) then (Q is MB)
5. If (V is M) and (\dot{V} is Z) then (Q is M)
6. If (V is M) and (\dot{V} is N) then (Q is MS)
7. If (V is S) and (\dot{V} is P) then (Q is M)
8. If (V is S) and (\dot{V} is Z) then (Q is MS)
9. If (V is S) and (\dot{V} is N) then (Q is S)

The membership functions for fuzzy variables V , \dot{V} , and Q with normalised universe of discourses are chosen as in Fig. 7.1 (a)-(c). The fuzzy inference surface is shown in Fig. 7.1 (d).

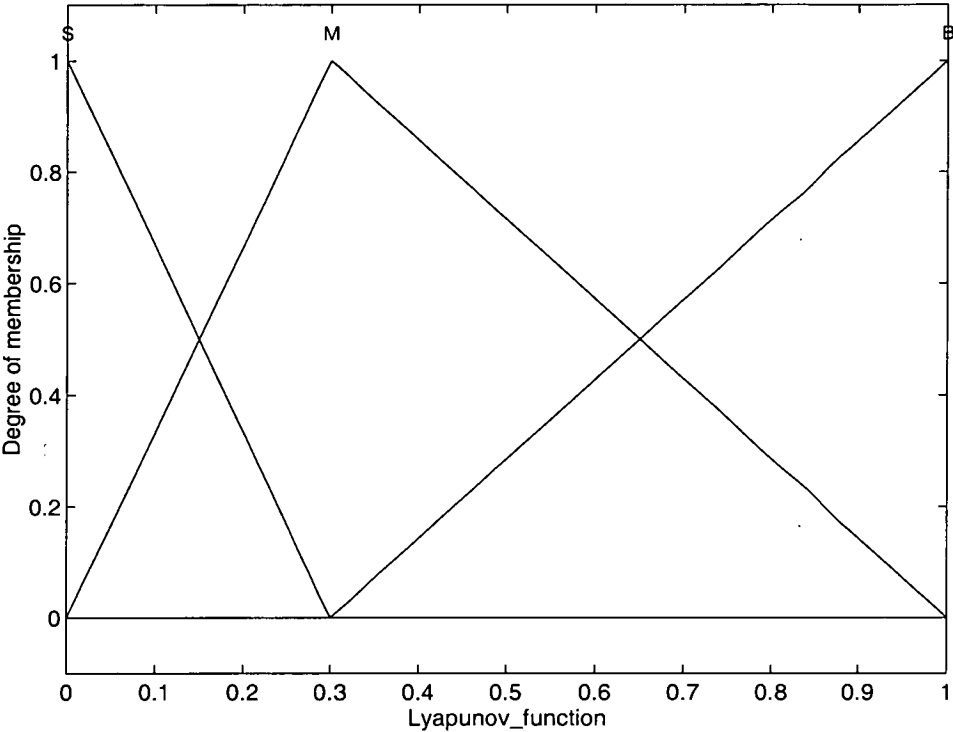


Fig. 7.1 (a) Fuzzy subsets of V

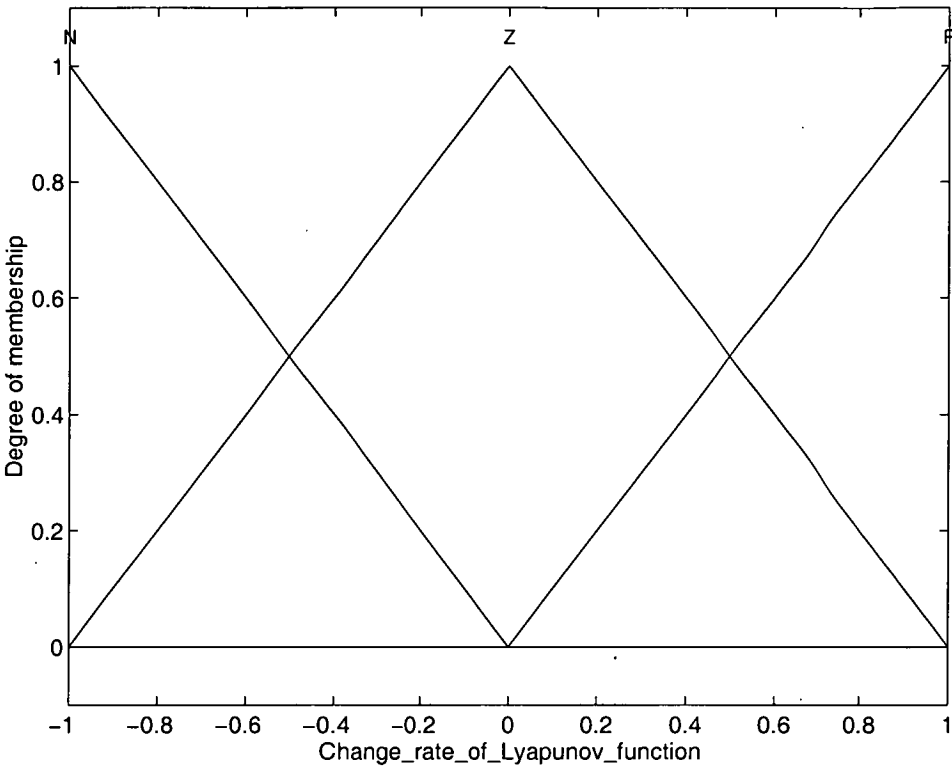


Fig. 7.1 (b) Fuzzy subsets of \dot{V}

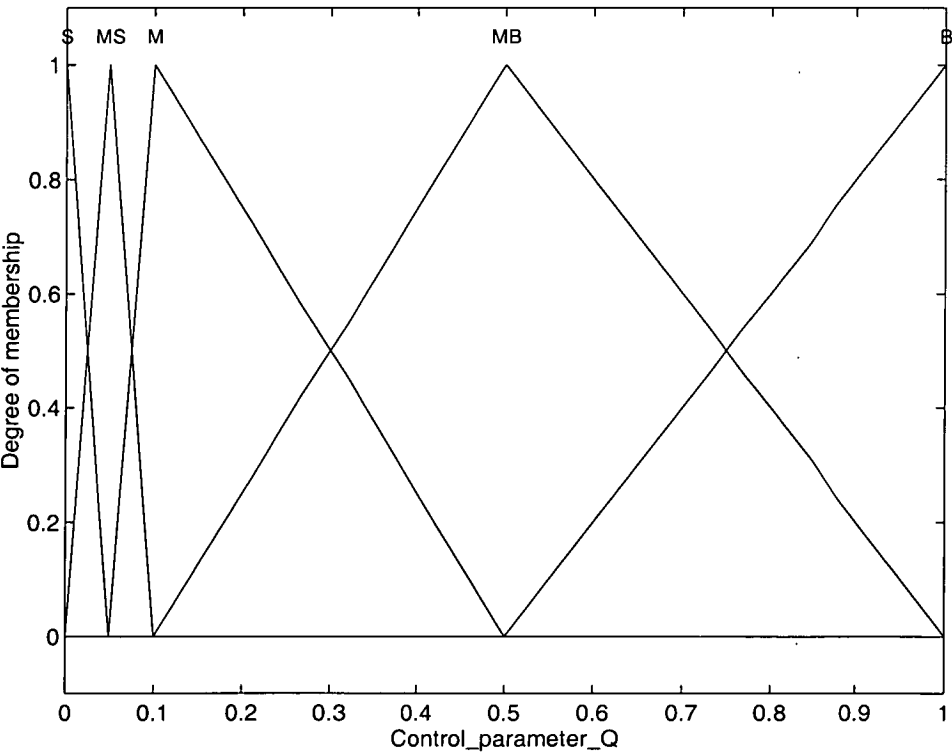
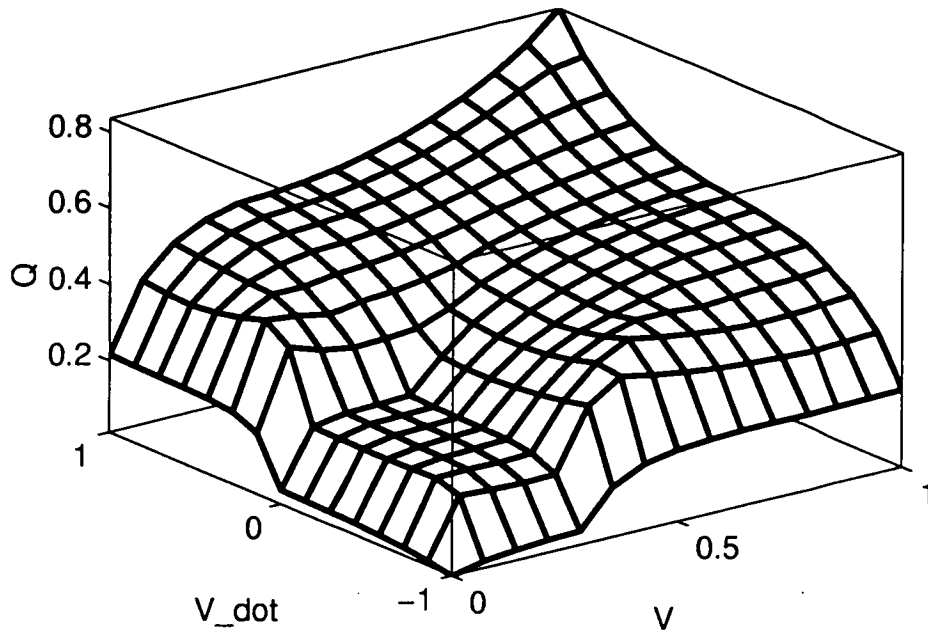


Fig. 7.1 (c) Fuzzy subsets of Q

Fig. 7.1 (d) Surface of $Q(V, \dot{V})$

7.3 An illustrative example

A real-time implementation of the control strategy was developed for a five-bar robotic manipulator, as shown in Fig 7.2.

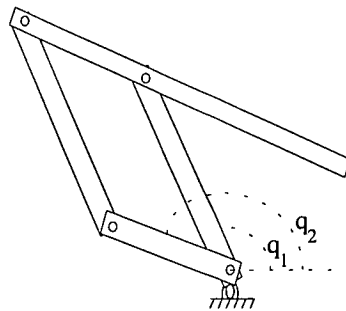


Fig.7.2 A five bar robotic manipulator

The dynamics of the resultant two degree of freedom system, operating in the horizontal plane, can be approximately described by

$$\begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} 0 & h\dot{q}_2 \\ -h\dot{q}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \quad (7.24)$$

where $H_{11} = a_1$, $H_{12} = H_{21} = -a_3 \cos(q_2 - q_1)$, $H_{22} = a_2$, $h = a_3 \sin(q_2 - q_1)$.

The hardware details are shown in Fig. 7.3. Two interface cards are installed in the computer with a Pentium Pro/200MHz CPU. The PC-30D by Eagle Tech consists of two 12-bit D/A converters with full scale output range from 0 to +10V, the output signals are fed to robot motors through a Servo amplifier by Baldor (TSD-050-05-02-1, operated in torque mode). The PCL-833 by Advantech is a 3-axis quadrature encoder and counter add-on card, and receives quadrature signals from DC-Tacho/Encoders (with 500 counts/rev) mounted on the motor shafts for joint angle measurements. DC motors by Maxon (RE035-071-39, gear ratio 86:1) provide the joint actuation.

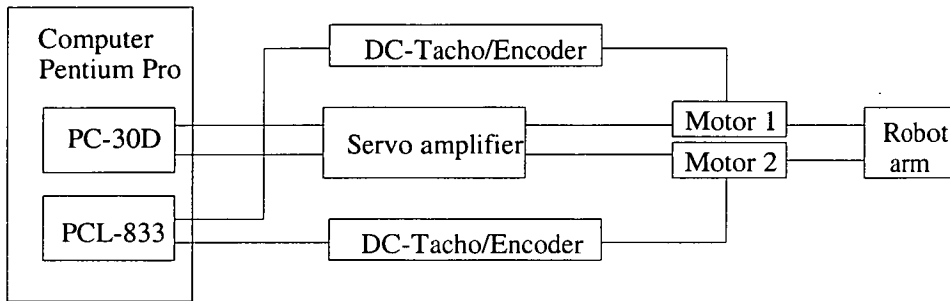


Fig. 7.3 Schematic diagram of the five-bar robot

The following second order dynamic model was chosen to generate the desired trajectory:

$$\dot{x}_m = A_m x_m + B_m r$$

where $x_m = [q_{d1} \ q_{d2} \ \dot{q}_{d1} \ \dot{q}_{d2}]^T$, with initial values $x_m(0) = [2.0 \ 1.5 \ 0 \ 0]^T$

$$A_m = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -6.25 & 0 & -5 & 0 \\ 0 & -6.25 & 0 & -5 \end{bmatrix}, \quad B_m = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and $r(t)$ is the set point.

The initial state of the robot was chosen as

$$[q_1 \ q_2 \ \dot{q}_1 \ \dot{q}_2]^T = [\frac{\pi}{2} \ 0 \ 0 \ 0]^T$$

The initial estimates of the system parameters were chosen as

$$[a_1 \ a_2 \ a_3]^T = [0.02 \ 0.04 \ 0.0015]^T$$

The following parameters were also chosen

$$K = 2I, \quad \Lambda = 10I,$$

$$\Gamma = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}$$

In this experiment, the sampling interval is 1ms.

Fig. 7.4- 7.10 show the experimental results with $\tau_2 = -Q \operatorname{sgn}(s)$ and $Q=0.5$ without fuzzy tuning. It is observed that there exists chattering in the control signal.

Fig. 7.11- 7.17 show the experimental results with $\tau_2 = -Q \operatorname{sgn}(s)$ and $Q_{\max}=0.5$ with fuzzy tuning. It is observed that the control chattering is diminished.

7.4 Concluding remarks

In this chapter, Lyapunov stability theory and fuzzy logic technique are combined together to design fuzzy adaptive sliding mode control systems. It is shown that an adaptive sliding mode control is first designed to learn the system parameters with bounded system uncertainties and external disturbances. A set of fuzzy rules are then used to adjust the controller's uncertainty bound based on the Lyapunov function and its time derivative. The robust adaptive sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering and the amplitude of the control signals, compared with the adaptive sliding mode controller without fuzzy tuning. Experimental example for a five-bar robot arm is given in support of the proposed control scheme.

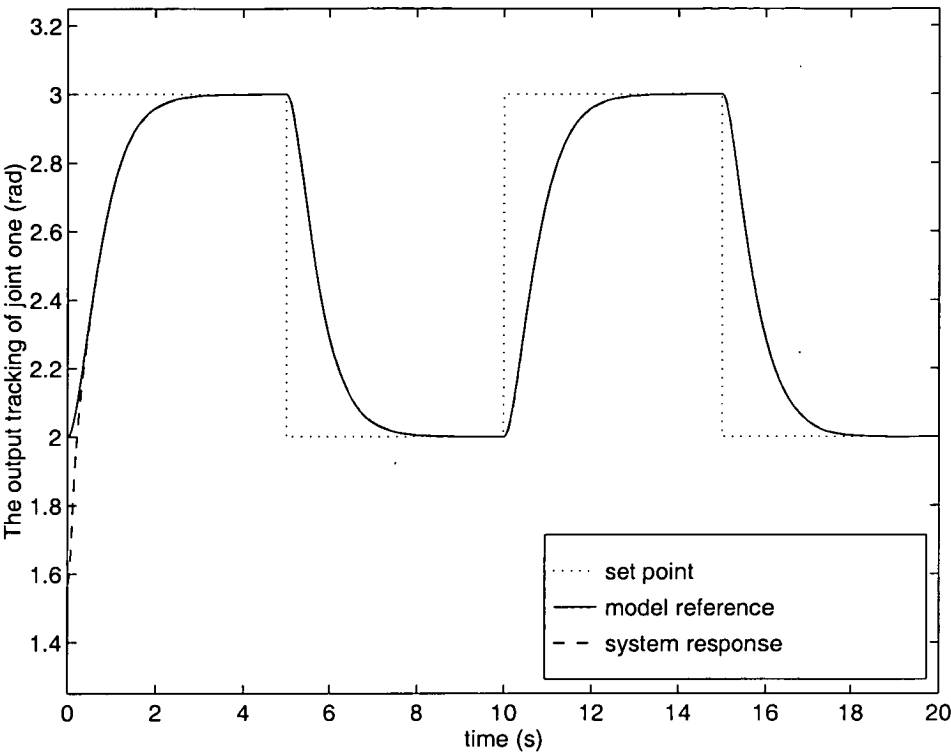


Fig. 7.4 Output tracking of joint 1(without fuzzy tuning)

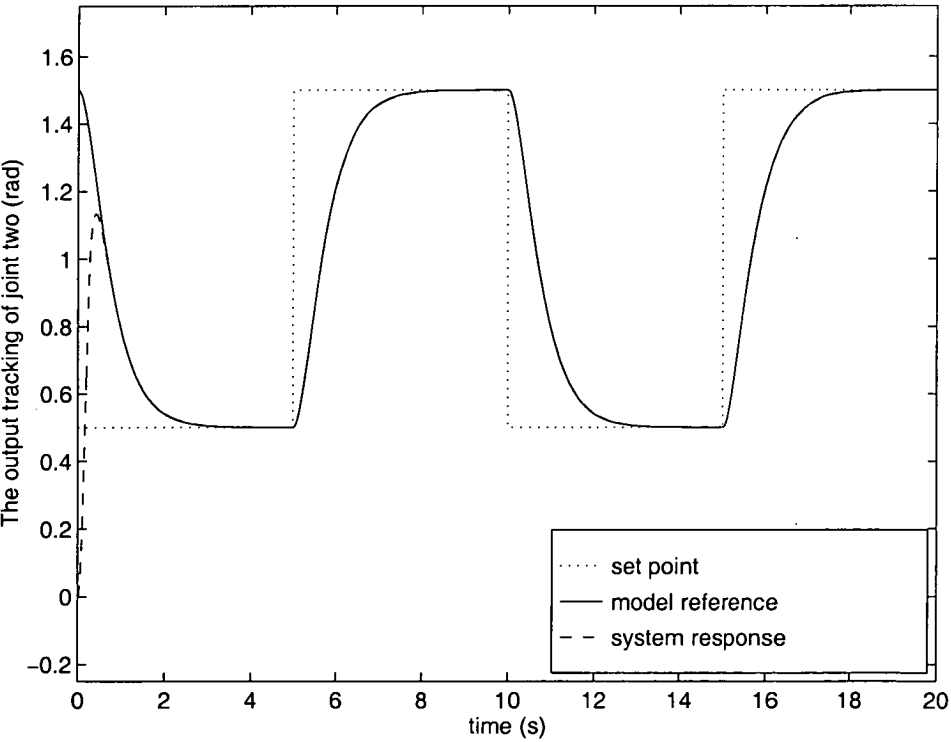


Fig. 7.5 Output tracking of joint 2(without fuzzy tuning)

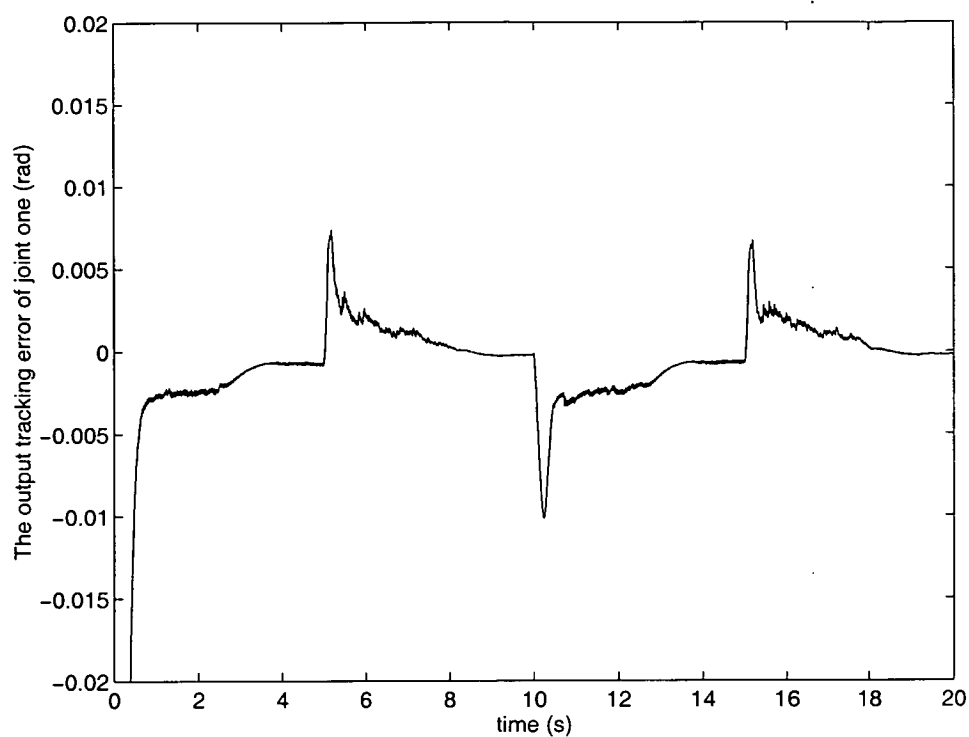


Fig. 7.6 Tracking error of joint 1 (without fuzzy tuning)

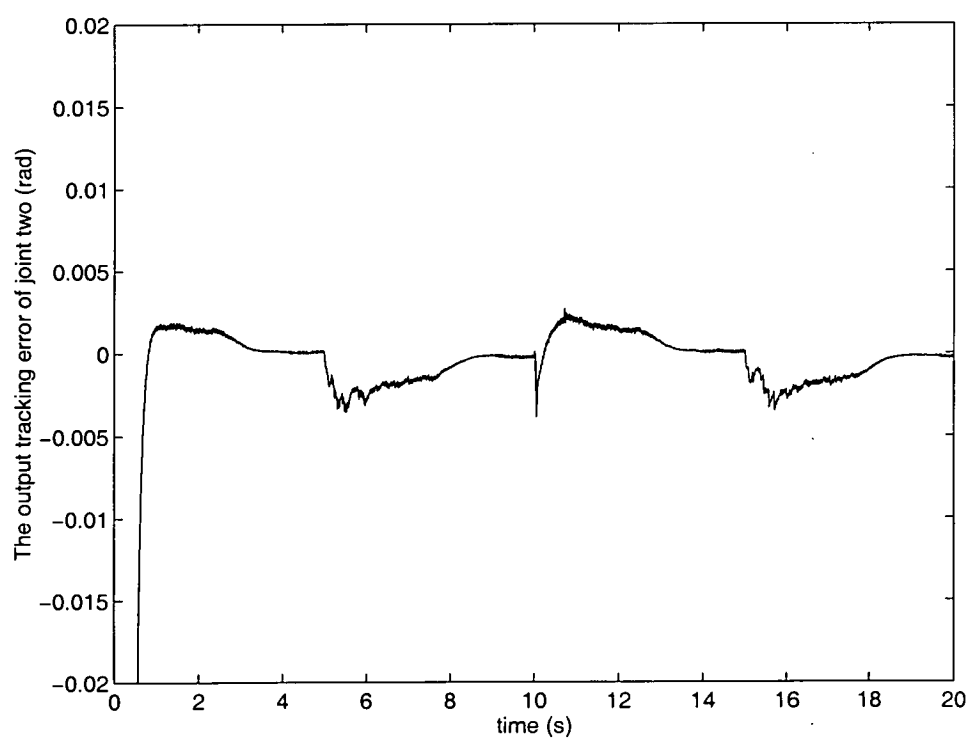


Fig. 7.7 Tracking error of joint 2 (without fuzzy tuning)

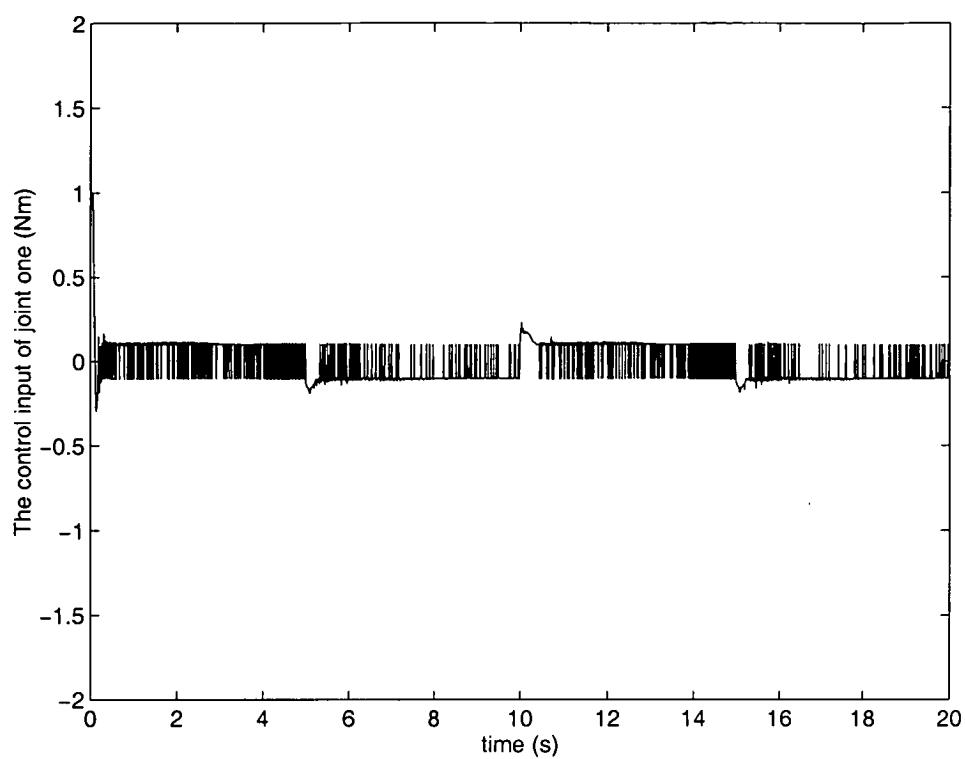


Fig. 7.8 Control input of joint 1 (without fuzzy tuning)

