To the Graduate Council:

I am submitting herewith a thesis written by Yongmei Wang entitled "Self-tuning Fuzzy-PID controller Design." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

J.D.Birdwell

J.D.Birdwell, Major Professor

We have read this thesis
and recommend its acceptance:

J.M. Bailey

Fei-Wei Wang

Accepted for the Council:

CW Minkel

Associate Vice Chancellor
and Dean of The Graduate School

SELF-TUNING FUZZY-PID CONTROLLER DESIGN

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Yongmei Wang

August, 1993

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to my major advisor, Dr. J. Douglas Birdwell, for his unusual patience, understanding, great encouragement and continued support throughout all phases of this thesis.

I am very thankful to Dr. Tse-Wei Wang, for her friendship and precious technical advice. I also want to acknowledge Dr. J. M. Bailey for his suggestions in the research and Dr. Dragana Brzakovic for her support and guidance through my first year of graduate study.

My deep appreciation goes to my fellow graduate students in the control research group: Zhibin Duan, Roger Horn, Jovan Ilic, Pramit Sarma, Gaiye Zhou and Yunzhou Zhu, for their friendship and constructive suggestions.

During my graduate study, I received invaluable suggestions and help from Dr. M. O. Pace, Dr. D. W. Bouldin, Dr. M. A. Abidi and Dr. Y. K. Kuo. My special appreciation also goes to my caring and supportive friends: Chandra Tan, Wayne Gao, Frank Zhang and Mr. and Mrs. Robert Pederson.

Most of all, I wish to express my deepest gratitude to my parents, my sister and brother-in-law, who have been watching every step in my life and giving me unconditional love and support all the time.

This research project is sponsored by Measurement and Control Engineering Center, University of Tennessee, Knoxville.

ABSTRACT

A self-tuning Fuzzy-PID structure is proposed as a new controller model. While it maintains the desirable characteristics of PID control, the Fuzzy-PID controller provides additional flexibility with its nonlinear gain characteristic and adaptive operation. Although the PID controller is by far the most common control algorithm and widely used in many industries, it is often poorly tuned, and the constant gain structure limits its control capability. Fuzzy control (FC) has received much attention and can be found in many applications, but the stability and robustness aspects of FC are not well-understood. FC has not been utilized in many real-time applications where safety is a key requirement. A new approach that merges these two control algorithms is proposed to take advantage of both the well developed PID structure and the potential of fuzzy knowledge representation. The control signal is generated by an input-dependent PID controller whose gain parameters are obtained from inference on a fuzzy rulebase. The rulebase can be adapted to different plants through successive refinement learning. Properly designed Fuzzy-PID controllers can achieve better performance and have greater control design flexibility than traditional controllers. An analysis of the Fuzzy-PID controller's stability and robustness properties in a closed-loop setting is provided.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

This thesis presents a self-tuning Fuzzy-PID controller capable of improved performance relative to traditional PID control approaches. In order to meet stringent control requirements, many automatic and intelligent control approaches have been developed over the past several decades. Among these are auto-tuning PID and fuzzy control methods. A new approach implementing fuzzy logic in the PID tuning process is proposed which achieves superior results compared to those possible by using either PID or fuzzy control alone. A successive refinement machine learning approach is used to develop and optimize a fuzzy rulebase. Stability and robustness are analyzed from a theoretical standpoint. Several plants are presented as case studies to compare the results of different controller designs.

PID controllers are found in large numbers in all industries. Most feedback loops are controlled by this algorithm or minor variations of it [2]. Experience has led to the development of a number of effective tuning methods. In Chapter 2, a short review of PID control is given and two commonly used tuning methods, Ziegler-Nichols and ITAE, are illustrated as the conventional PID tuning approaches. A simulated annealing procedure is also provided as a PID gain optimization tool.

In Chapter 3, several simulation tests are done to demonstrate system performance under PID control. The Ziegler-Nichols time response method and ITAE method are applied. A simulated annealing optimization algorithm is implemented to find an optimal PID controller. This provides a basis for comparison with the proposed Fuzzy-PID method.

Fuzzy control (FC) is based on fuzzy logic – a logical system which is much closer in spirit to human thinking and natural language than traditional logical systems. Fuzzy logic provides a means to convert a linguistic control strategy based on expert knowledge into an automatic control strategy [19]. In Chapter 4, the basic concepts of fuzzy sets and fuzzy operations are reviewed, and fuzzy system design is discussed. In many cases, FC can achieve better control than conventional PID methods, and FC can be considered a candidate when conventional methods fail; however, there remain some unsolved questions, such as the stability and robustness properties of fuzzy systems. A good linguistic control strategy and expert's knowledge about the system are necessary *a priori* information for FC design; encoding this information to construct a useful fuzzy rulebase requires lots of effort as well because of the lack of systematic methods to determine the large number of parameters in the fuzzy rulebase.

In Chapter 5, an approach which blends PID and fuzzy control is proposed to enhance the control and adaptation ability with respect to plant dynamic changes and external variations. As illustrated in Figure 1.1, the proposed structure is an input-

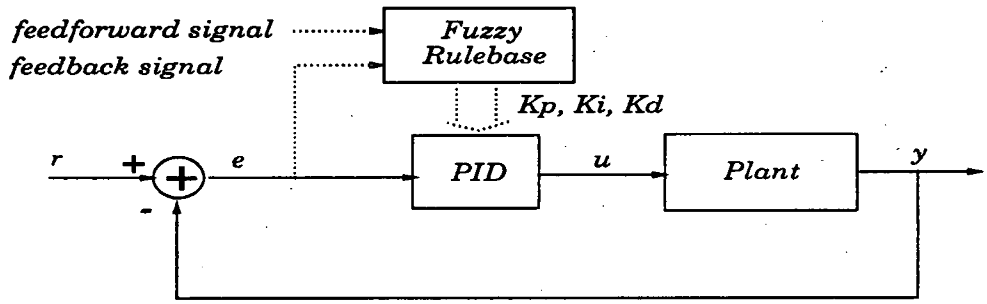


Figure 1.1: Self-tuning Fuzzy-PID controller model.

dependent PID controller. The PID gains to be used under different circumstances are encoded in the fuzzy rulebase. The control signal is generated by a modified PID controller with variable gains controlled by fuzzy rulebase inference. Usually, a fuzzy system is used to deal with approximate information. Here, it is also implemented to achieve a special non-linear mapping from the system performance or environmental changes to the PID gains. Because a fuzzy system can approximate any continuous function to any degree of accuracy, the proposed controller can emulate any conventional PID controller. For nonlinear plants, where the control signal's characteristics have a great influence on how the system performs, the Fuzzy-PID structure can provide an input signal which satisfies the design requirements when a PID controller does not. A machine learning algorithm is developed to acquire plant and environment information and to adjust the rulebase to achieve an optimal setting with respect to a specified measure of performance. Several experiments are provided to demonstrate the performance of the proposed controller structure. From the simulation, we can see that the Fuzzy-PID structure can provide superior performance compared to traditional PID control. Especially for some nonlinear plants, the system performance shows a significant improvement using the new structure. An approach to a theoretical analysis of the stability and robustness properties for fuzzy system is also provided.

CHAPTER 2

PID Control Theory

PID control is by far the most common control algorithm. PID controllers come in many different forms, are packaged as standard products, and are manufactured by the hundred thousands yearly. They are also embedded in all types of special purpose control systems. PID controllers have survived many changes in technology ranging from pneumatics to electron tubes, transistors, integrated circuits, and microprocessors [2]. This popularity is partly due to the variety of important functions that can be provided by their simple structure. However, PID controllers are often poorly tuned. For example, the difficulties involved in the tuning process frequently cause operators to switch off the derivative action in many control rooms [2], and PID control loops are often switched off and manually operated.

A number of effective tuning methods have been developed over the past several decades. These include the Ziegler-Nichols method [2], the reaction curve method [8], the Åström relay method [2], and the internal model-based control (IMC) method [27]. These methods have led to the development of many hardware devices and software packages for loop tuning which contribute significantly to the improvement of the performance of processes running under automatic control [8].

In this chapter, the basic PID control method will be presented, followed by illustration of the Ziegler-Nichols (ZN) and ITAE tuning algorithms. A simulated Annealing algorithm is also briefly reviewed. It is used in Chapter 3 to search for optimal PID parameter settings.

2.1 Basic PID control Method

PID controllers have several important functions. They provide feedback; they have the ability to eliminate steady-state offsets through integral action; they can provide anticipatory control through derivative action; and they can cope with actuator saturation [2].

A simple feedback controller has the structure shown in Figure 2.1. The *Plant* block

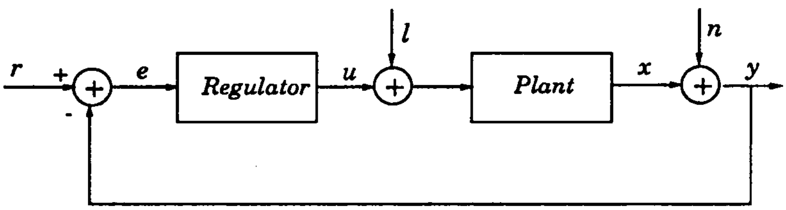


Figure 2.1: Simple feedback loop.

represents the process to be controlled. The control objective is to synthesize an input u to the plant, normally corrupted by noise l, which forces the plant output x, to follow a reference signal r. Design objectives are stated in terms of desired error dynamics between the plant output x and the reference signal r. Normally, the plant output x is not available for exact measurement. Rather, an output measurement y is made which is corrupted by measurement noise n. Therefore, the controller, or regulator, has available the error signal e which is the difference between the output measurement and the reference.

A controller is required which achieves the desired properties placed upon the error signal e, possibly within limits placed upon the plant input, or actuator signal, u. The

PID controller uses proportional, integral and derivative linear transformations of the error signal to synthesize the actuator signal. The basic form is:

$$u(t) \;=\; K \; [\; e(t) \;+\; \frac{1}{T_i} \int e(t)dt \;+\; T_d \, \frac{de(t)}{dt} \;] \tag{2.1}$$

Where

u is the control variable;

e is the control error ($e \;=\; r \;-\; y$);

r is the reference signal;

y is the plant output measurement;

K is the proportional gain;

T_i is the integration time; and

T_d is the derivative time.

The control variable is the sum of three terms, each of which has its own functionality:

- Proportional action provides the simplest form of feedback direct gain.
- Integral action tries to drive the process output to a constant reference signal at steady state; however, it decreases the stability margin.
- Derivative action predicts process output and is intended for improvement of the closed loop stability margins.

2.2 Modifications to the Basic Method

2.2.1 Integral Windup

Limitations must be placed upon the maximum and minimum achievable values of the actuator signal. These limits correspond to hardware constraints that cannot be violated. During the control process the control variable may reach these actuator limits. As a result the feedback loop is effectively broken because the actuator will remain at its limit independently of the process output. For a regulator with integral action, the resulting non-zero error will be integrated, causing internal controller variables to ramp toward positive or negative infinity. This phenomenon is called "windup" [2].

Several ways have been developed to avoid integral windup. One method presented here uses an additional feedback path to the integrator. The returned signal e_s is the difference between the controller output and actuator output. If saturation occurs, a feedback signal is generated which drives e_s to zero and prevents saturation [2].

2.2.2 Reference Values

Generally, feedback control uses the difference between the reference signal and process output measurement as an error feedback signal, as shown in Figure 2.1. With an error signal as the only feedback, the same mechanism tries to satisfy all the demands. In order to overcome this difficulty more sophisticated control systems have been developed which provide more signal paths from the set point to the system output. Here, two modifications are illustrated:

$$u = K \; [\; e_p + \frac{1}{T_i} \int e(t)dt + T_d \frac{de_d}{dt} \;] \tag{2.2}$$

- Proportional Modification

  $e_p = br - y$: The value of b affects the system response to changes in the reference signal. For b less than unity the percentage overshoot (a measure of the overcorrection caused by the controller) is usually decreased. This is at the expense of a large rise time (the time required to achieve 90% of the final value).

- Derivative Modification

  $e_d = -y$: Abrupt changes in the reference signal cause a drastic effect in its derivative dr/dt. In order to avoid this effect, only the system output is included as feedback to the derivative action.

2.2.3 Limitation of the Derivative Gain

As an attempt to prevent high amplitudes in the derivative control signal caused by high frequency components in the measurement signal, a modification which cascades a first-order low-pass system on the basic control structure with time constant T_d/N is implemented to limit the derivative gain to a maximum value of N [2]. The modified derivative term can be represented by the transfer function:

$$D(s) = -\frac{sKT_d}{1 + sT_d/N}\, y(s) \tag{2.3}$$

2.2.4 Modified PID Structure

Applying the modifications specified in the above sections, we have the form of the modified PID controller as shown in Figure 2.2; this is also the structure used in the simulation experiments.

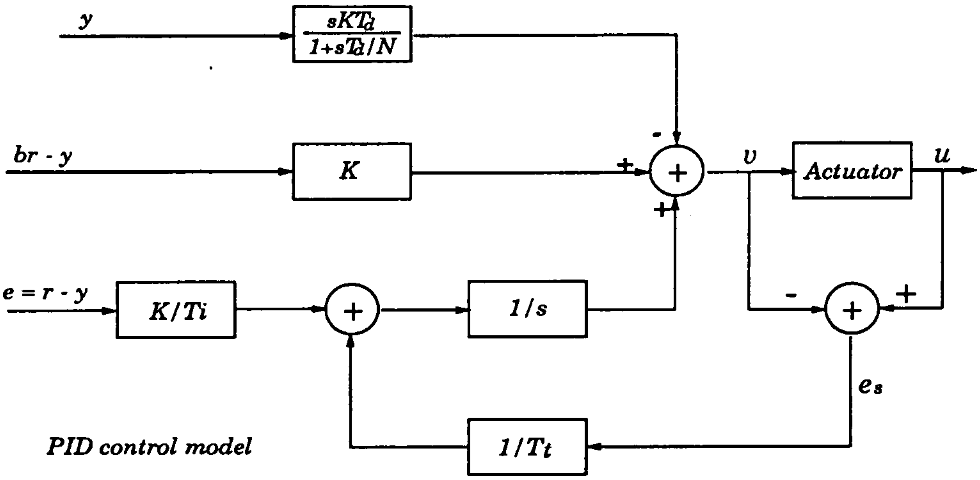


Figure 2.2: Modified PID controller model.

2.3 Ziegler-Nichols Step Response Method

The Ziegler-Nichols step and frequency response methods [2] are among the classical approaches for PID tuning. These methods characterize a system's dynamics by two parameters which are then used to select the proper PID parameters. For the step response method (ZN), two parameters are derived from the open-loop step response of the system as shown in Figure 2.3. A tangent to the plant's open loop step response at the point of maximum slope, or the inflection point, is drawn, and the intercepts of this tangent on the coordinate axes determine the parameters a and L. The PID design is based on the heuristic relationships in Table 2.1.

2.4 ITAE PID Design Method

The ITAE design method is one of several methods based on integral error criteria. Design relations for PID controllers have been developed that minimize a measure of

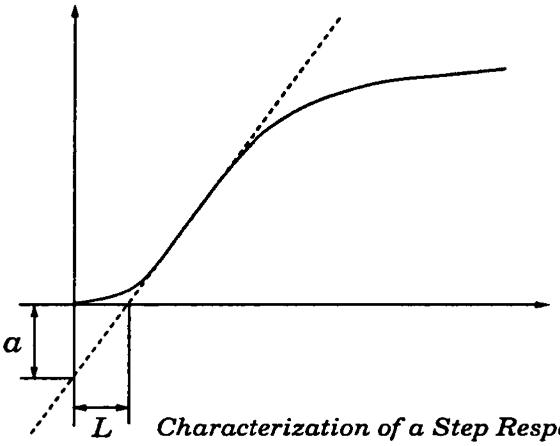


Figure 2.3: Ziegler-Nichols step response methods illustration.

Table 2.1: PID design using Ziegler-Nichols time response method.

Controller	K	T_i	T_d
P	$1/a$		
PI	$0.9/a$	$3L$	
PID	$1.2/a$	$2L$	$L/2$

total error due to a unit step change in the reference signal for simple approximate process models (basically, first-order plus time-delay) and particular types of load or set-point changes [27]. For the ITAE design method the total error is quantified by the ITAE error

$$\int_0^\infty t|e(t)|dt$$

.

A plant with stable open-loop response can be approximated as first-order with time-delay shown in Figure 2.4. The steepest tangent curve to the system open-loop

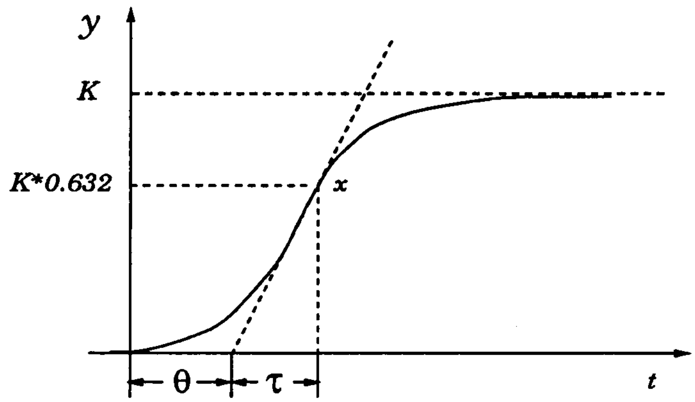


Figure 2.4: ITAE PID design illustration.

response is drawn. The parameter K is the steady state value of system response and the parameter x is the point (t,y) where the output reaches 63.2% of the final state. System open-loop stability is required. Two parameters are measured to represent system dynamics: θ, the approximate delay-time, and τ, the approximate time constant. They are used to select the PID settings. For a step change test in the reference signal, the PID controller design relations based on the ITAE performance index and a first order plus time-delay model are given by Equations 2.4, 2.5, and 2.6, with parameters

Table 2.2: PID design using ITAE method.

Mode	A	B
P	0.965	−0.85
I	0.796	−0.1465
D	0.308	0.929

A and B as listed in Table 2.2 [27]. The *Mode* column in the table specifies which is determined by the relationship in the corresponding row: proportional, integral or derivative action.

$$KK_c = A(\theta/\tau)^B \tag{2.4}$$

$$\tau/\tau_I = A + B(\theta/\tau) \tag{2.5}$$

$$\tau_D/\tau = A(\theta/\tau)^B \tag{2.6}$$

Where

K is open loop steady state gain;

K_c is the proportional gain;

τ, θ are the parameters indicated in Figure 2.4;

τ_I is the integration time constant and

τ_D is the derivative time constant.

2.5 Simulated Annealing Algorithm

A simulated annealing algorithm [18] is based on the analogy between the simulation of the process of solid annealing and problem solving for large combinatorial optimization problems. Annealing is the process of heating a solid and then slowly cooling it in order to remove the strain and crystal imperfections during which the free energy of the solid is minimized [18]. The process can be viewed as a sequence of configuration changes in the solid. The configuration of the solid can be represented by the positions and orientations of its crystalline particles. At each temperature T, the solid has a potential to reach thermal equilibrium. The probability of being in a state with energy E is given by the Boltzman distribution:

$$Pr\{\,\mathbf{E} = E\,\} = \frac{1}{Z(T)}\, exp\,(\,-\frac{E}{k_B T}\,) \tag{2.7}$$

where $Z(T)$ is a normalization factor, T is the temperature, k_B is the Boltzman constant and $exp\,(\,-\frac{E}{k_B T}\,)$ is the Boltzman factor. When the temperature decreases, the mean free energy, or the expected value of E, decreases. When the energy is zero, only the minimum state has non-zero probability of occurrence.

Simulating this evolution to thermal equilibrium, at each stage with temperature T, a random perturbation is applied which results in a difference in energy δE between the new and the old configurations. The resulting change may or may not be accepted. The probability of acceptance of this change is unity if δE is negative and $exp(\,-\frac{\delta E}{k_B T})$ if δE is positive. This acceptance rule is called the *Metropolis criterion.*

For optimization problems, a cost function C and a single real-valued control parameter c assume the roles of energy and temperature. An easily evaluated cost function is chosen for minimization and the optimization is initialized with a large control parameter c. Perturbations are randomly applied to the specified configuration, and the values of control parameter and configuration are updated on the acceptance rule until a convergence criterion is satisfied. The simulated annealing algorithm can be described as a pseudo-PASCAL code [18]:

```
                PROCEDURE SIMULATED ANNEALING

Begin
     INITIALIZE;
     M := 0;
     repeat
            PERTURB (config. i --> config. j);
            compute the change in cost ΔCij;
            if ΔCij ≤ 0 ,  then  accept
            else if  exp(-ΔCij/c) > random[0, 1],  then  accept;
            if  accept,  then  UPDATE (configuration j);
            update control factor  c_{M+1} :=  f(c_M );
            M := M + 1;
     until  stop criterion = true ( system is "frozen");
     equilibrium is approached sufficiently closely;
end.
```

Figure 2.5: Description of the simulated annealing algorithm in pseudo-PASCAL.

2.6 Discussion

In this chapter, the basic PID control algorithm and modified PID controller structure are reviewed. Ziegler-Nichols time response and ITAE methods, which are suitable for low-order plants, are given as the commonly used tuning approaches. The simulated annealing optimization algorithm is also briefly covered and will be used to find the optimal PID controller. All these tuning methods will be applied to the test plant simulations in the next chapter.

CHAPTER 3

Test Cases for PID Control

In this chapter, three linear and two nonlinear plant simulations implementing PID controller designed with the Ziegler-Nichols time response (Z-N) and ITAE methods are demonstrated using the modified PID structure in section 2.2.4. A simulated annealing optimization algorithm is also applied to obtain an optimal PID controller performance. This will form part of the basis for evaluation of the new fuzzy-PID controller design method.

For simplicity the parameters required for the modifications to the basic PID controller structure are held constant and are chosen *a priori* for all the experiments. These values are used:

- Overshoot improvement parameter: $b = 0.85$.

  b is the reference deduction parameter for the proportional feedback signal. From experiments with many plants, the value of 0.85 often results in decreased overshoot and small rise times.

- Signal amplification control factor: $T_d/N = 1$.

  To avoid high gains at high frequencies, the signal amplification limit in derivative action is arbitrarily chosen to be 1.

- Gain factor of antiwindup feedback loop: $T_t = 1$.

- Actuator operating range: $[-10, 10]$.

The cost function used in the simulated annealing optimization is of the same formula as the one that will be used in the rulebase learning process for the fuzzy-PID controller design in Chapter 5. The cost function is weighted summation of control requirements. For details, see Section 5.4.

3.1 Plant I

1. Plant Model

The test plant is a first order system with transport delay. This kind of model is often used as an approximation of plants in the chemical industry. The transfer function of the model used in this study is:

$$T(s) = \frac{e^{-9s}}{12s + 1} \tag{3.1}$$

This model forms the basis of the ITAE design methods, so one would expect an ITAE derived controller to perform well with respect to this performance criterion.

2. PID Parameters

From the open-loop system response, we can calculate the Ziegler-Nichols and ITAE PID control parameters: For the Ziegler-Nichols method, $a = 0.71$ and $L = 9$; for the ITAE design, $\theta = 9, \tau = 12.5$ and $K = 1$. The optimal PID gains are derived from SA algorithm. The PID parameters are listed in Table 3.1.

3. Results

Figure 3.1 and 3.2 shows the system performance using different controllers. The ITAE method gives excellent control for this model. The Ziegler-Nichols derived PID controller does drive the system to the set point; however, the plant oscillates for a

Table 3.1: Plant I: PID parameter settings using Ziegler-Nichols time response, ITAE and simulated annealing algorithms.

Method	Controller	K	T_i	T_d
Z-N	P	1.41		
Z-N	PI	1.26	27	
Z-N	PID	1.69	18	4.5
ITAE	PID	1.28	18.1	2.84
SA	PID	1.06	14.7	2.22

longer period of time before achieving steady state; the optimal PID controller derived from simulated annealing algorithm gives perfect control. The system reaches to steady state very quickly, with almost zero overshoot and undershoot.

3.2 Plant II

1. Plant Model

The test plant is a positional servomechanism model with two left half-plane poles and a third pole at the origin.

The transfer function of the model is:

$$T(s) = \frac{191}{s\,(s+1)\,(s+20)} \tag{3.2}$$

2. PID Parameters

Based on the Ziegler-Nichols system open-loop analysis, $a = 8.5$ and $L = 0.92$.

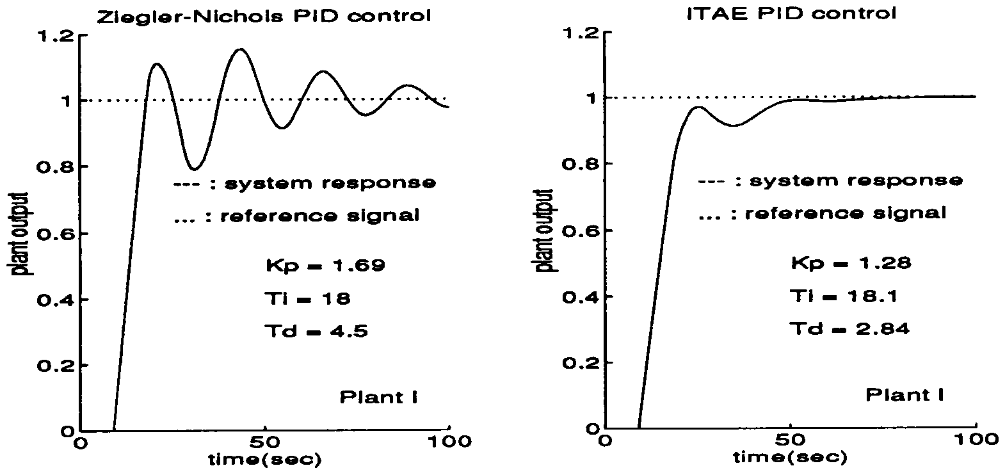


Figure 3.1: Plant I: Ziegler-Nichols and ITAE PID control.

