Fuzzy Sets and Systems 80 (1996) 295–313

A self-organizing adaptive fuzzy controller

Chien-Hui Lee, Sheng-De Wang *

Department of Electrical Engineering, EE Building, Rm. 441, National Taiwan University, Taipei 106, Taiwan

Received June 1994; revised May 1995

Abstract

There are two main parts in this paper. The first part presents a knowledge representation and reasoning scheme, called tree-searched neural networks (TNN). The TNN is based on a well-known intuitive knowledge representation (IKR) and can reduce the number of the processing nodes in the neural networks. The second part proposes a self-organizing adaptive fuzzy controller (SOAFC) based on the TNN model. It can help acquire control knowledge and thus can reduce the dependence on experts. Furthermore, designers do not need to predefine all membership functions to cover whole input space domain. For improving its performance further, we design a D-controller which is included within the SOAFC. Whether the fuzzy controller is incorporated with the D-controller or not, it is also guaranteed to be globally stable. Simulation results show that this approach has faster convergence speed, results in better transient response, and in addition requires less total control energy.

Keywords: Fuzzy sets; Neural networks; Process control; Adaptive controller

1. Introduction

Fuzzy logic controllers (FLCs) are used more and more generally and they have been shown to be specially effective in the control of mathematically ill-understood processes. FLCs have several significant advantages over conventional control approaches such as *robustness* and conformability to the linguistic rules. The rules in these FLCs normally come from experienced human operators or experts. But control concepts and operators' experience are generally difficult to be completely incorporated into the rule base. This could be a main disadvantage of FLCs. Another drawback is lack of mathematical proof of stability for the fuzzy systems. This may be the reason why fuzzy control has not been accepted in some area of applications. Recently there are many papers discussing this issue [16, 17, 26]. During the past several years, a lot of algorithms for developing *self-organizing* or *self-learning* fuzzy systems have been proposed [4, 13, 19, 20, 23].

Fuzzy rule representations are proposed in many papers [5, 24, 25], but few are aimed to be suitable for hardware implementation. Some representations are based on neural networks and require too many processing units. This drawback not only destroys the applicability of fuzzy systems but also incurs large cost for hardware implementation. In this paper, we shall combine a tree structure and neural networks as its database and inference engine, respectively. Thus the neural networks will not need so many processing units.

Basically, fuzzy logic reasoning can be viewed as one kind of combination of the complete or partial

* Corresponding author.

matching rules. If the distances between the rules are small, the performance will be found better as compared with the sparse rule base [6]. Also for improving the performance further, more rules should be explored for more critical situations. A technique for generating rules is described in [25]. It needs I/O pairs, off-line calculates matching degrees of I/O pairs and predefines membership functions to decide the final selected rules. No more new rules are generated on-line. So it could be thought as an off-line rule-generating technique.

In the past, designers of FLCs should define the membership functions covering a whole input domain and construct the rules for all possible conditions in order to satisfy the property of *completeness*. Otherwise in certain circumstance, no fuzzy rules will be fired. To write down all of the fuzzy rules is also a time consuming and difficult job even for experts.

A method based on TNN aiming at self-generating fuzzy logic rules is proposed here. By developing the self-organizing system, both the control rules and the domain basis are obtained automatically. We apply the adaptation law described in [17] to update the rules in the rule base. Its superiority comes from its ability in generating new rules and eliminating the useless rules. The approach proposed here is an on-line rule-generating algorithm which only generates the necessary rules. In fact, it is also satisfactory for completeness if new rules are generated. In other words, it generates rules along the trajectory of the states. This will reduce the rule base size especially in multi-input FLCs. Another advantage is that it is not necessary to predefine ranges of the states. If we set a threshold value and firing strengths are not larger than that, this means there is no rule satisfying the current states [12]. Hence generating a new rule for current states is necessary. Thus no matter where the states are, the self-organizing FLCs can always find matching rules. It can decrease the distances of the rules and obtain better inference result by setting a larger threshold value. Note, by this approach, we must make a trade-off between decreasing rule distances and reducing rule number.

In Section 2, we briefly describe the basic fuzzy logic and inference process. In Section 3, the conventional knowledge representations for fuzzy rules are introduced and the TNN model is proposed. In Section 4, we propose an approach to generate new fuzzy rules and incorporate a new D-controller (SOAFC+D) to improve its transient performance. We also prove that the nonlinear system is globally stable while applying the SOAFC+D. In Section 5, the unstable plant in [17] and other systems are simulated. Simulation results demonstrate the superiority of SOAFC+D. In Section 6, conclusions and future research are summarized.

2. Basic fuzzy logic systems

In this section, we shall simply describe some basic concepts of fuzzy set theory and fuzzy logic used in this paper. The more detailed discussions can be found in [7].

