UNIVERSITY OF ÇUKUROVA
INSTITUTE OF NATURAL AND APPLIED SCIENCE

MSc THESIS

Mehmet Uğraş CUMA

DESIGN AND APPLICATIONS OF FUZZY AND NEURAL CONTROLLERS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA, 2006

ÇUKUROVA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

DESIGN AND APPLICATIONS OF FUZZY AND NEURAL CONTROLLERS

Mehmet Uğraş CUMA

YÜKSEK LİSANS TEZİ

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez / / Tarihinde Aşağıdaki Jüri Üyeleri Tarafından Oybirliği İle Kabul Edilmiştir.

İmza.....................	İmza.....................	İmza.....................
Prof. Dr. Mehmet TÜMAY	Doç. Dr. Tankut YALÇINÖZ	Yrd. Doç. Dr. İlyas EKER
DANIŞMAN	ÜYE	ÜYE

Bu tez Enstitümüz Elektrik Elektronik Mühendisliği Anabilim Dalında hazırlanmıştır.

Kod No:

Prof. Dr. Aziz ERTUNÇ
Enstitü Müdürü

Bu çalışma Ç.Ü. Bilimsel Araştırma Projeleri Birimi tarafından desteklenmiştir.
Proje No: MMF2006YL19

Not: Bu tezde kullanılan özgün ve başka kaynaktan yapılan bildirişlerin, çizelge, şekil ve fotoğrafların kaynak gösterilmeden kullanımı, 5846 sayılı Fikir ve Sanat Eserleri Kanunundaki hükümlere tabidir.

ÖZ

YÜKSEK LİSANS TEZİ

BULANIK VE SİNİRSEL KONTROLCÜLERİN TASARIMI VE UYGULAMALARI

Mehmet Uğraş CUMA

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI
FEN BİLİMLERİ ENSTİTÜSÜ
ÇUKUROVA ÜNİVERSİTESİ

Danışman: **Prof. Dr. Mehmet TÜMAY**
Yıl: **Haziran 2006, Sayfa: 106**
Jüri: **Prof. Dr. Mehmet TÜMAY**
Doç. Dr. Tankut YALÇINÖZ
Yrd. Doç. Dr. İlyas EKER

Kontrolcüler ve kontrol algoritmaları, sistemlerin sorunsuz olarak çalışmasını sağlayan önemli unsurlardan biridir. Farklı kontrol stratejilerinin uygulanması, sistemler üzerinde farklı etkiler yaratabilir. PID kontrolcüler yaygın olarak kullanılırken, Bulanık ve Sinirsel kontrol ilgi çekmeye devam etmektedir. Tezde, Bulanık ve Sinirsel kontrolcülerin tasarımı gerçekleştirilmiş, sistemin kontrolcülere verdiği tepkileri incelenmiştir. Kontrolcü tasarımlarının uygulanabilirliğini göstermek ve tasarımlar arasında karşılaştırma yapabilmek için örnek olay incelemeleri yapılmıştır.

Gerçek zamanlı uygulamalarda kullanılmak üzere bulanık mantık kontrolcü için C++ kodu hazırlanmıştır. Bu kodun uygulamalarda nasıl kullanılacağı ve kontrolcü tasarımları için nasıl adapte edilebileceği açıklanmıştır.

Anahtar Kelimeler: Bulanık Mantık Kontrolcü, Sinirsel Kontrolcü

ABSTRACT

MSc THESIS

DESIGN AND APPLICATIONS OF FUZZY AND NEURAL CONTROLLERS

Mehmet Uğraş CUMA

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

INSTITUTE OF NATURAL AND APPLIED SCIENCES

UNIVERSITY OF ÇUKUROVA

Supervisor:	**Prof. Dr. Mehmet TÜMAY**
Year:	**June 2006, Pages: 106**
Jury:	**Prof. Dr. Mehmet TÜMAY**
	Assoc. Prof. Dr. Tankut YALÇINÖZ
	Asst. Prof. Dr. İlyas EKER

Controllers and control algorithms are important elements of systems, providing trouble-free operations. Implementing different control strategies may have different effects on systems. While PID controllers are widely used, Fuzzy and Neural controllers are still drawing attention. In the thesis, Fuzzy and Neural controllers are realized and the reactions of the system to the controllers are investigated. Several case studies are conducted to show the viability of controller designs and comparison of them.

A C++ implementation of a fuzzy controller is prepared to use in real time applications. The use of this code in applications and the adaptation for different controllers are explained.

Keywords: Fuzzy Logic Controller, Neural Controller

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof.Dr. Mehmet TÜMAY for his endless support, encouragements and great supervision. It has been a great honor to have Mr. Mehmet TÜMAY as supervisor.

I would like to thank K. Çağatay BAYINDIR his great support. His advices let me to finish this thesis.

I would like to thank Prof. Dr. Süleyman Güngör, Head of the Department, for his valuable suggestions and providing a good working environment.

I thank Assoc. Prof. Dr. Tankut Yalçınöz, Asst. Prof. Dr. İlyas Eker for serving as a member my committee.

I am very grateful to Mehmet Salih ÖZGÜR, Ahmet TEKE, Fırat KUMRU, Mutlu TOKSÖZ for their friendship and support.

I wish to thank Institute of Natural and Applied Science for financial supports during the preparation of this thesis. I wish also to extend my acknowledgements to all staff members in the department and my friends.

Finally, I would like to thank my fiancée Gaye ARIBOĞAN and my family for their endless support and encouragements.

CONTENT **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

e_{ss}	Steady state error
K_v	Velocity error constant
K_a	Acceleration error constant
T_d	Delay time
T_r	Rise time
T_s	Settling time
Δ	Characteristic equation
ω_n	Natural frequency
α	Damping factor
ξ	Damping ratio
c_{ss}	Steady state output
K_p	Proportional gain
K_i	Integral gain
K_d	Derivative gain
τ_0	Apparent dead time
τ_{ar}	Average residence time
τ_{at}	Apparent time constant
ϕ	Air gap flux
T_m	Motor torque
T_L	Load torque
T_d	Disturbance torque
I_a	Armature current
I_f	Field current
V_a	Armature voltage

V_f	Field voltage
R_f	Field resistance
L_f	Field inductance
R_a	Armature resistance
L_a	Armature inductance
J	Moment of inertia
K_m	Motor constant
A_α	Strong α cut
$A_{\overline{\alpha}}$	Weak α cut
V_{ave}	Average output voltage of inverter
m	Modulation index
n	Speed of the motor
y^q	Target output for neural network
x^q	Input to the neural network
E	Error constant
w_{ij}	Network weights
η	Learning rate

LIST OF ABBREVIATIONS

DC	Direct Current
F	Fahrenheit
FLC	Fuzzy Logic Controller
FLS	Fuzzy Logic System
HP	Horse Power
IEEE	Institute of Electrical and Electronics Engineers
IGBT	Insulated Gate Bipolar Transistor
MIMO	Multi Input – Multi Output
MISO	Multi Input – Single Output
MOM	Mean Of Maximum
SISO	Single Input – Single Output
P	Proportional
PI	Proportional – Integral
PID	Proportional – Integral – Derivative
PWM	Pulse Width Modulation

1. INTRODUCTION

Controllers and control algorithms are important elements of systems, providing trouble-free operations. Implementing different control strategies may have different affects on systems. While PID controllers are widely used, Fuzzy and Neural controllers are still drawing attention. The purpose of the thesis is to realize the Fuzzy and Neural controllers and investigate the reactions of the system to the controllers. Several case studies will be conducted to show the viability of controller designs and comparison of them.

Fuzzy logic controllers serve the same function as the more conventional controllers, but they manage complex control problems through heuristics and mathematical models provided by fuzzy logic, rather than via mathematical models provided by differential equations. Emulating human expertise in control gives us a new way to design controllers for complex plants. In recent years, neural solutions have been suggested for many industrial systems. Neural control refers to another control method when available data are in the form of measurements in the form of the plant's behavior (Nguyen, *et al*, 2002).

Fuzzy logic, which is the logic on which fuzzy control is based, is much closer in spirit to human thinking and natural language than the traditional logical systems. Basically, it provides an effective means of capturing the approximate, inexact nature of real world. Viewed in this perspective, the essential part of the fuzzy logic controller (FLC) is a set of linguistic control rules related by the dual concepts of fuzzy implication and the compositional rule of inference. In essence, then, the FLC provides an algorithm which can convert the linguistic control strategy based on expert knowledge into an automatic control strategy. Experience shows that the FLC yields results to those obtained by conventional control algorithms. In particular, the methodology of the FLC appears very useful when the processes are to complex for analysis by conventional quantitative techniques or when the available sources of information are interpreted qualitatively, inexactly, or uncertainly (Lee, 1990).

A fuzzy logic system is unique in that it is able to simultaneously handle numerical data and linguistic knowledge. It is a nonlinear mapping of an input data vector into a scalar output. Fuzzy set theory and fuzzy logic establish the specifics of the nonlinear mapping (Mendel, 1995).

Neural control refers to another control method when available data are in the form of measurements of plant's behavior. This is the case where information is only in the form of system behavior, either of the real plant or of its simulated model, expressed as input-output measurements. To build a neural network based controller that can force a plant to behave in some desirable way, it is needed to adjust its parameters from the observed errors that are the difference between the plant's outputs and the desired outputs. In direct control, a neural network directly implements the controller, that is, the controller is a neural network. The network must be trained as the controller according to some criteria, using either numerical input-output data or a mathematical model of the system (Nguyen, *et al*, 2002).

There are several applications for fuzzy and neural control. Salvatore, *et al,* 2000 deals with the design of fuzzy logic based controllers for dc and ac electric drives. The authors propose a minimum number of rules and the criteria, based on physical considerations, to determine the input and output gains instead of using trial and error method. The application of Takagi-Sugeno type fuzzy logic controller to a three phase shunt active power filter for the power quality improvement and reactive power compensation required by a nonlinear load (Bhende, *et al*, 2006). A fuzzy logic controller for static var compensator was put (Fang Li, *et al*, 2005). An effective, efficient and distributed fuzzy control algorithm is proposed to perform control over the contention parameters in the IEEE 802.11 media access control layer for supporting Quality of Service in wireless communications in (Chen, *et al*, 2005). Fuzzy control strategy for coke oven heating temperature control is introduced in (Guo, *et al*, 2004). Dzung, *et al*, 2005 introduces the ability of artificial neural networks in estimating speed and controlling the separately excited DC motor. An active control system is developed to prevent rollover in heavy vehicles using recurrent neural network is proposed in (Sanchez, *et al*, 2004). A novel robust speed control method for induction motor drives based on a two-layered neural network plant estimator and a two-layered neural network PI controller (Chen, *et al*, 2002).

An introduction is made with Chapter 1. Some applications of fuzzy and neural control are reported in this section.

The thesis is starting with the fundamental concepts of conventional control systems in Chapter 2. Open loop and closed loop control systems are introduced. The fundamental definitions, test signals are explained. The design process for a PID controller is realized. The results of the controller are given.

Chapter 3 is separated for fuzzy logic control systems. The fundamental concepts of fuzzy set theory are introduced first. Using these concepts, it is explained how to construct a fuzzy controller. The design process of a fuzzy controller is introduced.

Chapter 4 deals with neural control. The question why neural networks in control is tried to be answered. Use of MATLAB in neural control is explained. Then a neural controller is designed for a DC motor. The simulation results are given.

In Chapter 5, another application of fuzzy and neural control is introduced. First, a single-input single-output fuzzy controller is demonstrated, and to improve the performance, a second input is added to the design. A simple neural network controller is established to show that a neural network can easily placed instead of another controller.

2. LINEAR CONTROL SYSTEM FUNDAMENTALS

In recent years, automatic control systems have assumed an increasingly important role in the development and advancement of modern civilization and technology. Domestically, automatic controls in heating and air conditioning systems regulate the temperature and humidity of modern homes for comfortable living. Industrially, automatic control systems are found in numerous applications, such as quality control of manufactured products, automation, machine tool control, modern space technology and weapon systems, computer systems, transportation systems and robotics. The basic control system concept may be described by the simple block diagram shown in Figure 2.1. In more common terms, the controlled variable is the output of the system, and the actuating signal is the input (Kuo, 1975).

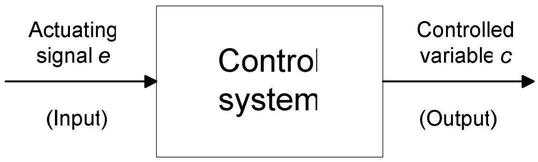


Figure 2.1 Basic control system

Open Loop Control Systems:

The word automatic implies that there is a certain amount of sophistication in the control. By automatic, it generally means that the system is usually capable of adapting to a variety of operating conditions and is able to respond to a class of inputs satisfactorily. However, not any type of control system has the automatic feature. Usually, the automatic feature is achieved by feeding the output variable back and comparing it with the command signal.

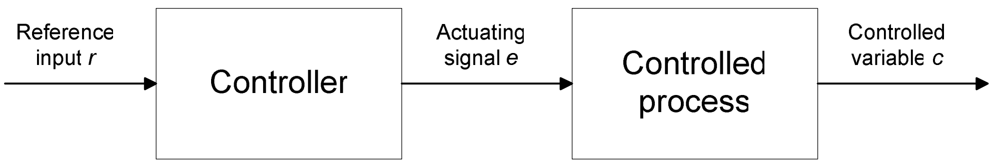


Figure 2.2 Block diagram of an open-loop system

When a system does not have the feedback structure, it is called an open-loop system, which is simplest and most economical type of control system. An input signal or command *r* is applied to the controller, whose output acts as the actuating signal *e*; the actuating signal actuates the controlled process (Kuo, 1975).

Closed Loop Control Systems:

In order to obtain more accurate control, the controlled signal *c(t)* must be fed back and compared with the reference input, and an actuating signal proportional to the difference of the output and the input must be sent through the system to correct the error. A system with one or more feedback paths is called a closed-loop system (Kuo, 1975).

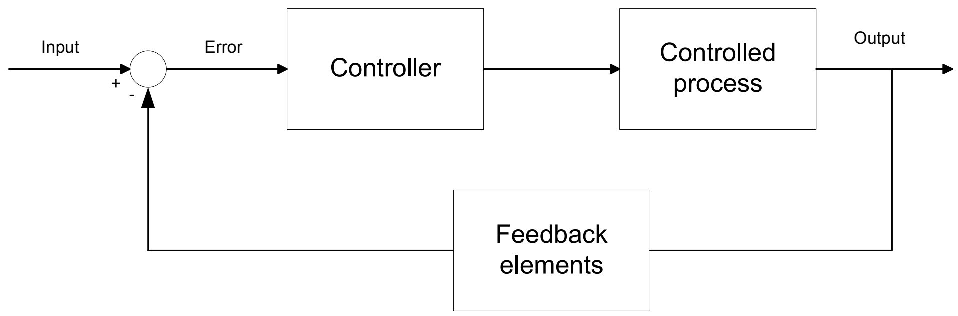


Figure 2.3 Basic elements of a feedback control system

2.1 Typical Test Signals for Time Response of Control Systems

2.1.1 Unit-step Function

The step input function represents an instantaneous change in the reference input variable. The mathematical representation of a step function is,

$$r(t) = \begin{cases} R & t > 0 \\ 0 & t < 0 \end{cases} \tag{2.1}$$

where R is a constant.

$$r(t) = Ru_s(t) \tag{2.2}$$

The step function is very useful as a test signal since its initial instantaneous jump in amplitude reveals a great deal about a system's quickness to respond.

2.1.2 Ramp Input Function

In the case of the ramp function, the signal is considered to have a constant change in value with respect to time. Mathematically, a ramp function is defined by,

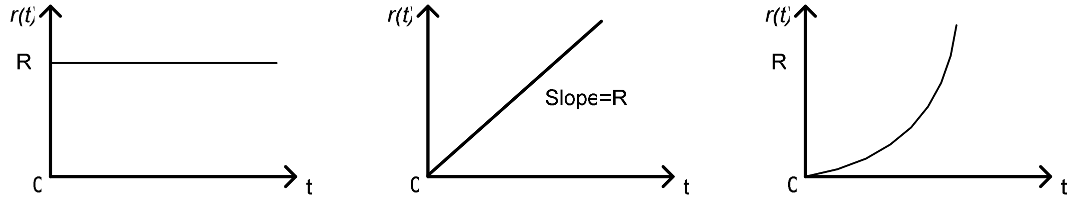
$$r(t) = \begin{cases} Rt & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{2.3}$$

The ramp function has the ability to test how the system would respond to a signal that changes linearly with time.

2.1.3 Parabolic Input Function

The mathematical representation of a parabolic input function is

$$r(t) = \begin{cases} Rt^2 & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{2.4}$$



a) Unit step function b) Ramp function c) Parabolic function

Figure 2.4 Basic time domain test signals

A parabolic function is one degree faster than a ramp function that is also used in system changing with time.

2.2 Steady-state Error

The steady-state error is a measure of system accuracy when a specific type of input is applied to control system. In a physical system, because of friction and the nature of the particular system, the steady state of the output response seldom agrees exactly with the reference input. Therefore, steady-state errors in control systems are almost unavoidable, and in a design problem one of the objectives is to keep the error to a minimum, or below a certain tolerable value. A nonunity feedback system given in Figure 2.5 is considered for steady-state error (Kuo, 1975).

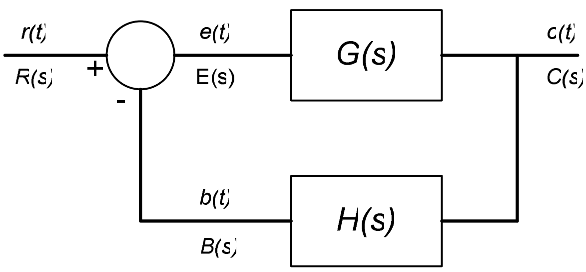


Figure 2.5 Nonunity feedback control system,

2.2.1 Steady-state Error Due to a Step Input

If the reference input to the control system of Figure 2.5 is a step input of magnitude *R*, the Laplace transform of the input is *R/s*.

$$e_{ss} = \lim_{s \to 0} \frac{sR(s)}{1+G(s)H(s)} = \lim_{s \to 0} \frac{R}{1+G(s)H(s)} = \frac{R}{1+\lim\limits_{s \to 0} G(s)H(s)} \tag{2.5}$$

Let,

$$K_p = \lim_{s \to 0} G(s)H(s) \tag{2.6}$$

where K_p is the positional error constant.

$$e_{ss} = \frac{R}{1+K_p} \tag{2.7}$$

For e_{ss} to be zero, when the input is a step function, K_p must be infinite.

2.2.2 Steady-state Error Due to a Ramp Input

If the input to the control system in Figure 2.5 is,

$$r(t) = Rtu_s(t) \tag{2.8}$$

where *R* is a constant, the Laplace transform of *r(t)* is,

$$R(s) = \frac{1}{s^2} \tag{2.9}$$

$$e_{ss} = \lim_{s\to 0} \frac{R}{s + sG(s)H(s)} = \frac{R}{\lim_{s\to 0} sG(s)H(s)} \tag{2.10}$$

Let,

$$K_v = \lim_{s\to 0} sG(s)H(s) \tag{2.11}$$

where *Kv* is the velocity error constant.

$$e_{ss} = \frac{R}{K_v} \tag{2.12}$$

Eq.(2.12) shows that for e_{ss} to be zero when the input is a ramp function, K_v must be infinite.

2.2.3 Steady-state Error Due to a Parabolic Input

If the input is described by,

$$r(t) = \frac{Rt^2}{2} u_s(t) \tag{2.13}$$

The Laplace transform of *r(t)* is,

$$R(s) = \frac{R}{s^3} \tag{2.14}$$

$$e_{ss} = \frac{R}{\lim\limits_{s \to 0} s^2 G(s) H(s)} \tag{2.15}$$

Let,

$$K_a = \lim_{s \to 0} s^2 G(s) H(s) \tag{2.16}$$

where K_a is the acceleration error constant.

$$e_{ss} = \frac{R}{K_a} \tag{2.17}$$

2.3 Unit-step Response and Time Domain Criterias

The transient performance of a control system is usually characterized by the use of an unit step input. Typical performance criteria that are used to characterize the transient response to a unit step include overshoot, delay time, rise time and settling time. Figure 2.6 illustrates a typical unit step response of a linear control system (Kuo, 1975).

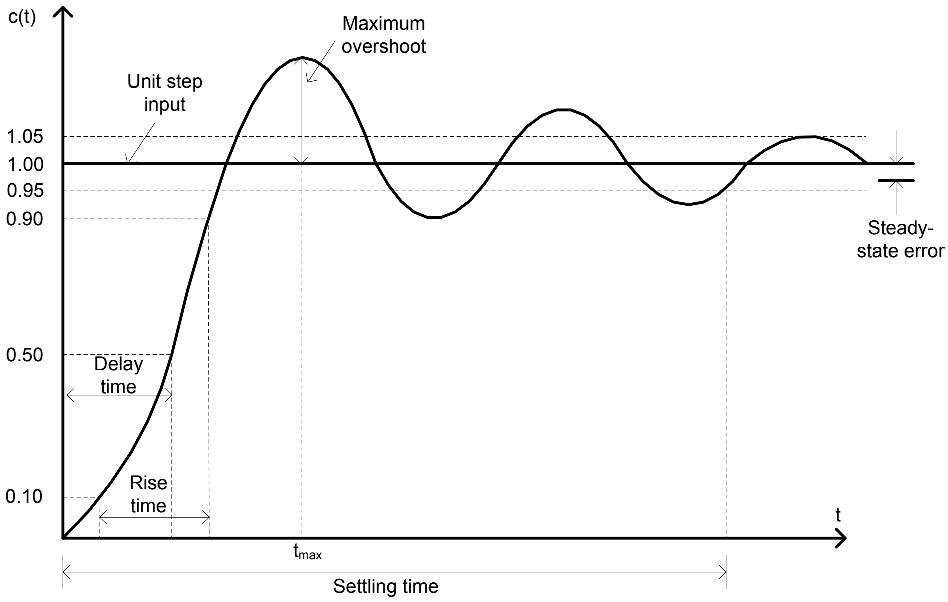


Figure 2.6 Typical unit step response of a control system

2.3.1 Maximum Overshoot

The maximum overshoot is defined as the largest deviation of the output over the step input during the transient state. The amount of maximum overshoot is also used as a measure of the relative stability of the system. The maximum overshoot is often represented as a percentage of the final value of the step response; that is,

*Percent maximum overshoot = (maximum overshoot / final value)*100%* (2.18)

2.3.2 Delay Time

The delay time T_d is defined as the time required for the step response to reach 50 percent of its final value.

2.3.3 Rise Time

The rise time T_r is defined as the time required for the step response to rise from 10 percent to 90 percent of its final value. Sometimes an alternative measure is to represent the rise time as a reciprocal of the slope of the step response at the instant that the response is equal to 50 percent of its final value.

2.3.4 Settling Time

The settling time T_s is defined as the time required for the step response to decrease and stay within a specified percentage of its final value.

2.4 Damping Ratio, Damping Factor and Natural Frequency

Consider the following transfer function of the second order system.

$$\frac{C(s)}{R(s)} = \frac{\omega_n^{\ 2}}{s^2 + 2\xi\omega_n s + \omega_n^{\ 2}} \tag{2.19}$$

The characteristic equation of the system is;

$$\Delta = s^2 + 2\xi\omega_n s + \omega_n^{\ 2} = 0 \tag{2.20}$$

For a unit step function input, $R(s) = 1/s$, the output response of the system is determined by taking the inverse Laplace transform of

$$C(s) = \frac{\omega_n^{\ 2}}{s(s^2 + 2\xi\omega_n s + \omega_n^{\ 2})} \tag{2.21}$$

c(t) is determined with zero initial states.

$$c(t) = 1 + \frac{e^{-\xi\omega_n t}}{\sqrt{1-\xi^2}} \sin\left[\omega_n\sqrt{1-\xi^2}\,t - \arctan\frac{\sqrt{1-\xi^2}}{-\xi}\right],\ t \geq 0 \tag{2.22}$$

It is interesting to study the relationship between the roots of the characteristics equation. The two roots of the characteristic equation are

$$s_1, s_2 = -\xi\omega_n \pm j\omega_n\sqrt{1-\xi^2} = -\alpha \pm j\omega \quad (2.23)$$

As seen from Eq.(2.21), $\alpha = \xi\omega_n$, and α appears as the constant that is multiplied to *t* in the exponential term of Eq.(2.22). Therefore, α controls the rate of rise and decay of the time response. In other words, α controls the damping of the system and is called damping factor.

When the two roots of the characteristic equation are real and identical, we call the system critically damped. Critical damping occurs when $\xi = 1$. Under this condition the damping factor is simply $\alpha = \omega_n$. Therefore, we can regard ξ as the damping ratio, which is the ratio between the actual damping factor and the damping factor when the damping is critical.

ω_n is defined as the natural frequency. When $\xi = 0$, the roots of characteristic equation are imaginary, and the step response is purely sinusoidal. Therefore, ω_n corresponds to the frequency of the undamped sinusoid (Kuo, 1975).