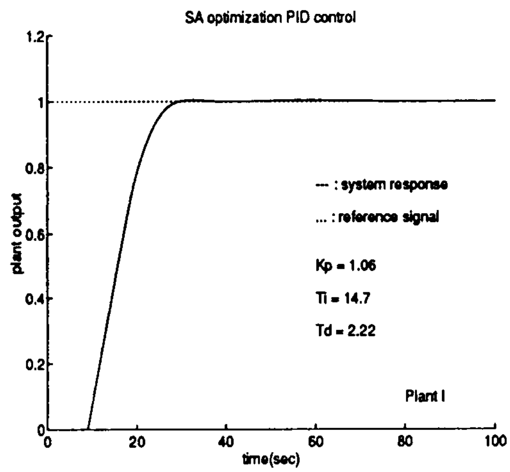


Figure 3.2: Plant I: PID controller derived from simulated annealing optimization.

Table 3.2: Plant II: PID design using Ziegler-Nichols time response and simulated annealing optimization method.

Method	Controller	K	T_i	T_d
Z-N	P	0.1176		
Z-N	PI	0.1059	2.76	
Z-N	PID	0.1412	1.84	0.46
SA	PID	0.16	8	0.5

The ITAE method is not applicable to this plant because the plant is not open-loop stable.

The PID parameters derived from different methods are given by Table 3.2.

3. Results

The left plot in Figure 3.3 showns the Ziegler-Nichols PID control performance. It stablizes the system, but the overshoot is more than 40%.

A simulated annealing (SA) optimization algorithm is applied to find the optimal PID settings, and the result is shown in the right plot of Figure 3.3. We can see that both the overshoot and rise time are improved using the optimal parameter settings.

3.3 Plant III

1. Plant Model

The test plant is a non-minimum-phase model. It has a right half plane zero at unity and three poles on the left half plane located at $-5, -1+0.9i$ and $-1-0.9i$.

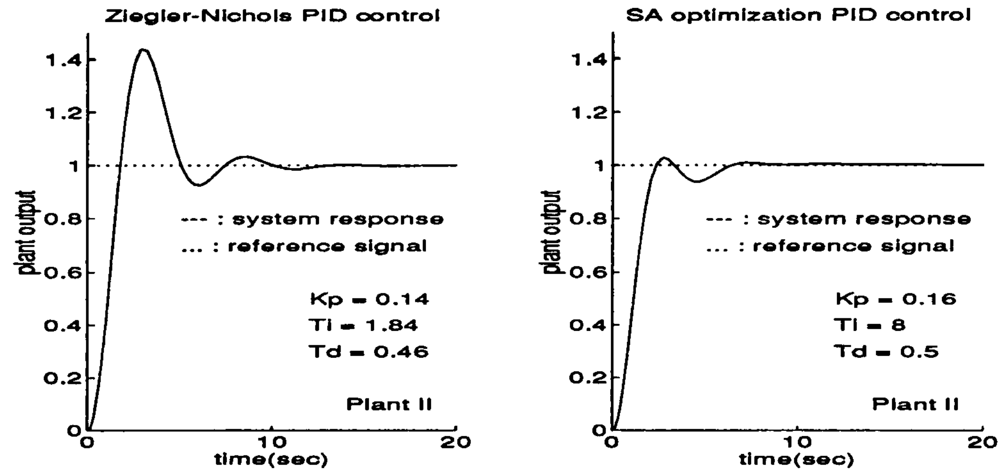


Figure 3.3: Plant II: Ziegler-Nichols and simulated annealing optimal PID control.

The transfer function of the model is:

$$T(s) = \frac{-2\,(s-1)}{(s+5)\,(s^2+2s+1.81)} \tag{3.3}$$

2. PID Parameters

The Ziegler-Nichols method yields parameters $a = 0.225$ and $L = 1.35$; the ITAE method yields parameters $K = 0.232, \theta = 1.35$ and $\tau = 0.97$; a simulated annealing algorithm is also applied to derive the optimal PID gains. The corresponding PID parameters are listed in Table 3.3.

3. Results

The left plot in Figure 3.4 shows the closed-loop step response using a Ziegler-Nichols PID controller. The derived PID parameters setting does not drive the system to the desired steady state; in fact, it makes the plant just marginally stable. The right plot is the closed-loop step response using an ITAE tuned PID controller. The plant reaches steady state; but it takes a long time.

Table 3.3: Plant III: PID design using Ziegler-Nichols time response, ITAE and simulated annealing optimization method.

Method	K	T_i	T_d
Z-N	5.33	2.7	0.675
ITAE	3.14	1.64	0.41
SA	1.50	0.94	0.24

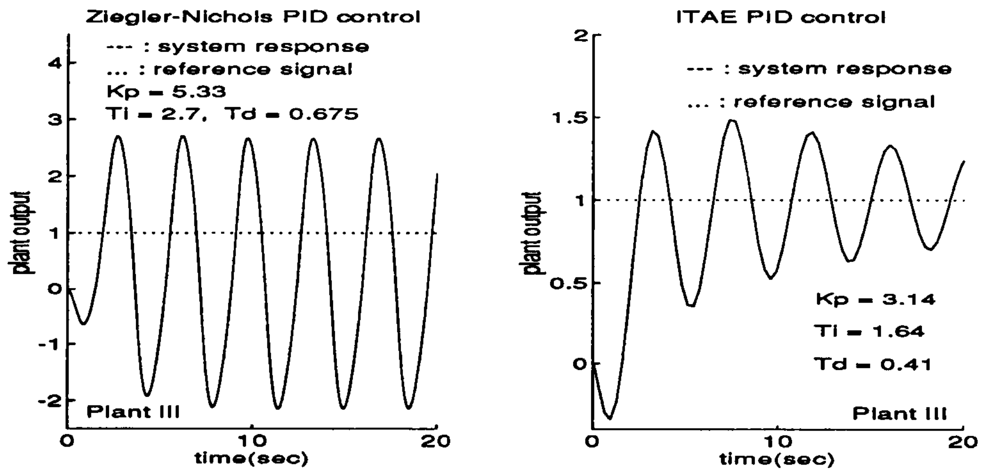


Figure 3.4: Plant III: Ziegler-Nichols and ITAE PID control.

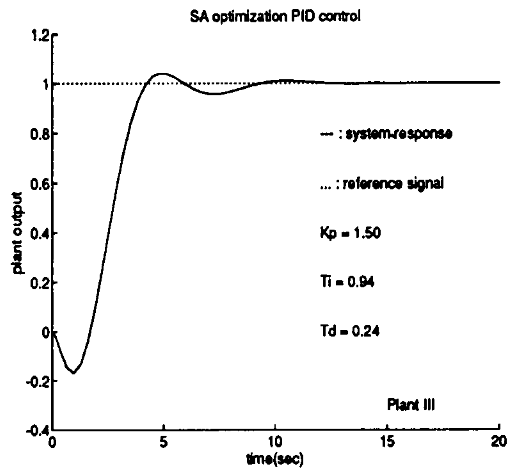


Figure 3.5: Plant III: Simulated annealing optimal PID control.

Figure 3.5 shows the response using the simulated annealing optimal PID controller. The plant output goes to steady state very fast with small overshoot and undershoot.

3.4 Plant IV

1. Plant Model

The test plant is a second order nonlinear model with the state space representation:

$$\begin{cases} \dot{x}_1 & = & x_2 \\ \dot{x}_2 & = & -\sin(x_1) + u * \cos(x_1) \\ y & = & x_1 \\ x_1(0) & = & 0 \\ x_2(0) & = & 0 \end{cases}$$

2. PID Parameters

Figure 3.6 shows the plant open-loop step response which is oscillating. Because of

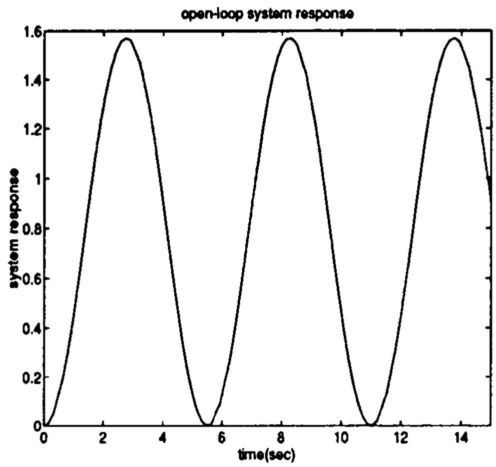


Figure 3.6: Plant IV: System open-loop step response.

the nonlinearity of the system, it can not be approximated as lower order system. Thus the derived Ziegler-Nichols and ITAE PID setting may not provide good control. A simulated annealing method is used to find the optimal PID controller.

For the Ziegler-Nichols time response method, $a = 0.43$ and $L = 0.5$; for the ITAE design, $\theta = 0.5$, $\tau = 1.07$ and $K = 1.58$. The corresponding PID settings are listed in Table 3.4.

3. Results

Neither the Ziegler-Nichols or ITAE PID design results in a closed-loop stable system. Neither method is applicable to this plant.

However, the optimal setting generated by the simulated annealing approach does give reasonably good control. In Figure 3.7, the left plot shows small overshoot and undershoot with a long rise time; the right plot shows a quick response, but with a comparatively big overshoot (25%) and some oscillations. This is an example where there is no unique optimal PID controller. If the requirements for overshoot and rise

Table 3.4: Plant IV: PID parameter settings using Ziegler-Nichols time response method, ITAE methods and Simulated Annealing optimization algorithm.

Method	K	T_i	T_d
Z-N	2.79	1	0.25
ITAE	1.17	1.47	0.16
SA(1)	0.34	0.92	9.71
SA(2)	0.99	0.26	8.21

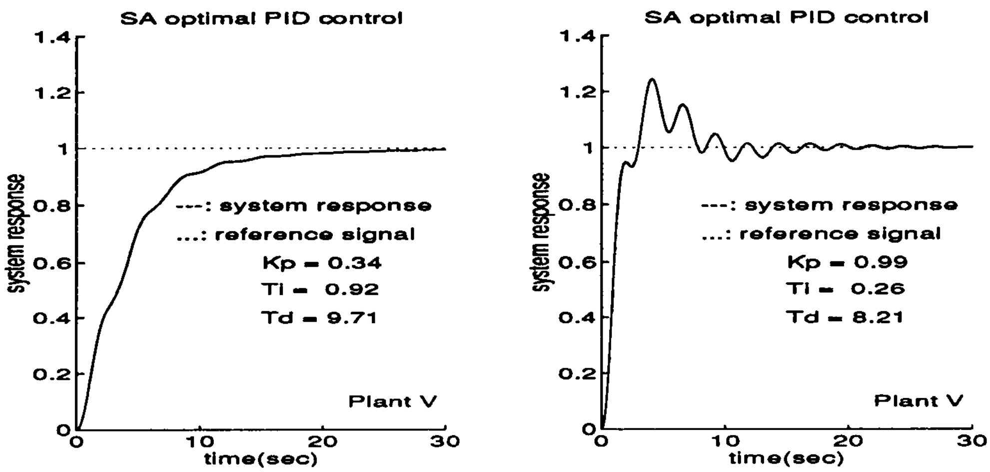


Figure 3.7: Plant IV: Simulated annealing optimal PID control.

Table 3.5: Plant V: PID parameter method derived from simulated annealing optimization algorithm.

Method	K	T_i	T_d
SA(1)	4.27	2.94	1.57
SA(2)	1.85	1.02	4.22

time are not simultaneously stringent, a PID controller can do a good job. If both requirements are very tight, a better control algorithm is needed.

3.5 Plant V

1. Plant Model

The test plant has a second order nonlinear structure with the state space representation:

$$\begin{cases} \dot{x}_1 &= x_1 * x_1 + u \\ \dot{x}_2 &= 0.2 * x_1 - x_2 + u \\ y &= x_1 - x_2 \\ x_1(0) &= 0 \\ x_2(0) &= 0 \end{cases}$$

2. PID Parameters

The system is not open-loop stable and the PID settings from the Ziegler-Nichols time response and ITAE method can not be derived. The simulated annealing algorithm is applied to find the optimal PID controller. The derived PID gains are listed in Table 3.5.

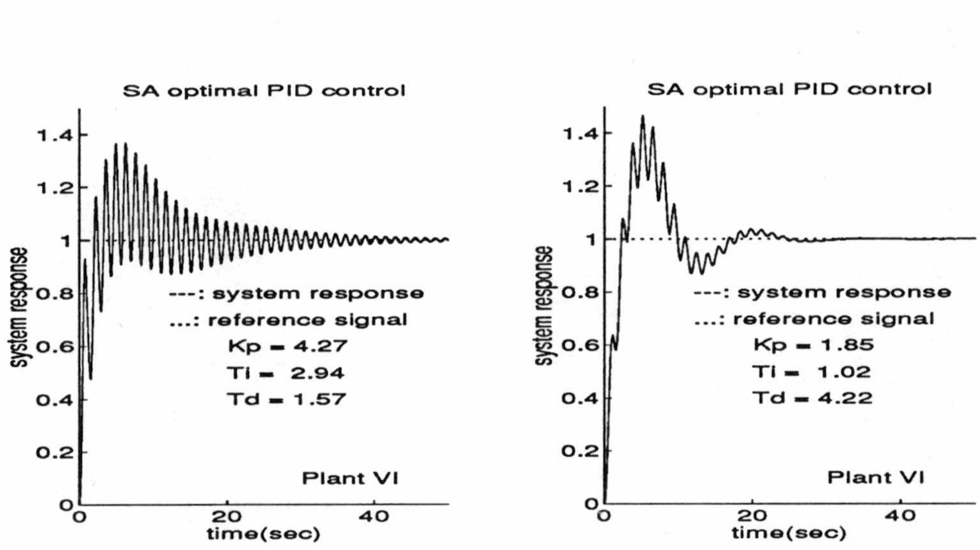


Figure 3.8: Plant V: Simulated annealing optimal PID control.

3. Results

Figure 3.8 shows the optimal PID control derived from simulated annealing optimization. Because the plant is nonlinear, it is very difficult to find a good PID control. During tuning, it is very easy to drive the plant to unstable states. Even the "optimal" result is not very good. The left plot shows severe oscillation; the right one has less oscillation, but it still suffers from a big overshoot. For this plant, a more sophisticated controller structure may solve this problem and achieve better control.

3.6 Discussion

The requirements placed upon a control system may include many factors. It may seem surprising that the control provided by such a simple PID structure is sufficient for many control problems, particularly where there are benign process dynamics and modest performance requirements.

The Ziegler-Nichols time response and ITAE methods are easy to apply because they are both based upon the system's open-loop step response. From the simulations we can see that the derived PID designs work well for many systems. Plants can be stablized with reasonable values for the percentage overshoot, rise time and settling time. However, neither method can be applied to all plants. For non-minimum phase plant III and nonlinear plants IV and V, neither method provides for a good controller. This is because both methods assume low-order linear models. Many plants can be approximated with this kind of structure, but for more complicated or nonlinear plants, more complex tuning methods are required.

One way to find the optimal PID setting is through the simulated annealing approach. The derived "near optimal" PID setting generates very good results for almost all the linear test plants. Even for many nonlinear plants good PID settings can be found which provide reasonably good results. This shows the control capabilities of the PID structure.

However, for some nonlinear plants, especially those with very complicated structures, the PID structure is too simple to handle very stringent control requirements. In cases where significant nonlinearity exists, the system dynamics are not well understood, the operating environment is variable, or very stringent goals are required, a more effective control algorithm is necessary.

CHAPTER 4

Fuzzy Control

During the past several years fuzzy control has emerged as one of the most active areas for research in the applications of fuzzy set theory, especially in the realm of industrial processes, which do not lend themselves to control by conventional methods because of a lack of quantitative data regarding the input-output relations. Fuzzy controllers, which are based upon approximate reasoning not requiring analytical models, have demonstrated a number of successful applications, for example in the subway system in the city of Sendai in Japan [30] and in nuclear reactor control [3]. These applications have mainly concentrated on emulating the performance of an expert in the form of linguistic rules [4]. The IF-THEN linguistic control rule structure provides a strategy to represent the experts' knowledge which is used as a reference during the control process. For many systems, because these rules are easy to understand; and fuzzy controllers are relatively easy to design, fuzzy control has become more and more popular. In this chapter, the basic fuzzy set concepts and the fuzzy system structure are briefly reviewed.

4.1 Basic Concept

The kernel of a fuzzy controller is the IF-THEN rulebase. Fuzzy sets are used to represent the linguistic terms in the rules. Fuzzy operations are applied to infer control actions given the rulebase and measurement data.

4.1.1 Fuzzy Sets

A fuzzy set, originally defined by Zadeh [32], is an extension of a conventional or crisp set.

The crisp set dichotomizes the individuals in some given universe of discourse into two groups, members and nonmembers, by its characteristic function:

$$\mu_A : X \to \{0,1\}, \quad \mu_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

Eliminating the sharp boundary between members and nonmembers, the fuzzy set allows modeling of vagueness and constitutes an important framework to represent uncertainty and complexity. As crisp sets can be represented by their characteristic functions, fuzzy sets are defined by membership functions which define a mapping from set X to a range space P, which is typically the unit interval $[0,1]$:

$$\mu : X \to P([0,1])$$

The value of this function represents the degree of membership of each individual.

In many processes, such as information analysis, decision making and prediction, instead of dealing with precise data, one encounters uncertainty. With the help of fuzzy set representations, rough, noisy numerical information can be translated into meaningful linguistic terms that can be handled by experience. For example, if a car runs on the highway alone, there is no significant difference between 50 and 55 mph, since the meaningful information is that the car is running at normal speed. The fuzzy term "normal speed" needs to be defined, and every speed has a degree of membership in this set.

Table 4.1: The axiomatic properties of fuzzy operations, $a, b, d \in [0,1]$.

Fuzzy Operation	Complement	Union	Intersection
Representation	$c : [0,1] \to [0,1]$	$u : [0,1] \times [0,1] \to [0,1]$	$i : [0,1] \times [0,1] \to [0,1]$
Boundary conditions	$c(0) = 1$ $c(1) = 0$	$u(0,0) = 0, u(1,1) = 1$ $u(0,1) = u(1,0) = 1$	$i(0,0) = 0, i(1,1) = 1$ $i(0,1) = i(1,0) = 0$
Monotonicity	if $a < b$ then $c(a) \geq c(b)$	if $a \leq a'$ and $b \leq b'$ then $u(a,b) \leq u(a',b')$	if $a \leq a'$ and $b \leq b'$ then $i(a,b) \leq i(a',b')$
Communitivity	—	$u(a,b) = u(b,a)$	$i(a,b) = i(b,a)$
Associativity	—	$u(u(a,b),d) = u(a,u(b,d))$	$i(i(a,b),d) = i(a,i(b,d))$

4.1.2 Fuzzy Operation

Complement, union and intersection are fundamental operations on sets. Fuzzy set operations are generalized from these operations to operate on the unit interval with several axiomatic properties: boundary conditions, monotonicity, commutativity and associativity, as illustrated in Table 4.1. All functions that possess these desirable features are allowable fuzzy operations. Different operators have different strengths and generate different results.

Boundary conditions and monotonic nonincreasing characteristics are the two axiomatic requirements of fuzzy complement, which is an unary operator. For fuzzy union and intersection, which are binary operators, boundary condition, commutativity, monotonicity and associativity are the axiomatic skeleton [15]. As long as these requirements are satisfied, the generalizations are suitable fuzzy operations.

Amongst the many possible fuzzy operations, the simplest ones are used in this design and illustrated in Figure 4.1:

$$Fuzzy\ \ complement: \ \ \mu_A(x) = 1 - \mu_A(x). \tag{4.1}$$

$$Fuzzy\ \ union: \ \ \mu_{A \cup B(x)} = max[\mu_A(x), \mu_B(x)]. \tag{4.2}$$

$$Fuzzy\ \ intersection: \ \ \mu_{A \cap B(x)} = min[\mu_A(x), \mu_B(x)]. \tag{4.3}$$

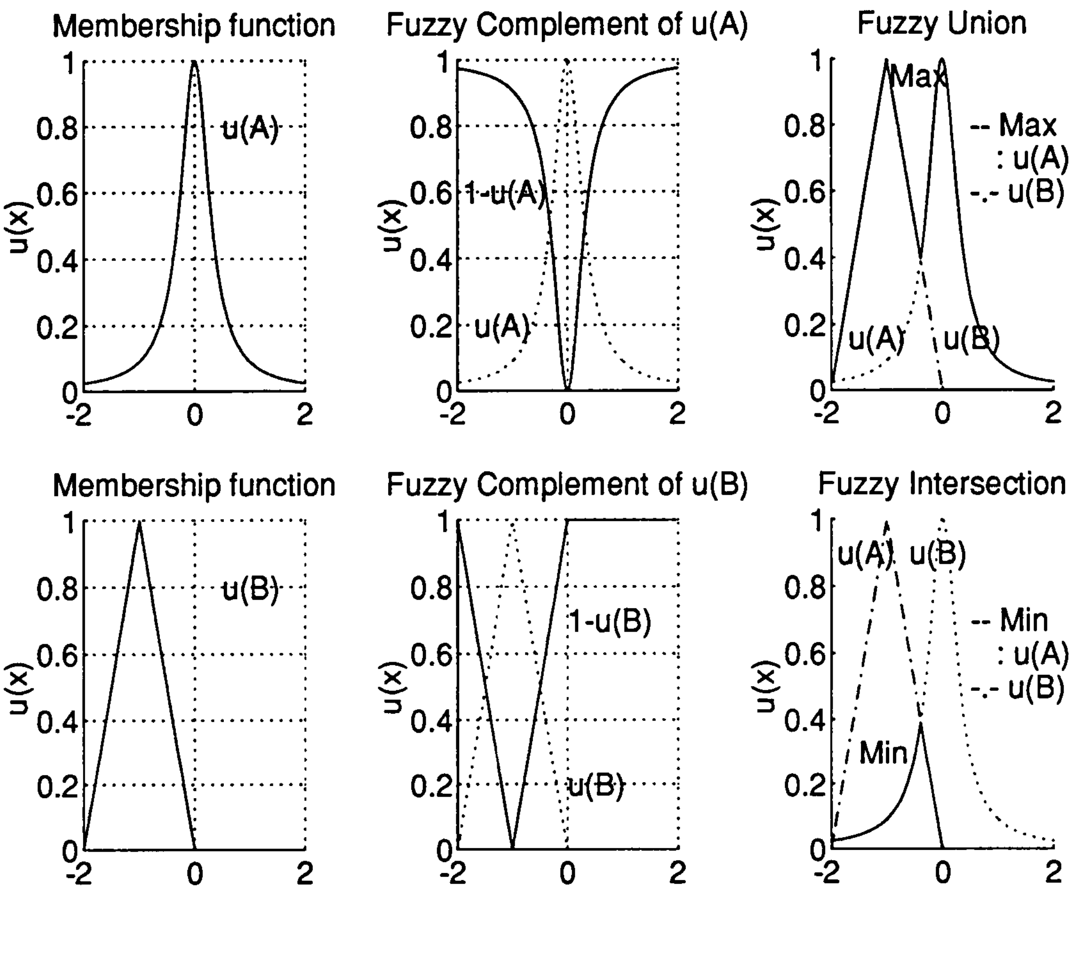


Figure 4.1: Fuzzy membership function and fuzzy operation.

4.2 Fuzzy Controller Design

4.2.1 Fuzzy System

Fuzzy control was initially conceptualized and implemented by Mamdani [21] in a successful attempt to control a boiler steam engine. The fuzzy system implements fuzzy set and fuzzy logic theory in controller design. This idea was later applied in many other system designs. Under circumstances when complete knowledge of the underlying dynamics is unavailable, acting as a skilled human operator, fuzzy systems can provide efficient controls for many ill-defined complex processes [19]. In a fuzzy system, a fuzzy block replaces the conventional controller. It takes several inputs $I_{1,2,\ldots,p}$ and generates control signals $O_{1,2,\ldots,q}$ to the plant. The control algorithm is achieved by the fuzzy inference mechanism which provides proper mapping from the input domain to the output domain. A simple architecture of a fuzzy logic controller is shown in Figure 4.2.

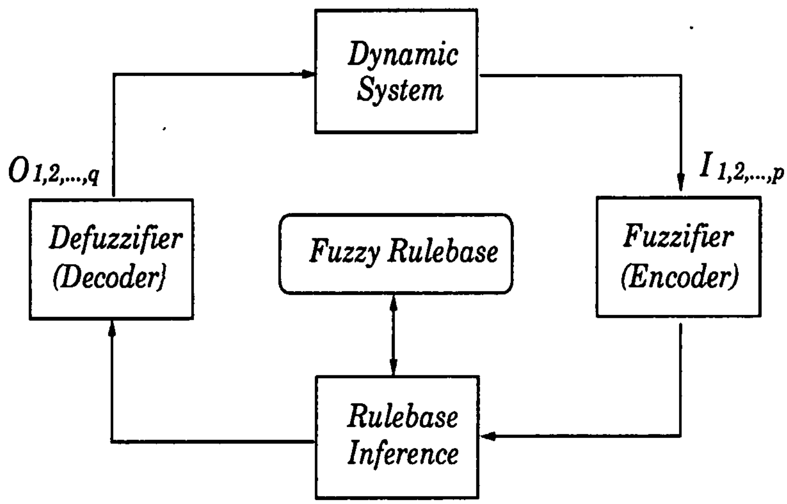


Figure 4.2: A simple architecture of a fuzzy controller.