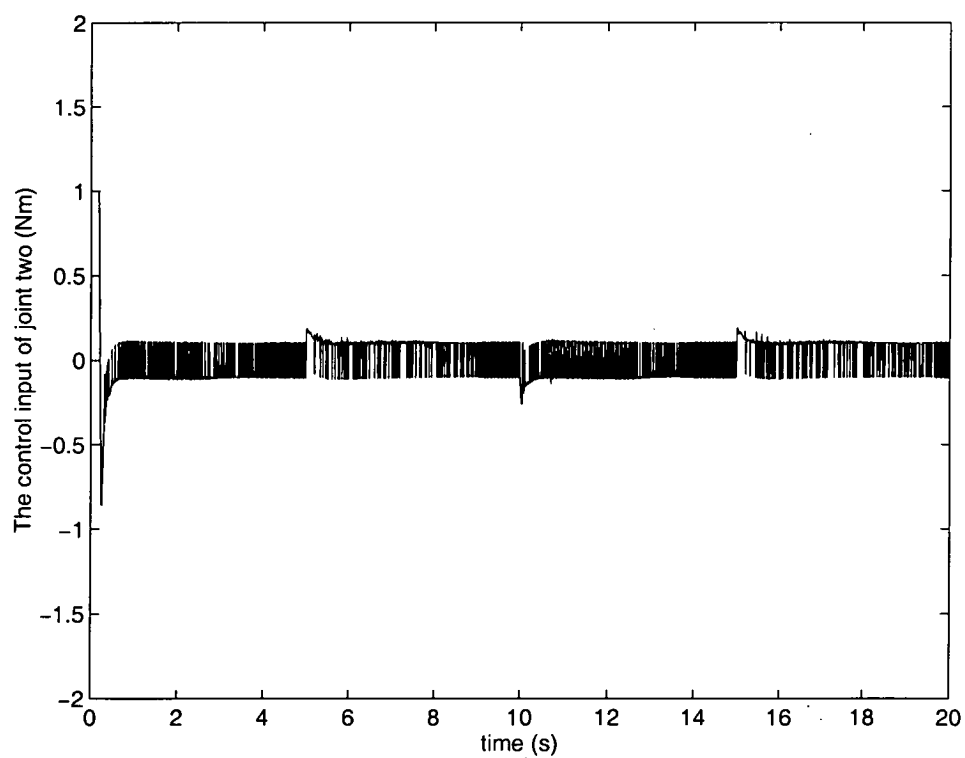


Fig. 7.9 Control input of joint 2 (without fuzzy tuning)

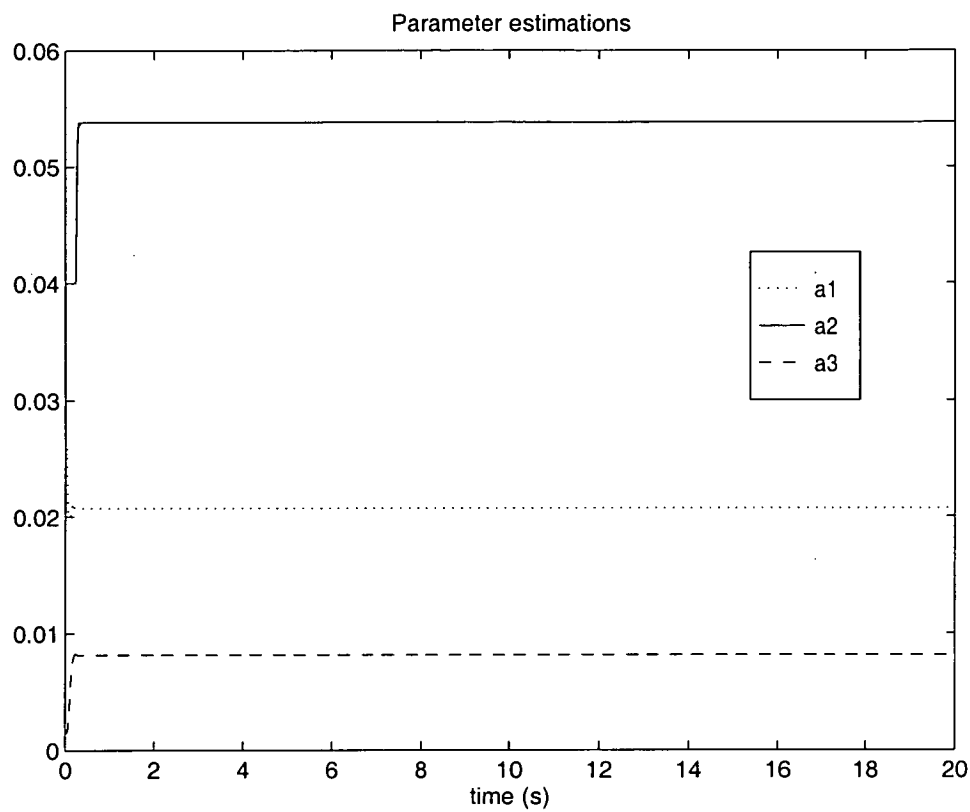


Fig. 7.10 Parameter estimations (without fuzzy tuning)

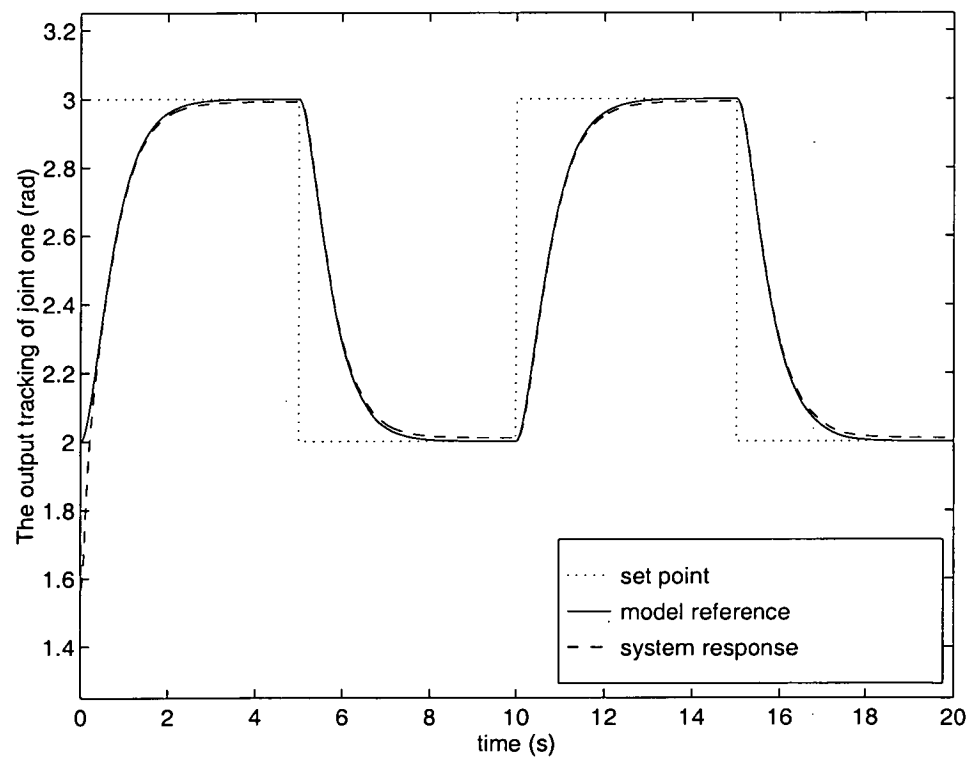


Fig. 7.11 Output tracking of joint 1 (with fuzzy tuning)

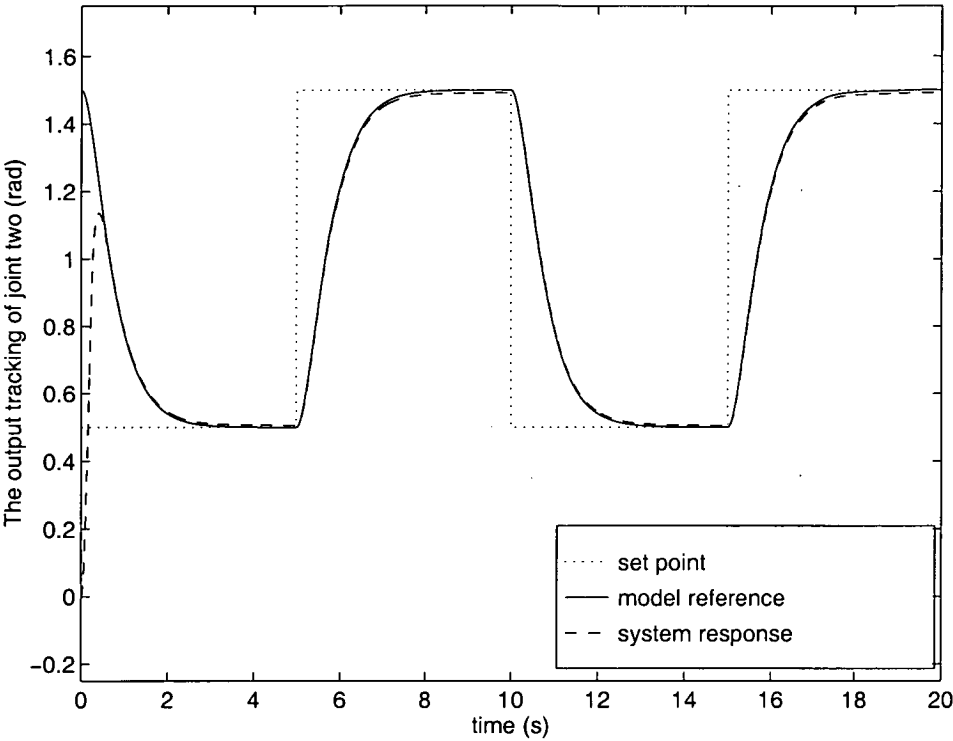


Fig. 7.12 Output tracking of joint 2 (with fuzzy tuning)

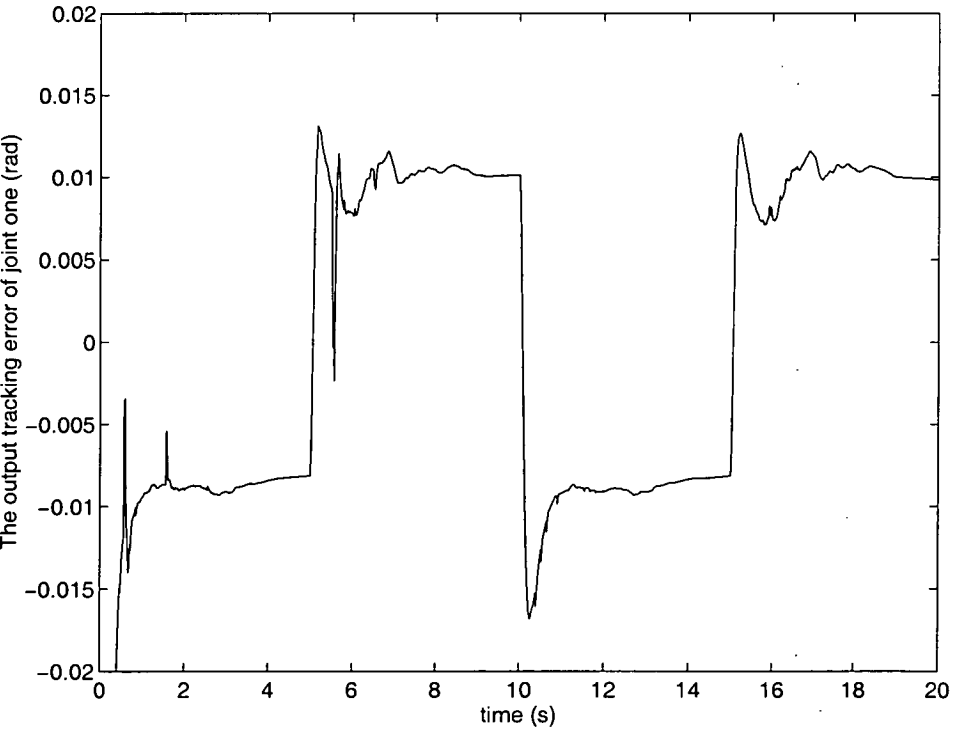


Fig. 7.13 Tracking error of joint 1 (with fuzzy tuning)

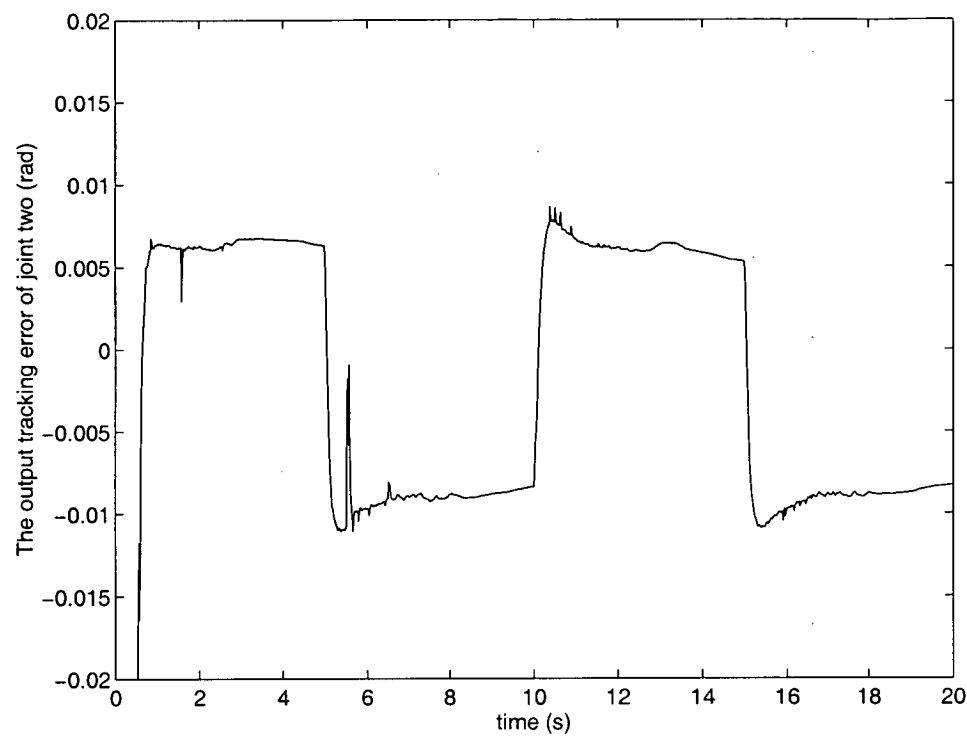


Fig. 7.14 Tracking error of joint 2 (with fuzzy tuning)

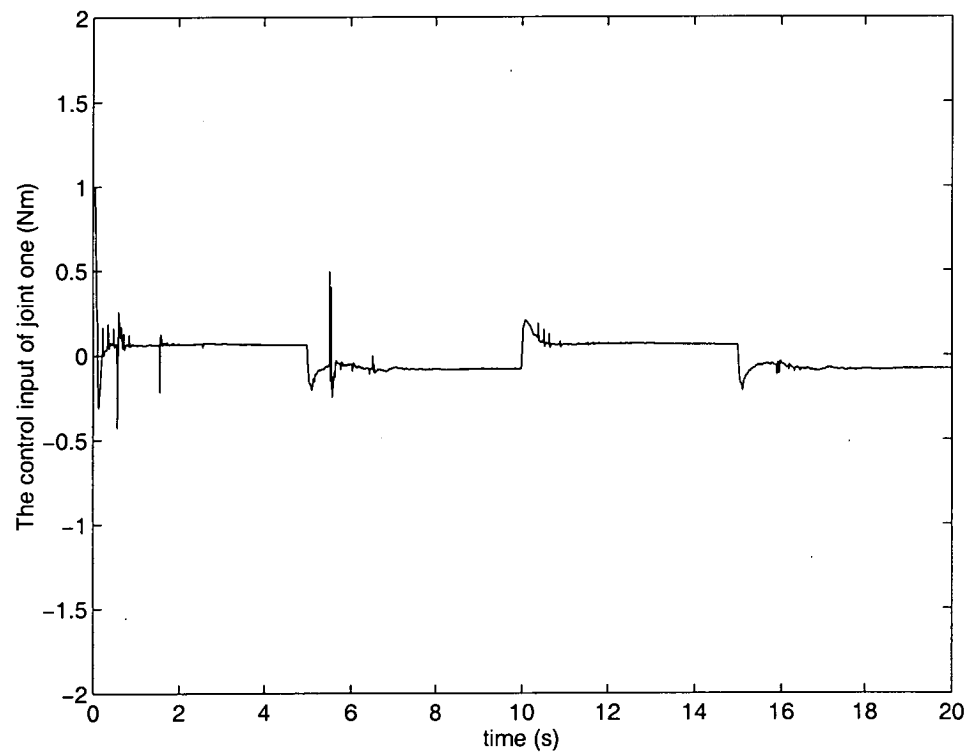


Fig. 7. 15 Control input of joint 1 (with fuzzy tuning)

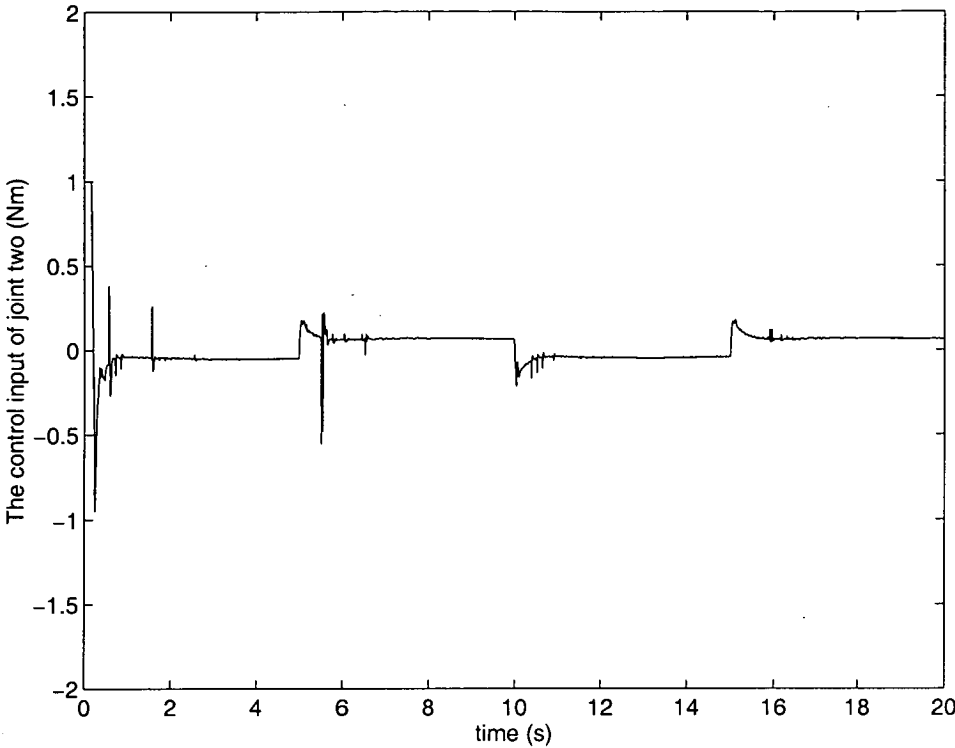


Fig. 7.16 Control input of joint 2 (with fuzzy tuning)

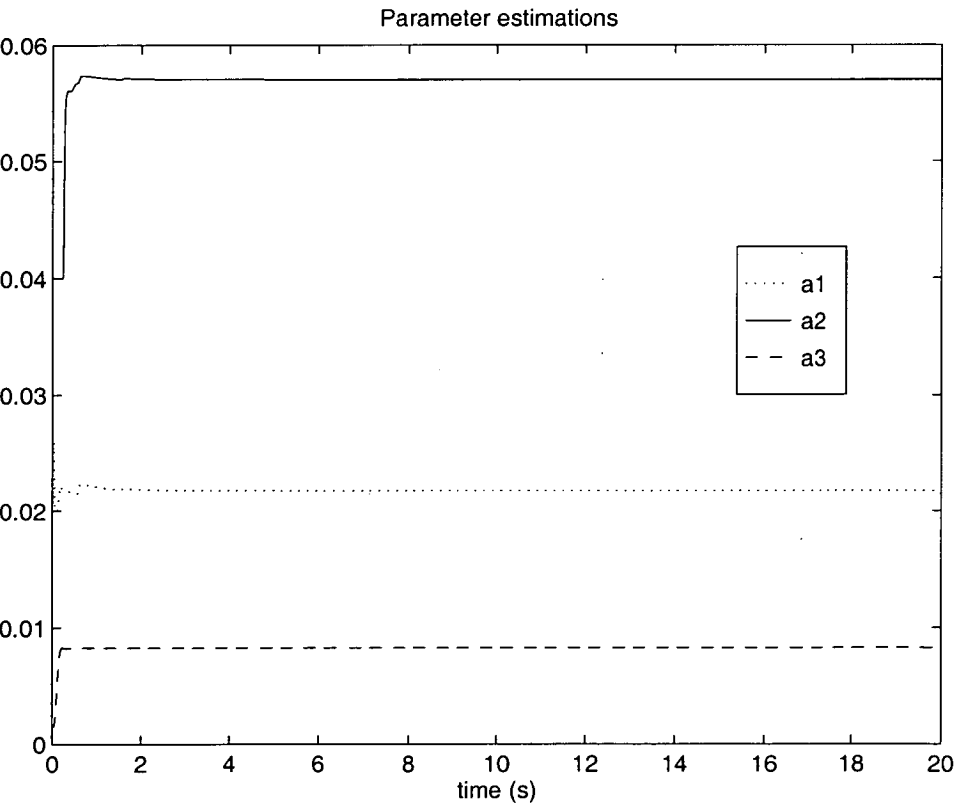


Fig. 7.17 Parameter estimations (with fuzzy tuning)

Chapter 8

Robust Adaptive Sliding Mode Control of Robots

8.1 Introduction

Adaptive control of mechanical manipulators has been studied extensively by many researchers using various methodologies (Craig et al., 1986; Slotine and Sastry, 1984; Balestrino et al., 1979; Young, 1978), such as hyperstability theory, Lyapunov stability method, self-tuning regulator control and sliding mode control (SMC). Among the various methodologies, the SMC demonstrates its good robustness and low computational cost. SMC is a discontinuous feedback control that switches the system control structure during the evolution of the system state so that the system states remain in a prescribed subspace. A SMC controller for robot manipulators suggests that each manipulator link matches a first order sliding motion such that when the system state reaches the sliding mode, it chatters along the sliding mode and exhibits insensitivity to the system parameter variations and external disturbances (Slotine & Li, 1987).

One of the well known results in SMC design for trajectory tracking of robotic manipulators was obtained by Slotine and Li (1987, 1991). Their adaptive SMC is based on the linear parametrisation approach for the robotic systems which do not involve any uncertainties and disturbances. The validity of the control algorithm is based on the assumption that the system parameters must be constant and the desired trajectory should be sufficiently “rich” to guarantee parameter convergence. To overcome the robustness problem of Slotine and Li's controller, Su and Leung (1993) have proposed a modification that employs the bound estimation algorithms in the controller to give rise to a robust adaptive SMC controller without *a priori* knowledge of upper bounds of the system parameters. This controller is able to handle large and time varying variations in the parameters. However, the problem is that their control torques designed incur very large chattering which is not desirable in practice, especially in mechanical systems.

In this chapter, a new adaptive SMC law is proposed for trajectory tracking of rigid robotic manipulators with uncertainties to achieve robustness as well as desired qualities such as speedy tracking, minimal chattering and small control torques. We first show that, under a mild assumption, and if the control input does not contain the acceleration signal, then the system uncertainty is bounded by a positive function of the position and velocity measurements. This result forms the foundation of the derivation of the proposed adaptive SMC law which does not require measurement of acceleration signals. We then propose an adaptive SMC strategy that integrates the Slotine and Li's controller for estimation of the constant part of the system parameters with a switching controller that takes care of the uncertain part of the system parameters with its switching amplitude adaptively learned by a Gaussian RBF neural

network. The advantage of the proposed adaptive SMC strategy is that the switching control part only controls the "real" uncertain part of the system parameters, and the minimum control effort is achieved by adaptive learning of the uncertainty bound. When the magnitude of the uncertainties is small, the chattering caused by the switching is small.

This chapter is organised as follows: In Section 8.2, the robust adaptive SMC design is presented. Section 8.3 describes the experimental results for confirmation of the theoretical results. Section 8.4 gives conclusions.

8.2 The Robust Adaptive SMC Design

Consider the dynamics of the n -joint robotic manipulators represented by the following second order nonlinear vector differential equation

$$H(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau(t) + \varepsilon(t) \quad (8.1)$$

where $q(t)$ is the $n \times 1$ vector of joint angular positions, $H(q)$ is the $n \times n$ symmetric positive definite inertia matrix, $C(q, \dot{q})\dot{q}$ is the $n \times 1$ vector containing Coriolis, centrifugal forces, $g(q)$ represents the derivation of the manipulators potential energy, $\tau(t)$ is the $n \times 1$ vector of applied joint torques which are actually control inputs, and $\varepsilon(t)$ is the $n \times 1$ bounded input disturbances vector.

We assume that the robotic manipulators have uncertainties, i.e.

$$H(q) = H_0(q) + \Delta H(q) \quad (8.2)$$

$$C(q, \dot{q}) = C_0(q, \dot{q}) + \Delta C(q, \dot{q}) \quad (8.3)$$

$$g(q) = g_0(q) + \Delta g(q) \quad (8.4)$$

where $H_0(q), C_0(q, \dot{q}), g_0(q)$ are the nominal parts and $\Delta H(q), \Delta C(q, \dot{q}), \Delta g(q)$ are the uncertain parts. The uncertain components represent bounded small possible time varying, unmodelled dynamics or load changes. Substituting the expressions (8.2)-(8.4) into the dynamics (8.1) gives rise to

$$H_0(q)\ddot{q} + C_0(q, \dot{q})\dot{q} + g_0(q) = \tau(t) + \rho(t) \quad (8.5)$$

with

$$\rho(t) = \varepsilon(t) - \Delta H\ddot{q} - \Delta C(q, \dot{q})\dot{q} - \Delta g(q) \quad (8.6)$$

representing all the uncertain terms. When $\rho(t)=0$, we call the dynamics as the “nominal dynamics” of the robotic manipulators.

Some mild assumptions are made prior to further discussion. These assumptions are usually satisfied in practice.

Assumption 8.1: The matrix $H(q)$ is bounded and invertible, i.e. for some unknown constant γ^h ,

$$\|H(q)\| < \gamma^h \quad (8.7)$$

and $H^{-1}(q)$ exists for all q .

Assumption 8.2: The vectors $C(q, \dot{q})$ and $g(q)$ satisfy

$$\|C(q, \dot{q})\dot{q}\| < \gamma_1^c + \gamma_2^c\|q\| + \gamma_3^c\|\dot{q}\|^2 \quad (8.8)$$

$$\|g(q)\| < \gamma_1^g + \gamma_2^g\|q\| \quad (8.9)$$

where $\gamma^h, \gamma_1^c, \gamma_2^c, \gamma_3^c, \gamma_1^g, \gamma_2^g$ are positive constants but do not have to be known.

According to the characteristics of industrial robots, These assumptions are not stringent and have been used by many researchers (Abdallah et al., 1991). With these mild assumptions, the following theorem is derived which will be very useful for later derivation of the robust adaptive SMC controller for robotic manipulators.