2.5 Control System Design

Starting from the control process given in Figure 2.7, the control system design is formed by following three steps:

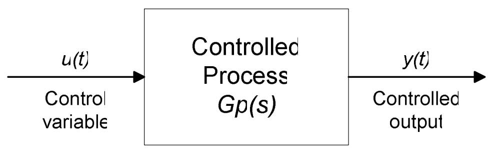


Figure 2.7 Controlled process

- Determination of what system will be doing and how.
- Determination of how the controller will be added to system.
- Determination of controller parameters to achieve design targets (Kuo, 1975).

Multimode controller design is a system compensation process in which the type of the system id changed so as to meet desired performance requirements (Driels, 1996).

2.5.1 Proportional Control

A proportional controller is defined to be one in which the output is simply proportional to the input, where the constant of proportionality is the gain of the controller. Consider initially the first-order system with a disturbance input, shown in Figure 2.8.

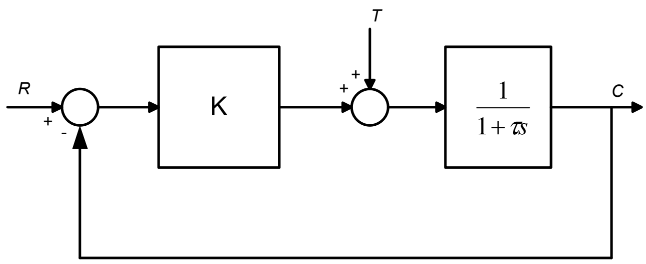


Figure 2.8 First order system with disturbance

The transfer function relating output C to the two inputs R and T are

$$\frac{C}{R} = \frac{K/(1+K)}{1+\tau s/(1+K)} \tag{2.24}$$

$$\frac{C}{T} = \frac{1/(1+K)}{1+\tau s/(1+K)} \tag{2.25}$$

If the input R is a unit step, the steady-state output becomes

$$c_{ss}(t) = \frac{K}{1+K} \tag{2.26}$$

If the disturbance T is a unit step, the corresponding steady-state output becomes

$$c_{ss}(t) = \frac{1}{1+K} \tag{2.27}$$

If K is made large, the output for unit *R* approaches unity, while the output for *T* approaches zero, which is a good result, but note that the error cannot be made equal to zero, since there will be practical limits on the magnitude of *K*. Consider a second order system, also subjected to external disturbance, as shown in Figure 2.9.

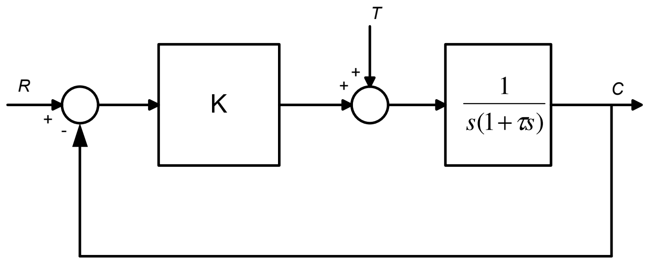


Figure 2.9 Second order system with disturbance

The transfer functions from two inputs *R* and *T* to the output *C* as

$$\frac{C}{R} = \frac{K}{\tau s^2 + s + K} \tag{2.28}$$

$$\frac{C}{T} = \frac{1}{\tau s^2 + s + K} \tag{2.29}$$

The steady-state output for a unit step input *R* is

$$c_{ss}(t) = 1 \tag{2.30}$$

and for a unit step disturbance *T* it is

$$c_{ss}(t) = \frac{1}{K} \tag{2.31}$$

The steady-state error for the regular input is zero, as expected, but the disturbance does produce a finite error. Although this may be made small by letting

K be large. Simple proportional control is of limited success is trying to obtain good performance in terms of steady-state error, disturbance rejection and transient response (Driels, 1996).

2.5.2 Proportional – Integral Control

A proportional – integral controller, where the output is proportional to the input plus the integral of the input. This type of control is called two-mode control. The transfer function is seen to be

$$\theta_o = \left[K_p + \frac{K_i}{s} \right] \theta_i \tag{2.32}$$

Consider this controller replaces proportional controller in the first order system. The new system looks like that shown in Figure 2.10. The new transfer function relating the output to the two inputs become,

$$\frac{C}{R} = \frac{K_p s + K_i}{\tau s^2 + s(1 + K_p) + K_i} \tag{2.33}$$

$$\frac{C}{T} = \frac{s}{\tau s^2 + s(1 + K_p) + K_i} \tag{2.34}$$

When the input *R* is a unit step, the steady-state output *C* becomes

$$c_{ss}(t) = 1 \tag{2.35}$$

and the steady-state output for unit *T* is

$$c_{ss}(t) = 0 \tag{2.36}$$

Proportional plus integral control has reduced the steady-state error to zero and completely rejected the external disturbance.

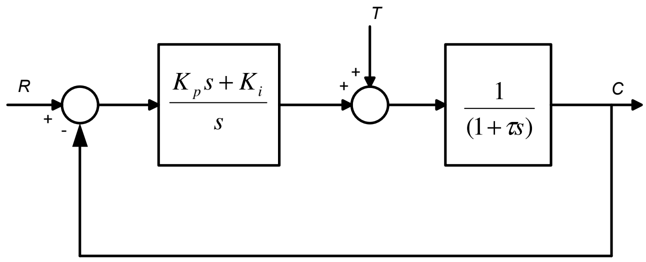


Figure 2.10 PI control of first-order system

If the transfer function is second-order system as shown in Figure 2.11. The transfer function become

$$\frac{C}{R} = \frac{K_p s + K_i}{\tau s^3 + s^2 + K_p s + K_i} \tag{2.37}$$

$$\frac{C}{T} = \frac{s}{\tau s^3 + s^2 + K_p s + K_i} \tag{2.38}$$

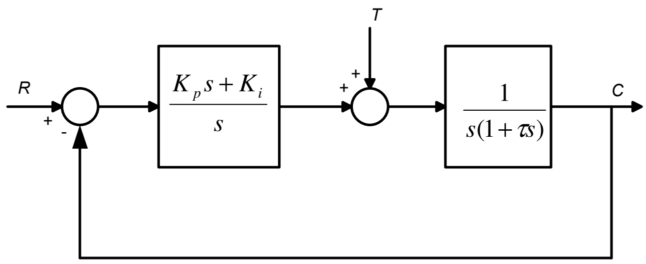


Figure 2.11 PI control of second order system

Letting R be a unit step produces

$$c_{ss}(t) = 1 \tag{2.39}$$

and for the unit disturbance

$$c_{ss}(t) = 0 \tag{2.40}$$

Using the open-loop transfer function

$$GH(s) = \frac{K_p s + K_i}{s^2(1+\tau s)} \tag{2.41}$$

From the transfer function, its seen that the system has double pole at the origin, another pole at $s = 1/\tau$, and a zero at $s = -K_i / K_p$. In order for the system to be stable, the zero must be closer to the imaginary axis than the system pole.

$$\left|\frac{1}{\tau}\right| > \left|\frac{K_i}{K_p}\right| \tag{2.42}$$

This places some restrictions on the selection of the proportional and integral gains (Driels, 1996).

2.5.3 Proportional - Integral - Derivative Control

The transfer function of a PID (Proportional-Integral-Derivative) is may be written as

$$\frac{\theta_o}{\theta_i} = K_p + \frac{K_i}{s} + K_d s \tag{2.43}$$

PID controllers are very effective in adjusting the performance of control systems, and they are available as commercial components intended to be added to the forward path of a control system in order to improve its performance. Selection of values for $K_p,\ K_i$ *and* K_d is known as tuning the controller, and it may be accomplished in many ways. The transfer function can be rewritten as (Driels, 1996)

$$\frac{\theta_o}{\theta_i} = K_p(1 + \frac{1}{T_i s} + T_d s) \tag{2.44}$$

2.5.3.1 Trial and Error Method

In this approach, the value of K_p is selected first in order to achieve the transient response desired while setting T_i and T_d to zero. Then T_i is adjusted in order to satisfy any steady-state error requirements. This will probably degrade the transient response due to the inclusion of a closed loop zero. The transient response may be restored by selecting a suitable T_d. Several iterations of this process may be required before the system behaves as desired.

For certain modes of control, Ziegler and Nichols have developed empirical settings for the three parameters in a PID controller. These settings were derived from the observation of many real systems in which PID controllers were adjusted in order to obtain optimum performance (Driels, 1996).

2.5.3.2 Ziegler and Nichols Step Response Method

This method only applies if the open-loop step response of the plant has no overshoot. If this is the case, the unit step response is obtained and takes the form shown in Figure 2.12.

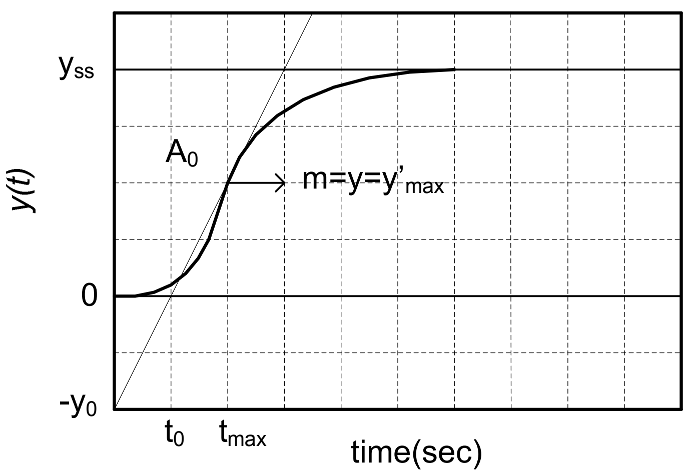


Figure 2.12 Typical step response

The equation of the line in the Figure 2.12 is given by

$$y = (y'_{\max})t + y(t_{\max}) - (y'_{\max})(t_{\max}) \tag{2.45}$$

From the Eq.(2.45),

$$y_0 = (y'_{\max})(t_{\max}) - y(t_{\max}) \tag{2.46}$$

$$\tau_0 = \frac{y_0}{y'_{\max}} \tag{2.47}$$

τ_0 is called apparent dead time. Another time constant τ_{ar} called average residence time is defined as

$$\tau_{ar} = \frac{A_0}{y_{ss}} \tag{2.48}$$

where A_0 is the area between y_{ss} and $y(t)$ and computed by the following integral.

$$A_0 = \int_0^{\infty} \left| y_{ss} - y(t) \right| dt \tag{2.49}$$

Last time constant that will be used is apparent time constant, τ_{at} defined as

$$\tau_{at} = \tau_{ar} - \tau_0 \tag{2.50}$$

Table 2.1 Ziegler-Nichols gain selection table

Controller	K_p	K_i	K_d
PI	$\frac{9}{10y_0}$	$\frac{3}{10y_0\tau_0}$	0
PID	$\frac{6}{5y_0}$	$\frac{3}{5y_0\tau_0}$	$\frac{3\tau_0}{5y_0}$

By means of values determined from the step response test, y_0 and τ_0, controller gains are can be selected from Ziegler-Nichols gain selection table given in Table 2.1

2.5.3.3 Chein-Hrones-Reswick Method

Chein-Hrones-Reswick have made some modifications on Ziegler-Nichols method. They used y_0, τ_0 and τ_{at} values determined from the step response test. Gain selection table for the Chein-Hrones-Reswick method is given in Table 2.2.

Table 2.2 Chein-Hrones-Reswick gain selection table

Controller	K_p	K_i	K_d
PI	$\frac{7}{20y_0}$	$\frac{7}{24y_0\tau_{at}}$	0
PID	$\frac{3}{5y_0}$	$\frac{3}{5y_0\tau_{at}}$	$\frac{3\tau_0}{10y_0}$

2.6 Case Study: Mathematical Model of a DC Motor and its Control

The DC motor is a power actuator device that delivers energy to load, as shown in Figure 2.14. The DC motor converts direct current (DC) electrical energy into rotational mechanical energy. A major fraction of the torque generated in the motor of the motor is available to drive an external load.

The transfer function of the DC motor will be developed for a linear approximation to an actual motor, and second order effects, such as hysteresis and the voltage drop across the brushes, will be neglected. The input voltage may be applied to the field and armature terminals.

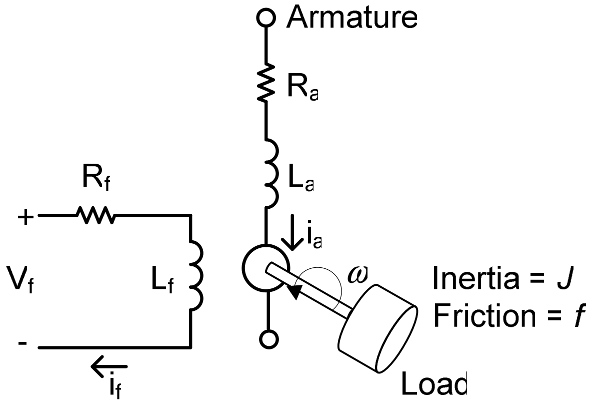


Figure 2.13 DC motor wiring diagram

The air-gap flux of the motor is proportional to the field current, provided the field is unsaturated, so that

$$\phi = K_f i_f(t) \tag{2.51}$$

The torque developed by the motor is assumed to be related linearly to ϕ and the armature current as follows:

$$T_m = K_1 \phi . i_a(t) = K_1 K_f i_f(t) i_a(t) \tag{2.52}$$

It is clear from Eq.(2.52) that, to have a linear element, one current must be maintained constant while the other current becomes the input current. First, we shall consider the field current controlled motor, which provides a substantial power amplification.

$$T_m(s) = (K_1 K_f I_a) I_f(s) = K_m I_f(s) \tag{2.53}$$

where $i_a = I_a$ is a constant armature current and K_m is defined as the motor constant. The field current is related to the field voltage as

$$V_f(s) = (R_f + sL_f) I_f(s) \tag{2.54}$$

The motor torque $T_m(s)$ is equal to the toque delivered to the load. This relation may be expressed as

$$T_m(s) = T_L(s) + T_d(s) \tag{2.55}$$

where $T_L(s)$ is the load torque and $T_d(s)$ is the disturbance torque. The load torque for rotating inertia as shown in Figure 2.13 is written as

$$T_L(s) = Js^2\theta(s) + fs\theta(s) \tag{2.56}$$

Rearranging the equations,

$$T_L(s) = T_m(s) - T_d(s) \tag{2.57}$$

$$T_m(s) = K_m I_f(s) \tag{2.58}$$

$$I_f(s) = \frac{V_f(s)}{R_f + sL_f} \tag{2.59}$$

Therefore the transfer function of the motor-load combination with $T_d(s) = 0$,

$$\frac{\theta(s)}{V_f(s)} = \frac{K_m}{s(Js+f)(sL_f + R_f)} = \frac{K_m / JL_f}{s(s+f/J)(s+R_f/L_f)} \tag{2.60}$$

Alternatively, the transfer function may be written in terms of the time constants of the motor as

$$\frac{\theta(s)}{V_f(s)} = G(s) = \frac{K_m / fR_f}{s(\tau_f s+1)(\tau_L s+1)} \tag{2.61}$$

where $\tau_f = L_f / R_f$ and $\tau_L = J / f$.

The armature controlled DC motor utilizes a constant field current, and therefore motor torque is

$$T_m(s) = (K_1 K_f I_f) I_a(s) = K_m I_a(s) \tag{2.62}$$

The armature current is related to the input voltage applied to the armature as

$$V_a(s) = (R_a + sL_a)I_a(s) + V_b(s) \tag{2.63}$$

where $V_b(s)$ is the back electromotive force voltage proportional to the motor speed.

$$V_b(s) = K_b \omega(s) \tag{2.64}$$

and the armature current is

$$I_a(s) = \frac{V_a(s) - K_b \omega(s)}{(R_a + sL_a)} \tag{2.65}$$

The load torque is

$$T_L(s) = Js^2\theta(s) + fs\theta(s) = T_m(s) - T_d(s) \tag{2.66}$$

The transfer function of armature controller DC motor where $T_d(s) = 0$ is given by

$$G(s) = \frac{\theta(s)}{V_a(s)} = \frac{K_m}{s[(R_a + sL_a)(Js + f) + K_b K_m]} = \frac{K_m}{s(s^2 + 2\xi\omega_n + \omega_n{}^2)} \tag{2.67}$$

It is of to note that K_m is equal to K_b. Let $K = K_m = K_b$ then the transfer function can be written as

$$G(s) = \frac{\theta(s)}{V_a(s)} = \frac{K}{s[(R_a + sL_a)(Js + f) + K^2]} \tag{2.68}$$

It is easy to see that the transfer function from the input voltage $V_a(s)$ to angular velocity ω is (Dorf, 1995)

$$G(s) = \frac{\omega(s)}{V_a(s)} = \frac{K}{(R_a + sL_a)(Js + f) + K^2} \tag{2.69}$$

Given the following motor data, it will be investigated different controllers.

Moment of inertia of motor	$J = 0.01 kg.m^2$
Friction of mechanical system	$f = 0.099 Nms$
Back electromotive force constant	$K = 0.01 Nm / A$
Armature resistance/inductance	$R = 1\Omega / 0.5H$

To determine step response of open loop system, $G(s) = \dfrac{\omega(s)}{V_a(s)}$ the Simulink model shown in Figure 2.14 is used. The simulation time is set to be 10 seconds. At time t = 0, the unit step input is applied to the system.

The transfer function of the system is;

$$G(s) = \frac{K}{(R_a + L_a s)(Js + f) + K^2} = \frac{0.01}{(1 + 0.5s)(0.01s + 0.099) + (0.01)^2}$$

$$= \frac{0.01}{0.005s^2 + 0.01495s + 0.01} \tag{2.70}$$

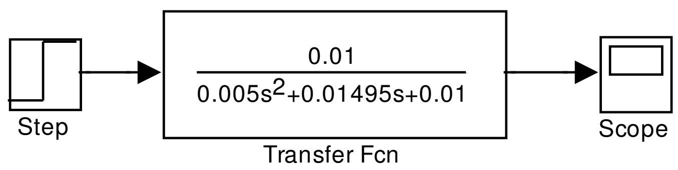


Figure 2.14 Simulink model to determine the open-loop response of the motor

The step response determined from the simulation is shown in Figure 2.15. A PID controller will be designed to control the DC motor speed. The design procedure will be introduced. Simulink model given in Figure 2.16 is used to determine necessary constant to design the PID controller. Step response test is conducted on the motor's transfer function. In the model, scopes are graphical elements that are showing the connected signals to their input.

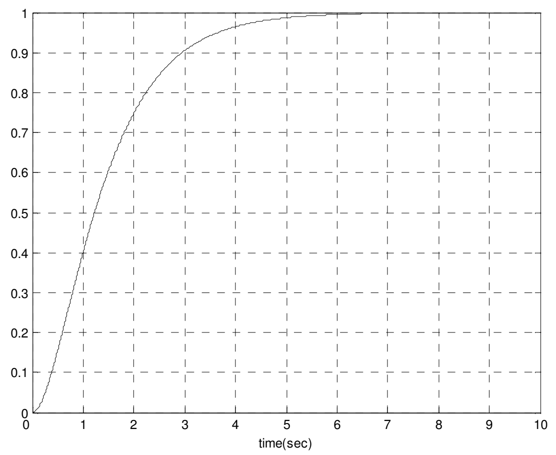


Figure 2.15 Step response of the open-loop system

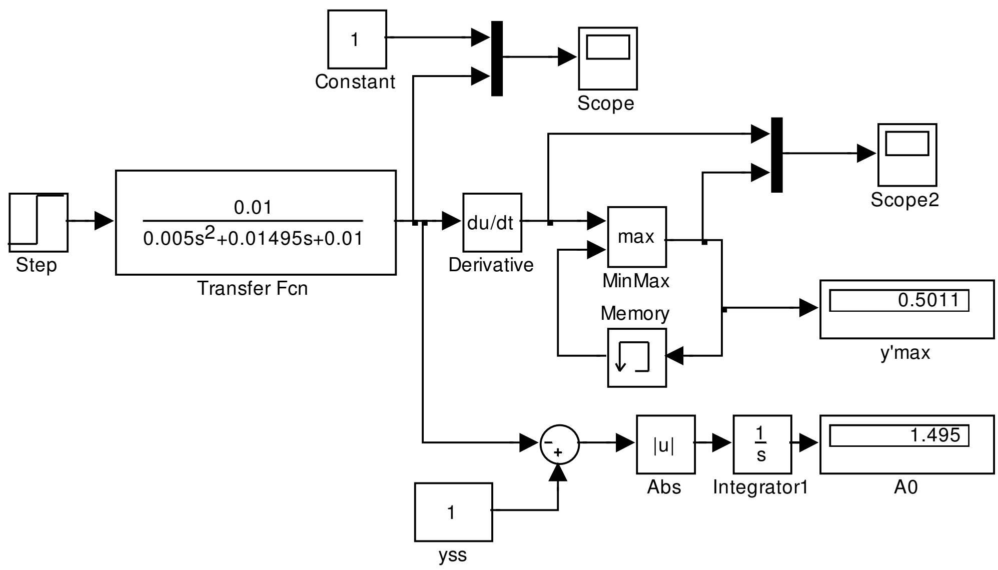


Figure 2.16 Simulink model to determine step response of the DC motor

The maximum slope, $y'_{\max}$, is the value shown in the display block, $y'_{\max} = 0.5011$. It is calculated by taking the derivative of the output and comparing this derivative with the previous maximum value using memory block.

$t_{\max}$ is the time where the maximum slope occurs. It is easily found from the Figure 2.17 which is the output waveform taken from Scope2 in the Simulink model. The maximum time is, $t_{\max} = 0.687$ sec.

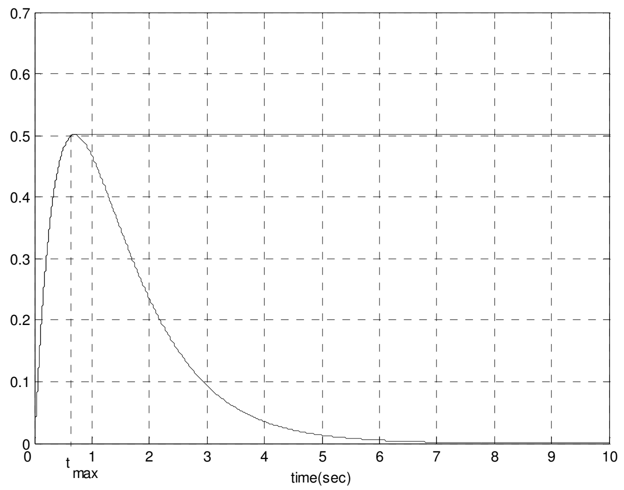


Figure 2.17 Determination of $t_{\max}$

Going one step further, $y(t_{\max})$ has to be determined as stated in design procedure.

$$y(t_{\max}) = y(0.687) = 0.2473 \tag{2.71}$$

y_0 must be determined next,

$$y_0 = (y'_{\max})(t_{\max}) - y(t_{\max})$$

$$y_0 = (0.5011)(0.687) - (0.2473) = 0.097 \tag{2.72}$$

A_0 is the area between y_{ss} and $y(t)$. It is determined with an absolute value and an integrator. In the model, $y(t)$ is simply subtracted from the y_{ss} which is 1, than the absolute value of the result of this subtraction is integrated and A_0 is computed. Second display block in the model named A0 shows the computed value of this area, 1.495.

τ_0 is the apparent dead time and found as

$$\tau_0 = \frac{y_0}{y'_{\max}} = \frac{0.097}{0.5011} = 0.1935 \text{ sec} \tag{2.73}$$

Average residence time τ_{ar} and apparent time constant τ_{at} is computed as follows

$$\tau_{ar} = \frac{A_0}{y_{ss}} = \frac{1.495}{1} = 1.495 \text{ sec} \tag{2.74}$$

$$\tau_{at} = (\tau_{ar} - \tau_0) = (1.495 - 0.1935) = 1.3015 \text{ sec} \tag{2.75}$$

Using Ziegler-Nichols and Chein-Hrones-Reswick methods, two PID controllers will be tuned to control the DC motor. Simulink model of the PID controlled system is given in Figure 2.18. Only the controller gains will be changed for each controller case, not the model.