For an n-input and single output fuzzy system, it implements a mapping from $U \subset \mathbb{R}^n$ to $\mathbb{R}$. Assume the universes of discourse of the n-inputs are $U_1, U_2, \ldots, U_n$, respectively; i.e. $U = U_1 \times U_2 \times \cdots \times U_n$. For U_i, $i = 1, 2, \ldots, n$, there are m_i membership functions defined. Each membership function μ_{A_i} maps the input to the interval $[0, 1]$. Each fuzzy set can be viewed as a linguistic term and in fact it is a condition of the premise of a fuzzy rule. Since there are n inputs and the ith input has m_i membership functions, it comes at most $m_1 \cdot m_2 \cdots m_n$ rules. The Cartesian product of $A_1, \ldots, A_n$ is a fuzzy set in the product space $U_1 \times U_2 \times \cdots \times U_n$ with the membership function

$$\mu_{A_1 \times \cdots \times A_n}(u_1, u_2, \ldots, u_n) = \min[\mu_{A_1}(u_1), \mu_{A_2}(u_2), \ldots, \mu_{A_n}(u_n)] \tag{1}$$

or

$$\mu_{A_1 \times \cdots \times A_n}(u_1, u_2, \ldots, u_n) = \mu_{A_1}(u_1) \cdot \mu_{A_2}(u_2) \cdots \mu_{A_n}(u_n) \tag{2}$$

where $u_i \in U_i$. For calculating firing strengths for each rule, we assume that there are n conditions in the premise of each rule and its consequence is described only by a crisp set. Each condition is represented with a linguistic fuzzy set A_i. μ_{A_i} is the matching degree of input u_i and the ith condition. Then for a specific rule, the firing strength is obtained through Cartesian product (1) or (2). Next, we shall combine all the firing rules by a defuzzifying strategy. Assume the form of the ith rule is:

Rule i: if u_1 is A_1^i and ... and u_n is A_n^i

then u is a_i, (3)

where A_j^i is the fuzzy set of the jth condition of the ith rule, and a_i is a fuzzy singleton for simplicity. Let the firing strength of the ith rule be μ_i; i.e.

$$\mu_i = \min[\mu_{A_1}(u_1), \mu_{A_2}(u_2), \ldots, \mu_{A_n}(u_n)] \tag{4}$$

or

$$\mu_i = \mu_{A_1}(u_1) \cdot \mu_{A_2}(u_2) \cdots \mu_{A_n}(u_n). \tag{5}$$

The concept of defuzzification is: the larger the firing strength is, the more it contributes and the defuzzifying formula is written as

$$u = \sum^{n} \mu_i \cdot a_i \Big/ \sum^{n} \mu_i. \tag{6}$$

3. Fuzzy knowledge representation

3.1. Review of knowledge representation

A rule base can be viewed as the brain of a fuzzy controller. How to describe the knowledge in the brain is an important issue. A good rule base for fuzzy systems should contain the following properties:

- high speed: search the matching rules fast,
- flexibility: add, modify or delete rules easily,
- simplicity: an intuitive and clear rule base is easily understood,
- low cost: it must not cost too much for hardware implementation.

Almost all recent papers about fuzzy logic mainly discussed theory and learning algorithms of fuzzy rules, because these jobs seem the kernels in this field. On the other hand, if the knowledge representation of fuzzy rules can be improved, it could not only speed up the fuzzy reasoning process but also reduce the sizes of fuzzy rule bases. This will also be helpful for hardware design and implementation. In the past, the fuzzy rules were generally described as linguistic statements of the form:

IF ⟨*Conditions*⟩ THEN ⟨*Actions*⟩. (7)

Such exhibition is not efficient enough as the inference engine must spend much time to check every predicative condition in the IF–THEN rule base. The

Table 1
An example of look-up table.

	NL	*NS*	*ZE*	*PS*	*PL*
NL	*NL*	*NL*	*PL*	*PL*	*PL*
NM	*NL*	*NL*	*NL*	*PL*	*PL*
NS	*PL*	*NM*	*NL*	*PS*	*NL*
ZE	*PL*	*PM*	*ZE*	*NM*	*NL*
PS	*PL*	*NS*	*PL*	*PS*	*NL*
PM	*NL*	*NL*	*PL*	*PM*	*PL*
PL	*NL*	*NL*	*PL*	*PL*	*PL*

IF–THEN rules can be described as the form of look-up tables, and it is still widely applied. An example of look-up tables is shown in Table 1, where *PL*, *ZE* and *NS*, respectively, represent linguistic terms "positive large", "zero" and "negative small" and so on. In fact, a look-up table is an explicit representation if the dimension of the rule base is low. When the dimension of the rule base is higher than two, this method will be difficult to accommodate to this situation. It must include many look-up tables as Table 1 to complete the representation of all the rules. This will be too complex and time consuming to search the firing rules. In [6], a look-up table is reduced from 3 to 2 dimensions, whereas the fuzzy rules must be regular in some sense.

3.2. An intuitive knowledge representation based on neural networks (IKR)

Recently, a knowledge representation combining neural networks with fuzzy logic seems a promising approach in artificial intelligence [3, 5, 11, 21]. The main advantage of neural networks is fast computation due to parallel processing. Therefore, many researchers have applied neural networks not only as a database but also as an inference engine [5, 11]. Another advantage of neural networks is that it has mature learning algorithms. For example, the back-propagation (BP) learning algorithm and the learning vector quantization algorithm are most popular approaches for learning internal representation. The BP algorithm can to some extent approximate a fuzzy controller mapping by repeatedly updating the weights [11]. But it has serious problems – adding a new rule, removing a useless rule or modifying a bad rule will be difficult; i.e. it contradicts the