Theorem 8.1: With Assumption 8.1-2, if the control input vector $\tau(t)$ does not contain the acceleration signal \ddot{q} , then the system uncertainty term $\rho(t)$ is bounded by a positive function of the position and velocity measurements:

$$\|\rho(t)\| < F_d(q, \dot{q}) \quad (8.10)$$

where $F_d(q, \dot{q})$ is a positive function to be expressed in the following.

Proof: Expression (8.5) can be written in the following form

$$\ddot{q} = H_0(q)^{-1}(\tau(t) + \rho(t)) - H_0(q)^{-1}h_0(q, \dot{q}) \quad (8.11)$$

with $h_0 = C_0(q, \dot{q})\dot{q} + g_0(q)$. Using expression (8.11) in expression (8.6), we have

$$\begin{aligned} \rho(t) &= -\Delta H(q)\ddot{q} - \Delta h(q, \dot{q}) + \varepsilon(t) = -\Delta H(q)H_0(q)^{-1}\tau(t) \\ &\quad -\Delta H(q)H_0(q)^{-1}\rho(t) - \Delta H(q)H_0(q)^{-1}h_0(q, \dot{q}) - \Delta h(q, \dot{q}) + \varepsilon(t) \end{aligned} \quad (8.12)$$

which gives

$$\begin{aligned} \rho(t) &= -(I + \Delta H(q)H_0(q)^{-1})^{-1}\Delta H(q)H_0(q)^{-1}\tau(t) \\ &\quad + (I + \Delta H(q)H_0(q)^{-1})^{-1}(-\Delta h(q, \dot{q}) + \varepsilon(t)) \\ &\quad - (I + \Delta H(q)H_0(q)^{-1})^{-1}\Delta H(q)H_0(q)^{-1}h_0(q, \dot{q}) \end{aligned} \quad (8.13)$$

Then

$$\begin{aligned} \|\rho(t)\| &\leq \|(I + \Delta H(q)H_0(q)^{-1})^{-1}\| \|\Delta H(q)H_0(q)^{-1}\| \|\tau(t)\| \\ &\quad + \|(I + \Delta H(q)H_0(q)^{-1})^{-1}\| \|\Delta h(q, \dot{q})\| + \|(I + \Delta H(q)H_0(q)^{-1})^{-1}\| \|\varepsilon(t)\| \\ &\quad + \|(I + \Delta H(q)H_0(q)^{-1})^{-1}\| \|\Delta H(q)H_0(q)^{-1}\| \|h_0(q, \dot{q})\| \end{aligned} \quad (8.14)$$

Considering the assumptions 8.1-2 and the fact that the input disturbance vector is upper bounded, we have the following inequalities:

$$\|(I + \Delta H(q)H_0(q)^{-1})^{-1}\| < \alpha_1 \quad (8.15-a)$$

$$\|\Delta H(q)H_0(q)^{-1}\| < \alpha_2 \quad (8.15-b)$$

$$\|\varepsilon(t)\| < \varepsilon_1 \quad (8.15-c)$$

$$\|h_0(q, \dot{q})\| < \alpha_3 + \alpha_4\|q\| + \alpha_5\|\dot{q}\|^2 \quad (8.15-d)$$

$$\|\Delta h(q, \dot{q})\| < \alpha_7 + \alpha_8\|q\| + \alpha_9\|\dot{q}\|^2 \quad (8.15-e)$$

where $\alpha_1, \dots, \alpha_9$ and ε_1 are unknown positive constants. Using inequalities (8.15-a)-(8.15-e) in expression (8.14), we have

$$\begin{aligned} \|\rho(t)\| &< \alpha_1\alpha_2\|\tau(t)\| \\ &\quad + (\alpha_1\alpha_7 + \alpha_1\varepsilon_1 + \alpha_1\alpha_2\alpha_3) \\ &\quad + (\alpha_1\alpha_8 + \alpha_1\alpha_2\alpha_4)\|q\| \\ &\quad + (\alpha_1\alpha_9 + \alpha_1\alpha_2\alpha_5)\|\dot{q}\|^2 \end{aligned} \quad (8.16)$$

Now because the control input does not contain acceleration signal, i.e.,

$$\tau(t) = F(q, \dot{q}) \quad (8.17)$$

and there exists a positive function $F_p(q, \dot{q})$ such that

$$\|\tau(t)\| = \|F(q, \dot{q})\| < F_p(q, \dot{q}) \quad (8.18)$$

Using expression (8.18) in expression (8.16), we have

$$\begin{aligned} \|\rho(t)\| &< \alpha_1\alpha_2F_p(q, \dot{q}) \\ &\quad + (\alpha_1\alpha_7 + \alpha_1\varepsilon_1 + \alpha_1\alpha_2\alpha_3) \\ &\quad + (\alpha_1\alpha_8 + \alpha_1\alpha_2\alpha_4)\|q\| \\ &\quad + (\alpha_1\alpha_9 + \alpha_1\alpha_2\alpha_5)\|\dot{q}\|^2 \\ &\equiv F_d(q, \dot{q}) \end{aligned} \quad (8.19)$$

QED.

Remark 8.1: It has been seen from above discussion that the upper bound of the system uncertainty $\rho(t)$ is input-related because of the high nonlinearities in robot systems.

Remark 8.2: As a special case of theorem 8.1, if the control system uses the following polynomial-type of controller,

$$\|\tau(t)\| < \lambda_0 + \lambda_1 \|q\| + \lambda_2 \|\dot{q}\|^2 = F_p(q, \dot{q}) \quad (8.20)$$

where λ_0 , λ_1 and λ_2 are positive numbers, then by expression (8.19), we have

$$\|\rho(t)\| < b_1 + b_2 \|q\| + b_3 \|\dot{q}\|^2 \quad (8.21)$$

where

$$b_1 = \alpha_1 \alpha_2 \lambda_0 + \alpha_1 \alpha_7 + \alpha_1 \varepsilon_1 + \alpha_1 \alpha_2 \alpha_3,$$

$$b_2 = \alpha_1 \alpha_2 \lambda_1 + \alpha_1 \alpha_8 + \alpha_1 \alpha_2 \alpha_4,$$

$$b_3 = \alpha_1 \alpha_2 \lambda_2 + \alpha_1 \alpha_9 + \alpha_1 \alpha_2 \alpha_5.$$

Bounded property in expression (8.21) has been used by some researchers (Abdallah et al., 1991, Man & Palaniswami, 1994).

In order to track a desired trajectory using robotic manipulators whose parameters are composed of large, fixed parts and small, bounded uncertain parts, we propose a controller structure that consists of two components: one is switching controller with amplitude adaptively learned by using Gaussian RBF neural networks which controls only the uncertain parts; the other is similar to Slotine and Li's controller that takes care of the certain parts.

Suppose $\rho_0(q, \dot{q})$ is the optimal bounding function for $\rho(t)$, i.e.,

$$\|\rho(t)\| \leq \rho_0(q, \dot{q}) \quad (8.22)$$

In this paper, we choose the Gaussian RBF neural network to adaptively learn the bounding function $\rho_0(q, \dot{q})$. It has been shown (Poggio & Girosi, 1990) that a linear superposition of Gaussian RBFs results in an optimal mean square approximation to an unknown function which is infinitely differentiable and whose values are specified at a finite set of points. Further, it has been proven (Poggio & Girosi, 1990) that any continuous functions, not necessarily infinitely smooth, can be uniformly approximated by a linear combination of Gaussian RBFs.

The Gaussian RBF neural network is a particular network architecture (Poggio & Girosi, 1990) that consists of three layers: the input layer, the hidden layer that contains the Gaussian RBF and the output layer. At the input layer, the input space $x \in R^n$ is divided into grids with a Gaussian function at each node defining a receptive field in R^n with c_i as its centre and σ^2 as its variance. The value of variance essentially defines the extent of localisation of the effect of the input. The approximation of the bounding function can be written as,

$$\begin{aligned} \hat{\rho}_0(x) &= \hat{\theta}^T \phi(x) \\ &= \sum_i \hat{\theta}_i \phi_i(x) \end{aligned} \quad (8.23)$$

where $x = [q^T \quad \dot{q}^T]^T$, θ is the weight vector, and

$$\phi_i(x) = \exp\left(-\frac{\|x - c_i\|^2}{\sigma^2}\right) \quad (8.24)$$

Assume that there exists a constant weight vector θ^* so that $\theta^{*T} \phi(x)$ achieves the minimum control effort of the sliding mode control and

$$\theta^{*T} \phi(x) - \rho_0(x) \geq \Delta \quad (8.25)$$

where Δ is a small positive number.

Let the parameters to be estimated be represented

$$a = a_0 + \Delta a \quad (8.26)$$

The sliding function is

$$s = \ddot{q} + \Lambda \tilde{q} = \dot{q} - \dot{q}_r \quad (8.27)$$

where $\dot{q}_r = \dot{q}_d - \Lambda \tilde{q}$, and q_d is the desired trajectory while Λ is a positive constant matrix. We have the following result:

Theorem 8.2: The control torques and the adaptive learning laws given below, ensure convergence of the sliding function s to zero:

$$\tau = \tau_1 + \tau_2 \quad (8.28-a)$$

$$\tau_1 = Y\hat{a} - Ks, \quad (8.28-b)$$

$$\tau_2 = -\hat{\theta}^T \phi(x) \text{sgn}(s) \quad (8.28-c)$$

$$\dot{\hat{a}} = -\Gamma Y^T s, \quad (8.29)$$

$$\dot{\hat{\theta}} = \Omega \phi(x) \|s\| \quad (8.30)$$

where Y is the regressor matrix satisfying

$$H_0 \ddot{q}_r + C_0 \dot{q}_r + g_0 = Y a_0$$

K is a positive definite constant matrix, \hat{a} the estimated parameter, $\tilde{a} = \hat{a} - a_0$, θ the weight vector for bounding function, Ω a positive definite diagonal matrix and Γ a constant positive definite matrix.

Proof: To analyse the convergence and stability, we consider the Lyapunov function

$$V(t) = \frac{1}{2} [s^T H_0 s + \tilde{a}^T \Gamma^{-1} \tilde{a} + \tilde{\theta}^T \Omega^{-1} \tilde{\theta}] \quad (8.31)$$

The derivative of $V(t)$ is

$$\dot{V}(t) = s^T \dot{H}_0 s + \frac{1}{2} s^T \dot{H}_0 s + \dot{\tilde{a}}^T \Gamma^{-1} \tilde{a} - \tilde{\theta}^T \Omega^{-1} \dot{\tilde{\theta}} \quad (8.32)$$

Substituting $\dot{s} = \ddot{q} - \ddot{q}_r$ and $H_0 = H - \Delta H$ yields

$$\dot{V}(t) = s^T (H\ddot{q} - \Delta H\ddot{q} - H_0\ddot{q}_r) + \frac{1}{2} s^T \dot{H}_0 s + \dot{\tilde{a}}^T \Gamma^{-1} \tilde{a} - \tilde{\theta}^T \Omega^{-1} \dot{\tilde{\theta}} \quad (8.33)$$

Since $H\ddot{q} + C\dot{q} + g = \tau$, therefore

$$H\ddot{q} = \tau - (C_0 + \Delta C)\dot{q} - (g_0 + \Delta g) \quad (8.34)$$

So, we have

$$\dot{V}(t) = s^T (\tau - \Delta H\ddot{q} - \Delta C\dot{q} - \Delta g - H_0\ddot{q}_r - C_0\dot{q} - g_0) + \frac{1}{2} s^T \dot{H}_0 s + \dot{\tilde{a}}^T \Gamma^{-1} \tilde{a} - \tilde{\theta}^T \Omega^{-1} \dot{\tilde{\theta}} \quad (8.35)$$

Now, using $s = \dot{q} - \dot{q}_r$ and $\frac{1}{2} s^T (\dot{H}_0 - 2C_0)s = 0$ (Slotine & Li, 1987; 1991) we have

$$s^T (-C_0\dot{q}) + \frac{1}{2} s^T \dot{H}_0 s = -s^T C_0 s + \frac{1}{2} s^T \dot{H}_0 s - s^T C_0 \dot{q}_r = -s^T C_0 \dot{q}_r,$$

Substituting $\tau = \tau_1 + \tau_2$ we get

$$\dot{V}(t) = s^T (\tau_1 - H_0\ddot{q}_r - C_0\dot{q}_r - g_0) + s^T (\tau_2 - \Delta H\ddot{q} - \Delta C\dot{q} - \Delta g) + \dot{\tilde{a}}^T \Gamma^{-1} \tilde{a} - \tilde{\theta}^T \Omega^{-1} \dot{\tilde{\theta}} \quad (8.36)$$

Since $H_0\ddot{q}_r + C_0\dot{q}_r + g_0 = Y a_0$ and (8.28)-(8.30), therefore

$$\begin{aligned} \dot{V}(t) &= s^T (Y\hat{a} - Ks - Y a_0) + \dot{\tilde{a}}^T \Gamma^{-1} \tilde{a} - \tilde{\theta}^T \Omega^{-1} \dot{\tilde{\theta}} \\ &\quad + s^T (\tau_2 - \Delta H\ddot{q} - \Delta C\dot{q} - \Delta g). \end{aligned} \quad (8.37)$$

But $s^T \text{sgn}(s) = \|s\|$, Choosing $\dot{\hat{a}}^T = (-\Gamma Y^T s)^T = -s^T Y \Gamma$, we have

$$\dot{\hat{a}}^T \Gamma^{-1} \tilde{a} = -s^T Y \Gamma \Gamma^{-1} (\hat{a} - a_0). \quad (8.38)$$

Therefore

$$\begin{aligned} \dot{V}(t) &= -s^T K s - s^T (\Delta H \ddot{q} + \Delta C \dot{q} + \Delta g) - s^T \hat{\theta}^T \phi(q, \dot{q}) \operatorname{sgn}(s) - \tilde{\theta}^T \Omega^{-1} \dot{\hat{\theta}} \\ &= -s^T K s - s^T \rho(t) - \hat{\theta}^T \phi(x) \|s\| - (\theta^* - \hat{\theta})^T \phi(x) \|s\| \\ &\leq -s^T K s - [\theta^{*T} \phi(x) - \rho_0(x)] \|s\| \\ &\leq 0 \end{aligned} \quad (8.39)$$

Tracking convergence is then proved, i.e. $s \rightarrow 0$.

QED.

Remark 8.3: From (8.28) one can see that the control torque contains two parts: one is identical to the control law of Slotine and Li (1987;1991) which is used to estimate the constant part of the system parameters; the other part is an adaptively learned switching control by using the Gaussian RBF network. The validity of the combination of these two control structures lies in theorem 1 which ensures that the uncertainty can be represented in terms of q and \dot{q} . The advantage of the proposed controller is that the minimum control effort can be achieved by adaptively learning of the bounding function, thus, when the magnitude of uncertainties is small, the magnitude of chattering is small.

8.3 Experimental Results

To demonstrate the validity of the proposed robust algorithm (8.28)-(8.30), a real-time implementation of the control strategy was developed for a five bar robotic manipulator similar in design to that used by a number of researchers (Mills and Lockhorst,1993; Gourdeau and Schwartz, 1991). As shown in Figure 8.1, the links of

the robot form a parallelogram enabling the position of the end effector to be uniquely specified by the rotations of links one and two.

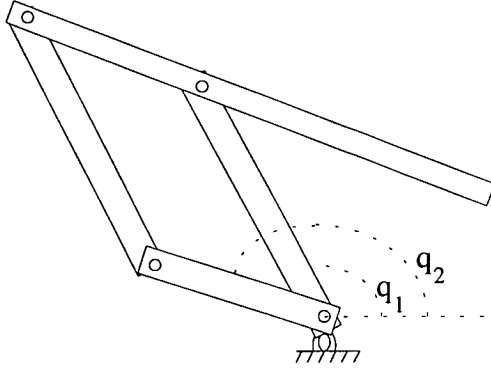


Fig.8.1 A five bar robotic manipulator

The dynamics of the resultant two degree of freedom system, operating in the horizontal plane, can be approximately described by (Mills and Lockhorst, 1993)

$$\begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} 0 & h\dot{q}_2 \\ -h\dot{q}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

where $H_{11} = a_1$, $H_{12} = H_{21} = -a_3 \cos(q_2 - q_1)$, $H_{22} = a_2$, $h = a_3 \sin(q_2 - q_1)$.

The following second order dynamic model was chosen to generate the desired trajectory:

$$\dot{\mathbf{x}}_m = \mathbf{A}_m \mathbf{x}_m + \mathbf{B}_m \mathbf{r}$$

where $\mathbf{x}_m = [q_{d1} \ q_{d2} \ \dot{q}_{d1} \ \dot{q}_{d2}]^T$, with initial values $\mathbf{x}_m(0) = [2.0 \ 1.5 \ 0 \ 0]^T$

$$\mathbf{A}_m = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -6.25 & 0 & -5 & 0 \\ 0 & -6.25 & 0 & -5 \end{bmatrix}, \quad \mathbf{B}_m = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and $\mathbf{r}(t)$ is the set point.

The hardware details of the control system are shown in Figure 8.2. Two interface cards are installed in the computer with a Pentium Pro/200MHz CPU. The PC-30D by Eagle Tech consists of two 12-bit D/A converters with full scale output range from 0 to +10V, the output signals are fed to robot motors through a Servo amplifier by Baldor (TSD-050-05-02-1, operated in torque mode). The PCL-833 by Advantech is a 3-axis quadrature encoder and counter add-on card, and receives quadrature signals from DC-Tacho/Encoders (with 500 counts/rev) mounted on the motor shafts for joint angle measurements. DC motors by Maxon (RE035-071-39, gear ratio 86:1) provide the joint actuation.

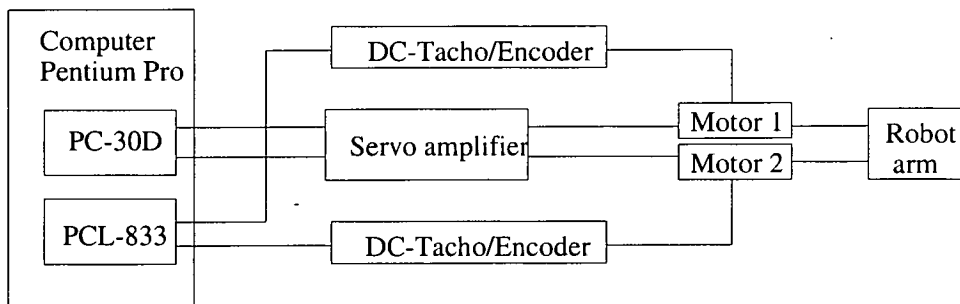


Fig.8.2 Schematic diagram of robot control system

The initial state of the robot was chosen as

$$[q_1 \ q_2 \ \dot{q}_1 \ \dot{q}_2]^T = [\frac{\pi}{2} \ 0 \ 0 \ 0]^T$$

The initial estimates of the system parameters were chosen as

$$[a_1 \ a_2 \ a_3]^T = [0.015 \ 0.04 \ 0.0015]^T$$

The centres of the Gaussian RBFs were chosen as follows: 1.4 and 3.4 for q_1 , 0 and 2.0 for q_2 , -1 and 1 for \dot{q}_1 and \dot{q}_2 . Therefore, totally $2 \times 2 \times 2 \times 2 = 16$ Gaussian RBFs were used in the experiment.

The following parameters were also chosen

$$K = 2I, \ \Lambda = 10I, \ \Omega = 2I, \ \sigma^2 = 2.0$$

$$\Gamma = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}$$

In this experiment, the sampling interval is 1ms. Figure 8.3(a)-(c) show the graphs of the output tracking, tracking errors and control torques. Figure 8.4 shows the parameter estimations. Figure 8.5 illustrates the bounding function estimations.

8.4 Concluding remarks

A new adaptive sliding mode controller has been developed in this chapter for trajectory tracking in robotic manipulators. This controller is able to estimate the constant part of the system parameters as well as adaptively learn the uncertain part of the system parameters by the Gaussian neural network. We have shown that under a mild assumption, the proposed control law does not require measurement of acceleration signals. This new control law exhibits the good aspects of Slotine and Li's and keeps the chattering to a minimum level. An experiment with a five bar robotic system was done and the results have confirmed the effectiveness of the approach.

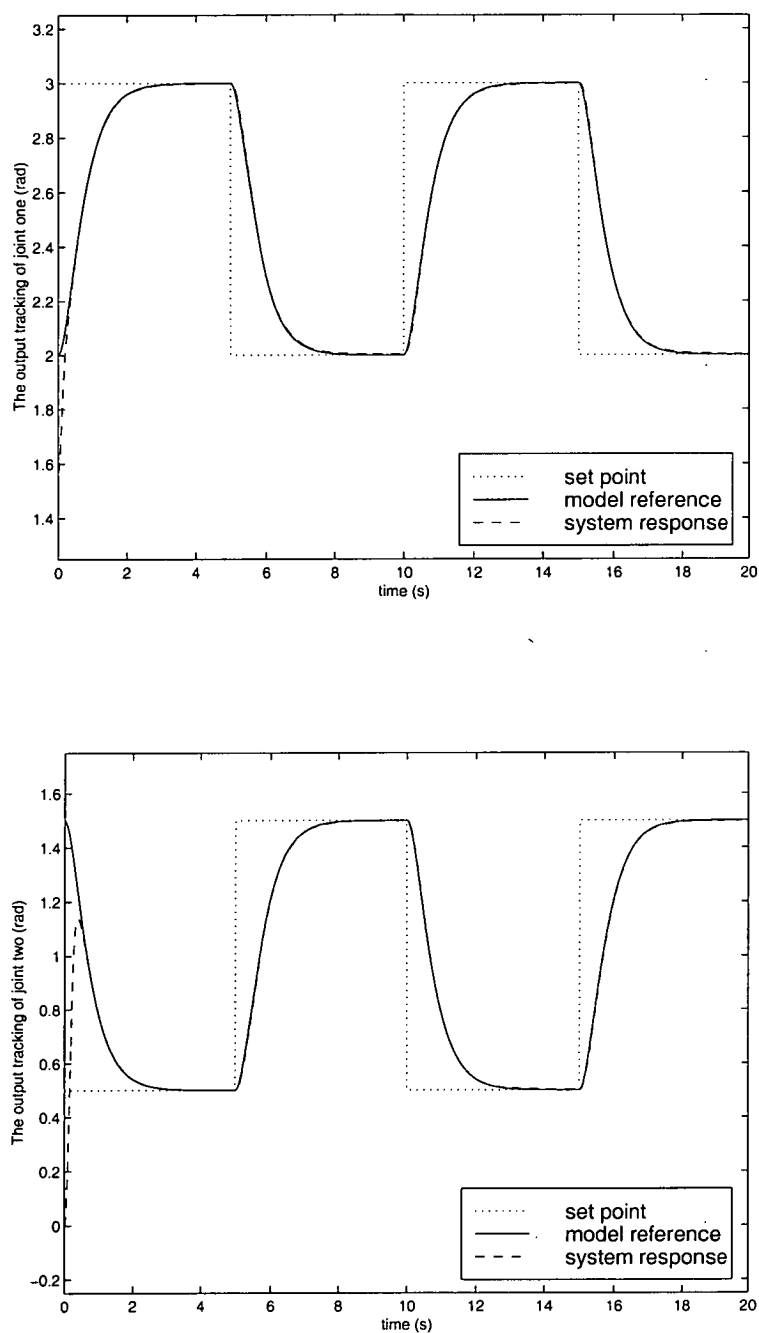


Fig. 8.3(a) The output tracking of joint one and two.

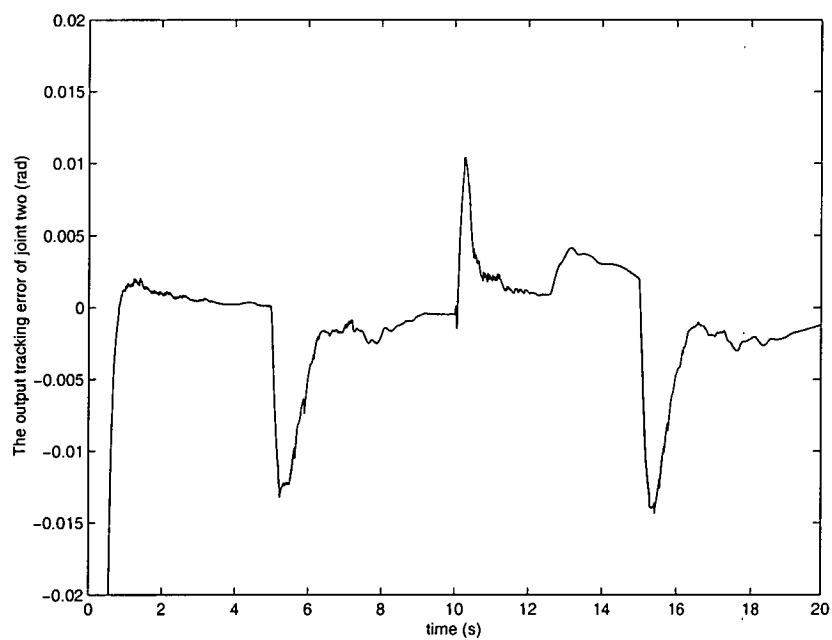
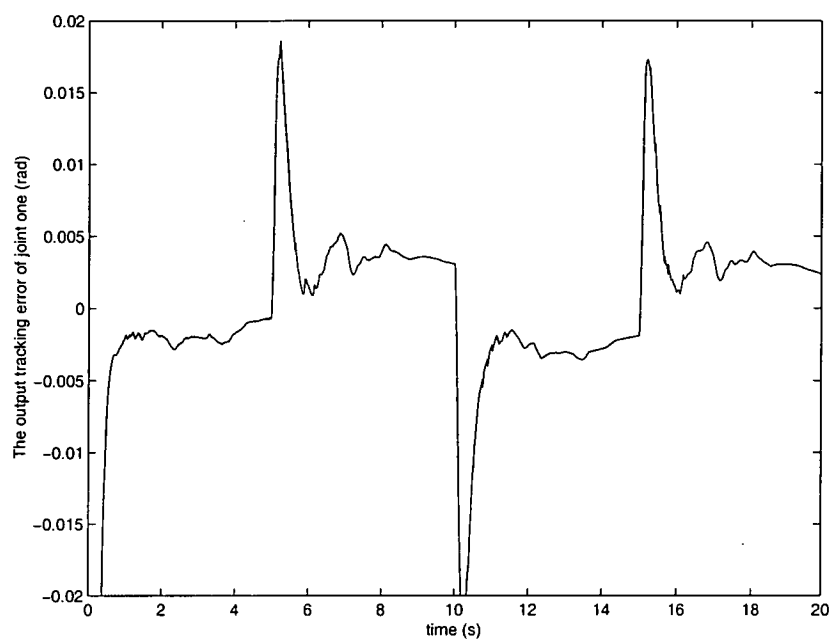


Fig. 8.3(b) The tracking errors of joint one and two.

