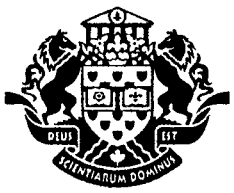


Université d'Ottawa · University of Ottawa

Implementation of A Power Adaptive Fuzzy Control System for End Milling Processes

A thesis submitted to
the School of Graduate Studies and Research
in partial of Master of Applied Science in
Mechanical Engineering

By
Seyed Saeed Rahmati

Ottawa-Carleton Institute for
Mechanical and Aeronautical Engineering
University of Ottawa
Ottawa, Ontario, Canada

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. M. Liang and co-supervisor Dr. T. Yeap for initiating this research, and I am grateful to their support and guidance throughout the study. The technical assistance of Mr. Z. Han and the machine shop staff is gratefully acknowledged.

Thanks are also due to the defense committee, Dr. D. Necsulescu and Dr. W. G. Richarz for their time and effort in reviewing and examining my thesis.

To my family, friends and colleagues, specially my wife Ying, I am greatly indebted for their moral support and encouragement throughout this research thesis.

Abstract

This thesis reports a fuzzy control system designed for power control of end milling processes. As compared to most of the existing end milling control systems, the proposed fuzzy control system has the following advantages: a) multi-parameter adjustment; b) insensitive to changes in work piece geometry, cutter immersion rate, and workpiece material; c) cost-efficient and easy to implement; and d) mathematically modeling-free. The proposed fuzzy controller is a two-input two-output system with simple triangular membership functions for both feedrate and spindle speed. The system also features a scaling factor adjustment mechanism used to tune the gain coefficients. The system is first examined by simulation using Simulink and Matlab fuzzy logic toolbox and then verified by various experiments on a CNC milling machine. The experiments carried out include: i) steel cutting with different change patterns (gradual linear change, gradual curved change, and abrupt step change) in depth of cut; ii) steel cutting with full and partial immersion rates; iii) steel cutting with variable immersion rate; and iv) aluminum cutting with step change in depth of cut. The experimental results show that as compared to single parameter (feedrate) adjustment the material removal rate can be improved by up to 25% when both feedrate and spindle speed are adjusted. It is also shown that the proposed system is stable and displays very good transient performance under all of our experimental conditions, indicating sound robustness. The use of power sensor has significantly reduced investment cost and avoided tedious setups. The simplicity in design and implementation of the system has been demonstrated in our development process.

Table of Contents

List of Figures

List of Tables

Nomenclature

CP	Power change
E	Power Error
f	Feedrate
f'	Fuzzified feedrate
f_0	Defuzzified feedrate
f_{com}	Feedrate command
f_t	Maximum allowable feedrate per tooth
K_{CP}	Scaling factor for power change
K_E	Scaling factor for power error
K_f	Scaling factor for feedrate
K_v	Scaling factor for spindle speed
P	Actual power
P_{ref}	Reference power
v	Spindle speed
v'	Fuzzified spindle speed
v_0	Defuzzified spindle speed
v_{com}	Spindle speed command
x	A vector of fuzzy sets
x_0	A vector of crisp input variables to fuzzifier
z	Number of cutter teeth
α	Tuning factor exponent
$\mu(x)$	Membership degree
λ	Tuning factor

Chapter 1

Introduction

1.1 Background

Milling is one of the most useful yet complex machining processes. Since 18th century when machining operations started growing, researchers and inventors were eager to improve this technology. The most significant event in this improvement process was the introduction of Numerical Controlled (NC) machine tools by Parson's Machine Tool Company in the 1950s. Later, when computers became less expensive and more powerful, the servo-control function was implemented using on-board computers instead of hard-wired digital circuits, and the NC machines evolved to Computer Numerically Controlled (CNC) machines.

Although CNC machine tools have improved the machining technology they also have a major drawback: the machine operating parameters, such as feedrates and spindle speeds are preset and fixed throughout the operation. These parameters are selected by well-experienced CNC programmers, and in order to avoid tool breakage they use conservative values. Therefore, the production time is usually much longer than the optimum one. This issue has drawn attention of researchers and engineers around the world to design a process control system based on the on-line monitored information. Many researchers have investigated and developed variety of control systems. However,

successful industrial implementation of these systems has not been reported in the accessible literature.

1.2 Motivation

A closer examination of existing control system, as revealed in the next chapter, indicates that they mostly suffer from the following:

- Lengthy development process;
- Lack of robustness;
- High investment cost; and
- Single parameter adjustment.

The time required for developing a control system is largely affected by the system dynamics identification and modeling processes. Due to the complexity of a milling process, extensive experiments have to be carried out to provide a reliable system model. This will inevitably cause a significant amount of machine down time, which may not be acceptable by most manufacturing companies.

The lack of robustness is mainly caused by the fact that many existing control systems are developed for a particular machine-tool-workpiece combination. It is well known that today's machine shop features small batch sizes and wide variety of products, requiring variant tools and fixtures. Any change in parts, tools or even fixtures may alter system dynamics substantially. As a result, any variation in machine, tool or workpiece will lead to deteriorated system performance with excessive oscillation and overshoots. The application of such system is hence very limited.

The majority of the existing control systems are forced-based. Force sensors are very expensive, in the range of $35,000~$60,000. The workpieces are usually mounted on the top of a dynamometer. Due to the direct exposure to chips, cutting fluid, and possible overload, an expensive dynamometer may soon be ruined. Other drawbacks of using dynamometers include tedious workpiece and fixture setup and instability introduced by the mounting process, and additional sensor component in the physical system.

To the best of our knowledge, all of the existing control system developed for milling processes are based on single parameter (feedrate) adjustment. In this case, if the feedrate is too high for a constant spindle speed, the adjustment may cause excessive tooth load, edge built-up, tool breakage, vibration and hence the adjustment may have to be stopped, thereby limiting further improvement in material removal rate. On the other hand, if the feedrate is too low for a constant spindle speed, the tool may unnecessarily experience crater wear and plastic deformation due to the higher temperature.

Hence, it is proposed to design a new adaptive control system in end milling operations by using an alternative sensor and a methodology that is independent of mathematical modeling. The alternative sensor is a Load Controls PH-3A power sensor, which costs about $1000. The other advantages of using power sensor include: a) spindle power is linearly proportional to cutting loads; b) the system setup is simple and the sensor is not exposed to the cutting environment; and c) since the power is measured from the power supply, the monitoring process will not affect the cutting dynamics. A simple but powerful control technique is fuzzy logic control, which is based on a series of linguistic rules. These rules are in fact commonly used by well-experienced machine

operators in machine shops to regulate the cutting parameters such as feed rate and spindle speed without analyzing the complex cutting dynamics (Tarng and Cheng 1992). Thus, these rules can be adopted by a fuzzy logic controller for on-line adjustment of machining parameters. Another advantage of fuzzy control is that it tolerates incomplete knowledge and less accurate information, and thus the lengthy system identification process for different machine-tool-workpiece combinations can be avoided. Furthermore, multi-input, multi-output control can be easily implemented, thereby making it possible for multi-parameter adjustment.

1.3 Objective

The main objective of this thesis is to develop an end milling control system which is

a) capable of adjusting both feedrate and spindle speed;
b) insensitive to changes in cutting geometry, immersion rate, and workpiece material;
c) easy to develop and implement; and
d) cost-efficient.

To this end, the following have been accomplished:

1. A fuzzy logic control algorithm has been developed. The relationships between feedrate and spindle power, and between spindle speed and spindle power are identified and translated into a series of linguistic rules. A fuzzy rule base is accordingly constructed using these rules.

2. The proposed fuzzy control system is evaluated by simulation using Simulink and Matlab fuzzy toolbox.
3. Experiments have been carried out to examine the performance of the proposed control system and the advantages of adjusting both feedrate and spindle speed.

The remainder of the thesis is divided into four chapters. Chapter 2 consists of a brief review of adaptive control in machining processes, its classifications and previous research work. The design of the adaptive fuzzy control as well as the simulation evaluation of the system will be discussed in chapter 3. Chapter 4 presents the experimental results of the spindle power adaptive fuzzy control in end milling operations. Conclusion and possible future research are given in Chapter 5.

Chapter 2

Review of Adaptive Control in Machining Processes

Over the last two decades, a number of adaptive control systems have been developed for machining processes. According to (Koren 1983, Ulsoy *et al* 1983), control methods used in these systems may be classified into Adaptive Control with Optimization (ACO) and Adaptive Control with Constraints (ACC). The recent years have seen significant development in fuzzy logic and neural network based control systems. In this chapter both conventional ACO, ACC and unconventional (fuzzy logic and neural networks) ACC control systems will be briefly reviewed.

2.1 Development of Adaptive Control with Optimization (ACO) systems

Essentially, ACO systems aimed at optimizing machining processes based on a performance index (Tarng and Wang 1994). An economic indicator such as the ratio of production rate and tool wear rate is usually selected as the performance index. The first ACO system was designed in 1960s which was only demonstrated in the laboratory (Centner 1964, Huber and Centner 1968). The system consisted of on-line optimization and adaptive control strategies. However, the major drawback was unavailability of the accurate on-line measurement of tool wear. Even today, a reliable tool wear measurement

device that can operate in an industrial environment is still not available (Colwell *et al* 1978, Koren 1991).

Due to unavailability of a reliable device for tool wear measurement, model-based schemes for estimation of tool wear rate from indirect measurements such as cutting forces (Danai and Ulsoy 1987, Billatos *et al* 1991, Chiang *et al* 1993) were developed. In their work, tool wear was included in the model as a state variable. The measured output was the cutting force, and was used in conjunction with the model to estimate the state of tool wear of the tool. In the work of Billatos *et al* (1991) a linear recursive adaptive tool failure identifier model and a cutting condition constraint model using cutting forces were developed. They also developed a knowledge-based machining system for the optimization purpose. An application of artificial intelligence in ACO system was reported by Chiang *et al* (1993) who utilized two different neural networks for on-line determination of optimal cutting conditions. In their work, the two networks were used for different tasks: one is used to model the cutting process and the other to determine the optimum cutting parameters by parallelizing the augmented Lagrangian multiplier algorithm.

Although the ACO systems appear to be a logical choice by virtue of their clear control objective, they have not yet obtained industrial acceptance in spite that various indirect tool wear measurement methods have been developed.

2.2 Development of Adaptive Control with Constraints (ACC) systems

2.2.1 Classical ACC Systems

Conventional ACC systems can be classified into fixed gain and adjustable gain systems. In some manufacturing literature, the systems with fixed gains are termed adaptive. However, according to the control literature (Lauderbaugh and Ulsoy 1983), they are not adaptive systems.

2.2.1.1 Fixed Gain controllers

One of this type of systems is reported by Milner (1975). He developed a fixed gain Proportional-Integral (PI) control scheme to adjust the feedrate in order to maintain the end milling cutter deflection constant at a reference level. Experimental results indicated that the controller work was acceptable only when the variation of depth of cut is not too large. Later, a fixed gain dynamic model parallel to the controlled process was used to assist the gain adjustment of the PI controller such that a constant open-loop gain specified by the parallel model was always maintained, despite variations in the controlled process gain (Stute and Goetz 1976).

Since all these fixed gain controllers performed well only when process parameter variations were small, researchers turned to approaches with better adaptation technique such as Self-Tuning Control (STC) and Model Reference Adaptive Control (MRAC) systems. In spite of several restrictions in these types of control algorithms, many reports indicated that STC and MRAC approaches could be implemented in machining processes.

2.2.1.2 Self-Tuning Control (STC) Systems

Elbestawi and Sagharian (1987) presented one of the earliest reports on STC application. They investigated several self-tuning control strategies and showed that in general they had good performance and stability. However, better performance can be obtained using the deadbeat controller of increased order (DB2) and the discrete Proportional-Integral-Derivative (PID) controller. This could be observed clearly during the transients of a step change in axial depth of cut.

Hsu and Hsieh (1994) also investigated the use of STC in machining processes. They applied two self-tuning control algorithms, a deadbeat self-tuning control with increased order and a PID-type self-tuning control based on the pole-placement, as well as a variable gain MRAC to the CNC milling cutting system. Their experimental results indicated that the deadbeat self-tuning controller performance was very close to the MRAC performance. However, the PID-type self-tuning controller performed slightly better than the last two. The PID-type controller had fewer oscillations during the transient and shorter transient period.

2.2.1.3 Model Reference Adaptive Control (MRAC) Systems

Tomizuka *et al* (1983) were probably the first one who provided extensive investigation of parameter adaptive force control for end milling. A discrete-time MRAC algorithm based on the Independent Tracking and Regulation (ITR) technique (Landau and Lozano 1981) was adopted to regulate the cutting force in the end milling operation. They used a first-order discrete-time model for the controller design. To eliminate the periodic force component due to cutter runout, an average force per revolution was used

to represent the cutting force level. Though oscillating transient behaviors were observed when both process time constant and gain were estimated, a better regulation performance was obtained by estimating the process gain while maintaining the pole of the cutting process at a constant value.

In an attempt to improve the mathematical model of a machining process, Lauderbaugh and Ulsoy (1989) developed a second-order empirical model based on feedrate input and resultant force output. The discrete-time MRAC technique that they applied was first designed by Goodwin and Sin (1984). The MRAC system was utilized to control end milling processes. Their digital simulation results showed that the MRAC approach could lead to better transient performance as compared to the fixed gain PI controller in the presence of large parameter variations. However, in real cutting processes the controller did not perform well. This could be due to the biased estimates caused by measurement noise due to cutter runout. Finally, they concluded that the primary problems with the MRAC controllers were the complexity of the design and the difficulties associated with measurement noise.

Han (1996) reported a deadbeat MRAC system for both force and power control in end milling processes. The performance is reasonably good within his experimental scope. However, his work did not go beyond the barrier of single parameter adjustment.

The major drawback of all the above systems is the necessity of the mathematical modeling process. Due to non-linearity and complexity of the machining processes and the machines dynamics structure, obtaining an accurate model of the system is very difficult if not impossible. This is the main reason why most of the mathematical models reported so far are of first order, which may not be adequate for a reliable process control

system. For this reason, researchers have started to examine alternative techniques in particular, the fuzzy logic and neural networks based approaches.

2.2.2 Unconventional ACC Systems

In recent years, artificial intelligence has drawn considerable attention from many research institutions and industries. Reported studies include fuzzy logic control and neuro-fuzzy control of machining processes. The main advantage of this technique in machining process control is that this technology often provides controllers with performance similar to the performance of an expert human operator (Berenji 1989) without constructing a complex process model.

Tarng and Cheng (1992) developed a fuzzy logic controller for end milling operations. The purpose was to achieve on-line adjustment of feed rate and to maintain a constant cutting force. Their design was based on simple triangular shape fuzzy sets with seven membership functions for both inputs and outputs of the fuzzy controller. Even though their system did not include adaptive mechanism, the experimental results showed a significant improvement in robustness and stability compared to several conventional ACC systems. They later applied a similar controller, with different linguistic rules, to turning operations to attain on-line adjustment of feed rate with constant cutting forces (Tarng and Wang 1994).

In another report, a neuro-fuzzy controller was used to achieve self-organizing control for turning (Yeh *et al* 1995). The neuro-fuzzy controller was a hybrid system consisting of a self-organization network and a perceptron-like network. This hybrid system was functionally equivalent to a fuzzy controller. The back-propagation neural

network was used to learn the dynamic behaviors of the system. The advantage of this system was that it could start from an empty control rule base. However, its disadvantage was that it had to be trained off-line before it could be used for on-line control. The combination of fuzzy logic and neural network was also used for end milling control (Tarng and Hwang 1995). The system consisted of a feedforward neural network and a fuzzy logic controller. The neural network was first developed to acquire the inverse-dynamics of the controlled plant. Then, a fuzzy feedback mechanism performed an adaptive modification of connection weights for the neural network. The purpose of this controller was to adjust feedrate to achieve a constant force.

A two-input, two output fuzzy logic controller was recently developed by Haber *et al* (1996) for a supervisory control of end milling. In addition to feedrate, they also took spindle speed into consideration. Hence, a constant force was maintained by on-line adjustment of feedrate and spindle speed. Tuning was not considered in their controller.

In addition to force control, fuzzy logic has also been adopted for other machining applications. Chen *et al* (1995) reported a fuzzy expert system for the design of machining operations. The system consisted of four modules: a database, a cutting selection module, a cutting condition design module and a learning module. This system was used to select cutters and cutting conditions based on partial and incomplete information. Du *et al* (1992) presented a study on tool condition monitoring in turning using fuzzy set theory. The tool conditions considered included tool breakage, several states of tool wear, and chatter. Several sensors such as force, vibration and power sensors were used. In another report (Ralston *et al* 1992), fuzzy logic was used to control chip form during turning. They used an acoustic emission sensor along with a fuzzy logic

controller. At the end of the report, the authors concluded that the fuzzy logic technique performed more reliable than a conventional linear controller.

The above literature review indicates that a robust and cost-efficient fuzzy controller capable of adjusting both feedrate and spindle speed is still not available though it is highly desirable for shop floor machining control. This study is therefore directed towards the development of such a controller. The design and implementation of the controller are detailed in the following chapters.

Chapter 3

Design of the Spindle Power Adaptive Fuzzy Controller

As revealed in the previous chapter, fuzzy logic technique has the following advantages:

- Complex mathematical modeling process is not required.
- Inputs, outputs and control response can be specified in a way similar to those used by a human expert.
- Sophisticated knowledge based on the experience of the domain expert can be incorporated into the system in a relatively understandable format.
- A control system can be quickly developed.
- Incomplete knowledge and inaccurate information can be tolerated.

As machining processes are time-varying and complex, accurate mathematical modeling is very time-consuming. Accurate information and knowledge are difficult to obtain. Furthermore, as the combination of tool-machine-part varies frequently, a model developed based on one combination of tool-machine-part may not work for others and it is virtually impossible to develop such models for all possible tool-machine-part combinations.

For these reasons, an adaptive fuzzy control system based on scaling factor tuning is proposed for machining process control. In this chapter, the structure of the adaptive fuzzy control system will be illustrated and its simulation will also be presented.

3.1 Adaptive Fuzzy Controller Structure

As shown in Figure 3.1, the adaptive fuzzy control structure used in this thesis consists of the following components:

1. Fuzzification.
2. Membership Function.
3. Control Rules.
4. Fuzzy Inference System.
5. Defuzzification.
6. Scaling Factors Tuning.

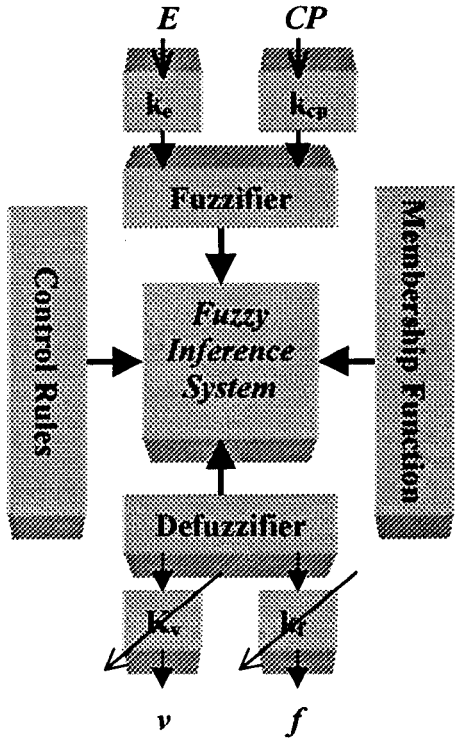


Figure 3.1 Power adaptive fuzzy controller

These components are explained below.

3.1.1 Fuzzify Inputs

The first step of fuzzy control policy is to take the inputs and determine the degree to which they belong to each of the appropriate fuzzy set in the various input

universes of discourse. According to most of the previous works reported in the literatures (e.g., Batur and Kasparian 1993, Tarng *et al* 1994, Yeh *et al* 1995), the inputs to the process controller are obtained from sensor signals. In this thesis, the measured power, P_i, is compared with the reference power, P_{ref} . This gives two inputs to the controller: the power error, E, and the power change, CP. At each sampling instant, i, the power error and the power change are respectively calculated as

$$E_i = P_{ref} - P_i \tag{3-1}$$

$$CP_i = P_i - P_{i-1} \tag{3-2}$$

Further, to normalize the inputs and to map them into suitable linguistic values, they are multiplied by scaling factors, k_e and k_{cp}.

The input data are crisp, and fuzzification is required to map the range of crisp inputs to corresponding fuzzy values for the system input variables. This process can be expressed by:

$$x = fuzzifier\ (x_0) \tag{3-3}$$

where x_0 is a vector of crisp values of one input variable from the process, and x is a vector of fuzzy sets defined for the variable.

3.1.2 Membership Functions

To transform crisp inputs into fuzzy inputs, membership functions must first be determined for each input. Once membership functions are specified, a real time input value, such as a power error, is sampled and used to produce fuzzy inputs via membership function. There are usually several fuzzy inputs corresponding to a single crisp input since the crisp input can have partial membership grades in several fuzzy sets.

The membership grading is represented by a real number ranging between 0 and 1 within the closed interval.

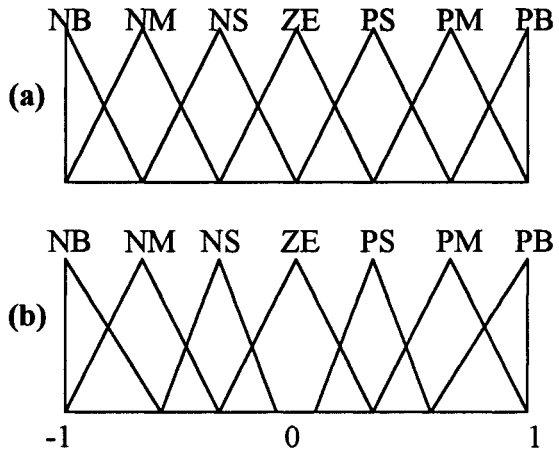


Figure 3.2 Triangular membership functions for the inputs (a) and outputs (b) of the fuzzy controller

To simplify calculations, triangular shape membership functions are used in this thesis as widely adopted in the literature (e.g., Tarng and Cheng 1993). Seven fuzzy sets are used for the inputs and outputs of the controller (Figure 3.2(a) and (b)), i.e., NB, negative big; NM, negative medium; NS, negative small; ZE, zero; PS, positive small; PM, positive medium; and PB, positive big. The corresponding values of the membership functions are listed in Appendix C. The universe of discourse of all inputs and outputs is within the range of [–1, 1]. It is noted that any output deviating from zero will trigger an adjustment when the NS and PS sets intersect at zero and hence lead to unnecessary oscillation. It is therefore proposed that a narrow open area be kept between NS and PS sets in Figure 3.2 (b).

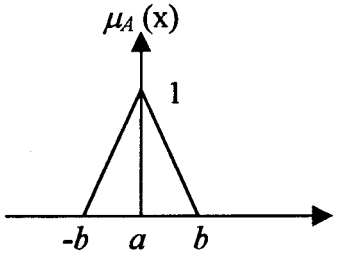


Figure 3.3 Evaluation of membership degree of a triangular fuzzy set.

Mathematically, the membership function, μ_A, for a fuzzy set, A, is given by

$$\mu_A : X \rightarrow [0,1] \tag{3-4}$$

Hence, the degree of membership for triangular fuzzy sets (see Figure 3.3) can be defined as:

$$\mu_A(x) = \begin{cases} \dfrac{x+b}{a+b} & for\ -b \le x \le a \\ \dfrac{x-b}{a-b} & for\ \ a \le x \le b \end{cases} \tag{3-5}$$

3.1.3 Control Rules

One of the most crucial components of the fuzzy controller structure is the control rule module. The set of control rules defines the system behavior and replaces the mathematical modeling of the system (Lee and Wang 1996). The fuzzy rules, which use fuzzy inputs to determine system actions, are obtained from the skilled operators, experiments, and prior knowledge of the end milling processes. Each of the rules can be written as an *IF-THEN* statement that describes the action to be taken in response to various fuzzy inputs, e.g.,

IF ***E*** *is* ***NB*** *AND* ***CP*** *is* ***ZE*** *THEN* ***f*** *is* ***NM*** *also* ***v*** *is* ***PB***

The above rule, in natural language, states that if power error, E, falls in the negative big fuzzy set and power change, CP, is in the zero set, then the suggested feedrate adjustment is negative medium and the suggested speed adjustment is positive big. Each statement such as the one above is called a premise. Usually there are several premises for one input, which leads to a consequence or an action by the controller. This process will be discussed in detail in the next section.

Once the rules are formulated, decision-making tables such as Table 3.1(a) and (b) for the two-input, two-output system can be constructed. After a set of experiments and a careful examination of the results, it is observed that some of the rules in the table have no effect on the control process since the associated cases usually do not exist. Accordingly they can be eliminated. This is beneficial in reduction of the computational time. The reduced decision tables are shown in Tables 3.2(a) and (b).

E \ CP	NB	NM	NS	ZE	PS	PM	PB
NB	NS	NS	NM	NM	NM	NB	NB
NM	ZE	NS	NM	NM	NB	NB	NB
NS	PS	ZE	ZE	NS	NS	NM	NB
ZE	PS	PS	PS	ZE	ZE	NS	NS
PS	PM	PM	PS	PS	ZE	NS	NS
PM	PB	PM	PM	PS	PS	ZE	NS
PB	PB	PB	PB	PM	PM	PS	ZE

Table 3.1(a) Fuzzy rules for adaptive control of feedrate before rule reduction

E \ CP	NB	NM	NS	ZE	PS	PM	PB
NB	PM	PM	PM	PB	PB	PB	PM
NM	PS	PS	PM	PB	PB	PB	PB
NS	NS	ZE	ZE	PS	PS	PM	PM
ZE	NS	NS	ZE	ZE	ZE	PS	PM
PS	NM	NM	NS	ZE	ZE	PS	PS
PM	NB	NM	NM	NS	NS	ZE	PS
PB	NB	NB	NM	NM	NM	NS	ZE

Table 3.1(b) Fuzzy rules for adaptive control of spindle speed before rule reduction

E \ CP	NB	NM	NS	ZE	PS	PM	PB
NB			NM	NM	NM		
NM		NS	NM	NM	NB		
NS	PS	ZE	ZE	NS	NS	NM	
ZE		PS	PS	ZE	ZE	NS	NS
PS	PM	PM	PS	PS	ZE	NS	NS
PM		PM	PM	PS	PS		
PB			PB	PM	PM		

Table 3.2(a) Fuzzy rules for adaptive control of feedrate after rule reduction.

E \ CP	NB	NM	NS	ZE	PS	PM	PB
NB			PM	PB	PB		
NM		PS	PM	PB	PB		
NS	NS	ZE	ZE	PS	PS	PM	
ZE		NS	ZE	ZE	ZE	PS	PM
PS	NM	NM	NS	ZE	ZE	PS	PS
PM		NM	NM	NS	NS		
PB			NM	NM	NM		

Table 3.2(b) Fuzzy rules for adaptive control of spindle speed after rule reduction

3.1.4. Fuzzy Inference System

In this stage of the process, the rules along with the membership degree of the fuzzy inputs will determine the fuzzy outputs. A technique called max-min inference is used to calculate a numerical value representing the aggregate effect of all the triggered rules by an input value. The result is a fuzzy output for each type of consequent action.