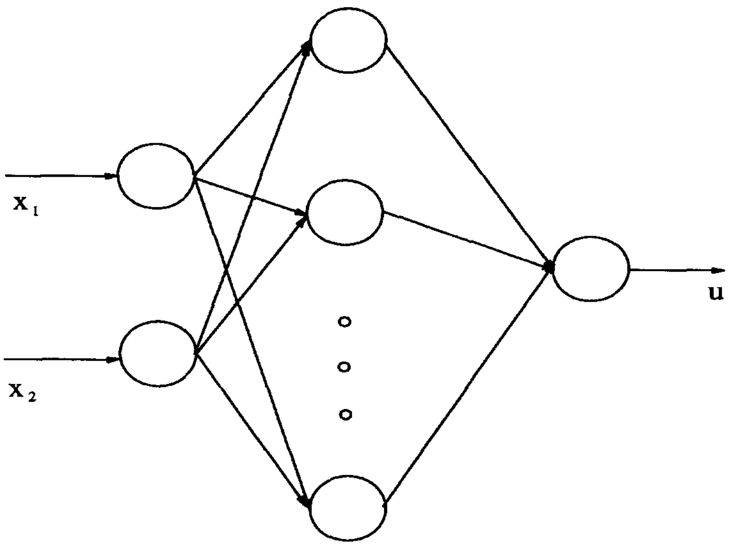


Fig. 1. The configuration of IKR.

aforementioned second property of a good fuzzy rule base. In other words, the approach only suits static rule bases. In the following, we shall recommend an intuitive knowledge representation for fuzzy rules. It can suit both static and dynamical rule base. As shown in Fig. 1, only three layers are required.

The nodes in the first layer just transmit input values to the next layer directly. The second-layered nodes are used to compute firing strength for fuzzy rules. Firstly, we must define membership functions. In the second layer, the membership functions used most often are the Gaussian or triangular functions. The Gaussian function is given by

$$g_G(x) = \exp\left[-\frac{1}{2\sigma}(x-w)^2\right] \tag{8}$$

where σ is called the *width* of the Gaussian function because the larger the value of σ is, the more the Gaussian spreads out, and w is called the *center*. The triangular function is

$$g_T(x) = \begin{cases} 1 - \dfrac{|x-w|}{b} & \text{if } |x-w| < b, \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

where b and w are the half-base length and base center of a triangular function, respectively. Both (8) and (9) achieve maximum value (=1.0) while $x = w$. Note that no matter which of the two membership function is used, it is in fact a function of $|x - w|$ and the larger the $|x - w|$ is, the smaller the membership value is.

For an n-input system, we define the second-layer input as $|x_i - w_i|$ for $1 \leqslant i \leqslant n$, which is similar to the distance measure in Kohonen's feature map [9]. A basic structure of a node in IKR is shown in Fig. 2 [5].

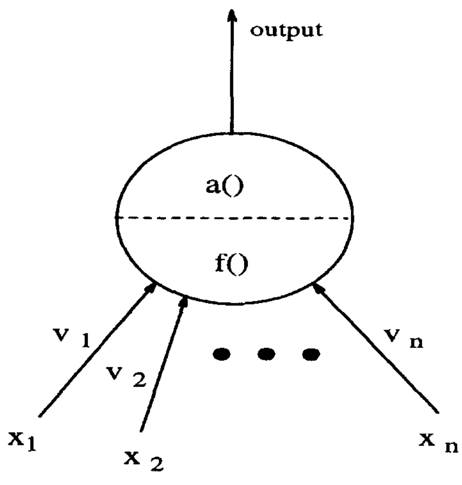


Fig. 2. Basic structure of a node in a neural network.

In the conventional approach, the net input of a node is given by

$$\text{net-input} = f(x_1,\dots,x_n; w_1,\dots,w_n) = \sum_{i=1}^{n} x_i w_i. \tag{10}$$

The output of the node is an activation value

$$\text{output} = a(f) \tag{11}$$

where $a(\cdot)$ denotes the activation function; for example a sigmoid function.

In IKR, we define the function f in the second layer by

$$f(x_1,\dots,x_n; w_1,\dots,w_n) = \max(|x_1 - w_1|,\dots,|x_n - w_n|) \tag{12}$$

and

$$\text{output} = g_T(f) \text{ or } g_G(f). \tag{13}$$

The outputs of nodes in layer 2 are the firing strengths for fuzzy rules, which correspond to the results of applying max-min compositional operation. The number of the nodes in the second layer represents number of the fuzzy rules.

For an on-line system, inference speed is the most important issue. Hence we assume that the consequences of the rules are singleton sets [7]; i.e. their membership functions are simply pulses rather than triangular or Gaussian functions. The singleton set is defined as

$$s(x) = \begin{cases} 1 & \text{if } x = v, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

There is only one node in the third layer. Assume v_i is the weight connecting the ith node in layer 2 and the node in output layer. Then the weights are recorded as

$$v_i = a_i \tag{15}$$

where a_i is the action part of the ith rule. The third-layer node achieves a defuzzifying job. Assume the output for the ith node in layer 2 is μ_i, that is the firing strength of the ith rule. The node output in layer 3 is given by

$$u = \sum^{n} \mu_i \cdot a_i \Big/ \sum^{n} \mu_i. \tag{16}$$

Thus the fuzzy inference result is extracted from IKR.

The IKR is very clear and just completes fuzzy inference step by step. Nevertheless, while the rules are copious or the dimension of rule base is high, the processing units will be numerous.