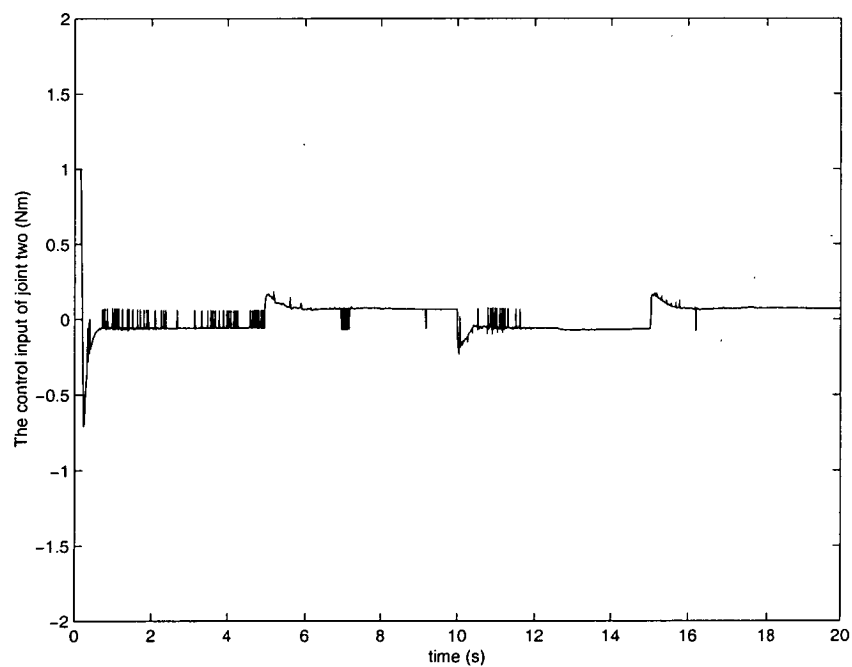
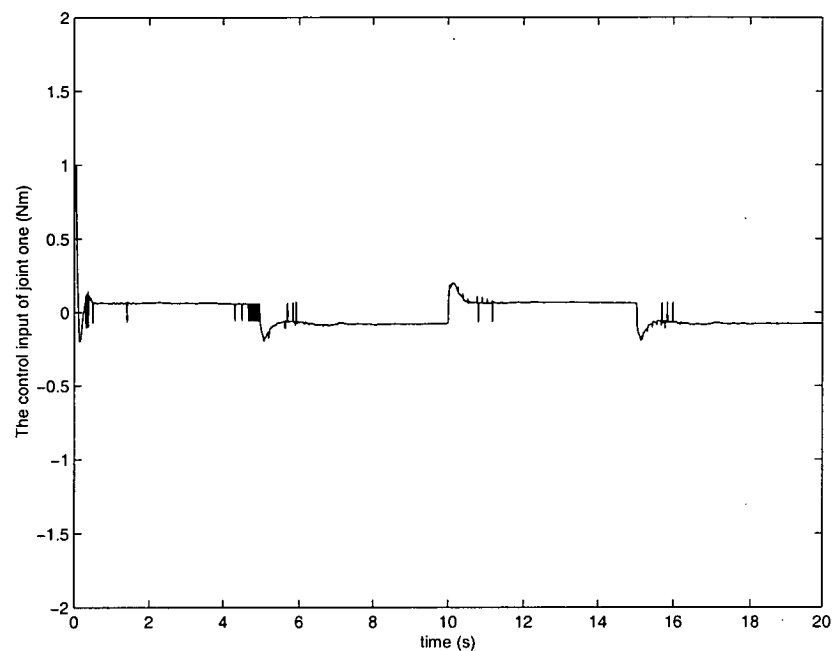


Fig. 8.3(c) The control inputs of joint one and two.

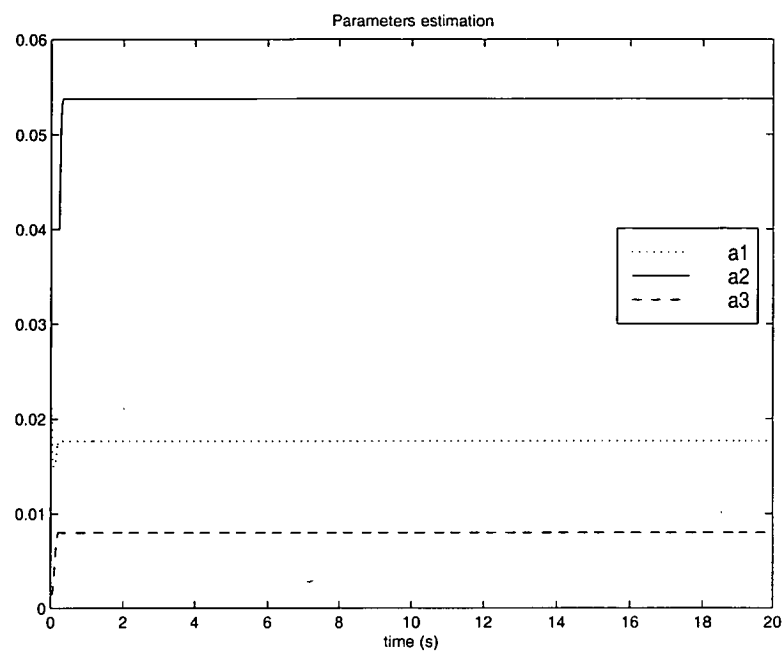


Fig. 8.4 The system parameters estimations

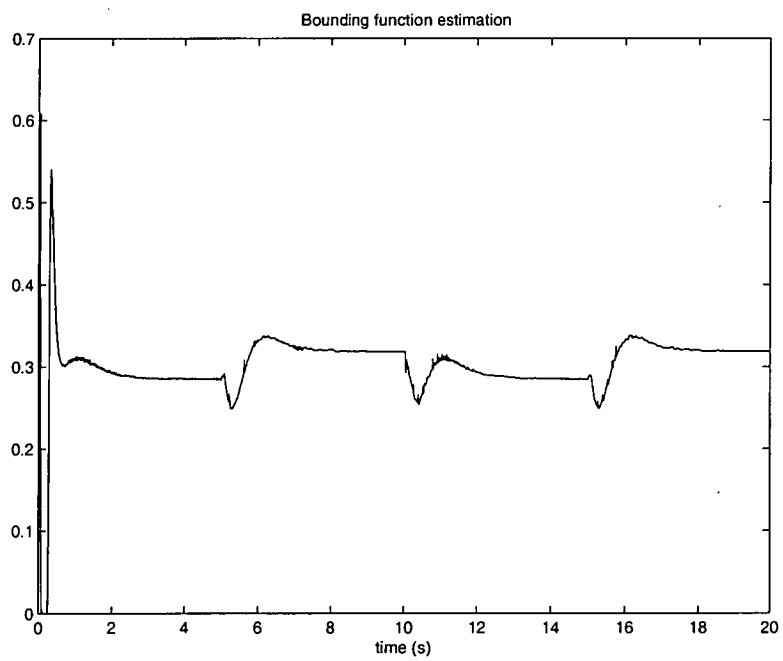


Fig. 8.5 Bounding function estimation

Chapter 9

Conclusions

9.1 Summary

Variable structure control technique is a powerful approach for the control of nonlinear systems. It is advocated to solve complex control problems that are not in the scope of simple linear feedback controllers and adaptive controllers. However, the over estimation of system uncertainties and the inherent control chattering will still be the main issue of the variable structure controller design. Therefore, this thesis has been mainly concerned with the study and improvements of robust control schemes by employing artificial intelligent technologies.

Chapter two of this thesis has provided a brief survey of variable structure control theory. Robust variable structure controller design for robotic manipulators has been presented by using reaching law method. The limitations of these results have also been highlighted.

Chapter three has provided a background for fuzzy logic and fuzzy logic control techniques being applied in the thesis. Fuzzy sets, fuzzy set operations, and fuzzy

linguistic representation such as linguistic variables and linguistic modifiers (hedged) have been briefly outlined. The advantages and disadvantages of fuzzy logic controller have been discussed.

In chapter four, a robust tracking control scheme is proposed for a class of nonlinear systems. A nominal fuzzy system model for a nonlinear system is established by fuzzy synthesis of a set of linearised local subsystems, where the conventional linear feedback control technique is used to design a feedback controller for the fuzzy nominal system. A variable structure compensator is then designed to eliminate the effects of the approximation error and system uncertainties. Strong robustness with respect to large system uncertainties and asymptotic convergence of the output tracking error are obtained.

In chapter five, Lyapunov stability theory and fuzzy logic technique are combined together to design sliding mode control systems. It is shown that a sliding mode is first designed to describe the desired system dynamics for the controlled system. A set of fuzzy rules are then used to adjust the controller's parameters based on the Lyapunov function and its time derivative. The desired system dynamics are then obtained in the sliding mode. The sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering of the control signals, compared with the conventional sliding mode controllers. The fuzzy tuning algorithm has also been applied to the adaptive sliding mode control.

In chapter six, a robust continuous sliding mode control scheme for linear systems with uncertainties has been developed. The controller consists of three components:

equivalent control, continuous reaching mode control and robust control. It retains the positive properties of sliding mode control but without the disadvantage of control chattering. The proposed control scheme has been applied to the tracking control of a one-link robotic manipulator by fuzzy modelling of the nonlinear system.

In chapter seven, Lyapunov stability theory and fuzzy logic technique are combined together to design fuzzy adaptive sliding mode control systems. It is shown that an adaptive sliding mode control is first designed to learn the system parameters with bounded system uncertainties and external disturbances. A set of fuzzy rules are then used to adjust the controller's uncertainty bound based on the Lyapunov function and its time derivative. The robust adaptive sliding mode controllers with fuzzy tuning algorithm show the advantage of reducing the chattering and the amplitude of the control signals, compared with the adaptive sliding mode controller without fuzzy tuning. Experimental example for a five-bar robot arm is given in support of the proposed control scheme.

In chapter eight, a new adaptive sliding mode controller has been developed in this chapter for trajectory tracking in robotic manipulators. This controller is able to estimate the constant part of the system parameters as well as adaptively learn the uncertain part of the system parameters by the Gaussian neural network. It is shown that under a mild assumption, the proposed control law does not require measurement of acceleration signals. This new control law exhibits the good aspects of Slotine and Li's (1987) and keeps the chattering to a minimum level. An experiment of a five bar robotic system was done and the results have confirmed the effectiveness of the approach.

In summary, the thesis has provided several new and improved robust variable structure control schemes by employing fuzzy logic and neural networks technologies aimed at achieving robustness and convergence with optimised control input in the presence of system uncertainties and external disturbances. These results have been applied to robotic manipulators and have been proved both in simulation and experiment.

9.2 Suggestions for further work

Some areas for further research related to the thesis are:

- (1) The sliding mode control technique may be used for the design of adaptive filters.

For example, a Lyapunov function of the tracking error between desired signal and output of the filter can be defined first. Then the adaptation law of the filter parameters can be obtained based on the sliding mode control theory, so that the tracking error can asymptotically converge to zero. The potential advantages of the sliding mode technique based adaptive filters over the existing LMS and RLS algorithms would be the simplicity and robustness.

- (2) The sliding mode control method may be used for the real time training of neural networks parameters to improve the convergence and robustness properties. A Lyapunov function of the error between the desired and actual outputs of the neural networks can be defined first. Then the error is backward-propagated based on Lyapunov stability theory so that the weights of the inner layers of the neural networks can be adjusted adaptively. The distinct advantage over the standard BP or modified BP would be that the system will not get stuck in a local minimum.

- (3) The approach of the terminal sliding mode (Man et al., 1994, Yu et al., 1996) may be used for BP training of weights of neural networks. As in terminal sliding mode design, we can replace the inertial term by following form (LiMin Fu, 1991, pp.83),

$$W_{ji}(t+1) = W_{ji}(t) + \eta \delta_j O_i + \alpha [W_{ji}(t) - W_{ji}(t-1)]^{q/p}$$

The convergence and robustness performance could be expected to be improved.

- (4) Last, but not least, fuzzy set theory may be used in the design of fuzzified linear sliding mode control and terminal sliding mode control. When the system is far from the desired sliding mode, linear sliding mode control scheme can be used to achieve fast convergence, while the system is getting closer to the desired sliding mode, the terminal sliding mode control scheme can be used to guarantee finite time convergence property. Therefore, optimal system performance can be achieved by fuzzified control strategy.

References

- Abbass, A. F., and Ozguner, U. (1985). Decentralised model reference adaptive system using a variable structure control. Proc. 24th Conf. Decision & Control, pp. 1473-1478.
- Abdallah, C., and Jordan, R. (1990). A positive-real design for robotic manipulators. Proc. IEEE Amer. Contr. Conf. pp. 991-992.
- Abdallah, C., Dawson, D., Daorato, P., and Jamshidi, M. (1991). Survey of robust control for rigid robotics. IEEE Control Systems, vol. AC - 11, pp. 24-30.
- Ambrosino, G., Celentano, G., and Garofalo, F. (1984). Variable structure model reference adaptive control systems. Int. J. Control, vol. 39, pp. 1339 - 1349.
- Amestegui, M. R., Ortega, R., and Ibarra, J. M. (1987). Adaptive linearizing-decoupling robotic control: a comparative study of different parametrizations. Proc. 5th Yale Workshop on Applications of adaptive systems theory, New haven, Connecticut.
- Anderson, B. D. O. et al. (1989). Stability of Adaptive Systems: passivity and Averaging Techniques. Englewood Cliffs, NJ: Prentice - Hall.
- Astrom K. J., Hang C. C., Persson P., and Ho W. K. (1992). Toward intelligent PID control. Automatica, Vol. 28, No. 1, pp. 41-53.
- Bahnasawi, A. A., Eid, S. Z., and Mahmoud, M. S. (1991) Adaptive model-following control based on variable structure systems. Int. J. Systems Sci., vol. 22, pp. 333 - 349.

- Baily, E., and Arapostathis, A. (1987). Simple sliding mode control scheme applied to robot manipulators. *Int. J. Contr.*, vol. 45, pp. 1197-1209.
- Balestrino, A., DeMaria, G. and Seiavikko, L. (1979). An adaptive model following control for robot manipulators. *J. Dynamic Systems, Measurement and Control*, Vol.15, pp143-152.
- Barmish, B. R., Petersen, I. R., and Fever, A. (1983). Linear ultimate boundedness control of uncertain dynamical systems. *Automatica*, vol. 19, pp. 523-532.
- Bartolini, G., and Zolezzi, T. (1985). Variable structure nonlinear in the control law. *IEEE Trans. Automat. Contr.* vol. AC-30, pp. 681-684.
- Becker, N., and Grimm, W. M. (1988). On L_2 and L_∞ stability approaches for the robotic manipulators. *IEEE Automat. Contr.*, vol. AC-33, pp. 118-122.
- Bengiamin, N. N., Kauffmann, B. Variable structure position control. *Contr. Syst. Mag.*, pp. 3 - 8.
- Bondi, P., Casalino, G., and Gambardella, L. (1988) On the iterative learning control theory for robotic manipulators. *IEEE J. Robotics and Automation*. vol. 4, pp. 14 - 22.
- Bremer, H. (1985). Parameter-insensitive swivel control of an industrial robot. *Automatisierungstechnik*, vol. 33, pp. 74 - 81, (written in German).
- Canudas De Wit, C., and Slotine, J-J. E. (1991). Sliding observers for robotic manipulators. *Automatica*, vol. 27, pp. 859 - 864.
- Canudas de Wit, C., and Fixot, N. (1991). Robot control via robust estimated state feedback. *IEEE Trans. Automat. Contr.*, vol. AC-36, pp. 1497 - 1501.
- Cao S. G., Ree N. W. and Feng G. (1994). Stability analysis of fuzzy control systems. *Proceedings of the IEEE 2nd Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS-94)*, Brisbane, Australia, pp. 219-223.

- Cao, S. G., Rees, N. W., Feng, G. (1996). Stability analysis and design for a class of continuous-time fuzzy control systems. *Int. J. Control*, vol. 64, pp. 1069-1087, 1996.
- Chang, L. W. (1991). A MIMO sliding control with a first-order plus integral sliding condition. *Automatica*, vol. 27, pp. 853 - 858.
- Chang, S. S. L., and Zadeh, L. A. (1972). Fuzzy mapping and control. *IEEE Trans. Syst., Man, Cybern.*, Vol. SCM-2, No. 1, pp.30-34.
- Corless, M. J. (1989). Tracking control for uncertain systems: application to a manutec r3 robot'. *ASME J. Dynamic System, Measurement, and Control*, vol. 111, pp. 609 - 617.
- Corless, M. J., and Leitmann, G. (1981). Continuous state feedback guaranteeing uniform ultimate boundedness for uncertain dynamic systems. *IEEE Trans. Automat. Contr.*, vol. AC-26, pp. 437 - 443.
- Cox E. (1994). *The fuzzy systems handbook*. Cambridge, MA: AP Professional.
- Crag, J. J., Hsu, P., and Sastry, S. S. (1986). Adaptive control of mechanical manipulators. *IEEE Int. Conf. on Robotics and Automation*, pp. 190-195.
- Crag, J. J., Hsu, P., and Sastry, S. S. (1987). Adaptive control of mechanical manipulators. *Int. J. Robotics Research*, vol. 6, pp. 16-28.
- Craig, J-J. (1986). *Introduction to robotics: machine and control*. Addison-Wesley Company Inc.
- Craig, J. J. (1988). *Adaptive control mechanical manipulators*. Addison-Westey, Reading, Massachusetts.
- Craig, J. J., Hsu, P. and Sastry S. S. (1986). Adaptive control of mechanical manipulators. *Int. J. Control*, Vol.44, No.4, pp1185-1191.
- Cvetkovic, V., and Vukobratovic, M. (1982). One robust dynamic control algorithm for manipulator systems. *Int. J. Robotics Res.*, vol. 1, pp. 15 - 28.

- Davari, A., and Zhang, Z. (1991). Application of the three-segments variable structure systems. American Control Conference, pp. 62 - 63.
- DeCarlo, R. A., Zak, S. H. and Matthews, G. P. (1988). Variable structure control of nonlinear multivariable systems: A tutorial. Proc. of the IEEE, vol. 76, pp. 212 - 232.
- Desa, S., and Roth, B. (1985). Synthesis of control systems for manipulators using multivariable robust servo mechanism theory. Int. J. Robotic Res., vol. 4, pp. 18-34.
- Desoer, C., and Vidyasagar, M. (1975). Feedback Systems: Input - output Properties. New York: Academic.
- Dorling, C. M., and Zinober, A. S. I. (1986). Two approaches to hyperplane design in multivariable variable structure control systems. Int. J. Control, vol. 44, pp. 65 - 82.
- Drazenovic, B. (1969). The invariance conditions in variable structure system. Automatica, vol. 5, pp. 287 - 295.
- Dubois D. and Prade H., (1984). Fuzzy logic and the generalised modus ponens revisited. Cybern. Syst., Vol. 15, No. 1, pp. 3-4.
- Emelyanov, S. V., and Fedotova, A. I. (1962). Design of astatic tracking systems with variable structure. Automat. Remote Contr., pp. 1223 - 1235.
- Emelyanov, S. V., Bermant, M. A., and Kostyleva, N. E. (1966). Design principles for variable structure control systems. In Proc. 3rd IFAC Congr., vol. 1, book 3, pp. 40c.1 - 40c.6.
- Emelyanov S.Y. (1967). *Variable structure control systems*. Moscow:Nauka, (in Russian).
- Fadali, M. S., Zohdy, M., and Adamczyk, B. (1989). Robust pole assignment for computed torque robotic manipulators control. in Proc. IEEE Amer. Contr. Conf., pp.37 - 41.

- Feng G., Cao S. G., Rees N. W., and Chak C. K. (1997). Design of fuzzy control systems with guaranteed stability. *Fuzzy Sets and Systems*, vol.85, pp.1-10.
- Freund, E. (1982). Fast nonlinear control with arbitrary pole-placement for industry robots and manipulators. *Int. J. Rob. Res.*, vol. 1, pp. 65-78.
- Fu, L. C., and Liao, T. L. (1990). Globally stable robust tracking of nonlinear systems using variable structure control and with an application to a robotic manipulator. *IEEE Trans. Automat. Contr.*, vol. 35, pp. 1345 - 1350.
- Fu, LiMin. (1991). *Neural Networks in Computer Intelligence*. McGraw-Hill, Inc.
- Furuta, K. (1990). Sliding mode control of a discrete system. *System & Control Letters*, vol. 14, pp. 145 - 152.
- Gao W. B., and Cheng M. (1989). Quality of variable structure control systems. *Cont. Decision*, Vol.4, no.4, pp. 1-7. (in Chinese)
- Gao W. B., and Hung, J. C. (1993). Variable structure control of nonlinear systems:A new approach. *IEEE Trans. Ind. Electron.*, Vol. 40, no.1, pp.45-55.
- Gavel, D. T., and Hsia, T. C. (1987). Decentralised adaptive control of robotic manipulators. *proc. IEEE Int. Conf. on Robotics and Automations*, pp. 1230 - 1235.
- Gilbert, E. G., and Ha, I. J. (1984). An approach to nonlinear feedback control with applications to robotics. *IEEE Trans. Sys., Man, and Cyber.*, vol. 14, pp. 879 - 884.
- Gourdeau R., and Schwartz H. M. (1991). Adaptive control of robotic manipulators: experimental results. *Proc. IEEE Intl. Conf. Robotics and Auto.*, pp.8-15.
- Grimm, W. M. (1990). Robot non-linearity bounds evaluation techniques for robust control. *Int. J. Adaptive Control and Signal Processing*, vol. 4, pp. 501 - 522.
- Grimm, W. M. (1990). Robustness analysis and synthesis of model-based robot control. *VDI-Fortschrittberichte, Reihe 8, No. 202*, VDI Duesseldorf.

- Grimm, W. M., Becker, N., and Frank, P. M. (1988). Robust stability design framework for robot manipulator control. *IEEE Int. Conf. on Robotics and Automation*, pp. 1042 - 1047.
- Gutman, S. (1979). Uncertain dynamic system - A Lyapunov min-max approach. *IEEE Trans. Automat. Contr.*, vol. AC-24, pp. 434 - 443.
- Gutman, S., and Polmor, Z. (1982). properties of min-max controllers in uncertain dynamical systems. *SIAM. J. Contr. Optimization*. vol. 20, pp. 850-861.
- Ha Q. P. (1996). PI controllers with fuzzy tuning. *IEE Electronics Letters*, Vol. 32, No. 11, pp. 1043-1044.
- Ha Q. P. Negnevitsky M., and Man Z. (1996). Sliding mode control with fuzzy tuning. *Proceedings of the 4th IEEE Australian and New-Zealand Conference on Intelligent Information Systems*, Adelaide, Australia, pp. 216-219.
- Hellendoorn H. and thomas C. (1993). Defuzzification in Fuzzy controllers. *J. Intell. Syst.*, Vol. 1, pp. 109-123.
- Horn, B. K. P., and Raibert, M. H. (1978). Manipulator control using the configuration space method. *The industrial Robot*, vol. 5, pp. 69 - 73.
- Hsu, L., and Costa, R. R. (1989). Variable structure model reference adaptive control using only input and output measurements. *Int. J. Control*, vol. 49, pp. 399 - 416.
- Hua O. W., Tanaka K., and Griffin M. F. (1996). An approach to fuzzy control of nonlinear systems: stability and design issues. *IEEE Transactions on Fuzzy Systems*, vol. 4, pp. 14-23.
- Hung, J. Y., Gao, W. B., and Hung, J. C. (1993). Variable structure control: A survey. *IEEE Trans. Ind. Electron.*, vol.40, no.1, pp.2-22.
- Ioannou, P. A. (1986). Decentralised adaptive control of interconnected systems. *IEEE Trans. Automat. Contr.*, vol. AC-31, pp. 291 - 298.
- Itkis, U. (1976). *Control systems of variable structure*. John Wiley, New York.

- Jabbari, F., and Schmitendorf, W. E. (1990). A noninteractive method for the design of linear robust controllers. *IEEE Trans. Automat. Contr.*, vol. AC-35, pp. 954 - 957.
- Jager R. (1995). Fuzzy logic in control. PhD thesis, Technische Universiteit Delft.
- Jang J.-S. R. and Gulley N. (1995). Fuzzy logic toolbox for use with MATLAB, Natick, MA: The Mathworks, Inc..
- Jang J.-S. R. and Sun C. T. (1995). Neuro-fuzzy modelling and control. *Proceedings of the IEEE*, Vol. 83, No. 3, pp.378-406.
- Khurana, H., Ahson, S. I., and Lamba, S. S. (1986). On the stabilization of large-scale control systems using variable structure system theory. *IEEE Trans. Automat. Contr.* vol. AC - 31, pp. 176 - 178.
- Klein, G., and Moore, B. C., (1977). Eigenvalue-generalised eigenvector assignment with state feedback. *IEEE Trans. Automat. Contr.* vol.AC- 22, pp. 140-141.
- Kosko B. (1992). *Neural networks and fuzzy systems: a dynamical systems approach to machine intelligence*. Englewood Cliffs, NJ: Prentice-Hall.
- Kreutz, K. (1989). On manipulator control by exact linearization. *IEEE Trans. Automat. Contr.* vol. AC-34, pp. 763 - 767.
- Kuo, C. Y., and Wang, S. P. T. (1989). Nonlinear robust industrial robot control. *Trans. ASME J. Dyn. Syst. Means. Control*, vol. 111, pp. 24 - 30.
- Kwong W. A., Passion K. M., Laukonen E. G. and Yurkovich S. (1995). Expert supervision of fuzzy learning systems for fault tolerant aircraft control. *Proceedings of the IEEE*, Vol. 83, No. 3, pp. 466-482.
- Larkin L. I. (1985). A fuzzy logic controller for aircraft flight control. in *Industrial Applications of Fuzzy Control*, Sugeno M. (Ed.), Amsterdam: North Holland, pp.87-104.