According to Zadeh (1965), the union of fuzzy sets A and B is expressed as

$$C := A \vee B : \tag{3-6}$$

and the membership function of C is given by

$$\mu_C(x) = {}_{def} MAX\{\mu_A(x), \mu_B(x)\} \text{ for all } x \in X \tag{3-7}$$

Similarly, the intersection of fuzzy sets A, B, is written as

$$D := A \wedge B : \tag{3-8}$$

and the membership function of D is

$$\mu_D(x) = {}_{def} MIN\{\mu_A(x), \mu_B(x)\} \text{ for all } x \in X \tag{3-9}$$

Now, for the two-input, two-output fuzzy system, the operation can be described as follows (also see Figure 3.4):

Premise 1: If E is a_1 AND CP is b_1 Then f is c_1 also v is d_1

Premise 2: If E is a_2 AND CP is b_2 Then f is c_2 also v is d_2

Premise n (Knowledge): If E is a_n AND CP is b_n Then f is c_n and v is d_n

Premise n+1 (Fact): E is A AND CP is B

Consequence 1: f is C

Consequence 2: v is D

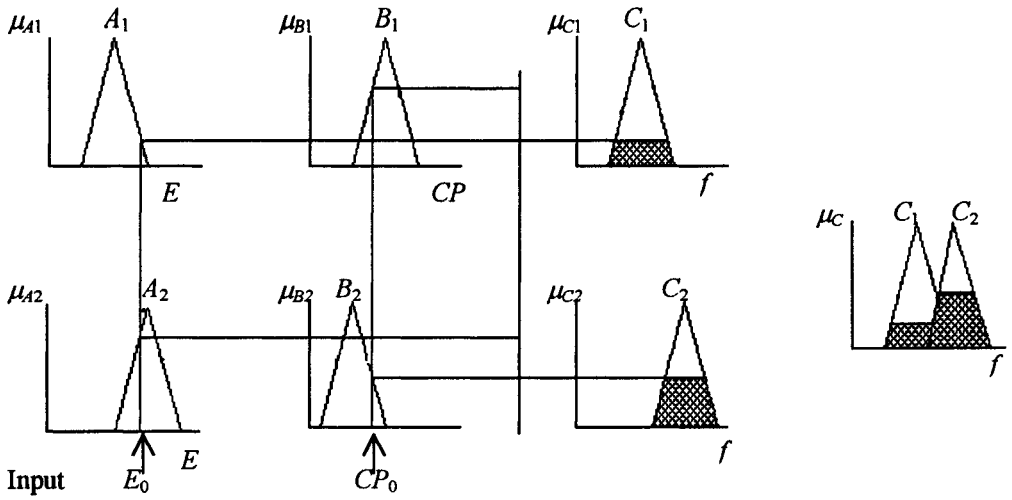


Figure 3.4 MAX-MIN fuzzy inference under crisp inputs (for feedrate)

Figure 3.4 shows the max-min inferencing process for the crisp input values E_0 and CP_0 which are regarded as fuzzy singletons.

In general, for an n-rule controller with inputs $A = E_0$ and $B = CP_0$, the consequences C and D can be expressed as:

$$\mu_C(f') = (\mu_{A1}(E_0) \wedge \mu_{B1}(CP_0) \wedge \mu_{C1}(f')) \vee \ldots (\mu_{An}(E_0) \wedge \mu_{Bn}(CP_0) \wedge \mu_{Cn}(f')) \quad (3\text{-}10)$$

$$\mu_C(S') = (\mu_{A1}(E_0) \wedge \mu_{B1}(CP_0) \wedge \mu_{C1}(v')) \vee \ldots (\mu_{An}(E_0) \wedge \mu_{Bn}(CP_0) \wedge \mu_{Cn}(v')) \quad (3\text{-}11)$$

3.1.5 Defuzzification

Defuzzification is the process of mapping the inferred fuzzy control actions to crisp control actions. A defuzzification strategy is aimed at producing a non-fuzzy control action that best represents the possibility distribution of the inferred fuzzy control action. This can be expressed by:

$$f_0 = Defuzzifier\ f' \tag{3-12}$$

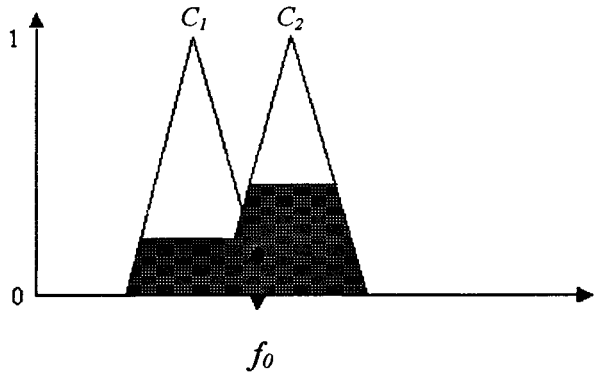


Figure 3.5 Defuzzification of feedrate using center of area (COA)

where f' is the fuzzy control action, f_0 the crisp control action, and defuzzifier the defuzzification operator.

Center of Area (COA) method is the most commonly used defuzzification strategy in real-time implementations of fuzzy logic control (Tumizoka 1990, Dasaradah *et al* 1991, Haber *et al* 1996). The COA method generates the center of gravity of the possible distribution of a control action. The crisp output, f_0, associated to the center of gravity for the resulting fuzzy set C (refer to the equation (10) and Figure 3.5) is obtained using the COA method as follow:

$$f_0 = \frac{\int f' \mu_C(f') df'}{\int \mu_C(f') df'} \tag{3-13}$$

The calculation for the spindle speed is similar to the one for feedrate.

Given the crisp output values f_0 and v_0, the control commands can be calculated for each sampling period. The feedrate and spindle speed commands for the period i are obtained as follows:

$$f_{com}(i) = f_{com}(i\text{-}1) + K_{f(i)}\, f_0(i) \tag{3-14}$$

$$v_{com}(i) = v_{com}(i\text{-}1) + K_{v(i)}\, v_0(i) \tag{3-15}$$

where $f_{com}(i\text{-}1)$ and v $(i\text{-}1)$ are the control commands of the previous sampling period, K_f and K_v are the scaling factors for feedrate and spindle speed deviations respectively.

3.1.6 Scaling Factor Tuning

There are several adaptation techniques for fuzzy controller, such as:

(a) membership function tuning,

(b) input, output scaling factors tuning, and

(c) linguistic rule tuning.

The second technique, also referred to as gain coefficient tuning, appears to be more effective and simpler for implementation of a control policy (Mallampati and Shenoi 1991). The other two techniques usually require additional algorithms such as neural networks and genetic algorithms, as well as an off-line learning procedure.

Tuning the scaling factors of the output parameters, feedrate and spindle speed deviations, provides better response to the changes in the cutting process. In other words, this technique leads to the development of an adaptive fuzzy controller whose control actions change with respect to the machining processes and the environments in which it operates.

The tuning procedure involves the adjustment of the scaling constants of the feedrate and spindle speed in order to avoid, or minimize, the overshoots when sudden changes in a cutting process occur. Also it prevents the power from continuous decline below the reference level. This can be achieved by examining the trend of the power error and the change of power. Based on the trend of power error and the change of power, the scaling factors K_{fi} and K_{vi} in Equations (3-14) and (3-15) are obtained as

$$K_{fi} = \lambda_i \, K_{fi-1} \tag{3-16}$$

and

$$K_{vi} = \lambda_i \, K_{vi-1} \tag{3-17}$$

To determine λ_i, define the two consecutive power errors by

$$E_i = P_{ref} - P_{i-1}$$

$$E_{i-1} = P_{ref} - P_{i-2} \tag{3-17}$$

and the two consecutive changes of power by:

$$CP_i = P_i - P_{i-1}$$

$$CP_{i-1} = P_{i-1} - P_{i-2} \tag{3-18}$$

λ_i can be tuned as follows

$$\begin{aligned}
&\textit{if} \quad \left|\frac{CP_i}{CP_{i-1}}\right| > 1 \\
&\quad \textit{if} \;\; \textit{sign} \; CP_i \neq CP_{i-1} \qquad \textit{set} \quad \lambda_i = 1 \\
&\quad \textit{else if} \;\; |E_i| \leq |E_{i-1}| \qquad \textit{set} \;\; \lambda_i = \left|\frac{CP_{i-1}}{CP_i}\right|^{\alpha} \\
&\quad \textit{else if} \;\; |E_i| > |E_{i-1}| \qquad \textit{set} \;\; \lambda_i = \left|\frac{CP_i}{CP_{i-1}}\right|^{\alpha} \\
&\textit{else} \;\; \textit{set} \;\; \lambda_i = 1
\end{aligned}$$

where $0 \leq \alpha \leq 1$.

The main idea of this algorithm is to make adjustment based on the latest trend of power error and power change. If the current situation is worse than before, or, in other words, the trend is away from the reference level, more adjustment should be made, i.e., λ_i should be greater than 1. If the current situation is better than the previous case, or, the trend is towards the reference level, less adjustment is preferred. Accordingly, λ_i is set to be less than 1. If the current situation is the same as before, or no clear trend can be observed, it is desirable to keep K_f or K_v unchanged, i.e., $\lambda_i = 1$. For example, the condition specified by $\left|\frac{CP_i}{CP_{i-1}}\right| > 1$, sign($CP_i$)=sign($CP_{i-1}$), and $|E_i| \leq |E_{i-1}|$ simply means that the situation between sampling points i and i-1 is better than that between sampling points i-1 and i-2, and the power level is approaching the reference level in a faster pace. Apparently a smaller λ_i is prepared to avoid over-adjustment. The opposite is true, if a situation occurs such that $\left|\frac{CP_i}{CP_{i-1}}\right| > 1$, sign($CP_i$) $\equiv$ sign(CP_{i-1}) and $|E_i| > |E_{i-1}|$.

The selection of α in the above algorithm is often situation-dependent. In this study, it appears α=0.15 fits the tuning process very well. The initial values of k_f and k_v are found, by trial and error, to be 0.86 and 120 respectively.

In the meantime, the input scaling factors k_e and k_{cp}, for power error and power error change, are adjusted offline for different type of machines. This is accomplished by selecting appropriate scaling factors to normalize the inputs. In this thesis, the minimum actual power when the machine is idle at the lower spindle speed limit, which is set to 200 rpm, is recorded to be 0.45 HP. The power reference is set to 0.85 HP, and the maximum actual power is selected as 1.25 HP. Accordingly, absolute value of the

difference between the power reference and each power limit, minimum and maximum, is 0.4. Therefore, the scaling factor, k_e, for mapping E values into the interval of -1 and 1 of the universe of discourse is calculated as 2.5. Similarly for CP, when minimum actual power is 0.45 HP and maximum actual power is 1.25 HP the absolute value of the difference is 0.8, hence $k_{cp} = 1.25$.

3.2 Simulation of the Fuzzy Controller

In order to evaluate the fuzzy logic system before conducting experiments, the fuzzy controller was simulated using MATLAB and SIMULINK software. The input data were arranged to get a set of data for regulated spindle power when a multi-step shape 1018 cold rolled steel workpiece as shown in Figure 3.6 was machined. Outputs were the feedrate and spindle speed deviations. The fuzzy controller for the simulation was designed using the MATLAB Fuzzy Logic Toolbox (Figure 3.7). The inputs and outputs of the fuzzy controller were the same as those described in previous sections. Inputs were power error and power change, and outputs were feed rate and spindle speed deviations each with 7 membership functions. The rules were tabulated in Tables 3.1 and 3.2. The center of gravity or centroid method was used for defuzzification. The Matlab program listing is given in Appendix B.

Figure 3.8 presents the simulation diagram. The diagram consists of several components including a fuzzy inference system, a milling block which generates the inputs for the fuzzy controller, a set of connections (Mux and Demux), a saturation block which regulates the input data from the sim.dat file, and graphs to show the outputs results. The output results were shown in Figures 3.9 and 3.10, for feed rate change and

spindle speed change respectively. As can be seen from Figures 3.9 and 3.10, both feed rate and spindle speed react to the power changes rapidly and they decrease when the power increases. Also the figures show that the rate of decrease depends on the amount of power increase.

The simulation results demonstrated the validity of the proposed fuzzy controller. The implementation of the controller is given in the next chapter.

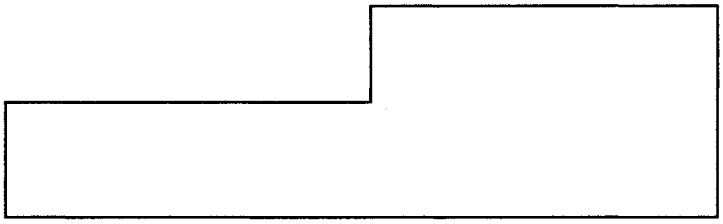


Figure 3.6 A multi-step workpiece used for the simulation

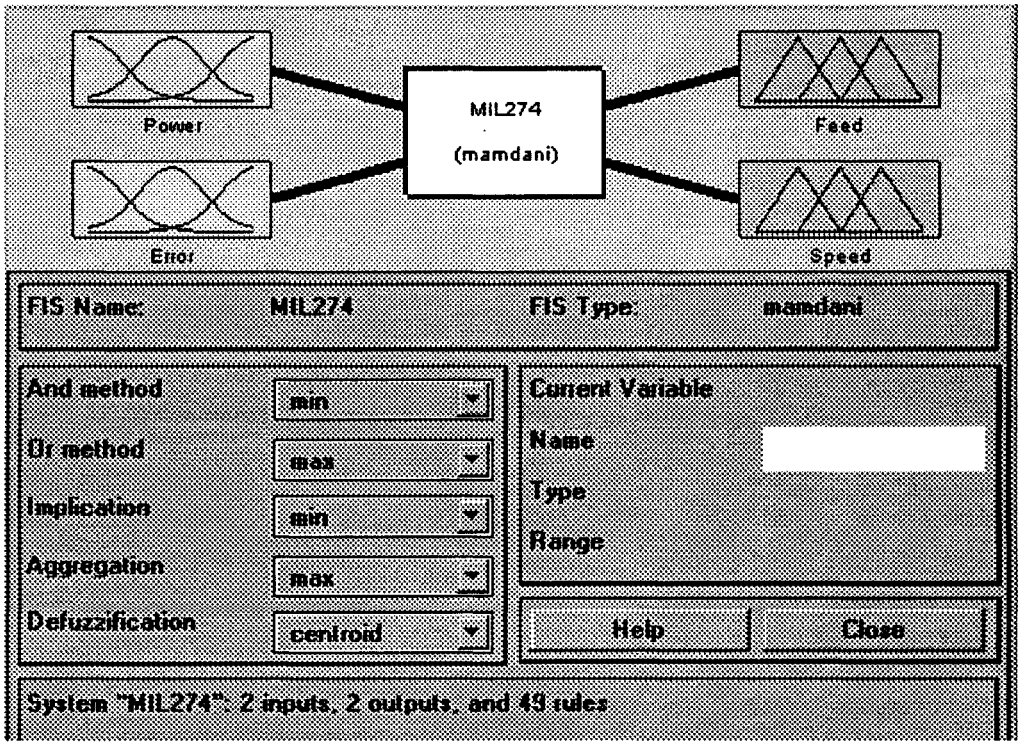


Figure 3.7 Fuzzy Inference system from MATLAB Fuzzy Toolbox used in simulation

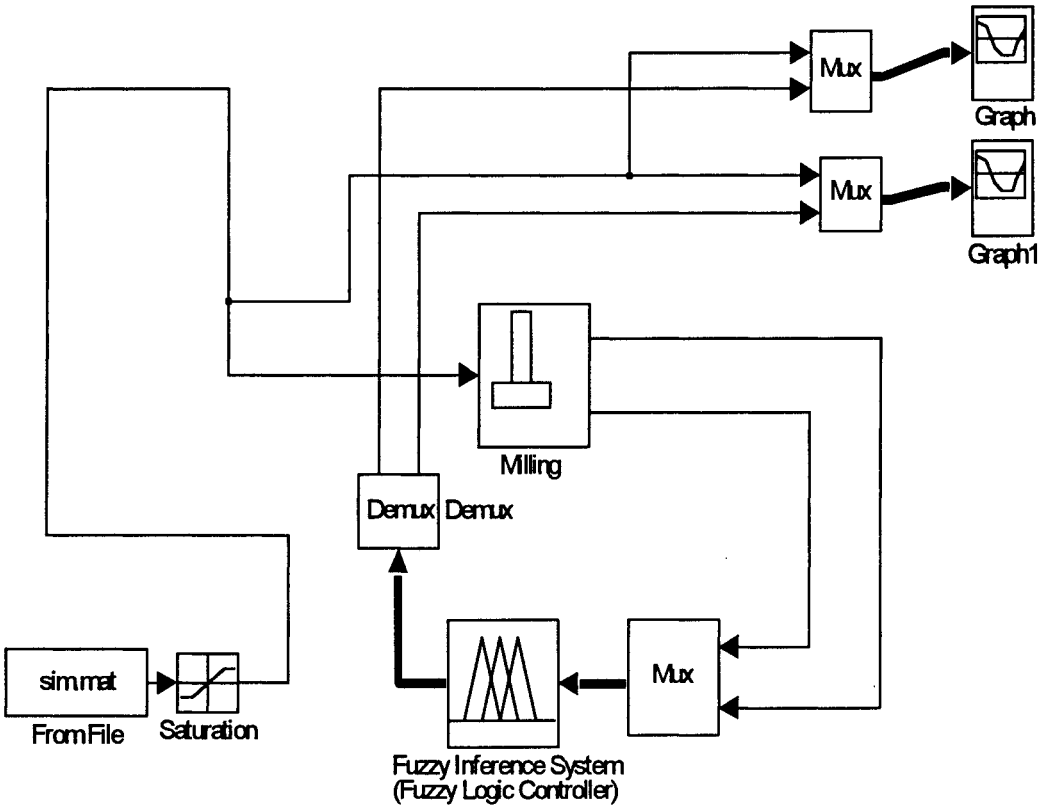


Figure 3.8 The fuzzy control system for constant spindle power simulation diagram

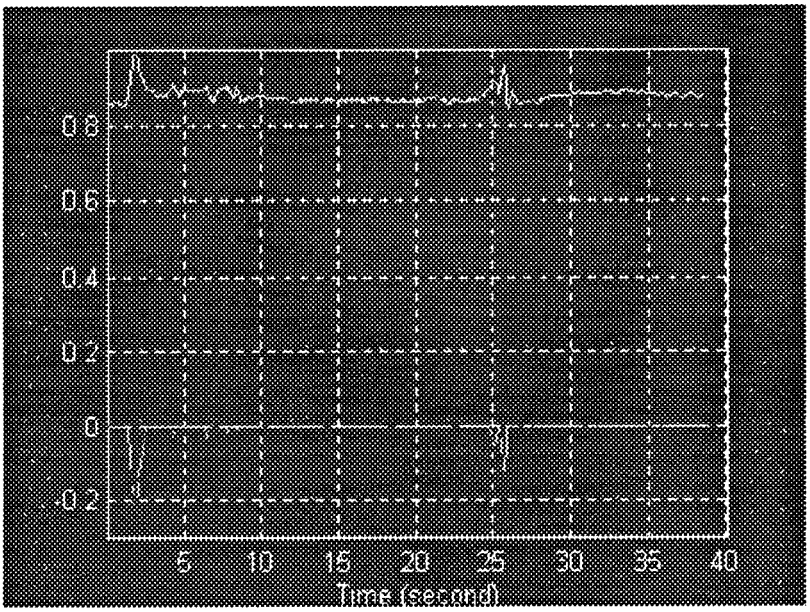


Figure 3.9 The simulation result for the feed rate deviation

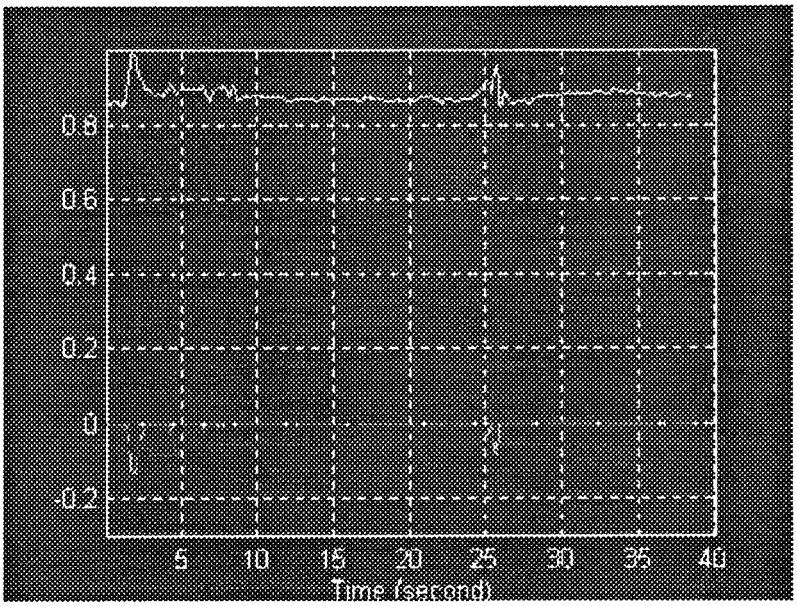


Figure 3.10 The simulation result for the spindle speed deviation

Chapter 4

Implementation of the Adaptive Fuzzy Controller

This chapter reports the implementation of the proposed adaptive fuzzy controller on a CNC milling machine. Experiments on various workpieces with different cutter immersion rates are also carried out to examine the performance of the controller.

4.1 Experimental Apparatus

The power control system consists of the following main hardware components:

1. a CNC Milling machine
2. a power sensor
3. a low-pass filter
4. an A/D and a D/A converter, and
5. a pentium PC

The following sections describe the role of each component as well as the overall milling control system.

4.1.1 Milling Machine

A servo 2000 vertical CNC milling machine built by the Servo Products Co. was used for implementation of the proposed controller. The spindle shaft is driven by a three-phase G3, 3HP AC motor. The machine has a gear box which provides 2 speed

systems, high and low. In the high gear the motor output power is transmitted to the spindle shaft directly from the spindle shaft hub, providing a range of spindle speed from 450 to 5100 RPM. In the low gear, the power from the spindle shaft hub is transmitted to the spindle shaft through a set of pulleys and gears. The spindle speed for the low gear ranges from 45 to 510 RPM.

The machine has three sliding axes, with the X (table) and Y (cross) axes being driven by axis drive motors and lead screws, and Z axis being operated manually. The axis drive motors are 3-phase Variable Reluctance stepping motors with resolution of 84 primary steps per revolution.

4.1.2 Power Sensor and Signal Processing

A load controls PH-3A power sensor is used to measure the spindle power signals through three cables from the spindle motor passing through the cells. The power sensor has a full-scale power capacity of 10HP and a voltage output in the range of 0 to 10 volts, leading to sensitivity of 1 V/HP. The output signals from the power sensor are filtered by a 4-th order Butterworth low-pass filter with cut-off frequency of 2Hz to eliminate noise due to cutter run out, belt oscillation and spindle motor speed (Fussell 1987, Stein and Wang 1996).

4.1.3 A/D, D/A Converter

The filtered signals from the power sensors are digitized by a 12 bit A/D converter, which is a part of an analog to digital and digital to analog converter PC card. The digital signals are the crisp inputs for the fuzzy logic controller. The outputs of the

controller, spindle speed and feedrate, are going back to the D/A section of the converter and then to the feedrate drive and spindle speed control box. The command feedrate, in a form of overriding percentage of the full scale feedrate and the command spindle speed in actual rpm are converted to an analog voltage signal of 0 to 5V and 0 to 10V respectively by the 12 bit D/A converter.

4.1.4 Feedrate and Spindle Speed Commands Implementation

The analog output signals from the controller to the feedrate and spindle speed drivers are implemented through the pendant and spindle speed control box. The feedrate can be changed from 0 to 150 percent of the full scale feedrate at the System Operation Console (SOC). The lower and higher speed limits of the spindle are set in the controller program via the PC.

4.2. Experimental Work

4.2.1 Experimental Setup

Figure 4.1 illustrates the experimental setup of the power control system. The experiments were conducted using a High Speed Steel (HSS) 14.3 mm end milling cutter with four helical flutes and 30° helix angles. Slot milling (i.e., full immersion) and ¾ immersion milling (as illustrated in Figure 4.2) with coolant was performed along Y-axis on 1018 cold rolled steel workpieces. Based on cutting tests with different sampling frequencies, a sampling interval of 0.26 seconds was selected.

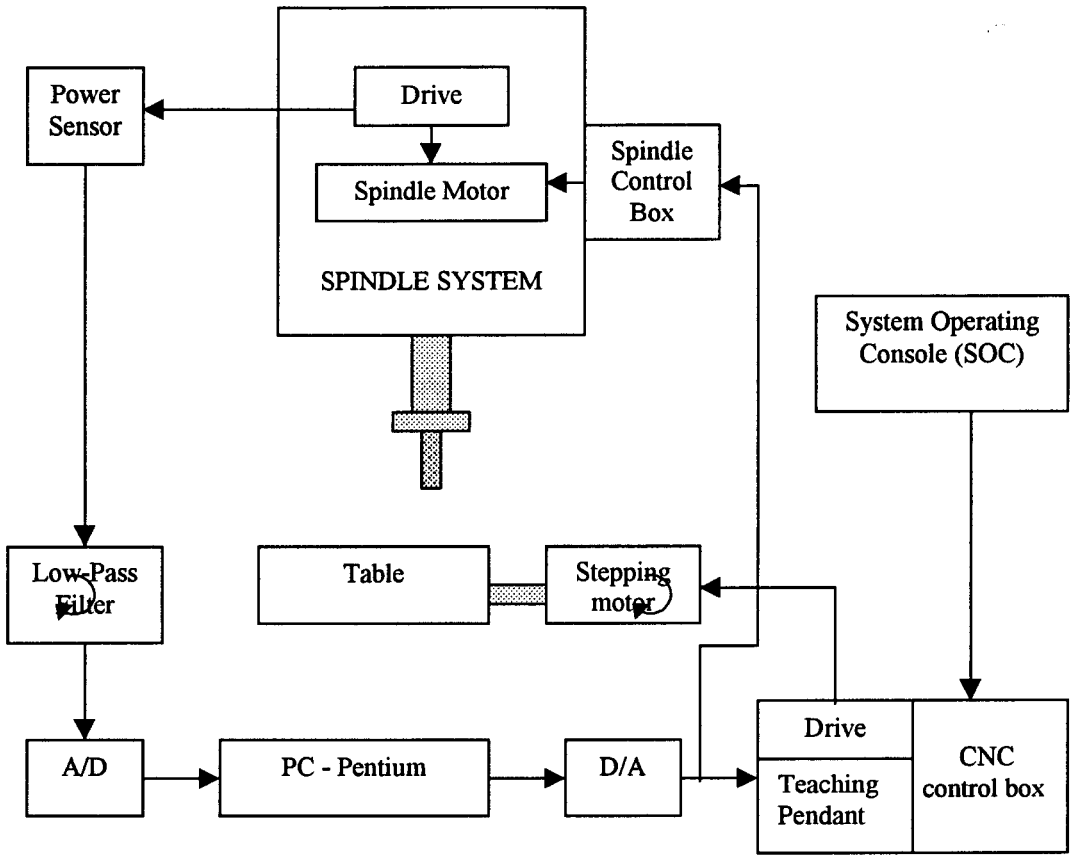


Figure 4.1 Experimental setup of the spindle power control system

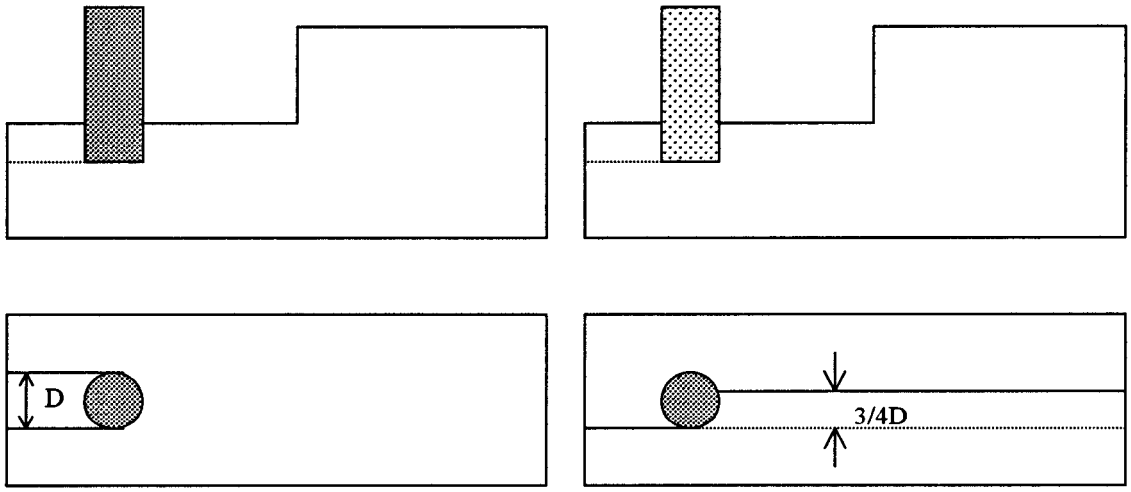


Figure 4.2 Schematic of slot-milling and ¾ immersion slot-milling

The maximum and minimum feedrate commands were set to 120 mm/min and 25 mm/min respectively. The upper spindle speed was set to 350 RPM and the lower limit is determined based on workpiece material and cutter specifications. In this study, the following relation is proposed to specify the lower limit of the spindle speed to avoid tool breakage:

$$\frac{f}{z\,v} \le f_{t(Max)} \tag{4-1}$$

where z is the number of cutter teeth, and $f_{t(Max)}$ is the maximum feed per tooth which is tool-workpiece dependent. This condition states that the feed/speed relation is restricted by the maximum allowable feed per tooth. According to a metal cutting handbook (Zhao 1992), for 1018 cold rolled material and HSS cutter, the value of $f_{t(max)}$ is 0.08 mm/tooth.