3.3. Tree-searched neural networks as a knowledge representation

As for hardware realization, too many processing units could result in large implementation cost and difficulty in design. For reducing the number of the processing units, we first consider a set of membership functions shown in Fig. 3 in domain X. The linguistic labels are marked beside the curves. We can find that the membership functions of *VNL*, *NS* and *PM* are almost nonoverlapping, where *VNL* stands for linguistic "very negative large", and so on. Suppose we define

$$\varepsilon = \min_{x\in X} \max\{\mu_{VNL}(x), \mu_{NS}(x), \mu_{PM}(x) \mid x \in X\}, \tag{17}$$

where μ_{VNL} is the membership function for linguistic term *VNL* and so on. ε is a very small positive number or zero. There is only one of the membership values for the three functions could be larger than ε for any $x \in X$; in other words, only one node is necessary to complete the three membership functions. The other two membership values can be thought zeros without influencing the fuzzy inference almost. Similarly, the other six functions can be achieved by two nodes, since we could ignore them if the membership values are smaller than ε, that is the concept of α-cut [4, 14]. If we define the α-cut of a fuzzy set A, A_ε, as follows:

$$A_\varepsilon = \{x \mid \mu_A(x) \geqslant \varepsilon\}, \quad 0 < \varepsilon \leqslant 1, \; x \in X, \tag{18}$$

therefore, it is only necessary to compute $\mu_A(x)$ if $x \in A_\varepsilon$.

Hence we provide a modified knowledge representation as Fig. 4. Basically the inference engine is the same as Fig. 1, except the weights in the second and third layers are dynamical and are determined according to the input states. We use tree structure as a database and the weights are put into it. Fig. 4(a) is an example showing how a tree in TNN stores parameters of fuzzy rules. This is equivalent to the fuzzy rules in Fig. 4(b), if the membership functions of linguistic terms "1" and "−1" are almost nonoverlapping.

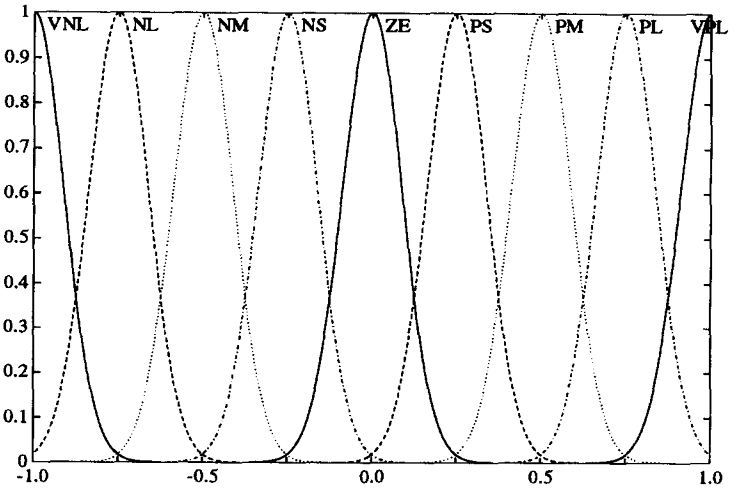
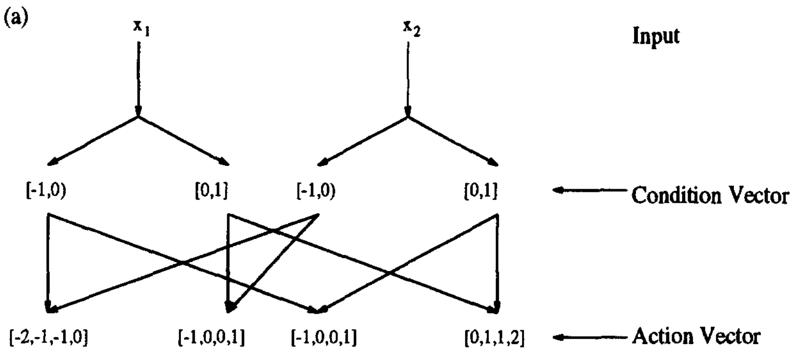


Fig. 3. An example of membership functions.

(a)



(b)

x_1 \ x_2	-1	0	1
-1	-2	-1	0
0	-1	0	1
1	0	1	2

Fig. 4. (a) An example of a tree memorying parameters of fuzzy rules. (b) The same parameters as (a) are represented by a look-up table.

The advantages of using trees as knowledge base are (1) searching is quick and (2) there are existent algorithms for adding and deleting subtrees. This is the reason why we use trees as a database. Note the depth of the trees used for memorizing the weights is usually shallow. It would not cost too much time to search. Thus, the neural network is a pure inference engine and no more a database now.

Assume each element in input vector is normalized to $[-1, 1]$. When an input vector is fed into the fuzzy controller, it will search the suitable weights from the trees and send to the neural network to get the approximate fuzzy inference result. Referring to Fig. 4, for example $x_1 = 0.3$ and $x_2 = -0.3$, we will search the centers 0, 1 for input x_1 and -1, 0 for input x_2 and place these centers on the weights in the second layer and ignore other membership functions. Evidently, there are at most 4 rules fired. The weights of the third layer are -1, 0, 0, 1, which are the consequence parts of the 4 fuzzy rules; i.e. these weights are decided by the searching results of the second layer. In fact, this process would not slow down the inference since while searching the third-layer weights, the second-layer nodes are also progressing with their jobs. Thus 5 nodes could be economized in this case.