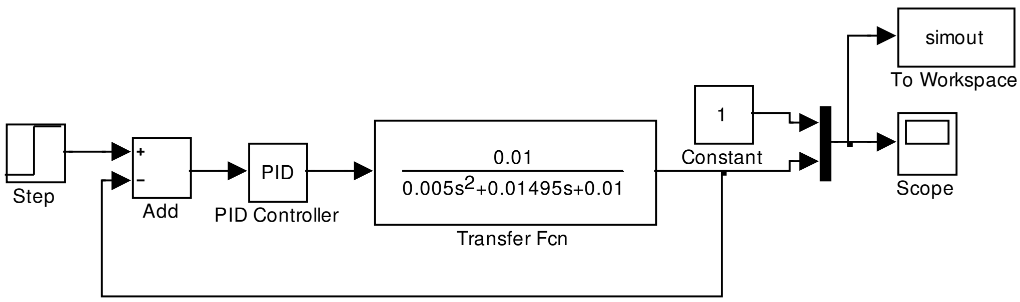


Figure 2.18 Simulink model of PID controllers

Table 2.1 gives the controller gains K_p, K_i and K_d for Ziegler-Nichols method. Using these formulas, the first controller gains are determined as;

$$K_p = \frac{6}{5y_0} = \frac{6}{5(0.097)} = \frac{6}{0.485} = 12.4 \tag{2.76}$$

$$K_i = \frac{3}{5y_0\tau_0} = \frac{3}{5(0.097)(0.1935)} = \frac{3}{0.0938} = 31.9 \quad (2.77)$$

$$K_d = \frac{3\tau_0}{5y_0} = \frac{3(0.1935)}{5(0.097)} = \frac{0.5805}{0.485} = 1.2 \quad (2.78)$$

Table 2.2 gives the controller gains K_p, K_i and K_d for Chein-Hrones-Reswick method. Using these formulas, the second controller gains are determined as;

$$K_p = \frac{3}{5y_0} = \frac{3}{5(0.097)} = \frac{3}{0.485} = 6.2 \quad (2.79)$$

$$K_i = \frac{3}{5y_0\tau_{at}} = \frac{3}{5(0.097)(1.3015)} = \frac{3}{0.632} = 4.7 \quad (2.80)$$

$$K_d = \frac{3\tau_0}{10y_0} = \frac{3(0.1935)}{10(0.097)} = \frac{0.5805}{0.97} = 0.6 \quad (2.81)$$

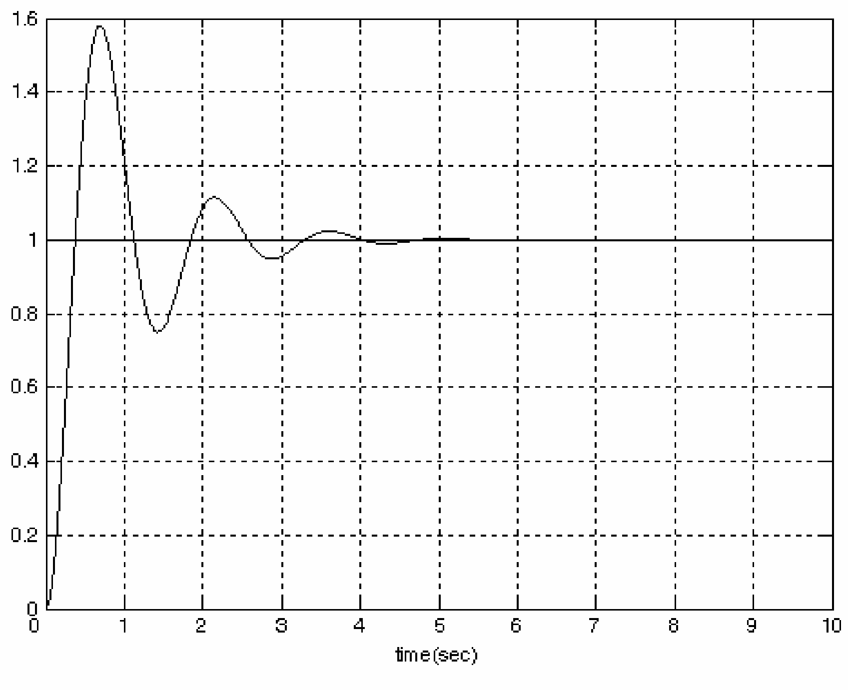


Figure 2.19 PID controller response tuned with Ziegler-Nichols method

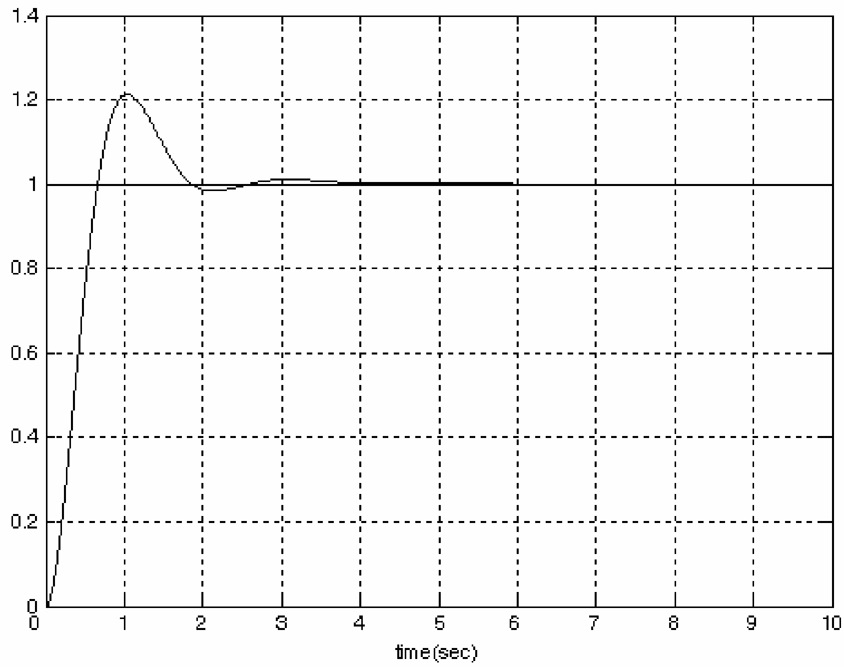


Figure 2.20 PID controller response tuned with Chein-Hrones-Reswick method

3. FUZZY CONTROL SYSTEM FUNDAMENTALS

Control applications are the kinds of problems for which fuzzy logic has had the greatest success. Many of consumer products that we use today involve fuzzy control. A paradigm shift is being realized in the area of fuzzy control, given its successes for some problems where classical control has not been effective or efficient. In the quote, above, such a paradigm shift can be explained. It was not long ago that fuzzy logic and fuzzy systems were the subject of ridicule and scorn in the scientific communities, but the control community moved quickly in accepting the new paradigm and its success in now manifested in the marketplace (Ross, 2005).

The success of fuzzy logic controllers is mainly due to their ability to cope with knowledge represented in a linguistic form instead of representation in the conventional mathematical framework. Control engineers have traditionally relied on mathematical models for their designs. However, the more complex a system, the less effective the mathematical model. This is the fundamental concept that provided the motivation for fuzzy logic (Cirstea, *et al*, 2003).

3.1 Fuzzy Sets

One can view fuzzy sets as a generalization of classical sets, or crisp sets as they are sometimes called. Classical sets and their operations are particularly useful in expressing classical logic and they lead to Boolean logic and its applications in digital systems. Fuzzy sets and fuzzy operations, on the other hand, are useful in expressing ideas of fuzzy logic leading to applications such as fuzzy controllers (Ibrahim, 2004).

3.1.1 Classical Sets

A set is defined as a collection of objects that may share certain characteristics. Each individual object is referred to as an element or member of the set. In a classical set an object x either a member of a given set *A* (expressed as $x \in A$) or not a member (expressed as $x \notin A$); partial membership is not allowed (Ibrahim, 2004).

Universal Set: The set that consists of all the elements of interest for a particular application (the universe of discourse) is referred to as the universal set and denoted by *I*.

Empty Set: A set that has no elements is referred to as an empty set and is denoted by the symbol $\varnothing$.

Cardinality: The number of elements of a given set is called the cardinality of the set. The cardinality of a set *A* is denoted by # *A*. The cardinality is a finite number for finite sets.

3.1.2 Set Operations

Complement: The complement or absolute complement of a given set is denoted by $\overline{A}$. It is defined by (Ibrahim, 2004),

$$A = \{x | x \in I, \text{ and } x \notin A\} \tag{3.1}$$

If the membership function of set *A* is $\mu_A(x)$ and that of $\overline{A}$ is $\mu_{\overline{A}}(x)$, then one can write,

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \tag{3.2}$$

This is illustrated in Figure 3.1.

Union: The union of sets *A* and *B* is defined by

$$A \cup B = \{x | x \in A \text{ or } x \in B\} \tag{3.3}$$

It is illustrated in Figure 3.2.

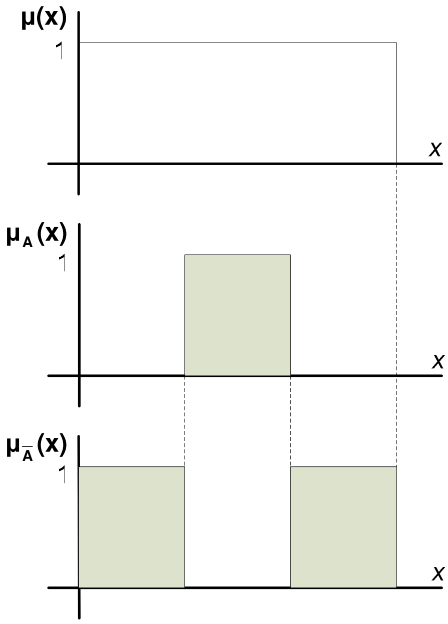


Figure 3.1 Complement of set *A*

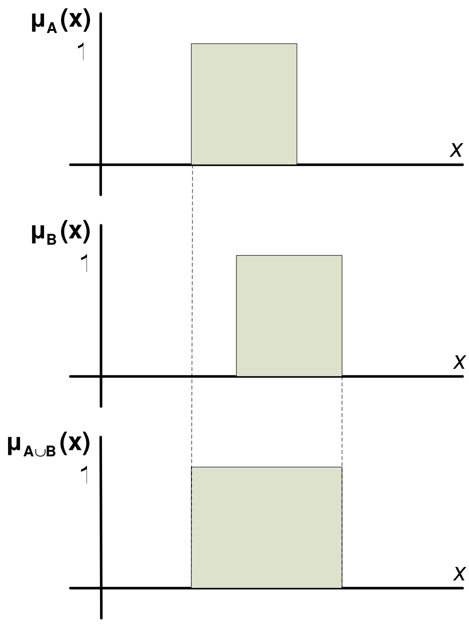


Figure 3.2 Union of sets *A* and *B*

Intersection: Intersection of sets *A* and *B* is defined by

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\} \tag{3.4}$$

It is illustrated in Figure 3.3 in terms of membership function.

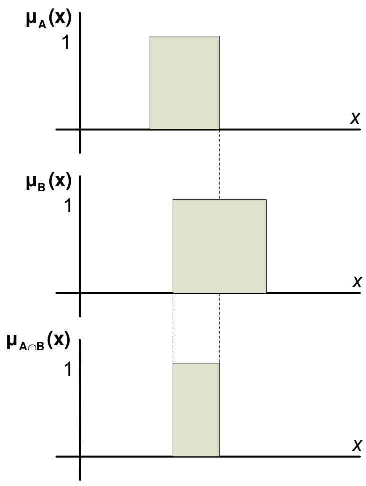


Figure 3.3 Intersection in terms of membership function.

Subtraction: The difference of sets *A* and *B* is a set that contains of all the elements which belong to *A*, but do not belong to *B.* It can be expressed as

$$A - B = A - A \cap B = \{x | x \in A \text{ and } x \notin B\} \tag{3.5}$$

3.1.3 Boolean Logic

The basic building blocks of digital systems, including computers and embedded systems, are electronic switching devices, such as transistors, or algorithms. Since such devices assume two conditions, on and off, the universe of discourse can be represented by $\{0,1\}$. A set with one element $\{0\}$, referred to as a singleton, can be represented by off condition, another singleton $\{1\}$ can represent the on condition. Operations on such sets are special cases of the general set operations; they are referred to as Boolean operations (Ibrahim, 2004).

3.1.4 Basic Concepts of Fuzzy Sets

A fuzzy set is a set where degrees of membership between 1 and 0 are allowed, it allows partial membership. Fuzzy sets can thus better reflect the way intelligent people think. For example, intelligent person will not classify people as either friends or enemies; there is a range between these two extremes. Not recognizing that there are degrees in every trait can lead to erroneous decisions.
Vague human expressions such as tall, hot, cold, etc. can be expressed by fuzzy set of the form

$$A = \{(x, \mu_A(x)) | x \in X\} \tag{3.6}$$

where X represents the universe of discourse and $\mu_A(x)$ assumes values in the range from 1 to 0.

Fuzzy sets can also be defined by assigning a continuous function to describe the membership either analytically or graphically. Some commonly used membership functions are illustrated in Figure 3.4 (Ibrahim, 2004).

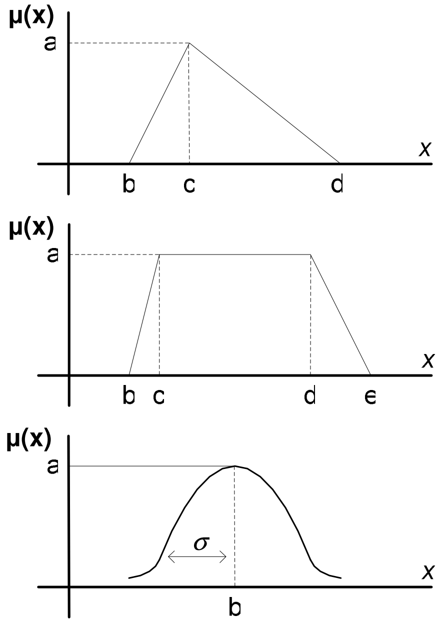


Figure 3.4 Common membership functions
a) Triangular b) Trapezoidal c) Gaussian

The triangular membership function in Figure 3.4.a can also be expressed as

$$\mu(x) = \begin{cases} a\dfrac{(b-x)}{(b-c)}, b \geq x \leq c \\ a\dfrac{(d-x)}{(d-c)}, c \geq x \leq d \\ 0, otherwise \end{cases} \tag{3.7}$$

The trapezoidal function shown in Figure 3.4.b can be expressed as

$$\mu(x)=\begin{cases} a\dfrac{(b-x)}{(b-c)};b\geq x\leq c \\ a;c\geq x\leq d \\ a\dfrac{(e-x)}{(e-d)};d\geq x\leq e \\ 0;otherwise \end{cases} \tag{3.8}$$

The Gaussian function in Figure 3.4.c can be expressed as

$$\mu(x)=a\exp(-(x-b)^2/2\sigma^2) \tag{3.9}$$

3.1.5 Fuzzy Sets Properties

Empty Fuzzy Set: A fuzzy set is referred to as empty if and only if the value of the membership function is zero for all possible members under consideration. In a short hand form this statement would read

$$A=\varnothing \text{ if and only if } \mu_A(x)=0, \forall x\in X \tag{3.10}$$

Universal Fuzzy Set: A fuzzy set is universal if and only if the value of the membership function is one for all members under consideration.

Equal Fuzzy Sets: A fuzzy set may be completely characterized by its α-cuts, defined as follows

Strong α-cuts, $A_\alpha=\{x|\mu_A(x)>\alpha\};\alpha\in[0,1]$

Weak α-cuts, $A_{\overline{\alpha}}=\{x|\mu_A(x)\geq\alpha\};\alpha\in[0,1]$

Thus, an α-cut is a crisp set that consists of all the elements of a fuzzy set whose membership functions have values greater than a specified value α, or greater than or equal to a specified value; the first condition leads to strong α-cuts and the second to weak α-cuts. All the cuts of a fuzzy set form a family of crisp sets.

Support: The support of a fuzzy set A is a crisp set supp(A) of all $x \in X$ such that $\mu_A(x) > 0$. It is a strong α-cut for $\alpha = 0$. The element $x \in X$ at which $\mu_A(x) = 0.5$ is referred to as the cross-over point. A fuzzy set whose support is a single element in X with $\mu_A(x) = 1$ is referred to as a fuzzy singleton.

Core: The core of a fuzzy set A is a crisp set core(A) of all $x \in X$ such that $\mu_A(x) = 1$. The core of a fuzzy set may be an empty set.

Height: The height, h(A) of a fuzzy set A is the largest value of μ_A for which the α-cut is not empty. In other words, it is the largest value of membership function attained by an element in the set. A fuzzy set with h(A) = 1 is referred to as normal, otherwise it will be referred to as sub-normal. The concepts of α-cuts, support, core and height are illustrated in Figure 3.5 (Ibrahim, 2004).

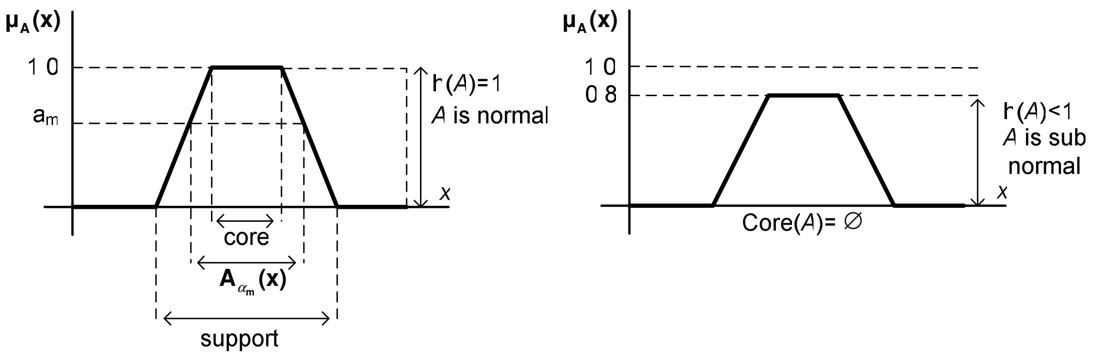


Figure 3.5 α-cuts, support, core and height

3.1.6 Operations on Fuzzy Sets

Complement: The absolute complement of a fuzzy set A is denoted by $\overline{A}$ and its membership function is defined by

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \text{ for all } x \in X \tag{3.11}$$

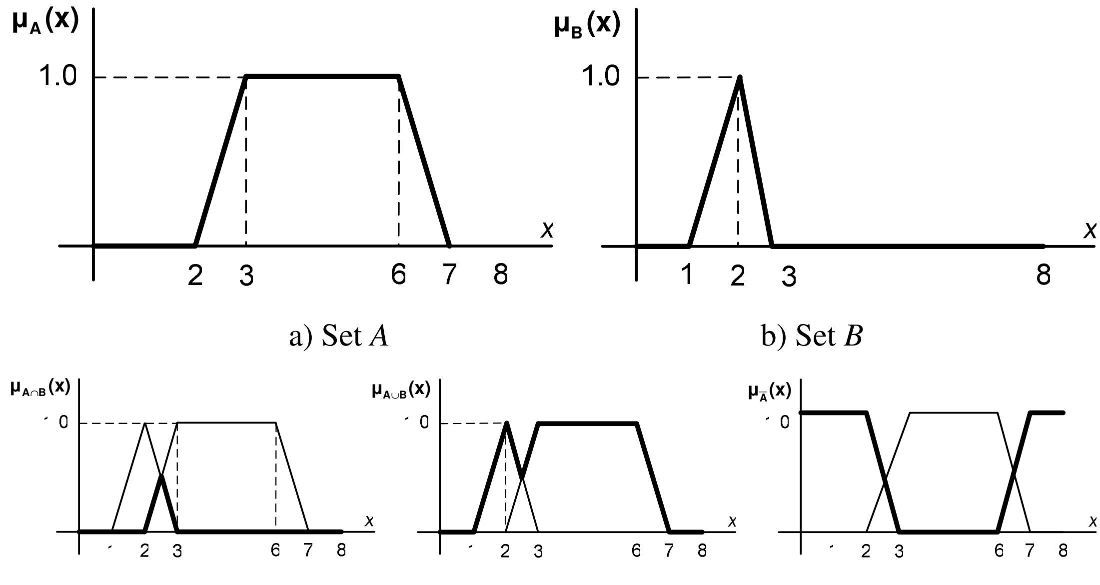
Union: The union of two fuzzy sets A and B is a fuzzy set whose membership function is defined by

$$\mu_{A\cup B}(x) = \max[\mu_A(x), \mu_B(x)] \tag{3.12}$$

Intersection: The intersection of two fuzzy sets A and B is a fuzzy set whose membership function is defined by

$$\mu_{A\cap B}(x) = \min[\mu_A(x), \mu_B(x)] \tag{3.13}$$

A graphical illustration of basic fuzzy sets operations is presented in Figure 3.6



a) Set A b) Set B

c) $A \cap B$ d) $A \cup B$ e) $\overline{A}$

Figure 3.6 Graphical illustrations of basic fuzzy set operations

Cartesian Multiplication: The Cartesian multiplication of two sets A and B is a fuzzy set C such that

$$C = A \times B = \{\mu_C(x)/(a,b) \,|\, a \in A, b \in B, \mu_C(c) = \min[\mu_A(a), \mu_B(b)]\} \tag{3.14}$$

Algebraic Multiplication: The algebraic product of two fuzzy sets *A* and *B* leads to a fuzzy set *C* such that

$$AB = \{\mu_A(a)\mu_B(b)/x \mid x \in A, x \in B\} \tag{3.15}$$

Exponent: Raising a set *A* to the power of α is a special case of algebraic multiplication. It is defined by

$$A^{\alpha} = \{(\mu_A(x))^{\alpha}/x \mid x \in A\} \tag{3.16}$$

Concentration and Dilation: These two operations are unique to fuzzy sets; they do not have counterparts in classical sets. The concentration and dilation are defined as

$$CON(A) = A^2 \tag{3.17}$$

$$DIL(A) = A^{0.5} \tag{3.18}$$

Algebraic Sum: The algebraic sum of two sets *A* and *B* is a fuzzy set with a membership function given by

$$\mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x) \tag{3.19}$$

Bounded Sum: The symbol $\oplus$ denotes the bounded sum of two fuzzy sets. The operation leads to a fuzzy set with a membership function defined by

$$\mu_{A \oplus B}(x) = \min[1, (\mu_A(x) + \mu_B(x))] \tag{3.20}$$

3.2 Fuzzy Logic Systems

In general a Fuzzy Logic System (FLS) is a nonlinear mapping of an input data vector into a scalar output where the vector output case decomposes into a collection of independent multi input/single output systems. The richness of fuzzy logic is that there are enormous numbers of possibilities that lead to lots of different mappings. This richness does require a careful understanding of fuzzy logic and the elements that comprise a FLS (Mendel, 1995).

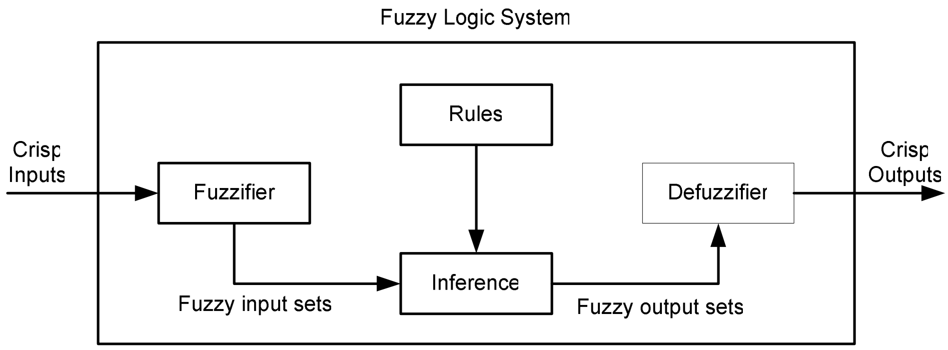


Figure 3.7 Fuzzy logic system

The basic structure of a FLS system is given in Figure 3.7. It is possible to write a mathematical formula that relates the output of the FLS to its inputs.

Fuzzification involves the following functions;

a) measures the values of input variables

b) performs a scale mapping that transfers the range of values of input variables into corresponding universes of discourse

c) performs the function of fuzzification that converts input data into suitable linguistic values which may be viewed as labels of fuzzy sets.

Rules comprise a knowledge of the application domain and the attendant control goals. The rule base characterizes the control goals and control policy of the domain experts by means of a set of linguistic control rules.

The decisionmaking logic is the kernel of FLS; it has the capability of simulating human decision making based on fuzzy concepts and of inferring fuzzy control actions employing fuzzy implication and the rules of inference in fuzzy logic.

Defuzzifier performs the following functions;

a) a scale mapping, which converts the range of values of output variables into corresponding universe of discourse

b) defuzzification, which yields a nonfuzzy control action from an inferred fuzzy control action (Lee, 1990).

3.2.1 Rules

In a FLS, the dynamic behavior of a fuzzy system is characterized by a set of linguistic description rules based on expert knowledge. The expert knowledge is usually of the form

IF (a set of conditions are satisfied) THEN (a set of consequences can be inferred)

Since the antecedents and the consequents of these IF-THEN rules are associated with fuzzy concepts (linguistic terms), they are often called fuzzy conditional statements. A fuzzy control rule is a fuzzy conditional statement in which the antecedent is a condition in its application domain and the consequent is a control action for the system under control. Basically, fuzzy control rules provide a convenient way for expressing control policy and domain knowledge. Furthermore, several linguistic variables might be involved in the antecedents and the conclusions of these rules. When this is the case, the system will be referred to as a multi-input-multi-output (MIMO) fuzzy system. In the case of two input one output fuzzy system (MISO), fuzzy control rules have the form;

R_1: IF x is A_1 AND y is B_1 THEN z is C_1,

R_2: IF x is A_2 AND y is B_2 THEN z is C_2,

…

R_n: IF x is A_n AND y is B_n THEN z is C_n

where x, y and z are linguistic variables representing two process state variables and one control variable; A_i, B_i and C_i are linguistic values of linguistic variables x, y and z in the universe of discourse U, V and W, respectively, with i=1,2,…,n; and implicit sentence connective also links the rules into a rule set or, equivalently, a rule base.