4.2.2 Overall Closed Loop of Process Control

The block diagram of the overall process control system for end milling process is illustrated in Figure 4.3. The system consists of three main components: adaptive fuzzy controller, the feed and spindle drive system, and the milling process. The adaptive controller is further composed of a feed back fuzzy control loop and a tuning loop. The details of the fuzzy large controller and the tuning mechanism have been explained in chapter 3.

4.3 Experimental Results

The main purposes of the experiments are to maintain the spindle power at the reference level by adjusting single-parameter (feedrate) or multi-parameter (feedrate and

spindle speed) and to compare the two adjustment strategies. Hence, two sets of experiments were accordingly conducted on three types of steel workpieces with different geometries. To examine the robustness of the controller, experiments are also carried out for partial immersion machining, variable immersion cutting, and aluminum cutting. The results of the experiments are presented in Figures 4.4 through 4.14.

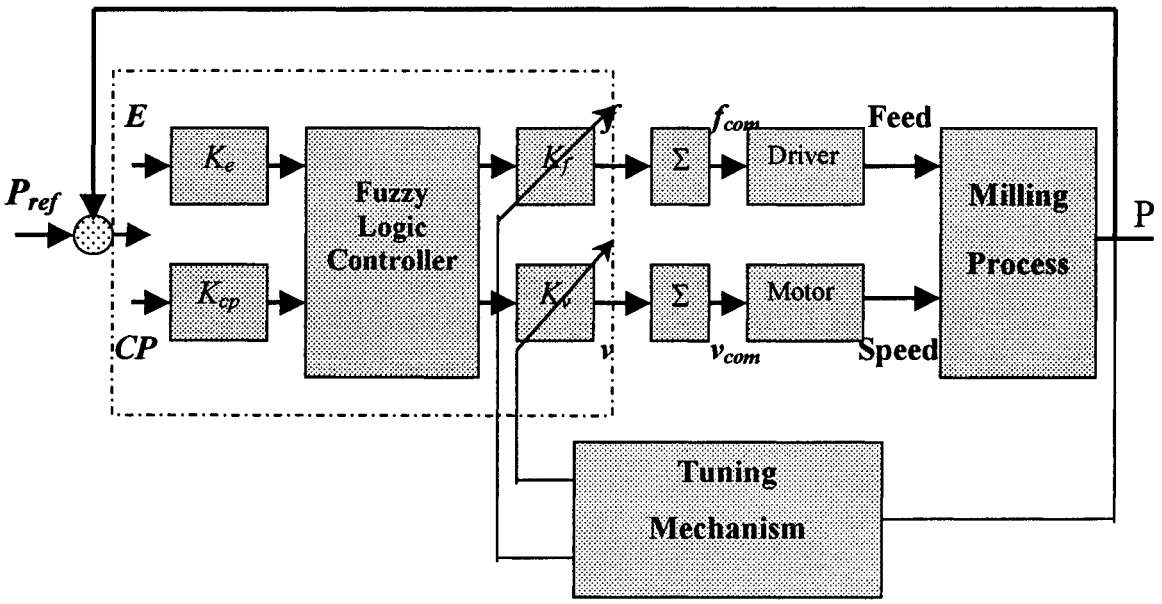


Figure 4.3 block diagram of the overall process control system for end milling process

The initial feedrate and spindle speed for all cases were chosen as 25mm/min and 300 rpm respectively, though selection of any other initial feed and spindle speed had no impact on the control performance so long as condition (4-1) is satisfied. The initial distance between the cutter and the workpiece was set to approximately 5 mm.

The adaptive fuzzy controller program was coded in C++ language for DOS operating system environment with Graphical User Interface (GUI). The user can select

tool size, workpiece material, sensor (for this thesis only power sensor was used), adjustment parameters (feedrate alone or both feedrate and spindle speed) from the menus. The program listing is given in Appendix C.

The experiment results are classified into six categories according to workpiece geometry, immersion rate, and workpiece material. The details are explained as follows.

Case i) full immersion steel cutting with curved change of depth of cut (Figures 4.4 and 4.5)

When cutting and control process started, the cutter was about 5 mm away from the workpiece. At the time the cutter reached the workpiece, the feedrate reached its maximum value (120 mm/min). From both figures it can be seen that the controller responds quickly to bring the power down to its reference point smoothly. However, Figure 4.4 displays higher overshoots as compared to Figure 4.5 when both feed rate and spindle speed are adjusted. The cutting time in Figure 4.5 was about 30 seconds less than the one for feedrate adjustment alone, representing a 22% of reduction in cutting time.

Case ii) full immersion steel cutting with sloping change of depth of cut (Figures 4.6 and 4.7)

Both figures show a very good power regulation. The feedrate and spindle speed decrease very smoothly as the depth of cut increases. The significant difference between the two processes was the cutting time. The cutting time for the process with dual-parameter adjustment was 36 seconds shorter than that with only feedrate adjustment, achieving almost a 25% reduction in cutting time. It is also observed that dual-parameter

adjustment has a better transient performance with less overshoot during the initial tool-part engagement period.

Case iii) full immersion steel cutting with step changes of the depth of cut (Figures 4.8 and 4.9)

Again it is seen that the spindle power was well regulated around the reference level. Though no significant overshoots are observed in both cases, dual-parameter adjustment shows a better transient performance during the engagement periods of the three steps. With dual-parameter adjustment the cutting time was 20 seconds less than the other case, approximately an 18% improvement in material removal rate.

Case iv) partial immersion steel cutting with curved, sloping, step changes of the depth of cut (Figures 4.10 to 4.12)

To examine the robustness of the controller, ¾ immersion cutting on the workpiece with all three curved, sloping and step changes in depth of cut was carried out. The results are displayed in Figures 4.10 to 4.12. As the advantages of dual-parameter adjustment over the single-parameter adjustment have been clearly demonstrated in Figures 4.4 to 4.9, the experiments were conducted only for the former case. Again, as observed in the full immersion tests, the controller shows a stable and smooth performance for all the three different part geometries. This indicates that the controller is not sensitive to cutting immersion rate change, though the change in immersion rate may vary the cutting dynamics. As expected, it is also observed that the cutting times have been reduced significantly due to the reduced immersion rates for all the three part

geometires. For the curved change in depth of cut (Figures 4.5 and 4.10) show a 24% cutting time reduction over the full immersion. Comparing Figures 4.7 and 4.11, for the sloping change in depth of cut indicates that cutting with ¾ immersion is about 33% faster than full immersion cutting. Similarly, a 28% cutting time reduction was obtained for parts with step changes in depth of cut (Figures 4.9 and 4.12).

Case v) variable immersion cutting (Figures 4.13)

To further examine the adaptiveness of the controller in response to the variation of immersion rate, an experiment was conducted using a workpiece with three step changes in immersion rate. The cutting started with full immersion followed by two consecutive step reductions in immersion rate as shown in Figure 4.13. The stable and smooth result as shown in Figure 4.13 demonstrates that the fuzzy controller is very robust even when the process is experiencing abrupt dynamics changes.

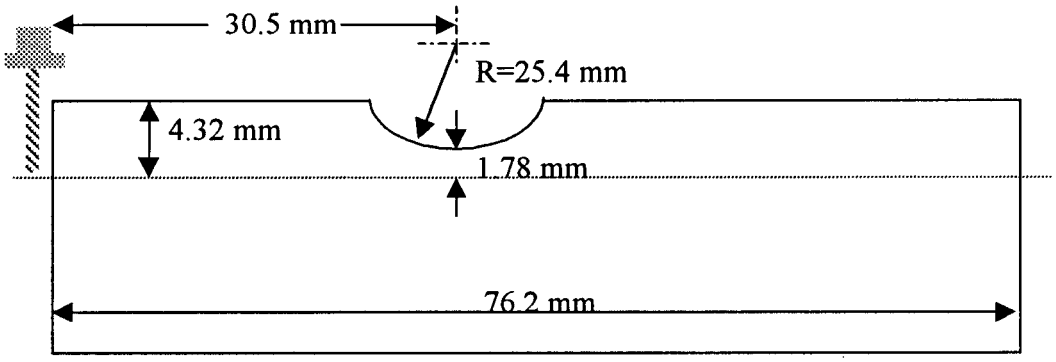
Case vi) full immersion aluminum cutting with step changes in the depth of cut (Figures 4.14)

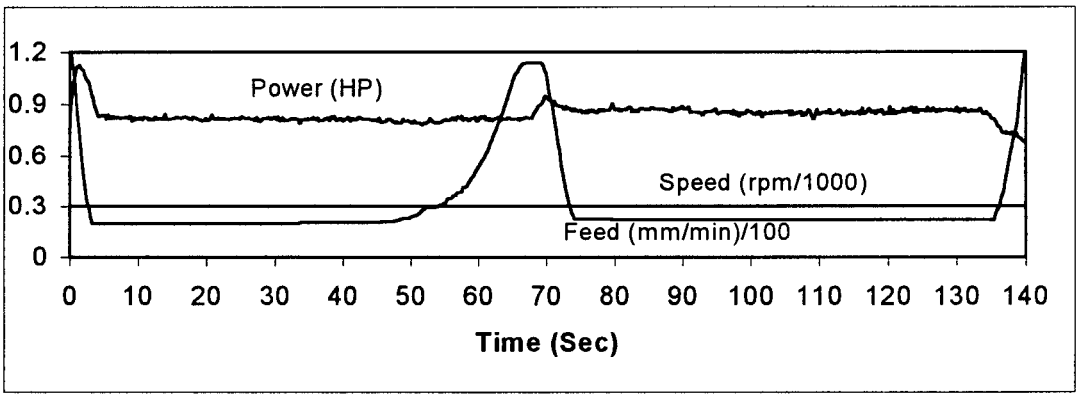
The purpose of conducting this experiment was to examine the effect of workpiece material properties on the controller performance. In this experiment, due to the fact that aluminum consumes much less cutting power than steel, the cutting power reference was set to 0.80 HP. The part geometry is shown in Figure 14. The cutting started with depth of cut of 3.125 mm and then changed to 6.25mm, and finally 9.375 mm at the end of cutting process.

As it can be seen from Figure 4.14, the controller performed very well in this experiment also. The overshoots for this part were higher than those observed for steel workpieces. That is probably caused by the abruptly increased contact area at each of engagement points, which contributes to a sudden increase of cutting power.

It is observed that the power level has been quickly brought back to the reference level after each of the overshoots without causing oscillation. This again demonstrates the good transient performance of the controller.

Though the process has experienced a few overshoots, the control process performed reasonably well. It is therefore fair to say that the controller is stable and robust even when the workpiece material properties have been changed substantially.





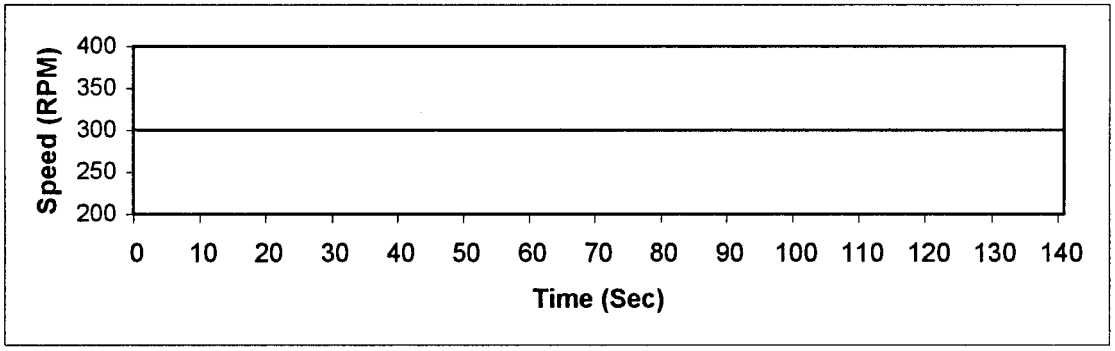
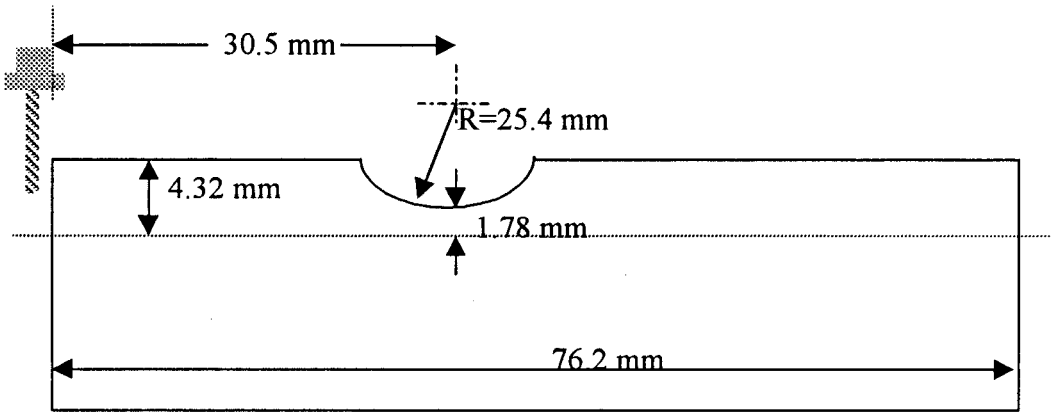
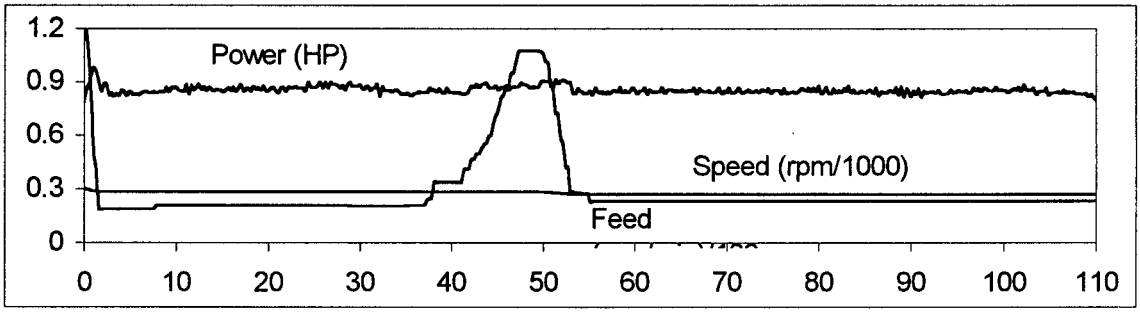


Figure 4.4 Experimental Results with variable feedrate and constant spindle speed for variable depth of cut (curved change, full immersion)





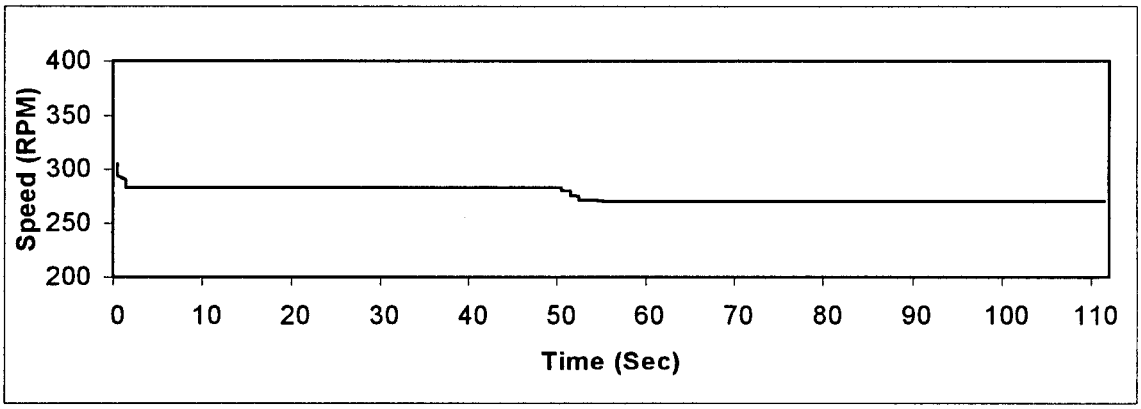
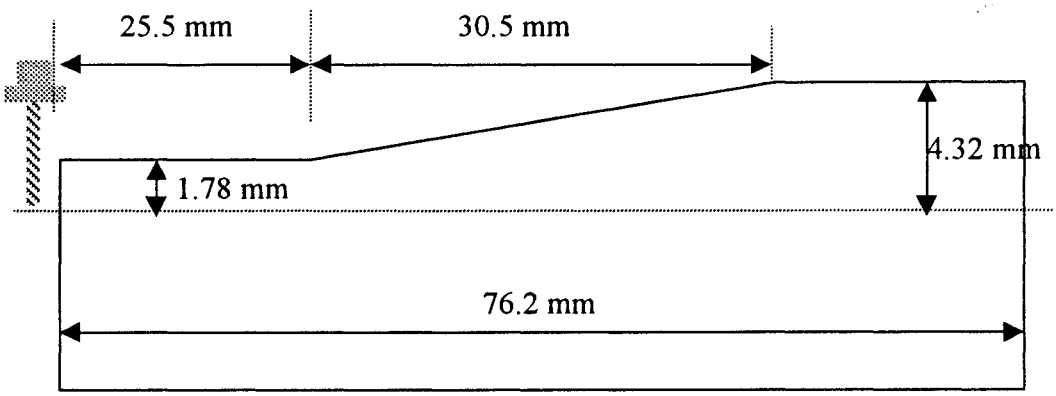
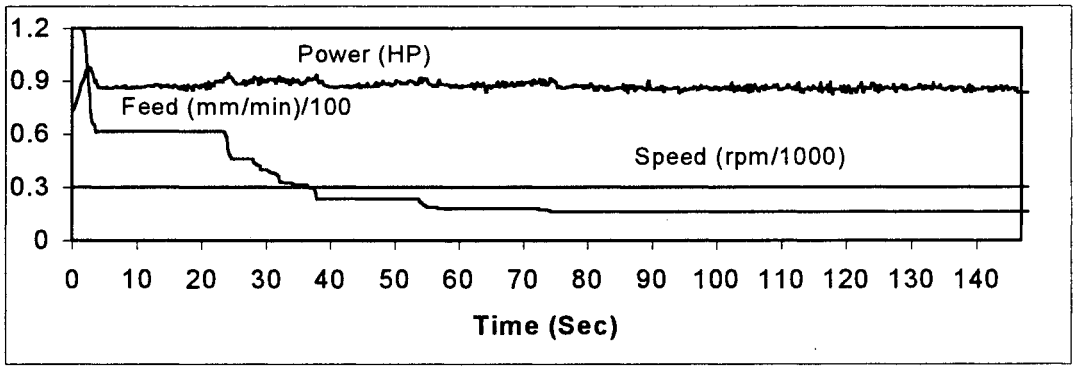


Figure 4.5 Experimental Results with variable feedrate and spindle speed for variable depth of cut (curved change, full immersion)





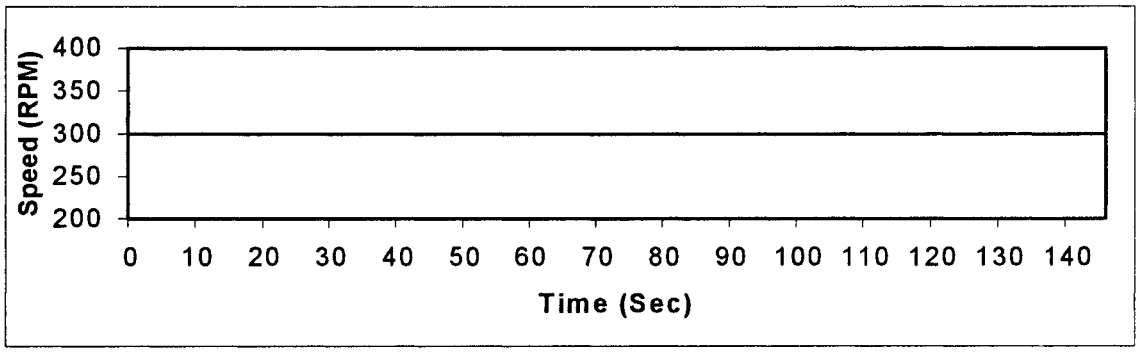
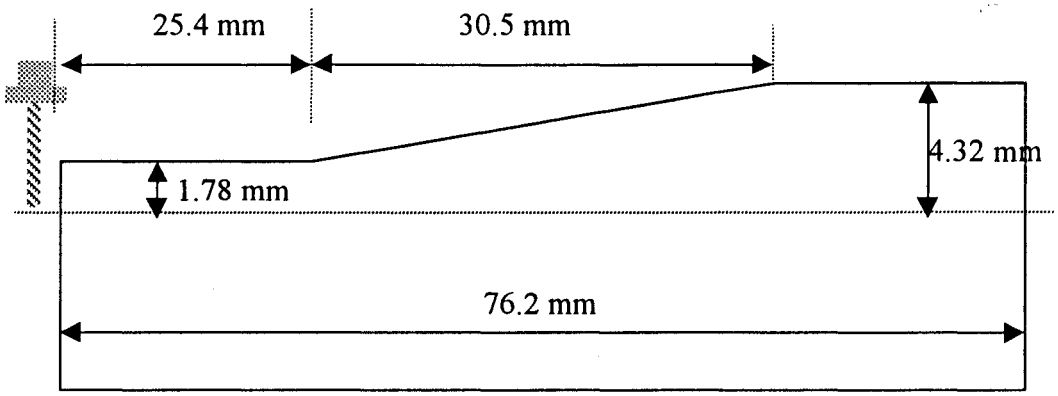
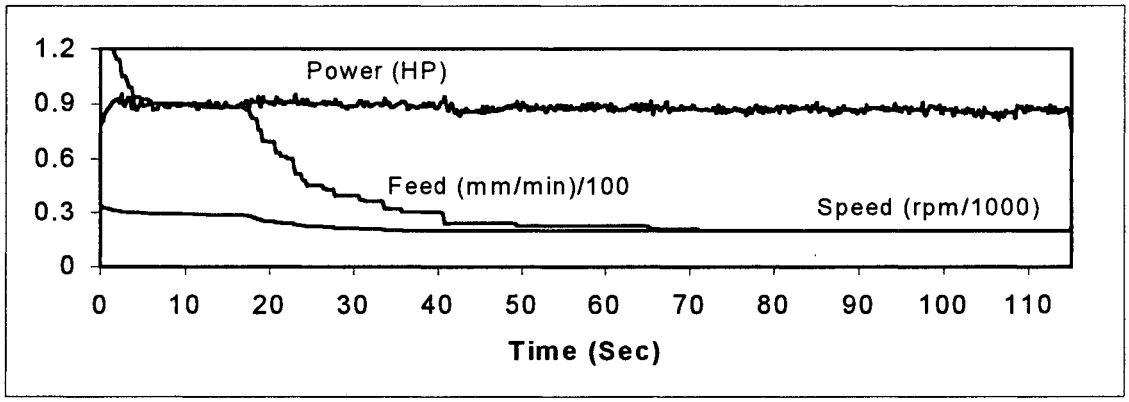


Figure 4.6 Experimental Results with variable feedrate and constant spindle speed for variable depth of cut (sloping change, full immersion)





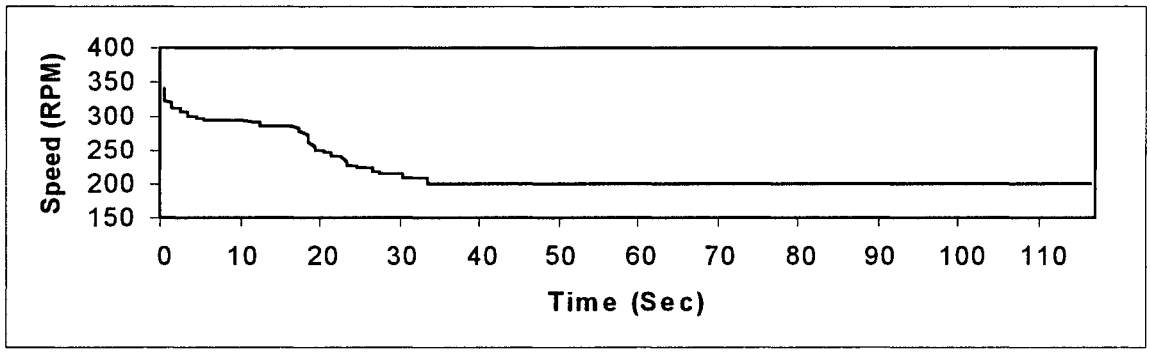
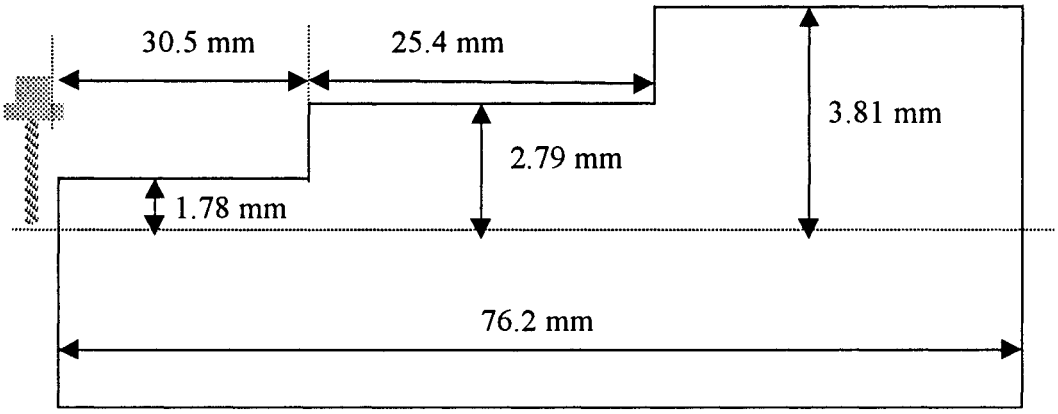
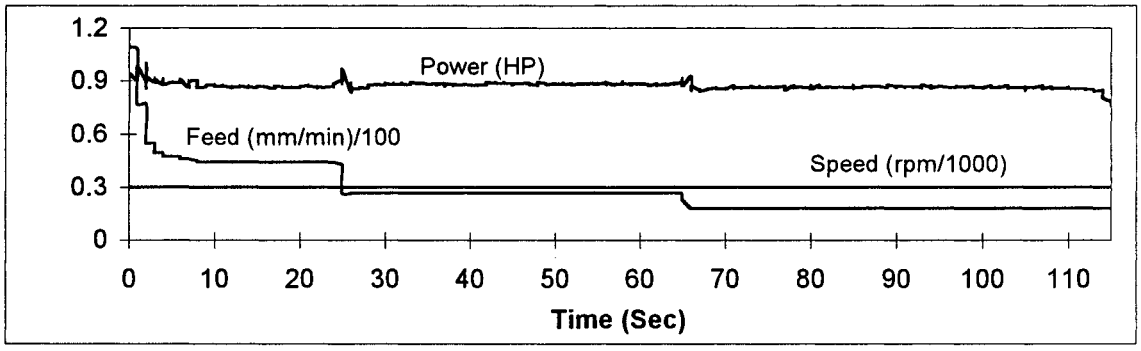


Figure 4.7 Experimental Results with variable feedrate and spindle speed for variable depth of cut (sloping change, full immersion)