4.2.2 Fuzzy Rulebase

The amount of information contained in a fuzzy rulebase and the mapping from the input to the output domain that it provides determine the effectiveness of the controller.

- Each input/output variable takes values according to fuzzy sets, defined for the variable such as *very small, small, medium, big* and *very big*, are their associated membership functions.

- The control strategy is encoded as IF-THEN rules that relate the inputs and outputs of the fuzzy system. Every fuzzy rule has *antecedent* and *consequent* parts prescribing the value of output actions. For example:

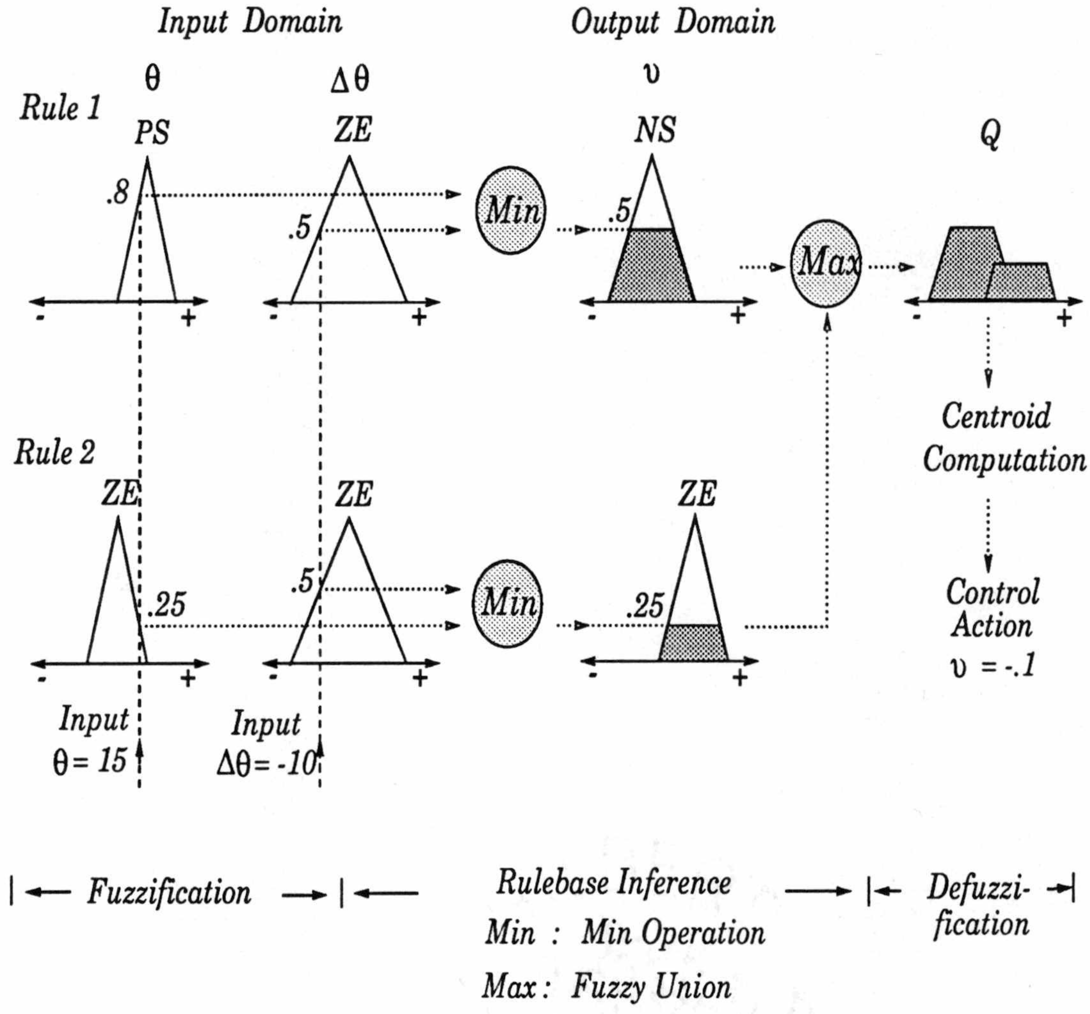
  IF I_1 *is very small AND* I_2 *is very big*, THEN O_1 *is big,* O_2 *is small.*

4.2.3 Fuzzy Inference

The fuzzy inference engine generates control actions according to the input information and the rulebase. The overall procedure includes the fuzzification, rulebase inference and defuzzification processes. Here, one of the simplest and most commonly used methods is implemented as shown in Figure 4.3:

1. Fuzzification process

   For each input variable, the fuzzification process converts the numerical data into a membership function on the variable's domain. In the example shown in Figure 4.3 [17], there are two inputs $\theta = 15$ and $\Delta\theta = -10$ which enter the fuzzy block.



Rule1: If $\theta = PS$ *and* $\Delta\theta = ZE$, *then* $\upsilon = NS$.
Rule2: If $\theta = ZE$ *and* $\Delta\theta = ZE$, *then* $\upsilon = ZE$.

Figure 4.3: Fuzzy inference.

For the input value θ, a membership degree $\mu_{\theta_{PS}}$ of 0.8 is assigned to fuzzy set θ_{PS}; $\mu_{\theta_{PS}} = 0.8$;

A membership degree $\mu_{\theta_{ZE}}$ of 0.25 is assigned to fuzzy set θ_{ZE}; $\mu_{\theta_{ZE}} = 0.25$.

For the input value $\Delta\theta$, a membership degree $\mu_{\Delta\theta_{ZE}}$ of 0.5 is assigned to fuzzy set $\Delta\theta_{ZE}$; $\mu_{\Delta\theta_{ZE}} = 0.5$.

Thus numerical data are converted into linguistic terms for further processing.

2. Rulebase inference

A rule is used in the output variable's calculation (fired) if the membership degrees of all fuzzy sets appearing in the antecedent are positive. (This assumes that the antecedent is a conjunction of terms of the form {crisp value = fuzzy membership function}.)

The overlapping preconditions of rules and their partially matching features cause several rules to fire at once. The combination of all the fired rules produces the fuzzy control output which is then defuzzified to provide the crisp control action.

Suppose a fuzzy rule structure is:

IF I_A *is* I_{A_i} *and* I_B *is* I_{B_j}, THEN O *is* O_{ij} $\quad (i = 1, 2, ..., N_A; \quad j = 1, 2, ..., N_B)$.

Where

I_A and I_B	are the input variables;
I_{A_i} and I_{B_j}	are the corresponding *ith* and *jth* fuzzy sets with which the input variables are computed;
N_A and N_B	are the numbers of fuzzy sets for variables I_A and I_B, respectively;

O is the output variable; and

O_{ij} is a fuzzy set of O which corresponds to the (i,j) rule.

After fuzzification, we have the degree of membership functions of each input variable: $\mu_{I_{A_i}}$ and $\mu_{I_{B_j}}$ $(i = 1,2,...,N_A, j = 1,2,...,N_B)$.

- First, we do a *Min* operation of all the variables in the IF part of each rule: $Min(\mu_{I_{A_i}}, \mu_{I_{B_j}})$. The degree of membership function shows the fit value, and the Min operation yields a minimum fit value.

  As shown in Figure 4.3, for rule 1, $Min(\mu_{\theta_{PS}}, \mu_{\Delta\theta_{ZE}}) = Min(0.8, 0.5) = 0.5$; for rule 2, $Min(\mu_{\theta_{ZE}}, \mu_{\Delta\theta_{ZE}}) = Min(0.25, 0.5) = 0.25$.

- Second, a *Fuzzy Union* operation is taken. If we apply the minimum fit value from the first step to each corresponding output fuzzy set, by computing the pointwise minimum over the output membership function's domain, and combine all of the results using a pointwise maximum, or do a fuzzy union operation, we obtain a fuzzy set that represents the desired control action. In Figure 4.3, fuzzy set Q represents the control action. It is generated by inference using rule 1 and rule 2.

3. Defuzzification process

The defuzzification process calculates the final control action from its fuzzy set representation. A crisp value is derived from the fuzzy linguistic description. The centroid method is one of the most widely used algorithms for defuzzification.

Suppose a fuzzy set can be described by a set of discrete numbers $\vec{X} = \{X_1, X_2, ..., X_N\}$, with degree of membership function $\vec{\mu} = \{\mu_1, \mu_2, ..., \mu_N\}$.

where $\vec{X}$ is the vector representing the N possible values for the input variable and $\vec{\mu}$ is the corresponding vector representing the degree of membership function.

The centroid of this fuzzy set is :

$$Centroid_X = \frac{< \vec{\mu}, \vec{X} >}{\|\mu\|_1} \tag{4.4}$$

The resulting crisp value is the action generated by fuzzy rulebase inferencing.

If the membership function were defined on a continuum, the inner product and norm would be evaluated using integrals.

4.2.4 Fuzzy Controller Model

Figure 4.4 shows the most commonly used fuzzy controller model. The variable u is the control input signal to the plant. Its variation du is generated by a fuzzy rulebase with two inputs: the difference signal e between reference r and system response y, and its variation de. Usually, this structure is implemented in digital controller design, where

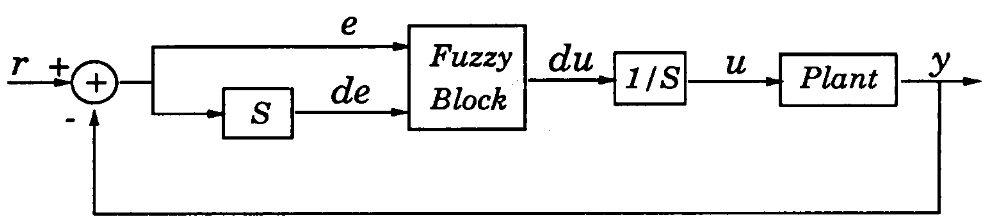


Figure 4.4: Fuzzy controller model.

several modifications has to be made accordingly. In discrete cases, measurements are obtained at each sampling time. Signal de will be generated by a "difference" block (instead of derivative block) which calculates the change of the error signal compared with the signal at the previous sampling time. Signal du will go through an accumulator (instead of an integrator) which provides control input to the plant.

4.2.5 Fuzzy Rulebase Learning

A fuzzy rulebase can store experts' knowledge about the controller's desired behavior and enable the fuzzy controller to take the correct actions under different circumstances; however, when such knowledge is not fully accessible, a learning process is necessary to acquire useful information and construct the rulebase. Figure 4.5 shows the common trial and error learning process. Before each trial a rulebase tuning action is assigned and a new rulebase is constructed according to the changes. After simulation, the new system performance is judged by an *evaluation block* which determines acceptance or rejection of the new rulebase.

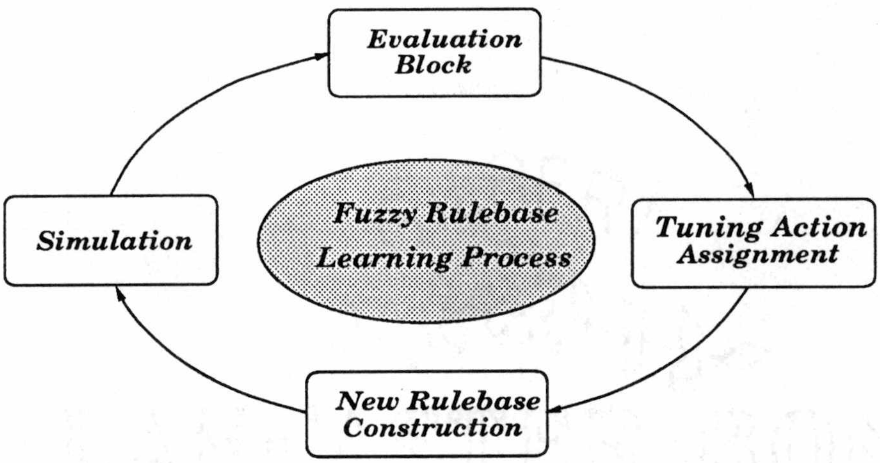


Figure 4.5: Fuzzy rulebase learning.

4.3 Discussion

In this chapter, basic fuzzy set theory and fuzzy operations are reviewed, along with an illustration of a commonly used fuzzy controller structure and the basic concept of fuzzy control. Fuzzy control encodes human knowledge that can deal with uncertainty and complexity in a rulebase in the form of IF-THEN structured rules. A fuzzy set, which is an extension of crisp set, provides a numerical representation for the linguistic terms that are used in fuzzy rules. Fuzzy operations, which are the generalization of crisp set operations, provide an mechanism to inference fuzzy rulebase and generate control actions to the plants.

CHAPTER 5

Fuzzy-PID Control

An approach which merges PID and and fuzzy control is proposed in this chapter to take advantage of both the well developed features of PID control and the potential of fuzzy knowledge representation. The parameters of a PID controller are obtained by fuzzy inference. The controller behaves as a trouble shooter and adapts its behavior to different operating conditions.

Usually the control system design must satisfy a detailed set of performance specifications. These specifications may impose a number of design restrictions, which are often mutually exclusive or contradictory. To assure satisfactory response, the only design technique under such circumstances is the trial-and-error synthesis method, which is really an art rather than a science [1]. In order to resolve the conflicts between various specifications, an optimal synthesis method is applied and explained in Section 5.4.

Expert's knowledge is encoded in a fuzzy rulebase for PID control. When this information is not available a successive refinement trial and error learning process can be used to construct the proper rulebase. As described in Section 5.6, stochastic approaches and probability theories are implemented to speed up the learning process.

Several Fuzzy-PID controllers are developed for the plants used for PID control in Chapter 2 as case studies. Some simulation tests, such as stability, robustness, disturbance tolerance and different command input tests are made on the developed systems to demonstrate the system performance.

5.1 Control System Structure

The structure of the Fuzzy-PID controller is shown in Figure 5.1. It has the basic PID structure, but the PID parameter settings are generated by fuzzy inference, which provides a nonlinear mapping from the error signal e (difference between system output and reference signal) to the PID gains K_p, K_i and K_d. The nonlinear function is tuned from an initial function that approximates these constant gains.

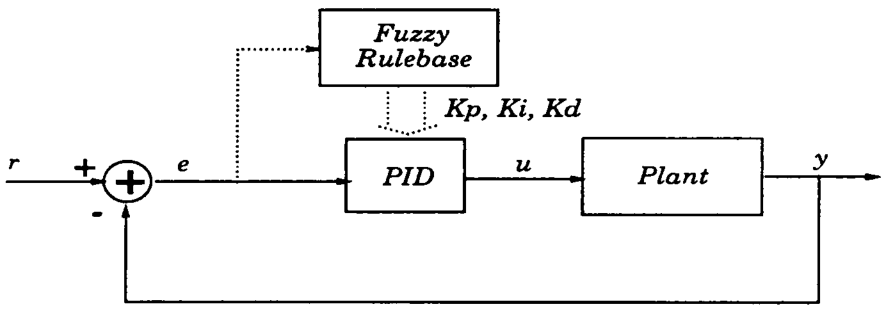


Figure 5.1: Self-tuning Fuzzy-PID controller model.

5.2 Controller Design Procedure

The design procedure flow chart is shown in Figure 5.2. The fuzzy rulebase is the kernel of the controller where the control strategy is stored. It is initialized to provide an approximate fixed PID mapping and updated through an iterative learning process until an optimal setting is obtained. The complete process includes rulebase initialization and rulebase tuning. Each tuning cycle has several steps: rulebase perturbation, new rulebase construction, simulation, performance evaluation, new rulebase acceptance or rejection, rulebase update and end criteria check.

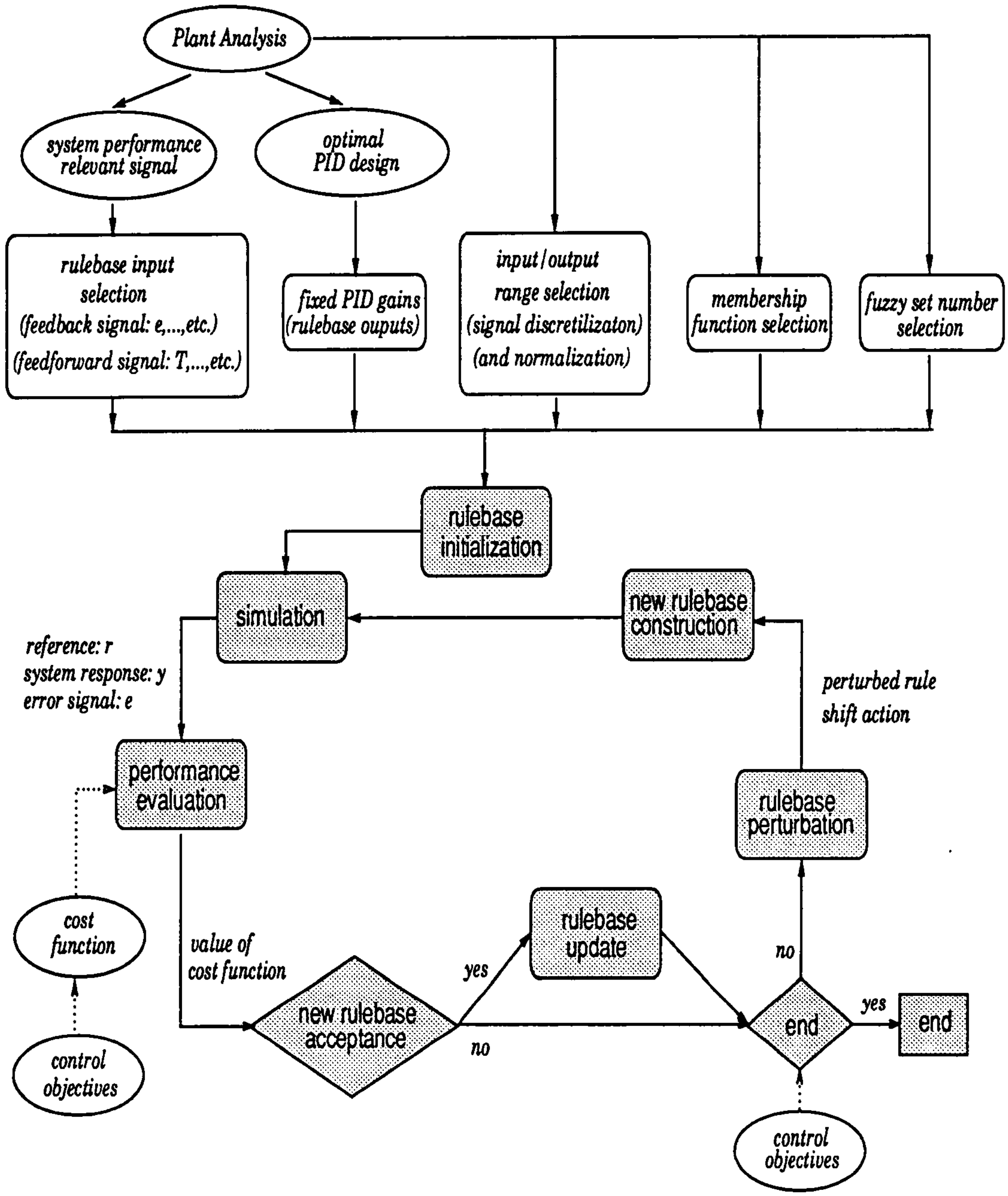


Figure 5.2: Self-tuning Fuzzy-PID controller design flow chart.

5.3 Fuzzy Rulebase Structure

The fuzzy rulebase acts as a parameter tuner for the PID control. As illustrated in Figure 5.3, it takes the error signal e between the reference and system output as its input and generates three outputs: K_p, K_i and K_d for the PID controller based on the fuzzy rules.

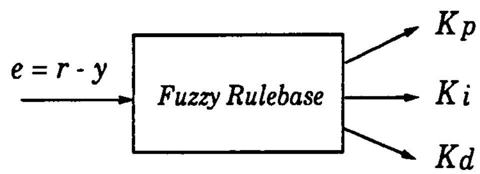


Figure 5.3: Fuzzy rulebase.

5.3.1 Fuzzy Set Number

N fuzzy sets, or membership functions, are used to represent each of the input and output variables. Usually, a larger N results in a finer partition of the variable space and provides a more detailed mapping that represents the input-output relationships and can achieve more accurate tuning. However, a larger N also results in exponentially increasing memory storage for fuzzy rules and heavier computation load for the fuzzification and defuzzification process. A much longer time is required to achieve a proper rulebase during the rulebase constructing stage for large values of N. Although recently some new approaches have been developed to reduce the growth rate of the rulebase with increasing N [23], more research is required to solve this problem.

In the simulation, values of N of 3, 5 and 7 are tested. For plants with a simple structure, an N of 3 can provide an acceptable mapping; for some plants, the choice of N does not make a big difference on the system performance, especially for values of N of 5 or 7. For many nonlinear plants, with values of N of 3 and 5, it is difficult to find a mapping which is accurate enough to generate effective control. In order to reduce the memory storage, speed up the learning process and at the same time provide a sufficiently accurate rulebase for acceptable system performance, for the linear test plants, a value of N of five is selected for each variable; for nonlinear plants, a value of N of seven is applied.

5.3.2 Variable Range

The allowable range of each variable determines the domain of its membership functions. Appropriate variable ranges should be wide enough to include the *optimal* solution and narrow enough to reduce the range of the search space. This range should also keep the system stable during tuning. If the *optimal* parameter setting lies outside the preselected range, if the specified range is too wide, or if the unstable region is included as a possible area for parameter selection, a good control will be impossible or very difficult to achieve.

A self-adjusting mechanism is provided to assist in proper variable range selection. Before each tuning cycle, a detection and modification step can be taken to adjust the input and output variables' ranges.

5.3.3 Membership Function

Many shapes, such as Gaussian, triangle and echelon, can be used to represent fuzzy membership functions. In this design, the triangular shape is implemented. Three parameters (the left and right endpoints of the triangle and the top's position) define a fuzzy set.

Fuzzy membership functions are most commonly represented by a set of discrete numbers in digital implementations. Discretization also simplifies operations on the fuzzy sets. In the experiments, each variable's domain is divided into 32 contiguous intervals. Hence, each fuzzy set can be described by 32 numbers.

5.3.4 Fuzzy Rules

There are three sets of fuzzy rules dedicated to the three different relationships in the design: $e \rightarrow K_p,\ \ e \rightarrow K_i\ \ and\ \ e \rightarrow K_d$.

$$\text{If } e \text{ is } MI_k, \text{ then } K_p \text{ is } MKp_k. \quad (\textit{for } e \rightarrow K_p)$$

$$\text{If } e \text{ is } MI_k, \text{ then } K_i \text{ is } MKi_k. \quad (\textit{for } e \rightarrow K_i)$$

$$\text{If } e \text{ is } MI_k, \text{ then } K_d \text{ is } MKd_k. \quad (\textit{for } e \rightarrow K_d)$$

where $k = 1, ..., N$.

N is the number of fuzzy sets of each variable;

MI_k is the k^{th} fuzzy set of the input e;

MKp_k is the fuzzy set of the output K_p specified by the k^{th} rule;

MKi_k is the fuzzy set of the output K_i specified by the k^{th} rule; and

MKd_k is the fuzzy set of the output K_d specified by the k^{th} rule;

Instead of using linguistic terms, such as "small", "medium" and "large" to describe fuzzy sets, the index k is used to label them.

The N input fuzzy sets cover the entire input domain to provide control actions for all possible situations. In order to simplify tuning, they retain their original shapes during adaptations.

Output fuzzy sets provide the mapping for the control law. These parameter settings are updated and improved through a successive refinement approach. Thus, the free parameters which are modified during adaptation are the sets of parameters defining the output membership functions.

5.4 System Performance Evaluation

A performance index is a quantitative measure of system behavior and is chosen to emphasize important system specifications. Such a quantitative measure is desirable to avoid the difficulties of meeting several specifications simultaneously. Also, it is necessary for automation of the parameter optimization process in order to design optimal control systems [1].

An optimal system maximizes the performance index. The optimal system's behavior under the derived optimal control is determined by the definition of the performance index.

During the learning process the performance evaluation block acts as a judge of the system behavior and delivers messages to guide the next perturbation of the rulebase. Constructing a sensitive performance index is very important for effective tuning.

Many criteria can be used to judge system performance. Even though the transient response to a unit step-function is a very limited representation of system behavior, the control requirements are often specified in terms of it, especially for linear time invariant systems, and in practice, most of the reference signals are constant set points except for occasional changes. For nonlinear plants, more specifications are often imposed because of their complex structures and difficult to predict behavior.

In the experiments, system performance is indicated by the negative of a cost function calculated from the step response with a zero initial state; the task of the tuning process is to minimize the cost function.

Considering all the factors, the cost function can be formulated as a weighted summation [33]:

$$Cost = \sum_{i=1}^{N} Weight_i * Factor_i \tag{5.1}$$

Basically, the cost function consists of two components. The first one is the weighted sum of system performance measures: ITAE, overshoot and rise time. The second part indicates the satisfactory status of the current performance with respect to the control objectives.