A fuzzy control rule, such as "IF x is A_i AND y is B_i THEN z is C_i" is implemented by a fuzzy implication function (fuzzy relation) R_i and is defined as follows:

$\mu_{R_i} = [\mu_{A_i}(u) \text{ and } \mu_{B_i}(v)] \rightarrow \mu_{C_i}(w)$

where → denotes a fuzzy implication function (Lee, 1990).

3.2.2 Fuzzy Inference Engine

Fuzzy inference is the kernel in a fuzzy logic system. It has the capability of simulating human decision making based on fuzzy concepts and of inferring fuzzy control actions employing fuzzy implication and the rules of inference in fuzzy logic. In the fuzzy inference engine, fuzzy logic principles are used to combine fuzzy IF – THEN rules from the fuzzy rule base into a mapping from fuzzy input sets to fuzzy output sets.

Max-Min inference is widely used inference technique. In max-min inference, the implication operator used is min, i.e,

$$m_{ij} = truth(a_i \rightarrow b_j) = \min(a_i, b_j) \tag{3.21}$$

where $a_i \rightarrow b_j$ is a fuzzy implication which effectively means if a_i then b_j. Given two fuzzy sets *A* and *B*, it is illustrated in Figure 3.8 (Song, *et al*, 1997).

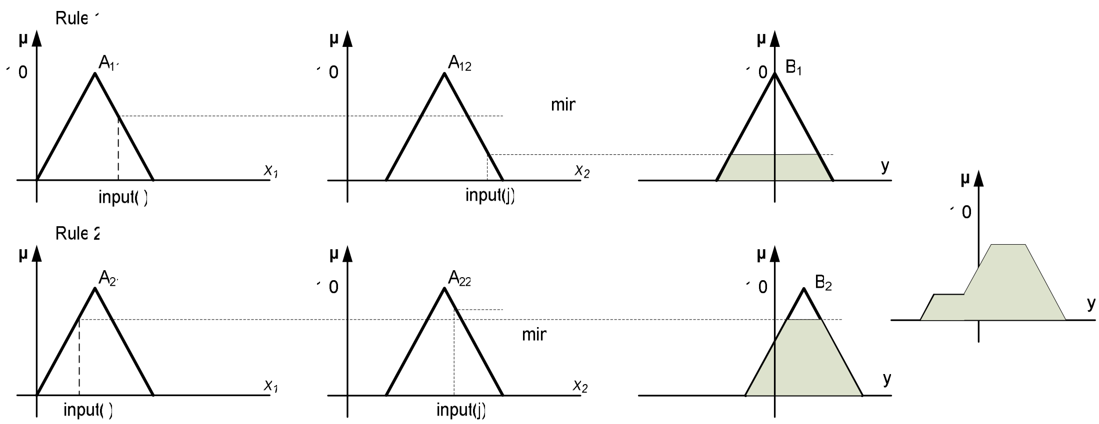


Figure 3.8 Max-min inference

There are more inference operators other than max-min inference, the FLC studied in this thesis includes the max-min inference mechanism.

3.2.3 Fuzzification

Fuzzification is the operation of transforming a crisp set to a fuzzy set, or a fuzzy set to a fuzzifier set. The operation translates crisp input or measured values into linguistic concepts. This, in a way, is similar to what people may do in numerous situations to reach a decision (Ibrahim, 2004).

Fuzzification is related to the vagueness and imprecision in a natural language. It is a subjective valuation which transforms a measurement into a valuation of a subjective value, and hence it could be defined as a mapping from an observed input space to fuzzy sets in certain input universes of discourse. Fuzzification plays an important role in dealing with uncertain information which might be objective or subjective in nature. In fuzzy control applications, the observed data are usually crisp. Since the data manipulation in an FLC is based on fuzzy set theory, fuzzification is necessary during an carlier stage.

A fuzzification operator interprets an input x_0 as a fuzzy set A with the membership function $\mu_A(x)$ equal to zero except at the point x_0, at which $\mu_A(x_0)$ equals one (Lee, 1990).

3.2.4 Defuzzification

Basically, defuzzification is a mapping from a space of fuzzy control actions defined over an output universe of discourse into a space of non-fuzzy control actions. It is employed because in many practical applications a crisp control action is required.

A defuzzification strategy is aimed at producing a non-fuzzy control action that best represents the possibility distribution of an inferred fuzzy control action. Unfortunately, there is no systematic procedure for choosing a defuzzification strategy. At present, the commonly used strategies may be described as the max criterion, the mean of maximum, and the center of area.

- The max criterion:

The max criterion produces the point at which the possibility distribution of the control reaches a maximum value.

- The mean of maximum (MOM):

The MOM strategy generates a control action which represents the mean value of all local control actions whose membership functions reach the maximum value. More specifically, in the case of discrete universe, the control action may be expressed as

$$y_0 = \sum_{i=1}^{n} \frac{w_i}{n} \tag{3.22}$$

where w_i is the support value at which the membership function reaches the maximum value $\mu(w_i)$, and n is the number of such support values.

- Center of gravity

The widely used strategy generates the center of gravity of the possibility distribution of a control action. In the case of discrete universe, this method yields,

$$y_0 = \frac{\sum_{i=1}^{n} \mu(w_i) w_i}{\sum_{i=1}^{n} \mu(w_i)} \tag{3.23}$$

where n is the number of quantization levels of the output.

3.3 Fuzzy Logic Controller Design

3.3.1 Description

The main building units of an FLC are a fuzzification unit, a rule base, an inference mechanism, and a defuzzification unit. Defuzzification is the process of converting inferred fuzzy control actions into a crisp control action.

The essence of a fuzzy logic controller is thus based on a linguistic model as opposed to a mathematical model, as is the case with PID controller. Fuzzy logic controllers are used to reduce the development time or to improve the performance of an existing PID controller. In the case of highly complex systems, fuzzy logic could be the only solution (Ibrahim, 1990).

3.3.2 Design

In the design of an FLC system, it is assumed that;

- A solution exists.
- The input and output variables can be observed and measured.
- An adequate solution is acceptable.
- A linguistic model can be created based on the knowledge of a human expert.

In order to model a system linguistically, one needs to;

- Identify the input and output variables of the process to be controlled.
- Define subsets that cover the universe of discourse of each variable and assign a linguistic label to each one.
- Form a rule base by assigning relationships between inputs and outputs.
- Determine a fuzzification method.
- Determine a defuzzification method to be used to generate a crisp output from the fuzzy outputs generated from the rule base (Ibrahim, 1990).

3.3.3 Analysis

Fuzzy logic is particularly useful in the control of highly nonlinear processes difficult to control by conventional methods. The central idea is to use the knowledge of an expert (human operator) to create a linguistic model. However, conventional analysis techniques of control systems are based on mathematical models and hence cannot be used for the analysis of a fuzzy logic control system. One may argue that if the model is based on the knowledge of an expert then the analysis is not needed. However, one must realize that human knowledge could be incomplete. In addition, fidelity of the mapping using rule base, inference rules, and defuzzification methods needs to be assured. Extensive simulations, in general, would be required to establish confidence in the reliability of the system. One should examine the static properties of a designed FLC system and observe aspects such as;

- The consistency of the IF/THEN rules. Inconsistencies could occur due to an attempt to accommodate contradictory design criteria or simply due to faulty formulation of the rules.
- The existence of enough rules to assure smooth control action, but not to many to slow the operation.
- A proper control action should be inferable for every state of the process.
- Fuzzy sets are properly assigned in the defined universe of discourse. Overlapping of neighboring membership functions ensures completeness and robustness, but may lead to distortions in the output.
- The defuzzification method used is suitable to ensure correct and smooth control action.

As for the dynamic properties, numerous techniques have been put forward for the analysis of the stability on an FLC system. They can be categorized into twp groups;

- The first group consists of methods that require a mathematical model of the process to be controlled. This category includes the cases of linear and nonlinear plants. The describing function technique and the circle criterion are example of the first. The invariance principle and Lyapunov function are examples of the second.

- The second group consists of methods that do not require a mathematical model. They include phase plane trajectory, energetistic stability, and linguistic stability methods (Ibrahim, 1990).

3.4 Case Study

DC motor control will be studied to demonstrate the differences between the control strategies. In this case, a 5 HP DC motor with 5Nm load is used to demonstrate the fuzzy controller. The parameters for the test motor are given in Table 3.1.

Table 3.1 DC motor parameters

Armature voltage (V)	240	Field inductance (H)	156
Field voltage (V)	150	Field-armature inductance (H)	1.976
Armature resistance (Ω)	11.2	Total inertia (kg.m^2)	0.02215
Armature inductance (H)	0.1215	Viscous friction coefficient (N.m.s)	0.002953
Field resistance (Ω)	281.3	Coulomb friction torque (N.m)	0.5161

DC motor can be controlled either by controlling the field current or armature current. In the case study, it is assumed that, the field voltage is fixed. Armature voltage so the armature current is controlled by the fuzzy logic controller. The Simulink model of the system is given in Figure 3.9.

The 1 arm universal bridge in the model is used to generate the controlled armature voltage to the DC motor. There are two Insulated Gate Bipolar Transistors (IGBT) connected in series as shown in Figure 3.10.

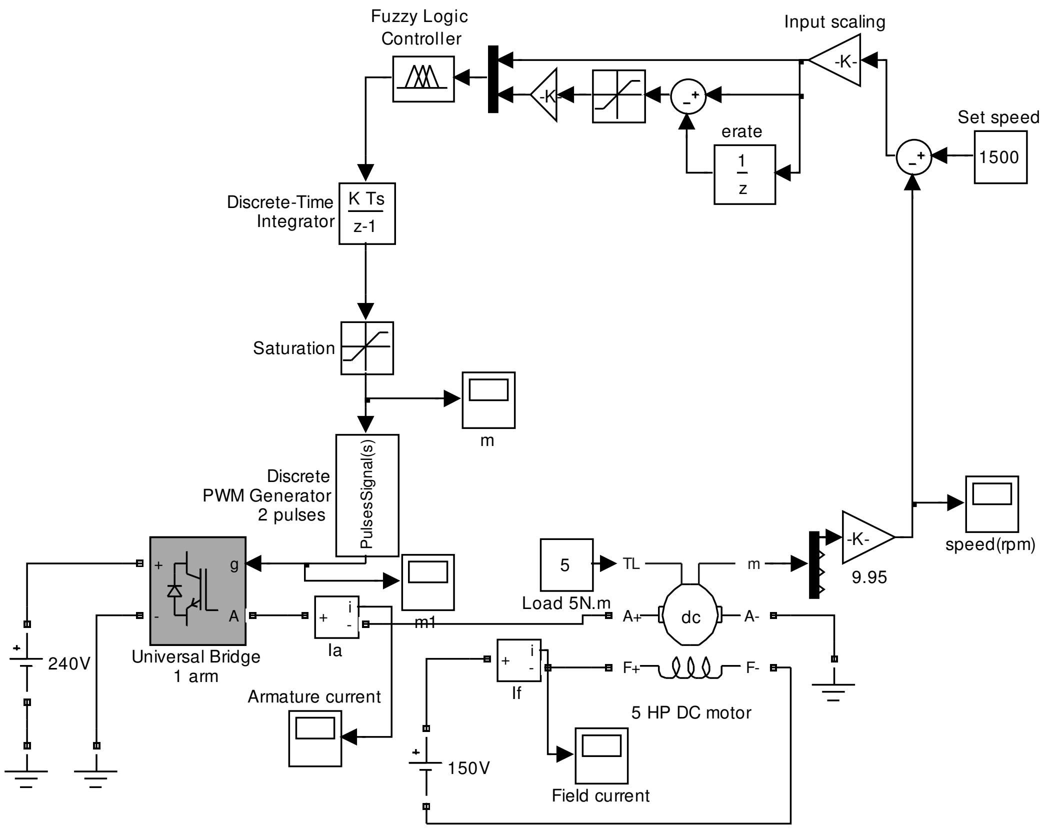


Figure 3.9 Fuzzy logic controller in action

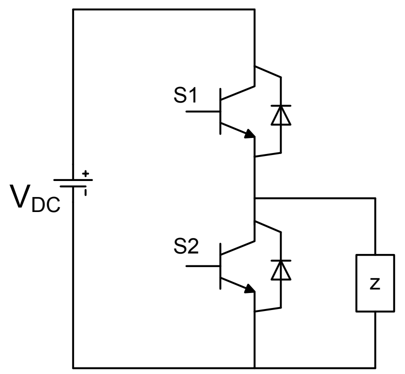


Figure 3.10 1 arm universal bridge

S1 and *S2* switches are controlled by the Discrete PWM Generator. The discrete PWM generator accepts a signal which is taking values between $0 \leq m \leq 1$ where *m* is known as modulation index. Modulation index controls the average value of DC output of the DC-DC converter. The average output voltage of the 1 arm universal bridge is given by the following formula,

$$V_{ave} = \frac{(m+1)}{2} V_{dc} \tag{3.24}$$

where m is the modulation index.

The armature voltage is controlled by this 1 arm universal bridge where the switching signals are generated by the discrete PWM generator. The control signal, m, is obtained from,

$$m = m + \Delta m \tag{3.25}$$

where Δm is the output of the fuzzy logic controller. This equation is simulated by using a discrete time integrator.

The Δm output is taking relatively large values to increase the motor speed to reach its set point. Because m determines the armature voltage of the DC motor and Δm determines m. When the desired speed is achieved the Δm control signal is taking values approximately 0 to keep the motor speed constant at the set point. The modulation index is m is fixed to give the desired speed by using the Eq.3.24. The fuzzy logic controller has two inputs, error and error rate (change of error) given by the following equations.

$$error = n_{set}(k) - n_{act}(k) \tag{3.26}$$

$$erate = e(k) - e(k-1) \tag{3.27}$$

where $n_{set}(k)$ is desired (set) speed of the DC motor and $n_{act}(k)$ is the actual speed of the motor. erate is the error rate (change of error) which is found by using a $1/z$ unit delay element.

In the design process of fuzzy logic controller, first, the system is simulated without a controller with different modulation index values and system behavior is observed. An experimental knowledge is obtained from these simulations. The inputs of the controllers, error and erate are than scaled to -1 to 1 values to make the system variables normalized. The output of the controller, Δm, is also set to take values between -1 to 1. Input scalers shown in the Figure 3.9 are used for scaling these values. As a result of this, the universe of discourse for all, error, erate and Δm is -1 to 1.

The design procedure continues with the selection of membership functions for the inputs and the output of the controller. error and erate have the following membership functions shown in Figure 3.11 and Figure 3.12, respectively.

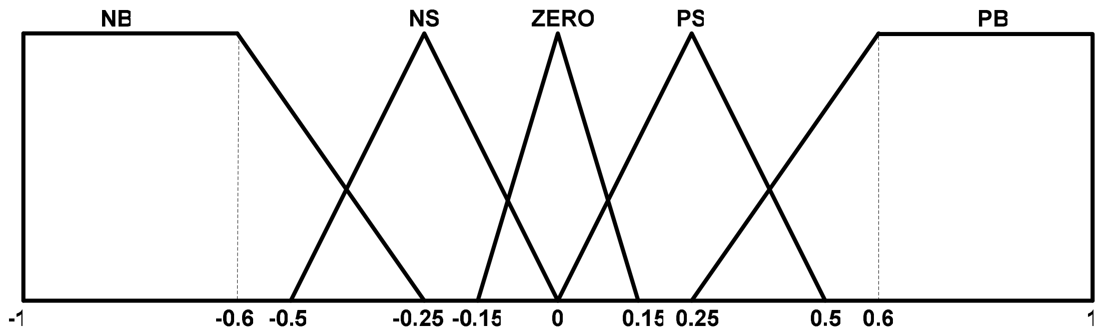


Figure 3.11 Membership functions for error input

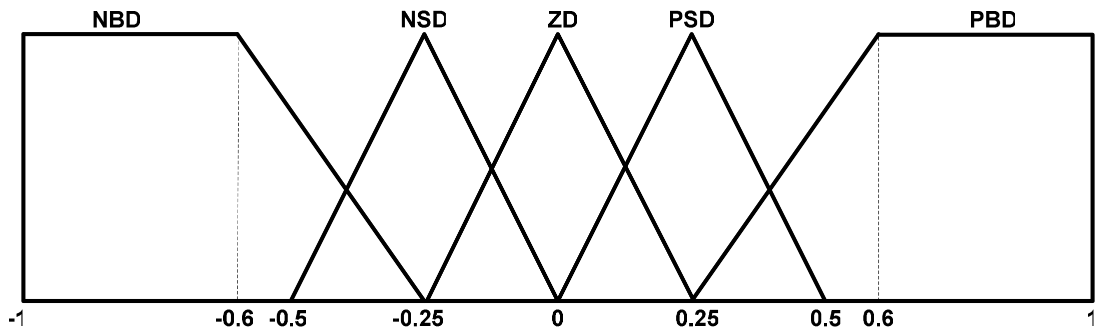


Figure 3.12 Membership functions for erate input

The fuzzy logic controller has only one output, Δm, the membership functions for this output is selected as shown in Figure 3.13.

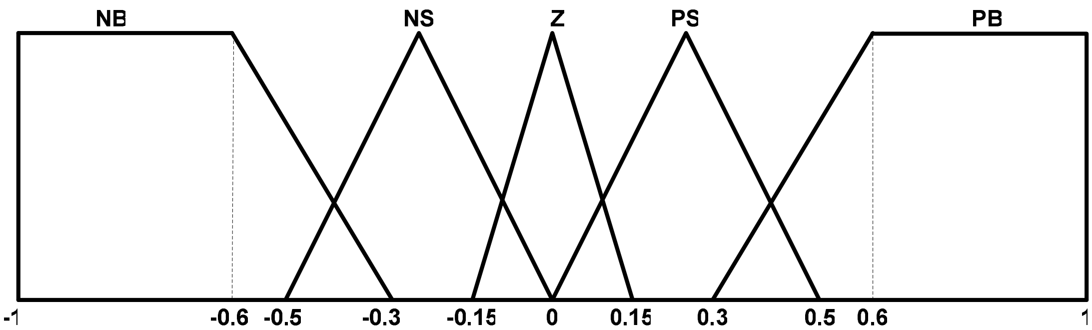


Figure 3.13 Membership functions for Δm output

There two types of membership functions used in the fuzzy logic controller, triangle shaped and trapezoid shaped.

Trapezoidal Membership Functions

This type of membership functions has four parameters *a*, *b*, *c*, *d*. A typical trapezoidal membership functions are illustrated in Figure 3.14 with these parameters with different values. Note that the trapezoid on the left hand side have *a* and *b* parameters equal, and the trapezoid on the right hand side have the *c* and *d* parameters equal.

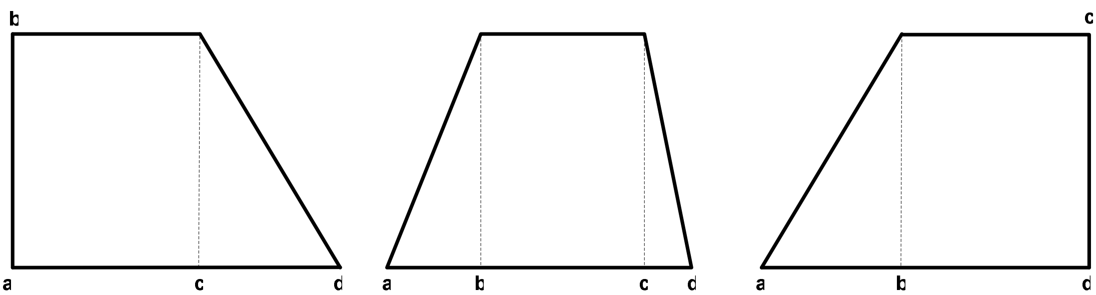


Figure 3.14 Different trapezoidal membership functions

For trapezoidal membership functions there are regions where the input has full membership and regions that input has partial membership. For instance, trapezoid in the center in Figure 3.14 has three different regions, the first region is where the input take values between a and b, in this region the input has partial membership. The second region, where the input take values between b and c, has full membership. In the last region where the input is laying between c and d, the input has again partial membership.

Triangle Membership Functions

The second type of membership function used is triangle shaped. Triangle membership functions has three parameters a, b and c. Figure 3.15 shows some different configuration triangle membership functions.

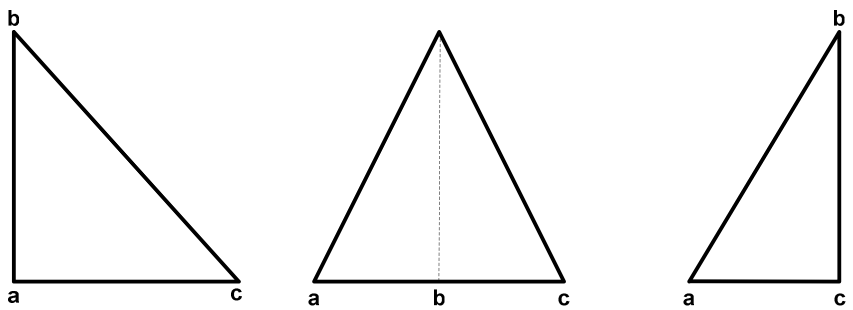


Figure 3.15 Different triangle membership functions

For triangle membership functions, generally the input has partial membership at all points except the top point of the triangle. As an instance, the membership function on the left of the Figure 3.15, the input has full membership at point a = b, and partial membership between b and c. The membership function on the right of the figure, the input has full membership at point b=c, and partial membership between a and c. For the triangle function on the center, the input has partial membership between a and b, b and c, full membership at point b.

The linguistic terms for membership functions that will be used in the rule base are labeled on each of the corresponding membership function. These are given in Table 3.2.

Table 3.2 Linguistic terms for the error, erate and Δm

error		erate		Δm	
NB	Negative Big	NBD	Negative Big Dot	NB	Negative Big
NS	Negative Small	NSD	Negative Small Dot	NS	Negative Small
ZERO	Zero	ZD	Zero Dot	Z	Zero
PS	Positive Small	PSD	Positive Small Dot	PS	Positive Small
PB	Positive Big	PBD	Positive Big Dot	PB	Positive Big

The fuzzy logic controller use linguistic knowledge instead of mathematical equations. This knowledge is obtained from the experts, experiments, and human experience. The fuzzy controller has rules that uses the linguistic terms defined in the above table to construct the control rules. The rule base of the controller determines the output of the controller. The rule base of fuzzy logic controller is developed from the typical step response of a system (Elmas, 2003). A typical step response of a system is shown in Figure 3.16.

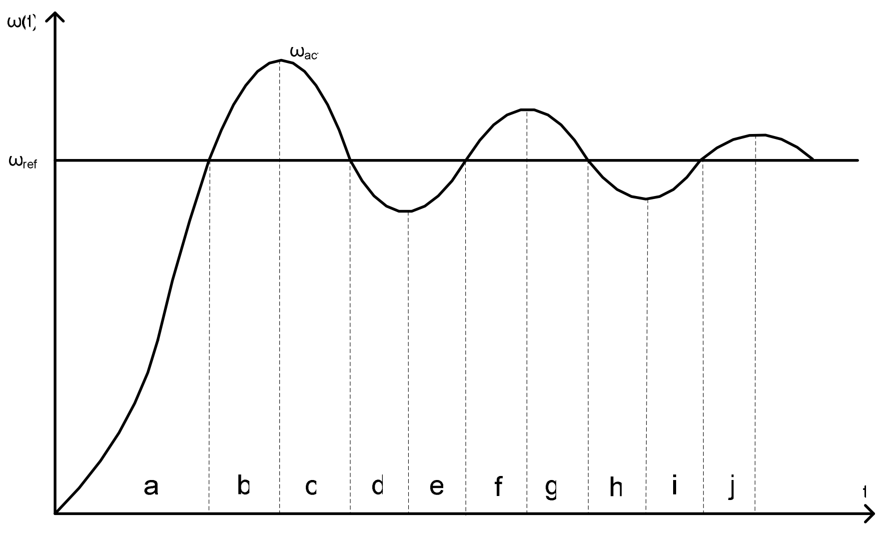


Figure 3.16 Typical step response of the system

In the following table, Table 3.3, the signs of error and erate are pointed out for regions a, b, c, d, e, f, g, h, i and j shown in Figure 3.16.

Table 3.3 Signs of the error and erate signals

	a	b	c	d	e	f	g	h	i	j
error	+	-	-	+	+	-	-	+	+	-
erate	-	-	+	+	-	-	+	+	-	-

For region a, error is positive big, error rate is negative big, then the control signal Δm must be positive big to decrease the speed error. In region b, error is negative big, and error rate is also negative big, then the control action must be negative to take speed down to the reference speed. The remaining rules are determined similar to this approach from the following regions of step response.