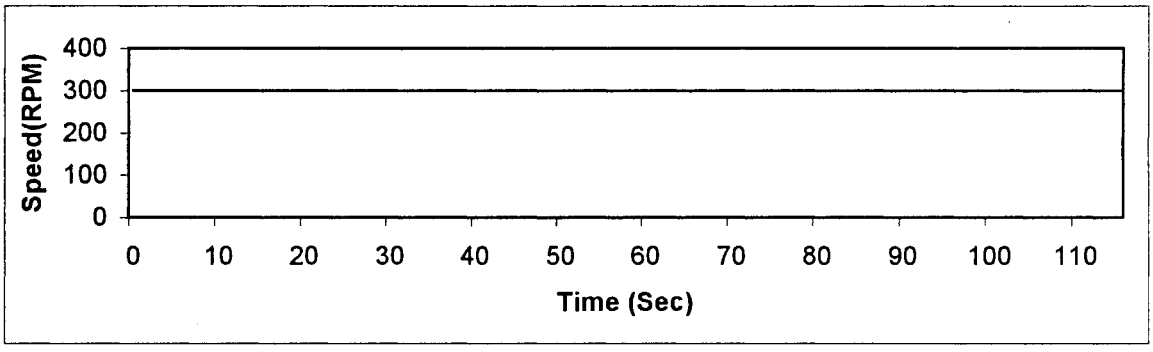
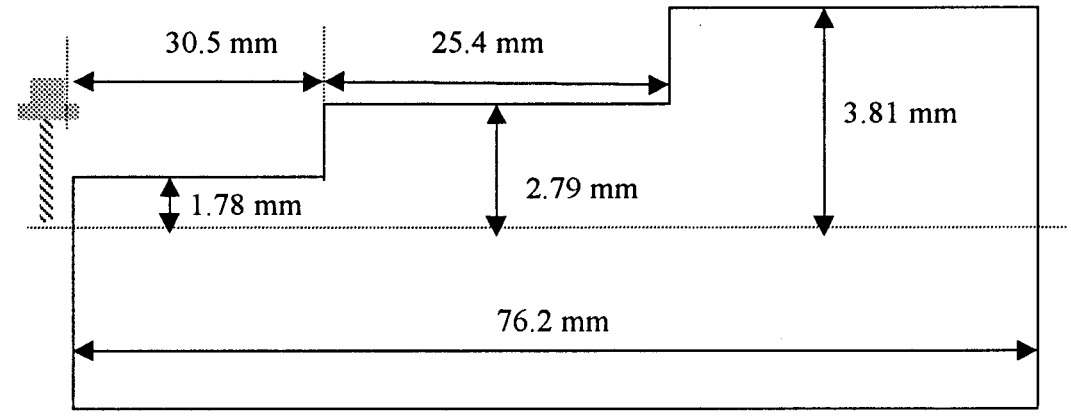
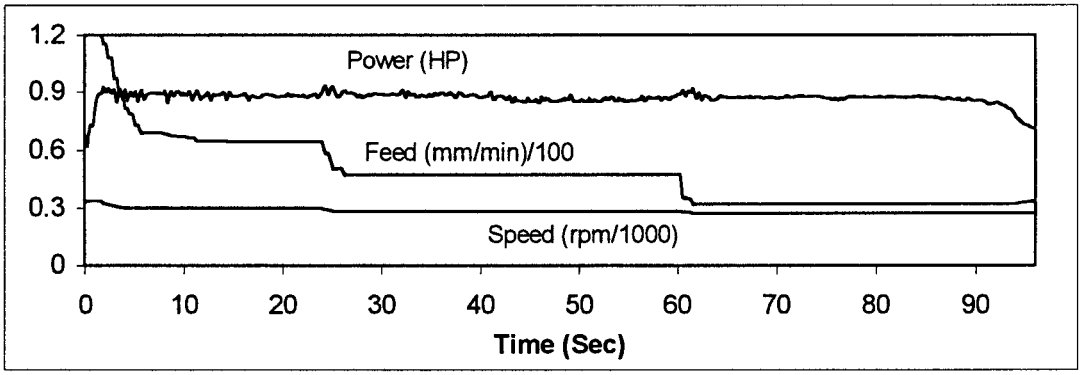


Figure 4.8 Experimental Results with variable feedrate and constant spindle speed for variable depth of cut (step change, full immersion)





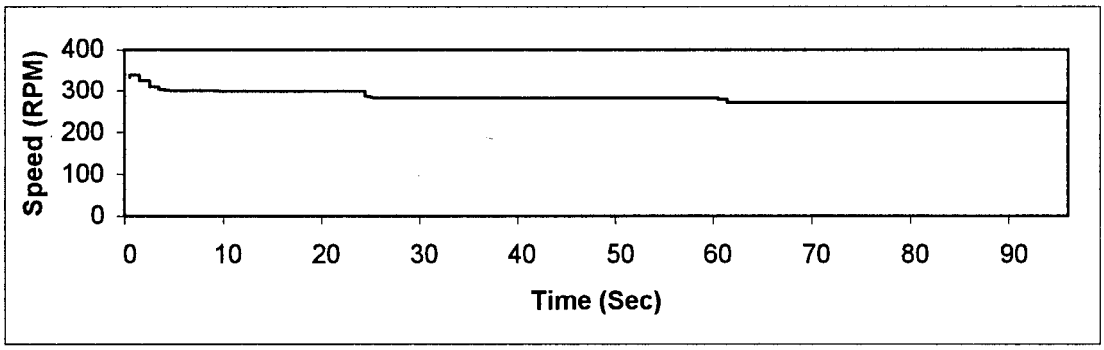
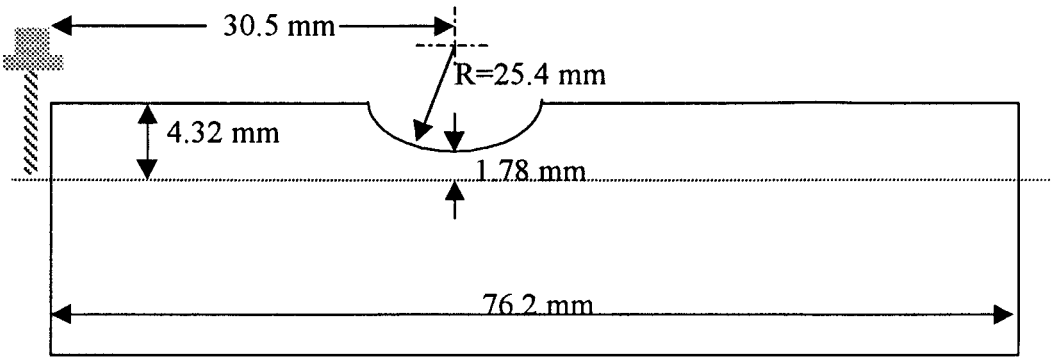
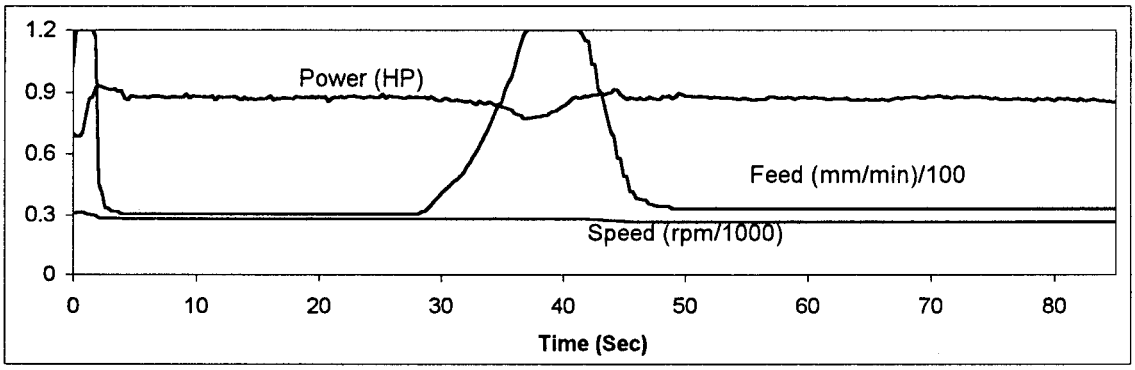


Figure 4.9 Experimental Results with variable feedrate and spindle speed for variable depth of cut (step change, full immersion)





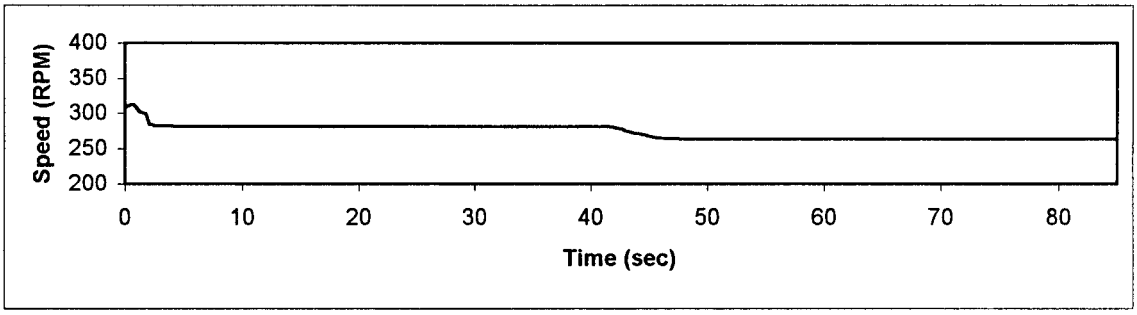
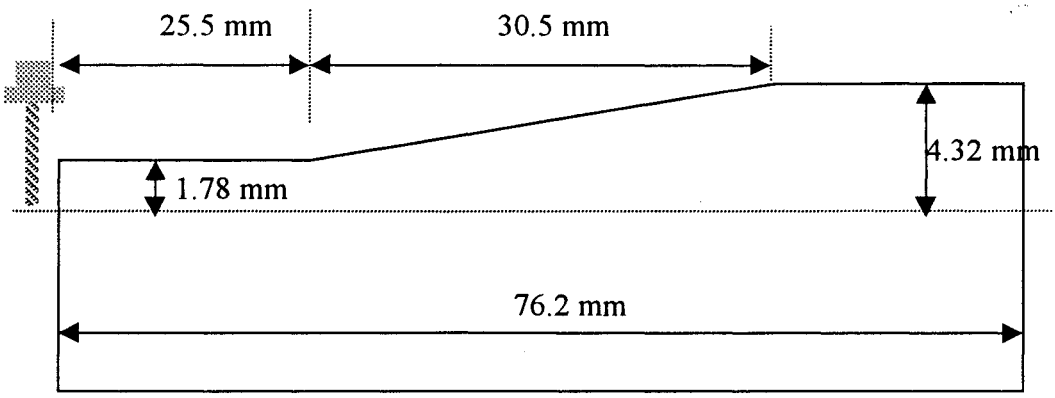
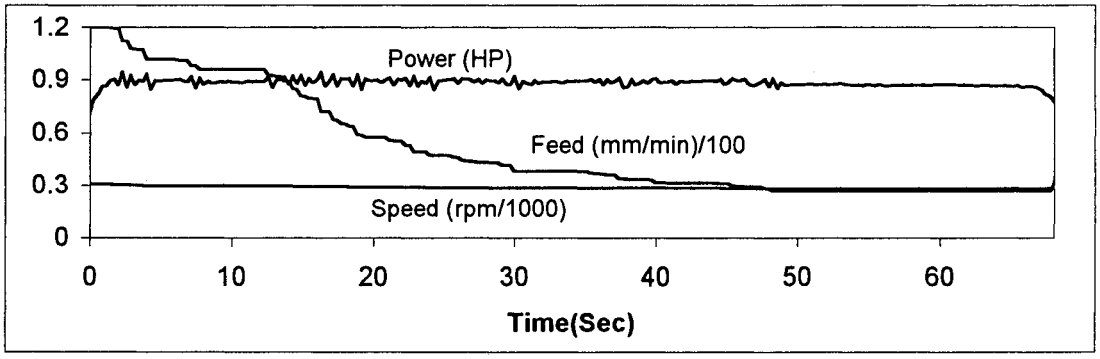


Figure 4.10 Experimental Results with variable feedrate and spindle speed for variable depth of cut (curved change, 3/4 immersion)





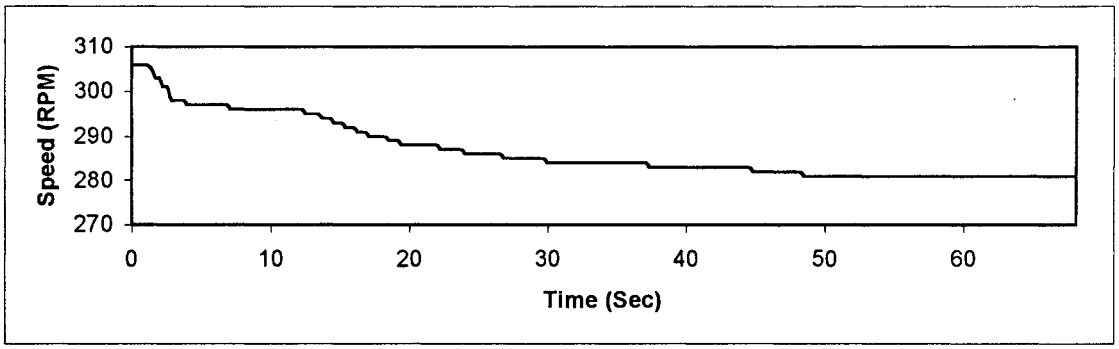
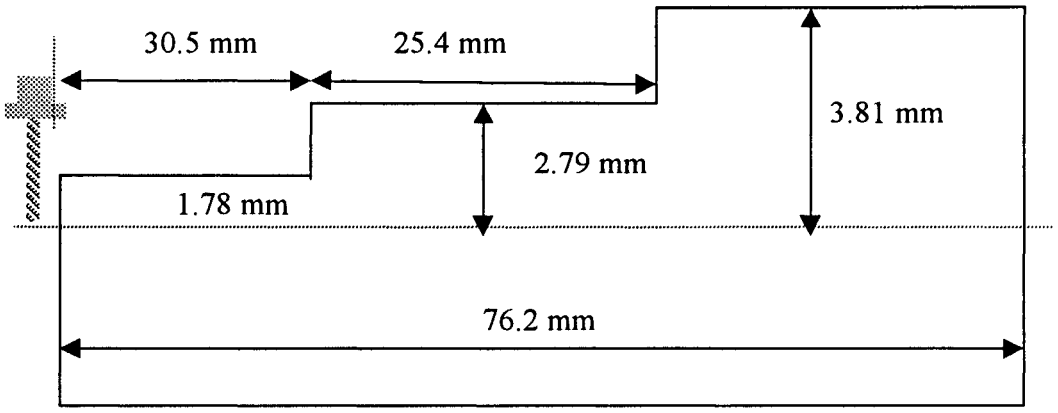
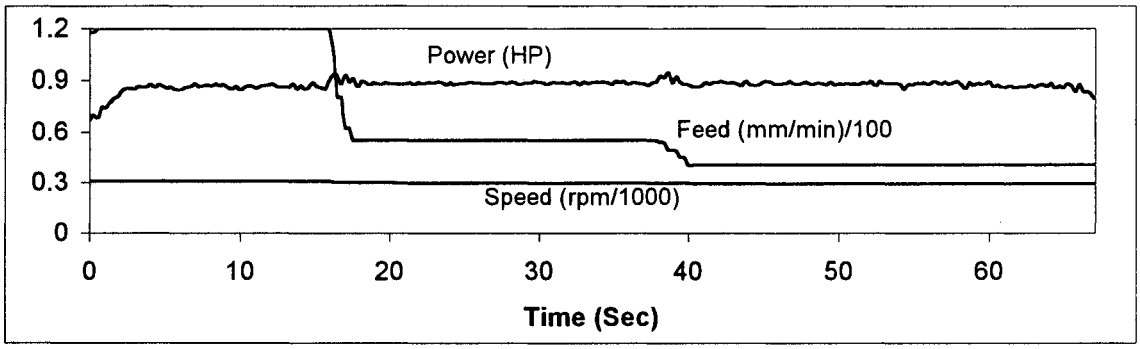


Figure 4.11 Experimental Results with variable feedrate and spindle speed for variable depth of cut (sloping change, 3/4 immersion)





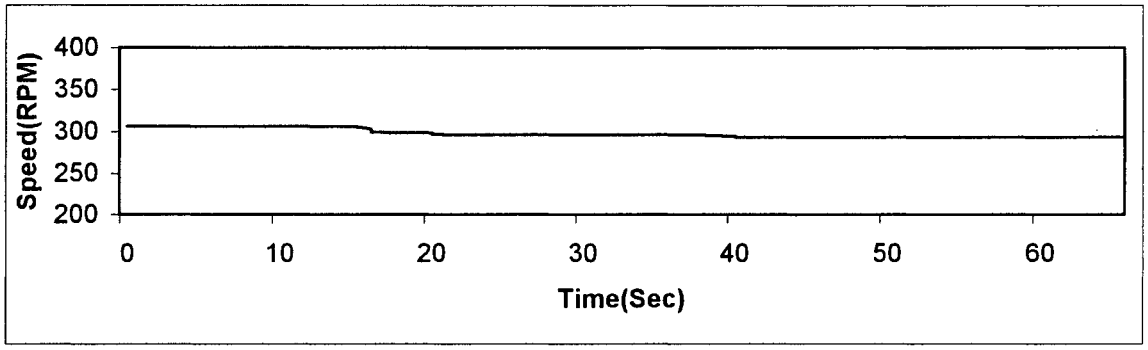
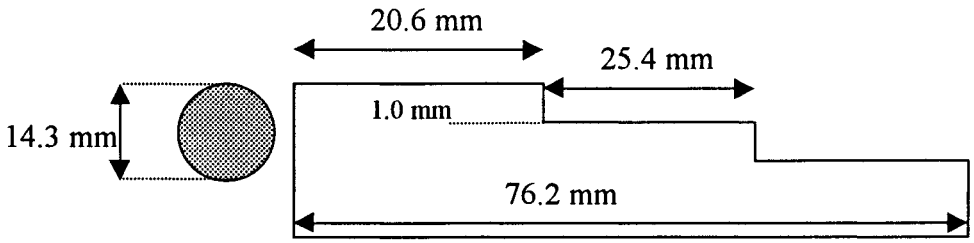
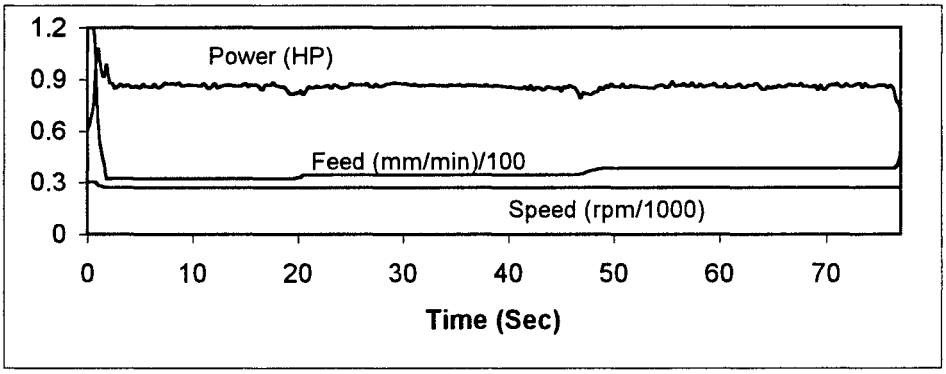


Figure 4.12 Experimental Results with variable feedrate and spindle speed for variable depth of cut (step change, 3/4 immersion)





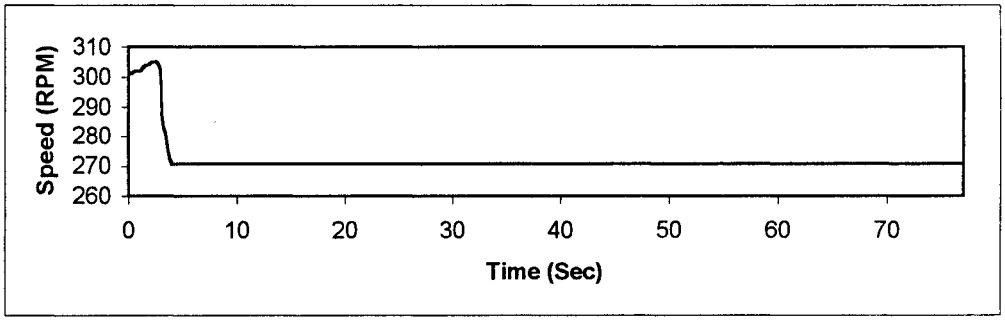
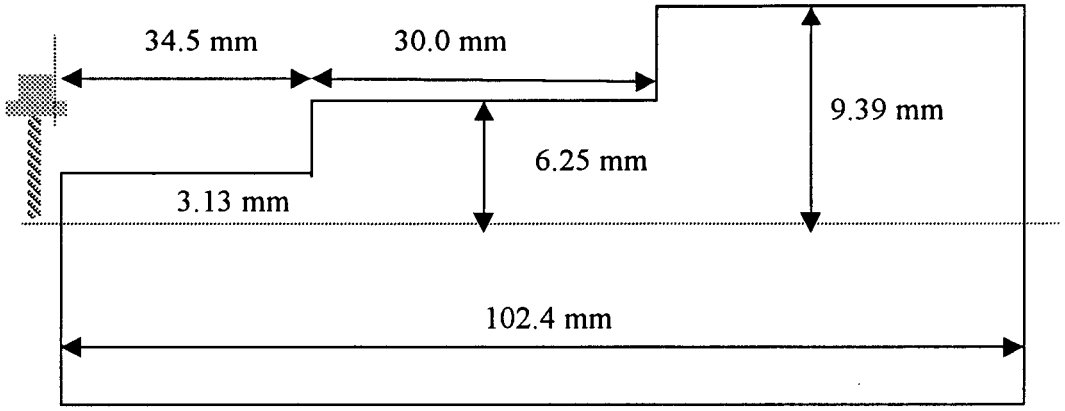
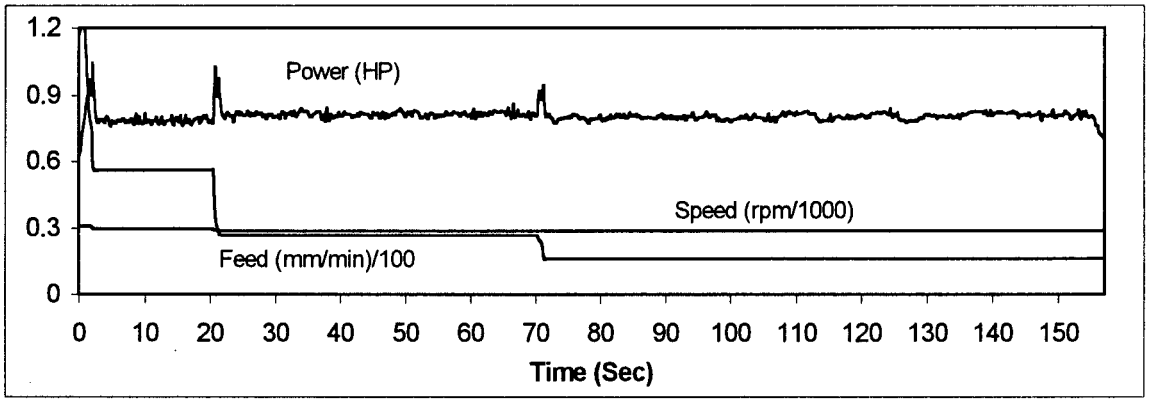


Figure 4.13 Experimental Results with variable feedrate and spindle speed for variable immersion size





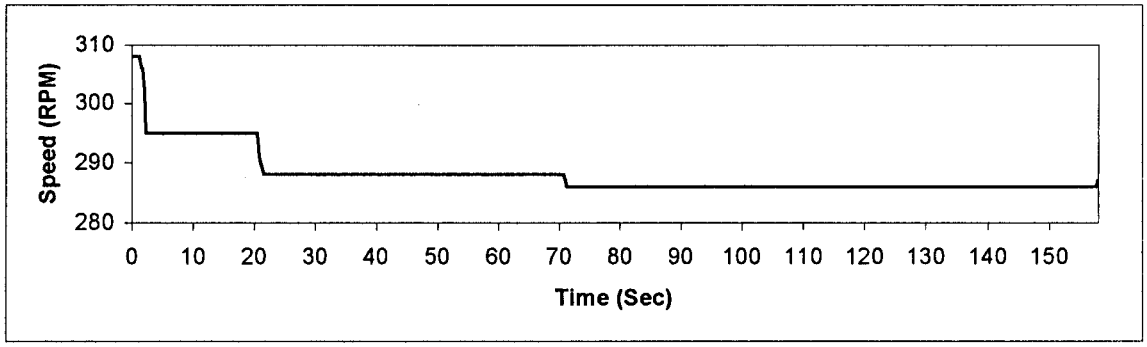


Figure 4.14 Experimental Results with variable feedrate and spindle speed for variable depth of cut (step change, Aluminum workpiece, full immersion)

Chapter 5

Conclusions and Future Research

5.1 Conclusions

In this study, an adaptive power controller has been developed. Based on the power error and power change inputs, the controller can maintain the spindle power around the reference level by adjusting both feedrate and spindle speed. A controller capable of adjusting both feedrate and spindle speed for end milling has not been reported in the accessible literature. The major advantages of this controller are summarized below:

1. *It is simple and modeling free.* As shown in the thesis, the controller does not require mathematical modeling and time-consuming system identification processes. This is particular important in today's manufacturing environment featuring frequent changes of machine-tool-workpiece combinations. The traditional controller developed based on a particular machine-tool-workpiece combination may soon become obsolete even before it is actually applied due to the lengthy development and implementation process.
2. *It controls the cutting processes based on spindle power signals rather than force signals and hence is easy to implement and cost-efficient.* Most of the reported controllers for end milling are force-based systems. The main drawbacks of such controllers include high investment cost, lengthy setup, limited part size, and direct

exposure of sensor to the harsh cutting environment, thus restricting their industrial implementation. The power-based controller can avoid all of these difficulties and provides better application opportunities.

3. *It can significantly reduce cutting time.* As demonstrated by the experimental results, adjusting both feedrate and spindle speed can further improve material removal rate up to 25% as compared to the commonly used single parameter adjustment. It also leads to less overshoots and better stability.
4. *It is robust.* The experimental results have clearly shown that the proposed controller is insensitive to part geometry change, immersion rate change, and part material property change. As a result, the control system can be used for a wide range of machining conditions.
5. *Its adaptation mechanism is based on on-line scaling factor tuning and hence it does not need off-line learning.* Many fuzzy controllers include a neural network module as their adaptation component, which require a great amount of time for training and testing. With the scaling factor tuning, such training or testing processes can be eliminated and the controller can be directly applied to many new machining processes.

Though some minor fine-tuning may be needed when the controller is implemented on different machines, the above does indicate very good industrial application potential of the proposed controller.

5.2 Future Research

In the progress of this study, it is realized that research in the following directions could be worth exploring:

1. *Integration of a chatter suppression module into the proposed controller.* It is observed that adjusting cutting parameter sometimes causes shatter. Severe chatter could halt the adjustment process and thus the control goal may not be achieved. This reveals a need to develop a chatter suppression module for the control system.
2. *On-line selection of initial values of the gain factors.* When the production process involves frequent changes in part materials, cutter and machines, it will be tedious to specify the initial values for gain factors. To this end, an on-line module is needed to select such initial values automatically.
3. *Control based on feed power.* In this study, feedrate is adjusted indirectly based on spindle power signals. Adjusting feedrate based on the directly measured table feed power could be more accurate.

References

1. Aguero, E., Rodriguez, C., Alique, J.R. and Ros, S., Fuzzy Model of Cutting Process on a Milling Machine. *Intelligent Systems Engineering*, 1994, pp. 236-244.

2. Batur, C. and Kasparian, V., Fuzzy Adaptive Control. *International Journal of Systems Science*, Vol. 24, No. 2, 1993, pp. 301-314.

3. Berenji, H.R., An Intelligent controller based on approximate reasoning and reinforcement learning. *Proc. of IEEE Int. Symposium on Intelligent Control*, Albany, NY, 1989, pp. 69-93.

4. Brown, M. and Harris, C., *Neurofuzzy Adaptive Modeling and Control*. New York, Prentice-Hall, 1994.

5. Chiang, S.T., Liu, D.I., Lee, A.C. and Chieng, W.H., Adaptive Control Optimization in End Milling using Neural Networks. *International Journal of Machine Tools and Manufacturing*, Vol. 34, No. 5, 1995, pp. 637-660.

6. Chen, YueDong, Hui, Albert and Du, R., A Fuzzy Expert System for the Design of Machining Operations. *International Journal of Machine Tools and Manufacturing*, Vol. 35, No. 12, 1995, pp. 1605-1621.

7. Christian, E., *LC Filters: Design, Testing and Manufacturing*. New York: John-Willey & Sons Inc., 1983.

8. Danai, K. and Ulsoy, A.G., A Dynamic State Model for On-line Tool Wear Estimation in Turning. *ASME Journal of Engineering for Industry*, Vol. 109, Nov. 1987, pp. 396-399.

9. Driankov, D., Hellendoorn, H. and Reinfrank, M., *An Introduction to Fuzzy Control*. Berlin: Spring-Verlag, 1993.

10. Du, R.X., Elbestawi, M.A. and Li, S., Tool Condition Monitoring in Turning using Fuzzy Set Theory. *International Journal of Machine Tools and Manufacturing*, Vol. 32, No. 6, 1992, pp. 781-796.