How many nodes are required while applying TNN model? Assume η is the threshold for generating rules and the membership functions are all defined as (9). For input x_i, the half-base length is b_i. Then for an n-input system, at least $(\lceil 1/(1-\eta)\rceil)^n$ nodes in the second layer are required, where $\lceil\cdot\rceil$ is the ceiling function. Note the number of the nodes required in layer 2 is independent of half-base length of the membership functions but it is dependent only on η.

If we choose (8) as membership functions, the membership values are nonzero except for the input x_i where $|x_i - w_i|$ is infinity. We will discard it if membership value is smaller than $\varepsilon\,(>0)$. For the same system, at least $(\lceil\sqrt{\ln\varepsilon/\ln\eta}\rceil)^n$ nodes in the second layer are needed; similarly it is also independent of width of the Gaussian function.

Suppose that each node could perform p membership functions and there are q terms in each dimension. Then for an n-input fuzzy controller, IKR will require q^n nodes in the second layer for all fuzzy rules; nevertheless the TNN just needs $(\lceil q/p\rceil)^n$ nodes in the second layer to represent the q^n rules. Then we have

$$\frac{\text{Total node number of TNN}}{\text{Total node number of IKR}} = \frac{n+(\lceil q/p\rceil)^n+1}{n+q^n+1}. \quad (19)$$

For example, $q = 9$, $p = 3$ and $n = 3$, the node number of TNN is almost reduced to 31/733 of IKR. It is very evident that the number of the processing units has been reduced significantly.

3.4. Rule-generating algorithms

In this subsection, we give the rule-generating algorithms for both, the IKR and TNN models. The rule-generating algorithm for IKR is:

```
begin
  if (maximal firing strength) < $\eta$,
  {
     add a node,
     calculate $a_{r+1}$,
     the second-layer weights are written as the
        current states,
     the third-layer weight is equal to $a_{r+1}$,
  }
end.
```

And the following is the rule-generating algorithm for TNN:

```
begin
  if (maximal firing strength) < $\eta$,
  {
     for i = 1 to n
     {
        if $|x_i - c_{i,m}| > d$,
        {
           let $c_{i,m} = x_i$,
           add a subtree and insert $c_{i,m}$ into the
              condition vector,
        }
     }
     calculate $a_{r+1}$,
     connect the matching terms for each input,
     put $a_{r+1}$ and relative consequences to the new
        action vector,
  }
end.
```

In the TNN algorithm, d is a threshold for generating new term defined by user and $c_{i,m}$ is the center closest to x_i. And how to calculate a_{r+1} is stated in the next section.

4. A self-organizing adaptive fuzzy controller (SOAFC)

4.1. Self-generating rule

In this section, we shall show how and when to generate new rules. In the past, the design of a FLC was intensively dependent on expertise. Setting all membership functions, writing all the control rules and deciding scaling factors are all done by human experts. Furthermore, some parameters are necessary to be determined by the boring trial-and-error procedure.

In fact, operators' experience is difficult to be transformed into the rule form completely. Therefore, many rules coming from experts also need to be improved. By this reason, several self-learning and self-organizing algorithms have been widely studied. In the following, our objective is to design a FLC and without experts' help.

In the first, we consider an nth-order nonlinear system of the form

$$x^{(n)} = f(x,\dot{x},\ldots,x^{(n-1)}) + bu, \quad y = x, \tag{20}$$

where f is an unknown continuous function, b is a positive unknown constant, and $u, y \in \mathbb{R}$. Define the state vector $\boldsymbol{x} = (x_1,x_2,\ldots,x_n)^{\mathrm{T}} = (x,\dot{x},\ldots,x^{(n-1)})^{\mathrm{T}} \in \mathbb{R}^n$. Suppose the upper bound of $|f(\boldsymbol{x})|$ and lower bound of b are known as f^{U} and b_{L}, respectively. There are two parameters, the width and the center, in a Gaussian function (8). For simplicity, the width σ of the membership functions is thought a constant and is defined by users. The centers are the only parameters that can be adjusted. If there are some available linguistic rules or numerical data, they are easily put into the IKR or TNN as *prior* knowledge. The desired output is $y_m(t)$ and tracking error is $e \equiv y_m - y$. Assume we consider that e is in the interval $[-\alpha,\alpha]$ with high possibility, where α is a positive value set by users. But e can be out of the interval, for instance, in the case of additive large noise. So in fact, it is not necessary to definitely confine the range of e. Let the magnitudes of the consequences of fuzzy rules be all constrained below M_θ which is allowable maximal consequence of fuzzy rule. At first, we just consider a SISO FLC. If no *prior* fuzzy rules can be applied initially, two boundary rules are defined as

$$R_1: \quad \text{IF } e \text{ is } L_1, \text{ THEN } u \text{ is } \min(f^{\mathrm{U}}/b_{\mathrm{L}}, M_\theta), \tag{21}$$
$$R_2: \quad \text{IF } e \text{ is } L_2, \text{ THEN } u \text{ is } \max(-f^{\mathrm{U}}/b_{\mathrm{L}}, -M_\theta),$$

where

$$\mu_{L_1}(x) = \exp\left(-\frac{1}{2\sigma}(x-\alpha)^2\right),$$
$$\mu_{L_2}(x) = \exp\left(-\frac{1}{2\sigma}(x+\alpha)^2\right)$$

and the L_i stands for the ith linguistic term. The meaning is that a possible largest control is given while the error is also the largest. The two rules may not be "rules of thumb", but they at least would not drive the state to the wrong way in the beginning and decrease the possibility touching the bound M_x of state x, where M_x, defined by users, is allowable maximal magnitude of x. If α is not defined, two initial rules should be given. The initial rules could also be trained if they are fired. Clearly, only the two rules are insufficient to complete a good control. In the situation that no experts can provide rules, the only way is to generate rules through on-line learning.