For the designed controller, there are two inputs error and erate, which have 5 membership functions for each. Normally, there must be 25 rules in the rule base. The rule base for the fuzzy controller is listed below and Table 3.4 summarizes the rule base;

1- IF error is (NB) AND erate is (NBD) THEN Δm is (NB)
2- IF error is (NB) AND erate is (NSD) THEN Δm is (NB)
3- IF error is (NB) AND erate is (ZD) THEN Δm is (NB)
4- IF error is (NB) AND erate is (PSD) THEN Δm is (NS)
5- IF error is (NB) AND erate is (PBD) THEN Δm is (NS)
6- IF error is (NS) AND erate is (NBD) THEN Δm is (NB)
7- IF error is (NS) AND erate is (NSD) THEN Δm is (NS)
8- IF error is (NS) AND erate is (ZD) THEN Δm is (NS)
9- IF error is (NS) AND erate is (PSD) THEN Δm is (Z)
10- IF error is (NS) AND erate is (PBD) THEN Δm is (Z)
11- IF error is (ZERO) AND erate is (NBD) THEN Δm is (NS)
12- IF error is (ZERO) AND erate is (NSD) THEN Δm is (Z)
13- IF error is (ZERO) AND erate is (ZD) THEN Δm is (Z)
14- IF error is (ZERO) AND erate is (PSD) THEN Δm is (Z)
15- IF error is (ZERO) AND erate is (PBD) THEN Δm is (PS)
16- IF error is (PS) AND erate is (NBD) THEN Δm is (Z)

17- IF error is (PS) AND erate is (NSD) THEN Δm is (Z)

18- IF error is (PS) AND erate is (ZD) THEN Δm is (PS)

19- IF error is (PS) AND erate is (PSD) THEN Δm is (PS)

20- IF error is (PS) AND erate is (PBD) THEN Δm is (PB)

21- IF error is (PB) AND erate is (NBD) THEN Δm is (PS)

22- IF error is (PB) AND erate is (NBD) THEN Δm is (PS)

23- IF error is (PB) AND erate is (NBD) THEN Δm is (PB)

24- IF error is (PB) AND erate is (NBD) THEN Δm is (PB)

25- IF error is (PB) AND erate is (NBD) THEN Δm is (PB)

Table 3.4 Rule base of the fuzzy logic controller

erate / error	**NBD**	**NSD**	**ZD**	**PSD**	**PBD**
NB	NB	NB	NB	NS	NS
NS	NB	NS	NS	Z	Z
Z	NS	Z	Z	Z	PS
PS	Z	Z	PS	PS	PB
PB	PS	PS	PB	PB	PB

Several simulations are conducted to achieve a good performance, the rule base is changed during these simulations and the one given in above table is the resulting rule base.

When designing a fuzzy logic control system, some methods must be chosen for AND method, implication method, aggregation method and defuzzification method. The following methods are chosen for the designed controller and are given in Table 3.5.

Table 3.5 Methods used in the controller

AND method	MIN
Implication method	MIN
Aggregation method	MAX
Defuzzification method	Center of gravity

To illustrate the methods used in the controller, consider the situation when the error is 0.2 and erate is 0.37. For this situation, error has only partial PS membership and erate has both partial PSD and PBD memberships. From the rule base, there are two rules, 19 and 20, affecting the output. These are;

19. IF error is (PS) AND erate is (PSD) THEN Δm is (PS)

20. IF error is (PS) AND erate is (PBD) THEN Δm is (PB)

Figure 3.17 is the graphical illustration of the rule 19. error has partial membership value of 0.8 in PS and erate has partial membership value of 0.48 in PSD.

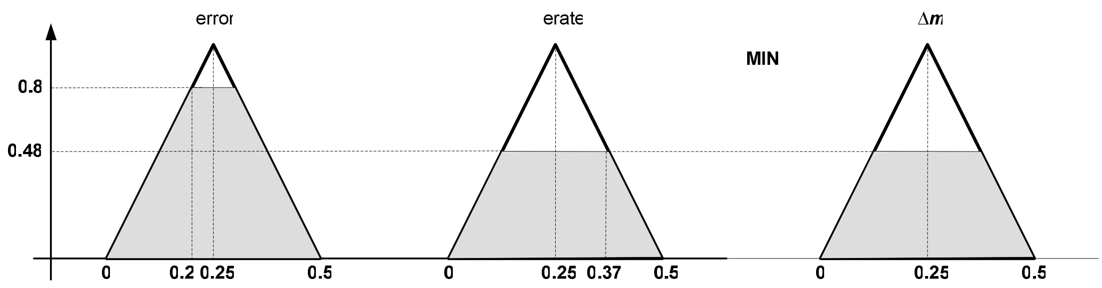


Figure 3.17 Illustration of rule 19.

Figure 3.18 is the graphical illustration of rule 20. error has the partial membership value of 0.8 in PS and erate has partial membership value of 0.23 in PBD.

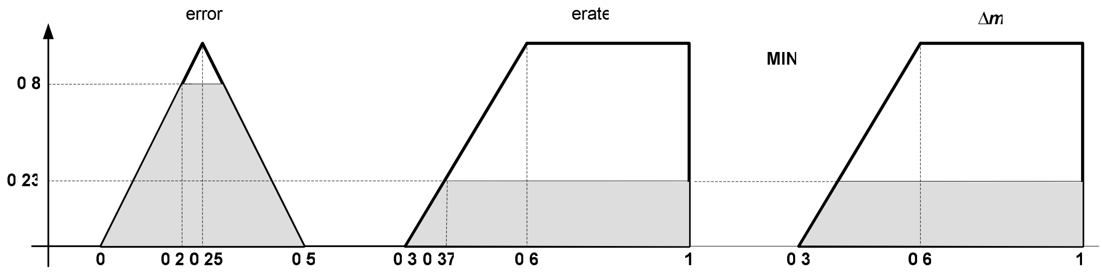


Figure 3.18 Illustration of rule 20.

As seen from figures 3.17 and 3.18, the output has affected from two membership functions, PS and PB partially. MIN is the implication operator when calculating the output membership value.

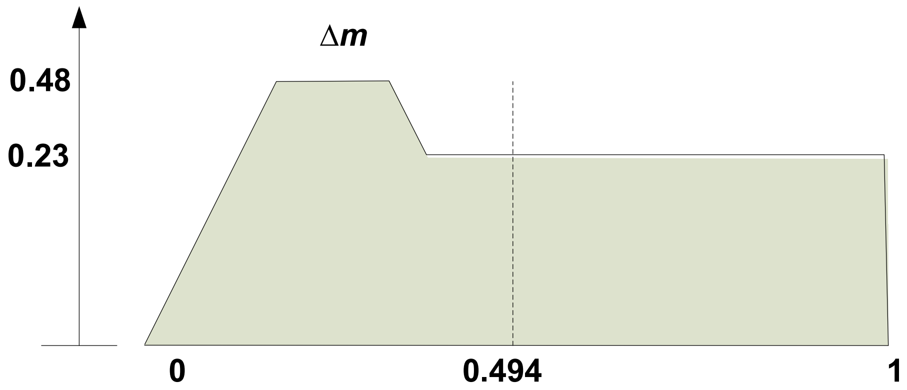


Figure 3.19 The output fuzzy set for the situation

The resulting fuzzy set is shown in Figure 3.19 for the output, Δm. This figure is obtained after applying aggregation method which is MAX. In this method, for each of the output membership function, the membership function with the maximum value is selected and than the all membership functions are aggregated to obtain the output membership function.

Lastly, defuzzification method must be implemented. The one used in the design of fuzzy controller is center of gravity. The center of gravity for the above figure is calculated as 0.494. This value is the defuzzified output of the fuzzy controller.

The design procedure is finished, and the resulting controller is tested with simulation studies. It must be noted that the rules, the membership function parameters are changed several times to obtain a good performance. The performance of the controller depends on the experience of the designer. As a result, it can be said that the more experienced the designer, more powerful the controller.
The simulation results for the DC motor speed control problem are shown in the following figures. In Figure 3.20, n_{set} is 1500 rpm and load torque is 5 Nm. Figure 3.21 depicts the speed response of the controller where the load torque is switched from 3 Nm to 5 Nm at time equals to 3 sec.

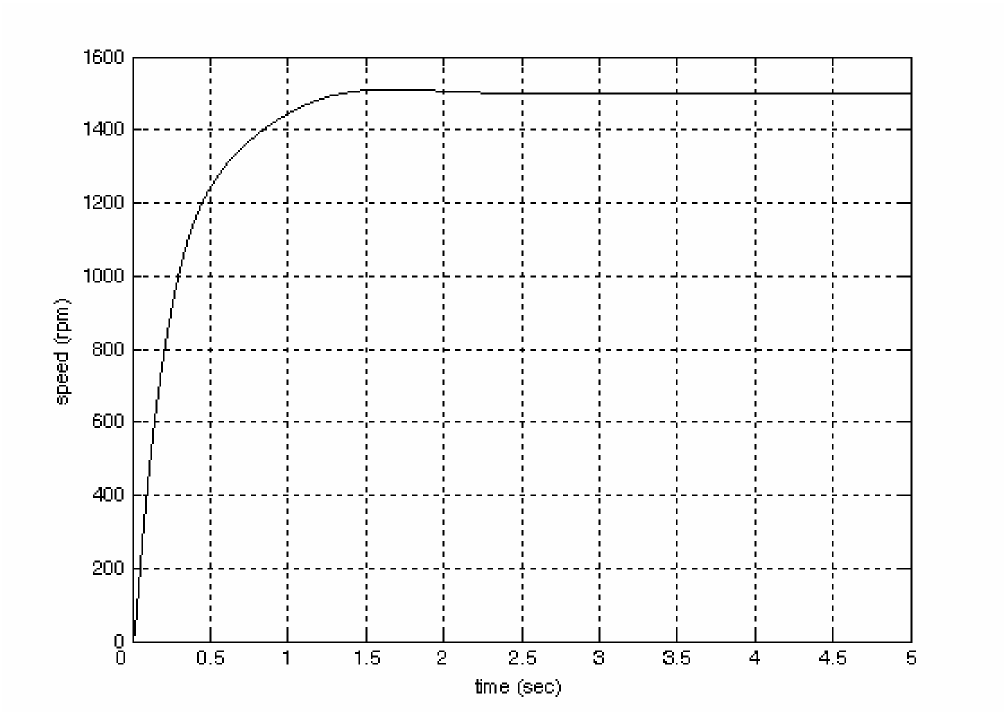


Figure 3.20 Speed response of the fuzzy controller, n_{set} = 1500 rpm

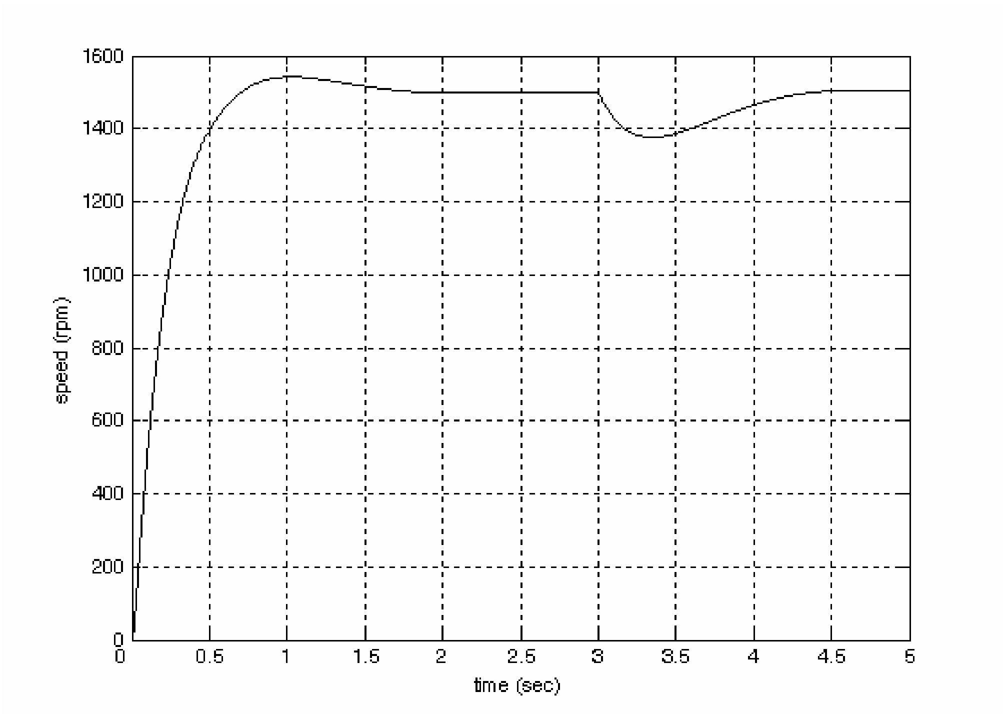


Figure 3.21 Speed response of fuzzy controller under load changing 3 Nm to 5 Nm

It is seen from the figures that the fuzzy logic controller controls the speed of the motor. The rise time seems slow but it must be noted that the motor parameters determines this time. When full armature current applied with no load on the motor, the motor needs approximately 1.75 sec to settle its rated speed.

4. NEURAL CONTROL FUNDAMENTALS

Neural control refers both to a methodology in which the controller itself is a neural network, and to a methodology in which controllers are designed based on a neural network model of the plant. These two basically different approaches for implementing neural networks in control are referred to as direct and indirect design methods.

Controlling a dynamical system means forcing it to behave in some desired way. Once the desired goal is specified, the design of the controller is dictated by the knowledge about the plant and the data available. Fuzzy control is a control method relying on perception based information expressed in fuzzy logic. This is the case where the available data is in the form of a collection of linguistic IF-THEN rules. In other words, fuzzy control is a mathematical method for implementing control strategies expressed in a natural language. This situation arises mostly in the control of complex systems, a situation that human operators handle well and for which natural language is an appropriate means for describing control strategies.

As its name indicates, neural control refers to another control method when available data are in the form of measurements (observed numerical data) of plant's behavior. This is the case where information is only in the form of system behavior, either of the real plant or of its simulated model, expressed as input – output measurements. In view of the generality of neural networks as function approximation devices, it is natural to use neural networks in control situations such as this.

Specifically, when mathematical models of the plant dynamics are not available, neural networks can provide a useful method for designing controllers, provided we have numerical information about the system behavior in the form of input-output data. In other words, a neural network can be used as a "black box" model for a plant. Also, controllers based on neural networks will benefit from neural networks learning capability that is suitable for adaptive control where controllers need to adapt to changing environment, such as time-variant systems. In practice, neural networks have proved to be most useful for time-invariant systems.

Basically, to build a neural network based controller that can force a plant to behave in some desirable way, we need to adjust its parameters from the observed errors that are the difference between the plant's outputs and desired outputs. Adjustment of the controller parameters will be done by propagating back these errors across the neural network structure. This is possible if the mathematical model of the plant is known. When the mathematical model of the plant is not known, we need to know at least an approximate model of the plant in order to do above. An approximate model of the plant is called identified model. When we use input – output data from the plant to train a neural network to provide an approximate model to the plant, we obtain a neural network identified model of the plant (Nguyen, 2002).

4.1 Neural Networks

It is well known that biological systems can perform complex tasks without recourse to explicit quantitative operations. In particular, biological organisms are capable of learning gradually over time. This learning capability reflects the ability of biological neurons to learn through exposure to external stimuli and generalize. Such properties of nervous systems make them attractive as computation models that can be designed to process complex data.

Neural networks, or more specifically, artificial neural networks are mathematical models inspired from our understanding of biological nervous system. They are attractive as computation devices that can accept a large number of inputs and learn solely from training samples. As mathematical models for biological nervous system, artificial neural networks are useful in establishing relationships between inputs and outputs of any kind of system.

Roughly speaking, a neural network is a collection of artificial neurons. An artificial neuron is a mathematical model of a biological neuron in its simplest form. From our understanding, biological neurons are viewed as elementary units for information processing in any nervous system. Without claiming its neurobiological validity, the mathematical model of an artificial neuron is based on the following theses:

- Neurons are the elementary units in a nervous system at which information processing occurs.
- Incoming information is in the form of signals that are passed between neurons through connection links.
- Each connection link has a proper weight that multiplies the signal transmitted.
- Each neuron has an internal action, depending on a bias or firing threshold resulting in an activation function being applied to the weighted sum of the input signals to produce an output signal.

Thus, when input signals $x_1, x_2, ..., x_n$ reach the neuron through connection links with associated weights $w_1, w_2, ..., w_n$, respectively, the resulting input to neuron, called the net input, is the weighted sum $\sum_{i=1}^{n} w_i x_i$. If the firing threshold is b and the activation function is f, then the output of that neuron is

$$y = f\left(\sum_{i=1}^{n} w_i x_i - b\right) \tag{4.1}$$

In the first computational model for artificial neurons, proposed by McCulloch and Pitts, 1943, outputs are binary, and the function f is the step function defined by

$$f(x) = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases} \tag{4.2}$$

This is depicted in Figure 4.1.

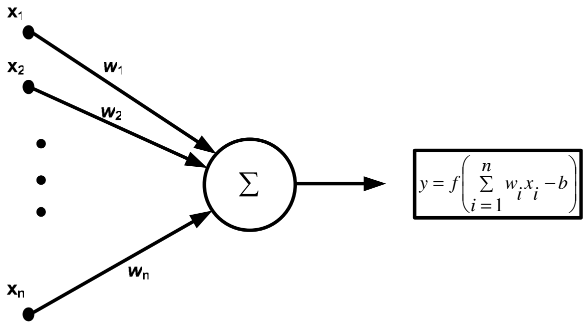


Figure 4.1 First model for artificial neuron

An artificial neuron is characterized by the parameters

$$\theta = (w_1, w_2, ..., w_n, b, f) \tag{4.3}$$

Feed forward neural networks, that is, information propagates only forward as indicated by the direction of arrows. Viewing artificial neurons as elementary units for information processing, we arrive at simple neural networks by considering several neurons at a time. The neural network in Figure 4.2 consists of an input layer, a layer of input nodes, and one output layer consisting of neurons. This is referred as to as a single layer neural network because the input layer is not a layer of neurons, that is, no computations occur at the input nodes. This single layer neural network is called a perceptron.

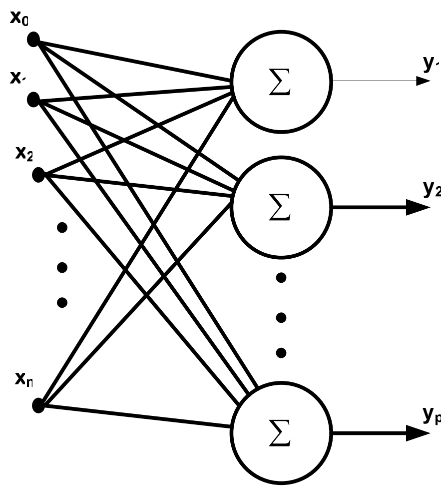


Figure 4.2 Perceptron

A multi layer neural network is a neural network with more than one layer of neurons. Note that the activation functions of the different neurons can be different. The neurons from one layer have weighted connections with neurons in the next layer, but no connections between neurons of the same layer. A two layer neural network is depicted in Figure 4.3. The input layer has *n+1* nodes, the middle layer, called the hidden layer, has *p* nodes, and the output layer has *m* nodes. This is called "n-p-m" neural network.

Neurons in each layer are somewhat similar. Neurons in the hidden layer are hidden in the sense that we cannot directly observe their output. From input patterns, we can only observe the output patterns from the output layer. A multi layer neural network can have more than one hidden layer.

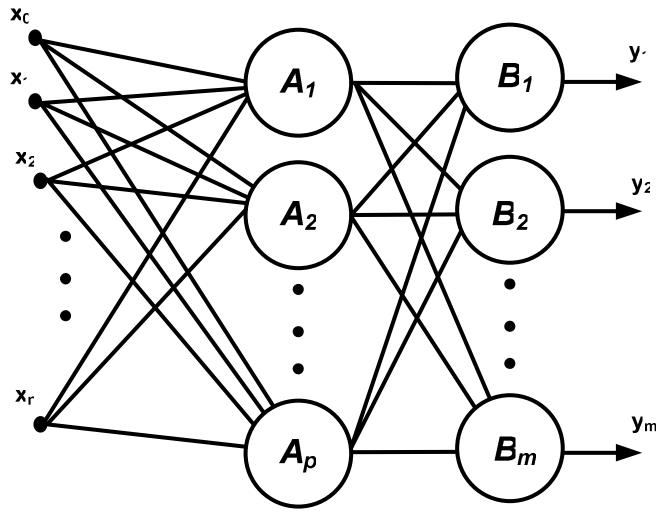


Figure 4.3 Two layer neural network

The two layer neural network depicted in Figure 4.3 is a typical multilayer perceptron, a multi layer neural network whose neurons perform the same function on inputs, usually a composite of the weighted sum and a differentiable nonlinear activation function, or transfer function, such as hyperbolic tangent function. Multi-layer perceptrons are the most commonly used neural network structures for a broad range of applications (Nguyen, 2002).

4.2 Learning

Artificial neural networks are simplified mathematical models of brain-like systems that function as parallel, distributed computing networks. As stated earlier, neural networks need to be taught or trained from examples before they can be put in use.

The learning problem is nothing more than choosing a function from a given class of functions according to some given criteria. If we address general relationships expressed as functions from R^n to R^m, then neural networks with smooth activation functions can approximate continuous functions with compact support. This universal approximation property of neural networks was proved using the Stone-Weierstrass theorem in the theory of approximation of functions. Specifically, all continuous functions whose domains are closed and bounded in R^n, that is, having compact support, can be approximated to any degree of accuracy by a neural network of one hidden layer with sigmoid or hyperbolic tangent activation functions. This theoretical result means this: It is possible to design an appropriate neural network to represent any continuous function having compact support. This is only an existence theorem, not a constructive one. The significance of the theorem is that it is reassuring, since most functions of practical interest are continuous.

To find an appropriate neural network to represent a given relationship, we need some information from a given relationship. In the context of learning, the information is provided in the form of a training set consisting of known input-output pairs. Learning from information of this type is referred to as supervised learning, or learning with a teacher.

Suppose we have a set T of training examples

$$T = \{(x^q, y^q), q = 1,..., N\} \tag{4.4}$$

$$x^q = (x_1{}^q, x_2{}^q,..., x_n{}^q) \in R^n \tag{4.5}$$

$$y^q = (y_1{}^q, y_2{}^q,..., y_m{}^q) \in R^m \tag{4.6}$$

and we wish to use this data to adjust the weights and biases of a neural network with *n* input nodes, and *m* output neurons with one hidden layer, for example. Then from a common sense viewpoint, we should compare the output

$$o^q = (o_1{}^q, o_2{}^q, ..., o_m{}^q) \tag{4.7}$$

from the input pattern x^q, with the corresponding target output y^q. This is the same as approximating a function from which the input-output pairs have been drawn.

The error correction idea is simple; a change in weights should be made or not according to the comparison of the actual output with the desired output. This idea is formulated in terms of a suitable overall performance measure. In a general sense, the learning problem may be stated as follows: Given a class of functions and a performance criterion, find a function in this class that optimizes the performance criterion on the basis of a set of training examples.

A control law is a function φ from an input space *X* to an output space *Y* that predicts the output from a given input. If we do not have a mathematical model, or even if we do but do not have an analytic method to find φ from it, we need to look for other alternatives.

Consider the situation where the information that will help us find φ is a set of desirable pairs (x_i, y_i), $i = 1,...,m$, where $y_i = \varphi(x_i)$. Then the obvious problem is to find a function φ_N that fits through these points in such a way that prediction is good, that is, for new x the fitted function φ_N will produce values $\varphi_N(x)$ close enough to $\varphi(x)$.

To carry out the program above, we need the specification of performance criteria and a way to construct φ_N that meets these criteria. The performance criterion for a good approximation is specified as an acceptable error of approximation. A neural network is a tool that can be used to achieve these two requirements simultaneously.

The neural network approach consists of two phases, creating a set of models and providing a mechanism for finding a good approximation. First, the architecture of a neural network provides a set *F* of functions from *X* to *Y*. The fact that *F* is a good set of models for φ is due to the universal approximation property of neural networks, that is, φ can be approximated by some element in *F*. Then the computer-oriented backpropagation algorithms of multi layer neural networks are designed to implement this approximation.

Instead of having desired pairs, or sample data, suppose we only have expert's knowledge as information to find φ. Then we are in the domain of applicability of fuzzy control. The set of models is created by expert's linguistic "if-then" rules. A similar universal approximation property of fuzzy systems exists. However, the search for a good approximation is done by tuning, that is, there is no systematic procedure to reach the approximate function φ_N. This drawback can be compensated by using neural networks in fuzzy control, leading to what is called neuro-fuzzy control, in which the formation used to construct φ is a fuzzy rule base represented in a neural network structure so that the systematic search by backpropagation algorithms of neural networks can be used to find a good approximation for φ. Of course, this requires experiments on the controlled system to extract samples. The advantage to this approach is that we have incorporated all available information in our process of constructing the control law. This process replaces the tuning process and is referred to as optimization of fuzzy rules. Indeed, when linguistic labels in fuzzy rules are modeled by parametric membership functions, the learning algorithm of neural networks will optimize the parameters in fuzzy rules, leading to the desired approximation of φ.