In the simulation, the cost function is composed of:

- *a.1 ITAE:* C_{ITAE}

  C_{ITAE} is the integral of the time-weighted absolute error: $\int t|error|dt$. A small value of C_{ITAE} implies a small steady state error. However, it ignores the contribution of initial, or transient behaviors. Thus overshoot and rise time are added to strengthen the contribution of the transient response characteristic.

- *a.2 Overshoot:* $C_{overshoot}$

  The percentage of maximum overcorrection with respect to the input step signal caused by the controller.

- *a.3 Rise time:* $C_{risetime}$

  The time required to achieve 90% of the final value of the system response.

- *b. Objective satisfaction status:* $C_{satisfaction}$

Five criteria are specified as the control objectives: overshoot, undershoot, rise time, settling time and steady state error. The difference between the current performance and control objective is calculated. A satisfied specification contributes nothing to the total cost, while unsatisfied one adds a large penalty.

Mathematically, the cost function used in the simulation tests is defined by:

$$Cost = C_{ITAE} + C_{overshoot} * 100 + C_{risetime} * 100 + C_{satisfaction} \tag{5.2}$$

$$C_{ITAE} = \sum_{t=0}^{end} t * |error_t| \quad (where\ \Delta t = 0.05s) \tag{5.3}$$

$$C_{satisfaction} = (\sum_{i=1}^{5} Satisfaction_i) * Penalty \tag{5.4}$$

$$Penalty = 1000 * \text{Total simulation time} \tag{5.5}$$

$$Satisfaction_i = \begin{cases} 0 & \text{if the } i^{th} \text{ requirement is satisfied} \\ \dfrac{\text{performance index } i}{\text{control objective } i} & \text{otherwise} \end{cases} \tag{5.6}$$

Where performance indices (control objectives) are the overshoot, undershoot, rise time, settling time and steady state error. The control objectives for the test plants are given by Table 5.1.

5.5 Fuzzy Rulebase Initialization

The initial fuzzy rulebase significantly impacts performance tuning. A good initialization can speed up the tuning process and increase the quality of control. Some conventional methods, such as Ziegler-Nichols, ITAE or the simulated annealing approaches have been applied to construct the initial rulebase.

However, under circumstances where no *a priori* information is available, a heuristic guess can be used as the best substitute. Although the initial parameter setting makes

Table 5.1: Control objective specifications for the test plants I–V

Plant	overshoot	undershoot	rise time	steady state error	settling time
I	1%	1%	21s	0.5%	50s
II	2%	2%	2s	0.2%	10s
III	1%	1%	4.8s	0.5%	6s
IV	2%	2%	3s	0.2%	20s
V	2%	2%	3s	0.2%	20s

a big difference on the tuning speed and final result, any reasonable initialization is acceptable.

In the simulation, the initial setting approximates a conventional PID controller. The input-output relationships generate fixed PID parameter settings for any input signal. All the output fuzzy sets are defined as symmetrical triangles centered at the corresponding PID parameters. For example, if $K_p = c$, all the top values of the membership functions $MKp_i, (i = 1, 2, ..., N)$ are assigned to be c; this generates a constant output $K_p = c$ for any input e.

5.6 Successive Refinement Tuning

In order to achieve optimal mapping from fuzzy inputs to outputs, the rulebase is updated and refined through a successive refinement tuning process where stochastic approaches and heuristic guess have been implemented. This process contains several steps: rulebase perturbation, system performance evaluation, acceptance or rejection of the perturbation and iteration.

5.6.1 Punishment and Reward

Credit assignment, given the performance (results) of a process, distributes reward or blame to the individual elements contributing to that performance. In rulebase systems, this means the assignment of credit or blame to individual rules (or the parameters that are used in each rule) engaged in the problem solving process.

A punishment and reward credit assignment strategy is applied in the trial and error tuning process. After rulebase perturbation and system performance evaluation, a decision should be made either to accept the modified fuzzy rulebase or to retain the original one. If the cost function decreases, the modified rulebase should be accepted; otherwise, the original setting remains. The current performance also affects the historical credit record which is used to select the next rulebase disturbance during tuning.

Some ideas borrowed from the simulated annealing method [18] are applied to search over more areas in the parameter domain during the early stage of the process. Rulebase modifications which worsen performance are accepted using a random decision rule. The probability of accepting a poor modification is defined as an exponentially decreasing function of the iteration number. During the early trials, modifications which worsen performance are more likely to be accepted, increasing the chances to search over a larger area. This probability of acceptance decreases until convergence to an optimal solution is reached.

5.6.2 Successive Refinement

The optimization approach used here is called successive refinement because of the rulebase perturbation and control objective update strategies.

A. Rulebase perturbation strategy

In each trial cycle, a small perturbation is applied to the fuzzy rulebase. If the performance is improved, in the next trial, the same perturbation is applied. This approach will be repeated for up to a pre-selected maximum number ($T_{max\_perturb}$) of trials, or until performance deteriorates. This is similar to the line search strategy frequently utilized in numerical optimization.

Small perturbations are expected to keep the system stable during tuning. However, convergence usually takes a long time if there is no connection between successive trials. Sometimes several trials have to be made before a favorable perturbation can be found because of the number of degrees of freedom in the rulebase. Performing like a human operator, the tuning process views a good perturbation as the correct direction of adjustment and applies it once more; it views a bad perturbation as the wrong direction and applies the opposite perturbation in the next cycle.

The process is a stochastic approach with a significant degree of randomness. In order to reduce the effect of "wrong intuition", to limit the amount of "human involvement" and to prevent the unbalanced tuning of the fuzzy rulebase (a particular rule shifts too much at one time), a maximum limit for the number of identical favorable successive perturbations $T_{max\_perturb} = 5$ is introduced. For favorable perturbations, although this will end the current favorable modification, the probability of applying this kind of adjustment again in future trials is still high because of the rule selection mechanism.

B. Control objective update strategy

The final control objective is specified in terms of overshoot, undershoot, rise time, settling time and steady state error. A different current control objective is used at each iteration which starts from a lenient objective and is gradually updated to a more stringent level upon satisfaction until the final goal is met. Before each rulebase perturbation, a test is made to check if all the current control objectives are satisfied. If all of them are satisfied, all the performance index requirements are updated to a more stringent level; otherwise, they remain at the original values. It is difficult to satisfy all the requirements at once because they often contradict with each other. Successive updates divide the difficulty into several stages. They also avoid unbalanced rulebase tuning which satisfies some requirements and ignores the rests. When a special objective is very difficult to reach, an option which updates all the other requirements is provided to continue the refinement tuning, but this can result in some unsatisfied specifications. This strategy is similar to penalty function, objective relaxation, and homotopy methods in numerical optimization.

5.6.3 Rulebase Perturbation

A smart perturbation is the expert's shortcut to reach the final solution. It is essential to make valuable trials that provide new information in order to save computation time and human effort.

Rulebase parameter variation in the fuzzy rulebase is achieved in two steps: First, select a particular rule from the $3N$ candidates; Next, decide how much perturbation should be applied to the selected rule.

A. Rule Selection

An intelligent 1 out of $3N$ choice needs to be made each time to get the most information out of every trial. Subsection $A.1$ describes the theoretical guidance in rule selection; Subsection $A.2$ describes the implementation.

A.1 Rule Selection Guidance

- *System cultivation*

  System cultivation [6] methods are used to analyse measurement data, extract process knowledge and guide rule selection. The error signal is recorded for each tuning cycle. Its distribution histogram can be calculated to provide information about approximately how often the system is running under different conditions. If the input domain is divided into N sub-intervals, each of which corresponds to an input fuzzy set according to its location, the histogram distribution of signal e gives an estimation about how often the system is running under each condition.

  Poor control is caused by ineffective control laws. It implies a mismatch in the $e \rightarrow K_p, K_i, K_d$ relationships. Suppose most of the time a system is operating near the range $e_{often}(\neq 0)$ which corresponds to fuzzy set MI_k; if performance is poor, we know that the relationship provided by one or more of the rules which determine $MI_k \rightarrow MKp_k$, $MI_k \rightarrow MKi_k$ or $MI_k \rightarrow MKd_k$ is not effective. On the other hand, if the system seldom operates near the range e_{seldom}, rulebase tuning around this range will not have a large influence on system behavior. Therefore, the assignment of a suitable probability according to the operating conditions will reduce "waste trials" and speed up the learning process. This rule selection probability vector is obtained by computation using the histogram information.

- *Control objective guidance*

  Plant performance can also guide the tuning process. For example, if a large steady state error is obtained, the priority to tune the $e \rightarrow K_i$ relationship should be increased; if a large overshoot or undershoot has been detected, tuning the $e \rightarrow K_p$ relationship around the proper range is helpful in many cases. A weighting vector is formulated from experiments and used to modify the rule selection probability vector obtained from the system cultivation calculation.

A.2 Rule Selection Implementation

All the $3N$ rules are divided into N groups according to the range of e. The k^{th} group contains rules $MI_k \rightarrow MKp_k$, $MI_k \rightarrow MKi_k$, $MI_k \rightarrow MKd_k$, for each $k = 1, ..., N$.

The rule selection process is divided into two stages. First, one group is chosen from the N candidates. Second, a particular rule within the selected group is chosen.

- *First Stage*

  Suppose N fuzzy sets are used to represent each variable. If the input domain is divided into N consecutive intervals, from the system cultivation analysis and the history of plant performance modification, a value can be assigned to each interval representing the likelihood of the corresponding rules to be selected for tuning. The task of the first step in the rulebase perturbation is to select one from the total N groups of rules. This selection is made based on the derived $N-element$ probability vector. The bigger the value of an element of this vector, the more likely the corresponding group will be selected. This concept is realized by roulette wheel selection.

1. *Roulette wheel selection*

   Roulette wheel selection [10] is a random selection approach based on probability. Suppose that in making a choice from k options where each one has its particular probability, if we divide a wheel according to the options' probabilities, the position of a randomly tossed die can be used as the choice.

2. *Adjustment for very low probability rules*

   A small non-zero base probability is assigned to guarantee that all rule will be attempted with probability one over an infinite horizon of iterations.

3. *Adjustment for zero error signal rules*

   The ideal goal is to drive the error signal to zero throughout the plant's operating regime. If during most of the time, the error signal falls in a range around zero, (corresponding to steady state operation), then, based on the roulette wheel selection approach, almost all of the perturbations would concentrate on the zero error rules, and little improvement would be expended to improve the systems' transient response. In such cases, deliberately reducing the probability of selecting zero error rules and increasing the probabilities of its neighbors will improve the refinement process.

- *Second Stage*

In the second step, a one out of three choice is made. The three rules correspond to the relationships between $e \rightarrow K_p$, $e \rightarrow K_i$ and $e \rightarrow K_d$. A selection probability derived from historical system performance is used with a roulette wheel scheme to choose the rule to be modified.

B. Shift Action

After rule selection, the amount of perturbation needs to be selected. For simplicity, in the experiments perturbation is achieved by fuzzy set shift actions which maintains the shapes of the membership functions and modify only the centers (top locations).

The shift direction (increase or decrease) is selected randomly, but it is based on the *historical record* which remembers the success or failure of the previous trials. A better record indicates a better chance of successful action and is more likely to be chosen. After each trial, the *historical record* is updated accordingly.

The shift amount is randomly generated within a range specified by the user. The lower limit provides the minimum perturbation to guarantee enough difference from the previous trial for new information. The upper bound limits the maximum change in the fuzzy rulebase during each trial. Usually, smaller upper limits slow down the tuning process but lead to a more refined result.

5.6.4 Learning Speed and Convergence Rate

Because of the heuristics implemented in the learing process, the convergence speed is greatly improved. In the experiment, the average percentage of successful perturbations is 37%, ranging from 26% to 51%.

5.7 Case Studies

To evaluated the approach, several tests have been made to determine system performance before and after tuning. In order to make comparisons, the tested plants are the same ones used to design the previous PID controllers. Variable ranges, rulebase settings and system performance before and after tuning are given for each test.

A variable's range is the range used to normalize the corresponding parameter settings. All possible values of each variable lie within the range, though they may not reach the maximum and minimum bounds of the range. In the simulation plots, the values of K_p, K_i, K_d are normalized to [0, 1] based on their ranges; values of E are normalized to [1, 32].

The fuzzy rulebase is shown graphically by the membership functions and plots of the mappings $e \rightarrow K_p, e \rightarrow K_i$ and $e \rightarrow K_d$. The plant output trajectory shows the system performance in response to a step output.

The control objectives for each test plant are listed in Table 5.1.

There are two figures for each plant simulation test. The first one describes the membership functions for variables K_p, K_i, K_d and *error*. The dot or circle lines represent the original positions of these fuzzy sets, and the solid lines show their positions after tuning. Because the original setting approximates a common PID controller, a constant mapping is provided from the fuzzy input to the fuzzy output by setting the output membership functions to be equal. The second figure demonstrates the comparison of $error \rightarrow K_p, K_i, K_d$ relationships and system response before and after tuning.

Table 5.2: Plant I. Fuzzy rulebase variable range before and after tuning.

Variable Range	Before tuning	After tuning
Error	[−0.5 , 0.5]	[−0.5 , 0.5]
K_p	[0 , 2]	[0 , 2]
K_i	[−0.1 , 0.2]	[−0.1 , 0.2]
K_d	[0 , 5]	[0 , 5]

5.7.1 Plant I

1. *Plant Model*

The tested plant is a first order system with transport delay:

$$T(s) = \frac{e^{-9s}}{12s + 1} \tag{5.7}$$

2. *Variable range*

See Table 5.2.

3. *Simulation result*

SA derived PID settings are used to initialize the fuzzy rulebase. Figure 5.4 demonstrates the membership function positions before and after tuning. Figure 5.5 shows the $e \rightarrow K_p, K_i, K_d$ relationships and system performance before and after tuning. Optimal PID controller provides excellent control with a very small overshoot, undershoot, settling time. The Fuzzy-PID controller can also provide quality control as good as optimal PID methods. From the system performance comparison in Figure 5.5, we see a smaller rise time is achieved by the derived Fuzzy-PID controller.

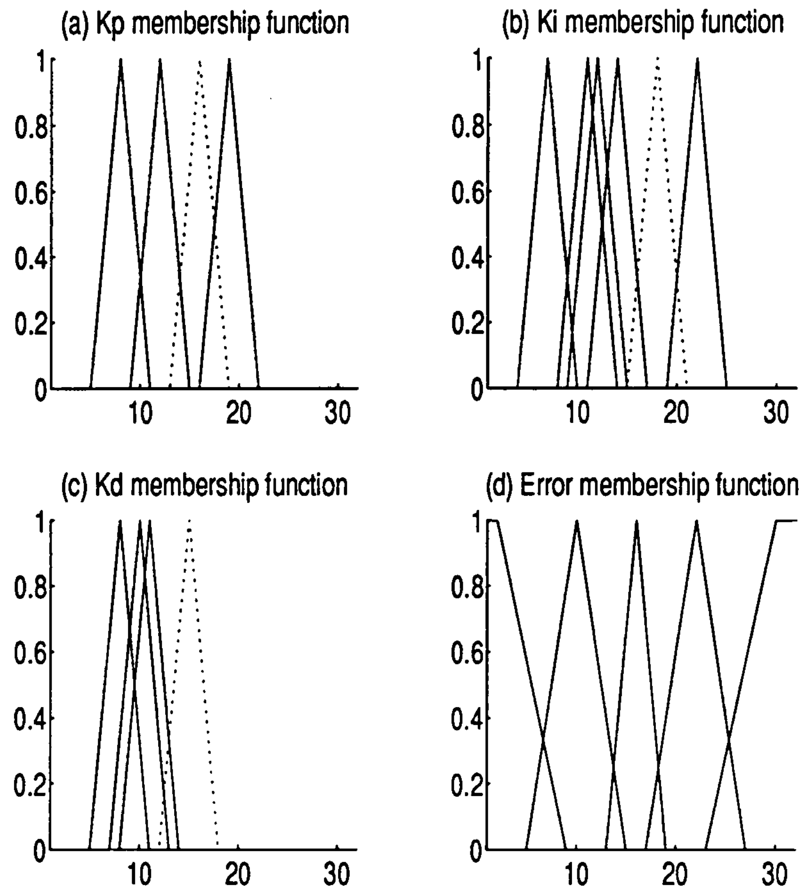


Figure 5.4: Plant I: Fuzzy set locations before (:) and after tuning (-). (a) K_p (proportional gain). (b) K_i (integral gain) (c) K_d (derivative gain). (d) e (error).

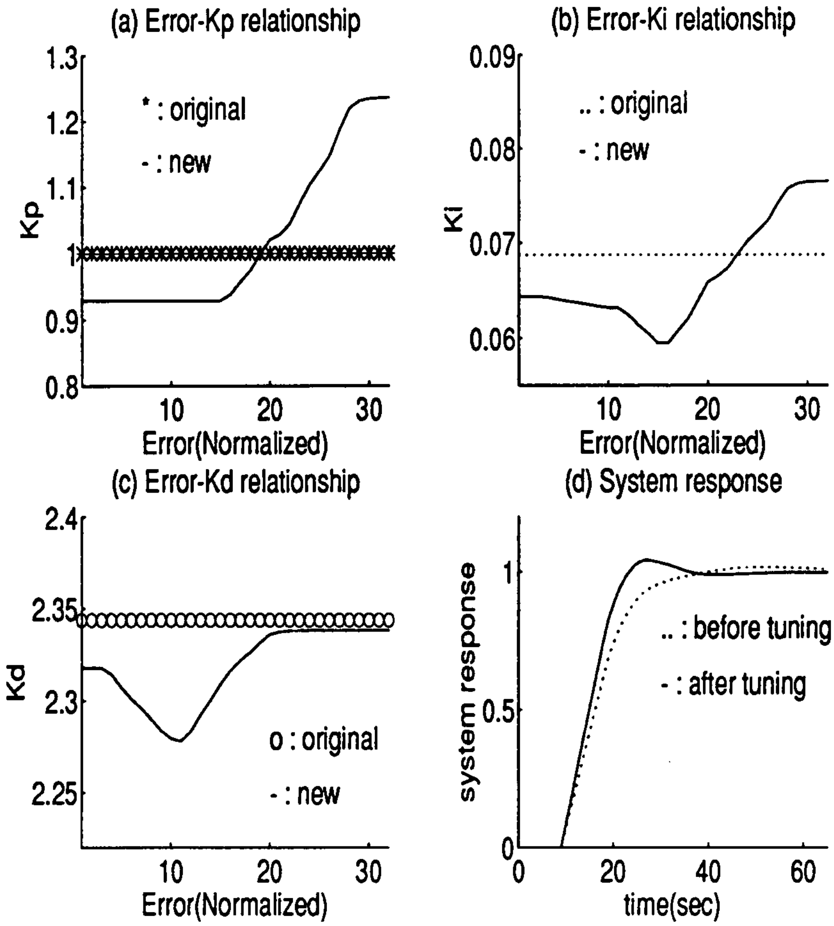


Figure 5.5: Plant I: Fuzzy input-output mappings and system performance before and after tuning. (a) $e \rightarrow K_p$ mapping before (*) and after (-) tuning. (b) $e \rightarrow K_i$ mapping before (:) and after (-) tuning. (c) $e \rightarrow K_d$ mapping before (o) and after (-) tuning. (d) system performance before (:) and after (-) tuning.

Table 5.3: Plant II. Fuzzy rulebase variable range before and after tuning.

Variable Range	Before tuning	After tuning
Error	[−0.5 , 0.5]	[−0.5 , 0.5]
K_p	[−0.5 , 2]	[−0.51 , 2]
K_i	[−0.2 , 0.5]	[−0.2 , 0.5]
K_d	[0 , 0.3]	[0 , 0.3]

5.7.2 Plant II

1. Plant Model

The tested plant is a positional servomechanism model with two left half plane poles and a third pole at the origin:

$$T(s) = \frac{191}{s\,(s+1)\,(s+20)} \tag{5.8}$$

2. Variable range

See Table 5.3.

3. Simulation result

Figure 5.6 demonstrates the membership function positions before and after tuning. Figure 5.7 shows the comparison of $E \rightarrow K_p, K_i, K_d$ relationships and system performance before and after tuning.

The implemented fuzzy initialization is based on the SA optimal PID setting.

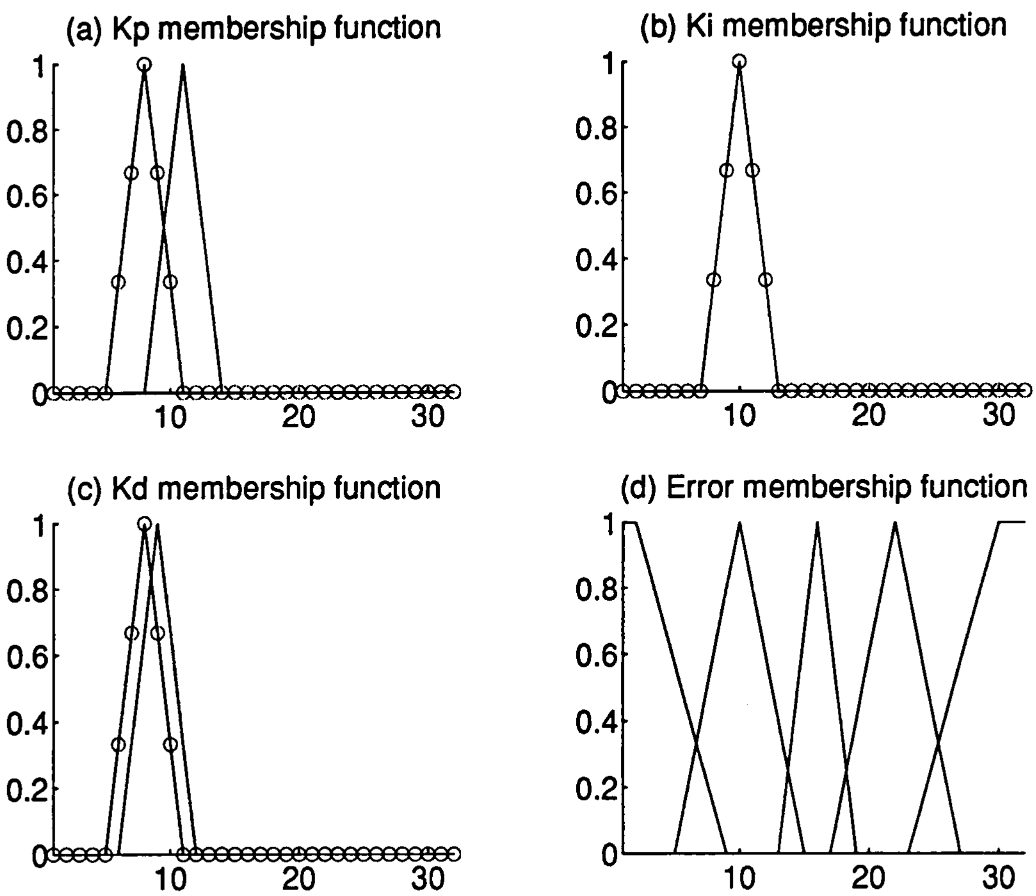


Figure 5.6: Plant II: Fuzzy set locations before (o) and after tuning (-). (a) K_p (proportional gain). (b) K_i (integral gain) (c) K_d (derivative gain). (d) e (error).

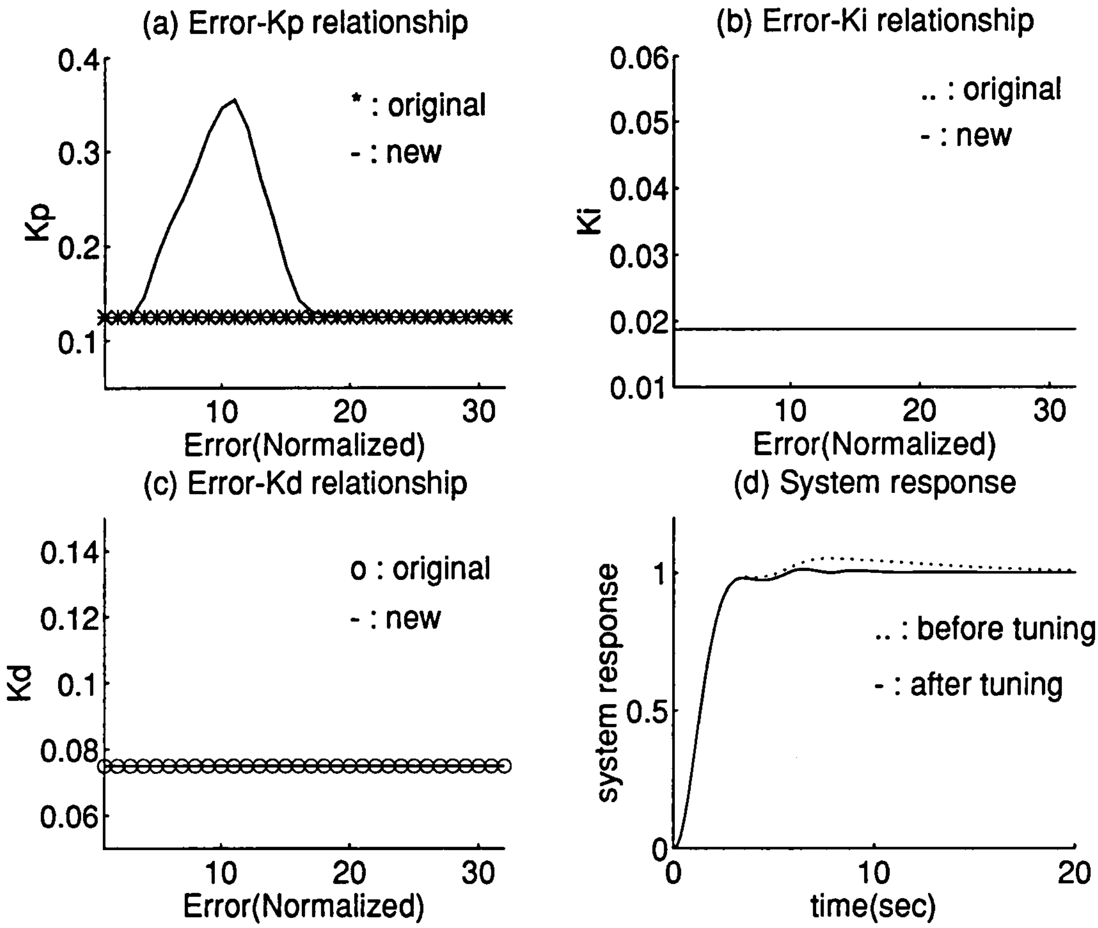


Figure 5.7: Plant II: Fuzzy input-output mappings and system performance before and after tuning. (a) $e \rightarrow K_p$ mapping before (*) and after (-) tuning. (b) $e \rightarrow K_i$ mapping before (:) and after (-) tuning. (c) $e \rightarrow K_d$ mapping before (o) and after (-) tuning. (d) system performance before (:) and after (-) tuning.