Assume there are r rules, $R_1,R_2,\ldots,R_r$, in the rule base at time t. Let $c_1,c_2,\ldots,c_r$ and $a_1,a_2,\ldots,a_r$ are the centers of $L_1,L_2,\ldots,L_r$, and the consequences of $R_1,R_2,\ldots,R_r$, respectively. Now while signal $e(t)$ is fed to the controller, compute the firing strengths $\mu_1,\mu_2,\ldots,\mu_r$ for the r rules by the neural network mentioned in the last section. If the condition

$$\max_i \mu_i < \eta, \quad i = 1,2,\ldots,r \tag{22}$$

holds, this means that the existing r rules cannot satisfy current circumstance. Hence a new rule should be generated for current state. η should be defined as a positive value near 1.0 by users to decrease the distances between rules, since a small η may lead to a sparse rule base and bad performance. In fact, defining η is equivalent to giving the lower bound of the distances between rules. The first step is to find the two neighbor rules R_u and R_v such that $c_u < e(t) < c_v$, where $u,v \in \{1,2,\ldots,r\}$, and $\forall w \in \{1,2,\ldots,r\} - \{u,v\}$, either $c_w > c_v$ or $c_w < c_u$ is satisfied. Then a new rule is generated by *interpolation*:

$$R_{r+1}: \quad \text{IF } e \text{ is } L_{r+1}, \text{ THEN } u \text{ is } a_{r+1}, \tag{23}$$

where

$$\mu_{L_{r+1}}(x): \quad \exp\left(-\frac{1}{2\sigma}(x-c_{r+1})^2\right), \quad c_{r+1} = e(t), \tag{24}$$

and

$$a_{r+1} = a_u + \mathrm{sgn}(c_{r+1} - c_u) \cdot \frac{|c_{r+1} - c_u|}{|c_v - c_u|}(a_v - a_u). \quad (25)$$

In fact, it can produce two rules at once to speed up the rule-generating rate. The other rule is symmetric to (23)–(25)

$$R_{r+2}: \quad \text{IF } e \text{ is } L_{r+2}, \text{ THEN } u \text{ is } a_{r+2}, \quad (26)$$

where

$$\mu_{L_{r+2}}(x): \quad \exp\left(-\frac{1}{2\sigma}(x + c_{r+1})^2\right) \quad (27)$$

and

$$a_{r+2} = -a_{r+1}. \quad (28)$$

Generating one or two rules at once are decided by users and they are also dependent on limitation of rule base size. For many cases, more rules are required while error approaching zero, then η can be dynamically increased to reduce distance between rules. Furthermore, for restricting rule base size, the rule number must be below $\bar{r}$. When (22) holds and $r = \bar{r}$, it means the rule base is full and acquires a new rule. We shall replace the farthest rule with the new rule. The farthest rule is considered the rule R_k which satisfies

$$k = \arg \max_{1 \leqslant i \leqslant r} |e(t) - c_i|. \quad (29)$$

This substitution can be done just replacing the weights in IKR; while in TNN, it needs to delete a subtree and add another subtree. From the introduction in this subsection, the algorithm is evidently an on-line approach of generating rules. Based on this point, it is obviously different with the off-line method in [25].

Note if by any chance, the error is out of the interval $[-\alpha, \alpha]$, an extrapolation similar to interpolation can be applied to generate a better fuzzy rule. Find two closest rules R_u and R_v to the current state, i.e. $c_u < c_v < e(t)$ or $e(t) < c_u < c_v$, and the extrapolation formula is the same as (25). The magnitude of the consequence of new rules should also be smaller than or equal to M_θ. We have

$$a_{r+1} = \max\left\{-M_\theta, \min\left[M_\theta, a_u + \mathrm{sgn}(c_{r+1} - c_u) \times \frac{|c_{r+1} - c_u|}{|c_v - c_u|}(a_v - a_u)\right]\right\}. \quad (30)$$

For describing how to generate a new fuzzy rule with multiple antecedents, we give two distinct vectors $\boldsymbol{c}_1$, $\boldsymbol{c}_2$ and define $\boldsymbol{c}_1 > \boldsymbol{c}_2$ if all the elements of $\boldsymbol{c}_1 - \boldsymbol{c}_2$ are greater than or equal to 0s. Thus the interpolation (extrapolation) method is similar to the fuzzy rules with single antecedent. For interpolation, finding rules R_u and R_v such that $\boldsymbol{c}_u < \boldsymbol{e}(t) < \boldsymbol{c}_v$, $u, v \in \{1, 2, \ldots, r\}$ and $\forall w \in \{1, 2, \ldots, r\} - \{u, v\}$, either $\boldsymbol{c}_w > \boldsymbol{c}_v$ or $\boldsymbol{c}_w < \boldsymbol{c}_u$ is satisfied. For extrapolation, find two closest rules R_u and R_v such that $\boldsymbol{c}_u < \boldsymbol{c}_v < \boldsymbol{e}(t)$ or $\boldsymbol{e}(t) < \boldsymbol{c}_u < \boldsymbol{c}_v$ holds. Then we have