The delta rule is a learning algorithm for single layer neural networks. It is a precursor of the backpropagation algorithm for multi layer neural network.
The idea is to define a measure of the overall performance of a network, such as shown in Figure 4.4, then find a way to optimize that performance. Obviously, learning algorithms should change the weights so that the output o^q becomes more

and more similar to the target output y^q for all $q = 1,2,...,m$ when presenting the input x^q to the network.

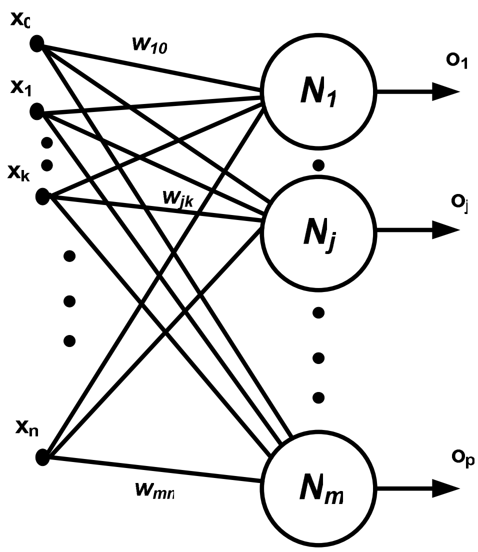


Figure 4.4 Network with weights w_{ij}

A suitable overall performance measure is

$$E = \sum_{q=1}^{N} E^q \tag{4.8}$$

where

$$E^q = \frac{1}{2}\sum_{i=1}^{m}(y_i{}^q - o_i{}^q)^2 \tag{4.9}$$

The goal is to minimize E with respect to the weights w_{ij} of the network.

Let w_{ij} denote the weight from the node j to the output neuron i. To use the gradient descent method for optimization, we need the w_{ij} 's to be differentiable. This boils down to requiring

$$o_i{}^q = f_i\left(\sum_{j=0}^{n} w_{ij} x_j{}^q\right) \tag{4.10}$$

to be differentiable. That is, the activation function f_i of the ith neuron should be chosen to be differentiable function. Note that the step function such as

$$f(x) = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases} \tag{4.11}$$

are not differentiable.

The sigmoid activation function, shown in Figure 4.5 is differentiable. Note that the step activation function models neurons that either fire or do not fire, while continuous activation function model neurons such as sigmoids that provide a gradual degree of firing, a fuzzy property, that is more realistic.

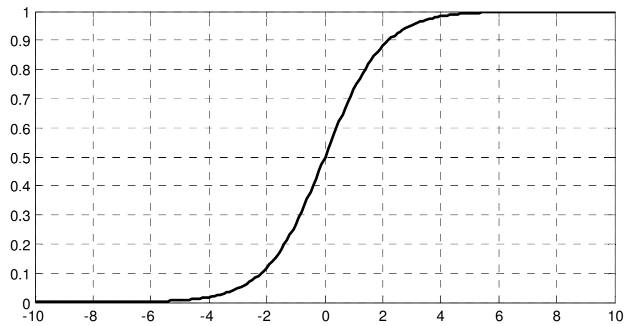


Figure 4.5 Sigmoid activation function

The error E is a function of the variables $w_{ij}, i = 1,2,...,m, j = 1,2,...,n$. Recall that the gradient ∇E of E at a point w with components w_{ij} is the vector of partial derivatives $\partial E / \partial w_{ij}$. Like the derivative of a function of one variable, the gradient always points to the uphill direction of the function E. The downhill direction of E at W is $-\nabla E$. Thus, to minimize E, we move proportionally to the negative of ∇E, leading to the updating of each weight w_{jk} as

$$w_{jk} \rightarrow w_{jk} + \Delta w_{jk} \tag{4.12}$$

where

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{4.13}$$

and $\eta > 0$ is a number called learning rate. We have,

$$\frac{\partial E}{\partial w_{ij}} = \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{ij}} \tag{4.14}$$

and

$$\frac{\partial E^q}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\frac{1}{2} \sum_{i=1}^{m} (y_i^{\ q} - o_i^{\ q})^2 \right) = (o_j^{\ q} - y_j^{\ q}) \frac{\partial}{\partial w_{ij}} f_j \left(\sum_{i=0}^{n} w_{ji} x_i^{\ q} \right) \tag{4.15}$$

$$\frac{\partial}{\partial w_{ij}} (o_i^{\ q} - y_i^{\ q}) = 0 \ \text{ for } \ i \neq j \tag{4.16}$$

$$o_j^{\ q} = f_j \left(\sum_{i=0}^{n} w_{ji} x_i^{\ q} \right) \tag{4.17}$$

$$\frac{\partial E^q}{\partial w_{jk}} = x_k^{\ q} (o_j^{\ q} - y_j^{\ q}) f_j{}' \left(\sum_{i=0}^{n} w_{ji} x_i^{\ q} \right) = \delta_j^{\ q} x_k^{\ q} \tag{4.18}$$

$$\delta_j^{\ q} = (o_j^{\ q} - y_j^{\ q}) f_j{}' \left(\sum_{i=0}^{n} w_{ji} x_i^{\ q} \right) \tag{4.19}$$

for the *j*th output neuron. Therefore,

$$\Delta w_{jk} = \sum_{q=1}^{N} \Delta^q w_{jk} = -\eta \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{jk}} = \sum_{q=1}^{N} -\eta \delta_j^{\ q} x_k^{\ q} \tag{4.20}$$

The delta rule leads to the search for a weight vector w^* such that the gradient of E at w^* is zero.

The backpropagation is a generalization of the delta rule. If we look closely at the delta rule for single layer neural networks, we realize that to update a weight a weight w_{ik} when the learning input pattern x^q is presented, we need

$$\Delta w_{ik}{}^{q} = -\eta \delta_i{}^{q} x_k{}^{q} \tag{4.21}$$

That is, we need to compute the function

$$\delta_i^{q} = (o_i{}^{q} - y_i{}^{q}) f'\left(\sum_{j=0}^{n} w_{ij} x_j^{q} \right) \tag{4.22}$$

and this is possible since we have at our disposal the value y_i^q that is known to us as the target output for the output node *i*.

Consider, for simplicity, the two layer n-m-p neural network depicted in Figure 4.6. It seems that we cannot update a weight like w_{ik} on the link connecting the *k*th input node in the input layer to the *i*th hidden neuron in the hidden layer, since we are not given the target pattern of the *i*th neuron from the input x^q. Note that patterns y^q are target patterns of neurons in the output layer, and not the hidden layers. Thus, when we look at the errors $(o_j^q - y_j^q)^2$ at the output layer, we cannot detect which hidden neurons are responsible.

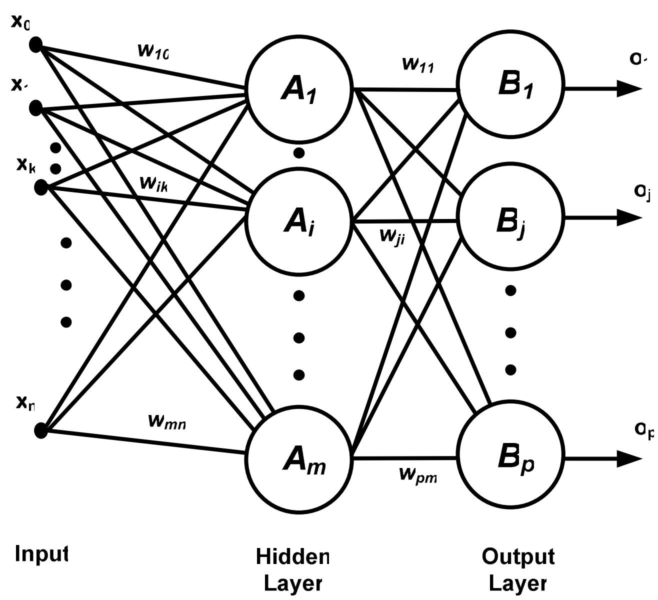


Figure 4.6 n-m-p neural network

It turns out that to update these weights, it suffices to be able to compute $\partial E / \partial o_i$, the partial derivative of the global error E with respect to output o_i, $i = 1,2,..., p$. For differentiable activation functions, the gradient descent strategy can still be applied to find the network's weight configuration w^* that minimizes the error *E*. We will write down the updating formulas for weights in a two layer neural network. The generalization to more than two layers is just a matter of notation. Let v_{ji} be the weight of the link connecting the hidden neuron *i* to the output neuron *j*, and w_{ik} the weight of the link connecting the input node *k* to the hidden neuron *i.*

First, the updating rule for the v_{ji} is the same as in the delta rule, by viewing now the hidden layer as an input layer. For output neuron *j* and hidden neuron *i,* the weight v_{ji} is updated using

$$\Delta v_{ji} = \sum_{j=1}^{N} \Delta^q v_{ji} = -\eta \sum_{j=1}^{N} \frac{\partial E^q}{\partial v_{ji}} = \sum_{j=1}^{N} (-\eta \delta_j^q z_i^q) \tag{4.23}$$

where z_i^q is the net input to the hidden neuron *i,*

$$z_i^q = f_i\left(\sum_{k=0}^{n} w_{ik} x_k^q \right) \tag{4.24}$$

$$\delta_j^q = (o_j^q - y_j^q) f_i'\left(\sum_{k=1}^{m} v_{jk} z_k^q \right) \tag{4.25}$$

For a hidden neuron *i* and input node *k,* the weight w_{ik} is updated as follows:

$$\Delta^q w_{ik} = -\eta \frac{\partial E^q}{\partial w_{ik}} \tag{4.26}$$

$$\frac{\partial E^q}{\partial w_{ik}} = \frac{\partial E^q}{\partial o_i^q} \frac{\partial o_i^q}{\partial w_{ik}} \tag{4.27}$$

where o_i^q is the output of the hidden neuron *i*,

$$o_i^q = f_i\left(\sum_{l=0}^{n} w_{il} x_l^q\right) = z_i^q \tag{4.28}$$

$$\frac{\partial o_i^q}{\partial w_{ik}} = f_i'\left(\sum_{l=0}^{n} w_{il} x_l^q\right) x_k^q \tag{4.29}$$

Let $\delta_i^q = \dfrac{\partial E^q}{\partial o_i^q}$, the hidden layer. Observe that,

$$\delta_i^q = \frac{\partial E^q}{\partial o_i^q} = \sum_{j=1}^{p} \frac{\partial E^q}{\partial o_j^q} \frac{\partial o_j^q}{\partial o_i^q} \tag{4.30}$$

where *j* is the output layer. The $\partial E^q / \partial o_j^q$ are known from previous calculations using the delta rule.

$$o_i^q = f_j'\left(\sum_{l=1}^{m} v_{il} z_l^q\right) v_{ji} \tag{4.31}$$

Thus the δ_i^q, for hidden neurons *i*, are computed from the already known values of δ_j^q for all *j* in the output layer.

Because of this fact, this generalized delta rule is called backpropagation algorithm. First feed the input patterns x^q forward to reach the output layer and then calculate the δ_j^q s for all output neurons *j*. Next propagate these δ_j^q s backward to the layer below in order to calculate the δ_i^q s for all neurons *i* of that layer.

There are two approaches for training.

The batch approach: The weights w_{ik} are changed according to

$$\Delta w_{ik} = -\eta \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{ik}} \tag{4.32}$$

after all N training patterns are presented to the neural network.

The incremental approach: Change the weights w_{ik} after every training pattern q is presented to the neural network, that is, using,

$$\Delta^q w_{ik} = -\eta \frac{\partial E^q}{\partial w_{ik}} \tag{4.33}$$

to update w_{ik}.

Note that the batch update Δw_{ik} is just the sum of the incremental updates $\Delta^q w_{ik}$, $q = 1,2,...,N$.

In the backpropagation algorithm, we start out by the calculation of the δ value in the output layer. Then we propagate the error vector backward from the output layer towards the input terminals. When all the weights are updated with $\Delta^q w_{ik}$, the next training pattern x^q is presented, and the procedure starts again. The stopping criterion can be either a threshold or the error function *E.*

In summary, the backpropagation algorithm can be described by the following steps.

- Initialize weights w_{ik} with small random values, and select the learning rate η.
- Apply the pattern x^q to the input layer.
- Propagate x^q forward from the input layer to the output layer using

$$o_i = f_i\left(\sum_{k=0}^{p} w_{ik} o_k\right) \tag{4.34}$$

- Calculate the error $E^q(w)$ on the output layer using

$$E^q(w) = \frac{1}{2}\sum_{i=1}^{p}(o_i^q - y_i^q)^2 \tag{4.35}$$

- Calculate the δ values of the output layer using

$$\delta_i^q = f_i'\left(\sum_{k=1}^{n} v_{ik} z_k\right)(o_i^q - y_i^q) \tag{4.36}$$

- Calculate the δ values of the hidden layer by propagating the δ values backward, that is,

$$\delta_i^q = f_i'\left(\sum_{k=0}^{n} w_{ik} x_k^q\right)\sum_{j=1}^{p} v_{ij}\delta_j^q \tag{4.37}$$

- Use

$$\Delta^q w_{ik} = -\eta\delta_i^q o_k^q \tag{4.38}$$

for all w_{ik} of the neural network.

- $q \to q+1$ and go to step 2

Both batch and incremental approach can be applied for updating the weight configuration w. The relative effectiveness of the two approaches depends on the problem, but the incremental approach seems superior in most cases, especially for every regular or redundant training sets.

The algorithm leads to w^*, such that

$$\Delta(E)(w^*) = 0 \tag{4.39}$$

which could be a local minimum (Nguyen, 2002).

4.3 Neural Network in Direct Neural Control

Direct design means that a neural network directly implements the controller, that is, the controller is a neural network as shown in Figure 4.7. The network must be trained as the controller according to some criteria, using either numerical input-output data or a mathematical model of the system.

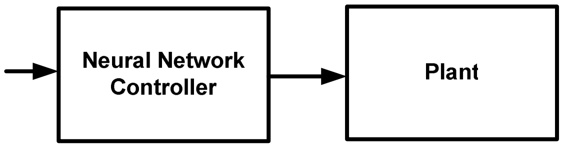


Figure 4.7 Direct design

There are several types of neural network architectures. Multi layer perceptron neural networks are composed of configurations of simple perceptrons in a hierarchical structure forming a feedforward network. They have one or more hidden layers of perceptrons between the input and output layers. It is permissible to have any prior layer nodes connected to subsequent layer nodes via a corresponding set of weights. Different learning algorithms can be used for MLPs, but the most common ones have been the delta rule and error backpropagation algorithms. These algorithms do work fairly well but they tend to be slow. Faster and more efficient algorithms have been developed.

4.4 Neural Networks Using MATLAB

MATLAB/Simulink provides an excellent environment for both designing and simulation a control system. Several toolboxes such as Control System Toolbox, Fuzzy Logic Toolbox, Neural Network Toolbox allows the users to implement their design in computer environment easily. Fuzzy Logic and Neural Network toolboxes are used in the studies in this thesis. The neural network toolbox and its neural network implementation commands are introduced here.

4.4.1 Neuron Model

An elementary neuron with R inputs is shown in Figure 4.8. Each input is weighted with an appropriate w. The sum of weighted inputs and the bias forms the input to the transfer function f. Neurons may use any differentiable transfer function f to generate their output.

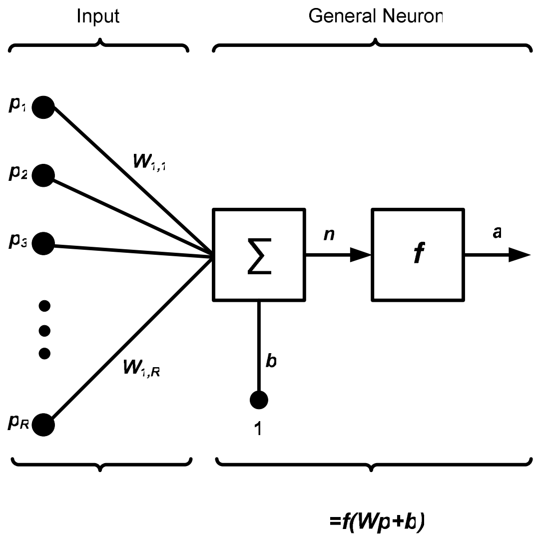


Figure 4.8 Basic neuron model used in MATLAB

Multilayer networks often use the log sigmoid transfer function *logsig*. The function *logsig* generates outputs between 0 and 1 as the neuron's net input goes from negative to positive infinity. Alternatively, multilayer networks may use the tan sigmoid transfer function *tansig*. Occasionally, the linear transfer function *purelin* is used in backpropagation networks. Figure 4.9 shows the *logsig*, *tansig*, and *purelin* transfer functions.

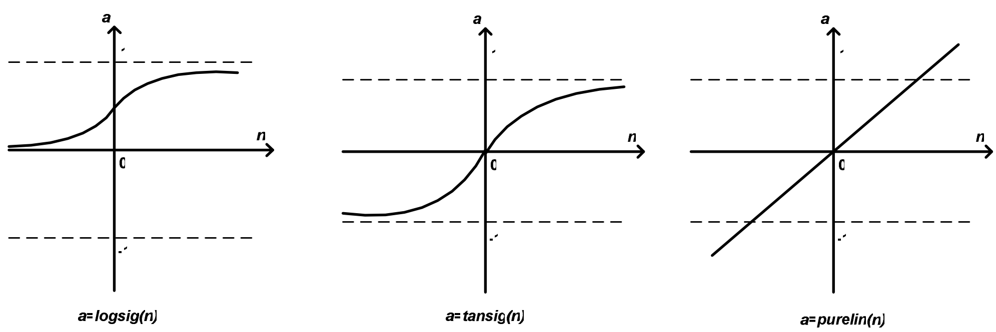


Figure 4.9 *logsig*, *tansig*, *purelin* transfer functions

If the last layer of a multilayer network has sigmoid neurons, then the outputs of the network are limited to a small range. If linear output neurons are used the network outputs can take on any value. In backpropagation it is important to be able to calculate the derivatives of any transfer function used (Demuth, *et al*, 1998).

4.4.2 Feed Forward Network

A single layer network of *S* logsig neurons having *R* inputs is shown in Figure 4.10 in full detail on the left and layer diagram on the right.

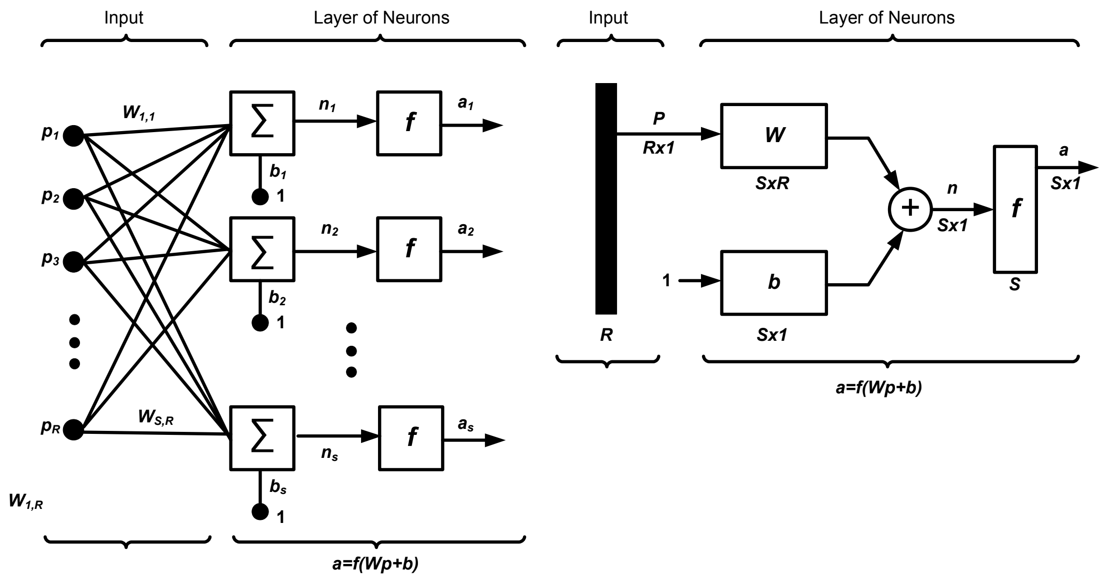


Figure 4.10 Single layer neural network

Feed forward network often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between the input and output vectors. The linear output layer lets the network produce values outside of -1 and 1. On the other hand, if it is desirable to constrain the outputs of a network then the output layer should use a sigmoid transfer function.

For multiple layer networks, the number of the layers is used to determine the superscript of the weight matrices. The appropriate notation is used in the two layer tansig/purelin network shown in Figure 4.11.

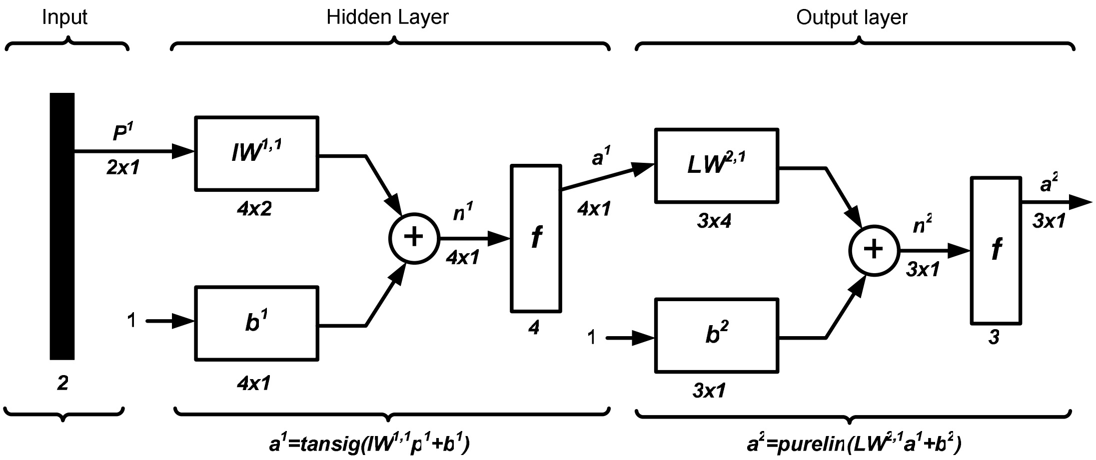


Figure 4.11 Two layer neural network example

This network can be used as a general function approximator. It can approximate any function with a finite number of discontinuities, arbitrary well, given sufficient neurons in the hidden layer.

MATLAB creates a network object to realize a neural network. The first step in training a feed forward network is to create the network object. The function *newff* creates a trainable feed forward network. It requires four inputs and returns the network object. The first input is an *R* by 2 matrix of minimum and maximum values of each of the *R* elements of the input vector. The second input is an array containing the sizes of each layer. The third input is a cell array containing the names of the transfer functions to be used in each layer. The final input contains the name of the training function to be used.

The function *sim* simulates a network, *sim* takes the network input *p,* and the network object, and returns the network outputs.

Neural networks needs to know what to do; training the network answers this question. The training process requires a set of examples of proper network behavior, network inputs *p* and target outputs *t.* During training the weights and biases of the network are iteratively adjusted to minimize the network performance function, that is mean square error by default, is the average squared error between the networks outputs *a* and the target outputs *t.*

There are several different training algorithms for feed forward networks. All of these algorithms use the gradient of the performance function to determine how to adjust the weights to minimize the performance. The gradient is determined using backpropagation (Demuth, *et al*, 1998).

4.4.3 Training Algorithms in MATLAB

- Gradient Descent (learngd)

For the basic steepest descent algorithm, the weights and biases are moved in the direction of the negative gradient of the performance function. The function *learngd* has one learning parameter associated with it, learning rate *lr*. The changes to the weights and biases of the network are obtained by multiplying the learning rate times the negative of the gradient.

- Gradient Descent with Momentum (learngdm)

In addition to *learngd*, there is another incremental learning algorithm for feed forward networks that often provides faster convergence, *learngdm*, steepest descent with momentum. Momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a low pass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network may get stuck in a local minimum. With momentum a network can slide through such a minimum.

- Batch Training (train)

The alternative to incremental training is batch training. In batch mode the weights and biases of the network are updated only after the entire training set has been applied to the network. The gradients calculated at each training example are added together to determine the change in the weights and biases.

- Batch Gradient Descent (traingd)

The batching equivalent of the incremental function *learngd* is *traingd*, which implements the batching form of the standard steepest descent training function. The weights and biases are updated in the direction of negative gradient of the performance function.

There are seven training parameters associated with the function *traingd*, *epochs, show, goal, lr, time, min_grad* and *max_fail*. The *lr* is multiplied times the negative of the gradient to determine the changes to the weights and biases.