11. Fussell, B.K., *Modeling and Adaptive Force Control of End Milling Operations*. Ph.D. thesis, Ohio State University. 1987.

12. Goodwin, G.C. and Sin, K.S., *Adaptive Filtering, Prediction and Control*. Englewood, New Jersey: Prentice-Hall Inc., 1984.

13. Gulley, N. and Jang, R., *Fuzzy Logic Toolbox for Use With MATLAB*. Natick, MA: The MathWorks Inc., 1995.

14. Haber, R., Jose, R. Alique, S., Clodeinir, R., Fuzzy Supervisory Control of End Milling Process. *Information Sciences* Vol. 89, 1996, pp.95-106.

15. Han, Z.X., *Implementation of Power and Force Adaptive Control of the End Milling Process*. M.A.Sc. Thesis, University of Ottawa, 1996.

16. Hsu, P.L and Hsieh, M.Y., Applications of Self-Tuning Control on Industrial CNC Machines. *International Journal of Machine Tools and Manufacturing*, Vol. 34, No. 6, 1994, pp. 859-877.

17. Huber, J. and Centner, R., Test Results with an Adaptively Controlled Milling Machine. ASTME Paper No. MS68-638, 1968.

18. Johnson, D.E., *Introduction to Filter Theory*, Eaglewood. New Jersey: Prentice-hall Inc., 1976.

19. Kaufman, A., *Introduction to the Theory of Fuzzy Subsets*. Vol. 1, New York, Academic Press, 1975.

20. Kaufman, A. and Gupta, M., *Fuzzy Mathematical Models in Engineering and Management Science*. Amsterdam, North-Holland, 1988.

21. Kistler Instrument Corporation, *Dual Mode Amplifier Type 5010A Operating Instructions*. 1994.

22. Kistler Instrument Corporation, *3-Component Dynamometer Type 9257B, Calibration Sheets*. 1993.

23. Kistler Instrument Corporation, *Quartz 3-Component Dynamometer Type 9257B, Operating Instructions*. 1993.

24. Kistler Instrument Corporation, *Dual Mode Amplifier Model 5010A10, Certificates of Calibration*. 1995.

25. Koren, Y., *Computer Control of Manufacturing Systems*. New York: McGraw-Hill Book Co., 1983.

26. Koren, Y. and Masory, O., Adaptive Control with Process Estimation. *Annals of CIRP*, Vol. 30, No. 1, 1981, pp. 373-376.

27. Kosko, B., *Neural Networks and Fuzzy Systems*. England Cliffs, NJ, Prentice-Hall, 1992.

28. Landau, I.D. and Lozano, R., Unification of Discrete Time Explicit Model Reference Adaptive Control Designs. *Automatica*, Vol. 17, No. 4, 1981, pp. 593-611.

29. Lauderbaugh, L.K. and Ulsoy, A.G., Dynamic Modeling for Control of the Milling Process. *Journal of Engineering for Industry*, Vol.110, 1988, pp.367-375.

30. Lauderbaugh, L.K. and Ulsoy, A.G., Model Reference Adaptive Force Control in Milling. *Journal of Engineering for Industry*, Vol.111, 1989, pp. 13-21.

31. Lent, A.F. and Miatkowski, S., *Practical Applications Circuits Handbook*. San Diego, California: Academic Press., 1989.

32. Load Controls Inc., *PH-3A, PH-1000A Power Cell Power Transducer, Installation*. 1993.

33. Mallampati, D., and Shenoi, S., Adaptive Fuzzy Logic Controllers. *4th International Conference on Industrial and Engineering Application of Artificial Intelligence*, 1991, pp. 62-70.

34. Mandl, M., *Solid-State Circuit Design User's Manual*. Reston, Virginia: Reston Publishing Co., 1977.

35. Milner, D.A., Controller System Design for Feedrate Control by Deflection Sensing of a Machining Process. *International Journal of Machine Tool Design and Research*, Vol. 15, 1975, pp. 19-30.

36. Rahman, M, Zhou, Q. and Hong, G.S., On-line Cutting State Recognition in Turning using a Neural Network. *International Journal of Advanced Manufacturing Technology*, Vol. 10, 1995, pp. 87-92.

37. Ralston, P.A.S., Stoll, K.E. and Ward, T.L., Fuzzy Logic Control of Chip Form During Turning. *Computers Industrial and Engineering*, Vol. 22, No. 3, 1992, pp. 223-230.

38. Servo Products Inc., *2000/3000 Mill, System Operation and Programming*. Preliminary Manual, 1993, Version 1.0.

39. Stute, G. and Goetz, F.R., Adaptive Control System for Variable Gain ACC Systems. *Proceedings of the 16th International Machine Tool Design and Research Conference*, 1976, pp. 117-121.

40. Tarng, Y.S., Automatic Generation of a Fuzzy Rule Base for Constant Turning Force. *Journal of Intelligent Manufacturing*, Vol. 7, 1996, pp. 77-84.

41. Tarng, Y.S. and Cheng, S.T., Fuzzy Control of Feed Rate in End Milling Operations. *International Journal of Machine Tools and Manufacturing*, Vol. 33, No. 4, 1992, pp. 643-650.

42. Tarng, Y.S., Hseih, Y.W. and Li, T.C., Automatic Selection of Spindle Speed for Suppression of Regenerative Chatter in Turning. *International Journal of Machine Tools and Manufacturing*, Vol. 11, 1996, pp. 12-17.

43. Tarng, Y.S. and Hwang, S.T., Adaptive Learning Control of Milling Operations. *Mechatronics*, Vol. 5, No. 8, 1995, pp. 937-948.

44. Tarng, Y.S. and Wang, Y.S., An Adaptive Fuzzy Control System for turning Operations. *International Journal of Machine Tools and Manufacturing*, Vol. 33, No. 6, 1993, pp.761-771.

45. Tarng, Y.S. and Wang, Y.S., A New Adaptive Controller for Constant Turning Force. *International Journal of Advanced Manufacturing Technology*, Vol. 9, 1994, pp. 211-216.

46. Tomizuka, M. and Langari, G., Fuzzy Linguistic Model Based Control. *5th IEEE International Symposium on Intelligent Control*, 1990, pp. 620-625.

47. Tomizuka, M., Oh J.H. and Dornfeld, D.A., Model Reference Adaptive Control of the Milling Process. *Control of Manufacturing Processes and Robotics Systems*, D.E. Hardt and W.J. Books, eds., ASME, New York, 1983, pp. 55-63.

48. Ulsoy, A.G., Koren, Y. and Rasmussen, F., Principle Developments in the Adaptive Control of Machine Tools. *Journal of Dynamic Systems, Measurement, and Control*, Vol. 105, 1983, pp. 107-112.

49. Wang, L. and Langari, R., Complex Systems Modeling via Fuzzy Logic. *Proceedings of the 33rd Conference on Decision and Control*, 1994, pp. 4136-4141.

50. Williams, A.B. and Taylor, F.J., *Electronic Filter Design Handbook: LC, Active, and Digital Filters*. New York: McGraw-Hill Publishing Co. 1988.

51. Yeh, Z., Tarng, Y. and Nian, C., A Self-Organizing Neural Fuzzy Logic Controller for Turning Operations. *International Journal of Machine Tools and Manufacturing*, Vol. 35, No. 10, 1995, pp. 1363-1374.

52. Zadeh, L.A., Fuzzy sets. *Information and Control*, Vol. 8, 1965, pp. 338-353.

53. Zhao, R., *Handbook for Machinist*. Shanghai Science and Technology Press, 1992, 3rd Edition.

Appendix A

Technical Data of Experimental Equipments

A.1 Technical Data of Load Controls PH-3A Power Sensor

Full scale capacity:	10 HP
Frequency:	3 Hz to 1 kHz
Response:	15 milliseconds
Accuracy:	2.5% of full scale
Compliance:	6 Volts
Impedance:	10 Volt output: 2000 ohm minimum impedance
	4- 20 MA output: 300 ohm maximum connected impedance

A.2 Technical Data of PCL-812 Multi-Lab Card

A.2.1 Analog Input (A/D Converter)

Channels:	16 Single-ended
Resolution:	12 bits

Input Range:	Bipolar: +/- 10V, +/- 5V, +/- 2V, +/- 1V
Conversion type:	Successive Approximation
Converter:	HADC574Z
Linearity:	+/- 1 bit

A.2.1 Analog Input (A/D Converter)

Channels:	2 channels
Resolution:	12 bits
Output Range:	0 to +5V with fixed –5V reference, +/- 10V with external DC or AC reference
Conversion type:	12 bit monolithic multiplying
Converter:	AD7541AKN
Linearity:	+/- 1/2 bit

Appendix B

Listing of the Matlab and Simulink for Process Control Simulation

```
function [ret,x0,str,ts,xts]=slmil(t,x,u,flag);
%SLMIL      is the M-file description of the SIMULINK system named SLMIL.
%       The block-diagram can be displayed by typing: SLMIL.
%
%       SYS=SLMIL(T,X,U,FLAG) returns depending on FLAG certain
%       system values given time point, T, current state vector, X,
%       and input vector, U.
%       FLAG is used to indicate the type of output to be returned in SYS.
%
%       Setting FLAG=1 causes SLMIL to return state derivatives, FLAG=2
%       discrete states, FLAG=3 system outputs and FLAG=4 next sample
%       time. For more information and other options see SFUNC.
%
%       Calling SLMIL with a FLAG of zero:
%       [SIZES]=SLMIL([],[],[],0),  returns a vector, SIZES, which
%       contains the sizes of the state vector and other parameters.
%               SIZES(1) number of states
%               SIZES(2) number of discrete states
%               SIZES(3) number of outputs
%               SIZES(4) number of inputs
%               SIZES(5) number of roots (currently unsupported)
%               SIZES(6) direct feedthrough flag
%               SIZES(7) number of sample times
%
%       For the definition of other parameters in SIZES, see SFUNC.
%       See also, TRIM, LINMOD, LINSIM, EULER, RK23, RK45, ADAMS, GEAR.

% Note: This M-file is only used for saving graphical information;
%       after the model is loaded into memory an internal model
%       representation is used.
```

```
% the system will take on the name of this mfile:
sys = mfilename;
new_system(sys)
simver(1.3)
if (0 == (nargin + nargout))
    set_param(sys,'Location',[36,52,617,417])
    open_system(sys)
end;
set_param(sys,'algorithm',     'RK-23')
set_param(sys,'Start time',    '0.0')
set_param(sys,'Stop time',     '1000')
set_param(sys,'Min step size', '0.1')
set_param(sys,'Max step size', '0.1')
set_param(sys,'Relative error','1e-3')
set_param(sys,'Return vars',   '')
set_param(sys,'Load callback',['MILL=readfis("MILL");']);
set_param(sys,'AssignSampleTimeColors','on');
set_param(sys,'AssignWideVectorLines','on');


%    Subsystem 'Milling'.

new_system([sys,'/','Milling'])
set_param([sys,'/','Milling'],'Location',[4,42,632,472])

add_block('built-in/Inport',[sys,'/','Milling/in_1'])
set_param([sys,'/','Milling/in_1'],...
                'position',[40,260,60,280])

add_block('built-in/Constant',[sys,'/','Milling/Constant'])
set_param([sys,'/','Milling/Constant'],...
                'Value','.85',...
                'position',[45,335,65,355])

add_block('built-in/Sum',[sys,'/','Milling/Sum'])
set_param([sys,'/','Milling/Sum'],...
                'inputs','-+',...
                'position',[230,285,250,305])

add_block('built-in/Memory',[sys,'/','Milling/Memory'])
set_param([sys,'/','Milling/Memory'],...
                'position',[135,155,175,185])

add_block('built-in/Sum',[sys,'/','Milling/Sum1'])
set_param([sys,'/','Milling/Sum1'],...
```

```
                'inputs','-+',...
                'position',[245,165,265,185])

add_block('built-in/Gain',[sys,'/','Milling/Gain'])
set_param([sys,'/','Milling/Gain'],...
                'Gain','1.3',...
                'position',[340,162,365,188])

add_block('built-in/Gain',[sys,'/','Milling/Gain1'])
set_param([sys,'/','Milling/Gain1'],...
                'Gain','1.5',...
                'position',[350,237,375,263])

add_block('built-in/Outport',[sys,'/','Milling/Outport2'])
set_param([sys,'/','Milling/Outport2'],...
                'Port','2',...
                'position',[440,235,460,255])

add_block('built-in/Outport',[sys,'/','Milling/Outport1'])
set_param([sys,'/','Milling/Outport1'],...
                'position',[435,165,455,185])
add_line([sys,'/','Milling'],[65,270;140,270;140,290;225,290])
add_line([sys,'/','Milling'],[70,345;155,345;155,300;225,300])
add_line([sys,'/','Milling'],[90,270;90,170;130,170])
add_line([sys,'/','Milling'],[180,170;240,170])
add_line([sys,'/','Milling'],[205,290;205,180;240,180])
add_line([sys,'/','Milling'],[270,175;335,175])
add_line([sys,'/','Milling'],[255,295;290,295;290,250;345,250])
add_line([sys,'/','Milling'],[370,175;430,175])
add_line([sys,'/','Milling'],[380,250;400,250;400,245;435,245])
set_param([sys,'/','Milling'],...
                'Mask Display','plot(-4,-1,4,5,[0 -1 1 0 3 2.9 -3 -2.9 0],[2 0 0 2 3 3.1 1.1
1.0 2],0.3*cos(linspace(0,2*pi,9))+2,0.3*sin(linspace(0,2*pi,9))+3.1);',...
                'Mask Type','Milling')
set_param([sys,'/','Milling'],...
                'Mask Dialogue','Milling|Initial Conditions:',...
                'Mask Translate','init_cond = @1;',...
                'Mask Help','Dynamics of the Milling system',...
                'Mask Entries','[0 0 0 0]\/')


%    Finished composite block 'Milling'.

set_param([sys,'/','Milling'],...
                'position',[255,141,310,199])
```

```
add_block('built-in/Mux',[sys,'/','Mux'])
set_param([sys,'/','Mux'],...
                'orientation',2,...
                'hide name',0,...
                'inputs','2',...
                'position',[330,269,375,316])

add_block('built-in/Mux',[sys,'/','Mux2'])
set_param([sys,'/','Mux2'],...
                'hide name',0,...
                'inputs','2',...
                'position',[425,83,455,112])

add_block('built-in/S-Function',[sys,'/',['Fuzzy Inference System',13,'(Fuzzy Logic
Controller)']])
set_param([sys,'/',['Fuzzy Inference System',13,'(Fuzzy Logic Controller)']],...
                'orientation',2,...
                'function name','sffis',...
                'parameters','fismatrix')
set_param([sys,'/',['Fuzzy Inference System',13,'(Fuzzy Logic Controller)']],...
                'Mask Display','plot(0, -0.3, 6, -0.3, 0, 1.2, 6, 1.2, [0 6], [0 0], [1 2 3],[0 1
0],[2 3 4],[0 1 0],[3 4 5],[0 1 0])',...
                'Mask Type','FIS')
set_param([sys,'/',['Fuzzy Inference System',13,'(Fuzzy Logic Controller)']],...
                'Mask Dialogue','FIS|FIS Matrix:',...
                'Mask Translate','fismatrix = @1;',...
                'Mask Help','Fuzzy Inference System',...
                'Mask Entries','MIL275V')
set_param([sys,'/',['Fuzzy Inference System',13,'(Fuzzy Logic Controller)']],...
                'position',[240,269,295,321])

add_block('built-in/Saturation',[sys,'/','Saturation'])
set_param([sys,'/','Saturation'],...
                'Lower Limit','0.4',...
                'Upper Limit','1.2',...
                'position',[105,283,135,307])

add_block('built-in/Demux',[sys,'/','Demux'])
set_param([sys,'/','Demux'],...
                'orientation',3,...
                'outputs','2',...
                'position',[195,210,235,240])


%    Subsystem 'Graph'.
```

```
new_system([sys,'/','Graph'])
set_param([sys,'/','Graph'],'Location',[0,59,274,252])

add_block('built-in/Inport',[sys,'/','Graph/x'])
set_param([sys,'/','Graph/x'],...
                'position',[65,55,85,75])

add_block('built-in/S-Function',[sys,'/',['Graph/S-function',13,'M-file which
plots',13,'lines',13,'']])
set_param([sys,'/',['Graph/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfuny',...
                'parameters','ax, color,dt',...
                'position',[130,55,180,75])
add_line([sys,'/','Graph'],[90,65;125,65])
set_param([sys,'/','Graph'],...
                'Mask
Display','plot(0,0,100,100,[90,10,10,10,90,90,10],[65,65,90,40,40,90,90],[90,78,69,54,40
,31,25,10],[77,60,48,46,56,75,81,84])',...
                'Mask Type','Graph scope.')
set_param([sys,'/','Graph'],...
                'Mask Dialogue','Graph scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Time range:|y-min:|y-max:|Line type (rgbw-:*). Seperate
each plot by "/":')
set_param([sys,'/','Graph'],...
                'Mask Translate','color = @4; ax = [0, @1, @2, @3]; dt = -1;')
set_param([sys,'/','Graph'],...
                'Mask Help','This block plots to the MATLAB graph window and can be
used as an improved version of the Scope block. Look at the m-file sfuny.m to see how it
works. This block can take scalar or vector input signal.')
set_param([sys,'/','Graph'],...
                'Mask Entries','40\/-0.3\/1\/"y-/g--/c-./w:/m*/ro/b+"\/')


%    Finished composite block 'Graph'.

set_param([sys,'/','Graph'],...
                'position',[500,21,530,59])

add_block('built-in/From File',[sys,'/','From File'])
set_param([sys,'/','From File'],...
                'File name','sim.mat',...
                'position',[20,281,90,309])

add_block('built-in/Mux',[sys,'/','Mux3'])
set_param([sys,'/','Mux3'],...
                'hide name',0,...
```

```
                'inputs','2',...
                'position',[420,31,450,64])


%    Subsystem 'Graph1'.

new_system([sys,'/','Graph1'])
set_param([sys,'/','Graph1'],'Location',[0,59,274,252])

add_block('built-in/Inport',[sys,'/','Graph1/x'])
set_param([sys,'/','Graph1/x'],...
                'position',[65,55,85,75])

add_block('built-in/S-Function',[sys,'/',['Graph1/S-function',13,'M-file which
plots',13,'lines',13,"]])
set_param([sys,'/',['Graph1/S-function',13,'M-file which plots',13,'lines',13,"]],...
                'function name','sfuny',...
                'parameters','ax, color,dt',...
                'position',[130,55,180,75])
add_line([sys,'/','Graph1'],[90,65;125,65])
set_param([sys,'/','Graph1'],...
                'Mask
Display','plot(0,0,100,100,[90,10,10,10,90,90,10],[65,65,90,40,40,90,90],[90,78,69,54,40
,31,25,10],[77,60,48,46,56,75,81,84])',...
                'Mask Type','Graph scope.')
set_param([sys,'/','Graph1'],...
                'Mask Dialogue','Graph scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Time range:|y-min:|y-max:|Line type (rgbw-:*). Seperate
each plot by "/":')
set_param([sys,'/','Graph1'],...
                'Mask Translate','color = @4; ax = [0, @1, @2, @3]; dt = -1;')
set_param([sys,'/','Graph1'],...
                'Mask Help','This block plots to the MATLAB graph window and can be
used as an improved version of the Scope block. Look at the m-file sfuny.m to see how it
works. This block can take scalar or vector input signal.')
set_param([sys,'/','Graph1'],...
                'Mask Entries','40\/-0.3\/1\/"y-/g--/c-./w:/m*/ro/b+"\/')


%    Finished composite block 'Graph1'.

set_param([sys,'/','Graph1'],...
                'position',[500,76,530,114])
add_line(sys,[460,100;495,95])
add_line(sys,[325,295;300,295])
add_line(sys,[140,295;160,295;160,235;40,235;40,55;130,55;130,170;250,170])
```

```
add_line(sys,[130,140;190,140;190,90;420,90])
add_line(sys,[315,155;455,155;455,305;380,305])
add_line(sys,[95,295;100,295])
add_line(sys,[330,90;330,40;415,40])
add_line(sys,[235,295;215,295;215,245])
add_line(sys,[205,205;205,55;415,55])
add_line(sys,[455,50;495,40])
add_line(sys,[315,185;420,185;420,280;380,280])
add_line(sys,[225,205;225,105;420,105])

drawnow

% Return any arguments.
if (nargin | nargout)
        % Must use feval here to access system in memory
        if (nargin > 3)
                if (flag == 0)
                        eval(['[ret,x0,str,ts,xts]=',sys,'(t,x,u,flag);'])
                else
                        eval(['ret =', sys,'(t,x,u,flag);'])
                end
        else
                [ret,x0,str,ts,xts] = feval(sys);
        end
else
        drawnow % Flash up the model and execute load callback
end
```

Appendix C

Listing of Fuzzy Logic Control Programs

C.1 Corresponding Values of Membership Functions

```
[System]
Name='MILL'
NumInputs=2
NumOutputs=2
NumRules=42
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='Power'
Range=[-1 1]
NumMFs=7
MF1='NB':'trimf',[-1.333 -1 -0.74]
MF2='NM':'trimf',[-1 -0.6666 -0.29]
MF3='NS':'trimf',[-0.74 -0.3334 -0.07]
MF4='ZE':'trimf',[-0.29 0 0.25]
MF5='PS':'trimf',[0.07 0.3334 0.74]
MF6='PM':'trimf',[0.25 0.6666 1]
MF7='PB':'trimf',[0.74 1 1.334]

[Input2]
Name='Error'
Range=[-1 1]
NumMFs=7
```

MF1='NB':'trimf',[-1.333 -1 -0.74]
MF2='NM':'trimf',[-1 -0.6666 -0.29]
MF3='NS':'trimf',[-0.74 -0.3334 -0.08]
MF4='ZE':'trimf',[-0.29 0 0.27]
MF5='PS':'trimf',[0.08 0.3334 0.74]
MF6='PM':'trimf',[0.27 0.6666 1]
MF7='PB':'trimf',[0.74 1 1.334]

[Output1]
Name='Feed'
Range=[-1 1]
NumMFs=7
MF1='NB':'trimf',[-1.333 -1 -0.6666]
MF2='NM':'trimf',[-1 -0.6666 -0.3334]
MF3='NS':'trimf',[-0.6666 -0.3334 0]
MF4='ZE':'trimf',[-0.3334 0 0.3334]
MF5='PS':'trimf',[0 0.3334 0.6666]
MF6='PM':'trimf',[0.3334 0.6666 1]
MF7='PB':'trimf',[0.6666 1 1.334]

[Output2]
Name='Speed'
Range=[-1 1]
NumMFs=7
MF1='NB':'trimf',[-1.333 -1 -0.6666]
MF2='NM':'trimf',[-1 -0.6666 -0.3334]
MF3='NS':'trimf',[-0.6666 -0.3334 0]
MF4='ZE':'trimf',[-0.3334 0 0.3334]
MF5='PS':'trimf',[0 0.3334 0.6666]
MF6='PM':'trimf',[0.3334 0.6666 1]
MF7='PB':'trimf',[0.6666 1 1.334]

[Rules]
1 3, 1 7 (1) : 1
1 4, 1 7 (1) : 1
1 5, 1 7 (1) : 1
1 6, 1 7 (1) : 1
1 7, 1 6 (1) : 1
2 2, 3 5 (1) : 1
2 3, 2 6 (1) : 1
2 4, 1 7 (1) : 1
2 5, 1 7 (1) : 1
2 6, 1 7 (1) : 1
2 7, 1 7 (1) : 1
3 1, 5 3 (1) : 1
3 2, 4 4 (1) : 1

3 3, 4 4 (1) : 1
3 4, 3 5 (1) : 1
3 5, 3 5 (1) : 1
3 6, 1 6 (1) : 1
3 7, 1 6 (1) : 1
4 1, 5 3 (1) : 1
4 2, 5 3 (1) : 1
4 3, 4 4 (1) : 1
4 4, 4 4 (1) : 1
4 5, 4 4 (1) : 1
4 6, 3 5 (1) : 1
4 7, 2 6 (1) : 1
5 1, 7 1 (1) : 1
5 2, 6 2 (1) : 1
5 3, 5 3 (1) : 1
5 4, 4 4 (1) : 1
5 5, 4 4 (1) : 1
5 6, 3 5 (1) : 1
5 7, 3 5 (1) : 1
6 1, 7 2 (1) : 1
6 2, 7 2 (1) : 1
6 3, 6 2 (1) : 1
6 4, 6 3 (1) : 1
6 5, 5 3 (1) : 1
6 6, 4 4 (1) : 1
7 1, 7 1 (1) : 1
7 2, 7 2 (1) : 1
7 3, 7 2 (1) : 1
7 4, 7 2 (1) : 1