Table 5.4: Plant III. Fuzzy rulebase variable range before and after tuning.

Variable Range	Before tuning	After tuning
Error	[−1 , 1]	[−1 , 1]
K_p	[0 , 4]	[0.31 , 2.79]
K_i	[0 , 3]	[0.56 , 2.06]
K_d	[0 , 2]	[0.10 , 0.38]

5.7.3 Plant III

1. Plant Model

The tested plant is a non-minimum phase plant with a right hand plane zero at unity and three poles in the left hand plane.

The transfer function of the model is:

$$T(s) = \frac{-2\,(s-1)}{(s+5)\,(s^2+2s+1.81)} \tag{5.9}$$

2. Variable Range

See Table 5.4.

3. Simulation result

Figure 5.8 illustrates the membership functions before and after tuning. Figure 5.9 gives the comparison of $E \rightarrow K_p, K_i, K_d$ relationships and system performance before and after tuning.

The fuzzy rulebase initialization is based on the SA optimal PID setting.

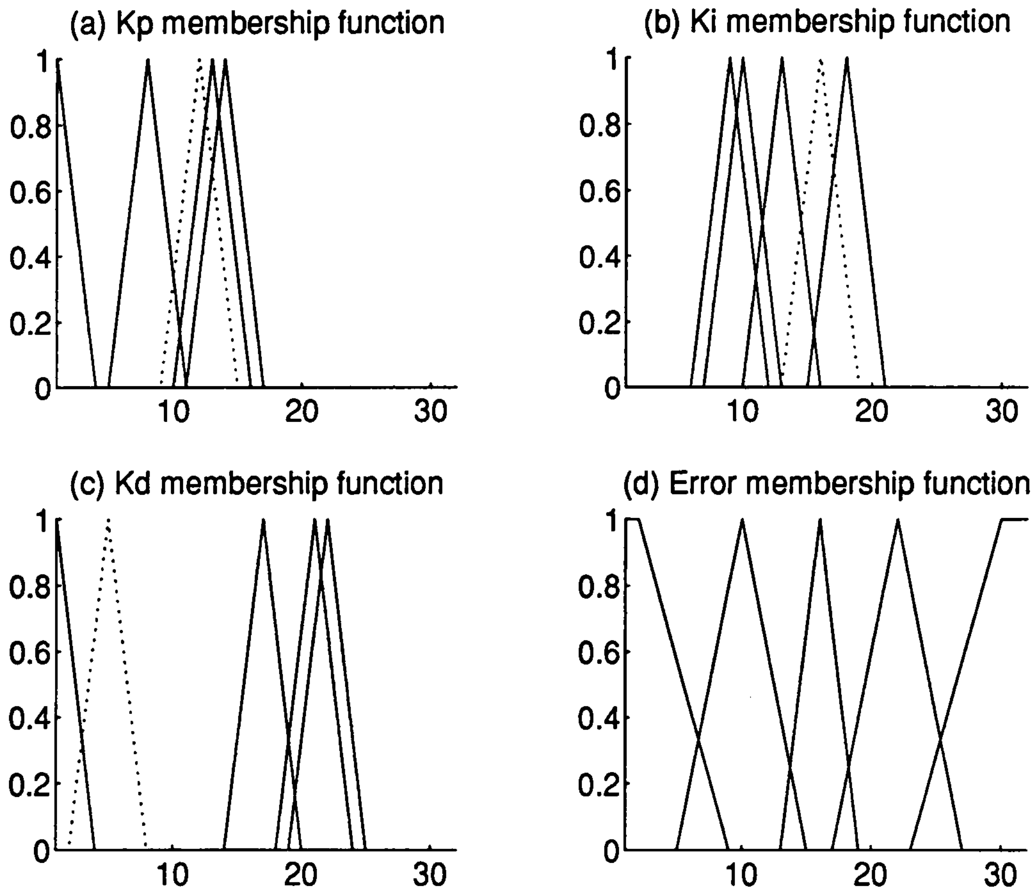


Figure 5.8: Plant III: Fuzzy set locations before (:) and after tuning (-). (a) K_p (proportional gain). (b) K_i (integral gain) (c) K_d (derivative gain). (d) e (error).

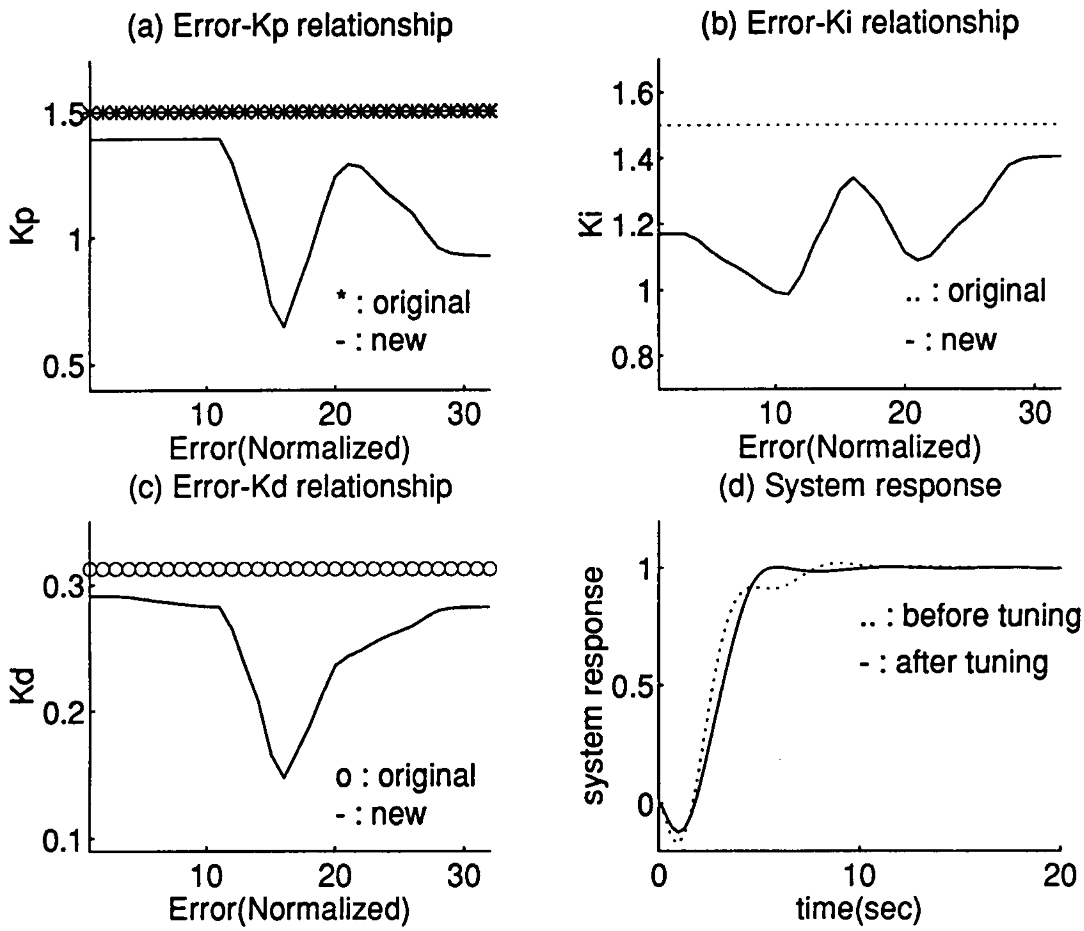


Figure 5.9: Plant III: Fuzzy input-output mappings and system performance before and after tuning. (a) $e \rightarrow K_p$ mapping before (*) and after (-) tuning. (b) $e \rightarrow K_i$ mapping before (:) and after (-) tuning. (c) $e \rightarrow K_d$ mapping before (o) and after (-) tuning. (d) system performance before (:) and after (-) tuning.

Table 5.5: Plant IV. Fuzzy rulebase variable range before and after tuning.

Variable Range	Before tuning	After tuning
Error	[−0.5 , 0.5]	[−0.5 , 0.5]
K_p	[−5 , 10]	[−5 , 10]
K_i	[1 , 9]	[1 , 9]
K_d	[5 , 14]	[5 , 14]

5.7.4 Plant IV

1. *Plant Model*

The test plant is a second order nonlinear model with the state space representation:

$$\begin{cases} \dot{x}_1 & = & x_2 \\ \dot{x}_2 & = & -\sin(x_1) + u * \cos(x_1) \\ y & = & x_1 \\ x_1(0) & = & 0 \\ x_2(0) & = & 0 \end{cases}$$

2. *Variable range*

See Table 5.5

3. *Simulation result*

Figure 5.10 and Figure 5.11 shows the fuzzy membership functions and system performance before and after tuning. The fuzzy rulebase initialization is based on the SA optimal setting.

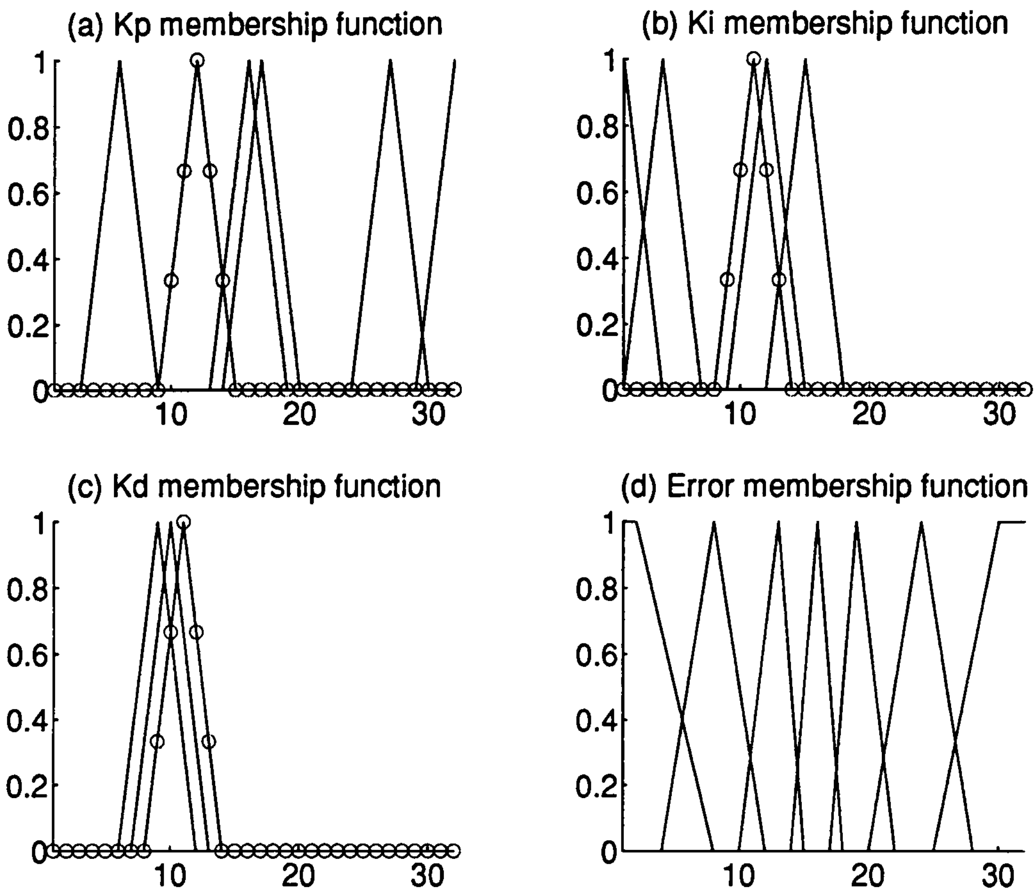


Figure 5.10: Plant IV: Fuzzy set locations before (o) and after tuning (-). (a) K_p (proportional gain). (b) K_i (integral gain) (c) K_d (derivative gain). (d) e (error).

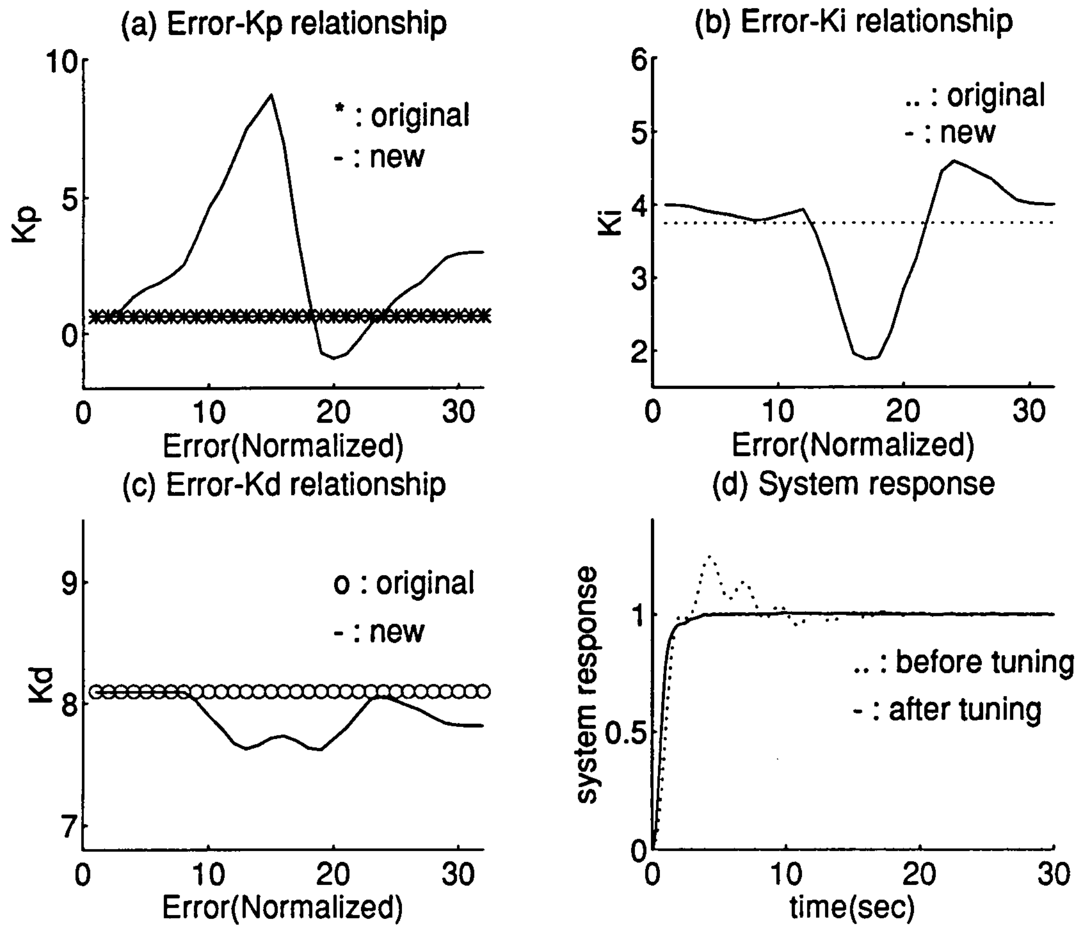


Figure 5.11: Plant IV: Fuzzy input-output mappings and system performance before and after tuning. (a) $e \rightarrow K_p$ mapping before (*) and after (-) tuning. (b) $e \rightarrow K_i$ mapping before (:) and after (-) tuning. (c) $e \rightarrow K_d$ mapping before (o) and after (-) tuning. (d) system performance before (:) and after (-) tuning.

Table 5.6: Plant V. Fuzzy rulebase variable range before and after tuning.

Variable Range	Before tuning	After tuning
Error	[−0.5 , 0.5]	[−0.5 , 0.5]
K_p	[−2 , 10]	[−2 , 10]
K_i	[−2 , 5]	[−2 , 5]
K_d	[0 , 12]	[0 , 12]

5.7.5 Plant V

1. Plant Model

The test plant has a second order nonlinear structure with the state space representation:

$$\begin{cases} \dot{x}_1 &= x_1 * x_1 + u \\ \dot{x}_2 &= 0.2 * x_1 - x_2 + u \\ y &= x_1 - x_2 \\ x_1(0) &= 0 \\ x_2(0) &= 0 \end{cases}$$

2. Variable range

See Table 5.6

3. Simulation result

Figure 5.12 and Figure 5.13 shows the fuzzy membership functions, the fuzzy input-output mapping and system performance before and after tuning. The fuzzy rulebase initialization is based on the SA optimal setting.

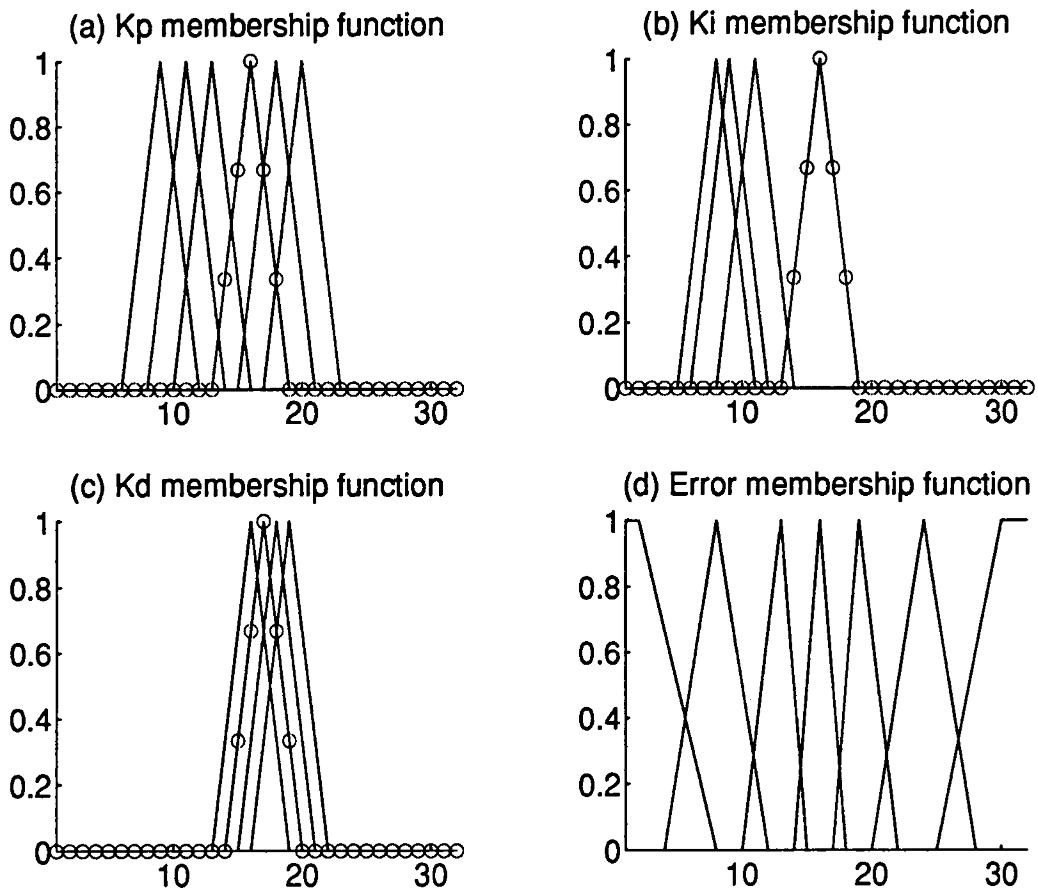


Figure 5.12: Plant V: Fuzzy set locations before (o) and after tuning (-). (a) K_p (proportional gain). (b) K_i (integral gain) (c) K_d (derivative gain). (d) e (error).

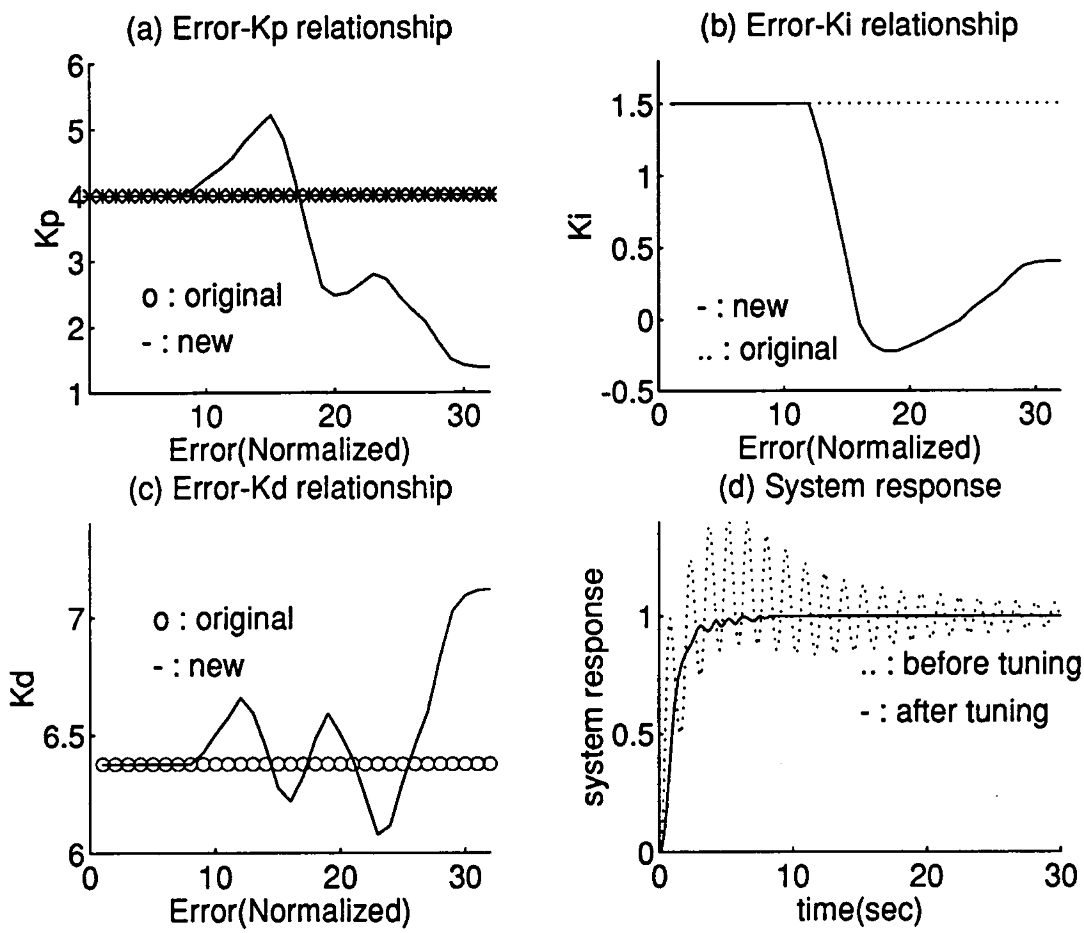


Figure 5.13: Plant V: Fuzzy input-output mappings and system performance before and after tuning. (a) $e \rightarrow K_p$ mapping before (*) and after (-) tuning. (b) $e \rightarrow K_i$ mapping before (:) and after (-) tuning. (c) $e \rightarrow K_d$ mapping before (o) and after (-) tuning. (d) system performance before (:) and after (-) tuning.

5.7.6 Simulation Discussion

The tests show that good controllers that meet very stringent requirements can be developed through the tuning process. From the simulations, we can see that:

- Comparing the system performance before and after tuning for each plant, as shown in the *system response* subplot, one observes a great improvement after the learning process, especially for the nonlinear plants. Almost all the factors specified in the cost function, overshoot, undershoot, rise time, steady state error, have greatly improved. The nonlinear input-output mapping provides a much better control law than that from the traditional PID control, so the system satisfies more restrictive requirements.

- Before learning, the fuzzy block approximates a fixed PID controller. As indicated by the star (*), dot(:) and circle (o) lines in the $e \rightarrow K_{p,i,d}$ relationship subplots, a constant relationship is provided between fuzzy input and output. The PID parameter setting is derived from either Ziegler-Nichols , ITAE or the SA optimization method in each case. These are the same settings that are presented in Chapter 3. The input fuzzy sets cover the entire input domain. All the output membership functions representing the same variable overlap with each other in order to achieve the fixed gain approximation.

- In the experiments, different fuzzy rulebase initializations have been tried. This has a big influence on convergence rates.