- Batch Gradient Descent with Momentum (traingdm)

The batch form of gradient descent with momentum is invoked using the training function *traingdm*.

- Levenberg - Marquardt (trainlm)

The Levenberg - Marquardt algorithm was designed to approach second order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares, then the Hessian matrix can be approximated as

$$H = J^T J \tag{4.40}$$

and the gradient can be computed as

$$g = J^T e \tag{4.41}$$

where J is the Jacobian matrix, which contains first derivatives of the network errors with respect to the weights and biases, and e is a vector of network errors. The Jacobian matrix can be computed through a standard backpropagation technique that is much less complex than computing the Hessian matrix.

The Levenberg-Marquardt algorithm uses this approximation to the Hessian matrix in the following Newton like update

$$x_{k+1} = x_k - \left[J^T J + \mu I\right]^{-1} J^T e \tag{4.42}$$

When the scalar μ is zero, this is just Newton's method, using approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. Newton's method is faster and more accurate near an error minimum, so the aim is to shift towards Newton's method as quickly as possible. Thus, μ is decreased after each successful step and is increased only when a tentative step would increase the performance function. In this way, the performance function will always be reduced at each iteration of the algorithm.

4.5 Case Study

DC motor system is used. It is desired to have a neural network controller that will make the speed regulation for the motor.

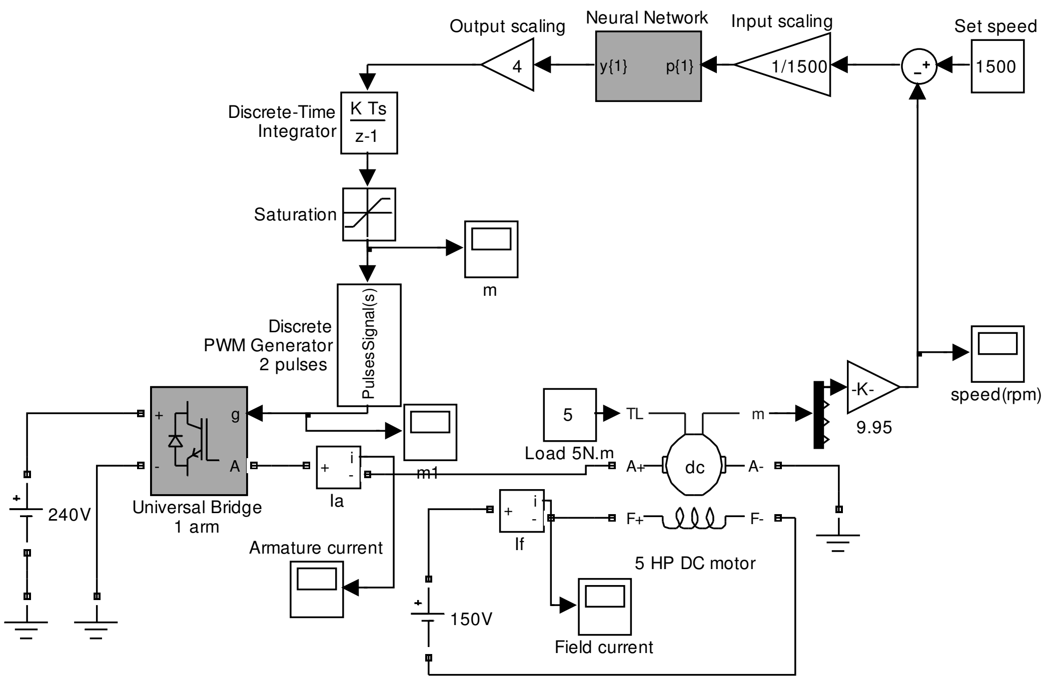


Figure 4.12 DC motor with neural network controller

Table 4.1 Training data for the neural network

Num.of Pairs	*P*	*T*		*Num.of Pairs*	*P*	*T*
1	1.000	0.300		28	-0.010	0.000
2	0.900	0.275		29	-0.020	0.000
3	0.800	0.250		30	-0.030	0.000
4	0.700	0.225		31	-0.040	-0.010
5	0.600	0.200		32	-0.050	-0.020
6	0.500	0.175		33	-0.060	-0.030
7	0.400	0.150		34	-0.070	-0.035
8	0.300	0.140		35	-0.080	-0.040
9	0.275	0.130		36	-0.090	-0.045
10	0.250	0.120		37	-0.100	-0.050
11	0.225	0.110		38	-0.120	-0.060
12	0.200	0.100		39	-0.140	-0.070
13	0.180	0.090		40	-0.160	-0.080
14	0.160	0.080		41	-0.180	-0.090
15	0.140	0.070		42	-0.200	-0.100
16	0.120	0.060		43	-0.225	-0.110
17	0.100	0.050		44	-0.250	-0.120
18	0.090	0.045		45	-0.275	-0.130
19	0.080	0.040		46	-0.300	-0.140
20	0.070	0.035		47	-0.400	-0.150
21	0.060	0.030		48	-0.500	-0.175
22	0.050	0.020		49	-0.600	-0.200
23	0.040	0.010		50	-0.700	-0.225
24	0.030	0.000		51	-0.800	-0.250
25	0.020	0.000		52	-0.900	-0.275
26	0.010	0.000		53	-1.000	-0.300
27	0.000	0.000				

The design procedure of the neural controller starts with obtaining an appropriate training data. The input-output relation, training data, is obtained from the previous examples blended by the knowledge of the designer about system dynamics. Then a network object is created.

The selection of activation functions depends on the designer. Then important training parameters are set. Several training algorithms are attempted to achieve the minimum network error and number of neurons in the hidden layer is also changed to show the effect on the performance of the network. The simulation model of the neural network controlled DC motor is shown in Figure 4.12.

The training data used in the neural network design is given in Table 4.1 where number of training pairs is arranged in the first column and the *P* is the input vector, and *T* is the target vector.

newff function of MATLAB is used to generate a network object in MATLAB workspace. This function has the form given in Eq.(4.43) and has several parameters;

net =newff(PR, [S1 S2 … SNl], {TF1, TF2,…,TFNl}, BTF) (4.43)

The parameters of the *newff* are;
PR is the *Rx2* matrix of minimum and maximum values of *R* input elements,
Si is the size of the *i*th layer,
TFi is the activation function of the *i*th layer,
BTF is the network training function.
The following command is used to generate the network object;

net = newff([-1 1], [14, 1],{'tansig', 'purelin'}, 'trainlm') (4.44)

[-1 1] determines the minimum and maximum values of input. As it can easily be obtained that the input may take minimum -1 and maximum 1 value.
[14 1] is the sizes of the layers. The network object that is created has two layers; there are 14 neurons in the first hidden layer and 1 neuron in the output layer.

{'tansig', 'purelin'} part of the command states that *tansig* activation functions will be used in the hidden layer and *purelin* activation functions will be used in the output layer.
'trainlm' is the training algorithm that will be used, Levenberg-Marquardt.

The generated network object has parameters that must be determined before starting to train the neural network. The network does not know about how to respond to the input, because it has not been trained yet.

$$net.trainParam.epochs = 500 \tag{4.45}$$

$$net.trainParam.lr = 0.01 \tag{4.46}$$

$$net.trainParam.goal = 0.001 \tag{4.47}$$

Epochs is the number of iterations that will be made when trying to decrease the network error to fit the given input output relation.
lr is the learning rate which is a constant coefficient that is used as a multiplier of gradient in training algorithm.

Goal determines maximum allowable network error. Above parameters state that the network has to adjust it's weights and biases in 500 epochs to minimize it's error less than 0.001 with a learning rate of 0.01.

After finishing parameter selection, the network is ready to be trained. Levenberg-Marquardt algorithm is used for training. The following MATLAB command starts training process of the network.

$$net1 = train(net, P, T) \tag{4.48}$$

where P is the input vector, T is the target output vector and *net* is the adjusted network object above. *net1* is the new network object that is trained. The desired neural network is ready to be used now.

Table 4.2 shows the network errors for different training algorithms and different learning rates. It can be easily observed that *trainlm* has the best performance. There are 10 neurons in hidden layer and 1 neuron in the output layer.

Table 4.2 Network errors after training with different training algorithm

Epochs = 500						
lr	**0.01**	**0.02**	**0.05**	**0.10**	**0.20**	**Algorithm**
	0.0528934	0.0271390	0.0064120	0.001118	NAN	traingd
	0.0196794	0.0126939	0.0047247	0.001606	0.000633	traingdm
	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	trainlm
Epochs = 1500						
lr	**0.01**	**0.02**	**0.05**	**0.10**	**0.20**	**Algorithm**
	0.009521	0.004326	0.001037	0.000237	NAN	traingd
	0.008908	0.003762	0.000800	0.000211	0.000217	traingdm
	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	trainlm
Epochs = 3000						
lr	**0.01**	**0.02**	**0.05**	**0.10**	**0.20**	**Algorithm**
	0.004327	0.001544	0.000236	$6.0544.\ 10^{-5}$	NAN	traingd
	0.003787	0.001214	0.000191	$7.7645.\ 10^{-5}$	0.000173	traingdm
	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	$3.1454.10^{-6}$	trainlm

The number of neurons in the hidden layer also determines the network performance, if it is desired to have smaller network errors; one may consider increasing the number of neurons.

The following commands are used to generate the neural network controller employed in the simulation model given Figure 4.12.

net = newff([-1 1], [14 1], {'tansig', 'purelin'}, 'trainlm') (4.49)

net.trainParam.epochs = 1500 (4.50)

net.trainParam.goal = 1e-6 (4.51)

net.trainParam.lr = 0.05 (4.52)

net1 = train(net, P, T) (4.53)

gensim(net1, 0.00001) (4.54)

Eq.(4.54) creates a new simulation model under Simulink with a sample time of 0.00001 seconds. The resulting neural network has 14 neurons in the hidden layer with *tansig* activation functions and 1 neuron with *purelin* activation function. The network is trained for 1500 times with a learning rate of 0.05 with a 1e-6 error goal. Resulting neural network is used as a controller in the DC motor control problem. It has one input that is DC motor speed error. Output of the neural controller is the change of modulation index as in fuzzy controller case.

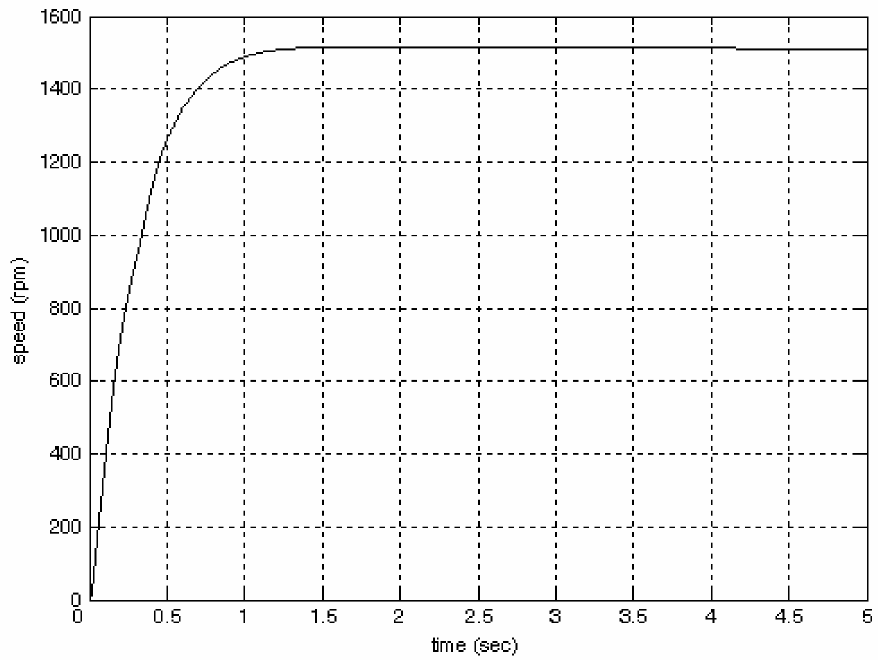


Figure 4.13 Speed response of the neural controller where set speed is 1500 rpm

The simulation results for the designed controller are given in the following figures, Figure 4.13, 4.14 and 4.15.

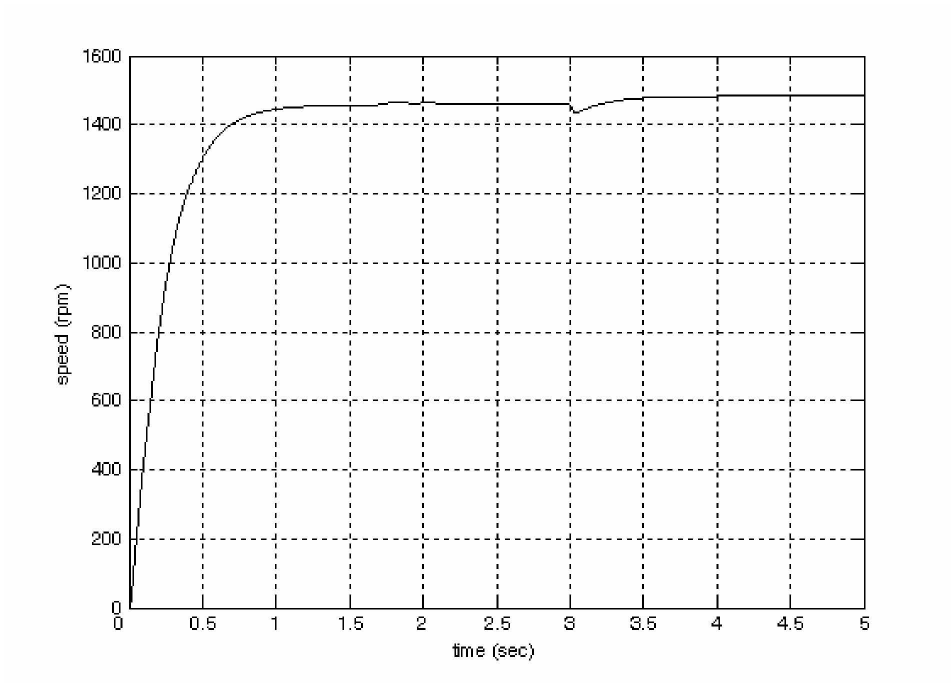


Figure 4.14 When the load increased from 3Nm to 5Nm at time t= 3sec

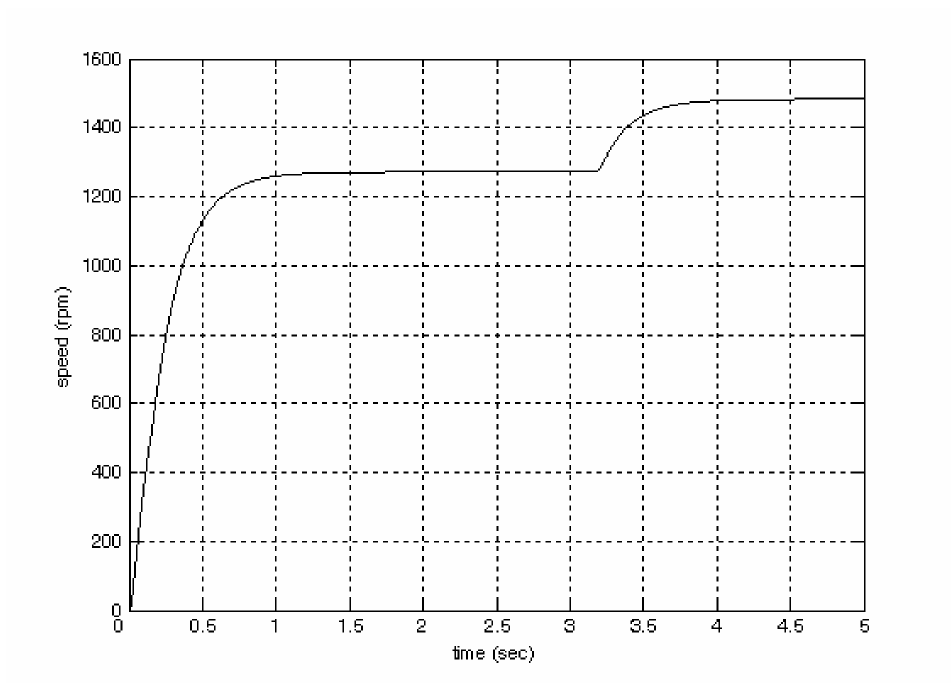


Figure 4.15 Speed response when speed is increased from 1300 rpm to 1500 rpm at time t = 3 sec.

The simulation results show that the neural controller can easily control the DC motor system. It must be noted that, the controller input and output gains have a large effect on the controller performance. And the controller itself can not change its parameters. The controller gain parameters have to be adjusted before putting motor system into the work.

5. APPLICATIONS OF FUZZY AND NEURAL CONTROL

5.1 Fuzzy Control Application

A hot liquid that is a product stream is coming from a poorly mixed stirred tank reactor. The reaction is exothermic so that the fluid leaving the reactor can get very hot. This fluid is closed with cooling water flowing through a counter-current heat exchanger, with the hot fluid on the shell side. The situation is depicted in Figure 5.1. The hot fluid temperature needs to be maintained at about 110 ºF, because it is used in another process. There is some leeway. The process that is accepting the new hot fluid can easily handle a fluctuation of ± 5 ºF. Differences much larger than this start to become a problem. The cooling water comes from a cooling tower where the temperature of the cold water is maintained at 85ºF. Since this temperature is constant, the only way of controlling the hot fluid temperature is to allow more or less cooling water to flow through the heat exchanger. The flow is controlled by opening and closing a control valve. The amount of flow through the valve is relative to the valve stem position. A stem position of 1 represents fully opened and a position of 0 equals fully closed.

$$New\ valve\ position = Old\ valve\ position + f * Range \qquad (5.1)$$

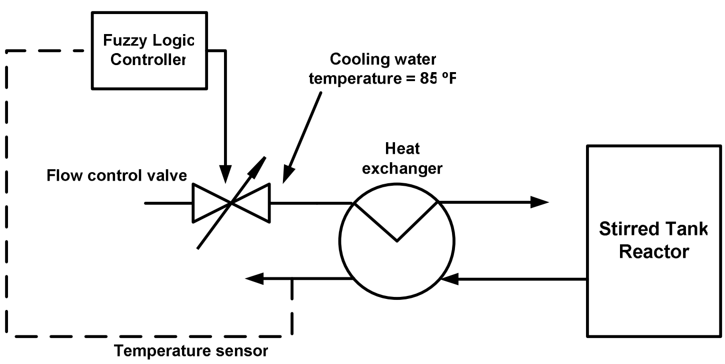


Figure 5.1 Fluid cooling example

The valve fractional change, *f,* is a fraction defined by the output membership functions of a *Range*. If the fraction, *f*, is greater than 0, the *Range* is defined as "full open" (1) minus the current valve position. If *f* is less than 0, the *Range* is defined as the current valve position. The control action is;

A single input single output controller will be sufficient for this case (Ross, 2005). The temperature error is the input for the controller. And the output is the *f* defined above. The membership functions for the fuzzy controller have the following form. There are five membership functions for input (error) and the output (f).

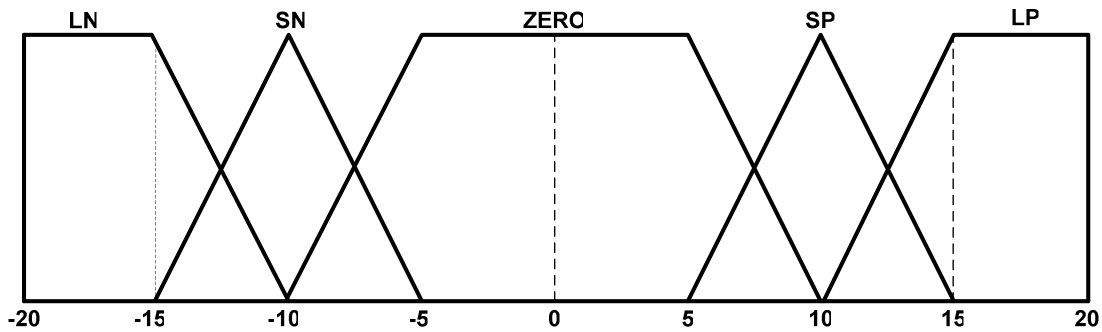


Figure 5.2 Membership functions for the input (error)

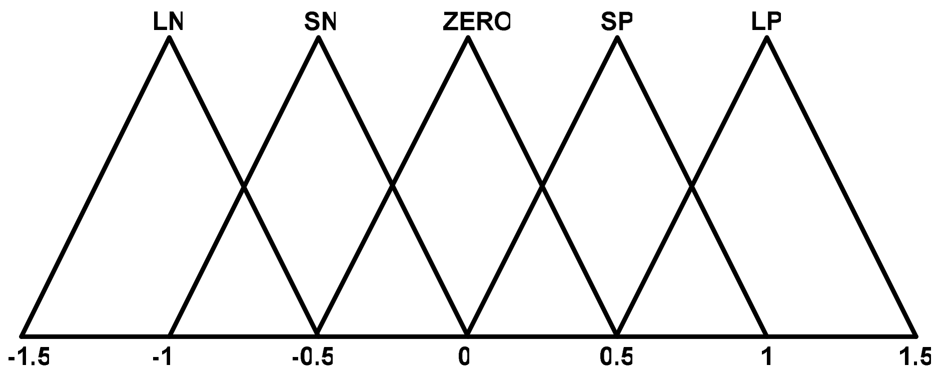


Figure 5.3 Membership functions for output (*f*)

The rule base is very simple for the controller.

1- IF error is (LN) THEN *f* is (LN)
2- IF error is (SN) THEN *f* is (SN)
3- IF error is (ZERO) THEN *f* is (ZERO)

4- IF error is (SP) THEN *f* is (SP)

5- IF error is (LP) THEN *f* is (LP)

The fuzzy controller does not care about the change of temperature. To have a more smooth response, it will be helpful to add a second input, error rate that is change of error.The membership functions for inputs error and error rate (erate) and the output (*f*) is given in Figure 5.4, Figure 5.5 and Figure 5.6.

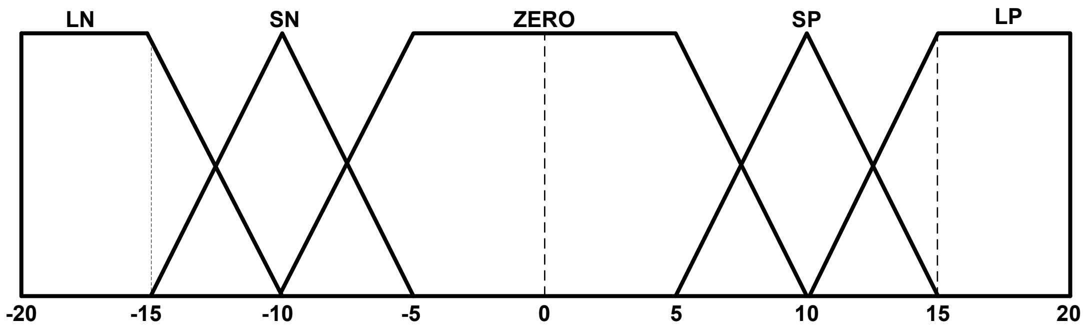


Figure 5.4 Membership functions for input error

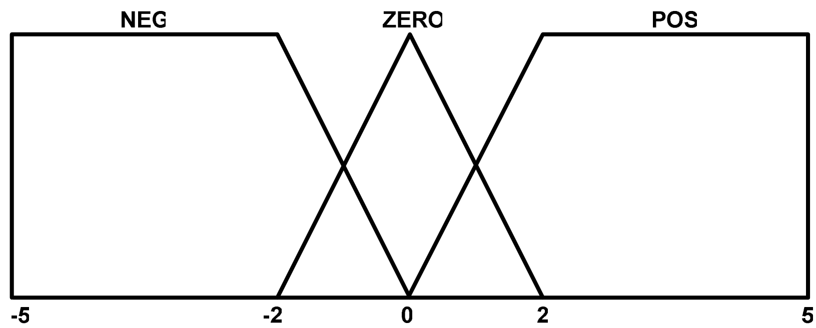


Figure 5.5 Membership functions for input erate

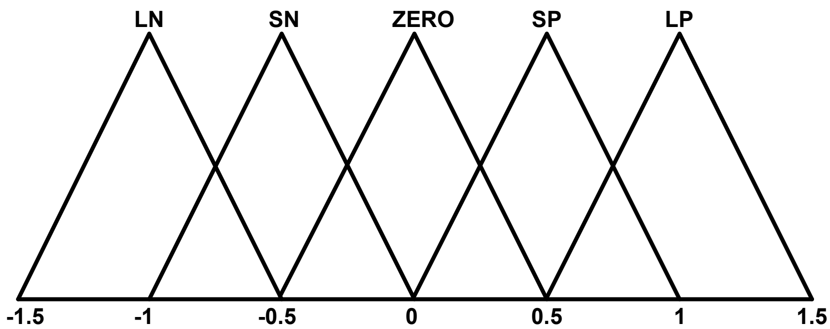


Figure 5.6 Membership functions for output *f*

Changing number of inputs is directly affecting the number of rules. The rule base of the fuzzy controller has to be upgraded to include the new input variable *erate*. Table 5.1 summarizes the rule base.