- Larsen P. M. (1980). Industrial applications of fuzzy logic control. *Int. J. Man, Mach. Studies*, Vol. 12, No. 1, pp. 3-10.
- Lee C. C. (1990). Fuzzy logic in control systems: fuzzy logic controller-Part I & Part II. *IEEE Trans. Syst., Man, Cybern.*, Vol. SCM-20, No 1, pp. 404-435.
- Leitmann, G. (1981). On the efficacy of nonlinear control in uncertain linear systems. *Trans. ASME J. Dyn. Syst. Means. Control*, vol. 102, pp. 95-102.
- Leung, T. P., Zhou, Q. J., and Su, C. Y. (1991). An adaptive variable structure model following control design for robotic manipulators. *IEEE Trans. Automat. Contr.* vol. AC-36, pp. 347-352.
- Lin, S. C., and Kung, C. C. (1992). A linguistic fuzzy-sliding mode controller. Submitted to 1992 ACC.
- Lu, Y. S., and Chen, J. S. (1994). A self-organising fuzzy sliding mode controller design for a class of nonlinear servo systems. *IEEE Trans. Ind. Electron.*, Vol. 41, No.5, pp.492-502.
- Luh, J. Y. S. (1983). Conventional controller design for industrial robots: a tutorial. *IEEE Trans. Sys., Man, & Cyber.*, vol. 13, pp. 298 - 316.
- Mamdani E. H. (1974). Application of fuzzy algorithms for control of a simple dynamic plant. *Proc. of the IEE*, Vol.121, No. 12, pp. 1585-1588.
- Mamdani E. H. and Assilian S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *Int. J. Man Machine Studies*, Vol 7, No. 1, pp. 1-13.
- Man Z. H., and Palaniswami M. (1993). An improved variable structure model following control for robotic manipulators. *Int. J. of Adaptive Control and Signal Processing*, vol. 7, pp.539 - 562, December, 1993.
- Man Z. H., and Palaniswami M. (1994). Robust tracking control for rigid robotic manipulators. *IEEE Trans. Automatic Control*, vol. 39, pp. 154 - 159.

- Man Z. H., Paplinski A. P. and Wu H. R. (1994). A robust MIMO terminal sliding mode control scheme for rigid robotic manipulators. *IEEE Trans. Automatic Control*, vol. 39, pp. 2464 - 2469.
- Man, Z. H. (1995). A robust adaptive tracking controller using neural networks for a class of nonlinear systems. *J. of Neural Computing & Applications*, vol. 3, pp. 157 - 163.
- Man Z. H., and Palaniswami M. (1995). A robust adaptive tracking control scheme for robotic manipulators with uncertain dynamics. *Int. J. of Computer and Electrical Engineering*, vol. 21, no. 3, pp. 211 - 220.
- Man Z. H. (1995). Parameter estimation of continuous linear time invariant systems using functional approximation. *Int. J. of Computer & Electrical Engineering*, vol. 21, no. 3, pp. 183 - 187.
- Man Z. H., and Palaniswami M. (1995). A decentralised three-segment nonlinear sliding mode control for rigid robotic manipulators. *Int. J. of Adaptive control and Signal Processing*, vol. 9, pp. 443 - 457.
- Man Z. H., and Yu X. H. (1997). Terminal sliding mode control of MIMO linear systems. *IEEE Trans. Circuits and Systems-I*, Vol. 44, No. 11, pp. 1065-1070.
- Man Z. H., and Habibi D. (1997). A robust adaptive sliding mode control for rigid robotic manipulators with arbitrary bounded input disturbances. *J. of Intelligent & Robotics Systems*, vol. 17, pp. 371-386.
- Man Z. H., and Yu X. H. (1997). Adaptive terminal sliding mode tracking control for rigid robotic manipulators with uncertain dynamics. *JSME Int. J. of Mechanical Systems, Machine Elements and Manufacturing*, Vol. 40, no. 3, pp.493-502.
- McNeil D., and Freiburger P. (1993). *Fuzzy logic*, NY: Simon & Schuster.

- Mei F., Man Z.H., Yu X.H. (1998). A robust tracking control scheme for a class of nonlinear systems with fuzzy nominal model. *International Journal of Applied Mathematics and Computer Science*, vol.8, no.1, pp.145-158.
- Mendel J. M. (1995). Fuzzy logic systems for engineering: a tutorial. *Proceedings of the IEEE*, Vol. 83, No. 3, pp. 345-377.
- Middleton R. H. and Goodwin, G. C. (1988). Adaptive computed torque control for rigid link manipulators. *Syst. Control Lett.*, vol. 10, pp.9 - 16.
- Mills J. K., and Lockhorst D. M. (1993). Stability and control of robotic manipulators during contact / non-contact transition. *IEEE Transactions on Robotics and Automation*, Vol.9, No.3, pp.335-345.
- Miyasato, Y., and Oshima, Y. (1989). Non-linear adaptive control for robotic manipulators with continuous control input. *Int. J. Control*, vol. 49, pp. 545 - 559.
- Morgan, R. G., and Ozguner, U. (1985). A decentralised variable structure control algorithm for robotic manipulators. *IEEE J. Robotics Automat.*, vol. RA - 1, pp. 57 - 65.
- Morre, B. C. (1976). On the flexibility offered by state feedback in multivariable systems beyond closed loop eigenvalue assignment. *IEEE Trans. Automat. Contr.* vol. AC-21, pp. 689-692.
- Nguyen, H. T., and Nguyen, D. K. (1995). Continuous sliding mode control. *Preprints of the Control 95. Melbourne, Australia*, Vol.1, pp.57-60, October.
- Nicosia, S., and Tomei, P. (1984). Model reference adaptive control algorithms for industrial robots. *Automatica*, pp. 635-644.
- Ogata K. (1990). *Modern Control Engineering*, Prentice-Hall International Inc., Englewood Cliffs, New Jersey.
- Ortega, R., and Spong, M. W. (1989). Adaptive motion control of rigid robots: a tutorial," *Proc. IEEE Conf. Dec. & Contr.*, pp. 1575-1584.

- Ozguner, U., Yurkovich, S., and Abbass, A. F. (1987). Decentralised variable structure control of a two-arm robotic system. *Proc. IEEE Conf. Robotic & Automation*, pp. 1248-1254.
- Palm R. and Rehfuss U. (1997). Fuzzy controllers as gain scheduling approximators. *Fuzzy Sets and Systems*, vol. 85, pp. 233-246.
- Perry T. S. (1995). Lotfi A. Zadeh. *IEEE Spectrum*, June, pp.32-35.
- Petersen, I. R., (1985). Structural stabilization of uncertain linear control systems. *SIAM J. Contr. Optimization*, vol. 23, no. 2, pp. 286-296.
- Procyk T. and Mamdani E. H. (1979). A linguistic self-organising process controller. *Automatica*, Vol. 15, No. 1, pp. 15-30.
- Rehg, J. A. (1985). *Introduction to robotics: a system approach*. Prentice-Hall, INC., Englewood Cliffs, New Jersey.
- Rhee, F. V. D. et al. (1990). Knowledge based Fuzzy control of systems. *IEEE Trans. Automat. contro.* vol. 35, pp. 148 - 155.
- Richter, S., Lefebvre, S., AND Decarlo, R. (1982). Control of a class of nonlinear systems by decentralised control. *IEEE Trans. Automat. Contr.* vol. AC-27, pp. 492 - 494.
- Rohrs, C., Athans, M., and Valavani, L. (1985). Robustness of continuous time adaptive control algorithms in the presence of unmodelled dynamics. *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 881 - 889.
- Ryan, E. P. (1983). A variable structure approach to feedback regulation of uncertain dynamical systems. *Int. J. Control*, vol. 38, pp. 1121 - 1134.
- Samson, C. (1983). Robust nonlinear control of robotic manipulators. in *Proc. 22nd IEEE Conf. Dec.Contr.*, pp. 1211 - 1216.
- Schmiedorf, W. E., and Barmash, B. R. (1986). Robust asymptotic tracking for linear system with unknown parameters. *Automatica*, vol. 22, pp. 353 - 360.

- Schwartz D. G., Klir K. J., Lewis 3 H. W., and Ezawa Y. (1994). Application of fuzzy sets and approximate reasoning. *Proceedings of the IEEE*, Vol. 82, No. 4, pp.482-498.
- Seraji, H. (1989). Decentralised adaptive control of manipulators: theory, simulation, and experimentation. *IEEE Trans. Robotics Automat.* vol. 5, pp. 183-201.
- Shao S. (1988). Fuzzy self-organising controller and its application for dynamic processes. *Fuzzy Sets & Syst.*, Vol. 26, pp. 151-164.
- Shoureshi, R., Momot, M. E., and Roesler, M. D. (1990). Robust control for manipulators with uncertain dynamics. *Automatica*, vol. 26, pp. 353 - 359.
- Shoureshi, R., Momot, M. E., and Roesler, M. D. (1990). Robust control for manipulators with uncertainties. *Automatica*, vol. 26, pp. 353-359.
- Silva, C. M. de. (1984). A motion control scheme for robotic manipulators in *proc. Canadian CAD/CAM and Robotics Conf.* vol. 13, pp. 131 - 137.
- Silva, C. W. de, and MacFarlane, A. G. J. (1989). *Knowledge-based control with application to robots*, Springer - Verlay.
- Singh, S. N. (1985). Adaptive model following control of nonlinear robotic systems. *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 1099 - 1100.
- Sinswat, V., and Fallside, F. (1977). Eigenvalue/eigenvector assignment by state-feedback. *Int. J. Contr.* vol. 26, pp. 389-403.
- Slotine, J-J. E. (1984). Sliding controller design for nonlinear systems. *Int. J. Control*, vol. 40, pp. 421 - 434.
- Slotine, J-J. E. (1985). The robust control of robotic manipulators. *Int. J. Rob. Res.*, vol. 4, pp. 49 - 64.
- Slotine, J-J. E., and Sastry, S. S. (1983). Tracking control of nonlinear system using sliding surface with application to robotic manipulators. *Int. J. Contr.*, vol. 38, no.2, pp. 465 - 492.

- Slotine, J. J. E., and Li, W. (1987). On the adaptive control of robotic manipulators
Int. J. Robotics Research, vol. 6, pp. 49-59.
- Slotine, J. J. E., and Li, W. (1991). Applied nonlinear control. Prentice Hall:
Englewood Cliffs, NJ.
- Song, Y. D., and Gao, W. B. (1991). Path tracking control of robot manipulators via
the vss approach. Int. J. Systems Sci., vol. 22, pp.151 - 163.
- Spong, M. W., and Ortega, R. (1990). On adaptive inverse dynamics control of rigid
robots. IEEE Trans. Automat. Contr. vol. AC-35, pp. 92-95.
- Spong, M. W., and Vidyasagar, M. (1987). Robust linear compensator design for
nonlinear robotic control. IEEE J. Rob. Automat., vol. 3, pp. 345-351.
- Spong, M. W., and Vidyasagar, M. (1989). Robot Dynamics and Control. Wiley,
New York.
- Spong, M. W., Thorp, J. S., and Kleinwaks, J. M. (1987). Robust microprocessor
control of robotic manipulators. Automatica, 23, pp. 373 - 379.
- Spurgeon, S. K. (1991). Choice of discontinuous control component for robust sliding
mode performance. Int. J. Control, vol. 53, pp. 163 - 179.
- Spurgeon, S. K., and Patton, R. J. (1990). Robust variable structure control of model
reference systems. IEE Proceedings, vol. 137, pp. 341 - 348.
- Su, C. Y., and Leung, T. P. (1993). A Sliding Mode Controller with Bound
Estimation for Robot Manipulators. IEEE Transactions on Robotics and
Automation, Vol 9, No. 2, pp 208-214.
- Sugeno M. and Nashida M. (1985). Fuzzy control of a model car. Fuzzy Sets &
Syst., Vol. 16, pp. 103-113.
- Sundareshan, M. K., and Koeing, M. A. (1985). Decentralised model reference
adaptive control of robotic manipulators. Proc. American Control Conf., pp. 44
- 49.

- Takagi, T., and Sugeno, M. (1985). Fuzzy identification of systems and its application to modelling and control. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 15, pp. 116-132.
- Tarn, T. J., and Bejczy, A. K., Isidori, A. and Chen, Y. (1984). Nonlinear feedback in robotic arm control. in *Proc. IEEE Conf. Dec. and Contr*, pp. 1569-1573.
- Truong T. T. and Hofmann W. (1995). A new speed controller with fuzzy tuning. *Proceedings of the IEEE 21st International Conference on Industrial Electronics, Control, and Instrumentation (IECON'95)*, Orlando, USA, Vol. 2, pp. 1484-1489.
- Tseng H. C. and Hwang V. H. (1993). Servocontroller tuning with fuzzy logic. *IEEE Trans. Contr. Syst. Technology*, Vol. 1, No. 4, pp. 262-269.
- Utkin, V. I. (1971). Equations of the sliding regime in the discontinuous systems. *Automat. Remote Contr.*, vol, 32, pp. 1987 - 1907.
- Utkin, V. I. (1977). Variable structure systems with sliding mode. *IEEE Trans. Automat. Contr.*, vol. AC-22, pp. 212 - 221.
- Utkin, V. I. (1983). Variable structure systems - present and future. *Automat. remote Contr.*, vol. 44, pp. 1105 - 1120.
- Utkin, V. I., and Young, K-K. D. (1978). Methods for constructing discontinuous planes in multidimensional variable structure systems. *Automation and Remote Control*, vol. 31, pp. 1466 - 1470.
- Van Brussel, K., and Vastmans, L. (1984). A compensation method for the dynamic control of robots. *Proc. Conf. Robotics Research*.
- Wang L. X., Mendel J. M. (1992). Fuzzy Basis Function, Universal Approximation, and Orthogonal Least-Squares Learning. *IEEE Transactions on Neural Networks*, vol. 3, pp.807-814.

- Warwick, K., and Pugh, A. (1988). Robust control: theory and applications. Peter Peregrinus Ltd., London, U. K.
- White, B. A., and Silson, P. M. (1984). Reachability in variable structure control systems. IEE Proc. vol. 131, pt. D., pp. 85-91.
- Whitehead, M., and Kamen, E. W. (1985). Control of serial manipulators with unknown variable loading. in Proc. IEEE Conf. Dec. & Contr., pp.
- Xu, X. H., Wu, Y. H., and Huang, W. H. (1990). Variable-structure control approach of decentralised model reference adaptive systems. IEE Proc. pt. D, vol. 137, pp. 302-306.
- Yager R. R. and Filev D. P. (1994). Essentials of fuzzy modelling and control. NY: John Wiley & Sons, Inc..
- Yeung, K. S., and Chen, Y, P. (1988). A new controller design for manipulators using the theory of variable structure systems. IEEE Trans. Automat. Contr., vol. AC - 33, pp. 200-206.
- Ying, H., Siler, W., and Buckley, J. J. (1990). Fuzzy control theory: a nonlinear case. Automatica, vol. 26, pp. 513 - 520.
- Young, K-K. D. (1978). Controller design for a manipulator using theory of variable structure systems. IEEE Trans. on Systems, Man, and Cybernetics, vol. 8, pp. 101 - 109.
- Young, K-K. D. (1978). Design of variable structure model following control system. IEEE Trans. Automat. Contr. vol. AC - 23, pp. 1079 - 1085.
- Young, K-K. D. (1988). A variable structure model following control design for robotics applications," IEEE Trans. Automat. Contr. vol. 4, pp. 556-561.
- Young, K-K. D., and Kwatny H. G. (1982) Variable structure servomechanism design and applications to overspeed protection control. Automatica, vol. 18, pp. 385 - 400.

- Young, K. K. D. (1988). A variable structure model following control design for robotics applications. *IEEE Trans. Automat. Contr.* vol. AC - 4, pp. 556-561.
- Yu X. H., and Man Z. H. (1996). Model reference adaptive control systems with terminal sliding modes. *Int. J. Control*, vol. 64, no. 6, pp. 1165-1176.
- Yu X. H., and Man Z. H. (1998). Adaptive sliding mode approach for learning in a feedforward neural network. to appear in *J. of Neural Computing & Applications*.
- Yu X. H., and Man Z. H. (1998). Adaptive terminal sliding mode control of SISO linear systems. to appear in *Automatica*.
- Yu X. H., and Man Z. H. (1998). Fuzzy sliding mode control for linear systems. to appear in *Fuzzy Set and Systems*.
- Zadeh, L. A., (1965). Fuzzy sets. *Inform. Control*, Vol.8, pp.338-353.
- Zadeh L. A. (1972). The rationale for fuzzy control. *J. Dynamic Syst. Meas. Contr.*, pp. 3-4.
- Zadeh L.A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. Syst., Man, Cybern.*, Vol. SCM-3, No. 1, pp.28-44.
- Zadeh L. A. (1975). The concept of a linguistic variable and its application to approximate reasoning. *Inf. Science*, Vol. 8, pp. 199-249.
- Zadeh L. A. (1985). Industrial applications of fuzzy control. Sugeno M., Ed. Amsterdam: North Holland.
- Zadeh L. A. (1996). Computing with words - a paradigm shift. *Proceedings of the 1st International Discourse on Fuzzy Logic and the Management of Complexity (FLAMOC'96)*, Sydney, Australia, Vol. 1, pp. 3-10.

Zhou F., Fisher D. G. (1992). Continuous sliding mode control. *Int. J. Control*, vol. 55, no.2, pp.313-327.

Zimmermann H. J. (1985). *Fuzzy set theory and its applications*. Norwell, MA: Kluwer Academic Publishers.

Appendix A

Hardware setup for a five-bar robot arm

A photograph of the laboratory setup for a five bar robot manipulator is displayed in Fig. A.1. The illustrative and schematic diagrams are shown in Fig. A.2 and A.3. The physical parameters are measured as follows,

$$m_1 = 0.11kg$$

$$m_2 = 0.1kg$$

$$m_3 = 0.1kg$$

$$m_4 = 0.055kg$$

$$l_1 = 0.275m$$

$$l_2 = 0.165m$$

$$l_3 = 0.275m$$

$$l_4 = 0.32m$$

$$l_{c1} = 0.12m$$

$$l_{c2} = 0.07m$$

$$l_{c3} = 0.135m$$

$$l_{c4} = 0.07m$$

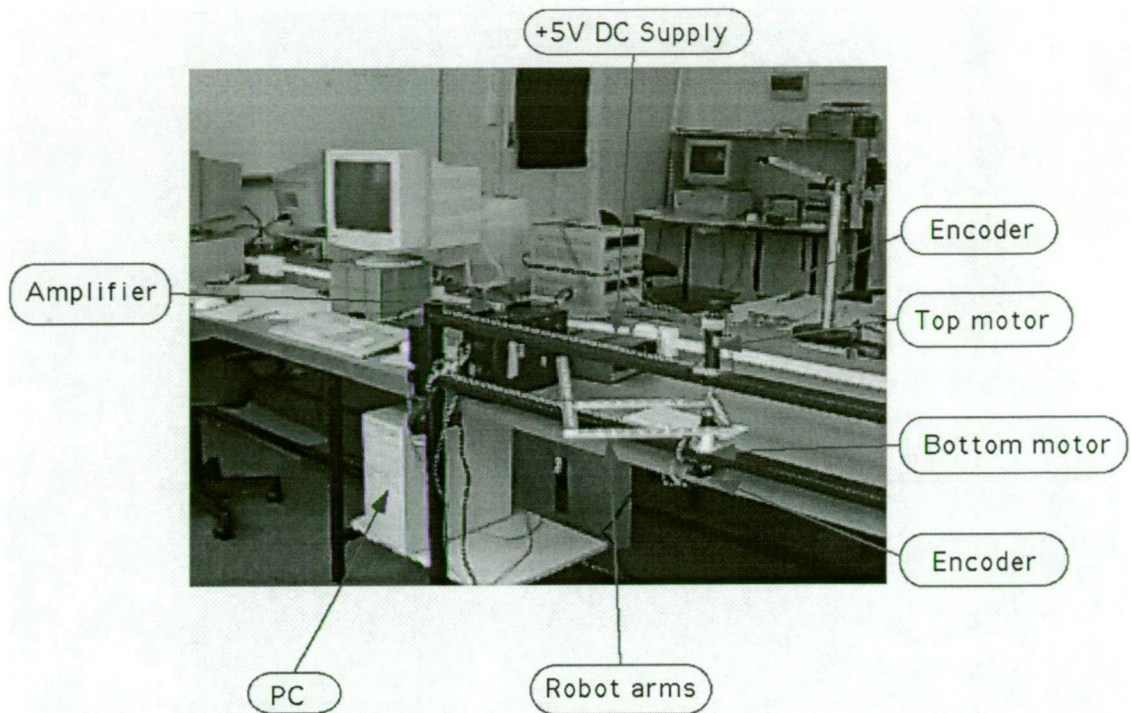


Fig.A.1 A photograph of the laboratory setup for a five-bar robot manipulator

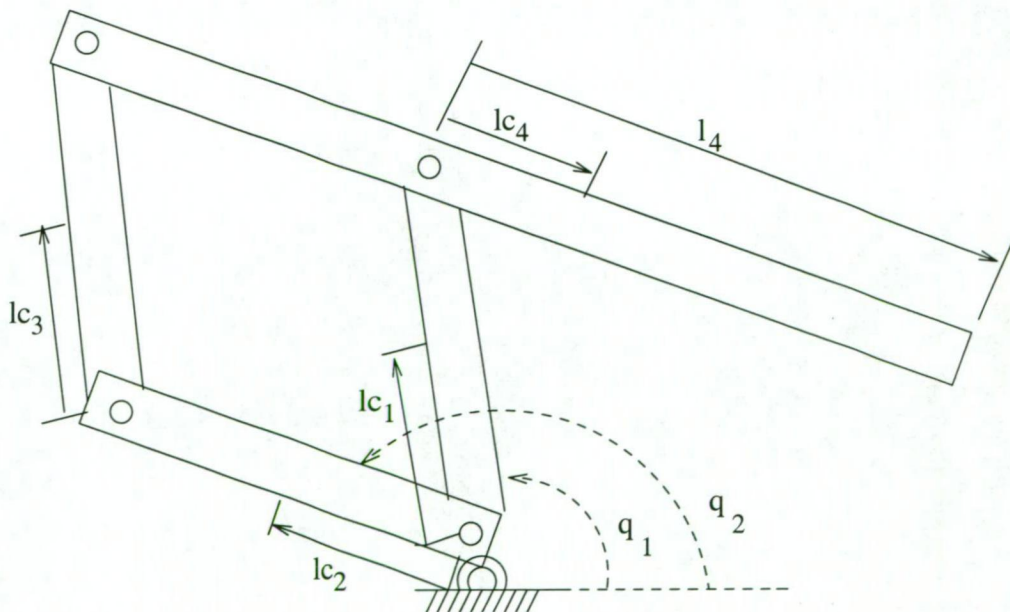


Fig.A.2 Illustrative diagram of a five-bar robot arm

Two interface cards are installed in the computer with a Pentium Pro/200MHz CPU. The PC-30D by Eagle Tech consists of two 12-bit D/A converters with full scale output range from 0 to +10V, the output signals are fed to robot motors through a Servo amplifier by Baldor (TSD-050-05-02-I, operated in torque mode). The PCL-833 by Advantech is a 3-axis quadrature encoder and counter add-on card, and receives quadrature signals from DC-Tacho/Encoders (with 500 counts/rev) mounted on the motor shafts for joint angle measurements. DC motors by Maxon (RE035-071-39, gear ratio 86:1) provide the joint actuation. Fig. A.4 –5 show the interface wiring.

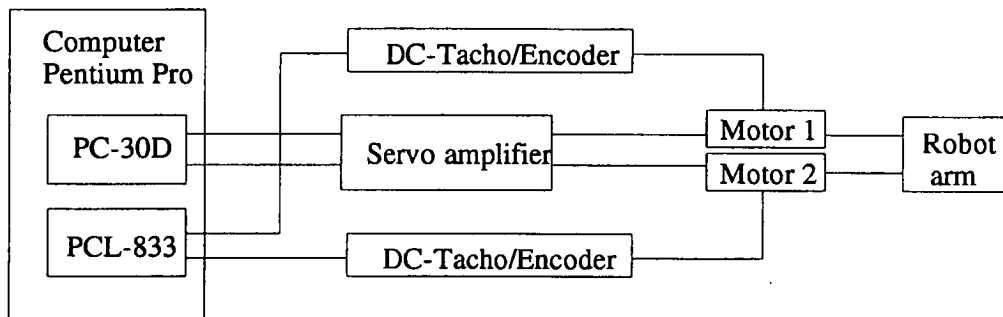


Fig. A.3. Schematic diagram of the five-bar robot arm

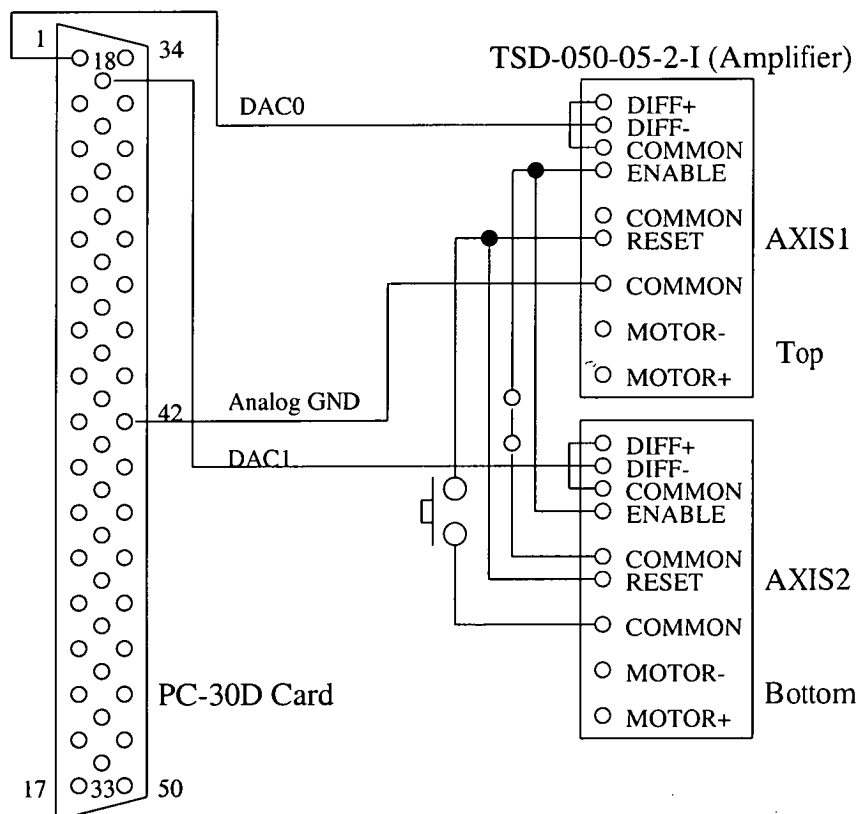


Fig. A.4 Interface wiring of PC30-D with servo amplifier

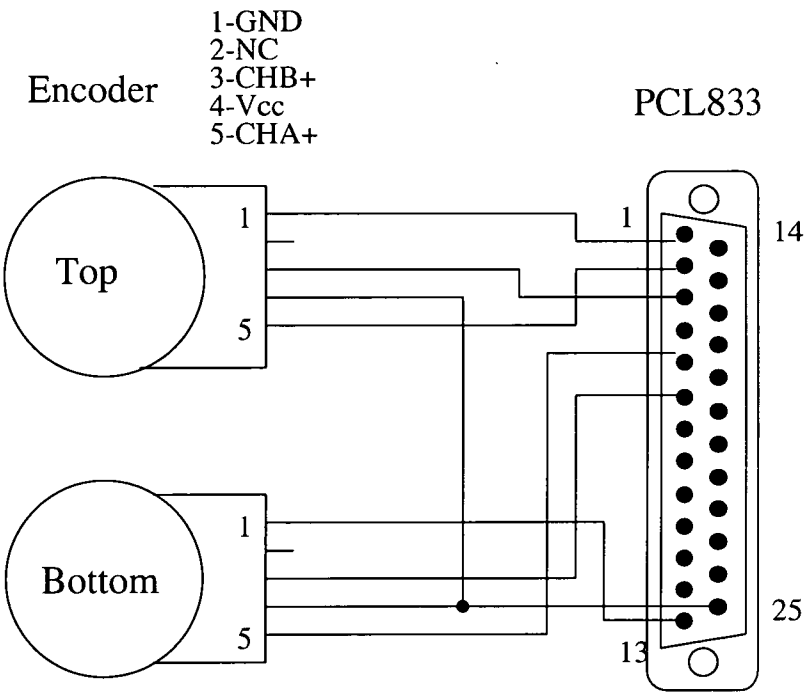


Fig. A.5 Interface wiring of PCL-833 card

Appendix B

C++ programs for a fuzzy sliding mode controller

FSMC.cpp

```
/*  
  
    A fuzzy sliding mode controller for a five bar robot  
manipulator  
  
*/  
//#include <iomanip.h>  
#include <iostream.h>  
  
#include <conio.h>  
#include <stdlib.h>  
#include <stdio.h>  
#include <fstream.h>  
  
#include "container.h"  
#include "defines.h"  
#include "my_math.h"  
#include "PC30 GiveIO.h"  
#include "pcl833.h"  
#include "My_Process.h"  
#include "FSMC Data.h"  
#include "Misc.h"  
#include "Path.h"  
#include "float.h"  
#include "Least Squares.h"  
#include "Vector.h"  
  
const char* dumpFile = "c:\\Temp\\Dump1";  
char* definesFile = "FSMC Data.txt";  
const numDefines = 23;
```

```
double h;
double Ts; // sample period
```

```

int k = 0;    //sample no.
int cycleNum = 0;

const int dataChannels =14;// 20;//20;//22;
const int dataSize = 1000;    //500

const double minSafe    = 0.4;    //0.4
const double maxSafe    = 2.75;    //2.75
const double speedLimit = 0.1;
const double brakeTorque= 5.0;

Vector torque;

array2D_double telemetry, ref, path;

//.....
//
//    DEFINES - loaded from txt file
//

double    Tcycle,numCycles,  Amp1,  AmQ1,  Q10,  Q20,  Q1Dot0,  Q2Dot0,
Q1r0, Q2r0,
          Q1rDot0,  Q2rDot0,modelGain,  maxTorque,  B1,  Q1Step,
Q2Step,forgetFactor;
int      StepsPerSample,sampleRate, sampleDelay, numLSQterms;
// circular path parameters

double revs;  // cycles per second of desired circular path

//=====

/*
    u = h + phi*[-Q_FSMC*sign(s) - Ks*s - C1*(q'-qr') + qr'']
    er = -Q_FSMC*sign(s) - Ks*s - C1*(q'-qr')
    temp = er + qr''
*/

//=====

Vector u(Vector& x, double t)
{
    Vector U;
    double er1,er2,_q1,_q2,_q1_Dot,_q2_Dot,phi11,phi22,phi12,
temp1,temp2,q1SMC,q2SMC,V1,V2,V1_Dot,V2_Dot,
h1,h2,S1,S2,S1_Dot,S2_Dot;

    _q1      = (x._(Q1) - ref._(k,Q1r));    // ~q1 = q1-q1r
    _q2      = (x._(Q2) - ref._(k,Q2r));
    _q1_Dot  = (x._(Q1_Dot) - ref._(k,Q1rDot));
    _q2_Dot  = (x._(Q2_Dot) - ref._(k,Q2rDot));

    if(_q1 > PI)  _q1 = _q1 - 2*PI;
    if(_q1 < -PI) _q1 = _q1 + 2*PI;

    if(_q2 > PI)  _q2 = _q2 - 2*PI;
    if(_q2 < -PI) _q2 = _q2 + 2*PI;