$$a_{r+1} = \max\left\{-M_\theta, \min\left[M_\theta, a_u + \mathrm{sgn}(\boldsymbol{c}_{r+1} - \boldsymbol{c}_u) \times \frac{\|\boldsymbol{c}_{r+1} - \boldsymbol{c}_u\|}{\|\boldsymbol{c}_v - \boldsymbol{c}_u\|}(a_v - a_u)\right]\right\}, \quad (31)$$

where $\|\cdot\|$ is some kind of distance measurement, for example Euclidean, and

$$\mathrm{sgn}[(c_1, c_2, \ldots, c_n)^{\mathrm{T}}] = \begin{cases} 1 & \text{if } \forall c_i > 0, \\ -1 & \text{if } \forall c_i < 0, \\ 0 & \text{otherwise.} \end{cases} \quad (32)$$

4.2. *Review of adaptive fuzzy systems*

The adaptation law in [17] will be simply described in this subsection. Since the adaptive fuzzy controller is globally stable, we shall apply this approach through this paper. For simplicity, the notations are the same as in [17].

The control is a combination of a fuzzy control $u_c(\boldsymbol{x}|\boldsymbol{\theta})$ and a supervisory control $u_s(\boldsymbol{x})$:

$$u = u_c(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{\xi}) + u_s(\boldsymbol{x}). \quad (33)$$

The fuzzy control is:

$$u_c(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{\xi}) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\xi}(\boldsymbol{x}), \quad (34)$$

where $\boldsymbol{\theta} = (a_1, a_2, \ldots, a_r)$ is a parameter vector which can be viewed as the consequence of the rules and $\boldsymbol{\xi}(\boldsymbol{x}) = (\xi^1(\boldsymbol{x}), \xi^2(\boldsymbol{x}), \ldots, \xi^r(\boldsymbol{x}))^{\mathrm{T}}$ is a regressive vector and can be thought as the firing strength vector. In [17], the dimension of the regressive vector is fixed because the rule number is fixed. Here, since the rule number is increased with time, the dimension of the regressive vector is also increased. Whenever a new rule is generated, $\boldsymbol{\theta}$ can be considered as another initial parameter vector. Then keep the elements in regressive vector and adjust the parameter vector using the adaptation law in [17]. Although $\boldsymbol{\theta}$ is viewed as an initial vector, in fact its elements are in better situations, because some elements have been trained and the new elements are generated by interpolation rather than at random. Hence the elements are not changed gravely while it is in steady state and it is expected that the system owns better transient response. Furthermore, it still keeps globally stable and could be proved as in [17].

Let $\boldsymbol{\theta}^*$ be the optimal parameter vector. Define the Lyapunov function candidate

$$V = \frac{1}{2}\boldsymbol{e}^{\mathrm{T}}P\boldsymbol{e} + \frac{b}{2\gamma}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi}, \tag{35}$$

where γ is a positive constant and can be thought as an updating rate and

$$\boldsymbol{\phi} = \boldsymbol{\theta}^* - \boldsymbol{\theta}. \tag{36}$$

Our task now is to decide u_{s} and adaptation law of parameter vector such that $\dot{V} \leqslant 0$. The result is shown in the following:

$$u_{\mathrm{s}}(\boldsymbol{x}) = I_1^* \operatorname{sgn}(\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b) \times \left[|u_{\mathrm{c}}| + \frac{1}{b_{\mathrm{L}}}(f^{\mathrm{U}} + |y_m^{(n)}| + |\boldsymbol{k}^{\mathrm{T}}\boldsymbol{e}|) \right], \tag{37}$$

where $I_1^* = 1$ if $V > \bar{V}$ and $I_1^* = 0$ if $V \leqslant \bar{V}$($\bar{V}$ is a upper bound defined by users) and $\boldsymbol{p}_n$ is the last column of P which is solved from Lyapunov equation.

The adaptation law is

$$\dot{\boldsymbol{\theta}} = \begin{cases} \gamma \boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n \boldsymbol{\xi}(\boldsymbol{x}) & \text{if } (|\boldsymbol{\theta}| < M_\theta) \text{ or } (|\boldsymbol{\theta}| = M_\theta \text{ and } \boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) \leqslant 0), \\ Prj\{\gamma \boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n\boldsymbol{\xi}(\boldsymbol{x})\} & \text{if } (|\boldsymbol{\theta}| \geqslant M_\theta \text{ and } \boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) > 0), \end{cases} \tag{38}$$

where $\dot{\boldsymbol{\theta}} = -\dot{\boldsymbol{\phi}}$, the projection operation $Prj\{*\}$ and the detailed proof is described in [17]. Note the dimension of $\boldsymbol{\theta}$ now increases with time. In (38), the condition $|\boldsymbol{\theta}| \geqslant M_\theta$ is little different with [17]. Since we shall use C language to simulate the control system, the sampling time is fixed and the value of the $|\boldsymbol{\theta}|$ may be a little larger than M_θ between two period time. Hence if we wish $\theta \leqslant M_\theta$ all the time, a value smaller than M_θ should be replaced in practice. Using the MATLAB command "ode23" may let $\theta \leqslant M_\theta$ almost always hold, but two consecutive control actions must be completed in very short time. Whether the fuzzy inference can be achieved in so small period time in practical applications should be considered.