1- IF error is (LN) AND erate is (NEG) THEN *f* is (LN)
2- IF error is (SN) AND erate is (NEG) THEN *f* is (LN)
3- IF error is (ZERO) AND erate is (NEG) THEN *f* is (SN)
4- IF error is (SP) AND erate is (NEG) THEN *f* is (SN)
5- IF error is (LP) AND erate is (NEG) THEN *f* is (ZERO)
6- IF error is (LN) AND erate is (ZERO) THEN *f* is (LN)
7- IF error is (SN) AND erate is (ZERO) THEN *f* is (SN)
8- IF error is (ZERO) AND erate is (ZERO) THEN *f* is (ZERO)
9- IF error is (SP) AND erate is (ZERO) THEN *f* is (SP)
10- IF error is (LP) AND erate is (ZERO) THEN *f* is (LP)
11- IF error is (LN) AND erate is (POS) THEN *f* is (ZERO)
12- IF error is (SN) AND erate is (POS) THEN *f* is (SP)
13- IF error is (ZERO) AND erate is (POS) THEN *f* is (SP)
14- IF error is (SP) AND erate is (POS) THEN *f* is (LP)
15- IF error is (LP) AND erate is (POS) THEN *f* is (LP)

Table 5.1 Improved fuzzy controller rule base

erate \ error	LN	SN	ZERO	SP	LP
NEG	LN	LN	SN	SN	ZERO
ZERO	LN	SN	ZERO	SP	LP
POS	ZERO	SP	SP	LP	LP

The fuzzy logic system is selected to be easy to demonstrate a C++ implementation of the fuzzy controller. The following C++ program is a simple fuzzy logic controller. It must be noted that when the number of membership functions and so the rules increase, performance of the controller in real time applications are limited by hardware that will run the code.

Listing 5.1 C++ implementation of fuzzy controller

```
#include "stdio.h"

#ifndef MAX
# define MAX(x,y) ( (x) > (y) ? (x) : (y) )
#endif
#ifndef MIN
# define MIN(x,y) ( (x) < (y) ? (x) : (y) )
#endif

double infer[5][3]; // Array that stores the membership values.

// Membership function parameters.
double in_LN[4] = {-20,-20,-15,-10};
double in_SN[3] = {-15,-10,-5};
double in_ZERO[4] = {-10,-5,5,10};
double in_SP[3] = {5,10,15};
double in_LP[4] = {10,15,20,20};
double out_LN[3] = {-1.5,-1,-0.5};
double out_SN[3] = {-1,-0.5,0};
double out_ZERO[3] = {-0.5,0,0.5};
double out_SP[3] = {0,0.5,1};
double out_LP[3] = {0.5,1,1.5};
double ine_NEG[4] = {-5,-5,-2,0};
double ine_ZERO[3] = {-2,0,2};
double ine_POS[4] = {0,2,5,5};

// Function that clears the membership values for every
// sampling interval.

void clear()
{
      int i,j;
      for (i = 0;i<5;i++)
            for (j=0;j<3;j++)
            infer[i][j]=0;
}

// Definition for triangle membership function. It has two
// input parameters, x is the crisp input, params[] determines
// the a, b, c. a is the left hand point of the triangle,
// b is the top center point of it and c is the right hand point
// of the triangle.

static double triangleMf(double x, double params[])
{
      double a = params[0], b = params[1], c = params[2];

      if (a>b)
```

```
      printf("Parameter a cannot be greater than parameter b");
      if (b>c)
      printf("Parameter b cannot be greater than parameter c");

      if (a == b && b == c)
            return(x == a);
      if (a == b)
            return((c-x)/(c-b)*(b<=x)*(x<=c));
      if (b == c)
            return((x-a)/(b-a)*(a<=x)*(x<=b));

      return(MAX(MIN((x-a)/(b-a), (c-x)/(c-b)), 0));
}

// Definition for trapezoid membership function. It has two
// input parameters, x is the crisp input, params[] determines
// the a, b, c, d. a is the left hand point of the trapezoid,
// b is the top left point, c is the top right point and
// d is the right hand point of the trapezoid.

static double trapezoidMf(double x, double params[])
{
      double a = params[0], b = params[1];
      double c = params[2], d = params[3];
      double y1 = 0, y2 = 0;

      if (a>b) {
      printf("Parameter a cannot be greater than parameter b");
      }
      if (b>c) {
      printf("Parameter b cannot be greater than parameter c");
      }
      if (c>d) {
      printf("Parameter c cannot be greater than parameter d");
      }

      if (b <= x)
            y1 = 1;
      else if (x < a)
            y1 = 0;
      else if (a != b)
            y1 = (x-a)/(b-a);

      if (x <= c)
            y2 = 1;
      else if (d < x)
            y2 = 0;
      else if (c != d)
            y2 = (d-x)/(d-c);

      return(MIN(y1, y2));

}
```

```
// Aggregation method is selected to be MAX.
// Determines the MAX of the output membership functions
// for each output membership.

double output(int mfno)
{
int i;
double tmp;

tmp = infer[mfno][0];

for (i=1;i<3;i++)
      tmp = MAX(infer[mfno][i],tmp);

return tmp;
}

// Defuzzification function.
// The formula used here is membership (u)*centerofgravity for
// the corresponding membership function

double defuzzification()
{

      double num;
      double denum;

      num = 0;
      denum = 0;

      for(int j=0;j<5;j++)
      {
            if (output(j)>0)
            {
                  if (j==0) //LN
                  {
                        num = num + output(j)*(-1);
                        denum = denum + output(j);
                  }
                  else if (j==1) //SN
                  {
                        num = num + output(j)*(-0.5);
                        denum = denum + output(j);
                  }
                  else if (j==2) //ZERO
                  {
                        num = num + output(j)*0;
                        denum = denum + output(j);
                  }
                  else if (j==3) //SP
                  {
                        num = num + output(j)*(0.5);
                        denum = denum + output(j);
                  }
                  else if (j==4) //LP
                  {
```

```
                        num = num + output(j)*(1);
                        denum = denum + output(j);
                  }
            }
      }

      printf("%f",num);
      printf("%f",denum);
      printf("%f",num/denum);

      return (num/denum);
}

void inference(double e, double er)
{

// RULE 1 LN - NEG - LN
if (trapezoidMf(e,in_LN)>0 && trapezoidMf(er,ine_NEG)>0)
{
printf("Rule 1 has effect \n"); // LN - 1 - infer[0][0]
infer[0][0] = MIN(trapezoidMf(e,in_LN),trapezoidMf(er,ine_NEG));
}

// RULE 2 SN - NEG - LN
if (triangleMf(e,in_SN)>0 && trapezoidMf(er,ine_NEG)>0)
{
printf("Rule 2 has effect \n"); // LN - 2 - infer[0][1]
infer[0][1] = MIN(triangleMf(e,in_SN),trapezoidMf(er,ine_NEG));
}

// RULE 3 ZERO - NEG - SN
if (trapezoidMf(e,in_ZERO)>0 && trapezoidMf(er,ine_NEG)>0)
{
printf("Rule 3 has effect \n"); // SN - 1 - infer[1][0]
infer[1][0] = MIN(trapezoidMf(e,in_ZERO),trapezoidMf(er,ine_NEG));
}

// RULE 4 SP - NEG - SN
if (triangleMf(e,in_SP)>0 && trapezoidMf(er,ine_NEG)>0)
{
printf("Rule 4 has effect \n"); // SN - 2 - infer[1][1]
infer[1][1] = MIN(triangleMf(e,in_SP),trapezoidMf(er,ine_NEG));
}

// RULE 5 LP - NEG - ZERO
if (trapezoidMf(e,in_LP)>0 && trapezoidMf(er,ine_NEG)>0)
{
printf("Rule 5 has effect \n"); // ZERO - 1 - infer[2][0]
infer[2][0]  = MIN(trapezoidMf(e,in_LP),trapezoidMf(er,ine_NEG));
}

// RULE 6 LN - ZERO - LN
if (trapezoidMf(e,in_LN)>0 && triangleMf(er,ine_ZERO)>0)
{
printf("Rule 6 has effect \n"); // LN - 3 - infer[0][2]
infer[0][2] = MIN(trapezoidMf(e,in_LN),triangleMf(er,ine_ZERO));
```

```
}

// RULE 7 SN - ZERO - SN
if (triangleMf(e,in_SN)>0 && triangleMf(er,ine_ZERO)>0)
{
printf("Rule 7 has effect \n"); // SN - 3 - infer[1][2]
infer[1][2] =  MIN(triangleMf(e,in_SN),triangleMf(er,ine_ZERO));
}

// RULE 8 ZERO - ZERO - ZERO
if (trapezoidMf(e,in_ZERO)>0 && triangleMf(er,ine_ZERO)>0)
{
printf("Rule 8 has effect \n"); // ZERO - 2 - infer[2][1]
infer[2][1] =  MIN(trapezoidMf(e,in_ZERO),triangleMf(er,ine_ZERO));
}

// RULE 9 SP - ZERO - SP
if (triangleMf(e,in_SP)>0 && triangleMf(er,ine_ZERO)>0)
{
printf("Rule 9 has effect \n"); // SP - 1 - infer[3][0]
infer[3][0] =  MIN(triangleMf(e,in_SP),triangleMf(er,ine_ZERO));
}

// RULE 10 LP - ZERO - LP
if (trapezoidMf(e,in_LP)>0 && triangleMf(er,ine_ZERO)>0)
{
printf("Rule 10 has effect \n"); // LP - 1 - infer[4][0]
infer[4][0] = MIN(trapezoidMf(e,in_LP),triangleMf(er,ine_ZERO));
}

// RULE 11 LN - POS - ZERO
if (trapezoidMf(e,in_LN)>0 && trapezoidMf(er,ine_POS)>0)
{
printf("Rule 11 has effect \n"); // ZERO - 3 - infer[2][2]
infer[2][2] = MIN(trapezoidMf(e,in_LN),trapezoidMf(er,ine_POS));
}

// RULE 12 SN - POS - SP
if (triangleMf(e,in_SN)>0 && trapezoidMf(er,ine_POS)>0)
{
printf("Rule 12 has effect \n"); // SP - 2 - infer[3][1]
infer[3][1] = MIN(triangleMf(e,in_SN),trapezoidMf(er,ine_POS));
}

// RULE 13 ZERO - POS - SP
if (trapezoidMf(e,in_ZERO)>0 && trapezoidMf(er,ine_POS)>0)
{
printf("Rule 13 has effect \n"); // SP - 3 - infer[3][2]
infer[3][2] = MIN(trapezoidMf(e,in_ZERO),trapezoidMf(er,ine_POS));
}

// RULE 14 SP - POS - LP
if (triangleMf(e,in_SP)>0 && trapezoidMf(er,ine_POS)>0)
{
printf("Rule 14 has effect \n"); // LP - 2 - infer[4][1]
infer[4][1] = MIN(triangleMf(e,in_SP),trapezoidMf(er,ine_POS));
```

```
}

//RULE 15 LP - POS - LP
if (trapezoidMf(e,in_LP)>0 && trapezoidMf(er,ine_POS)>0)
{
printf("Rule 15 has effect \n"); // LP - 3 - infer[4][2]
infer[4][2] = MIN(trapezoidMf(e,in_LP),trapezoidMf(er,ine_POS));
}
}

// Auxiliary function that calls the main functions needed
// for fuzzy logic controller.
void fuzzy(double e, double er)
{
      clear();
      inference(e,er);
      defuzzification();
}
```

5.2 Neural Control Application

The single-input single-output (SISO) fuzzy logic controller having 5 rules and in part 5.1 is used to generate the training data for the neural controller. The control surface of the fuzzy controller will also be the control surface of the neural network controller. Figure 5.7 shows the fuzzy logic controller surface.

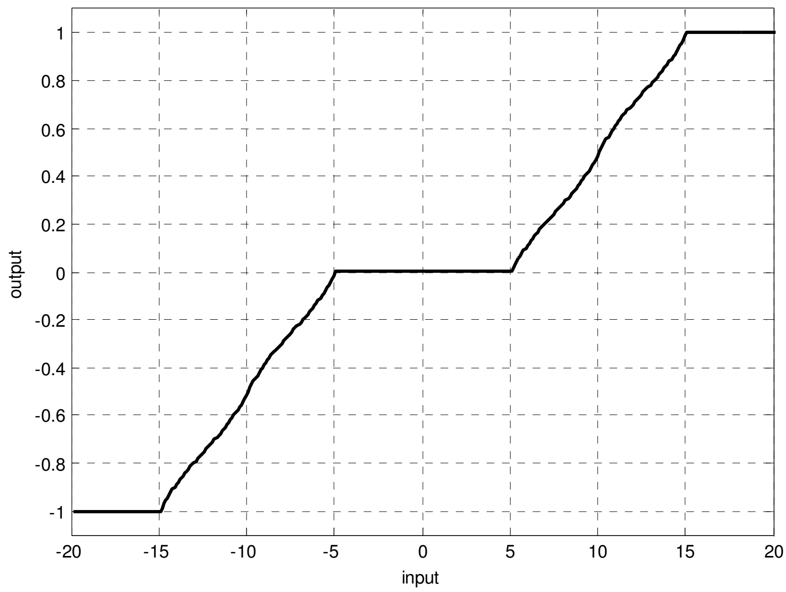


Figure 5.7 Fuzzy controller control surface

Figure 5.8 shows the control surface of the trained neural network with 12 neurons in the hidden layer and 1 neuron in the output neuron.

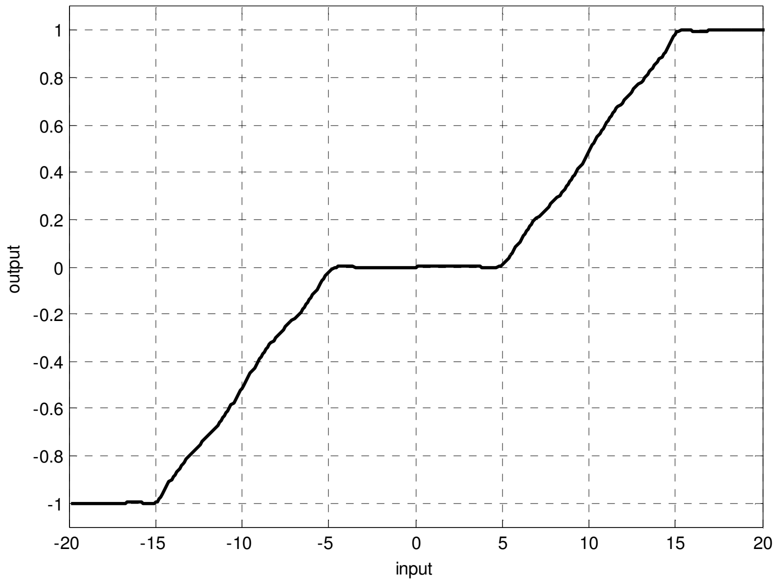


Figure 5.8 Neural network controller control surface

It can be easily seen that the neural controller can be used instead of a fuzzy controller. And note that there are some errors remaining in the output of the neural network controller. Because, neural network is used as a function approximator.

6. CONCLUSIONS AND FUTURE WORK

In conventional control systems, proportional, proportional-integral and proportional-integral-derivative controllers are widely used. The design of such controllers has a fixed procedure. Frequency domain or time domain methods are used in the design. It is necessary to have the mathematical model of the controlled system.

In this thesis, two PID controllers are designed by mathematical model of a DC motor to demonstrate the conventional design procedure. The controller gains are determined by using unit step response for both controllers. The first controller is tuned with Ziegler-Nichols method, while Chein-Hrones-Reswick method is used in the second. It is shown that, the both of the controllers improved the response of the modeled system. The first PID controller has a large overshoot but has smaller rise time and it takes longer time to settle a steady-state position. Second controller has a smaller overshoot as a result of smaller proportional and integral gain but as a consequence the rise time is higher than the first one. In the thesis, a way to determine the controller parameters is demonstrated by using MATLAB/Simulink.

Fuzzy logic controller manage complex control problems through heuristics and mathematical models provided by fuzzy logic, rather than via mathematical models provided by differential equations. This is particularly useful for controlling systems whose mathematical models are nonlinear or for which standard mathematical models are simply not available.

The design of a fuzzy logic controller is demonstrated in the study. A dc motor system is used in the demonstration. But no mathematical model is involved in the design. The simulation environment consists of a dc motor whose field voltage is fixed and armature voltage is controlled by 1 arm universal bridge. The firing signals for the IGBTs are generated by a PWM generator. The PWM generator has an input named modulation index which is the controlled variable in the system. The controller sets the modulation index to the required level to achieve a good performance.

The implementations of fuzzy control imitate the control laws that humans use. The new design techniques are based on more general types of knowledge than differential equations describing the dynamics of the plants. To obtain the required knowledge from the system, various simulations are conducted with different levels of modulation index. Two inputs of the fuzzy logic controller are selected as error signal that is the difference between a set point and the actual plant output, and error rate signal which is the difference between the error signal and the previous value of error signal. The output of the fuzzy logic controller is integrated to obtain the required modulation index. These inputs and the output must have membership functions. The selection of these functions and parameters of these functions are made by trial-and-error method. After repeatedly made simulations with parameter tuning, rule base tuning, input and output scaling factors tuning, an optimum performance is achieved.

Neural control used in which the controller itself is a neural network and method in which controllers are designed based on a neural network model of the plant. These two basically different approaches for implementing neural networks in control are referred to as direct and indirect design method.

The design of a neural network controller is a straight forward procedure if the input and output relation is known for the system. The most important section of the controller design is the training data generation. The training data for the neural controller is obtained from simulation results blended with the designer's knowledge.

Several simulations are made to show the performance of the designed controllers. Set point tracking and load variation tests are simulated and the performance of the controllers under these tests are investigated. Both fuzzy and neural controller track the set point with a minimum steady-state error. The fuzzy controller needs some time to restore the set speed after a load increase while the neural controller shows a better response. It must be noted that, the neural controller gain factors have to be changed for changing load conditions. An important improvement is that the overshoot of the system is much smaller than the PID controllers.

The fuzzy controller does not have a straight forward design procedure to adjust it's parameters while neural controllers only needs the input-output relation. When the input-output relation is known and the relation is nonlinear, it could be much simpler to design a neural controller to achieve the desired response. Fuzzy and neural controllers have superiority in the case of nonlinear control. The performance of both fuzzy and neural controllers depends on the knowledge of the designer.

As a result of this study, a flow chart can be developed in the case of where to use fuzzy controller, and where to use neural controller or is it enough to have a PID controller can be obtained. This study can also be used as a reference for an intelligent control systems course.

One of the aims of this thesis is to form a code base to use in real time applications. C++ is used to construct this code base because of its wide support from the processor producers. The main blocks of a fuzzy logic control system, fuzzification, inference and defuzzification are both explained in the code in a detailed manner.

Industrial applications that need smooth control could be implemented with adaptation of the fuzzy code which is developed in the thesis. This code could be easily downloaded to a digital signal processor (DSP) or general purpose microcontrollers for real applications.

It is planned to develop a C++ implementation for a neural network controller. The most difficult part of the design is the training of the neural network. MATLAB gives the tools for training. It can be used to find the controller with optimum performance. The trained network parameters can be input to the neural controller code and direct control implementation of the neural network controller can be realized.

A comparison between the two C++ implementations for fuzzy and neural controllers can be made. It could be a useful reference that answers which controller has better performance in real time.

REFERENCES

BHENDE, C.N, MISHRA, S., JAIN, S.K., 2006, TS - Fuzzy-Controlled Active Power Filter for Load Compensation. IEEE Transactions on Power Delivery, Future Issue.

CHEN, C., HSIAO,P., 2005, Supporting QoS in Wireless MAC by Fuzzy Control, IEEE Communication Society, WCNC, 1242-1247.

CHEN, T., SHEU, T., 2002, Model Reference Neural Network Controller for Induction Motor Speed Control, IEEE Transactions on Energy Conversion, 17 (2): 157- 163.

CIRSTEA, M.N., DINU, A., KHOR, J.G., MCCORMICK, M., 2003, Neural and Fuzzy Logic Control of Drives and Power Systems, Newnes, GB, 397 p.

DEMUTH, H., BEALE, M., 1998, Neural Network Toolbox User's Guide, Mathworks, 742 p.

DORF, R.C., 1995, Modern Control Systems, Addison-Wesley, MA, 828 p.

DRIELS, M., 1996, Linear Control Systems Engineering, McGraw-Hill, New York, 625 p.

DZUNG, P.Q., PHUONG, L.M., 2005, Control system DC motor with speed estimator by neural networks. International Conference on Power Electronics and Drive Systems, 1030-1035

ELMAS, C., 2003, Fuzzy Logic Controllers, Seçkin Publishing, 230 p.

FANG LI, L, LIU, K, MA, L, 2005, Intelligent Control Strategy of SVC", IEEE/PES Transmission and Distribution Conference,1-4.

GUO, Y., GONG, D., CHENG, J., 2004, Coke Oven Heating Temperature Fuzzy Control System, IEEE International Conference on Control Applications, 195-198.

IBRAHIM, A.M., 2004, Fuzzy Logic for Embedded Systems Applications, Newnes, USA, 289 p.

KUO, B., 1975, Automatic Control Systems, Prentice-Hall, New Jersey, 653 p.

LEE, C.C., 1990, Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part I. IEEE Transactions on Systems, Man, and Cybernetics, 20 (2): 404-418.

McCULLOCH, W.S., PITTS, W., 1943, A Logical Calculus of the Ideas Immanent in Nervous Activity. Bull of Math, Biophysics, 5: 115-133.

MENDEL, J.M., 1995, Fuzzy Logic Systems for Engineering: A tutorial. Proceedings of IEEE, 83 (3):345-377.

NGUYEN, H.T., PRASAD, N.R., WALKER, C.L., WALKER, E.A., 2002, A First Course in Fuzzy and Neural Control, CRC Press, 320 p.

PATYRA, M.J., GRANTNER, J.L., KOSTER, K., 1996, Digital Fuzzy Logic Controller: Design and Implementation. IEEE Transactions on Fuzzy Systems, 4 (4): 439-459.

ROSS, T.J., 2005, Fuzzy Logic with Engineering Applications, Wiley, New Mexico, 623 p.

SANCHEZ, E.N., RICALDE, L.J., LANGARI, R., SHAHMIRZADI, D., 2004, Recurrent Neural Control for Rollover Prevention on Heavy Vehicles. IEEE International Joint Conference on Neural Networks, 3: 1841-1846.

SONG, Y., JOHNS, A.T., 1997, Applications of Fuzzy Logic in Power Systems, Power Engineering Journal, 219-222.

BIOGRAPHY

I was born in Antakya, Turkey, in 1982. I completed my high school education in Adana and graduated from Adana Ticaret Odası Anatolian High School. I received the B.S. degree in Electrical and Electronics Engineering from Çukurova University, Adana, Turkey in 2004. After completion my B.S. training, I have started MSc degree in the department of Electrical and Electronics Engineering in Çukurova University and have been working in Computer Engineering Department as a research assistant since 2004.

My areas of interest include computer programming, fuzzy logic controllers, neural network controllers, digital signal processor applications.

I am a member of Turkish Chamber of Electrical Engineers.

LIST OF PUBLICATIONS

BAYINDIR K.C., CUMA, M.U., TÜMAY, M, 2006, Hierarchical neuro-fuzzy current control for a shunt active power filter. Neural Computing and Applications, Online published.

TÜMAY, M, BAYINDIR, K.C., CUMA, M.U., 2006, A Three-Phase Four-Wire Shunt Active Power Filter with High DC Link Voltage Utilization, AIESP 06, Madeira, Portugal

TÜMAY, M., EKİZ, A., BAYINDIR, K.C., CUMA, M.U., 2006, Power System Harmonics and Compensation for Energy Quality and Management in Real Plants, AIESP 06, Maderia, Portugal

TÜMAY, M., BAYINDIR K.C., CUMA, M.U., TOKUNÇ, C.O., 2005, Dynamic Voltage Restorer with Zero Sequence Compensation, ELMA 05, Sofia, Bulgaria

TÜMAY, M., CUMA, M.U., TEKE, A., BAYINDIR, K.C., 2005 Fuzzy Logic Control of Dynamic Voltage Restorer, ELMA 05, Sofia, Bulgaria

BAYINDIR, K.C., TÜMAY, M., CUMA, M.U., 2005 Fuzzy Logic PWM Current Controller based Shunt Active Power Filter for Power Quality Compensation, ELMA 05, Sofia, Bulgaria

TÜMAY, M., CUMA, M.U., BAYINDIR, K.C., 2005, Hierarchical Neuro-Fuzzy Current Control for a Shunt Active Power Filter, ELECO 2005, 357-361.

TÜMAY, M., TEKE, A., BAYINDIR, K.C., CUMA, M.U., 2005, Simulation and Modeling of a Dynamic Voltage Restorer, ELECO 2005, 31-35.