- From the comparison of fuzzy set location and input-output relation plots before and after tuning, we can see that the output membership functions are scattered by the shift actions. A nonlinear mapping is achieved from the input domain of e to the output domains of K_p, K_i and K_d. The new function provides for a better control law enabling the system to achieve a better performance.

5.7.7 Stability Analysis

The Fuzzy-PID controller's structure introduces nonlinearity into the system because of the fuzzy rulebase. Theoretical stability analysis of fuzzy systems is still in its infancy. However, we can view the Fuzzy-PID controller as a time varying model whose configuration is changing within a known compact set. As shown in Figure 5.14, $\vec{I}$ is the fuzzy input vector; $\vec{p}$ is the PID gain vector; and the fuzzy mapping $F(I)$ takes value on a compact set C. In this study, because there is only one fuzzy input e which is mapped to the integers between 1 and 32, this compact set indexes 32 known fixed PID controllers. The stability of the system can be guaranteed through analysis of the compact set. The set that represents the possible systems (loop transfer dynamics) where different PID configurations are used is denoted S_p.

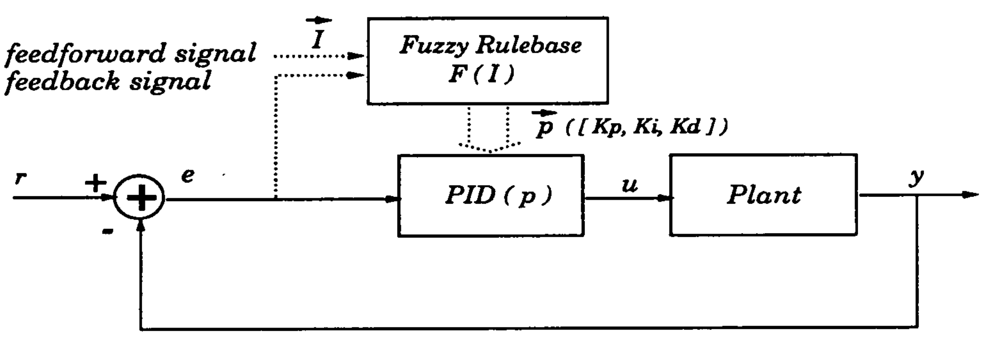


Figure 5.14: Fuzzy-PID control system.

Lemma 5.1 *The fuzzy mapping from e to K_p, K_i, K_d ($f_{FUZZY} : e \rightarrow K_p, K_i, K_d$) is continuous.*

Proof: The mapping from e to K_p, K_i and K_d is obtained from fuzzy inference which is a composition of *maximum* and *minimum* operations, which are continuous; thus this mapping is continuous.

Theorem 5.1 *For the Fuzzy-PID system with the structure shown as Figure 5.14, where $\vec{p}$ is the PID gain vector and $\vec{I}$ is the fuzzy input vector ($\vec{I} = e$), and where the fuzzy mapping $p = f_{FUZZY}(I)$ takes value on a compact set C, let $S = \{ S_p \mid p \in C \}$ is the set of the possible systems where different PID configurations are used.*
Suppose that there exists a Lyapunov function $V(x)$, where x is the state vector of the system, such that for all the systems $S_p \in S$ for $p \in C$, $V_p(x) > 0$ $\dot{V}_p(x) < 0$ for $x \neq 0$, where the subscript p denotes the closed-loop system with the application of $V(x)$ to the system indexed by $p \in C$, then $V(x)$ is a Lyapunov function for the Fuzzy-PID controller and the Fuzzy-PID controller is asymptotically stable.

Proof: The PID gain vector p can be considered as the parameter of the overall system. This parameter variation during operation is bounded by the compact set C. Note that there exists a Lyapunov function $V(x)$ for all S_p, $p \in C$, i.e. $V_p(x) > 0$, $\dot{V}_p(x) < 0$ and we want to prove that for the Fuzzy-PID system, $V(x) > 0$ and $\dot{V}(x) < 0$.
Because system behavior is constrained by a compact set C and suppose that:

Claim: $V(x)$ and $\dot{V}(x)$ are continuous functions.

We have:

$$V(x) \geq \min_{p \in C} V_p(x) > 0$$

$$\dot{V}(x) \leq \max_{p \in C} \dot{V}_p(x) < 0$$

where strict inequality is guaranteed by compactness. Thus $V(x)$ is a Lyapunov function of the Fuzzy-PID system, and the asymptotic stability of the Fuzzy-PID system is guaranteed.

Proof of Claim:

(1) $V(x)$ does not depend upon p and is continuous.

(2) $\dot{V}(x) = [\ \frac{\partial V}{\partial x}\ (x)\ ]^*\ \dot{x}$, where

$$\dot{x} = f_{SYSTEM}(x,t,p) = f_{SYSTEM}(x,t,f_{FUZZY}(e)).$$

Because both f_{SYSTEM} and f_{FUZZY} are continuous, their composition is continuous. Thus both $\dot{V}(x)$ and $V(x)$ are continuous functions.

corollary 1 *If the plant in the Fuzzy-PID system is linear time invariant, the systems S_p can be represented as $\dot{x} = A_p x$, where A_p is a system state matrix. Suppose there exist positive definite hermitian matrices M and N_p, $p \in C$, such that $A_p^* M + M A_p^* = -N_p$. Then $V(x) = x^* M x$ is the Lyapunov function of the Fuzzy-PID system, and the Fuzzy-PID system is asymptotically stable.*

5.7.8 Robustness Test

System models always drift from the nominal description during operation. It is difficult, and perhaps impossible, to prove robustness, or guaranteed stability in the presence of unmodeled plant dynamics, for the Fuzzy-PID controlled system. In regard to this, simulation tests are performed for selected plant variations. In the simulation tests, we provide parameter perturbation to the nominal model to test if the derived rulebase still works well.

1. Test a. Plant I

Plant I is used as the test model. The original transfer function is: $T(s) = \frac{e^{-9s}}{12s+1}$. Four perturbations tests are shown in Figure 5.15.

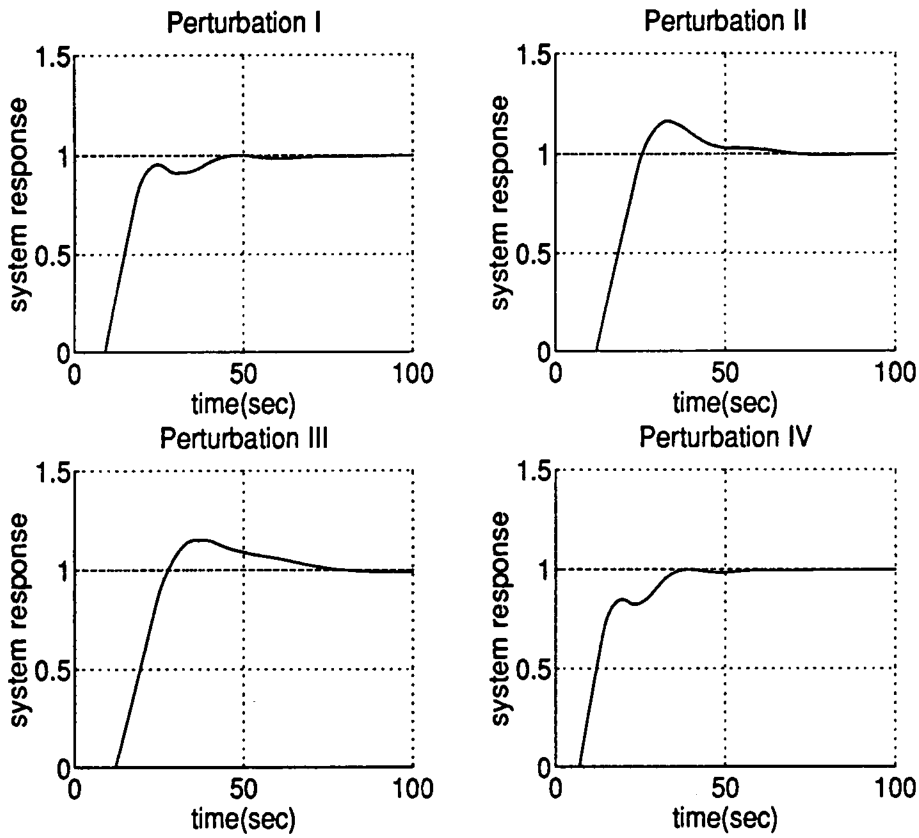


Figure 5.15: Plant I: Robustness test: four structural perturbation I – IV are introduced to the plant model.

In perturbation I, the transfer function is: $T(s) = \frac{1}{10s+1}e^{-9s}$

In perturbation II, the transfer function is: $T(s) = \frac{1}{14s+1}e^{-12s}$

In perturbation III, the transfer function is: $T(s) = \frac{1}{10s+1}e^{-12s}$

In perturbation IV, the transfer function is: $T(s) = \frac{1.1}{10s+1}e^{-7s}$

From Figure 5.15, we see that the controller still drives the system to steady state with "acceptable" overshoot, undershoot, rise time and settling time. However, the

Table 5.7: Plant IV. Structural perturbation I–IV description in the robustness test.

Perturbation I	Perturbation II
$\dot{x}_1 = x_2 * 0.9$	$\dot{x}_1 = x_2$
$\dot{x}_2 = -sin(x_1) + u * cos(x_1)$	$\dot{x}_2 = -1.1 * sin(x_1) + u * cos(x_1)$
$y = x_1$	$y = x_1$
$x_1(0) = 0$	$x_1(0) = 0$
$x_2(0) = 0$	$x_2(0) = 0$
Perturbation III	**Perturbation IV**
$\dot{x}_1 = x_2 * 0.9$	$\dot{x}_1 = x_2 * 0.9$
$\dot{x}_2 = -1.1 * sin(x_1) + u * cos(x_1)$	$\dot{x}_2 = -1.1 * sin(x_1) + 0.8 * u * cos(x_1)$
$y = x_1$	$y = x_1$
$x_1(0) = 0$	$x_1(0) = 0$
$x_2(0) = 0$	$x_2(0) = 0$

optimal controller dedicated to the nominal plant does not give optimal control to the shifted plants. In this example, especially for variations in delay time, the nominal controller is not the best design for the shifting plants.

2. Test b. Plant IV

Plant IV is used for the test. Parameter variations given by Table 5.7 are introduced to the nominal plant, and the simulation tests for the perturbed systems are shown in Figure 5.16. From the simulation, we can see that the derived Fuzzy-PID controller works quite well for the perturbed models, even if some of the parameters in the plant state space representation are changed 10 to 20%. The step response still shows a small overshoot, undershoot, rise time and acceptable settling time.

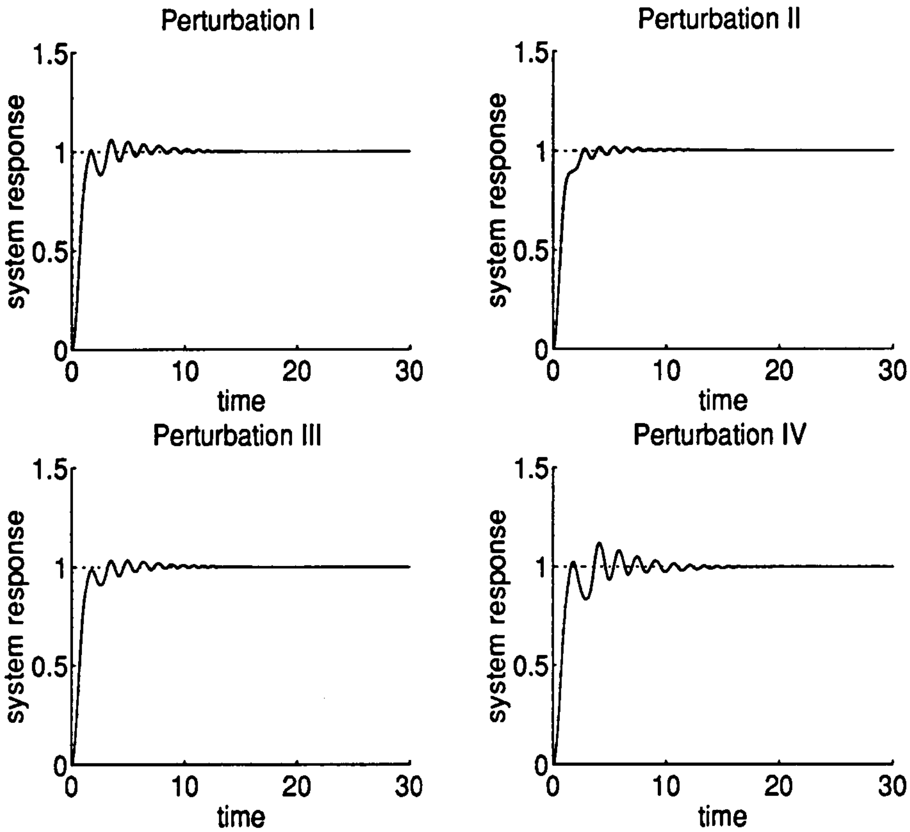


Figure 5.16: Plant IV. Robustness test: four structural perturbation I – IV are introduced to the plant model.

5.7.9 Disturbance Tolerance Test

Noise at the output measurement node, plant input node and controller input node are introduced to test the closed-loop system disturbance tolerance. As shown in Figure 5.17, the RMS (root mean square) value of disturbances increases from 1%. 2%, 5% to 8% for the subplot *Disturbance I, II, III and IV.* We can see that although the system performance degrades as the noise disturbance increases, the output is still centered at the desired value.

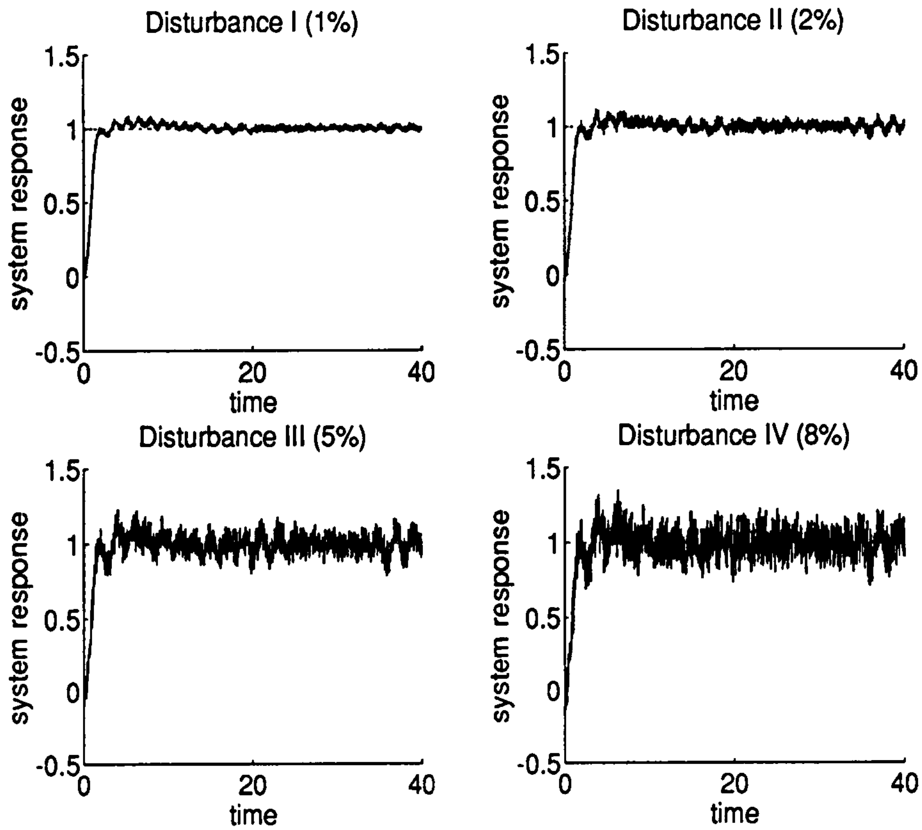


Figure 5.17: Plant IV. Noise tolerance test: RMS value of the noise starts from 1%, 2%, 5% to 8% for disturbance I–IV.

5.7.10 Different Command Following Test

Several numerical experiments where different signals are used as plant inputs have been tried to test if the derived controller works well under different operating conditions. Plant IV is used as the tested system.

Figure 5.18 shows the performance of the derived Fuzzy-PID controller and Figure 5.19 demonstrates the optimal PID control performance. In both figures, for the upper plot (test I), the input switches between several setpoints; for the lower plot (test

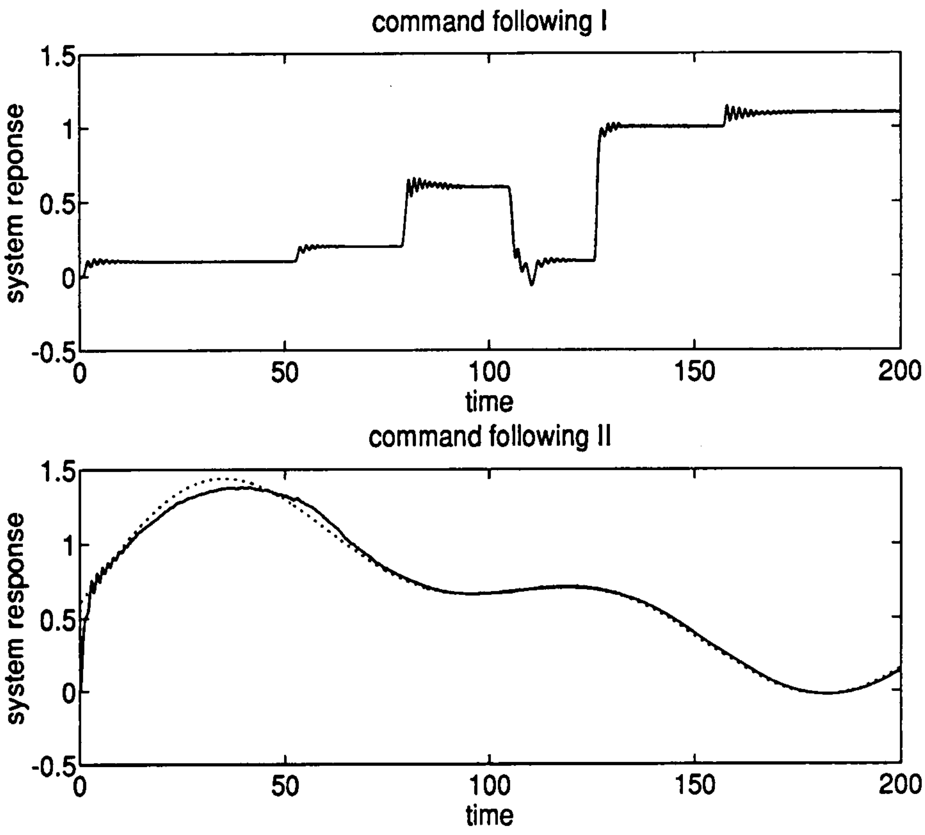


Figure 5.18: Plant IV (Fuzzy-PID control). Different input signal test: dashed line indicates the input signal, solid lines represents the system output.

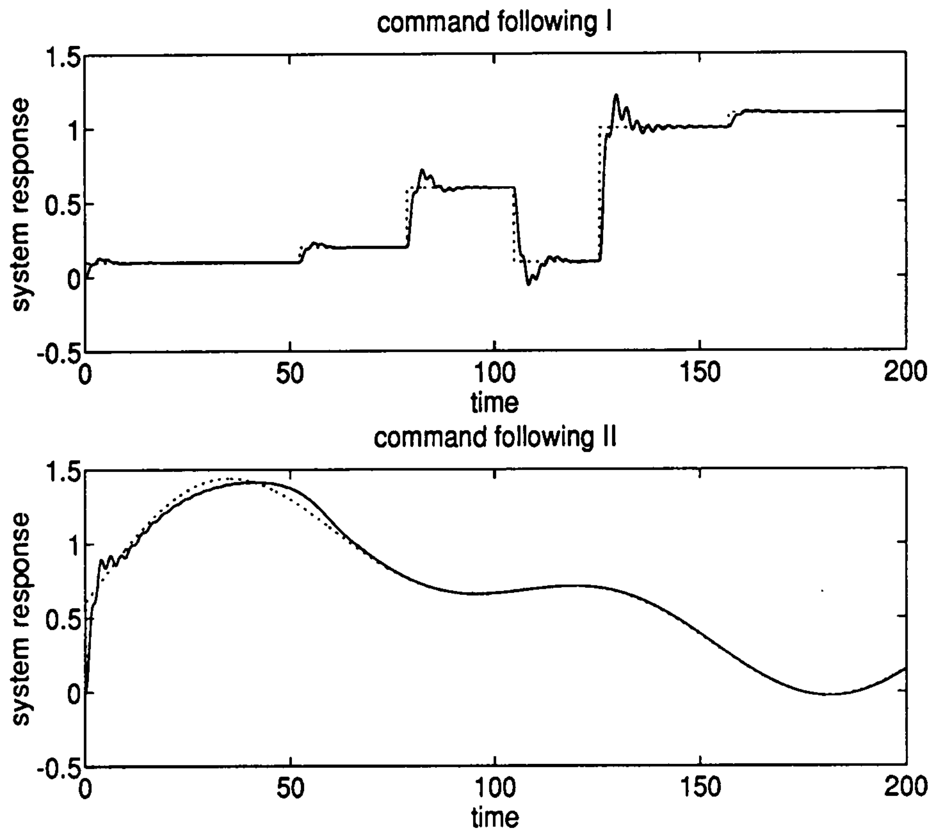


Figure 5.19: Plant IV (optimal PID control). Different input signal test: dashed line indicates the input signal, solid lines represents the system output.

II), the input is periodic. The dashed line in the plot indicates the reference signal, and the solid line represents the system response.

We can see that the output follows the input change most of the time; however, sometimes, a larger overshoot appears. As in the test of setpoint change, when the signal level decreases a lot, the system output results in a big overshoot. This is because the system is nonlinear, and during the rulebase construction, a unit step signal is used, the fuzzy input variable ($e = r - y$) is only [-0.5, 0.5]. The rulebase is designed for error signals in response to a unit step from zero. Design using only unit step tests is adequate for linear systems, this is not sufficient for nonlinear systems. If more information is added to the rulebase, the system response to other setpoint changes should improve.

In test II, the output follows the sinusoidal signal quite well, except for the early transient. The simulation shows the derived rulebase can generate proper control actions that drive the system output to follow the input signal.

Comparing the performance of Fuzzy-PID control and PID control, smaller overshoot and undershoot are achieved at most of the setpoint changes for Fuzzy-PID design; however, a little more oscillation around the final setpoints is present. For the trained unit step signal the Fuzzy-PID design does give better performance over PID control. Because the test plant is nonlinear, the system will response differently at different operating point and with respect to different input signal; the unit step function is just the first step of the investigation. In later research, more complicated signals such as randomly changing setpoint signals should be used as training inputs in order to obtain more information about the plant.

5.8 Discussion

5.8.1 Advantages and Comparison

In the Fuzzy-PID structure, the fuzzy block stores the control strategy and improves system performance as shown in the above simulations. It is able to construct a more flexible and more powerful controller in order to handle complicated situations.

1. A generalization of PID control

   PID control is adequate in many applications. However, linear constant-gain feedback may not be sufficient when the operating conditions change. For some complicated nonlinear plants, where system behavior depends on the operating point, the simple PID structure often does not provide for proper action at all times to meet very stringent control requirements. The Fuzzy-PID design method proposes an alternative approach to solve this problem. It maintains the original PID structure and adds more flexibility to cope with different circumstances. The performance of Fuzzy-PID controller is at least as good as the optimal PID controller.

   Lemma 5.2 *The Fuzzy-PID controller is capable of better performance than traditional PID controller. Any PID design can be implemented with the Fuzzy-PID structure. The derived Fuzzy-PID controller can provide control at least as good as the optimal PID controller.*

   Proof: The traditional PID controller can be viewed as a special Fuzzy-PID controller whose mappings from the fuzzy inputs to outputs are constants. Assuming the learning algorithm is designed to always remember the best controller has previously encountered, the optimal Fuzzy-PID controller after the rulebase learning process will perform at least as well as the optimal PID controller.

2. Fuzzy control

The Fuzzy-PID controller is a special fuzzy controller. The fuzzy rulebase generates outputs to adjust the gains of a PID controller. Fuzzy inputs and outputs can be any signal chosen by the designer. For the model used in the simulation, if the proportional and derivative signals are zero, and if the derivative of the error signal is added as another input to the fuzzy block, the Fuzzy-PID structure becomes the common fuzzy controller structure, which only generates the rate of change for the plant control input. The Fuzzy-PID structure is a more complicated and more flexible model than the commonly used fuzzy controller structure.

In practice, many plants are controlled by PID loops. The Fuzzy-PID structure provides an approach for modification of these PID loops and improved performance.

A central portion of the Fuzzy-PID structure is the PID configuration whose stability and robustness analysis has been well developed. However, fuzzy robustness and stability theory is still in its infancy. In many cases, the fuzzy design robustness is not guaranteed during the early design stage; instead, it is demonstrated during testing by simulation or real-time experiment. As suggested in Sections 5.7.7 and 5.7.8, if we can examine all the "time-shared" fixed PID controllers that compose the Fuzzy-PID model, then we hope the stability and robustness property of the design can be guaranteed. However, this is only a sufficient condition, not a necessary one and maybe very conservative. More effort is needed to explore this topic.

3. Gain scheduling

The proper gain adjustment of the PID controller under different circumstances provided by the fuzzy rulebase enables the system to improve its performance.