```



```

S1 = c11*_q1+_q1_Dot;           // S= [S1
S2 = c22*_q2+_q2_Dot;           // S2]

S1_Dot=(S1-S1_Old)/Ts;
S2_Dot=(S2-S2_Old)/Ts;

S1_Old=S1;
S2_Old=S2;

V1=0.5*S1*S1/V1_max;           //normalised Lyapunov function
V2=0.5*S2*S2/V2_max;
if (V1>1.0) V1=1.0;
if (V2>1.0) V2=1.0;

V1_Dot=S1*S1_Dot/V1_Dot_max;    //normalised time derivative
V2_Dot=S2*S2_Dot/V2_Dot_max;
if (fabs(V1_Dot)>1.0) V1_Dot=SIGN(1.0,V1_Dot);
if (fabs(V2_Dot)>1.0) V2_Dot=SIGN(1.0,V2_Dot);

int i1,i2,j1,j2;
i1=(int) (V1*20)+1;
i2=(int) (V2*20)+1;
j1=(int) (V1_Dot*10+10)+1;
j2=(int) (V2_Dot*10+10)+1;

q1SMC=30.0*q00[i1][j1];
q2SMC=30.0*q00[i2][j2];

er1=-q1SMC*SIGN(1.0,S1)-c11*_q1_Dot-Ks*S1;
er2=-q2SMC*SIGN(1.0,S2)-c22*_q2_Dot-Ks*S2;

int index;
index = k;//2 * (int) (ROUND(t/Ts));
temp1 = er1 + Amp1*ref._(k,Q1r) + AmQ1*ref._(k,Q1rDot) +
B1*path._(index,Q1r);
temp2 = er2 + Amp1*ref._(k,Q2r) + AmQ1*ref._(k,Q2rDot) +
B1*path._(index,Q2r);

phi11=0.0324;
phi12=-0.00222*cos(x._(Q2)-x._(Q1));
phi22=0.0677;//0.00677;
h1= 0.00222*sin(x._(Q2)-x._(Q1))*x._(Q2_Dot)*x._(Q2_Dot);
h2=-0.00222*sin(x._(Q2)-x._(Q1))*x._(Q1_Dot)*x._(Q1_Dot);

U.set(Q1, ABSLIMIT(h1+phi11*temp1+phi12*temp2,maxTorque));
U.set(Q2, ABSLIMIT(h2+phi12*temp1+phi22*temp2,maxTorque));

int numToSkip = samplesPerCycle / dataSize;

if (k % numToSkip ==0)
//if (k < dataSize)
{
    int kk = (k/numToSkip) + (cycleNum * dataSize);
    //int kk = k;

    telemetry.set(kk,0,ref._(k,Q1r));
    telemetry.set(kk,1,ref._(k,Q2r));
    telemetry.set(kk,2,U._(Q1));
    telemetry.set(kk,3,U._(Q2));
    telemetry.set(kk,4,x._(Q1));

```

```

        telemetry.set(kk, 5, x._(Q2));
        telemetry.set(kk, 6, t+cycleNum*Tcycle);
        telemetry.set(kk, 7, ref._(k, Q1rDot));
        telemetry.set(kk, 8, ref._(k, Q2rDot));
        telemetry.set(kk, 9, x._(Q1_Dot));
        telemetry.set(kk, 10, x._(Q2_Dot));
        telemetry.set(kk, 11, path._(k, Q1r));
        telemetry.set(kk, 12, path._(k, Q2r));
        telemetry.set(kk, 13, sampleDelay);
    }

    //    k++;

    return U;
}

//.....
.....

void startController (pc30 &adCard)
{
    // Set out voltages to 0.0
    adCard.daOut(LWRmot, 0.0);
    adCard.daOut(UPPmot, 0.0);

    // initialise PC30
    adCard.setDefaultClk();
}

//.....

void stopController (pc30 &adCard)
{
    adCard.daOut(LWRmot, 0.0);
    adCard.daOut(UPPmot, 0.0);
}

//.....

void getRobotVel (pcl833 &quadCard, Vector &robotState)
{
    static UD_factor lsq_Q1(numLSQterms, Ts, forgetFactor, 0.0);
    static UD_factor lsq_Q2(numLSQterms, Ts, forgetFactor, 0.0);

    if (k==0)
    {
        lsq_Q1.setThetaN(0, robotState._(Q1));
        lsq_Q1.setThetaN(1, 0);
        lsq_Q2.setThetaN(0, robotState._(Q2));
        lsq_Q2.setThetaN(1, 0);
    }

    lsq_Q1.LS(k, robotState._(Q1));
    lsq_Q2.LS(k, robotState._(Q2));

    robotState.set(Q1_Dot, lsq_Q1.YpDot());
    robotState.set(Q2_Dot, lsq_Q2.YpDot());
}

```

```

//.....

void getRobotPos (pcl833 &quadCard, Vector &robotState)
{
    double      lwrPos,uppPos;

    quadCard.readCounters();

    // Note initial zero position is lower motor zero deg, upp
motor 90deg.
    lwrPos = angle(quadCard.counter1() + quartRev);
    uppPos = angle(quadCard.counter0());

    //cout << "lwrPos= " << lwrPos << "    uppPos= " << uppPos
<<"\n";

    if (lwrPos > oneRevolution) lwrPos = lwrPos - oneRevolution;
    if (uppPos > oneRevolution) uppPos = uppPos - oneRevolution;

    robotState.set(Q1,lwrPos);
    robotState.set(Q2,uppPos);

};

//.....

void emergencyStop(pc30 &adCard,   pcl833 &quadCard,   Vector
&robotState)
{
    int notSafe = TRUE;
    while (notSafe AND (NOT kbhit()))
    {
        if (fabs(robotState._(Q1_Dot)) > speedLimit)
            adCard.daOut(LWRmot,-
SIGN(brakeTorque,robotState._(Q1_Dot)));

        if (fabs(robotState._(Q2_Dot)) > speedLimit)
            adCard.daOut(UPPmot,-
SIGN(brakeTorque,robotState._(Q2_Dot)));

        if ((fabs(robotState._(Q1_Dot)) <= speedLimit) AND
            (fabs(robotState._(Q2_Dot)) <= speedLimit)) notSafe
= FALSE;

        adCard.waitForUsrClk();

        getRobotPos(quadCard, robotState);
        getRobotVel(quadCard, robotState);
    }

    stopController(adCard);
}

//.....

int crashed(pc30 &adCard, pcl833 &quadCard, Vector &robotState)
{
    double diff = robotState._(Q1) - robotState._(Q2);
    if (diff < 0.0) diff = diff + oneRevolution;
    if (diff < minSafe OR diff > maxSafe)
    {

```

```

        emergencyStop(adCard,quadCard,robotState);
        stopController (adCard);
        cout << "EMERGENCY STOP - POSSIBLE CRASH CONDITION : ange
diff = ";
        cout << diff << "rad\n";
        return YES;
    }

    return NO;
}

//.....
.....

void setZeroPosition(pcl833 &quadCard)
{
    char instr = ' ';
    Vector robotState;

    cout << "Please place arms in zero position then enter
'r'.\n\n";
    cout << "Repeat until ok then enter 'q'.... \n";
    cin >> instr;

    quadCard.startCounters();

    while (instr != 'q')
    {
        getRobotPos(quadCard,robotState);
        cout << robotState._(Q1) << " " << robotState._(Q2) << "
";
        cout << robotState._(Q1_Dot) << " " <<
robotState._(Q2_Dot)<< "\n";
        cout.flush();

        cin >> instr;
        if (instr == 'r') quadCard.resetCounters();
    }

    Q10 = robotState._(Q1);
    Q20 = robotState._(Q2);
}

//.....
.....

void doExperiment()
{
    Vector torque,robotState;
    double t;
    int stopNow = FALSE;

    // ks=KS;

    pc30    adCard    = pc30();
    pcl833   quadCard = pcl833();

    startController(adCard);
    setZeroPosition(quadCard);

    setMaxPriority();

```

```

    adCard.waitForUsrClk();

    while ((cycleNum < numCycles) AND (NOT stopNow))
    {
        adCard.waitForUsrClk();

        getRobotPos (quadCard,robotState);
        getRobotVel (quadCard,robotState);

        t = k * Ts;

        if(crashed(adCard,quadCard,robotState))    stopNow = YES;
        else
        {
            torque = u(robotState,t);
            adCard.daOut(LWRmot,torque._(Q1));
            adCard.daOut(UPPmot,torque._(Q2));
            sampleDelay = adCard.divider();
        }

        k++;

        if (k >= samplesPerCycle)
        {
            k=0;
            cycleNum++;
        }
    }

    stopController(adCard);
    restoreStdPriority();
}

//.....
.....

void modelServiceRoutine (Vector x, double t)
{
    static int j=0;

    ref.set(j,Q1r,x._(Q1));
    ref.set(j,Q2r,x._(Q2));
    ref.set(j,Q1rDot,x._(Q1_Dot));
    ref.set(j,Q2rDot,x._(Q2_Dot));

    j++;
}

//.....

Vector modelRKfct(Vector x, double t) // fct(x,t)=xDot=Ax+b
{
    Vector xDot, U;
    double u1,u2,q1r_2Dot,q2r_2Dot;

    int index;

    index = (int) (t/Ts); //(int) ROUND((2.0*t/Ts));
    u1= path._(index,Q1r);

```

```

    u2= path._(index,Q2r);

    qlr_2Dot = B1*u1+AmP1*x._(Q1)+AmQ1*x._(Q1_Dot);
    q2r_2Dot = B1*u2+AmP1*x._(Q2)+AmQ1*x._(Q2_Dot);

    xDot.set(Q1,x._(Q1_Dot));
    xDot.set(Q2,x._(Q2_Dot));
    xDot.set(Q1_Dot,qlr_2Dot);
    xDot.set(Q2_Dot,q2r_2Dot);

    return xDot;
}

//.....
.....

void getReferenceModelResponse()
{
    Vector      modelState;
    int numPathSamples = samplesPerCycle;

    path.setSize(numPathSamples,3);
    ref.setSize(samplesPerCycle,4);

    getSquarePath
(path,numPathSamples,Tcycle,revs*2,Q1r0,Q1Step,Q2r0,Q2Step);

    //getCircularPath(path,numPathSamples,Ts/2,revs,L1,L4,X_centre,
Y_centre,radius);

    modelState.set(Q1,Q1r0);
    modelState.set(Q2,Q2r0);
    modelState.set(Q1_Dot,Q1rDot0);
    modelState.set(Q2_Dot,Q2rDot0);

    rungaKutta(modelState,    0.0,    Tcycle,    Ts,    modelRKfct,
modelServiceRoutine);
}

//.....
.....

void dumpTelemetry()
{
    ofstream dump(dumpFile);
    if (NOT dump) { cout << "Cannot open dump file \n";    exit (-
1); }

    dump << telemetry;

    dump.close();
}

//.....
.....

void loadDefines()
{
    array_double defines(numDefines);

    loadDefines(definesFile,defines);
}

```


FSMC Data.txt

```

NumCycles      = 2
Tcycle         = 10.0           //10.0
sampleRate     = 1000           // 5000in Hz
AmP1           = -6.25          //-6.25 //1.21 // -16      400
AmQ1           = -5             //-5.0  //2.2           // -8      40
Q10            = 1.5708         // robot initial conditions  X[0]= [  $\frac{\pi}{2}$    0   0
0]
Q20            = 0.0            //0.0
Q1Dot0         = 0.0
Q2Dot0         = 0.0
Q1r0           = 2.0            // 0 deg
Q2r0           = 1.5            // 90 deg
Q1rDot0        = 0.0
Q2rDot0        = 0.0
maxTorque      = 10             //.5 //2.0 //5.0
Q1Step         = 3.0
Q2Step         = 0.5
StepsPerSample = 1000           // 10 rk steps per sample
numLSQterms    = 2              // num coefficents in least squares
polynomial
forgetFactor    = 0.7           // recursive lsq forget factor
V1_max=0.009    //0.008;//0.01;//1.0;//10.0;
V2_max=0.015    //0.015;//2.0;//100.0; // 120.0;
V1_Dot_max=2.0  //1.5;//1.0;//5.0;//10.0;//200.0;
V2_Dot_max=2.6  //2.5;//1.0;//20.0;//500.0;//2000.0;

```


FSMC Data.h

```

// 2 degree of freedom robot - Made by Steve Avery, EEE Utas

#if !defined(__FSMC_DATA_H)
#define __FSMC_DATA_H

#include "my_math.h"
#include "defines.h"
#include "pcl833.h"

/* Dynamic Equations


$$\Phi(q)\dot{q} + H(\dot{q}, q) + G(q) = u$$



$$\Phi(q) = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix}$$



$$H(\dot{q}, q) = \begin{bmatrix} h_1(\dot{q}, q) \\ h_2(\dot{q}, q) \end{bmatrix}$$



$$G(q) = \begin{bmatrix} g_1(q) \\ g_2(q) \end{bmatrix}$$



$$q = \text{Transpose}([q_1 \ q_2])$$


$$u = \text{Transpose}([u_1 \ u_2])$$

*/

// Pc30D digital ports
#define POSITION 0
#define STATUS 2
#define CONTROL 1

// Control port - Motor control signal lines
#define MOTOREnable 5 // Connect to both axes of
Controller/Amp
#define MOTORreset 4

// Control port - Position decoder control signal lines
#define READ 0
#define MOTORselect 3 // select LSB or MSB of 16bit position
#define BYTEselect 2
#define COUNTERreset 1

// Status signal lines
#define NOTlatchedMot1 0
#define NOTlatchedMot2 1
#define LATCH1selected 2
#define LATCH2selected 3
#define LATCH3selected 4
#define LATCH4selected 5

// control constants
#define LWRmot 1

```

```

#define UPPmot 0

#define LEASTsigBYTE 0
#define MOSTsigBYTE 1

#define HOLD 1
#define RELEASE 0

#define HELD 0
#define RELEASED 1

#define ENABLE 1
#define DISABLE 0

// encoder constants
#define MULTIPLIER 4
#define GEARratio MULTIPLIER*2225.0/26.0
#define PULSESperREV 500.0

const int oneRev = (int) (GEARratio * PULSESperREV);
const int quartRev = (int) (GEARratio * PULSESperREV / 4);
const double resolution = 2 * PI / (GEARratio * PULSESperREV);

const double oneRevolution = 2*PI;
const double halfRevolution = PI;

#define LWRmotTachoGain 2.051838876782
#define UPPmotTachoGain 2.145587766782

double angle(int count) { return resolution * (count - STARTcount);
}; // count is encoder pulses

#define ADCchannelMask 3

#define LWRpot 1 // Q1
#define UPPpot 2 // Q2

#define TIME 0
#define Q1 1 // Lower motor
#define Q2 2 // Upper motor
#define Q1_Dot 3
#define Q2_Dot 4
#define Q1_2Dot 5
#define Q2_2Dot 6

#define Q1r 0
#define Q2r 1
#define Q1rDot 2
#define Q2rDot 3

#define L1 0.275 // length link 1
#define L4 0.320 // length link 4

// Reference model
/*
        xDot = Am x + bm r

```

```

      x = [ q1      Am = [ 0  0  1  0      bm = [0 0      r = r(t) =[
R1      q2          0  0  0  1          0 0
R2]      q1'        P1 0  Q1 0          1 0
          q2']      0  P1 0  Q1 1          0 1]
(constants in this case)

*/

#endif

```

Complex.cpp

```
// Complex.cpp - implementation of Class Complex
#include "complex.h"
#include "my_math.h"
#include <float.h>
#include <stdlib.h>
```

```
//-----
-----
```

```
// Class Complex
```

```
//Constructor
```

```
Complex::Complex(double r, double i)
{
    realPart = r;
    imagPart = i;
};
```

```
// Destructor
```

```
Complex::~~Complex()
{
    // nothing
};
```

```
//.....
```

```
// Access data members of Complex
```

```
double Complex::real()
{
    if (isFinite())
        return realPart;
    else
        return (0.0);
};
```

```
double Complex::imag()
{
    if (isFinite())
        return imagPart;
    else
        return imagPart;
};
```

```
//.....
```

```
// Return polar form of data
```

```
double Complex::mag()
{
    double m;

    m=sqrt(SQR(real())+SQR(imag()));

    if (_finite(m))
        return m;
    else
    {
```

```

        cout <<(char)7<<"Warning - Complex::mag() infinite result
returned as zero!\n";
        exit(-1);
        return 0.0;
    }

};

double Complex::arg()
{
    if (isZero())
        return 0.0;

    double a;

    a=atan2(imag(),real());

    if (_finite(a))
        return a;
    else
    {
        cout <<(char)7<<"Warning - Complex::arg() infinite result
returned as zero!\n";
        return 0.0;
    }
    return a;
};

//.....

boolean Complex::isZero()
{
    if ((fabs(real()) < TINY) AND (fabs(imag()) < TINY))
        return TRUE;
    else
        return FALSE;
}

//.....

boolean Complex::isFinite()
{
    if (_finite(realPart) AND _finite(imagPart))
        return TRUE;
    else
    {
        cout << (char) 7 << "Warning - Complex::isFinite()
returned FALSE !!!\n";
        return FALSE;
    }
}

//.....

Complex Complex::Exp ()
{
    double          realExp;
    Complex         temp = Complex();

    realExp = exp (real());
    temp.realPart = realExp * cos(imag());
    temp.imagPart = realExp * sin(imag());

```

```

        if (temp.isFinite())
            return temp;
        else
        {
            cout << (char)7 << "Warning - Complex::Exp() infinite
result returned as zero!!\n";
            return Complex(0.0,0.0);
        }
};

//.....

Complex Complex::Sqrt ()
{
    //Yet to be completed

    cout << "Warning -- Complex::sqrt called has not been
completed!\n";

    return Complex(0.0,0.0);
}

//.....
// Addition

Complex      Complex::operator + (double s)
{
    Complex temp = Complex();

    temp.realPart = real()+s;
    temp.imagPart = imag();

    return temp;
};

Complex      Complex::operator + (Complex c)
{
    Complex temp = Complex();

    temp.realPart = real()+c.real();
    temp.imagPart = imag()+c.imag();

    return temp;
};

//.....
// Negate

Complex Complex::operator - ()
{
    Complex temp = Complex();

    temp.realPart = - real();
    temp.imagPart = - imag();

    return temp;
};

//.....

```

```

// Subtraction

Complex Complex::operator - (double s)
{
    Complex temp = Complex();

    temp.realPart = real() - s;
    temp.imagPart = imag();

    return temp;
};

Complex Complex::operator - (Complex c)
{
    Complex temp = Complex();

    temp.realPart = real()-c.real();
    temp.imagPart = imag()-c.imag();

    return temp;
};

//.....
// Multiplication

Complex Complex::operator * (double s)
{
    Complex temp = Complex();

    temp.realPart = real() * s;
    temp.imagPart = imag() * s;

    return temp;
};

Complex Complex::operator * (Complex c)
{
    Complex temp = Complex();

    double m,a;

    m=mag()*c.mag();
    a=arg()+c.arg();

    temp.realPart = m*cos(a);
    temp.imagPart = m*sin(a);

    if(temp.isFinite())
        return temp;
    else
    {
        cout << (char)7<< "Warning - Complex::* infinite result
returned as zero!\n";
        return Complex(0.0,0.0);
    }
};

//.....
// Division

```

```

/*Complex Complex::operator / (double s)
{
    Complex temp = Complex();

    double m,a;

    m=mag()/s;

    temp.realPart = m*cos(imagPart);
    temp.imagPart = m*sin(imagPart);

    return temp;
};*/

Complex Complex::operator / (Complex c)
{
    Complex temp = Complex();

    double m,a;

    m=mag()/c.mag();
    a=arg()-c.arg();

    temp.realPart = m*cos(a);
    temp.imagPart = m*sin(a);

    if(temp.isFinite())
        return temp;
    else
    {
        cout << (char)7<< "Warning - Complex::/ infinite result
returned as zero!\n";
        return Complex(0.0,0.0);
    }
};

//.....
// Assignment

Complex      Complex::operator= (double s)
{
    realPart = s;
    imagPart = 0.0;

    return *this;
};

Complex      Complex::operator= (Complex c)
{
    realPart = c.real();
    imagPart = c.imag();

    return *this;
};

Complex      Complex::operator-= (Complex c)
{
    realPart -= c.real();
    imagPart -= c.imag();
}

```



```

        return *this;
};

//.....
// stream operators

ostream &operator<<(ostream &stream, Complex c)
{
    stream << "( " << c.real() << ", " << c.imag() << " ";

    return stream;
};

//-----
// Misc fcts

Complex exp (Complex c)
{
    double          realExp;
    Complex         temp = Complex();

    realExp = exp (c.real());
    temp.realPart = realExp * cos(c.imag());
    temp.imagPart = realExp * sin(c.imag());

    if(temp.isFinite())
        return temp;
    else
    {
        cout << (char)7<< "Warning - exp(complex) infinite result
returned as zero!\n";
        return Complex(0.0,0.0);
    }
};

//-----
// END Class Complex
////////////////////////////////////

```

Container.cpp

```

#####
/*
    Container classes ie. array, set etc.

*///+++++
+++++
#include <stdlib.h>
#include <math.h>
#include <process.h>    // exit()
#include <iostream.h>   // cout <<
#include <iomanip.h>

#include "defines.h"    // pascal like types etc. eg. int, AND ...
#include "my_math.h"
#include "Container.h"

//=====
// class array_double
//.....

array_double::array_double (int length, double initarray_double) //
constructor
{
    size = length;
    array = new double [size];
    if (NOT array)
    {
        cout << "ERROR- array_double constructor - failure in
'new' operator\n";
        exit(-1);
    }
    for (int i=0; i<size; i++)        array[i] =
initarray_double;
}    // end array_double    ( constructor )
//.....

array_double::~~array_double()
{
    delete [] array;
}

//.....

void array_double::setSize (int length)    // returns TRUE if
successfull
{
    delete [] array;
    array = new double [length];

    size = length;
}    // end array_double::setSize

```

```

//.....

int array_double::setSample (int k, double sample)
{
    if (k < size)        array[k] = sample;
    else
    {
        cout << "ERR (array_double::setSample) index (";
        cout << k << ") exceeds (>=) size (" << size << ")\n";
        int temp;
        cin >> temp;
        exit(-1);
        return FALSE;
    }
    return TRUE;
}    // end array_double::setSample
//.....

double array_double::sample (int k)
{
    if (k < size)        return array[k];
    else
    {
        cout << "ERR (array_double::sample) index (";
        cout << k << ") exceeds (>=) size (" << size << ")\n";
        int temp;
        cin >> temp;
        exit(-1);
        return (-1);
    }
}    //end array_double::sample
//.....

void array_double::append (array_double &d)
{
    int oldSize = size;
    setSize (size+d.length());

    for (int i=0; i < d.length(); i++)  array[i+oldSize]=d._(i);
}
//.....

array_double      &array_double::operator= (array_double &a)
{
    setSize(a.length());

    for (int i = 0; i<size; i++)
        array[i] = a._(i);

    return *this;
};
//.....

ostream &operator<<(ostream &stream, array_double &a)
{
    int i;

    for (i=0;i<a.length();i++)
        //stream << setiosflags( ios::fixed ) << setw(9) <<
        setprecision(5)<< a._(i);
}