C.2 Listing of Fuzzy Logic Control Program

```
#include <graphics.h>
#include <float.h>
#include <bios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
#include <io.h>
#include <stdarg.h>
#include "pclab.drv"
```

```
/*************************************************************************
 Macros and definitions
 *************************************************************************/
/* key definition */
#define UPARR        0x4800
#define DNARR        0x5000
#define ESC          0x11b
#define RTARR        0x4d00
#define LTARR        0x4b00
#define PGUP         0x4900
#define PGDN         0x5100
#define HOME         0x4700
#define ENTER        0x1c0d
#define BACKSP       0x0e08
#define F10          0x4400
/* working parameter */
#define  ADSET       10
#define OK           55
#define FAIL         0xAA
#define ON           '1'
#define OFF          '0'
#define KNULL        -1
#define TRUE         0x50
#define FALSE        0x05

/* operation parameter */

#define SETUP        1          /* setup window command */
#define RECOVER      2          /* erase window command */
#define TOTALSENSOR  3          /* available SENSORS number */
#define Pow_Sen      0          /* SENSOR codes */
#define For_Sen      1
#define Bot_Sen      2

#define TOTALMATERIAL  3        /* available MATERIALs number */
#define H_HSS        0          /* MATERIAL codes */
#define C_HSS        1
#define ALUMIN       2

#define TOTALPARAMETER  2       /* available Parameters number */
#define FEED         0          /* Parameter codes */
#define FEED_SPEED   1

#define TOTALMENU    4
#define SENSORTYPE   0          /* selectable menu no. */
#define MATERIALTYPE  1         /* menu code */
```

```
#define Tool_RANGE     2
#define PARAMETERTYPE  3
#define MAXFEED 100  //maximum feedrate(mmpm)
#define Dim 850      //Dimension of the arrays for storing the control data
#define ABS(x)   ( (x) > (0) ? (x): (-(x)) )
#define MAX(x,y) ( (x) > (y) ? (x) : (y) )
#define MIN(x,y) ( (x) < (y) ? (x) : (y) )
#define MF_PARA_N 4
#define GET_FIS 500
#define MF_POINT_N 101
#define FREEARRAY(array) free(array)
#define FREEMAT(mat,m) fisFreeMatrix(mat,m)
/*************************************************************************
 Data types
 *************************************************************************/
        int  iv;
        int Speed, Feed, Select,InPut_N;
        float one_hundred, zero, desired_percentage,upper_speed, lower_speed;
        float Feed_LowerLimit = 0.05, Feed_UpLimit = 1.2;
        float xorig=100., yorig=410.;
        float xend=600., yend=120.;
        float P_ref = 0.85, F_ref=0.5, Pmin=0.5;
        int ymaxscale=1;
        float NoPixelsX, NoPixelsY;
  int rdkey;
        int Tool_item_no_no=4;              /* available Tool range no. */
        int Tool_item_no=1;                 /* current Tool range */
        int piece_partI=C_HSS;                                      /* current
MATERIAL type */
        int Sens_Type=Pow_Sen;               /* current sensor  */
        int Para_Type=FEED_SPEED;            /* current parameter */
        int menuI=SENSORTYPE;                /* current menu */
        char vbuf[30][20][2];               /* video buffer for window process */
        int monochrome=ON;                   /* current monitor type */
  typedef struct mf_node {
        char label[GET_FIS];
        char type[GET_FIS];
        double (*mfFcn)(); /* pointer to a mem. fcn */
        double para[MF_PARA_N];
        double value;
        double *value_array;
        int userDefined;
        } MF;
typedef struct fis_node {
        int handle;
        int load_param;
```

```
        char name[GET_FIS];
        char type[GET_FIS];
        char andMethod[GET_FIS];
        char orMethod[GET_FIS];
        char impMethod[GET_FIS];
        char aggMethod[GET_FIS];
        char defuzzMethod[GET_FIS];
        int userDefinedAnd;
        int userDefinedOr;
        int userDefinedImp;
        int userDefinedAgg;
        int userDefinedDefuzz;
        int in_n;
        int out_n;
        int rule_n;
        int **rule_list;
        double *rule_weight;
        int *and_or;    /* AND-OR indicator */
        double *firing_strength;
        double *rule_output;
        struct io_node **input;
        struct io_node **output;

        double (*andFcn)();
        double (*orFcn)();
        double (*impFcn)();
        double (*aggFcn)();

        double (*defuzzFcn)();
        double *BigOutMfMatrix;
        double *BigWeightMatrix;
        double *mfs_of_rule; /* MF values in a rule */
        struct fis_node *next;
} FIS;

typedef struct io_node {
        char name[GET_FIS];
        int mf_n;
        double bound[2];
        double value;
        struct mf_node **mf;
} IO;

 /*********************************************************************/
/* Initialization of the feed rate just to start the control process*/
 /*********************************************************************/
```

```
 /* Default parameter setting */

piece_part_default()
{

 switch (piece_partI)
 {
        case H_HSS:
                Tool_item_no_no=4;

                break;

        case C_HSS:
                        Tool_item_no_no=4;

                        break;

        case ALUMIN:

                        Tool_item_no_no=4;

                        break;

 }

 return(0);
}


/* draw test picture */
drawscreen()
{
        char gtable[20][25];
        int x,y;
        char ch;
        char msg[60];
        int gdriver = DETECT, gmode, errorcode;
        clrscr();
        initgraph(&gdriver, &gmode, "d:\\bc_45\\bgi");
        /* read result of initialization */
        errorcode = graphresult();
        /* an error occurred */
        if (errorcode != grOk)
        {
                printf("Graphics error: %s\n", grapherrormsg(errorcode));
```

```
                printf("Press any key to halt:");
                getch();
                exit(1);
        }
        setbkcolor(CYAN);
                x = getmaxx() / 2;
                y = getmaxy() / 3;
        settextjustify(CENTER_TEXT, CENTER_TEXT);
                settextstyle(BOLD_FONT, HORIZ_DIR, 1);
                setcolor(RED);
                sprintf(msg, " Intelligent Adaptive Control System for");
                outtextxy(x, y, msg);
                settextstyle(BOLD_FONT, HORIZ_DIR, 1);
                setcolor(RED);
                sprintf(msg, "  END MILLING ");
                outtextxy(x, y+50, msg);
                settextstyle(COMPLEX_FONT, HORIZ_DIR, 1);
                setcolor(WHITE);
                sprintf(msg, "Please wait while program is starting ...");
                outtextxy(x-5, y+100, msg);
                settextstyle(SMALL_FONT, HORIZ_DIR, 4);
                        setcolor(WHITE);
                outtextxy(x-180, y+250, msg);
                delay(2000);
                cleardevice();
                setbkcolor(5);
        closegraph();
 setback(CYAN);
 clrscr();
 settcolor(YELLOW);
 gotoxy(1,1);
 cputs("IAFC Ver 1.1          INTELLIGENT ADAPTIVE FUZZY CONTROLLER
");
 settcolor(BROWN);
 gotoxy(1,2);

cputs("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ\n");
 settcolor(WHITE);
 gotoxy(1,3);
 cputs(" SENSOR     PART MATERIAL      TOOL SIZE      MACHINE
PARAMETER ");
 settcolor(BROWN);
 gotoxy(1,4);
```

```
cputs("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ\n");

 }
/* display help ,hints messages */
showhelp(no)
int no;
{
 setback(BLUE);
 settcolor(WHITE);
 switch(no) {
        case KNULL:
          gotoxy(1,24);
          delline();
          gotoxy(1,24);
          cputs("LEFT/RIGHT ARROW-menu select        ENTER-confirm          F10-
Exit the menu ");
   gotoxy(1,25);
          delline();

          break;
        case MATERIALTYPE:
        case Tool_RANGE:
  case PARAMETERTYPE:

          gotoxy(1,24);
          delline();
          gotoxy(1,24);
          cputs("LEFT/RIGHT ARROW-menu select     UP/DOWN ARROW-item select
ENTER-confirm ");
          gotoxy(1,25);
          delline();
          gotoxy(1,25);
          cputs("ESC-exit without change                                      ");
          break;

               default:
          break;
 }
}


/* display menu */
showmenu(no)
int no;
```

```
{
 int indx;
 struct menutable{          /* menu function screen table */
                    char name[18];      /* menu function name */
                    int xpos;          /* menu function location */
                    int ypos;
                 }menufun[TOTALMENU]={
 {"SENSOR\0",3,3},{"PART MATERIAL\0",15,3},{"TOOL
SIZE\0",35,3},{"MACHINE PARAMETER\0",52,3}};

 setnormal(CYAN);
 for (indx=0;indx<TOTALMENU;indx++) {
        if (indx==no) setinverse(MAGENTA);
        gotoxy(menufun[indx].xpos,menufun[indx].ypos);
        cputs(menufun[indx].name);
        setnormal(CYAN);
 }
}

/* display items */
showitem(menutype,itemno)
int menutype;
int itemno;
{
 int indx;

 char Sensor_type[TOTALSENSOR][20]={
 {" POWER  \0"},
 {" FORCE  \0"},
 {" BOTH  \0"}

 };

 char Parameter_type[TOTALPARAMETER][20]={
 {"  FEED  \0"},
 {" FEED& FORCE  \0"}

 };

 char MATERIAL_str[TOTALMATERIAL][20]={
 {" HOT-HSS  \0"},
 {" COLD-HSS  \0"},
 {" ALUMINIUM  \0"}

 };
```

```
char Tool_Size[TOTALMATERIAL][4][10]={
{{"5/8\0"},{"9/16\0"},{"7/16\0"},{"5/16\0"}},
{{"5/8\0"},{"9/16\0"},{"7/16\0"},{"5/16\0"}},
{{"5/8\0"},{"9/16\0"},{"7/16\0"},{"5/16\0"}}
};

int gain_code;

settcolor(WHITE);
switch (menutype) {
        case SENSORTYPE:
         for (indx=0;indx<TOTALSENSOR;indx++) {
                gotoxy(2,2+indx);
                if (indx==itemno) setinverse(MAGENTA);
                cputs(Sensor_type[indx]);
                setnormal(BLUE);
         }
         break;

         case MATERIALTYPE:
         for (indx=0;indx<TOTALMATERIAL;indx++) {
                gotoxy(2,2+indx);
                if (indx==itemno) setinverse(MAGENTA);
                cputs(MATERIAL_str[indx]);
                setnormal(BLUE);
         }
         break;

        case Tool_RANGE:
         if (Tool_item_no_no>0) {
                for (indx=0;indx<Tool_item_no_no;indx++) {
                        gotoxy(2,2+indx);
                        if (indx==itemno) setinverse(MAGENTA);
                        cputs(Tool_Size[piece_partI][indx]);
                        setnormal(BLUE);
                }
         }
         break;

         case PARAMETERTYPE:
         for (indx=0;indx<TOTALPARAMETER;indx++) {
                gotoxy(2,2+indx);
                if (indx==itemno) setinverse(MAGENTA);
                cputs(Parameter_type[indx]);
                setnormal(BLUE);
         }
```

```
            break;
            default:
            break;
 }
 return(0);
}

/* menu window display process */
winproc(menutype,opcode)
int menutype;
int opcode;
{
 int cnt;

 switch (menutype) {
        case SENSORTYPE:
                        frame(2,4,12,TOTALSENSOR+5,opcode);
                        if (opcode==SETUP) showitem(SENSORTYPE,Sens_Type);
                        break;
        case MATERIALTYPE:
                        frame(14,4,28,TOTALMATERIAL+5,opcode);
                        if (opcode==SETUP) showitem(MATERIALTYPE,piece_partI);
                        break;

        case Tool_RANGE:
                        if (Tool_item_no_no>0) {
                                frame(35,4,42,Tool_item_no_no+5,opcode);
                                if (opcode==SETUP)
showitem(Tool_RANGE,Tool_item_no);
                        }
                        break;

        case PARAMETERTYPE:
                        frame(52,4,68,TOTALPARAMETER+5,opcode);
                        if (opcode==SETUP)
showitem(PARAMETERTYPE,Para_Type);
                        break;

        default:
                        break;
 }
}

/* window process */
frame(left,top,right,bottom,fun)
int left;
```

```
int top;
int right;
int bottom;
int fun;
{
 int xx,yy;
 int length,width;
 int line;
 switch (fun) {
        case SETUP:
                                gettext(left,top,right,bottom,vbuf);
                                window(left,top,right,bottom);
                                setback(BLUE);
                                clrscr();
                                length=bottom-top+1;
                                width=right-left+1;
                                for (xx=1;xx<=width;xx++)
                                        for (yy=1;yy<=length;yy++) {
                                                gotoxy(xx,yy);
                                                putch(' ');
                                        }
                                settcolor(YELLOW);
                                gotoxy(1,1);putch('É');
                                gotoxy(width,1);putch('»');
                                gotoxy(1,length-1);putch('È');
                                gotoxy(width,length-1);putch('¼');
                                gotoxy(1,length-1);insline();
                                for (line=2;line<width;line++) {
                                        gotoxy(line,1);putch('Í');
                                        gotoxy(line,length);putch('Í');
                                }
                                for (line=2;line<length;line++) {
                                        gotoxy(1,line);putch('º');
                                        gotoxy(width,line);putch('º');
                                }
                                settcolor(WHITE);
                                break;
        case RECOVER:
                                puttext(left,top,right,bottom,vbuf);
                                window(1,1,80,25);
                                break;
 }
}

/* set background color */
setback(color)
```

```
int color;
{
 if (monochrome==OFF) textbackground(color);
 else textbackground(BLACK);
}

/* set text color */
settcolor(color)
int color;
{
 if (monochrome==OFF) textcolor(color);
 else textcolor(WHITE);
}

/* set item background color to inverse */
setinverse(color)
int color;
{
 if (monochrome==ON) {textbackground(WHITE);textcolor(BLACK);}
 else textbackground(color);
}

/* set item background color to normal */
setnormal(color)
int color;
{
 if (monochrome==ON) {textbackground(BLACK);textcolor(WHITE);}
 else textbackground(color);
}
/* read one key from keyboard */
int readkey()
{
 while (bioskey(1)==0);
 return(bioskey(0));
}

/* menu selection */
menuselect()
{
// int rdkey;
 int complete;
 int edtype;

 if (bioskey(1)==0) return;
 rdkey=bioskey(0);
 switch (rdkey) {
```

```
case LTARR:
                    if (--menuI<0) menuI=TOTALMENU-1;
                    showmenu(menuI);
                    break;
case RTARR:
                    if (++menuI>=TOTALMENU) menuI=0;
                    showmenu(menuI);
                    break;
case ENTER:
                    do
                    {
                        switch (menuI)
                        {
                            case SENSORTYPE:
                                complete=selectSENSOR();
                                break;

                            case MATERIALTYPE:
                                complete=selectMATERIAL();
                                break;

                            case Tool_RANGE:
                                switch (piece_partI)
                                {
                                    case H_HSS:
                                    case C_HSS:
                                    case ALUMIN:

complete=selectad();
                                                break;

                                    default:

complete=selectad();
                                                        break;
                                }
                                break;

                            case PARAMETERTYPE:
                                complete=selectPARAMETER();
                                break;
                        }
                    }
                    while (complete==FAIL);
                    break;
```

```
        case F10:

                default   :
                                break;
}
}

/*  SENSOR selection */
int selectSENSOR()
{
 //int rdkey;
 int defSENSOR;
 int defmenu;
 showhelp(SENSORTYPE);
 winproc(menuI,SETUP);
 defmenu=menuI;
 defSENSOR=Sens_Type;
 do {
        showitem(menuI,defSENSOR);
        switch (rdkey=readkey()) {
         case DNARR:
                                if (++defSENSOR>=TOTALSENSOR)
defSENSOR=0;
                                break;
         case UPARR:
                                if (--defSENSOR<0)
defSENSOR=TOTALSENSOR-1;
                                break;
         case ENTER:
                                Sens_Type=defSENSOR;
                                break;
         case LTARR:
                                if (--defmenu<0) defmenu=TOTALMENU-1;
                                break;
         case RTARR:
                                if (++defmenu>=TOTALMENU) defmenu=0;
                                break;
         default   :
                                break;
        }
 }
 while ((rdkey!=ENTER)&&(rdkey!=ESC)&&(rdkey!=LTARR)&&(rdkey!=RTARR));
 winproc(menuI,RECOVER);
 if (rdkey==ENTER) {

 //      piece_part_default();
```

```
        defmenu=MATERIALTYPE;
 }
 menuI=defmenu;
 showmenu(menuI);
 showhelp(KNULL);
 if ((rdkey==ENTER)||(rdkey==ESC)) {return(OK);}
 else return(FAIL);
}



/*  MATERIAL selection */
int selectMATERIAL()
{
// int rdkey;
 int defMATERIAL;
 int defmenu;
 showhelp(MATERIALTYPE);
 winproc(menuI,SETUP);
 defmenu=menuI;
 defMATERIAL=piece_partI;
 do {
        showitem(menuI,defMATERIAL);
        switch (rdkey=readkey()) {
         case DNARR:
                                        if (++defMATERIAL>=TOTALMATERIAL)
defMATERIAL=0;
                                        break;
         case UPARR:
                                        if (--defMATERIAL<0)
defMATERIAL=TOTALMATERIAL-1;
                                        break;
         case ENTER:
                                        piece_partI=defMATERIAL;
                                        break;
         case LTARR:
                                        if (--defmenu<0) defmenu=TOTALMENU-1;
                                        break;
         case RTARR:
                                        if (++defmenu>=TOTALMENU) defmenu=0;
                                        break;
         default   :
                                        break;
        }
```

```
}
while ((rdkey!=ENTER)&&(rdkey!=ESC)&&(rdkey!=LTARR)&&(rdkey!=RTARR));
winproc(menuI,RECOVER);
if (rdkey==ENTER) {

        piece_part_default();

         defmenu=Tool_RANGE;
}
menuI=defmenu;
showmenu(menuI);
showhelp(KNULL);
if ((rdkey==ENTER)||(rdkey==ESC)) {return(OK);}
else return(FAIL);
}



/* Tool range selection */
int selectad()
{
 //int rdkey;
 int defad;
 int defmenu;
 int indx;

 if (Tool_item_no_no>0) showhelp(Tool_RANGE);
 winproc(menuI,SETUP);
 defmenu=menuI;
 defad=Tool_item_no;
 do {
        showitem(menuI,defad);
        switch (rdkey=readkey()) {
          case DNARR:
                                  if (++defad>=Tool_item_no_no) defad=0;
                                  break;
          case UPARR:
                                  if (--defad<0) defad=Tool_item_no_no-1;
                                  break;
          case ENTER:
                                  Tool_item_no=defad;

                                  break;
          case LTARR:
                                  if (--defmenu<0) defmenu=TOTALMENU-1;
                                  break;
```

```
            case RTARR:
                                        if (++defmenu>=TOTALMENU) defmenu=0;
                                        break;
            default   :
                                        break;
           }
 }
 while ((rdkey!=ENTER)&&(rdkey!=ESC)&&(rdkey!=LTARR)&&(rdkey!=RTARR));
 winproc(menuI,RECOVER);
 if (rdkey==ENTER)
 {
/*        if (Tool_item_no_no>0) {

          } */
   defmenu=PARAMETERTYPE;
 }
 menuI=defmenu;
 showmenu(menuI);
 showhelp(KNULL);
 if ((rdkey==ENTER)||(rdkey==ESC)) {return(OK);}
 else return(FAIL);
}
/* Parameter selection */
int selectPARAMETER()
{
 //int rdkey;
 int defPARAMETER;
 int defmenu;
 showhelp(PARAMETERTYPE);
 winproc(menuI,SETUP);
 defmenu=menuI;
 defPARAMETER=Para_Type;
 do {
        showitem(menuI,defPARAMETER);
        switch (rdkey=readkey()) {
          case DNARR:
                                        if (++defPARAMETER>=TOTALPARAMETER)
defPARAMETER=0;
                                        break;
          case UPARR:
                                        if (--defPARAMETER<0)
defPARAMETER=TOTALPARAMETER-1;
                                        break;
          case ENTER:
                                        Para_Type=defPARAMETER;
                                        break;
```

```
            case LTARR:
                                        if (--defmenu<0) defmenu=TOTALMENU-1;
                                        break;
            case RTARR:
                                        if (++defmenu>=TOTALMENU) defmenu=0;
                                        break;
            default   :
                                        break;
          }
 }
 while ((rdkey!=ENTER)&&(rdkey!=ESC)&&(rdkey!=LTARR)&&(rdkey!=RTARR));
 winproc(menuI,RECOVER);
 if (rdkey==ENTER) {

  //      piece_part_default();

          defmenu=SENSORTYPE;
 }
 menuI=defmenu;
 showmenu(menuI);
 showhelp(KNULL);
 if ((rdkey==ENTER)||(rdkey==ESC)) {return(OK);}
 else return(FAIL);
}


/* Tool setting type selection */
int selecttype(deftype)
int *deftype;
{
// int rdkey;

 showhelp(ADSET);
 winproc(ADSET,SETUP);
 *deftype=1;
 do {
        showitem(ADSET,*deftype);
        switch (rdkey=readkey()) {
          case DNARR:
                                        if (++*deftype>=2) *deftype=0;
                                        break;
          case UPARR:
                                        if (--*deftype<0) *deftype=1;
                                        break;
          case ENTER:
                                        break;
```

```
        case LTARR:
                                if (--menuI<0) menuI=TOTALMENU-1;
                                *deftype=KNULL;
                                break;
        case RTARR:
                                if (++menuI>=TOTALMENU) menuI=0;
                                *deftype=KNULL;
                                break;
        case ESC:
                                *deftype=KNULL;
                                break;
        default   :
                                break;
       }
}
while ((rdkey!=ENTER)&&(rdkey!=ESC)&&(rdkey!=LTARR)&&(rdkey!=RTARR));
winproc(ADSET,RECOVER);
showmenu(menuI);
if ((rdkey==ENTER)||(rdkey==ESC)) return(OK);
else return(FAIL);
}


/*************************************************************/
        float init()
{

        float dp;
        clrscr();
        reset_ad();       /* reset A/D */
        reset_da();       /* reset D/A */


        /* Feedrate Initialization */
        /* D/A output pin: B15 */
        one_hundred = 3050;                 /* 100 % */
        zero = 1200;                    /* 0 % */
        daout(1, zero);
         Select=Sens_Type+1;
                InPut_N=2;
                if(Select==3) InPut_N=4;
                        daout(0, 0);
                if (piece_partI==2 && Para_Type==1 && Tool_item_no==1)
                {
                 lower_speed=200;
                 upper_speed=350;
```

```
                desired_percentage=20;
                }
               if(piece_partI==2 && Para_Type==0 && Tool_item_no==1)
               {
                lower_speed=300;
                upper_speed=300;
                desired_percentage=20;
                }
                if (piece_partI!=2 && Para_Type==1 && Tool_item_no!=2)
               {
                       lower_speed=210;
                        upper_speed=350;
                desired_percentage=15;
                       }
                        if (piece_partI!=2 && Para_Type==0 && Tool_item_no!=2)
               {
                       lower_speed=300;
                        upper_speed=300;
                desired_percentage=15;
                       }
        lower_speed=lower_speed*10 ;
        Speed=(int)lower_speed+900;
        daout(0, Speed);
         upper_speed = upper_speed *10;
        Feed = (int)((one_hundred-zero)*(desired_percentage/105.0)
                       + zero);
//      daout(1, Feed);     /* Write to port 1 of D/A */
         dp = desired_percentage/100.0;
        return dp;
}


void calncon(float u, int *pcoory_0, int *pcoory_1,  int *pcoory_2, int *pcoory_3,  int
*pcoorx_1, float *pPo_1, float *pFo_1)
{
        int  j;
        float vout1_sgl[5], vout1_max, vout1_min, vout1;
        float vout3_sgl[5], vout3_max, vout3_min, vout3;
        float vout2_sgl[5], vout2_max, vout2_min, vout2,Fo_1;
        int coory_0,coory_1,coory_2, coory_3,coorx_1;
        int gdriver = DETECT, gmode, errorcode;
        char outputtext[20];
        int y_coord = wherey();
        int dev=25;
        int coory, coorx;
        int yref_Nopix, Dim_Nopix;
```

```
u=desired_percentage/100.0;
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "d:\\bc_45\\bgi");

/* read result of initialization */
errorcode = graphresult();
/* an error occurred */
if (errorcode != grOk)
{
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
}

        /*********************************************************
                                POWER  and FORCE Measurments
         *********************************************************/

do
{
        for(j=0; j<5; j++)
        {
                do{
                        iv = adin (1);
                        vout1_sgl[j] = itov (iv);
                }while(vout1_sgl[j] < -4.0 || vout1_sgl[j] >4.0);
                do{
                        iv = adin (2);
                        vout2_sgl[j] = itov (iv);
                }while(vout2_sgl[j] < -4.0 || vout2_sgl[j] >4.0);

                do{
                        iv = adin (3);
                        vout3_sgl[j] = itov (iv);
                        vout3_sgl[j] += 0.2;
                }while(vout3_sgl[j] < 0 || vout3_sgl[j] >1.5);
        }

        vout1_min = vout1_sgl[0];
        vout1_max = vout1_sgl[0];
        vout2_min = vout2_sgl[0];
        vout2_max = vout2_sgl[0];

        vout3_min = vout3_sgl[0];
        vout3_max = vout3_sgl[0];
```

```
                for(j=1; j<5; j++)
                {

                        if (vout2_sgl[j] < vout2_min) vout2_min = vout2_sgl[j];
                        if (vout2_sgl[j] > vout2_max) vout2_max = vout2_sgl[j];

                        if (vout1_sgl[j] < vout1_min) vout1_min = vout1_sgl[j];
                        if (vout1_sgl[j] > vout1_max) vout1_max = vout1_sgl[j];

                if (vout3_sgl[j] < vout3_min) vout3_min = vout3_sgl[j];
                        if (vout3_sgl[j] > vout3_max) vout3_max = vout3_sgl[j];
                        }

                        vout1 = (vout1_sgl[0] + vout1_sgl[1] + vout1_sgl[2] +
vout1_sgl[3]\
                        + vout1_sgl[4] - vout1_min - vout1_max) / 3.;

                        vout2 = (vout2_sgl[0] + vout2_sgl[1] + vout2_sgl[2] +
vout2_sgl[3]\
                        + vout2_sgl[4] - vout2_min - vout2_max) / 3.;

                        vout3 = (vout3_sgl[0] + vout3_sgl[1] + vout3_sgl[2] +
vout3_sgl[3]\
                        + vout3_sgl[4] - vout3_min - vout3_max) / 3.;

                        delay(5);
                Fo_1 = sqrt(vout1*vout1 + vout2*vout2);
        printf("\n\n");
        delay(100);

        gotoxy(1,3);
                 printf(" FEED=0     Power= %.4f     FORCE=%.4f
Speed=%d",vout3,Fo_1,Speed/10);
        gotoxy(1, 23);

         setbkcolor(BLUE);
        setcolor(getmaxcolor());
        line(xorig, yorig, xend, yorig);
        line(xorig, yorig, xorig, yend-50);
  settextstyle(SMALL_FONT, HORIZ_DIR, 4);
        outtextxy((xend-xorig)/2+xorig, yorig+26, "Iterations");
        setcolor(YELLOW);
        settextstyle(COMPLEX_FONT, HORIZ_DIR, 1);
        outtextxy(xorig+20, yorig+40, "Press ENTER to Start or Stop the Process");
         setcolor(getmaxcolor());
```

```
        settextstyle(SMALL_FONT, HORIZ_DIR, 4);
       outtextxy(xorig, yorig+dev, "0");
        NoPixelsX = (xend-xorig)/Dim;
        NoPixelsY = (yorig-yend)/ymaxscale;
        Dim_Nopix = (Dim*NoPixelsX);
        coorx = xorig + Dim_Nopix;
       line(coorx, yorig, coorx, yorig+10);
       sprintf(outputtext, "%d", Dim);
       outtextxy(coorx, yorig+dev, outputtext);
        yref_Nopix= (ymaxscale*NoPixelsY);
       coory = yorig - yref_Nopix;
       line(xorig, coory, xorig-10, coory);
       sprintf(outputtext, "%d", ymaxscale);
       outtextxy(xorig-dev, coory, outputtext);
       outtextxy(xorig-35, coory+20, "(HP)");

       setcolor(12);
       line(xorig-50, yorig, xorig-50, yend-50);
       line(xorig-50, yorig, xorig-50-10, yorig);
       line(xorig-50, yend, xorig-50-10, yend);
       sprintf(outputtext, "%d", 100);
       outtextxy(xorig-50-33, coory, outputtext);
       outtextxy(xorig-50-30, yorig, "0");
       outtextxy(xorig-100, coory+20, "(mmpm)");

       coory = yorig - (.90*NoPixelsY);
       setcolor(3);
       line(xorig, coory, xend, coory);
       coory_0=yorig-(Fo_1*NoPixelsY);
        coory_1=yorig-(vout3*NoPixelsY);
        coory_2=yorig-(u/100*NoPixelsY);
        coory_3=yorig-(Speed/10000*NoPixelsY);
        coorx_1=xorig;
        *pcoory_0=coory_0;
        *pcoory_1=coory_1;
        *pcoory_2=coory_2;
        *pcoory_3=coory_3;
        *pcoorx_1=coorx_1;
        *pPo_1 = vout3;
        *pFo_1 = sqrt(vout1*vout1 + vout2*vout2);
delay(90);

              if (kbhit()) break;
      }while(1);
 getche();
       return;
```

```
        }

/*************************************************************************
 File, arrays, matrices operations
 *************************************************************************/

/* display error message and exit */
static void

fisError(char *msg)

{
#ifndef NO_PRINTF
        printf(msg);
        printf("\n");
#endif
exit(1);
}

/* an friendly interface to fopen() */
static FILE *

fisOpenFile(char *file,char *mode)

{
        FILE *fp, *fopen();

        if ((fp = fopen(file, mode)) == NULL){
                printf("The file '%s'", file);
                fisError(" cannot be opened.\n");
        }
        return(fp);
}

static char **

fisCreateMatrix(int row_n, int col_n,int element_size)

{
        char **matrix;
        int i;

        if (row_n == 0 && col_n == 0)
                return(NULL);
```

```
        matrix = (char **)calloc(row_n, sizeof(char *));
        if (matrix == NULL)
                fisError("Error in fisCreateMatrix!");
        for (i = 0; i < row_n; i++) {
                matrix[i] = (char *)calloc(col_n, element_size);
                if (matrix[i] == NULL)
                        fisError("Error in fisCreateMatrix!");
        }
        return(matrix);
}
/* won't complain if given matrix is already freed */
static void

fisFreeMatrix(void **matrix, int row_n)

{
        int i;
        if (matrix != NULL) {
                for (i = 0; i < row_n; i++) {

                }
                free(matrix);
        }
}

/*************************************************************************
membership functions
 *************************************************************************/

/* Triangular membership function */
static double

fisTriangleMf(double x, double *para)

{
        double a = para[0], b = para[1], c = para[2];

        if (a>b)
                fisError("Illegal parameters in fisTriangleMf() --> a > b");
        if (b>c)
                fisError("Illegal parameters in fisTriangleMf() --> b > 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));
}


static double

fisGeneralizedBellMf(double x, double *para)

{
        double a = para[0], b = para[1], c = para[2];
        double tmp;
        if (a==0)
                fisError("Illegal parameters in fisGeneralizedBellMf() --> a = 0");
        tmp = pow((x-c)/a, 2.0);
        if (tmp == 0 && b == 0)
                return(0.5);
        else if (tmp == 0 && b < 0)
                return(0.0);
        else
                return(1/(1+pow(tmp, b)));
}


/* returns the number of parameters of MF */
static int

fisGetMfParaN(char *mfType)

{
        if (strcmp(mfType, "trimf") == 0)
                return(3);
        if (strcmp(mfType, "gbellmf") == 0)
                return(3);

        printf("Given MF type (%s) is unknown.\n", mfType);
        exit(1);
        return(0);      /* get rid of compiler warning */
}


static double

fisMin(double x, double y)
```

```
{return((x) < (y) ? (x) : (y));}

static double
fisMax(double x, double y)
{return((x) > (y) ? (x) : (y));}

static double
fisProduct(double x, double y)
{return(x*y);}

static double
fisProbOr(double x, double y)
{return(x + y - x*y);}

static double
fisSum(double x, double y)
{return(x + y);}

/* apply given function to an array */
static double
fisArrayOperation(double *array, int size, double (*fcn)(double, double))
{
        int i;
        double out;

        if (size == 0)
                fisError("Given size is zero!");

        out = array[0];
        for (i = 1; i < size; i++)
                out = (*fcn)(out, array[i]);
        return(out);
}

/* return the center of area of combined output MF (specified by mf) of output m */
/* MF_POINT_N is the number of partition for integration */
static double
defuzzCentroid(FIS *fis, int m, double *mf)
{
        double min = fis->output[m]->bound[0];
        double max = fis->output[m]->bound[1];
        double step = (max - min)/(MF_POINT_N - 1);
        double total_mf = 0;
        double sum = 0;
        int i;
```

```
        for (i = 0; i < MF_POINT_N; i++){
                total_mf += mf[i];
                sum += mf[i]*(min + step*i);
        }
        if (total_mf == 0) {
                printf("Total area is zero in defuzzCentroid() for output %d!\n", m+1);
                printf("Average of the range of this output variable is used as the output
value.\n\n");
                return((fis->output[m]->bound[0] + fis->output[m]->bound[1])/2);
        }
        return(sum/total_mf);
}
 /* return the bisector of area of mf */
static double
defuzzBisector(FIS *fis, int m, double *mf)
{
        double min = fis->output[m]->bound[0];
        double max = fis->output[m]->bound[1];
        double step = (max - min)/(MF_POINT_N - 1);
        double area, sub_area;
        int i;

        /* find the total area */
        area = 0;
        for (i = 0; i < MF_POINT_N; i++)
                area += mf[i];

        if (area == 0) {
                printf("Total area is zero in defuzzBisector() for output %d!\n", m+1);
printf("Average of the range of this output variable is used as the output value.\n");
                return((fis->output[m]->bound[0] + fis->output[m]->bound[1])/2);
        }

        sub_area = 0;
        for (i = 0; i < MF_POINT_N; i++) {
                sub_area += mf[i];
                if (sub_area >= area/2)
                        break;
        }
        return(min + step*i);
}

static double
defuzzMeanOfMax(FIS *fis, int m, double *mf)
{
        double min = fis->output[m]->bound[0];
```

```
        double max = fis->output[m]->bound[1];
        double step = (max - min)/(MF_POINT_N - 1);
        double mf_max;
        double sum;
        int count;
        int i;

        mf_max = fisArrayOperation(mf, MF_POINT_N, fisMax);

        sum = 0;
        count = 0;
        for (i = 0; i < MF_POINT_N; i++)
                if (mf[i] == mf_max) {
                        count++;
                        sum += i;
                }
        return(min+step*sum/count);
}

static double
defuzzSmallestOfMax(FIS *fis, int m, double *mf)
{
        double min = fis->output[m]->bound[0];
        double max = fis->output[m]->bound[1];
        double step = (max - min)/(MF_POINT_N - 1);
        double mf_max;
        int i, min_index = 0;
        double min_distance = pow(2.0, 31.0)-1;
        double distance; /* distance to the origin */

        mf_max = fisArrayOperation(mf, MF_POINT_N, fisMax);
        for (i = 0; i < MF_POINT_N; i++)
                if (mf[i] == mf_max) {
                        distance = ABS(min + step*i);
                        if (min_distance > distance) {
                                min_distance = distance;
                                min_index = i;
                        }
                }
        return(min + step*min_index);
}

/* Returns the largest (in magnitude) maximizing x of mf */
static double
defuzzLargestOfMax(FIS *fis, int m, double *mf)
{
```

```
        double min = fis->output[m]->bound[0];
        double max = fis->output[m]->bound[1];
        double step = (max - min)/(MF_POINT_N - 1);
        double mf_max;
        int i, max_index = 0;
        double max_distance = -(pow(2.0, 31.0)-1);
        double distance; /* distance to the origin */

        mf_max = fisArrayOperation(mf, MF_POINT_N, fisMax);
        for (i = 0; i < MF_POINT_N; i++)
                if (mf[i] == mf_max) {
                        distance = ABS(min + step*i);
                        if (max_distance < distance) {
                                max_distance = distance;
                                max_index = i;
                        }
                }
        return(min + step*max_index);
}

/**********************************************************************
 Data structure: construction, printing, and destruction
 **********************************************************************/

static IO *
fisBuildIoList(int node_n, int *mf_n)

{
        IO *io_list;
        int i, j;

        io_list = (IO *)calloc(node_n, sizeof(MF));
        for (i = 0; i < node_n; i++) {
                io_list[i].mf_n = mf_n[i];
                io_list[i].mf = (MF **)calloc(mf_n[i], sizeof(MF *));
                if (mf_n[i] > 0)        /* check if no MF at all */
                        io_list[i].mf[0] = (MF *)calloc(mf_n[i], sizeof(MF));
                for (j = 0; j < mf_n[i]; j++)
                        io_list[i].mf[j] = io_list[i].mf[0] + j;
        }
        return(io_list);
}

static void
fisAssignMfPointer(FIS *fis)
{
```

```
int i, j, k, mfTypeN = 4, found;
MF *mf_node;

struct command {
        char mfType[20];
        double (*mfFcn)();
} dispach[20];

strcpy(dispach[0].mfType, "trimf");
dispach[0].mfFcn = fisTriangleMf;

strcpy(dispach[1].mfType, "gbellmf");
dispach[1].mfFcn = fisGeneralizedBellMf;

strcpy(dispach[3].mfType, "linear");
dispach[3].mfFcn = NULL;
strcpy(dispach[4].mfType, "constant");
dispach[4].mfFcn = NULL;

/* input MF's */
for (i = 0; i < fis->in_n; i++)
        for (j = 0; j < fis->input[i]->mf_n; j++) {
                mf_node = fis->input[i]->mf[j];
                found = 0;
                for (k = 0; k < mfTypeN-2; k++) {
                        if (strcmp(mf_node->type, dispach[k].mfType) == 0) {
                                mf_node->mfFcn = dispach[k].mfFcn;
                                found = 1;
                                break;
                        }
                }
                if (found == 0) {

                        printf("MF type '%s' for input %d is unknown.\n",
                                mf_node->type, i+1);
                        printf("Legal input MF types: ");
                        for (i = 0; i < mfTypeN-2; i++)
                                printf("%s ", dispach[i].mfType);
                        /*
                        printf("\n\nIf '%s' is a customized MF, use M-file command
FISEVAL1 or FISEVAL2 instead.\n", mf_node->type);
                        */
                        printf("\n");
                        fisError("\n");
                }
        }
```

```
        /* output MF's */
        for (i = 0; i < fis->out_n; i++)
                for (j = 0; j < fis->output[i]->mf_n; j++) {
                        mf_node = fis->output[i]->mf[j];
                        found = 0;
                        for (k = 0; k < mfTypeN; k++) {
                                if (strcmp(mf_node->type, dispach[k].mfType) == 0) {
                                        mf_node->mfFcn = dispach[k].mfFcn;
                                        found = 1;
                                        break;
                                }
                        }
                        if (found == 0) {

                                printf("MF type '%s' for output %d is unknown.\n",
                                        mf_node->type, i+1);
                                printf("Legal output MF types: ");
                                for (i = 0; i < mfTypeN-1; i++)
                                        printf("%s ", dispach[i].mfType);
                                                        printf("\n");
                                fisError("\n");
                        }
                }
        }

static void
fisAssignFunctionPointer(FIS *fis)
{
        /* assign andMethod function pointer */
        if (strcmp(fis->andMethod, "prod") == 0)
                fis->andFcn = fisProduct;
        else if (strcmp(fis->andMethod, "min") == 0)
                fis->andFcn = fisMin;
        else {

                printf("Given andMethod %s is unknown.\n", fis->andMethod);
                fisError("Legal andMethod: min, prod");

        }

        /* assign orMethod function pointer */
        if (strcmp(fis->orMethod, "probor") == 0)
                fis->orFcn = fisProbOr;
        else if (strcmp(fis->orMethod, "max") == 0)
                fis->orFcn = fisMax;
```

```
        else {

                printf("Given orMethod %s is unknown.\n", fis->orMethod);
                fisError("Legal orMethod: max, probor");

        }

        if (strcmp(fis->impMethod, "prod") == 0)
                fis->impFcn = fisProduct;
        else if (strcmp(fis->impMethod, "min") == 0)
                fis->impFcn = fisMin;
        else {

                printf("Given impMethod %s is unknown.\n", fis->impMethod);
                fisError("Legal impMethod: min, prod");

        }

        /* assign aggMethod function pointer */
        if (strcmp(fis->aggMethod, "max") == 0)
                fis->aggFcn = fisMax;
        else if (strcmp(fis->aggMethod, "probor") == 0)
                fis->aggFcn = fisProbOr;
        else if (strcmp(fis->aggMethod, "sum") == 0)
                fis->aggFcn = fisSum;
        else {

                printf("Given aggMethod %s is unknown.\n", fis->aggMethod);
                fisError("Legal aggMethod: max, probor, sum");

        }

        /* assign defuzzification function pointer */
        if (strcmp(fis->defuzzMethod, "centroid") == 0)
                fis->defuzzFcn = defuzzCentroid;

        else {

                printf("Given defuzzification method %s is unknown.\n", fis-
>defuzzMethod);
                fisError("Legal defuzzification methods: centroid, bisector, mom, som,
lom, wtaver, wtsum");

        }
}
```

```
static void
fisFreeMfList(MF *mf_list, int n)
{
        int i;

        for (i = 0; i < n; i++) {
                free(mf_list[i].value_array);
        }
        free(mf_list);
}

static void
fisFreeIoList(IO *io_list, int n)
{
        int i;
        for (i = 0; i < n; i++) {
                if (io_list[i].mf_n > 0)/* check if no MF at all */
                        fisFreeMfList(io_list[i].mf[0], io_list[i].mf_n);
                free(io_list[i].mf);
        }
        free(io_list);
}

static void
fisFreeFisNode(FIS *fis)
{
        if (fis == NULL)
                return;