In this design, only the error signal between the reference and system output is used as the fuzzy input. However, the fuzzy input signal can be any variable that affects the system dynamics. In many systems, there are auxiliary variables that are closely related to system dynamics characteristics. For nonlinear systems the plant's measured output, or another desired signal, could be used to indicate the mode of the plant's operation, as in gain scheduled controllers. Measurement can be used as fuzzy inputs to determine the proper PID gains. For example, in a typical heat exchanger, the nonlinearities in temperature response have a great effect on system behavior. The controller works fine when the setpoint is maintained at a low temperature value; however, when the setpoint is increased to a higher value, the controller suddenly becomes unstable [8]. In such circumstances, auxiliary variables should be considered in the controller design. 197z The Fuzzy-PID controller can be viewed as a gain scheduling approach with fuzzy logic applied to realize the proper mapping. However, conventional gain scheduling is an open-loop compensation approach and can be viewed as a system with feedback control in which the feedback gains are adjusted by feedforward compensation. There is no feedback from the performance of the closed-loop system to compensate for an incorrect schedule [1]. In the Fuzzy-PID structure, signals from the feedback loop can also be introduced to the gain scheduling process, which admits more flexibility to achieve a better control. This is similar to techniques in adaptive control [1].

Fuzzy system can approximate any continuous function on a compact domain to any degree of accuracy [16]. Theoretically, any complicated and precise scheduling scheme can be encoded into a fuzzy block given sufficient memory capacity. The fuzzy also representation softens the effect of inaccurate measurements.

4. Self-Tuning Controller

Control loop tuning has been a problem since the earliest application of PID control. Unfortunately, no statistics exist on the economic losses resulting from the significant fraction of improperly tuned loops. A disturbance or a set point change often causes the process to exhibit an unsatisfactory transient response; as a result, the operator shifts the loop to manual control and, after the process returns to steady state, shifts the loop back to automatic control. This common practice greatly reduces the benefits of automatic control and increases the burden on the operator [8].

The basic concept of self-tuning, or adaptation, is that the controller changes its own PID settings during normal closed-loop operation so as to ensure optimal control performance at all times. It can correct improperly tuned control loops and adapt to changes in the operating environment [8].

The Fuzzy-PID structure provides an approach for self-tuning controller design. All the relevant factors of the system dynamics should be considered during the fuzzy rulebase construction stage; all the tuning strategies need to be stored in the fuzzy rulebase. If all the tuning information were encoded in the rulebase, the controller would be able to act as an expert and handle difficult situations by updating the PID settings.

5.8.2 Disadvantages

The Fuzzy-PID structure introduces enhanced flexibility in the controller design, however, it also introduces additional complexities into the system.

For linear systems, the nonlinear function destroys the original linearity and makes it harder to predict system behavior. Stability and robustness analysis become more difficult as well. No conventional method can be easily applied.

Compared with PID or the simple fuzzy controller structure, it is a more complicated model. An optimal rulebase can only be obtained with a thorough learning process which is time consuming and involves much computation. During tuning or real-time operation, the controller requires significant computing capabilities.

The proposed Fuzzy-PID controller is only suitable for those plants whose behavior can be controlled by an input signal that can be synthesized by the Fuzzy-PID model. Thus the synthesis capability of the Fuzzy-PID structure and the plant dynamics may limit the potential applications.

5.8.3 Potential Implementation

The Fuzzy-PID structure can be used to improve system performance, especially for those systems operating under changing conditions, whose structure is highly non-linear, or whose behaviors largely depend on the variations of the environment or some measurable signals.

With the continued development of VLSI technology, the fuzzy algorithm can be easily coded into an ASIC design, and the speed of fuzzy inference can be greatly increased. Reprogrammable gate-arrays can be used if retuning of the rulebase is desired.

CHAPTER 6

Conclusion

A new Fuzzy-PID structure is proposed in this study in order to improve closed-loop system performance and meet stringent control specifications, especially for plants with complicated dynamics, operating under changing conditions.

The widely used traditional PID structure is implemented for several test plants to demonstrate its control capabilities and its limitations. The simulation results of the PID controller derived from Ziegler-Nichols and ITAE methods show that these two tuning methods only apply to plants with simple dynamics which can be approximated as low-order systems. The performance of the optimal PID controller derived from the simulated annealing algorithm demonstrates that PID control is sufficient for many plants; however, it may not handle very complicated structures and dynamics with many nonlinearities or high degree of uncertainty.

Fuzzy set theory is implemented in the new controller design. The proposed Fuzzy-PID structure maintains the basic PID structure; however, its gains are updated continuously through fuzzy inference based on the information stored in the fuzzy rulebase and the current condition. This mechanism provides a way to cope under a variety of circumstances. The controller can generate adaptive control actions by encoding the information that affects the system behavior into the rulebase and applying extra inputs to the fuzzy block. The Fuzzy-PID structure can be generalized to provide efficient actions to deal with dynamical and external changes.

The fuzzy rulebase provides an optimal mapping from the input domain, which is the obtainable measurements to the output domain, which composes the control gains. The optimal mapping is achieved by optimal parameter settings in the fuzzy rulebase. This usually imposes a big problem for optimal rulebase construction. Here, a successive refinement learning process which applies probability theory and heuristics is developed to find the optimal rulebase setting.

The Fuzzy-PID controller generated through the successful refinement approach shows superior closed-loop system performance compared to PID controllers, especially for complex or nonlinear models, because the control action provided by the Fuzzy-PID structure is more flexible and can adapt to a broader set of variations. This statement holds regardless of the method used to determine the PID controller's gains.

The new structure demands better computation capability, and introduces significant nonlinearity into the closed-loop system dynamics, which makes the analysis of the closed-loop system behavior a lot more difficult. An attempt to analyze the stability and robustness properties of the new structure is provided. More theoretical analysis is needed in future research.

In summary, the Fuzzy-PID structure can provide excellent control for many complicated systems. It underlies a new approach for PID-tuned controller design to deal with stringent control requirement, nonlinearity and uncertainty.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Åström, K. J. and B. Wittenmark, *Adaptive Control.* Addison-Welsley Publishing Company, Reading, 1989.

[2] Åström, K. J., and T. Hägglund, *Automatic Tuning of PID Controllers.* Instrument Society of America, Research Triangle Park, 1988.

[3] Bernard, J. A., "Use of rule-based system for process control," *IEEE Control Systems Magazine*, vol. 8, no. 5, 1988, pp. 3-12.

[4] Berenji, H. R. and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. on Neural Networks*, vol. 3, no. 5, 1992, pp. 724-740.

[5] Camacho, E. F., F. R. Rubio, and F. M. Hughes, "Self-tuning control of a solar power plant with a distributed control field," *IEEE Control Systems*, vol. 12, no. 12, 1992, pp. 72-77.

[6] Chang, G., *System Cultivation Methods in Intelligent Process Supervision.* Ph.D. dissertation, Dept. of Electrical and Computer Engineering, University of Tennessee, Knoxville, TN., December, 1991.

[7] Chapnick, P., "Fuzzy wuzzy and then some," *AI Expert.* vol. 7, no. 3, 1992, pp. 20-21.

[8] Chia, T., "Some basic approaches for self-tuning controllers," *Control Engineering*, vol. 39, no. 13, Dec, 1992, pp. 49-52.

[9] Cook, P. A., *Nonlinear Dynamical Systems.* Prentice-Hall, Englewood Cliffs, 1986.

[10] Davis, L., *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[11] D'Azzo, J. J. and C. H. Houpis, *Linear Control System Analysis and Design: Conventional and Modern.* 3rd Edition, McGraw-Hill, New York, 1988.

[12] Dorf, R. C., *Modern Control Systems.* 5th Edition, Addison-Welsley, Reading, 1989.

[13] Jang, J. R., "Self-learning fuzzy controllers based on temporal back propagation," *IEEE Trans. on Neural Networks*, vol. 3, no. 5, 1992, pp. 714-723.

[14] Kirkpatrick, S., C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, 1983, pp. 671-680.

[15] Klir, G. J. and T. A. Folger, *Fuzzy Sets, Uncertainty and Information.* Prentice Hall, Englewood Cliffs, 1988.

[16] Kosko, B., "Fuzzy function approximation," *IEEE International Conference on Fuzzy Systems*, San Diego, March 8-12, 1992.

[17] Kosko, B., *Neural Networks and Fuzzy Systems: a dynamical systems approach to machine intelligence.* Prentice Hall, Englewood Cliffs, 1992.

[18] Laarhoven, P. J., *Simulated Annealing : Theory and Application.* D. Reidel Publishing Company, Dordrecht, Holland, 1987.

[19] Lee, C. C., "Fuzzy logic in conrol systems: Fuzzy logic controller – part I," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 20, no. 2, 1990, pp. 404-418.

[20] Maeda, M., T. Sato, and S. Murakami, "A design of the self-tuning fuzzy controller," *Proc. of the International Conference on Fuzzy logic and Neural Networks*, (Iizuka, Japan), pp. 393-396, July 20-24, 1990.

[21] Mamdani, E. H., "Application of fuzzy algorithms for control of simple dynamic plant," *Proc. IEE 121(12)*, pp.1585-1588, 1974.

[22] Ogata, K., *Modern Control Engineering.* Prentice Hall, Englewood Cliffs, N.J., 1970.

[23] Raju, G. V., J. Zhou and R. A. Kisner, "Hierachical fuzzy control," *Int. J. Control*, vol. 54, no. 5, 1991, pp. 1201-1216.

[24] Safonov, M. G., *Stability and Robustness of Multivariable Feedback Systems.* The M.I.T. Press, Cambridge and London, 1980

[25] Saito, Y. and T. Ishida, "Fuzzy PID Hybrid Control," *The 6th Fuzzy System Symposium*, pp. 65-69.

[26] Sarma, P., A. Ikonomopoulos, T. Wang, "Fuzzy logic control of a continuous exothermic reactor," Submitted for publication to the *Computers and Chemical Engineering Journal*, 1993.

[27] Seborg, D. E., T. F. Edgar and D. A. Mellichamp, *Process Dynamics and Control.* John Wiley & Sons, New York, 1989.

[28] Willems, J. C., *The Analysis of Feedack Systems.* The M.I.T. Press, Cambridge and London, 1971.

[29] Won, C., S. Kim and B. K. Bose, "Robust position control of induction motor using fuzzy logic control," presented at *IEEE Industrial Application Society Annual Meeting*, Houston, TX, October 1992.

[30] Yasunobu, S. and S. Miyamoto, "Automatic train operated by predictive fuzzy control," *Industrial Applications of Fuzzy Control*, (M.Sugeno, ed.) Holland,Amsterdam, 1985.

[31] Yoshida, M., Y. Tsutsumi and T. Ishida, "Gain tuning method for design of fuzzy control systems," *The 6th Fuzzy System Symposium*, 1990, pp. 405-408.

[32] Zadeh, L. A., "Fuzzy sets," *Information and Control,* vol. 8. pp. 338-353, 1965.

[33] Zhou, G., *Automatic design of autotuners for PID design.* Ph.D. Dissertation, Dept. of Electrical and Computer Engineering, University of Tennessee, Knoxville, TN., August 1993.

[34] *Matlab Reference Guide.* The MathWorks, Natick, 1992.

APPENDIX

```
%*******************************************************%
%    Functionality: initializing MSF parameters          %
%*******************************************************%

function NFDini

global fuzKp fuzKi fuzKd FuzKp FuzKi FuzKd
global LBe UBe LBKp UBKp LBKi UBKi LBKd UBKd
global RangeKp RangeKi RangeKd RangeE D  Width
global Num Nset MI MO MOrecd J delta  Y R Probrule
global BASE STATUS_original  PID_select s STATUS_min

                          % initializing the parameters
Nset = 5;                 % Fuzzy rule number: 5(or 3,7)
Num = 32;                 % The discrete number to represent a fuzzy set:32
MI = zeros(Nset, Num);       % MI: input fuzzy sets
MO = zeros(Nset*3, Num);  % MO: output fuzzy sets
MIrecd = zeros(Nset, 3);
MOrecd = zeros(Nset*3, 3);
delta0 = floor(Num/(Nset-1));
delta = max(1, delta0-3);
arisetime = 0.01;

Probrule = 0.5*ones(Nset*3,1);

LBe = -0.5;    UBe = 0.5;    RangeE = UBe - LBe;     % Plant I
LBKp = 0;      UBKp = 2;     RangeKp = UBKp-LBKp;
LBKi = 0;      UBKi = 0.5;   RangeKi = UBKi-LBKi;
LBKd = 0;      UBKd = 5;     RangeKd = UBKd-LBKd;

%LBe = -0.5;  UBe = 0.5;    RangeE = UBe - LBe;     % Plant 2
%LBKp = -0.5; UBKp = 2;     RangeKp = UBKp-LBKp;
%LBKi = -0.2; UBKi = 0.5;   RangeKi = UBKi-LBKi;
%LBKd = 0;    UBKd = 0.3;   RangeKd = UBKd-LBKd;

%LBe = -1;    UBe = 1;      RangeE = UBe - LBe;
%LBKp = 0;    UBKp = 4;     RangeKp = UBKp-LBKp;     % Plant 3
%LBKi = 0;    UBKi = 3;     RangeKi = UBKi-LBKi;
%LBKd = 0;    UBKd = 2;     RangeKd = UBKd-LBKd;

Width = [3; 3; 3; 3; 3];

MI(1,:) = msf([1,    2,  9]);           % input fuzzy set initialization
MI(2,:) = msf([10-5, 10, 10+5]);
MI(3,:) = msf([16-3, 16, 16+3]);
MI(4,:) = msf([22-5, 22, 22+5]);
MI(5,:) = msf([23,   30, 32]);

Kp00=1.5;                             % output fuzzy set initialization
Ki00=1.50;
Kd00= 0.36;

sKp = floor((Kp00-LBKp)/RangeKp*Num);
sKi = floor((Ki00-LBKi)/RangeKi*Num);
sKd = floor((Kd00-LBKd)/RangeKd*Num);

RecKp = [max(1,sKp-delta), max(1,sKp), min(Num,sKp+delta)];
RecKi = [max(1,sKi-delta), max(1,sKi), min(Num,sKi+delta)];
RecKd = [max(1,sKd-delta), max(1,sKd), min(Num,sKd+delta)];

XKp = msf(RecKp);
XKi = msf(RecKi);
XKd = msf(RecKd);

for i = 1:Nset,
        MOrecd(i,:) = [max(1, sKp-Width(i)), max(1, sKp),
                min(Num, sKp+Width(i))];
        MO(i,:) = msf(MOrecd(i,:));
        MOrecd(i+Nset,:) = [max(1, sKi-Width(i)), max(1, sKi),
                min(Num, sKi+Width(i))];
        MO(i+Nset,:) = msf(MOrecd(i+Nset,:));
        MOrecd(i+Nset*2,:) = [max(1, sKd-Width(i)), max(1, sKd),
                min(Num, sKd+Width(i))];
```



```
        MO(i+Nset*2,:) = msf(MOrecd(i+Nset*2,:));
end;


D = general(MI);
                                  % fuzzy input-output mapping
[FuzKp] = Fuzeval(MI,MO(1:Nset,:),D);
[FuzKi] = Fuzeval(MI,MO(Nset+1:2*Nset,:),D);
[FuzKd] = Fuzeval(MI,MO(Nset*2+1:3*Nset,:),D);

fuzKi = FuzKi;   fuzKp = FuzKp;   fuzKd = FuzKd;
[y,r,t] = simuD;  Y=y;  R=r;
[J,STATUS_min, BASE] = measure(y,r,t)

STATUS_original = STATUS_min;

figure(1);  clf;  hold on;

subplot(2,3,4);  hold on;
title('Error membership function');
for i = 1:Nset,
   plot(MI(i,:));
end;

subplot(2,3,1); hold on;
title('Kp membership function');
for i = 1:Nset,
    plot(MO(i,:));
end;

subplot(2,3,2); hold on;
title('Ki membership function');
for i = (Nset+1):Nset*2,
   plot(MO(i,:));
end;

subplot(2,3,3); hold on;
title('Kd membership function');
for i = (Nset*2+1):Nset*3,
   plot(MO(i,:));
end;

subplot(2,3,5); hold on;
plot(FuzKp);
plot(FuzKi,':');
plot(FuzKd,'-.');
axis([0 32 0 1]);
title('E--K_p,i,d relationship');
xlabel('Error(Normalized)');
ylabel('K_p,i,d');
text(12, 0.85, '-- : Kp');
text(12, 0.7,  '...... : Ki');
text(12, 0.55, '-.- : Kd');

subplot(2,3,6); hold on;
title('Initial system response');
plot(t, y, 'y');
plot(t, r, ':');
grid on;
title('Initial system response');
xlabel('t(sec)');
ylabel('system response');
text(12, 0.5, '-- : y');
text(12, 0.3, '...... : r');


figure(6);  clf; hold on;
plot(t,y, 'c');
plot(t,r, ':');
```

1

```
%****************************************************************%
%   Functionality: Successive refinement tuning rulebase        %
%   Input parameters:                                           %
%                Trial -- total perturbation trials             %
%                sN -- perturbation step control factor         %
%                MINCH -- minimum change(shift) size            %
%                SMOOTH -- variable range adjusting option      %
%                        1/0: adjust(not) variable range        %
%                SC -- SA acceptance Probability factor         %
%****************************************************************%

function NaD(Trial,sN,MINCH,SMOOTH,SC)

global fuzKi fuzKp fuzKd Num Nset delta FuzKp FuzKi FuzKd
global MI MIrecd MO MOrecd J Y R Probrule STATUS_min
global BASE PENALTY PID_select STATUS_goal D Width

step = floor(Num/sN);       D = general(MI);

for i = 1:Trial,                              % iteration
      if SMOOTH,  seconderun2;   Satisfied;  end;
                                              % variable range adjustment
      if sum(BASE)==0 ,                       % control requirement
         [E] = Satisfied                      % satisfaction check
          if E,  break, end;                       % E = 1 -- satisfied
         BASE = STATUS_min > STATUS_goal;     % 0 -- unsatisfied
         J = J + sum(STATUS_min./STATUS_goal.*BASE) * PENALTY;
      end;

      if E, break, end;

      xx = rand;                        % rule perturb selection
      offsetrule = (xx>PID_select(1)) + (xx>PID_select(2));
      Rule = ruleselect(Y,R) + offsetrule*Nset

      Prule = Probrule(Rule);

      AddMinus = sign(rand-Prule);
      if AddMinus == 0,    AddMinus = 1;        end;

      trial_add_minus = 1;          % shift direction selection
      while trial_add_minus < 3,
          Changing = 1;   XX = 0; % construct a new rulebase
          while Changing & XX <10,     % which has enough difference
              XX = XX + 1;          % compared with the original one
              middle = floor(min(max(1,(MOrecd(Rule,2)
                     +AddMinus*(max(step*rand,MINCH)))),Num));
              shift = middle - MOrecd(Rule,2);
              jj = rem(Rule,Nset);
              if (jj == 0), jj=Nset; end;
              dW = Width(jj);
              mor = [max(1,middle-dW), middle, min(Num, middle+dW)];
              mo = MO;
              mo(Rule,:) = msf(mor);

              if Rule<=Nset,
                 fuzKp = Fuzeval(MI,mo(1:Nset,:),D);
                 if (max(abs(FuzKp-fuzKp))>0.01),  Changing = 0;  end;
              elseif Rule <= Nset*2,
                 fuzKi = Fuzeval(MI,mo((Nset+1):(2*Nset),:),D);
                 if (max(abs(FuzKi-fuzKi)>0.01)),  Changing = 0;  end;
              else
                 fuzKd = Fuzeval(MI,mo((Nset*2+1):(3*Nset),:),D);
                 if (max(abs(FuzKd-fuzKd)>0.01)),  Changing = 0;  end;
              end;
          end;

          figure(1);  clf;  hold on;     % plot the old and new mapping clf;
          subplot(1,3,1);  hold on;  plot(fuzKp);  plot(FuzKp,':');
                             axis([0 40 0 1]);  title('E--Kp');
          subplot(1,3,2);  hold on;  plot(fuzKi);  plot(FuzKi,':');
                             axis([0 40 0 1]);  title('E--Ki');
          subplot(1,3,3);  hold on;  plot(fuzKd);  plot(FuzKd,':');
```



```
                             axis([0 40 0 1]);  title('E--Kd');

          [y,r,t] = simuD;
          [j,s,BASE] = measure(y,r,t)

          figure(4);  clf;  hold on;  plot(t, y); plot(t, r, ':');
          title('system response')

          if sum(BASE)==0,                    % control requirement
            STATUS_min = s;                   % satisfaction check
            [E] = Satisfied
            if E, break, end;
            BASE = STATUS_min > STATUS_goal;
            J = J + sum(STATUS_min./STATUS_goal.*BASE) * PENALTY + 1;
            j = J - 1;
          end;
                                  % new setting acception/rejection
          trial_add_minus = trial_add_minus + 1 + (j<J);
          CT = 0;
          while ((j<J)|(rand<exp(-i*SC)))&(mor(2)>=1)&(mor(2)<=Num)&(CT<5),
              CT = CT + 1;          % accept new setting
              MO = mo;
              MOrecd(Rule,:) = mor;
              J = j
              STATUS_min = s;
              FuzKp = fuzKp;  FuzKi = fuzKi;  FuzKd = fuzKd;  R=r; Y=y;

              Changing = 1; XX = 0;                 % successive refinement
              while Changing & XX<10,     % reapply the same change
               XX = XX + 1;
               middle = middle + shift;
               mor = [max(1,middle-dW), middle, min(Num, middle+dW)];
               mo = MO;
               mo(Rule,:) = msf(mor);

               if Rule<=Nset,
                 fuzKp = Fuzeval(MI,mo(1:Nset,:),D);
                 if (max(abs(FuzKp-fuzKp))>0.01),  Changing = 0;  end;
               elseif Rule<=Nset*2,
                 fuzKi = Fuzeval(MI,mo((Nset+1):(2*Nset),:),D);
                 if (max(abs(FuzKi-fuzKi)>0.01)),  Changing = 0;  end;
               else
                 fuzKd = Fuzeval(MI,mo((Nset*2+1):(3*Nset),:),D);
                 if (max(abs(FuzKd-fuzKd)>0.01)),  Changing = 0;  end;
               end;
             end;

             [y,r,t] = simuD;
             [j,s,BASE] = measure(y,r,t)

             if sum(BASE)==0,
                STATUS_min = s;
                [E] = Satisfied
                if E,
                    FuzKp = fuzKp;  FuzKi = fuzKi;  FuzKd = fuzKd;  break;
                end;
                BASE = STATUS_min > STATUS_goal;
                J = J + sum(STATUS_min./STATUS_goal.*BASE) * PENALTY - 1;
                j = J + 1;
              end;
           end;

          if j<J,          % update the tuning historical record
                Probrule(Rule) = Prule/2 - 1/2*sign(AddMinus-1);
          else    Probrule(Rule) = Prule/2 + 1/2*sign(AddMinus+1);
          end;

          if trial_add_minus == 2,       % for bad perturbation,
            AddMinus = -AddMinus; % next trial apply the opposite
          end;                          % direction.
      end;
end;
```

Jun 11 1993 17:10:19 ini.m Page 1

```
%****************************************%
%    simulation parameter setting        %
%****************************************%

tstart = 0;
minstep = 1e-4;
maxstep = 10;
tol = 1e-3;
arisetime = 0.01;
```

```
%*****************************************************%
%    Functionality:  final(FGOAL) and                 %
%                    initial(STATUS_goal)             %
%                    control requirement specification %
%*****************************************************%

function main

global  OVERSHOOT RISETIME ERROR_SS SETTLETIME UNDERSHOOT
global  FGOAL STATUS_goal

OVERSHOOT = 0.3;                    % plant 1
UNDERSHOOT = 0.3;
RISETIME = 40;
ERROR_SS = 0.10;
SETTLETIME = 110;

FOVERSHOOT = 0.02;
FUNDERSHOOT = 0.02;
FRISETIME = 22;
FERROR_SS = 0.005;
FSETTLETIME = 50;

%OVERSHOOT = 0.3;                   % plant 2, 3
%UNDERSHOOT = 0.3;
%RISETIME = 3;
%ERROR_SS = 0.1;
%SETTLETIME = 15;

%FOVERSHOOT = 0.01;
%FUNDERSHOOT = 0.01;
%FRISETIME = 2.5;
%FERROR_SS = 0.005;
%FSETTLETIME = 6;

%OVERSHOOT = 0.3;                          % nonlinear 4,5
%UNDERSHOOT = 0.3;
%RISETIME = 5;
%ERROR_SS = 0.05;
%SETTLETIME = 10;

%FOVERSHOOT = 0.05;
%FUNDERSHOOT = 0.05;
%FRISETIME = 2;
%FERROR_SS = 0.005;
%FSETTLETIME = 6;

FGOAL = [FOVERSHOOT, FUNDERSHOOT, FRISETIME, FERROR_SS, FSETTLETIME];
STATUS_goal = [OVERSHOOT, UNDERSHOOT, RISETIME, ERROR_SS, SETTLETIME]
```