```

```

        stream << a._(i) << "  ";

        stream << "\n";

        return stream;
    }

//=====
//=====.....
....

//=====
//=====.....
....
/*class array2D_double
*/

array2D_double::array2D_double (int r, int c, double initData) //
constructor
{
    numRows = r;
    numColumns = c;

    array = new arrayPtr [r];
    for(int i=0; i<r; i++)
    {
        array[i] = new array_double(c,initData);
    }

} // end array2D_double ( constructor )
//.....

array2D_double::~~array2D_double()
{
    for (int i=0; i<numRows; i++) delete array[i];
    delete []array;
}

//.....

void array2D_double::setSize (int r, int c) // returns TRUE if
successfull
{
    int i;

    for ( i=0; i<numRows; i++) delete array[i];
    delete array;

    array = new arrayPtr [r];
    for( i=0; i<r; i++)
    {
        array[i] = new array_double(c,0.0);
    }

    numRows = r;
    numColumns = c;

} // end array2D_double::setSize

```

```

//.....

int array2D_double::setSample (int r, int c, double sample)
{
    if (r < numRows)          array[r]->set(c,sample);
    else
    {
        cout << "ERR (array2D_double::setSample) row (";
        cout << r << ") exceeds (>=) numRows (" << numRows <<
        ")\n";
        int temp;
        cin >> temp;

        exit(-1);
        return FALSE;
    }
    return TRUE;
}

//.....

double array2D_double::sample (int r,int c)
{
    if (r <= numRows)          return array[r]->_(c);
    else
    {
        cout << "ERR (array2D_double::sample) row (";
        cout << r << ") exceeds (>=) numRows (" << numRows <<
        ")\n";
        int temp;
        cin >> temp;
        exit(-1);
        return (-1);
    }
}

//end array2D_double::sample
//.....

array2D_double &array2D_double::operator= (array2D_double &a)
{
    setSize(a.rows(),a.columns());

    for (int i = 0; i<numRows; i++)
        *array[i] = *a.array[i];

    return *this;
};

//.....

ostream &operator<<(ostream &stream, array2D_double &a)
{
    int i;

    for (i=0;i<a.rows();i++) stream << *a.row(i);

    return stream;
}

//      end CONTAINER.CPP
//#####

```

Least Squares.cpp

```

#####
#####
/*
                                     Lest Squares Estimator
                                     _____

Contains objects which implement:

    - Least squares estimation of the process

          
$$y = a_0.x^0 + a_1.x^1 + \dots + a_{(n-1)}.x^{(n-1)}$$


    (1D case eg. given a series of noisy position readings
    at times (x) - find  $a_0, a_1, \dots$  etc)

    using Bierman's U-D factorisation.

    (Sec 13.5 p429 Astrom & Wittenmark
    "Comuter Controlled Systems 2nd ed"
    Prentice-Hall 1990)
*/

// Include modules (provided they have not been included before).

#include "Least Squares.h"
#include <iostream.h>    // for cerr <<
#include <process.h>    // for exit()

//=====
=====

// Member functions for UD_factor class.

// Constructor: sets size of coefficient and co-variance structures
//               and initilises same.
UD_factor::UD_factor (byte numC, double stepSize,
                     double forgetFactor,
                     double initTheta,
                     double initDiag,
                     double initOffDiag)

{
    byte i,j;

    if (numC <= maxCoef) numCoef = numC;
    else
    {
        cerr << "ERR (UD_factor constructor): number of
coefficents too large";
        exit (-1);
    }

    h = stepSize;
    lambda = forgetFactor;

```

```

        for (i=0; i<numCoef; i++)
        {
            theta[i]    = initTheta;
            diag[i]     = initDiag;
        }
        for (j=0; j<SIZEOffDiag(numCoef); j++) offDiag[j] =
initOffDiag;

    }        // end UD_factor constructor

//-----
// Lambda functions

int UD_factor::setLambda (double forgetFactor)    // lambda =
forgetFactor
{
    lambda = forgetFactor;
    return 1;
}

//.....

double UD_factor::Lambda ()                      //
returns lambda
{
    return lambda;
}

//-----
// Plant coefficient functions

// sets one coefficient in theta equal to t
int UD_factor::setThetaN (byte n, double t)
{
    if (n < numCoef)
    {
        theta[n] = t;
        return 0;
    }
    else
    {
        cerr << "ERR (setThetaN): array subscript too large";
        exit (-1);
        return (-1);
    }
}

//.....
// sets all coefficients in theta equal to t
int UD_factor::setAllTheta (double t)
{
    byte i;

    for (i=0; i<numCoef; i++)
        theta[i] = t;
    return 0;
}    // end setAllTheta

//.....
// replaces coefficient vector theta with newTheta
int UD_factor::setTheta (coefVector newTheta)
{
    byte i;

```

```

        for (i=0; i<numCoef; i++)
            theta[i] = newTheta[i];
        return 0;
    } // end setThetat

//.....
// copies coefficient vector, theta, to oldTheta
int UD_factor::getTheta (coefVector& oldTheta)
{
    byte i;

    for (i=0; i<numCoef; i++)
        oldTheta[i] = theta[i];
    return 0;
} // end getTheta

//.....
// Returns nth plant coefficient
double UD_factor::thetaN (byte n)
{
    if (n < numCoef)
        return theta[n];
    else
    {
        cerr << "ERR (thetaN): array subscript too large";
        exit (-1);
        return (-1);
    }
} // end thetaN

//-----
// previous input/output functions

// sets one input/output value to f
int UD_factor::setfiN (byte n, double f)
{
    if (n < numCoef)
    {
        fi[n] = f;
        return 0;
    }
    else
    {
        cerr << "ERR (setfiN): array subscript too large";
        exit (-1);
        return (-1);
    }
} // end setfiN

//.....
// sets all previous input/output values equal to f
int UD_factor::setAllfi (double f)
{
    byte i;

    for (i=0; i<numCoef; i++)
        fi[i] = f;
    return 0;
} // end setAllfi

//.....
// replaces input/output vector, fi, with newfi
int UD_factor::setfi (coefVector newfi)

```



```

{
    byte i;

    for (i=0; i<numCoef; i++)
        fi[i] = newfi[i];
    return 0;
} // end setfi

//.....
// copies input/output vector, fi, to oldfi
int UD_factor::getfi (coefVector& oldfi)
{
    byte i;

    for (i=0; i<numCoef; i++)
        oldfi[i] = fi[i];
    return 0;
} // end getfi

//.....
// Returns nth element of input/output vector
double UD_factor::fiN (byte n)
{
    if (n < numCoef)
        return fi[n];
    else
    {
        cerr << "ERR (fiN): array subscript too large";
        exit (-1);
        return (-1);
    }
} // end fiN
//-----
// Co-variance (diagonal) functions

// sets one co-variance diagonal value to d
int UD_factor::setDiagN (byte n, double d)
{
    if (n < numCoef)
    {
        diag[n] = d;
        return 0;
    }
    else
    {
        cerr << "ERR (setDiagN): array subscript too large";
        exit (-1);
        return (-1);
    }
} // end setDiagN

//.....
// sets all co-variance diagonal values equal to d
int UD_factor::setAllDiag (double d)
{
    byte i;

    for (i=0; i<numCoef; i++)
        diag[i] = d;
    return 0;
} // end setAllDiag

//.....
// replaces co-variance diagonal, diag, with newDiag

```

```

int UD_factor::setDiag (coefVector newDiag)
{
    byte i;

    for (i=0; i<numCoef; i++)
        diag[i] = newDiag[i];
    return 0;
} // end setDiag

//.....
// copies co-variance diagonal, diag, to oldDiag
int UD_factor::getDiag (coefVector& oldDiag)
{
    byte i;

    for (i=0; i<numCoef; i++)
        oldDiag[i] = diag[i];
    return 0;
} // getDiag

//.....
// Returns nth element of co-variance diagonal
double UD_factor::diagN (byte n)
{
    if (n < numCoef)
        return diag[n];
    else
    {
        cerr << "ERR (diagN): array subscript too large";
        exit (-1);
        return (-1);
    }
} // end diagN

//-----
// Co-variance Off-diagonal functions

// sets one co-variance off-diagonal value to o
int UD_factor::setOffDiagN (byte n, double o)
{
    if (n < SIZEOffDiag(numCoef))
    {
        offDiag[n] = o;
        return 0;
    }
    else
    {
        cerr << "ERR (setOffDiagN): array subscript too large";
        exit (-1);
        return (-1);
    }
} // end setOffDiagN

//.....
// sets all co-variance offDiagonal values equal to o
int UD_factor::setAllOffDiag (double o)
{
    byte i;

    for (i=0; i< SIZEOffDiag(numCoef); i++)
        offDiag[i] = o;
    return 0;
} // end setAllOffDiag

```

```

//.....
// replaces co-variance offDiagonal, offDiag, with newOffDiag
int UD_factor::setOffDiag (coefVector newOffDiag)
{
    byte i;

    for (i=0; i<SIZEOffDiag(numCoef); i++)
        offDiag[i] = newOffDiag[i];
    return 0;
} // end setOffDiag

//.....
// copies co-variance offDiagonal, offDiag, to oldOffDiag
int UD_factor::getOffDiag (coefVector& oldOffDiag)
{
    byte i;

    for (i=0; i<SIZEOffDiag(numCoef); i++)
        oldOffDiag[i] = offDiag[i];
    return 0;
} // end getOffDiag

//.....
// Returns nth element of co-variance offDiagonal
double UD_factor::offDiagN (byte n)
{
    if (n < SIZEOffDiag(numCoef))
        return offDiag[n];
    else
    {
        cerr << "ERR (offDiagN): array subscript too large";
        exit (-1);
        return (-1);
    }
} // end offDiagN

//.....
// Calculate least squares approximation for plant
// coefficients (theta) using (recursive) U-D factorisation

int UD_factor::LS (int k, double y)
{
    int          kf, ku, i, j, n;
    double        perr, fj, vj, alphaj, ajlast, pj, w;
    coefVector    kVect;

    n = numCoef;
    perr = y;
    i=0;
    for (i=0; i<n; i++)
        perr = perr - theta[i] * fi[i];
    // Calculate gain and covariance using U-D method
    fj      = fi[0];
    kVect[0] = vj = diag[0]*fj;
    alphaj = 1.0 + vj*fj;
    diag[0] = diag[0]/alphaj/lambda;
    if (n>1)
    {
        kf = ku = -1;
        for (j=1; j<n; j++)
        {
            fj = fi[j];
            for (i=0; i<j; i++)          // f = fi*U

```

```

        {      kf++;
            fj += fi[i]*offDiag[kf];
        } // end for i
        vj = fj*diag[j];                // v=D*f
        kVect[j] =vj;
        ajlast = alphaj;
        alphaj = ajlast + vj*fj;
        diag[j] = diag[j] * ajlast/alphaj/lambda;
        pj = -fj/ajlast;
        for (i=0; i<j; i++)                // kj+1 = kj+vj*uj,
uj=uj+pj*kj
        {      ku++;
            w = offDiag[ku] + kVect[i]*pj;
            kVect[i] += offDiag[ku]*vj;
            offDiag[ku] = w;
        } //end for i
    } // end for j
} // end if n>1
// update coefficient estimates
for (i=0; i<n; i++)    theta[i] += perr*kVect[i]/alphaj;
updateFi (k);

return 0;
} // end LS

//.....
// calculate prediction for next output.
double UD_factor::Yp ()
{
    byte i;
    double y = 0.0;

    for (i=0; i<numCoef; i++)    y += theta[i]*fi[i];
    return y;
} // end Yp
//.....
// calculate derivative
double UD_factor::YpDot ()
{
    byte i;
    double y = 0.0;

    for (i=1; i<numCoef; i++)    y += i*theta[i]*fi[i-1];
    return y;
} // end Yp

//.....
// Update previous input/output vector, fi, by one sample period
// value of fcts evaluated for x=xk
// ie. { 1, x, x^2, ... ,x^n-1 } for x=xk
int UD_factor::updateFi (int k)
{
    byte i;

    double x = k * h;

    fi[0] = 1.0;
    for (i=1; i<numCoef; i++)
        fi[i] = fi[i-1] * x;

    return 0;
}

```

```
}      // end updateFI

//.....
...
// Update previous input/output vector and return next prediction for
Y
double UD_factor::nextYp (int k)
{
    updateFi (k);
    return Yp();
}      // end nextYp

//.....
.....

// end member functions for class UD_factor
//=====
=====

// end ADAPTIVE.CPP
//#####
#####
```

Misc.cpp

```
#include "Misc.h"
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

// load defines from definesFile into arrayOfDouble
// each line must be in form
//
// identifier = value    // with or without inline comments
//
void loadDefines (char *definesFile, array_double &defines)
{
    ifstream file(definesFile);
    if (NOT file) { cout << "Cannot open defines file \n";    exit
(-1); }

    double defValue;

    for (int i=0; i<defines.length() ; i++)
    {
        file.ignore(60, '='); // ignore identifier
        file >> defValue; // read value
        file.ignore(80, '\n'); // ignore to end of line

        defines.set(i, defValue);
    }

    file.close();
}
```

My_math.cpp

```

#include "my_math.h"
#include "defines.h"

//+++++
// class Vector
//-----

Vector::Vector (word l,double initValue)
{
    word i;
    size = l;

    for (i=0;i< maxElements;i++)    vector[i] = initValue;
}

Vector::~Vector(){}

int Vector::sizeOf ()      { return size; }
void Vector::setSize(int l) { size = l; }

int Vector::setVector(arrayOfDouble &v)
{
    for (byte i = 0; i< size; i++)
        vector [i] = v[i];

    return 1;
}

void Vector::setElement (word i, double s) { vector[i] = s; }
double Vector::getElement (word i) {return vector[i];}

double Vector::norm()
{
    double result=0.0;

    for(byte i = 0; i<size; i++)
        result += SQR(vector[i]);

    return sqrt(result);
}

Vector Vector::operator+ (double s)
{
    Vector temp=Vector(size);

    for (byte i =0; i < size; i++)
        temp.vector[i] = vector[i] + s;

    return temp;
}

Vector Vector::operator+ (Vector v)
{

```

```

        Vector temp = Vector(size);

        for (byte i =0; i <  size; i++)
            temp.vector[i] =  vector[i] + v.vector[i];

        return temp;
    }

Vector Vector::operator- (Vector v)
{
    Vector temp = Vector(size);

    for (byte i =0; i <  size; i++)
        temp.vector[i] =  vector[i] - v.vector[i];

    return temp;
}

Vector      Vector::operator* (double s)
{
    Vector temp = Vector(size);

    for (byte i = 0; i< size; i++)
        temp.vector[i] =  vector[i] * s;

    return temp;
};

Vector      Vector::operator * (Vector v)
{
    Vector temp = Vector(size);

    for (byte i = 0; i< size; i++)
        temp.vector[i] =  vector[i] * v.vector[i];

    return temp;
};

Vector      Vector::operator/ (double s)
{
    Vector temp = Vector(size);

    for (byte i = 0; i< size; i++)
        temp.vector[i] =  vector[i] / s;

    return temp;
};

Vector      Vector::operator / (Vector v)
{
    Vector temp = Vector(size);

    for (byte i = 0; i< size; i++)
        temp.vector[i] =  vector[i] / v.vector[i];

    return temp;
};

```



```

};

double      Vector::operator & (Vector v)
{
    double temp = 0.0;

    for (byte i = 0; i<  size; i++)
        temp = temp +  vector[i] * v.vector[i];

    return temp;
};

Vector  Vector::operator= (double s)
{
    for (byte i = 0; i<  size; i++)
        vector[i] = s;

    return *this;
};

Vector      Vector::operator= (Vector v)
{
    size = v.size;

    for (byte i = 0; i<  size; i++)
        vector[i] = v.vector[i];

    return *this;
};

ostream &operator<<(ostream &stream, Vector v)
{
    word i;

    stream << "{ ";

    for (i=0;i<v.size-1;i++)
        stream << v.getElement(i) << ", ";

    stream << v.getElement(v.size-1) << " }";

    return stream;
}

//  end class Vector.....

//+++++
// class  complexVector
//-----

complexVector::complexVector (word l)
{
    word i;

    size = l;

```

```
        for (i=0;i< maxElements;i++)    vector[i] = Complex(0.0,0.0);
    }

complexVector::~complexVector()
{
}

int complexVector::sizeOf()
{
    return size;
}

void complexVector::setSize(int l)
{
    size = l;
}

int complexVector::setVector(arrayOfComplex &v)
{
    for (byte i = 0; i< size; i++)
        vector [i] = v[i];

    return 1;
}

void complexVector::setElement (word i, double s)
{
    vector[i] = Complex(s,0.0);
}

void complexVector::setElement (word i, Complex z)
{
    vector[i] = z;
}

Complex complexVector::getElement (word i)
{
    return vector[i];
}

double complexVector::norm()
{
    double result=0.0;

    for(byte i = 0; i<size; i++)
        result += SQR(vector[i].mag());

    return sqrt(result);
}

complexVector complexVector::real()
{
    complexVector temp;

    for (byte i = 0; i< size; i++)
        temp.vector [i] = vector[i].real();

    return temp;
}
```

```
}
```

```
complexVector complexVector::operator+ (double s)
{
    complexVector temp=complexVector(size);

    for (byte i =0; i < size; i++)
        temp.vector[i] = vector[i] + s;

    return temp;
}
```

```
complexVector complexVector::operator+ (Complex z)
{
    complexVector temp = complexVector(size);

    for (byte i =0; i < size; i++)
        temp.vector[i] = vector[i] + z;

    return temp;
}
```

```
complexVector complexVector::operator+ (complexVector v)
{
    complexVector temp = complexVector(size);

    for (byte i =0; i < size; i++)
        temp.vector[i] = vector[i] + v.vector[i];

    return temp;
}
```

```
complexVector complexVector::operator- (complexVector v)
{
    complexVector temp = complexVector(size);

    for (byte i =0; i < size; i++)
        temp.vector[i] = vector[i] - v.vector[i];

    return temp;
}
```

```
complexVector complexVector::operator* (double s)
{
    complexVector temp = complexVector(size);

    for (byte i = 0; i< size; i++)
        temp.vector[i] = vector[i] * s;

    return temp;
};
```

```
complexVector complexVector::operator* (Complex z)
{
    complexVector temp = complexVector(size);
```

```

        for (byte i = 0; i < size; i++)
            temp.vector[i] = vector[i] * z;

        return temp;

};

complexVector    complexVector::operator * (complexVector v)
{
    complexVector temp = complexVector(size);

    for (byte i = 0; i < size; i++)
        temp.vector[i] = vector[i] * v.vector[i];

    return temp;

};

complexVector    complexVector::operator/ (double s)
{
    complexVector temp = complexVector(size);

    for (byte i = 0; i < size; i++)
        temp.vector[i] = vector[i] / s;

    return temp;

};

complexVector    complexVector::operator/ (Complex z)
{
    complexVector temp = complexVector(size);

    for (byte i = 0; i < size; i++)
        temp.vector[i] = vector[i] / z;

    return temp;

};

complexVector    complexVector::operator / (complexVector v)
{
    complexVector temp = complexVector(size);

    for (byte i = 0; i < size; i++)
        temp.vector[i] = vector[i] / v.vector[i];

    return temp;

};

Complex    complexVector::operator & (complexVector v)
{
    Complex temp = (0.0,0.0);

    for (byte i = 0; i < size; i++)
        temp = temp + vector[i] * v.vector[i];

    return temp;

```

```

};

complexVector  complexVector::operator= (double s)
{
    for (byte i = 0; i<  size; i++)
        vector[i] = s;

    return *this;
};

complexVector  complexVector::operator= (complexVector v)
{
    size = v.size;

    for (byte i = 0; i<  size; i++)
        vector[i] = v.vector[i];

    return *this;
};

ostream &operator<<(ostream &stream, complexVector v)
{
    word i;

    stream << "{ ";

    for (i=0;i<v.size-1;i++)
        stream << v.getElement(i) << ", ";

    stream << v.getElement(v.size-1) << " }";

    return stream;
}

//  end class complexVector.....

//+++++
//class  complexMatrix
//-----

complexMatrix::complexMatrix (word l)
{
    word i;

    size = l;

    for (i=0;i< size;i++)
        matrix[i] = complexVector(l);
}

complexMatrix::~~complexMatrix()
{
}

void  complexMatrix::makeSize(word l)
{
    for (byte i=0; i<size; i++)
        matrix[i].size = l;
}

```

```
        size = 1;
    }

int complexMatrix::setMatrix(matrixOfComplex &m)
{
    for (byte i = 0; i < size; i++)
        matrix[i] = m[i];

    return 1;
}

void complexMatrix::setElement (word i, word j, Complex z)
{
    matrix[i].setElement(j,z);
}

Complex complexMatrix::getElement (word i, word j)
{
    return matrix[i].getElement(j);
}

complexVector complexMatrix::getVector (word i)
{
    return matrix[i];
}

complexMatrix complexMatrix::operator+ (double s)
{
    complexMatrix temp=complexMatrix(size);

    for (byte i =0; i < size; i++)
        temp.matrix[i] = matrix[i] + s;

    return temp;
}

complexMatrix complexMatrix::operator+ (Complex z)
{
    complexMatrix temp = complexMatrix(size);

    for (byte i =0; i < size; i++)
        temp.matrix[i] = matrix[i] + z;

    return temp;
}

complexMatrix complexMatrix::operator+ (complexMatrix m)
{
    complexMatrix temp = complexMatrix(size);

    for (byte i =0; i < size; i++)
        temp.matrix[i] = matrix[i] + m.matrix[i];

    return temp;
}
```

```

complexMatrix    complexMatrix::operator* (double s)
{
    complexMatrix temp = complexMatrix(size);

    for (byte i = 0; i < size; i++)
        temp.matrix[i] = matrix[i] * s;

    return temp;
};

complexMatrix    complexMatrix::operator* (Complex z)
{
    complexMatrix temp = complexMatrix(size);

    for (byte i = 0; i < size; i++)
        temp.matrix[i] = matrix[i] * z;

    return temp;
};

complexMatrix    complexMatrix::operator * (complexMatrix m)
{ //{{a1*b1, a2*b2}, {a3*b3, a4*b4}}
    complexMatrix temp = complexMatrix(size);

    for (byte i = 0; i < size; i++)
        temp.matrix[i] = matrix[i] * m.matrix[i];

    return temp;
};

complexMatrix    complexMatrix::operator/ (double s)
{
    complexMatrix temp = complexMatrix(size);

    for (byte i = 0; i < size; i++)
        temp.matrix[i] = matrix[i] / s;

    return temp;
};

complexMatrix    complexMatrix::operator/ (Complex z)
{
    complexMatrix temp = complexMatrix(size);

    for (byte i = 0; i < size; i++)
        temp.matrix[i] = matrix[i] / z;

    return temp;
};

complexMatrix    complexMatrix::operator / (complexMatrix v)
{
    complexMatrix temp = complexMatrix(size);

```

```

        for (byte i = 0; i < size; i++)
            temp.matrix[i] = matrix[i] / v.matrix[i];

        return temp;

};

complexVector      complexMatrix::operator & (complexVector v)
{
    complexVector temp = complexVector(size);

    for (byte i = 0; i < size; i++)
        temp.vector[i] = matrix[i] & v;

    return temp;

};

complexMatrix      complexMatrix::operator= (double s)
{
    for (byte i = 0; i < size; i++)
        matrix[i] = s;

    return *this;
};

complexMatrix      complexMatrix::operator= (complexMatrix v)
{
    size = v.size;

    for (byte i = 0; i < size; i++)
        matrix[i] = v.matrix[i];

    return *this;
};

ostream &operator<<(ostream &stream, complexMatrix m)
{
    word i;

    stream << "{";

    for (i=0;i<m.size-1;i++)
        stream << m.getVector(i) << ",\n";

    stream << m.getVector(m.size-1) << "}\n";

    return stream;
}

//.....
//  ...special functions ...
//.....

// end class complexMatrix.....

//+++++
//class      identityMatrix

```



```

//-----
identityMatrix::identityMatrix (word l) : complexMatrix(l)
{
    word i;
    Complex one = Complex(1.0,0.0);

    size = l;

    for (i=0;i< size;i++)
    {
        matrix[i] = 0.0;
        matrix[i].setElement(i,one);
    }
}

identityMatrix::~~identityMatrix()
{
    //identityMatrixs--;
    //cout << "num vectors = " << identityMatrixs << "\n";
};

// end class identityMatrix.....

//+++++
//class    linearSolve
//-----

//    complexMatrix LUdecomposition;
//    int *rowPermutations [maxElements];
//    float rowInterchanges;

linearSolve::linearSolve (word l) : complexMatrix (l)
{
    byte i;

    LU.size=l;
    for (i=0;i<LU.size;i++)
        LU.matrix[i]=complexVector(LU.size);
}

linearSolve::~~linearSolve(){}

linearSolve linearSolve::operator= (complexMatrix m)
{
    size = m.size;

    for (byte i = 0; i< (int)size; i++)
        matrix[i] = m.matrix[i];

    return *this;
};

linearSolve linearSolve::operator= (linearSolve l)
{
    LU=l.LU;
    size = l.size;
    for (int j = 0; j< (int)size; j++)
        matrix[j] = l.matrix[j];
    for (int i=0; i<(int)LU.size; i++)

```

```

        rowPermutations[i]=l.rowPermutations[i];
        rowInterchanges = l.rowInterchanges;

        return *this;
};

void linearSolve::doLUdecomposition()
{
//void ludcmp(float **a, int n, int *indx, float *d)

    int i,imax,j,k;
    Complex vv[maxElements];
    Complex temp,big,sum,dum;

    LU.setMatrix(matrix);
    rowInterchanges=1.0; /*d
    for (i=0;i<(int)size;i++) {
        big=Complex(0.0,0.0);
        for (j=0;j<(int)size;j++)
            if ((temp=LU.getElement(i,j)) > big) big=temp;
        if (big == Complex(0.0,0.0))
            //nrerror("Singular matrix in routine ludcmp");
            cout << "Singular Matrix in routine
doLUdecomposition\n";
        vv[i]=Complex(1.0,0.0)/big;
    }
    for (j=0;j<(int)size;j++) {
        for (i=0;i<j;i++) {
            sum=LU.getElement(i,j);
            for (k=0;k<i;k++) sum -=LU.getElement(i,k)*LU.getElement(k,j);
            LU.setElement(i,j,sum);
        }
        big=Complex(0.0,0.0);
        for (i=j;i<(int)size;i++) {
            sum=LU.getElement(i,j);
            for (k=0;k<j;k++)
                sum -= LU.getElement(i,k)*LU.getElement(k,j);
            LU.setElement(i,j,sum);
            if ( (dum=vv[i]*sum.mag()) >= big) {
                big=dum;
                imax=i;
            }
        }
        if (j != imax) {
            for (k=0;k<(int)size;k++) {
                dum=LU.getElement(imax,k);
                LU.setElement(imax,k,LU.getElement(j,k));
                LU.setElement(j,k,dum);
            }
            rowInterchanges= -rowInterchanges; /*d = -(*d);
            vv[imax]=vv[j];
        }
        rowPermutations[j]=imax;
        if (LU.getElement(j,j) == Complex(0.0,0.0))
            LU.setElement(j,j,TINY);
        if (j != (int)size-1) {
            dum=Complex(1.0,0.0)/(LU.getElement(j,j));
            for (i=j+1;i<(int)size;i++)
                LU.setElement(i,j,LU.getElement(i,j)*dum);
        }
    }
}