        fisFreeIoList(fis->input[0], fis->in_n);
        free(fis->input);
        fisFreeIoList(fis->output[0], fis->out_n);
        free(fis->output);
#ifdef FREEMAT
        FREEMAT((void **)fis->rule_list, fis->rule_n);
#else
        fisFreeMatrix((void **)fis->rule_list, fis->rule_n);
#endif

        free(fis->rule_weight);
        free(fis->and_or);
        free(fis->firing_strength);
        free(fis->rule_output);
        free(fis->BigOutMfMatrix);
        free(fis->BigWeightMatrix);
        free(fis->mfs_of_rule);
        free(fis);
```

```
}

/* This is done whenever new parameters are loaded */
static void
fisComputeOutputMfValueArray(FIS *fis)
{
        int i, j, k;
        double x, lx, ux, dx;
        MF *mf_node;

        for (i = 0; i < fis->out_n; i++) {
                lx = fis->output[i]->bound[0];
                ux = fis->output[i]->bound[1];
                dx = (ux - lx)/(MF_POINT_N - 1);
                for (j = 0; j < fis->output[i]->mf_n; j++) {
                        mf_node = fis->output[i]->mf[j];
                        if (!mf_node->userDefined)
                                for (k = 0; k < MF_POINT_N; k++) {
                                        x = lx + k*dx;
                                        mf_node->value_array[k] =
                                        (*mf_node->mfFcn)(x, mf_node->para);
                                }
                        else {  /* user defined MF */

                                printf("Cannot find MF type %s!\n", mf_node->type);
                                fisError("Exiting ...");

                        }
                }
        }
}
/* copy string (the first 'length' characters) from array to target string */
static void
fisGetString2(char *target, double *array, int max_leng)
{
        int i;
        int actual_leng;

        for (actual_leng = 0; actual_leng < max_leng; actual_leng++)
                if (array[actual_leng] == 0)
                        break;

        if (actual_leng + 1 > GET_FIS) {
                printf("actual_leng = %d\n", actual_leng);
                printf("GET_FIS = %d\n", GET_FIS);
                fisError("String too long!");
```

```
        }
        for (i = 0; i < actual_leng; i++)
                target[i] = (int)array[i];
        target[actual_leng] = 0;
}


/* Build FIS node and load parameter from FIS_data directly */
/* col_n is the number of columns of the FIS_data */
static void
fisBuildFisNode(FIS *fis, double **FIS_data, int col_n)
{
        int i, j, k;
        int *in_mf_n, *out_mf_n;
        IO *io_list;
        int start;

        fisGetString2(fis->name, FIS_data[0], col_n);
        fisGetString2(fis->type, FIS_data[1], col_n);
        fis->in_n = FIS_data[2][0];
        fis->out_n = FIS_data[2][1];

        /* create input node list */
        in_mf_n = (int *)calloc(fis->in_n, sizeof(int));
        for (i = 0; i < fis->in_n; i++)
                in_mf_n[i] = FIS_data[3][i];
        io_list = fisBuildIoList(fis->in_n, in_mf_n);
        free(in_mf_n);
        fis->input = (IO **)calloc(fis->in_n, sizeof(IO *));
        for (i = 0; i < fis->in_n; i++)
                fis->input[i] = io_list+i;

        /* create output node list */
        out_mf_n = (int *)calloc(fis->out_n, sizeof(int));
        for (i = 0; i < fis->out_n; i++)
                out_mf_n[i] = FIS_data[4][i];
        io_list = fisBuildIoList(fis->out_n, out_mf_n);
        free(out_mf_n);
        fis->output = (IO **)calloc(fis->out_n, sizeof(IO *));
        for (i = 0; i < fis->out_n; i++)
                fis->output[i] = io_list+i;

        fis->rule_n = FIS_data[5][0];

        fisGetString2(fis->andMethod, FIS_data[6], col_n);
```

```
fisGetString2(fis->orMethod, FIS_data[7], col_n);
fisGetString2(fis->impMethod, FIS_data[8], col_n);
fisGetString2(fis->aggMethod, FIS_data[9], col_n);
fisGetString2(fis->defuzzMethod, FIS_data[10], col_n);

start = 11;
/* For efficiency, I/O names and MF labels are not stored */
for (i = 0; i < fis->in_n; i++) {
        fis->input[i]->name[0] = '\0';
        for (j = 0; j < fis->input[i]->mf_n; j++)
                fis->input[i]->mf[j]->label[0] = '\0';
}
for (i = 0; i < fis->out_n; i++) {
        fis->output[i]->name[0] = '\0';
        for (j = 0; j < fis->output[i]->mf_n; j++)
                fis->output[i]->mf[j]->label[0] = '\0';
}

start = start + fis->in_n + fis->out_n;
for (i = start; i < start + fis->in_n; i++) {
        fis->input[i-start]->bound[0] = FIS_data[i][0];
        fis->input[i-start]->bound[1] = FIS_data[i][1];
}

start = start + fis->in_n;
for (i = start; i < start + fis->out_n; i++) {
        fis->output[i-start]->bound[0] = FIS_data[i][0];
        fis->output[i-start]->bound[1] = FIS_data[i][1];
}

/* update "start" to skip reading of MF labels */
for (i = 0; i < fis->in_n; start += fis->input[i]->mf_n, i++);
for (i = 0; i < fis->out_n; start += fis->output[i]->mf_n, i++);

start = start + fis->out_n;
for (i = 0; i < fis->in_n; i++)
        for (j = 0; j < fis->input[i]->mf_n; j++) {
                fisGetString2(fis->input[i]->mf[j]->type, FIS_data[start], col_n);
                start++;
        }

for (i = 0; i < fis->out_n; i++)
        for (j = 0; j < fis->output[i]->mf_n; j++) {
                fisGetString2(fis->output[i]->mf[j]->type, FIS_data[start], col_n);
                start++;
        }
```

```
fisAssignMfPointer(fis);
fisAssignFunctionPointer(fis);

/* get input MF parameters */
for (i = 0; i < fis->in_n; i++) {
        for (j = 0; j < fis->input[i]->mf_n; j++) {
                for (k = 0; k < MF_PARA_N; k++)
                        fis->input[i]->mf[j]->para[k] =
                                FIS_data[start][k];
                start++;
        }
}
if (strcmp(fis->type, "mamdani") == 0) {
        for (i = 0; i < fis->out_n; i++)
                for (j = 0; j < fis->output[i]->mf_n; j++) {
                        fis->output[i]->mf[j]->value_array =
                                (double *)calloc(MF_POINT_N, sizeof(double));
                        for (k = 0; k < MF_PARA_N; k++)
                                fis->output[i]->mf[j]->para[k] =
                                        FIS_data[start][k];
                        start++;
                }
        fisComputeOutputMfValueArray(fis);

} else {
        printf("fis->type = %s\n", fis->type);
        fisError("Unknown fis type!");
}

fis->rule_list = (int **)fisCreateMatrix
        (fis->rule_n, fis->in_n + fis->out_n, sizeof(int));
fis->rule_weight = (double *)calloc(fis->rule_n, sizeof(double));
fis->and_or = (int *)calloc(fis->rule_n, sizeof(int));
for (i = 0; i < fis->rule_n; i++) {
        for (j = 0; j < fis->in_n + fis->out_n; j++)
                fis->rule_list[i][j] = (int)FIS_data[start][j];
        fis->rule_weight[i] = FIS_data[start][fis->in_n+fis->out_n];
        fis->and_or[i] = (int)FIS_data[start][fis->in_n+fis->out_n+1];
        start++;
}

fis->firing_strength = (double *)calloc(fis->rule_n, sizeof(double));
fis->rule_output = (double *)calloc(fis->rule_n, sizeof(double));
if (strcmp(fis->type, "mamdani") == 0) {
        fis->BigOutMfMatrix = (double *)
```

```
                calloc(fis->rule_n*MF_POINT_N, sizeof(double));
        fis->BigWeightMatrix = (double *)
                calloc(fis->rule_n*MF_POINT_N, sizeof(double));
    }
    fis->mfs_of_rule = (double *)calloc(fis->in_n, sizeof(double));
}

/* load parameters and rule list from given FIS_data */
static void
fisLoadParameter(FIS *fis, double **FIS_data)
{
    int start;
    int i, j, k;

    start = 11 + 2*(fis->in_n + fis->out_n);
    for (i = 0; i < fis->in_n; start += fis->input[i]->mf_n, i++);
    for (i = 0; i < fis->out_n; start += fis->output[i]->mf_n, i++);
    for (i = 0; i < fis->in_n; start += fis->input[i]->mf_n, i++);
    for (i = 0; i < fis->out_n; start += fis->output[i]->mf_n, i++);

    /* get input MF parameters */
    for (i = 0; i < fis->in_n; i++) {
        for (j = 0; j < fis->input[i]->mf_n; j++) {
            for (k = 0; k < MF_PARA_N; k++)
                fis->input[i]->mf[j]->para[k] =
                    FIS_data[start][k];
            start++;
        }
    }
    /* get Mamdani output MF parameters */
    if (strcmp(fis->type, "mamdani") == 0) {
        for (i = 0; i < fis->out_n; i++)
            for (j = 0; j < fis->output[i]->mf_n; j++) {
                for (k = 0; k < MF_PARA_N; k++)
                    fis->output[i]->mf[j]->para[k] =
                        FIS_data[start][k];
                start++;
            }
        fisComputeOutputMfValueArray(fis);

    } else {
        printf("fis->type = %s\n", fis->type);
        fisError("Unknown fis type!");
    }

    for (i = 0; i < fis->rule_n; i++) {
```

```
                for (j = 0; j < fis->in_n + fis->out_n; j++)
                        fis->rule_list[i][j] = (int)FIS_data[start][j];
                fis->rule_weight[i] = FIS_data[start][fis->in_n+fis->out_n];
                fis->and_or[i] = (int)FIS_data[start][fis->in_n+fis->out_n+1];
                start++;
        }
}

/* load parameters contain in the given parameter array */
/* (Note that the array is compact, no zero padding */
static void
fisLoadParameter1(FIS *fis, double *para_array)
{
        int start = 0;
        int paraN;
        int i, j, k;

        /* get input MF parameters */
        for (i = 0; i < fis->in_n; i++)
                for (j = 0; j < fis->input[i]->mf_n; j++) {
                        paraN = fisGetMfParaN(fis->input[i]->mf[j]->type);
                        for (k = 0; k < paraN; k++)
                                fis->input[i]->mf[j]->para[k] = para_array[start++];
                }

        /* get Mamdani output MF parameters */
        if (strcmp(fis->type, "mamdani") == 0) {
                for (i = 0; i < fis->out_n; i++)
                        for (j = 0; j < fis->output[i]->mf_n; j++) {
                                paraN = fisGetMfParaN(fis->input[i]->mf[j]->type);
                                for (k = 0; k < paraN; k++)
                                        fis->output[i]->mf[j]->para[k] =
para_array[start++];
                        }
                fisComputeOutputMfValueArray(fis);

        } else {
                printf("fis->type = %s\n", fis->type);
                fisError("Unknown fis type!");
        }
}

/* Returns pointer if there is a match; otherwise signals error */
static FIS *
fisMatchHandle(FIS *head, int handle)
{
```

```
        FIS *p;

        for (p = head; p != NULL; p = p->next)
                if (p->handle == handle)
                        break;
        if (p == NULL) {
                printf("Given handle is %d.\n", handle);
                fisError("Cannot find an FIS with this handle.");
        }
        return(p);
}


/* If more than two are qualified, the largest handle is returned.  */
static FIS *
fisMatchName(FIS *head, char *name)
{
        FIS *p, *matched_p = NULL;

        for (p = head; p != NULL; p = p->next)
                if (strcmp(p->name, name) == 0)
                        matched_p = p;
        return(matched_p);
}

static int
fisFindMaxHandle(FIS *head)
{
        FIS *p;
        int max_handle = 0;

        if (head == NULL)
                return(0);

        for (p = head; p != NULL; p = p->next)
                if (p->handle > max_handle)
                        max_handle = p->handle;
        return(max_handle);
}


/* Compute MF values for all input variables */
static void
fisComputeInputMfValue(FIS *fis)
{
        int i, j;
```

```
        MF *mf_node;

        for (i = 0; i < fis->in_n; i++)
                for (j = 0; j < fis->input[i]->mf_n; j++) {
                        mf_node = fis->input[i]->mf[j];
                        if (!mf_node->userDefined)
                                mf_node->value = (*mf_node->mfFcn)
                                        (fis->input[i]->value, mf_node->para);
                        else {

                                printf("Given MF %s is unknown.\n", mf_node->label);
                                fisError("Exiting ...");

                        }
                }
}
/* Compute firing strengths */
static void
fisComputeFiringStrength(FIS *fis)
{
        double out = 0, mf_value;
        int i, j, which_mf;

        /* Compute original firing strengths via andFcn or orFcn */
        for (i = 0; i < fis->rule_n; i++) {
                if (fis->and_or[i] == 1) {      /* AND premise */
                        for (j = 0; j < fis->in_n; j++) {
                                which_mf = fis->rule_list[i][j];
                                if (which_mf > 0)
                                        mf_value =fis->input[j]->mf[which_mf-1]->value;
                                else if (which_mf == 0) /* Don't care */
                                        mf_value = 1;
                                else            /* Linguistic hedge NOT */
                                mf_value = 1 - fis->input[j]->mf[-which_mf-1]->value;
                                fis->mfs_of_rule[j] = mf_value;
                        }
                        if (!fis->userDefinedAnd)
                                out = fisArrayOperation(
                                        fis->mfs_of_rule, fis->in_n, fis->andFcn);
                        else {

                        printf("Given AND method %s is unknown.\n", fis->andMethod);
                                fisError("Exiting ...");

                        }
                } else {                        /* OR premise */
```

```
                for (j = 0; j < fis->in_n; j++) {
                        which_mf = fis->rule_list[i][j];
                        if (which_mf > 0)
                                mf_value =fis->input[j]->mf[which_mf-1]->value;
                        else if (which_mf == 0) /* Don't care */
                                mf_value = 0;
                        else            /* Linguistic hedge NOT */
                        mf_value = 1 - fis->input[j]->mf[-which_mf-1]->value;
                        fis->mfs_of_rule[j] = mf_value;
                }
                if (!fis->userDefinedOr)
                        out = fisArrayOperation(
                                fis->mfs_of_rule, fis->in_n, fis->orFcn);
                else {

                printf("Given OR method %s is unknown.\n", fis->orMethod);
                        fisError("Exiting ...");

                        }
                }
                fis->firing_strength[i] = out;
        }


        for (i = 0; i < fis->rule_n; i++)
                fis->firing_strength[i] =
                        fis->rule_weight[i]*fis->firing_strength[i];
}



static void
fisFinalOutputMf2(FIS *fis, int m, double *aggMF)
{
        int i, j, which_mf;


        for (i = 0; i < fis->rule_n; i++) {
                which_mf = fis->rule_list[i][fis->in_n+m];
                if (which_mf > 0)
                        for (j = 0; j < MF_POINT_N; j++)

                                fis->BigOutMfMatrix[j*fis->rule_n+i] =
                                        fis->output[m]->mf[which_mf-1]->value_array[j];
                else if (which_mf < 0)
                        for (j = 0; j < MF_POINT_N; j++)
```

```
                        fis->BigOutMfMatrix[j*fis->rule_n+i] =
                                1 - fis->output[m]->mf[-which_mf-1]-
>value_array[j];
                else    /* which_mf == 0 */
                        for (j = 0; j < MF_POINT_N; j++)
                                fis->BigOutMfMatrix[j*fis->rule_n+i] = 0;
        }


        for (i = 0; i < fis->rule_n; i++)
                for (j = 0; j < MF_POINT_N; j++)
                        fis->BigWeightMatrix[j*fis->rule_n+i] =
                                fis->firing_strength[i];

        /* apply implication operator */
          if (!fis->userDefinedImp)
                for (i = 0; i < (fis->rule_n)*MF_POINT_N; i++)
                        fis->BigOutMfMatrix[i] = (*fis->impFcn)(
                                fis->BigWeightMatrix[i], fis->BigOutMfMatrix[i]);
        else {

                        printf("Given IMP method %s is unknown.\n", fis-
>impMethod);
                        fisError("Exiting ...");

        }

        /* apply MATLAB aggregate operator */
        if (!fis->userDefinedAgg)
                for (i = 0; i < MF_POINT_N; i++)
                        aggMF[i] = fisArrayOperation(
                        fis->BigOutMfMatrix+i*fis->rule_n,
                        fis->rule_n, fis->aggFcn);
        else {

                        printf("Given AGG method %s is unknown.\n", fis-
>aggMethod);
                        fisError("Exiting ...");

        }
}

/*********************************************************************
 Evaluate the constructed FIS based on given input vector
 *********************************************************************/
```

```
/* compute outputs and put them into output nodes */
static void
fisEvaluate(FIS *fis)
{
        double out = 0;
        double total_w;
        int i, j;

        if (fis == NULL) {
                printf("FIS data structure has not been built yet.\n");
                fisError("Exiting ...");
        }

        fisComputeInputMfValue(fis);
        fisComputeFiringStrength(fis);
        total_w = fisArrayOperation(fis->firing_strength, fis->rule_n, fisSum);
        if (total_w == 0) {
                printf("Warning: no rule is fired for input [");
                for (i = 0; i < fis->in_n; i++)
                        printf("%f ", fis->input[i]->value);

                printf("]!\n");
printf("Average of the range of each output variable is used as default output.\n\n");
                for (i = 0; i < fis->out_n; i++)
                        fis->output[i]->value = (fis->output[i]->bound[0] +
                                fis->output[i]->bound[1])/2;
                return;
        }
 if (strcmp(fis->type, "mamdani") == 0)
        for (i = 0; i < fis->out_n; i++) {
                double aggMF[MF_POINT_N];
                double X[MF_POINT_N];
                double min = fis->output[i]->bound[0];
                double max = fis->output[i]->bound[1];
                double step = (max - min)/(MF_POINT_N - 1);
                for (j = 0; j < MF_POINT_N; j++)
                        X[j] = min + step*j;
                /* fill in aggMF */
                fisFinalOutputMf2(fis, i, aggMF);
                /* defuzzification */
                if (!fis->userDefinedDefuzz)
                        out = (*fis->defuzzFcn)(fis, i, aggMF);
                else { /*  user defined defuzzification */

        printf("Given defuzzification method %s is unknown.\n", fis->defuzzMethod);
```

```
                        fisError("Exiting ...");

                }
                fis->output[i]->value = out;
        }
        else {
        printf("Given %s is unknown.\n", fis->name);
        fisError("Exiting ...");
        }
}

/* given input vector and data structure, return output */

static void
Final_Output(double *input, FIS *fis, double *output)
{
        int i;

        /* dispach input */
        for (i = 0; i < fis->in_n; i++)
                fis->input[i]->value = input[i];

        fisEvaluate(fis);

        for (i = 0; i < fis->out_n; i++)
                output[i] = fis->output[i]->value;
}
/* return the next valid line without comments */
static char *
getNextLine(char *buf, FILE *fp)
{
        char  *returned_value;
        int i, j;

        returned_value = fgets(buf, GET_FIS, fp);
        if (NULL == returned_value)
                return(NULL);

                if (buf[0] == '%' || buf[0] == '\n')
                return(getNextLine(buf, fp));


        for (i = 0; buf[i]!='%' && buf[i]!='\n' && i < GET_FIS; i++);
                for (j = i; j < GET_FIS; j++)
                buf[j] = 0;
```

```
        return(returned_value);
}

static double
getNumber(char *buf, FILE *fp)
{
        int tmp;
        char string[GET_FIS];
        double num;

        if (getNextLine(buf, fp) == NULL)
                fisError("getNumber: Incomplete FIS file!");

        tmp = sscanf(buf, " %[^=] = %lf ", string, &num);
        if (tmp != 2) {
                printf("Error format in FIS file when parsing\n");
                printf("\"%s\"\n", buf);
                fisError("Error in getNumber().");
        }

        return(num);
}

/* find string x in "*******='x'" */
static void
getString(char *buf, FILE *fp, double *array)
{
        int i;
        char string1[GET_FIS];
        char string2[GET_FIS];
        int tmp;

        if (getNextLine(buf, fp) == NULL)
                fisError("getString: Incomplete FIS file!");

        tmp = sscanf(buf, " %[^'] '%[^']' ", string1, string2);
        if (tmp != 2) {
                printf("Error format in FIS file when parsing\n");
                printf("\"%s\"\n", buf);
                fisError("Error in getString().");
        }

        /* copy it to output array */
        for (i = 0; i < (int)strlen(string2); i++)
                array[i] = string2[i];
```

```
}

/* put a string "a b c" to an array [a b c]*/
/* return number of elements */
static int
getArray(char *string, double *array)
{
        int i;
        int start, end, index;
        char tmp[GET_FIS];

        start = 0;      /* start of a number */
        end = 0;        /* end of a number */
        index = 0;      /* index of array */
        while (start <= (int)strlen(string)-1) {
                /* find end */
                for (end = start; end < (int)strlen(string); end++)
                        if (string[end] == ' ')
                                break;
                for (i = start; i <= end; i++)
                        tmp[i-start] = string[i];
                tmp[i-start] = 0;
                array[index++] = atof(tmp);
                /* find new start */
                for (start = end; start < (int)strlen(string); start++)
                        if (string[start] != ' ')
                                break;
        }

        return(index);
}

static void
getMfN(char *filename, int in_n, double *in_mf_n, int out_n, double *out_mf_n)
{
        int i, tmp;
        char buf[GET_FIS];
        FILE *fp = fisOpenFile(filename, "r");
        for (i = 0; i < in_n+out_n; i++) {
                while (1) {
                        if (getNextLine(buf, fp) == NULL)
                                fisError("Not enough NumMFs in FIS file!");
                        if (sscanf(buf, " NumMFs = %d ", &tmp) == 1)
                                break;
                }
                if (i < in_n)
```

```
                in_mf_n[i] = tmp;
        else
                out_mf_n[i-in_n] = tmp;
    }
    fclose(fp);

}

/* return an empty matrix with right size */
static double **
returnEmptyFIS_data(char *filename, int *row_n_p, int *col_n_p)
{
    int in_n, out_n, rule_n, total_in_mf_n, total_out_mf_n;
    int row_n, col_n;
    char buf[GET_FIS], fisType[GET_FIS];
    char fisName[GET_FIS], IoName[GET_FIS];
    char tmp1[GET_FIS], tmp2[GET_FIS], tmp3[GET_FIS], tmp4[GET_FIS];
    FILE *fp;
    double *in_mf_n;
    double *out_mf_n;
    double **FIS_data;

        fp = fisOpenFile(filename, "rt");
        while (1) {
        if (getNextLine(buf, fp) == NULL)
                fisError("Cannot find NumInputs in FIS file!");
        if (sscanf(buf, " NumInputs = %d ", &in_n) == 1)
                break;
    }
        while (1) {
        if (getNextLine(buf, fp) == NULL)
                fisError("Cannot find NumOutputs in FIS file!");
        if (sscanf(buf, " NumOutputs = %d ", &out_n) == 1)
                break;
    }
        while (1) {
        if (getNextLine(buf, fp) == NULL)
                fisError("Cannot find NumRules in FIS file!");
        if (sscanf(buf, " NumRules = %d ", &rule_n) == 1)
                break;
    }
    fclose(fp);
    in_mf_n = (double *)calloc(in_n, sizeof(double));
    out_mf_n = (double *)calloc(out_n, sizeof(double));
    getMfN(filename, in_n, in_mf_n, out_n, out_mf_n);
```

```
        row_n = 11 + 2*(in_n+out_n) +
                3*(total_in_mf_n + total_out_mf_n) + rule_n;
        free(in_mf_n);
        free(out_mf_n);

                fp = fisOpenFile(filename, "rt");
                while (1) {
                if (getNextLine(buf, fp) == NULL)
                        fisError("Cannot find FIS Name in FIS file!");
                if (sscanf(buf, " Name = '%[^']' ", fisName) == 1)
                        break;
        }
        col_n = (int)strlen(fisName);
        col_n = MAX(col_n, 8);          /* 'centroid' defuzzification */
                while (1) {
                if (getNextLine(buf, fp) == NULL)
                        fisError("Cannot find FIS Type in FIS file!");
                if (sscanf(buf, " Type = '%[^']' ", fisType) == 1)
                        break;
        }
                while (getNextLine(buf, fp) != NULL) {
                if (sscanf(buf, " Name = '%[^']' ", IoName) == 1)
                        col_n = MAX(col_n, (int)strlen(IoName));
                if (sscanf(buf, " %[^'] '%[^']' : '%[^']' , [ %[^]] ",
                        tmp1, tmp2, tmp3, tmp4) == 4) {
                        col_n = MAX(col_n, (int)strlen(tmp2));
                        col_n = MAX(col_n, (int)strlen(tmp3));
                }
        }
        if (!strcmp(fisType, "mamdani"))
                col_n = MAX(col_n, MF_PARA_N);
        else
                fisError("Unknown FIS type!");

        fclose(fp);
        *row_n_p = row_n;
        *col_n_p = col_n;
        FIS_data = (double **)fisCreateMatrix(row_n, col_n, sizeof(double));
        return(FIS_data);
}


static double **
returnFIS_data(char *fis_file, int *row_n_p, int *col_n_p)
{
```

```
int i, j, k;
FILE *fp;
char buf[GET_FIS];
char str1[GET_FIS], str2[GET_FIS], str3[GET_FIS], str4[GET_FIS];
char fisType[GET_FIS];

int in_n, out_n, rule_n;
int mf_n;

int now;
double **FIS_data;
double *in_mf_n, *out_mf_n;

FIS_data = returnEmptyFIS_data(fis_file, row_n_p, col_n_p);

        fp = fisOpenFile(fis_file, "rt");
        while (1) {
        if (getNextLine(buf, fp) == NULL)
                fisError("Cannot find [System] in FIS file!");
        if (!strcmp(buf, "[System]")) /* found it! */
                break;
}

now = 0;
getString(buf, fp, FIS_data[now++]);/* name */
getString(buf, fp, FIS_data[now++]);/* type */
for (i = 0; i < GET_FIS && FIS_data[1][i] != 0; i++)
        fisType[i] = (int) FIS_data[1][i];
fisType[i] = 0;
in_n = (int)getNumber(buf, fp);
out_n = (int)getNumber(buf, fp);

FIS_data[now][0] = (double) in_n;
FIS_data[now][1] = (double) out_n;
now++;

in_mf_n = (double *)calloc(in_n, sizeof(double));
out_mf_n = (double *)calloc(out_n, sizeof(double));
getMfN(fis_file, in_n, in_mf_n, out_n, out_mf_n);
for (i = 0; i < in_n; i++)
        FIS_data[now][i] = in_mf_n[i];
now++;
for (i = 0; i < out_n; i++)
        FIS_data[now][i] = out_mf_n[i];
now++;
rule_n = (int)getNumber(buf, fp);
```

```
FIS_data[now++][0] = (double) rule_n;
getString(buf, fp, FIS_data[now++]);/* and method */
getString(buf, fp, FIS_data[now++]);/* or method */
getString(buf, fp, FIS_data[now++]);/* imp method */
getString(buf, fp, FIS_data[now++]);/* agg method */
getString(buf, fp, FIS_data[now++]);/* defuzz method */
fclose(fp);

fp = fisOpenFile(fis_file, "rt");
while (1) {
        if (getNextLine(buf, fp) == NULL)
                fisError("Cannot find the first Name in FIS file!");
        if (sscanf(buf, " Name = '%[^']' ", str1) == 1)
                break;
}
for (i = 0; i < in_n+out_n; i++) {
        while (1) {
                if (getNextLine(buf, fp) == NULL)
                        fisError("Not enough Name in FIS file!");
                if (sscanf(buf, " Name = '%[^']' ", str1) == 1)
                        break;
        }
        for (j = 0; j < (int)strlen(str1); j++)
                FIS_data[now][j] = str1[j];
        now++;
}
fclose(fp);

fp = fisOpenFile(fis_file, "rt");
for (i = 0; i < in_n+out_n; i++) {
        while (1) {
                if (getNextLine(buf, fp) == NULL)
                        fisError("Not enough Range in FIS file!");
                if (sscanf(buf, " Range = [ %[^]] ", str1) == 1)
                        break;
        }
        if (getArray(str1, FIS_data[now++]) != 2)
                fisError("Error in parsing I/O ranges.");
}
fclose(fp);

fp = fisOpenFile(fis_file, "rt");
for (i = 0; i < in_n+out_n; i++) {
        mf_n = i < in_n? in_mf_n[i]:out_mf_n[i-in_n];
        for (j = 0; j < mf_n; j++) {
                while (1) {
```

```
                    if (getNextLine(buf, fp) == NULL)
                        fisError("Not enough MF Labels in FIS file!");
                    if (sscanf(buf, " %[^']'%[^']' : '%[^']' , [ %[^]] ",
                        str1, str2, str3, str4) == 4)
                        break;
                }
                for (k = 0; k < (int)strlen(str2); k++)
                    FIS_data[now][k] = str2[k];
                now++;
        }
}
fclose(fp);

fp = fisOpenFile(fis_file, "rt");
for (i = 0; i < in_n+out_n; i++) {
        mf_n = i < in_n? in_mf_n[i]:out_mf_n[i-in_n];
        for (j = 0; j < mf_n; j++) {
                while (1) {
                    if (getNextLine(buf, fp) == NULL)
                        fisError("Not enough MF types in FIS file!");
                    if (sscanf(buf, " %[^']'%[^']' : '%[^']' , [ %[^]] ",
                        str1, str2, str3, str4) == 4)
                        break;
                }
                for (k = 0; k < (int)strlen(str3); k++)
                    FIS_data[now][k] = str3[k];
                now++;
        }
}
fclose(fp);

        fp = fisOpenFile(fis_file, "rt");
for (i = 0; i < in_n+out_n; i++) {
        mf_n = i < in_n? in_mf_n[i]:out_mf_n[i-in_n];
        for (j = 0; j < mf_n; j++) {
                while (1) {
                    if (getNextLine(buf, fp) == NULL)
                        fisError("Not enough MF parameters in FIS file!");
                    if (sscanf(buf, " %[^']'%[^']' : '%[^']' , [ %[^]] ",
                        str1, str2, str3, str4) == 4) {

                        break;
                    }
                }
                if (i < in_n) {
                    if (getArray(str4, FIS_data[now]) > MF_PARA_N) {
```

```
                                        fisError("Error in parsing input MF parameters.");
                                }
                        } else {
                                if (!strcmp(fisType, "mamdani")) {
                                        if (getArray(str4, FIS_data[now]) > MF_PARA_N)
{
                                        fisError("Error in parsing output MF parameters.");
                                        }
                                }

                        }
                        now++;
                }
        }
        fclose(fp);

        fp = fisOpenFile(fis_file, "rt");
                while (1) {
                if (getNextLine(buf, fp) == NULL)
                        fisError("Cannot find [Rules] in FIS file!");
                if (!strcmp(buf, "[Rules]")) /* found it! */
                        break;
        }
        for (i = 0; i < rule_n; i++) {
                if (getNextLine(buf, fp) == NULL)
                        fisError("Not enough rule list in FIS file!");
                /* get rid of ",", "(" and ")" */
                for (j = 0; j < (int)strlen(buf); j++)
                        if (buf[j]==',' || buf[j]=='(' || buf[j]==')' || buf[j]==':')
                                buf[j] = ' ';
                if (getArray(buf, FIS_data[now++]) != in_n + out_n + 2) {

                        fisError("Error in parsing rule list!");
                }
        }
        fclose(fp);


}

double **returnInput_data(int *row_n_p, int *col_n_p, float *P_2, float Po_1, float
Fo_1)
{
        double **Input_data;
        int row_n = 1, col_n = 4, j;
```

```
float vout1_sgl[5], vout1_max, vout1_min, FO, vout1;
float vout2_sgl[5], vout2_max, vout2_min, vout2;
float vout3_sgl[5], vout3_max, vout3_min, PO, vout3,Po_2;

Input_data = (double **)fisCreateMatrix(row_n, col_n, sizeof(double));
        vout1 = 0;
        vout2 = 0;
        vout3 = 0;

        {
                delay(1);
                for(j=0; j<5; j++)
                {
do{
                        iv = adin (1);
                        vout1_sgl[j] = itov (iv);
                }while(vout1_sgl[j] < -4.0 || vout1_sgl[j] >4.0);
                do{
                        iv = adin (2);
                        vout2_sgl[j] = itov (iv);
                }while(vout2_sgl[j] < -4.0 || vout2_sgl[j] >4.0);

                        do{
                        iv = adin (3);
                        vout3_sgl[j] = itov (iv);
                        vout3_sgl[j] += 0.2;
                        }while(vout3_sgl[j] <= 0 || vout3_sgl[j] >1.5);
                }
vout1_min = vout1_sgl[0];
                        vout1_max = vout1_sgl[0];
                        vout2_min = vout2_sgl[0];
                        vout2_max = vout2_sgl[0];
                        vout3_min = vout3_sgl[0];
                        vout3_max = vout3_sgl[0];

                for(j=1; j<5; j++)
                {
                        if (vout1_sgl[j] < vout1_min) vout1_min = vout1_sgl[j];
                        if (vout1_sgl[j] > vout1_max) vout1_max = vout1_sgl[j];

                        if (vout2_sgl[j] < vout2_min) vout2_min = vout2_sgl[j];
                        if (vout2_sgl[j] > vout2_max) vout2_max = vout2_sgl[j];

                        if (vout3_sgl[j] < vout3_min) vout3_min = vout3_sgl[j];
                        if (vout3_sgl[j] > vout3_max) vout3_max = vout3_sgl[j];
                }
```

```
                    vout1 = vout2 + (vout1_sgl[0] + vout1_sgl[1] + vout1_sgl[2] +
vout1_sgl[3]\
                    + vout1_sgl[4] - vout1_min - vout1_max) / 3.;

                    vout2 = vout2 + (vout2_sgl[0] + vout2_sgl[1] + vout2_sgl[2] +
vout2_sgl[3]\
                    + vout2_sgl[4] - vout2_min - vout2_max) / 3.;

                    vout3 = vout3 + (vout3_sgl[0] + vout3_sgl[1] + vout3_sgl[2]
        + vout3_sgl[3]\
                    + vout3_sgl[4] - vout3_min - vout3_max) / 3.;

            } //end of samples
                    PO = vout3 ;

                        if (FK-Fk_1 < 0.) Input_data[0][1]=- Input_data[0][1]; */
                            Input_data[0][1] = PO- Po_1;
                            Input_data[0][2] = (FO/F_ref)-1;
                            Input_data[0][3] = FO-Fo_1;
                        }
                                    if(Select==1){
                                        Input_data[0][0] = 2.5*(P_ref-PO);
            Input_data[0][1] = PO- Po_1;
                                        Po_2=Po_1;
                                        }
                                    if(Select==2){
                                        Input_data[0][0] = 1-(FO/F_ref);
                                        Input_data[0][1] = FO-Fo_1;
                                        }

        *row_n_p = row_n;
        *col_n_p = col_n;
        *P_2=Po_2;
        return(Input_data);
}



int main()
{
        int coory, coorx, dev=25;
        FIS *fis;
        int i, j,kk;
        int coory_0,coory_1,coory_2,coory_3, coorx_0,coorx_1, coorx_2,coorx_3;
        float u, Po_1,Fo_1,Po_st[Dim], uk_st[Dim],delta_power[Dim],Power,lam;
```

```
        float
Fo_st[Dim],Force,P_2,ep_1,ep_2,tun1,tun2,tun3,delta_Force[Dim],Speedf[Dim];
        double **Input_data, **FIS_data, **Output_data;
        char *fis_file, outputtext[20];
        int fis_row_n, fis_col_n, data_row_n, data_col_n, k=0;
        FILE *dat_out;
        /*********************************************************/

/*************************************************************/


      /* recode monitor type */
if ((biosequip()&0x30)==0x30) {monochrome=ON;textmode(MONO);}
else {monochrome=OFF;textmode(C80);}
drawscreen();
 showhelp(1);
showmenu(menuI);
selectSENSOR();

while (rdkey!=F10) {

        menuselect();
}
u=init();
        if (Select==3)
        fis_file = ("c:\pf275.fis");
         if (Select==1)
         fis_file = ("c:\mill.fis");
          if (Select==2)
          fis_file = ("c:\for274.fis");
          rename("output.txt", "output.bak");
                if ((dat_out = fopen("output.txt","wt")) == NULL)
        {
                fprintf(stderr, "Cannot open output data file.\n");
                exit(1);
        }
        /* obtain data matrix and FIS matrix */
        FIS_data = returnFIS_data(fis_file, &fis_row_n, &fis_col_n);
        /* build FIS data structure */
        fis = (FIS *)calloc(1, sizeof(FIS));
        if (fis == NULL)
                  printf("Not enough memory, no allocation!\n");
        fisBuildFisNode(fis, FIS_data, fis_col_n);
        calncon(u,&coory_0,&coory_1, &coory_2, &coory_3, &coorx_1, &Po_1,
&Fo_1);
                        coorx_3=coorx_2=coorx_0=coorx_1;
```

```
        Again:
    Input_data = returnInput_data(&data_row_n, &data_col_n, &P_2, Po_1, Fo_1);


            if(Select==3){
            Power=P_ref-(Input_data[0][0]/2.5);
            Po_1=Power;
            Force=F_ref *( 1-Input_data[0][2]);
            Fo_1=Force;
            }
                if(Select==1){
                Power=P_ref-(Input_data[0][0]/2.5);
                Po_1=Power;
                }
                 if(Select==2){
                 Force=F_ref *( 1-Input_data[0][0]);
                 Fo_1=Force;
                 }

                        /* error checking */

if (InPut_N < fis->in_n) {
  printf("Given FIS is a %d-input %d-output system.\n",
        fis->in_n, fis->out_n);
        printf("Given data file does not have enough input entries.\n");
                fisFreeMatrix((void **)Input_data, data_row_n);
                fisFreeMatrix((void **)FIS_data, fis_row_n);
                fisFreeFisNode(fis);
                fisError("Exiting ...");
        }

        /* create output matrix */
Output_data = (double **)fisCreateMatrix(1, fis->out_n, sizeof(double));
ep_1=P_ref-Po_1;
ep_2=P_ref-P_2;
if ( Po_1 != P_2){
tun1= fabs((Power-Po_1)/(Po_1-P_2));
printf("tun1 = %f\n", tun1);
tun2= fabs(ep_1);
printf("tun2 = %f\n", tun2);
tun3= fabs(ep_2);
printf("tun3 = %f\n", tun3);
if (tun1 > 1){
 if ((Power < Po_1 && Po_1 < P_2) || (Power > Po_1 && Po_1 > P_2)){
        if (tun2 <= tun3 )
         lam= pow((1/tun1),0.13);
```

```
        else
        lam=pow(tun1,0.13);
                                }
        else
        lam = 1.1;
                        }
         else
        lam = 1.0;
                        }
        else
         lam = 1.0;


uk_st[k]=(lam*0.8*Output_data[0][0])+(1.*u);
Speedf[k]=(lam*150.*-Output_data[0][1])+(1.*Speed);
Po_st[k]=Power;
Fo_st[k]=Force;
delta_power[k]= Input_data[0][1];
if (Select==3 || Select==2)
 delta_Force[k]= Input_data[0][3];

else
delta_Force[k]= Input_data[0][1];
        if( uk_st[k] < Feed_LowerLimit)
uk_st[k] = Feed_LowerLimit;
        else
                if ( uk_st[k] > Feed_UpLimit)
                uk_st[k] = Feed_UpLimit;
u=uk_st[k];
Feed = (int)(((one_hundred-zero)*uk_st[k]) + zero);

if( Speedf[k] <  lower_speed)           //u[0] in percentage/10.
        Speedf[k] =  lower_speed;
        else
        if ( Speedf[k] > upper_speed)
        Speedf[k] = upper_speed;
        Speed=(int)Speedf[k];


// Write to port 1 of D/A to change feedrate.
daout(1, Feed);
daout(0, Speed);
NoPixelsX = (xend-xorig)/Dim;
NoPixelsY = (yorig-yend)/ymaxscale;

setcolor(WHITE); //20
```

```
 coorx = xorig + (int)(k*NoPixelsX);
 coory = yorig - (int)(Force*NoPixelsY);
line(coorx_0, coory_0, coorx, coory);
coorx_0=coorx;
coory_0=coory;

setcolor(RED); //20
coorx = xorig + (int)(k*NoPixelsX);
 coory = yorig - (int)(Power*NoPixelsY);
line(coorx_1, coory_1, coorx, coory);
coorx_1=coorx;
coory_1=coory;

setcolor(18); //10 , green 18
coory = yorig - (int)(uk_st[k]*NoPixelsY);
line(coorx_2, coory_2, coorx, coory);
coorx_2=coorx;
coory_2=coory;

setcolor(YELLOW); //10 , green 18
coory = yorig - (int)((Speedf[k]/10000.)*NoPixelsY);
line(coorx_3, coory_3, coorx, coory);
coorx_3=coorx;
coory_3=coory;
gotoxy(1,3);
 printf("  FEED= %.4f     Power= %.4f     FORCE=%.4f
Speed=%d",uk_st[k],Power,Force,Speed/10);
                if(Select==3) delay (170);

          else delay (250);
                while (k != Dim){
                 if(kbhit()) break;
                gotoxy(1,1);
        k++;


        /* clean up memory */

        fisFreeMatrix((void **)Input_data, data_row_n);
        fisFreeMatrix((void **)Output_data, data_row_n);
        goto Again;
          }

fprintf(dat_out, "Count    power    DelForce    error    feed    Speed    Force\n");
fprintf(dat_out, "-------------------------------------------------------------------\n");
```

```
        for (i=0; i<k; i++)
        {
        fprintf(dat_out, "%03d    %.3f    %.3f    %.4f    %.3f    %.3f    %.3f\n",
                        i, Po_st[i], 1-(Fo_st[i]/F_ref), delta_Force[i],
uk_st[i],Speedf[i]/10000., Fo_st[i]);

        }
        fisFreeFisNode(fis);
        fisFreeMatrix((void **)FIS_data, fis_row_n);
        fisFreeMatrix((void **)Input_data, data_row_n);
        fisFreeMatrix((void **)Output_data, data_row_n);
  sound(1000);
        delay(100);
        nosound();
        cleardevice();

        setcolor(7);
        line(xorig, yorig, xend, yorig);
        line(xorig, yorig, xorig, yend-50);
        outtextxy((xend-xorig)/2+xorig, yorig+30, "Iterations");
        outtextxy(xorig, yorig+dev, "0");
        outtextxy(xorig-dev, yorig, "0");

        NoPixelsX = (xend-xorig)/k;
        NoPixelsY = (yorig-yend)/ymaxscale;
        coorx = xorig + (int)(k*NoPixelsX);
        line(coorx, yorig, coorx, yorig+10);
        sprintf(outputtext, "%d", k);
        outtextxy(coorx, yorig+dev, outputtext);

        coory = yorig - (int)(ymaxscale*NoPixelsY);
        line(xorig, coory, xorig-10, coory);
        sprintf(outputtext, "%d", ymaxscale);
        outtextxy(xorig-dev, coory, outputtext);
        outtextxy(xorig-35, coory+20, "(HP)");

        line(xorig-50, yorig, xorig-50, yend-50);
        line(xorig-50, yorig, xorig-50-10,
         yorig);
        line(xorig-50, yend, xorig-50-10, yend);
        sprintf(outputtext, "%d", 100);
        outtextxy(xorig-50-33, coory, outputtext);
        outtextxy(xorig-50-30, yorig, "0");
        outtextxy(xorig-100, coory+20, "(mmpm)");

        moveto(xorig, yorig - (int)(Po_st[0]*NoPixelsY));
```

```
setcolor(12); //20
for (kk=1; kk<k; kk++)
{
        coorx = xorig + (int)(kk*NoPixelsX);
        coory = yorig - (int)(Po_st[kk]*NoPixelsY);
        lineto(coorx, coory);
}

moveto(xorig, yorig - (int)(Fo_st[0]*NoPixelsY));
setcolor(WHITE); //20
for (kk=1; kk<k; kk++)
{
        coorx = xorig + (int)(kk*NoPixelsX);
        coory = yorig - (int)(Fo_st[kk]*NoPixelsY);
        lineto(coorx, coory);
}
coory = yorig - (int)(0.9*NoPixelsY);
setcolor(11);
line(xorig, coory, xend, coory);
setcolor(12);
outtextxy(xorig+22, yorig+50, "1) Power");

moveto(xorig, yorig - (int)(uk_st[0]*NoPixelsY));
setcolor(18); //10 , green 18
for (kk=1; kk<k; kk++)
{
        coorx = xorig + (int)(kk*NoPixelsX);
        coory = yorig - (int)(uk_st[kk]*NoPixelsY);
        lineto(coorx, coory);
}
outtextxy(xorig+170, yorig+50, "2) Feedrate");

moveto(xorig, yorig - (int)((Speedf[0]/10000.)*NoPixelsY));
setcolor(YELLOW); //10 , green 18
for (kk=1; kk<k; kk++)
{
        coorx = xorig + (int)(kk*NoPixelsX);
        coory = yorig - (int)((Speedf[kk]/10000.)*NoPixelsY);
        lineto(coorx, coory);
}
outtextxy(xorig+415, yorig+50, "3) Speed");
getch();
daout(1, zero);
daout(0, 0);
getch();
closegraph();
```

```
daout(0, 0);
 return 0;
```