Jun 11 1993 17:10:19 **msf.m** Page 1

```
%**************************************************%
%    Functionality: generate membership function    %
%           for a trianglur ship fuzzy set          %
%           from the Shape discription              %
%           by its top value(Shape(2))              %
%           two bottom points(Shape(1), Shape(3))   %
%**************************************************%

function (membershipf) = msf(Shape)

global Nset Num

Shape = min(max(Shape,1), Num);
n1 = Shape(1);
n2 = Shape(2);
n3 = Shape(3);

membershipf(1:Num) = zeros(1,Num);

if n1 == 1 & n2<Num/2/Nset,
     membershipf(1:n2) = ones(1,n2);
elseif  n1 ~= n2,
     step1 = 1/(n2-n1);
     membershipf(n1:n2) = 0:step1:1;
end;

if n3 == Num & n2+Num/2/Nset>Num
     membershipf(n2:n3) = ones(1,n3-n2+1);
elseif n2 ~= n3,
     step2 = -1/(n3-n2);
     membershipf(n2:n3) = 1:step2:0;
end;
```

1

```
%*************************************************************%
%    Functionality: calculates the mapping provided by fuzzy  %
%                rulebase (one input, one output).            %
%                For each input value, a crisp output is      %
%                generated based on the fuzzy rules.          %
%    Method used: Centroid calculation.                       %
%    Input parameters:  MI -- Input fuzzy sets.               %
%                       MO -- Output fuzzy sets.              %
%                       D  -- The fuzzified represention of   %
%                       the input domain.                     %
%    Output: FuzOutput -- output corresponds to different     %
%                         inputs.                             %
%    All the above parameters are in vector forms.            %
%*************************************************************%

function [FuzOutput] = Fuzeval(MI,MO,D)

global Nset Num         % Nset is the fuzzy set number.
                        % Num is the number of discrete value
                        %      that represents each fuzzy set.

FuzOutput = zeros(1,Num);

for i = 1:Num,
    FO = zeros(1,Num);
    for j = 1:Nset,
        x = min(D(j,i),MO(j,:));
        FO = max(FO, x);
    end;
    id = 1:Num;
    if sum(FO) == 0,
       FuzOutput(i)  = 0;
    else
       FuzOutput(i) = sum(id.*FO)/sum(FO)/Num;
    end;
end;


for i = 1:Num,                    % Smooth action
       FuzOutput(i) = mean(FuzOutput(max(1,i-1):min(Num, i+1)));
end;
```

```
%*******************************************************%
%   Functionality:  Find the fuzzy representation for   %
%                   all the values in input domain      %
%*******************************************************%

function [D] = general(MI);

global Nset Num
D = zeros(Nset,Num);

for i = 1:Nset
  for j = 1:Num
    a = msf([max(0,j-2), j, min(Num,j+2)]);
    m = min(a, MI(i,:));
    D(i,j) = max(m);
  end;
end;
```

1

Jun 11 1993 17:10:19 ruleselect.m Page 1

```
%******************************************************%
%    Functionality: generate group selection          %
%                   (1 out out Nset).                 %
%    Input parameters: y -- system output             %
%                      r -- reference signal          %
%******************************************************%

function [Rule] = ruleselect(y,r)

global LBe UBe Nset Num Probrule BASE Overshoot STATUS_min

Error = max(min(r-y,UBe), LBe);
E = [Error; LBe; UBe];

[Eh, ruleset] = hist(E, Num);        % histogram of E(r-y)

B = zeros(1,Nset);

B(1) = sum(Eh(1:8));              % divied input domain
B(2) = sum(Eh(9:14));
B(3) = sum(Eh(15:18));
B(4) = sum(Eh(19:24));
B(5) = sum(Eh(25:32));
                                  % decrease the prob.
                                  % of "ZERO" group
second_large = max([B(1), B(2), B(4), B(5)])
while second_large < B(3)*0.5,
       B(3) = B(3)*0.7;
end;

bottom = 0.05;               % provide small prob. for each rule
B = B + bottom;
SumB = sum(B);

B = B/SumB;

if BASE(5)==1,               % system performance modification
              Base = [1 1.5 2 1.5 1];
elseif BASE == [0 0 1 0 0],
              Base = [1 1 1 3 1.5];
elseif BASE(1)==1 & STATUS_min(1)>0.15,
              Base = [1.5 3 1 1.5 1];
elseif BASE(2)==1 & STATUS_min(2)>0.15,
              Base = [1 1.5 1 3 1.5];
else          Base = [1 1 1 1 1];
end;

h = B.* Base;
B = h/sum(h);

Rule = 0;
Upperbound = 0;

Randrule = rand;             % roulette-wheel-selection
while Randrule>Upperbound & Rule<Nset;
       Rule = Rule + 1;
       Upperbound = Upperbound + B(Rule);
end;

if Rule == 3,                % decrease the prob. of
       x = rand;             % "ZERO" rule selection
       Rule = Rule + (x>0.85);
       Rule = Rule - (x<0.15);
end;

adjustrule = rand;
Rule = max(1, Rule-(rand<0.1));
Rule = min(Nset, Rule+(rand>0.9));

figure(2);
clf; hold on;
bar(B);
```

1

```
%**************************************************%
%   Functionality:  simulation test                %
%   Return variable:  y -- system response         %
%                     r -- reference signal        %
%                     t -- time                    %
%**************************************************%

function [y,r,t] = simuD

global Kp Ki Kd arisetime Total_time

Total_time = 20; arisetime = 0.01;

figure(5);
%linsim('refinep0',Total_time, [], [1e-2, 0.05, 0.05]);
linsim('refinep0',Total_time);

figure(4);
clf;
subplot(1,2,1); hold on;  plot(t,y,'c');  plot(t,r,'m');
subplot(1,2,2); hold on;  plot(t,y,'c');  plot(t,s,'g');
```


1

Jun 11 1993 17:10:19 **measure.m** Page 1

```
%***************************************************************%
%   Functionality: evaluation system performance               %
%                  calculate cost function C,                   %
%                  satisfaction status STATUS and BASE          %
%   Input parameters:   signal -- system response               %
%                       reference -- reference signal           %
%                       t -- time                               %
%***************************************************************%

function [C, STATUS, BASE] = measure(signal, reference, t)

global Total_time OVERSHOOT RISETIME ERROR_SS SETTLETIME UNDERSHOOT
global STATUS_min STATUS_goal PENALTY STATUS NOISE PID_select

PENALTY = 1000*Total_time;

[M,N] = size(t);     M = max(M,N);

reference = reference * 0.9998;  stepindex = 1;  C = 0;

for currenttime = 0:0.05:Total_time                % ITAE calculation
    while (t(stepindex)<currenttime)&(stepindex<M),
        stepindex = stepindex + 1;
    end;
    error = abs(reference(stepindex)-signal(stepindex));
    C = C + error*currenttime;
end;
                  % settling time, steady_state-error calculation
[S_time, S_index, Ess] = settle(signal, reference, t);

                  % overshoot, rise time, undershoot calculation
[Overshoot, Risetime, Undershoot] = overshoot(signal, reference, t);

C = C + Overshoot*100 + Risetime*100;     % cost function synthesize

STATUS = [Overshoot, Undershoot, Risetime, Ess, S_time]

BASE = STATUS>STATUS_goal;

X = sum(STATUS ./ STATUS_goal .* BASE) * PENALTY ;

C = C + X;
                             % provide PID selection probability
if sign(abs(Ess)>ERROR_SS)  |  sign(S_time*0.9>SETTLETIME),
       PID_select = [0.3, 0.8];
elseif (sign(Overshoot>OVERSHOOT)&(Overshoot>0.15))
       | (sign(Undershoot>UNDERSHOOT) & (Undershoot>0.15))
       | (sign(S_time>SETTLETIME)),
              PID_select = [0.5, 0.75];
elseif sign(Risetime>RISETIME),
              PID_select = [0.65,0.75];
else          PID_select = [0.4, 0.8];
end;
```

Jun 11 1993 17:10:19 **overshoot.m** **Page 1**

```
%*********************************************************************%
%   Functionality: Calculate overshoot, risetime, undershoot given    %
%                 the system response and reference signal            %
%*********************************************************************%

function [Overshoot, Risetime, Undershoot] = overshoot(signal,reference,t)

global arisetime Total_time

[M,N] = size(t);  M = max(M,N);
error = signal - reference;

if max(error(t>arisetime)) > 0,
        [P_error,P_index] = max(error)
         REF = reference(P_index);
         if REF == 0,
                REF = 1;
        end;
        Overshoot = P_error/REF;
        Start = t(t<arisetime);
        [a,b] = size(Start);  start_index = max(a,b);
         Rising = abs(signal(start_index:P_index) - 0.9*REF);
         [j, R_index] = min(Rising)
         RiseTime = t(R_index+start_index);
else     P_index = 1;
        while reference(P_index)==0,
                P_index = P_index + 1;
        end;
        while error(P_index)<error(P_index+1) & P_index<M-1,
                P_index = P_index + 1;
        end;
        P_v = signal(P_index);
        Overshoot = abs(error(P_index))/reference(P_index)*2;
        x = signal - 0.9*reference(P_index);
        if max(x)>0,
                [xx,R_index] = min(abs(x(x>0)));
                 Risetime = t(R_index);
        else     Risetime = t(P_index);
        end;
end;

P_index = 1;
while (signal(P_index)<= signal(P_index+1) | reference(P_index)==0)
        & P_index < M-1  ,
        P_index = P_index + 1;
end;
Undershoot = abs(min(error(P_index:M)))/reference(P_index)
```

```
%*******************************************************************%
%   Functionality: Calculate settling time, steady_state error given  %
%                  system response and reference signal                %
%*******************************************************************%

function [S_time, S_index, Ess] = settle(signal, reference, t)

global Total_time ERROR_SS

[M,N] = size(t);
error = abs(signal - reference);

S_index = M;
while(S_index>2 & abs(error(S_index-1)-error(M))<ERROR_SS),
        S_index = S_index-1;
end;

S_index = min(M, S_index + 1);

S_time = t(S_index);

Ess = mean(error(S_index:M));
```

Jun 11 1993 17:10:20 **Satisfied.m** Page 1

```
%**********************************************************%
%   Functionality: update control requirement upon         %
%                  satisfaction and check if final         %
%                  specification is reached.               %
%   Return variable:                                       %
%         Ending == 1: final requirements are reached      %
%                   0: final requirements are not reached  %
%**********************************************************%

function [Ending] = Satisfied

global STATUS_min STATUS_goal STATUS_original FGOAL
global  OVERSHOOT RISETIME ERROR_SS SETTLETIME UNDERSHOOT BASE

if sum(BASE) ==0 & sum(STATUS_min<FGOAL)==5,
       Ending = 1;
else   Ending = 0;
end;

reduce_factor = 0.9;

if sum(BASE) == 0 & Ending == 0,
  while sum(STATUS_min<STATUS_goal)==5,
       STATUS_goal = max(FGOAL, STATUS_goal * reduce_factor);
  end;

  STATUS_goal

  OVERSHOOT = STATUS_goal(1);
  UNDERSHOOT = STATUS_goal(2);
  RISETIME = STATUS_goal(3);
  ERROR_SS = STATUS_goal(4);
  SETTLETIME = STATUS_goal(5);
end;
```

```
%***********************************************************%
%    Functionality:  update control requirement             %
%                    check if final specification is reached %
%***********************************************************%

function [Ending] = Satisfied1

global STATUS_min STATUS_goal STATUS_original FGOAL
global  OVERSHOOT RISETIME ERROR_SS SETTLETIME UNDERSHOOT BASE

a = STATUS_min>STATUS_goal;

if sum(BASE) ==0 & sum(STATUS_min<FGOAL)==5,
        Ending = 1;
else    Ending = 0;
end;

reduce_factor = 0.9;

b = 1 - a;
STATUS_goal(b) = max(FGOAL(b), STATUS_goal(b) * reduce_factor);

OVERSHOOT = STATUS_goal(1);
UNDERSHOOT = STATUS_goal(2);
RISETIME = STATUS_goal(3);
ERROR_SS = STATUS_goal(4);
SETTLETIME = STATUS_goal(5);
```

```
%****************************************%
%    Functionality: Smooth curve action   %
%****************************************%

function sec

global STATUS_min J

JJ = J;
i = 0;
seconderun20;
while J<JJ & i<5,
        JJ = J;
        seconderun20;
        i = i + 1;
end;
```

```
%*******************************************************************%
%    Functionality: adjust output variable range if the upper or    %
%                   lower limit is reached                          %
%*******************************************************************%

function secondrun2

global fuzKp fuzKi fuzKd FuzKp FuzKi FuzKd
global LBe UBe LBKp UBKp LBKi UBKi LBKd UBKd
global RangeKp RangeKi RangeKd RangeE Width
global Num Nset MI MO MOrecd J delta  Y R Probrule s
global OVERSHOOT RISETIME ERROR_SS SETTLETIME UNDERSHOOT
global FOVERSHOOT FRISETIME FERROR_SS FSETTLETIME FUNDERSHOOT
global STATUS_min STATUS_goal D BASE

UpdateKp = max(FuzKp)>0.9 | min(FuzKp)<0.1;
UpdateKi = max(FuzKi)>0.9 | min(FuzKi)<0.1;
UpdateKd = max(FuzKd)>0.9 | min(FuzKd)<0.1;

if UpdateKp | UpdateKi | UpdateKd,
   if UpdateKp,
      if (max(FuzKp)-min(FuzKp)) > 0.5,
              Probrule(1:Nset) = 0.5*ones(Nset,1);
      end;
      newRangeKp = max(0.3,(max(FuzKp) - min(FuzKp)))*3;
      LBKp = (min(FuzKp) - newRangeKp/3)*RangeKp + LBKp;
      UBKp = LBKp + newRangeKp*RangeKp;
      RangeKp = newRangeKp*RangeKp;

    for i = 1:Nset,
      Kp2 = mean(FuzKp(max(1,floor((i-1)*Num/Nset)):floor(i*Num/Nset)));
      sKp = floor((Kp2 - min(FuzKp) + newRangeKp/3)/newRangeKp*Num);
      RecKp = [max(1,sKp-Width(i)), max(1,sKp), min(Num,sKp+Width(i))];
      MO(i,:) = msf(RecKp);
      MOrecd(i,:) = RecKp;
    end;

    [FuzKp] = Fuzeval(MI,MO(1:Nset,:),D);
    fuzKp = FuzKp;
  end;

  if UpdateKi,
      if (max(FuzKi)-min(FuzKi)) > 0.5,
                Probrule(1+Nset:Nset*2) = 0.5*ones(Nset,1);
      end;
      newRangeKi = max(0.2,(max(FuzKi) - min(FuzKi)))*3;
      LBKi = (min(FuzKi) - newRangeKi/3)*RangeKi + LBKi;
      UBKi = LBKi + newRangeKi*RangeKi;
      RangeKi =  newRangeKi*RangeKi;

    for i = 1:Nset,
      Ki2 = mean(FuzKi(max(1,floor((i-1)*Num/Nset)):floor(i*Num/Nset)));
      sKi = floor((Ki2 - min(FuzKi) + newRangeKi/3)/newRangeKi*Num);
      RecKi = [max(1,sKi-Width(i)), max(1,sKi), min(Num,sKi+Width(i))];
      MO(i+Nset,:) = msf(RecKi);
      MOrecd(i+Nset,:) = RecKi;
    end;

    [FuzKi] = Fuzeval(MI,MO(Nset+1:2*Nset,:),D);
    fuzKi = FuzKi;
  end;

  if UpdateKd,
      if (max(FuzKd)-min(FuzKd)) > 0.5,
              Probrule(2*Nset+1:3*Nset) = 0.5*ones(Nset,1);
      end;
      newRangeKd = max(0.2,(max(FuzKd) - min(FuzKd)))*3;
      LBKd = (min(FuzKd) - newRangeKd/3)*RangeKd + LBKd;
      UBKd = LBKd + newRangeKd*RangeKd;
      RangeKd = newRangeKd*RangeKd;

    for i = 1:Nset,
      Kd2 = mean(FuzKd(max(1,floor((i-1)*Num/Nset)):floor(i*Num/Nset)));
```



```
      sKd = floor((Kd2 - min(FuzKd) + newRangeKd/3)/newRangeKd*Num);
      RecKd = [max(1,sKd-Width(i)), max(1,sKd), min(Num,sKd+Width(i))];
      MO(i+2*Nset,:) = msf(RecKd);
      MOrecd(i+2*Nset,:) = RecKd;
    end;

    [FuzKd] = Fuzeval(MI,MO(2*Nset+1:3*Nset,:),D);
    fuzKd = FuzKd;
  end;

  [y,r,t] = simuD;  Y=y;  R=r;
  [J,STATUS_min, BASE] = measure(y,r,t)

  figure(1);  clf;  hold on;
  for i = 1:Nset,
      plot(MI(i,:));
  end;

  figure(2);  clf;  hold on;
  for i = 1:Nset,
      plot(MO(i,:));
  end;

  figure(3);  clf;  hold on;
  for i = (Nset+1):Nset*2,
      plot(MO(i,:));
   end;

  figure(4);  clf;  hold on;
  for i = (Nset*2+1):Nset*3,
      plot(MO(i,:));
  end;

  figure(5);  clf;  hold on;
  plot(FuzKp);  plot(FuzKi,':');  plot(FuzKd,'-.')

end;
```

1

```
%**************************************************************%
%    Functionality: adjust variable range, smooth mapping curve    %
%**************************************************************%

function secondrun20

global fuzKp fuzKi fuzKd FuzKp FuzKi FuzKd
global LBe UBe LBKp UBKp LBKi UBKi LBKd UBKd
global RangeKp RangeKi RangeKd RangeE Width
global Num Nset MI MO MOrecd J delta  Y R Probrule s
global  OVERSHOOT RISETIME ERROR_SS SETTLETIME UNDERSHOOT STATUS_min
global  FOVERSHOOT FRISETIME FERROR_SS FSETTLETIME FUNDERSHOOT BASE

        newRangeKp = max(0.4,(max(FuzKp) - min(FuzKp)))*2;
        LBKp = (min(FuzKp) - newRangeKp/4)*RangeKp + LBKp;
        UBKp = LBKp + newRangeKp*RangeKp;
        RangeKp = newRangeKp*RangeKp;

        newRangeKi = max(0.3,(max(FuzKi) - min(FuzKi)))*2;
        LBKi = (min(FuzKi) - newRangeKi/4)*RangeKi + LBKi;
        UBKi = LBKi + newRangeKi*RangeKi;
        RangeKi = newRangeKi*RangeKi;

        newRangeKd = max(0.3,(max(FuzKd) - min(FuzKd)))*2;
        LBKd = (min(FuzKd) - newRangeKd/4)*RangeKd + LBKd;
        UBKd = LBKd + newRangeKd*RangeKd;
        RangeKd = newRangeKd*RangeKd;

for i = 1:Nset,
        Kp2 = mean(FuzKp(max(1,floor((i-1)*Num/Nset)):floor(i*Num/Nset)));
        sKp = floor((Kp2 - min(FuzKp) + newRangeKp/4)/newRangeKp*Num);
        RecKp = [max(1,sKp-Width(i)), max(1,sKp), min(Num,sKp+Width(i))];
        MO(i,:) = msf(RecKp);
        MOrecd(i,:) = RecKp;

        Ki2 = mean(FuzKi(max(1,floor((i-1)*Num/Nset)):floor(i*Num/Nset)));
        sKi = floor((Ki2 - min(FuzKi) + newRangeKi/4)/newRangeKi*Num);
        RecKi = [max(1,sKi-Width(i)), max(1,sKi), min(Num,sKi+Width(i))];
        MO(i+Nset,:) = msf(RecKi);
        MOrecd(i+Nset,:) = RecKi;

        Kd2 = mean(FuzKd(max(1,floor((i-1)*Num/Nset)):floor(i*Num/Nset)));
        sKd = floor((Kd2 - min(FuzKd) + newRangeKd/4)/newRangeKd*Num);
        RecKd = [max(1,sKd-Width(i)), max(1,sKd), min(Num,sKd+Width(i))];
        MO(i+2*Nset,:) = msf(RecKd);
        MOrecd(i+2*Nset,:) = RecKd;
end;

Probrule = 0.5*ones(Nset*3,1);

D = general(MI);
[FuzKp] = Fuzeval(MI,MO(1:Nset,:),D);
[FuzKi] = Fuzeval(MI,MO(Nset+1:2*Nset,:),D);
[FuzKd] = Fuzeval(MI,MO(2*Nset+1:3*Nset,:),D);

fuzKi = FuzKi;  fuzKp = FuzKp;  fuzKd = FuzKd;
[y,r,t] = simuD;  Y=y;  R=r;
[J,STATUS_min, BASE] = measure(y,r,t)

figure(1);  clf;  hold on;

subplot(2,3,4);  hold on;
title('E membership function');
for i = 1:Nset,
   plot(MI(i,:));
end;

subplot(2,3,1);  hold on;
title('Kp membership function');
for i = 1:Nset,
   plot(MO(i,:));
end;

```



```
subplot(2,3,2);  hold on;
title('Ki membership function');
for i = (Nset+1):Nset*2,
   plot(MO(i,:));
end;

subplot(2,3,3);  hold on;
title('Kd membership function');
for i = (Nset*2+1):Nset*3,
   plot(MO(i,:));
end;

subplot(2,3,5);  hold on;
title('E--K_p,i,d relationship');
plot(FuzKp, 'g');
plot(FuzKi, 'c');
plot(FuzKd, 'm');
axis([0 40 0 1]);
```

```
%***************************************************************%
%    Functionality: Find optimal PID settings                   %
%    Method used: Simulated Annealing                           %
%    Input parameters:                                          %
%       Kp00 -- initial Kp                                      %
%       Ki00 -- initial Ki                                      %
%       Kd00 -- initial Kd                                      %
%       Trial -- maximum trials                                 %
%       cc -- control factor start value                        %
%       ccspeed -- control factor change speed control          %
%       acceptfact -- acceptance prob.                          %
%***************************************************************%

function [KpT, KiT, KdT] = SApid(Kp00,Ki00,Kd00,Trial,cc,ccspeed)

global K                          % PID setting for simulation

K = zeros(3,1);                                     % initialization
K(1) = Kp00;  K(2) = Ki00;  K(3) = Kd00;
Kopt = zeros(3,1);  Kopt = K;
Probrule = [0 1/6, 2/6, 3/6, 4/6. 5/6, 1];

[y,r,t] = simuD;  Y=y;  R=r;               % simulation
[J] = measure(y,r,t)

KPID = [Kp00, Ki00, Kd00];

i = 0;

change = 1;  change_tol = 0.001;
while (change>change_tol) & (i<Trial),
        i = i + 1

        x = rand;  rule = 1;               % perturbation
        while x>Probrule(rule+1),          % find new K
                rule = rule + 1;
        end;

        addminus = rem(rule, 2);
        if (addminus == 0),
                addminus = -1;
        end;
        change = cc * exp(-i*ccspeed);
        changeR = change * rand;
        K = Kopt;
        PID = floor((rule-1)/2)+1
        K(PID) = K(PID) + addminus * changeR;

        KPID = [KPID; K'];

        [y,r,t] = simuD;  Y=y;  R=r;       % simulation for new K
        [j] = measure(y,r,t);

        if j>J,                    % update PID selection prob.
                shrink = (Probrule(rule+1) - Probrule(rule))/3;
                Probrule((rule+1):7) = max(0,Probrule((rule+1):7) - shrink);
        else    expand = min(0.3, (Probrule(rule+1) - Probrule(rule))*1.5);
                Probrule((rule+1):7) = Probrule((rule+1):7) + expand;
        end;
        Probrule = Probrule + [0 0.1 0.2 0.3 0.4 0.5 0.6];
        Probrule = Probrule/Probrule(7);

        if j<J | rand < exp(-i*acceptfact),       % accept/reject new K
                J = j;
                Kopt = K;
        end;

end;


figure(1); clf;  [m,n] = size(KPID);  KPID
i = 1:1:m;
plot3(KPID(i,1), KPID(i,2), KPID(i,3), '*');
```

```
%****************************************************************%
%    Functionality:       PID control for fuzzy-PID structure    %
%    Input parameter:     in                                     %
%                         in(1) -- system response               %
%                         in(2) -- reference signal              %
%****************************************************************%

function [y] = KPIDtuning(in)

global Kp Ki Kd fuzKp fuzKi fuzKd Num Nset
global LBe RangeE RangeKp RangeKi RangeKd LBKp LBKi LBKd

e = in(1);    sysout = -in(2);
r = e - sysout;  b = 0.85;

Kp = fuzKp(min(Num,max(1,floor((e-LBe)/RangeE*Num))))*RangeKp+LBKp;
Ki = fuzKi(min(Num,max(1,floor((e-LBe)/RangeE*Num))))*RangeKi+LBKi;
Kd = fuzKi(min(Num,max(1,floor((e-LBe)/RangeE*Num))))*RangeKd+LBKd;;

y = [Ki*e; Kp*(b*r+sysout); Kd*sysout];
```

Jun 11 1993 17:10:20 **KPIDpp.m** Page 1

```
%*****************************************%
%    Functionality: modified PID control    %
%               (traditional, fix gains)    %
%    Input parameters:  in                  %
%               in(1) -- error              %
%               in(2) -- system response    %
%*****************************************%

function [y] = KPIDpp(in)

global K

e = in(1);   sysout = -in(2);
r = e - sysout;  b = 0.85;

y = [K(2)*e; K(1)*(b*r+sysout);  K(3)*sysout];
```

VITA

Yongmei Wang was born in Beijing, China, on June 27, 1970. She finished her undergradate study at Beijing University, majoring in Radio-Electronics. She enrolled in the department of Electrical and Computer Engineering, University of Tennessee, Knoxville in August, 1991 for her M.S. study. She worked as a research assistant at the same time. In the first half of her graduate study, her work was concentrated in the area of image processing and pattern recognition. During the second half, her work was concentrated on system control and artificial intelligence.