```

```
}
```

```
complexVector linearSolve::solve(complexVector b)
```

```
{
    int i,ii=-1,ip,j;
    Complex sum = Complex(0.0,0.0);

    for (i=0;i<(int)size;i++) {
        ip=rowPermutations[i];
        sum=b.getElement(ip);
        b.setElement(ip,b.getElement(i));
        if (ii>-1)
            for(j=ii;j<=i-1;j++)
                sum -= LU.getElement(i,j)*b.getElement(j);
        else if (sum!=Complex(0.0,0.0)) ii=i;
        b.setElement(i,sum);
    }
    for (i=(int)size-1;i>=0;i--) {
        sum=b.getElement(i);
        for (j=i+1;j<(int)size;j++)
            sum -= LU.getElement(i,j)*b.getElement(j);
        b.setElement(i,sum/LU.getElement(i,i));
    }
    return b;
}
```

```
// end class linearSolve.....
```

My_Process.cpp

```

/*

class thread
{
    protected:

        DWORD threadID;
        HANDLE threadHandle;

        //.....

    public:

        thread (LPTHREAD_START_ROUTINE threadFctPtr,          //
pointer to thread function
                LPVOID argumentsPtr =
NULL, // argument for new thread
                LPSECURITY_ATTRIBUTES attributesPtr = NULL, //
pointer to thread security attributes
                DWORD initialStackSize = 0,
                // initial thread stack size, in bytes
                DWORD creationFlags = 0);
                // creation flags

        DWORD id      () { return threadID;};
        HANDLE handle () { return threadHandle;};

        DWORD waitFor (DWORD mSec = INFINITE);

};

//-----
-----

#endif
//#####
#####
*/

#include "My_Process.h"
#include "defines.h"
#include <windows.h>
//#include <process.h>
#include <iostream.h>
#include <time.h>

//#####

thread::thread (LPTHREAD_START_ROUTINE threadFunction,          //
pointer to thread function
                DWORD priorityClass,
                DWORD priorityLevel,
                LPVOID argumentsPtr,          // argument for new thread

```

```

        LPSECURITY_ATTRIBUTES  attributesPtr,
// pointer to thread security attributes
        DWORD
initialStackSize, // initial thread stack size, in bytes
        DWORD                                creationFlags)
        // creation flags
{
    threadHandle = CreateThread(attributesPtr,
                                initialStackSize,
                                threadFunction,
                                argumentsPtr,
                                creationFlags,
                                &threadID);

    if (NOT threadHandle)
    {
        cout << "ERROR in CLASS thread - 'CreateThread()'
returned with "
                << GetLastError() << "\n";
        exit(-1);
    }

    setPriorityClass(priorityClass);
    setPriorityLevel(priorityLevel);
}

//.....
...

void thread::setPriorityClass(DWORD priorityClass)
{
    BOOL result;

    HANDLE processHandle = GetCurrentProcess();
    result = SetPriorityClass(processHandle, priorityClass);

    if (NOT result)
    {
        cout << "ERROR in CLASS thread - 'setPriorityClass()'
returned with "
                << GetLastError() << "\n";
        exit(-1);
    }
}

//.....

void thread::setPriorityLevel(DWORD priorityLevel)
{
    BOOL result;

    result = SetThreadPriority(threadHandle, priorityLevel);

    if (NOT result)
    {
        cout << "ERROR in CLASS thread - 'setPriorityLevel()'
returned with "
                << GetLastError() << "\n";
        exit(-1);
    }
}

```

```

//.....
...

DWORD thread::waitFor (DWORD mSec)
{
    DWORD result;

    result = WaitForSingleObject(threadHandle,mSec);

    if (result == WAIT_FAILED)
    {
        cout << "ERROR in CLASS thread - 'waitFor()' returned
with "
            << GetLastError() << "\n";
        exit(-1);
    }

    return result;
}

//-----

void setMaxPriority()
{
    int result = SetPriorityClass(GetCurrentProcess(),
REALTIME_PRIORITY_CLASS);
    if (NOT result)
    {
        cout << "ERROR in setMaxPriority - 'setPriorityClass()'
returned with "
            << GetLastError() << "\n";
        exit(-1);
    }

    result = SetThreadPriority(GetCurrentThread(),
THREAD_PRIORITY_TIME_CRITICAL);
    if (NOT result)
    {
        cout << "ERROR in setMaxPriority - 'setPriorityLevel()'
returned with "
            << GetLastError() << "\n";
        exit(-1);
    }
}

//.....

void restoreStdPriority()
{
    int result = SetPriorityClass(GetCurrentProcess(), 32);
    if (NOT result)
    {
        cout << "ERROR in restoreStdPriority -
'setPriorityClass()' returned with "
            << GetLastError() << "\n";
        exit(-1);
    }

    result = SetThreadPriority(GetCurrentThread(), 0);
    if (NOT result)

```

```
{
    cout << "ERROR in restoreStdPriority -
'setPriorityLevel()' returned with "
        << GetLastError() << "\n";
    exit(-1);
}

}

//.....

void holdProcessFor (int seconds) // stops process for seconds
{
    time_t    start, finish;

    time( &start );
    finish = start;

    while ( difftime(finish,start) < seconds)
        time( &finish );
}

//#####
#####
// end MY_Process.cpp
//#####
#####
```

Path.cpp

```

#include "Path.h"
#include "Container.h"
#include <math.h>

void getCircularPath (array2D_double &path,
                      int          numSamples,      // path
                      double       Ts,              //
                      double       revs,            //
                      double       l1,              //
                      double       l4,              //
                      double       x0,              //
                      double       y0,              //
                      double       r)               //
radius
{
    double C2 = -2*l1;
    double C1 = (l4*l4-l1*l1)/C2;
    double w = 2 * PI * revs;                      // angular velocity

    double t,a,b,L,B,alpha,thetal1,theta2;

    for (int i=0; i<numSamples; i++)
    {
        //t = (2 * PI * i / numSamples)/w;
        t=i*Ts;
        a=x0+r*cos(w*t);
        b=y0+r*sin(w*t);
        L=sqrt(a*a+b*b);
        B=atan(b/a);

        alpha = acos(C1/L-L/C2);
        thetal1 = alpha + B;
        theta2 = asin(L*sin(alpha)/l4)+thetal1;

        path.set(i,0,thetal1);
        path.set(i,1,theta2);
    }
}

//-----

void getSquarePath (array2D_double &path,
                    int          numSamples,
                    double       time,
                    double       cps,
                    double       min1,
                    double       max1,
                    double       min2,
                    double       max2)

```



```
{  
  
    int samplesPerStep = (int)((double)numSamples / time / cps);  
  
    double setPoint1, setPoint2;  
  
    for (int i=0; i<numSamples; i++)  
    {  
        if(ODD(i/samplesPerStep))  
        {  
            setPoint1 = min1;  
            setPoint2 = min2;  
        }else  
        {  
            setPoint1 = max1;  
            setPoint2 = max2;  
        }  
  
        path.set(i,0,setPoint1);  
        path.set(i,1,setPoint2);  
    }  
}
```

PC30 GiveIO.cpp

```

/////////////////////////////////////////////////////////////////
//
// IoCard.cpp
//
// DriverX Version 3
//
// Copyright (C) 1995-1997, Tetradyne Software Inc.
// All rights reserved.

// #include <stdafx.h>
#include <time.h>
#include <iostream.h>

#include "defines.h"      // my defines
#include "Windows.h"
#include "PC30 GiveIO.h"

// #define TRAP_ERRORS
// .....

pc30::pc30()
{
    HANDLE h;

    h = CreateFile("\\\\.\\giveio", GENERIC_READ, 0, NULL,
                  OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
NULL);

    #if defined (TRAP_ERRORS)
    if(h == INVALID_HANDLE_VALUE) {
        cout<<"Couldn't access giveio device\n";
        exit(-1);
    }
    #endif

    CloseHandle(h);

    diocntrl = 0x00;
    digPortA = 0;
    digPortB = 0;
    digPortC = 0;

    // initialise pc30 - pc30D User Manual p68, #6.2
Initialisation
    outPort(ADMDE, 0x92);
    outPort(TMRCTR, 0x34);
    outPort(TMRCTR, 0x74);    //74
    outPort(TMRCTR, 0xB6);
    outPort(ADCCR, 0x02);
    outPort(DIOCNTRL, 0x00);

    clock_t start = clock();
    while (start == clock());    // wait for 1mSec

    Word temp;
    temp = inPort(ADDSR);

```

```

    temp = inPort(ADDSR);

    daOut(0,0.0);
    daOut(1,0.0);

    configureDigPort(DIGportA, INport);
    configureDigPort(DIGportB, OUTport);
    configureDigPort(DIGportC, INport);
}

//.....

pc30::~~pc30()
{
}

//.....
// Digital I/O functions

void pc30::configureDigPort (Word port, Word direction)
{
    if (direction == INport)
    {
        if (port == DIGportA) diocntrl = diocntrl BITor
DIGportAIn;
        else
        if (port == DIGportB) diocntrl = diocntrl BITor
DIGportBIn;
        else
        if (port == DIGportC) diocntrl = diocntrl BITor
DIGportCIn;
    }
    else // direction == OUTport
    {
        if (port == DIGportA) diocntrl = diocntrl BITand
DIGportAOut;
        else
        if (port == DIGportB) diocntrl = diocntrl BITand
DIGportBOut;
        else
        if (port == DIGportC) diocntrl = diocntrl BITand
DIGportCOut;
    }

    outPort(DIOCNTRL, diocntrl);
}
//.....
// note 0->+10V, 4095->-10V

void pc30::daOut(Word port, double value)
{
    #if defined(TRAP_ERRORS)
        // trap illegal port

        if(port < 0 OR port > 1)
        {

```

```

        cout << "pc30::adcOut(" << port << ", " << value << ") -
port not [0..1]\n";
        exit(-1);
    }

    // Limit value to +-10V
    if      (value < -10.0) value = -10.0;
    else if (value >  10.0) value =  10.0;

#endif

    // max binary equivalent value
    const Word binMax= 4095;

    // convert value to 8 or 12 bit unsigned
    Word num;
    num = (Word)((10.0-value)/20.0)*binMax);

    // shift left 4bits and split into high low bytes
    Word lwByte, hiByte;
    num = num << 4;
    lwByte = num & 0xFF;    // bitwise 'and' with 8bit mask
    hiByte = (num >> 8) & 0xFF;

    if (port == 0)
    {
        outPort(DADATH0,hiByte);
        outPort(DADATL0,lwByte);
    }
    else // port == 1
    {
        outPort(DADATH1,hiByte);
        outPort(DADATL1,lwByte);
    }
}

//.....

void pc30::setDefaultClk()
{
    setPrescaler();
    setDivider();
    setUserclk();
}

//.....

void pc30::setCounter(Word clk, Word count)
{
    Word lwByte,hiByte;

    lwByte = count & 0xFF;
    hiByte = (count >> 8) & 0xFF;

    outPort(clk,lwByte);
    outPort(clk,hiByte);
}

//.....

```

```

Word pc30::getCount(Word clk)
{
    Word lwByte,hiByte;

    lwByte = inPort(clk);
    hiByte = inPort(clk);

    return lwByte + (hiByte << 8);
}

//.....

Word pc30::wait(Word nTicks, Word &i,Word start)
{
    Word stop,count;

    count = prescaler();
    i=0;
    if(start == NOW) start = count;

    if (nTicks < start)
        stop = start - nTicks;
    else
        stop = MAXcount - (nTicks - (start+1));

    if (stop > start)
        while (count < start) {count = prescaler(); i++;}

    while (count > stop) {count = prescaler();i++;}

    return count;
}

//.....

Word pc30::waitForUsrClk()
{
    static Word oldClk=0;

    static Word clk=0;

    while (oldClk == clk) clk = userclk();

    oldClk = clk;
#ifdef TRAP_ERRORS
    return divider();
#else
    return divider();
#endif
}

```

PCL833.cpp

```

#include "PCL833.h"
#include "defines.h"
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>

//-----
// pcl833 Constructor
//-----

pcl833::pcl833()
{
    LRESULT errCode;
    char errMsg[80];          // 32bit pointer to char[]

    // .....
    // Open device
    //.....

    if((errCode = DRV_DeviceOpen((ULONG)PCL833_Device_Num, (LONG far
*)&driverHandle)) != 0)
    {
        DRV_GetErrorMessage(errCode, (LPSTR)errMsg);
        cout << "ERROR : pcl833 constructor -DeviceOpen : \n";
        cout << errMsg << "\n";
    }

    // .....
    // Configure counters
    //.....

    config_0.counter      = 0;
    config_0.LatchSrc      = SWLATCH;                // Latch when read
    config_0.LatchOverflow = YES;                    // allow counters to
overflow
    config_0.ResetOnLatch  = NO;                      // don't reset on
after latch
    config_0.ResetValue    = RESET_HALF; // start counters at
7FFFFFFh

    if((errCode = DRV_QCounterConfig(driverHandle,
(LPT_QCounterConfig)&config_0)) != 0)
    {
        DRV_GetErrorMessage(errCode, (LPSTR)errMsg);
        cout << "ERROR : pcl833 constructor - config counter 0 :
\n";
        cout << errMsg << "\n";
    }

    config_1.counter      = 1;
    config_1.LatchSrc      = SWLATCH;                // Latch when read
    config_1.LatchOverflow = YES;                    // allow counters to
overflow
    config_1.ResetOnLatch  = NO;                      // don't reset on
after latch

```

```

    config_1.ResetValue    = RESET_HALF; // start counters at
    7FFFFFFh

    if((errCode = DRV_QCounterConfig(driverHandle,
                                     (LPT_QCounterConfig)&config_1)) != 0)
    {
        DRV_GetErrorMessage(errCode, (LPSTR)errMsg);
        cout << "ERROR : pcl833 constructor - config counter 1  :
\n";
        cout << errMsg << "\n";
    }

    //.....
    // Set start structures to default values
    //.....

    start_0.counter    = 0;
    start_0.InputMode  = ABPHASEX4;

    start_1.counter    = 1;
    start_1.InputMode  = ABPHASEX4;

}

//-----
//
// pcl833 Destructor
//-----

pcl833::~pcl833()
{
    stopCounters();
    DRV_DeviceClose(&driverHandle);
}

//-----
//
// Start counters 0&1
//-----

int pcl833::startCounters()
{
    LRESULT errCode;
    char    errMsg[80];      // 32bit pointer to char[]

    if((errCode = DRV_QCounterStart(driverHandle,
                                     (LPT_QCounterStart)&start_0)) != 0)
    {
        DRV_GetErrorMessage(errCode, (LPSTR)errMsg);
        cout << "ERROR : pcl833 start counter 0 ' : \n";
        cout << errMsg << "\n";

        return ERROR;
    }

    counting_0 = YES;

```

```

        if((errCode = DRV_QCounterStart(driverHandle,
            (LPT_QCounterStart)&start_1)) != 0)
        {
            DRV_GetErrorMessage(errCode, (LPSTR)errMsg);
            cout << "ERROR : pcl833 start counter 1 ' : \n";
            cout << errMsg << "\n";

            return ERROR;
        }

        counting_1 = YES;

        return OK;
    }

//-----
// Stop counters
//-----

int pcl833::stopCounters()
{
    LRESULT errCode;
    char    errMsg[80];        // 32bit pointer to char[]

    if (counting_0)
    if((errCode = DRV_CounterReset(driverHandle,0)) != 0)
    {
        DRV_GetErrorMessage(errCode, (LPSTR)errMsg);
        cout << "ERROR : pcl833 stop counter 0 : \n";
        cout << errMsg << "\n";
        return ERROR;
    }

    if (counting_0)
    if((errCode = DRV_CounterReset(driverHandle,1)) != 0)
    {
        DRV_GetErrorMessage(errCode, (LPSTR)errMsg);
        cout << "ERROR : pcl833 stop counter 1 : \n";
        cout << errMsg << "\n";

        return ERROR;
    }

    return OK;
}

//-----
// Reset counters
//-----

int    pcl833::resetCounters()
{
    if(stopCounters() == ERROR) return ERROR;
    if(startCounters() == ERROR) return ERROR;
}

```



```
        return OK;
    }

//-----
// Read counters
//-----

void pcl833::readCounters()
{
    count_0 = (_inp(CH_0_hiByte) << 16)+(_inp(CH_0_midByte) <<
8)+_inp(CH_0_lowByte);
    count_1 = (_inp(CH_1_hiByte) << 16)+(_inp(CH_1_midByte) <<
8)+_inp(CH_1_lowByte);
    count_0 = (_inp(CH_0_hiByte) << 16)+(_inp(CH_0_midByte) <<
8)+_inp(CH_0_lowByte);
    count_1 = (_inp(CH_1_hiByte) << 16)+(_inp(CH_1_midByte) <<
8)+_inp(CH_1_lowByte);
}
```

Runga Kutta 4th order.cpp

```

#include "my_math.h"

// NOTE: x equates in most cases to a time variable.

void euler (Vector& y, // state
            double x0, //
            initial x value
            double x1, //
            final x value
            double h, //
            step size
            Vector (*f)(Vector, double), // yDot=A.y +
            bu=f(y,x)
            void (serviceRoutine)(Vector, double))
{
    double x = x0;

    while (x + h <= x1)
    {
        y = y + f(y,x) * h;

        x = x + h;

        if (serviceRoutine) serviceRoutine(y,x);
    }
}

// NOTE: x equates in most cases to a time variable.
//rungaKutta(modelState, 0.0, Tstop, Ts, modelRKfct,
modelServiceRoutine);

void rungaKutta(Vector& y, //
state
               double x0,
               // initial x value
               double x1,
               // final x value
               double h,
               // step size
               Vector (*f)(Vector, double), //
yDot=A.y + bu=f(y,x)
               void (serviceRoutine)(Vector, double))
{
    Vector k0 = Vector(y.sizeOf());
    Vector k1 = Vector(y.sizeOf());
    Vector k2 = Vector(y.sizeOf());
    Vector k3 = Vector(y.sizeOf());

    double x = x0;

    while (x + h <= x1)
    {
        k0 = f(y,x) * h;

```

```
k1 = f(y+k0*0.5,x+h*0.5) * h;  
k2 = f(y+k1*0.5,x+h*0.5) * h;  
k3 = f(y+k2,x+h) * h;  
  
y = y + (k0+k1*2.0+k2*2.0+k3)/6.0;  
  
x = x + h;  
  
if (serviceRoutine) serviceRoutine(y,x);  
    }  
}
```

Vector.cpp

```

#include "vector.h"
#include <iostream.h>
#include <stdlib.h>
#include <math.h>

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vector::vector ()
{
    if(DEBUG_VECTOR) cout << "Constructing Vector \n";
}

//+++++++++++++++++++++++++++++++++++++
+++++

vector::vector (int size, double init)
{
    if(DEBUG_VECTOR) cout << "Constructing Vector \n";

    array = new double [size];

    if (array == NULL)
    { cout << "MEMORY ALLOCATION ERROR - vector constructor -
vector(";
        cout << size << ", " << init << ") \n\n";
        exit(-1);
    }

    length = size;
    for (int i = 0; i < length; i++)
        array[i] = init;
}

//+++++++++++++++++++++++++++++++++++++
+++++

vector::~~vector()
{
    if(DEBUG_VECTOR) cout << "Destructing Vector \n";

    if(array) delete [] array;
}

//+++++++++++++++++++++++++++++++++++++
+++++

vector::vector (vector &v)
{
    if(DEBUG_VECTOR) cout << "Vector::Copy Constructor Called \n";

    array = new double [v.size()];

    if (array == NULL)
    { cout << "MEMORY ALLOCATION ERROR - vector copy constructor";
        exit(-1);
    }
}

```

```

        length = v.size();
        for (int i = 0; i < length; i++)
            array[i] = v[i];
    }

//+++++
+++++

void vector::initialise (int size, double init)
{
    if(DEBUG_VECTOR) cout << "Vector::Initialise Called \n";

    array = new double [size];

    if (array == NULL)
    { cout << "MEMORY ALLOCATION ERROR - vector constructor -
vector(";
        cout << size << ", " << init << ") \n\n";
        exit(-1);
    }

    length = size;
    for (int i = 0; i < length; i++)
        array[i] = init;
}

//+++++
+++++

int vector::sameLength(vector &v, char *where)
{
    if (length != v.size())
    {
        cout << "ERROR - vector incompatible length\n";
        cout << "        - vector::" << where;
        return 0;
    }
    return 1;
}

//+++++
+++++

int vector::sameLength(double *a, char *where)
{
    if (length != a[0])
    {
        cout << "ERROR - array incompatible length\n";
        cout << "        - vector::" << where;
        return 0;
    }
    return 1;
}

//+++++
+++++

vector & vector::operator= (vector &v)
{
    if (!sameLength(v, "operator= (vector&v)")) exit(-1);

    for (int i = 0; i < length; i++) array[i] = v[i];
}

```

```

        return *this;
    }

//+++++
// copy double[] to vector: assumes size in a[0]

vector & vector::operator= (double *a)
{
    if (!sameLength(a,"operator= (double *a)")) exit(-1);

    for (int i = 0; i < length; i++) array[i] = a[i+1];
    return *this;
}

//+++++

vector & vector::operator= (double d)
{
    for (int i = 0; i < length; i++) array[i] = d;
    return *this;
}

//+++++

double & vector::operator[] (int i)
{
    if (i > length - 1)
    {
        cout << "ERROR - vector::operator[] - index [";
        cout << i << "] out of range\n\n";
        exit(-1);
    }

    return array[i];
}

//+++++

vector vector::operator+ (vector &v)
{
    if (!sameLength(v,"operator+ (vector&v)")) exit(-1);

    vector temp(length);

    for (int i = 0; i < length; i++) temp[i] = array[i] + v[i];

    return temp;
}

//+++++

vector vector::operator+ (double *a)
{
    if (!sameLength(a,"operator+ (double *a)")) exit(-1);

    vector temp(length);

```

```

        for (int i = 0; i < length; i++) temp[i] = array[i] + a[i+1];

        return temp;
    }

//+++++
+++++

vector vector::operator+ (double d)
{
    vector temp(length);

    for (int i = 0; i < length; i++) temp[i] = array[i] + d;

    return temp;
}

//+++++
+++++

vector vector::operator- (vector &v)
{
    if (!sameLength(v,"operator- (vector&v)")) exit(-1);

    vector temp(length);

    for (int i = 0; i < length; i++) temp[i] = array[i] - v[i];

    return temp;
}

//+++++
+++++

vector vector::operator- (double *a)
{
    if (!sameLength(a,"operator- (double *a)")) exit(-1);

    vector temp(length);

    for (int i = 0; i < length; i++) temp[i] = array[i] - a[i+1];

    return temp;
}

//+++++
+++++

vector vector::operator- (double d)
{
    vector temp(length);

    for (int i = 0; i < length; i++) temp[i] = array[i] - d;

    return temp;
}

//+++++
+++++

```

```

vector vector::operator* (double d)
{
    vector temp(length);

    for (int i = 0; i < length; i++) temp[i] = array[i] * d;

    return temp;
}

//+++++
+++++

vector vector::operator/ (double d)
{
    vector temp(length);

    for (int i = 0; i < length; i++) temp[i] = array[i] / d;

    return temp;
}

//+++++
+++++

vector & vector::operator+= (vector &v)
{
    if (!sameLength(v,"operator+= (vector&v)")) exit(-1);

    for (int i = 0; i < length; i++) array[i] = array[i]+v[i];
    return *this;
}

//+++++
+++++

vector & vector::operator+= (double *a)
{
    if (!sameLength(a,"operator+= (double *a)")) exit(-1);

    for (int i = 0; i < length; i++) array[i] = array[i]+a[i+1];
    return *this;
}

//+++++
+++++

vector & vector::operator+= (double d)
{
    for (int i = 0; i < length; i++) array[i] = array[i]+d;
    return *this;
}

//+++++
+++++

vector & vector::operator-= (vector &v)
{
    if (!sameLength(v,"operator-= (vector&v)")) exit(-1);

    for (int i = 0; i < length; i++) array[i] = array[i]-v[i];

```



```

        return *this;
    }

//+++++
+++++

vector & vector::operator-= (double *a)
{
    if (!sameLength(a,"operator-= (double *a)")) exit(-1);

    for (int i = 0; i < length; i++) array[i] = array[i]-a[i+1];
    return *this;
}

//+++++
+++++

vector & vector::operator-= (double d)
{
    for (int i = 0; i < length; i++) array[i] = array[i]-d;
    return *this;
}

//+++++
+++++

vector & vector::operator*= (double d)
{
    for (int i = 0; i < length; i++) array[i] = array[i]*d;
    return *this;
}

//+++++
+++++

vector & vector::operator/= (double d)
{
    for (int i = 0; i < length; i++) array[i] = array[i]/d;
    return *this;
}

//+++++
+++++

vector vector::operator+ ()
{
    return *this;
}

//+++++
+++++

vector vector::operator- ()
{
    vector temp(length);
    for (int i = 0; i < length; i++) temp[i] = -array[i];
    return temp;
}

```

```

//+++++
+++++

double vector::magnitude()
{
    double sumSquares = 0.0;

    for (int i = 0; i < length; i++)
        sumSquares += array[i]*array[i];

    return sqrt(sumSquares);
}

//+++++
+++++

double vector::distance (vector &v)
{
    double sumSquaresOfDiff = 0.0;

    for (int i = 0; i < length; i++)
        sumSquaresOfDiff += (array[i]-v[i])*(array[i]-v[i]);

    return sqrt(sumSquaresOfDiff);
}

//+++++
+++++

ostream & operator<<(ostream & stream, vector &v)
{
    if (!v.valid())
    {
        cout << "ERROR - operator<<(vector) - invalid
vector\n\n";
        exit(-1);
    }

    stream << "{ ";
    for (int i = 0; i < v.size() -1; i++) cout << v[i] << ", ";
    stream << v[v.size()-1]<< " }";

    return stream;
}

//+++++
+++++

```