C.3 A/D, D/A Converter Driver

```
#define base  0x0220
#define ad_low base+4
#define ad_high base+5
#define da_low base+6
#define da_high base+7
#define mux base+10
#define control base+11
#define trigger base+12

void reset_da()
{
int l,h;
l = 2048&0xff;
h = (2048&0xff00)>>8;
outportb(ad_low,l);
outportb(ad_high,h);
outportb(da_low,l);
outportb(da_high,h);
}

void reset_ad()
{
//int dummy;
inportb(ad_low);
outportb(control,0x01);
}


void daout(int ch,int val)
{
int low_byte,high_byte,vv;
low_byte = val & 0xff;
high_byte = (val & 0x0f00)>>8;
if(ch == 0)
{
outportb(ad_low,low_byte);
outportb(ad_high,high_byte);
}
if(ch== 1)
```

```
{
outportb(da_low,low_byte);
outportb(da_high,high_byte);
}
} /*daout */


int adin(int ch)
{

int low_byte,high_byte,vv;
outportb(mux,ch);
outportb(trigger,0);
while((high_byte=inportb(ad_high))>15)
high_byte= inportb(ad_high);
low_byte = inportb(ad_low);
high_byte = (high_byte &0xf) << 8;
return(low_byte|high_byte);

} /* adin */


int vtoi(float volts)
{
return(ceil(volts/5.0*2048)+2047);
}

float itov(int iv)
{
return((iv-2048)*5./2048.);
}

float i_to_deg(int iv,float cp1,float cp2)
{
return(iv*cp1+cp2);
}
```