4.3. Combine a D-controller

In the traditional PID control, the mathematical analysis is consummate. For example, the effect of an I- or D-controller can be found in the control textbook [15]. Basically, a D-controller can do well for the transient performance of a control system. Due to the slow convergent speed of the conventional adaptive control, we shall combine the fuzzy controller with a D-controller, i.e.

$$u = u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta}) + u_{\mathrm{s}}(\boldsymbol{x}) + u_{\mathrm{d}}(\boldsymbol{x}) \tag{39}$$

where

$$u_{\mathrm{d}} = \operatorname{sgn}(\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\dot{e})k_{\mathrm{d}}\dot{e} \tag{40}$$

and k_{d} is a positive constant and $\dot{e}$ is the second element of vector $\boldsymbol{e} = (e, \dot{e}, \ldots, e^{n-1})$. We can easily find $\operatorname{sgn}(\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\dot{e})$ is available. The reason why we define (40) will be explained in the next subsection. The SOAFC in this section simply considers error rather than change of error. In fact, we can also on-line generate a two-input (error and change of error) fuzzy controller, if the desired parameters are supported [6]. But the parameters must be designed by control experts. Since the error and change of error are two different scales, we need one or more parameters similar to PD controller to combine linearly the two states into a control. These parameters are generally the keys to a better response. For reducing the dependence of experts, we shall just discuss a one-dimensional rule-generating fuzzy system. In (39) and (40), it also contains a parameter k_{d}. Whereas the k_{d} can be assigned an arbitrary positive value below an upper bound given by users, it can improve the adaptive fuzzy control system without causing instability. An analysis of the stability is discussed in the next subsection.

4.4. *Stability analysis*

In this subsection, we shall simply describe the proof of stability applying the approach in the last subsection. Referred to [17], the first step is to specify the $k_1,\ldots,k_n$ such that all roots of

$$s^n + k_1 s^{n-1} + \cdots + k_n = 0 \tag{41}$$

are in the open left-half-plane and specify a positive-definite $n \times n$ matrix Q. After manipulating, the error dynamic equation is

$$e^{(n)} = -\boldsymbol{k}^{\mathrm{T}}\boldsymbol{e} + b[u^* - u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta},\boldsymbol{\xi}) - u_{\mathrm{s}}(\boldsymbol{x}) - u_{\mathrm{d}}], \tag{42}$$

where

$$u^* = \frac{1}{b}\left[-f(\boldsymbol{x}) + y_m^{(n)} + \boldsymbol{k}^{\mathrm{T}}\boldsymbol{e}\right]. \tag{43}$$

Eq. (42) is equivalent to

$$\dot{\boldsymbol{e}} = \Lambda_{\mathrm{c}}\boldsymbol{e} + \boldsymbol{b}_{\mathrm{c}}[u^* - u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta},\boldsymbol{\xi}) - u_{\mathrm{s}}(\boldsymbol{x}) - u_{\mathrm{d}}], \tag{44}$$

where $\boldsymbol{e} = (e,\dot{e},\ldots,e^{(n-1)})^{\mathrm{T}}$ and

$$\Lambda_{\mathrm{c}} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ -k_n & -k_{n-1} & \cdots & \cdots & \cdots & \cdots & -k_1 \end{bmatrix},$$

$$\boldsymbol{b}_{\mathrm{c}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b \end{bmatrix}. \tag{45}$$

Define $V_e = \frac{1}{2}\boldsymbol{e}^{\mathrm{T}}P\boldsymbol{e}$, where P is a symmetric positive-definite matrix satisfying the Lyapunov equation

$$\Lambda_{\mathrm{c}}^{\mathrm{T}}P + P\Lambda_{\mathrm{c}} = -Q, \tag{46}$$

where Q is also a positive-definite matrix. Applying (40), (44) and (46), we have

$$\begin{aligned} \dot{V}_e &= -\tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} + \boldsymbol{e}^{\mathrm{T}}P\boldsymbol{b}_{\mathrm{c}}[u^* - u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta},\boldsymbol{\xi}) \\ &\qquad - u_{\mathrm{s}}(\boldsymbol{x}) - u_{\mathrm{d}}] \\ &\leqslant -\tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} + |\boldsymbol{e}^{\mathrm{T}}P\boldsymbol{b}_{\mathrm{c}}|(|u^*| + |u_{\mathrm{c}}|) - \boldsymbol{e}^{\mathrm{T}}P\boldsymbol{b}_{\mathrm{c}}u_{\mathrm{s}} \\ &\quad -\boldsymbol{e}^{\mathrm{T}}P\boldsymbol{b}_{\mathrm{c}}u_{\mathrm{d}}. \end{aligned} \tag{47}$$

Substituting (37), (40) and (43) into (47) and considering $I_1^* = 1$ case, we have

$$\begin{aligned} \dot{V}_e &\leqslant -\tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} - \boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\,\mathrm{sgn}(\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\dot{e}) \cdot k_{\mathrm{d}}\dot{e} \\ &\leqslant -\tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} \\ &\leqslant 0. \end{aligned} \tag{48}$$

While $I_1^* = 0$, let the minimal approximation error be

$$\tilde{u} = u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta}^*) - u^*. \tag{49}$$

The error equation (44) can be written as

$$\begin{aligned} \dot{\boldsymbol{e}} &= \Lambda_{\mathrm{c}}\boldsymbol{e} + \boldsymbol{b}_{\mathrm{c}}\left[u^* - u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta}) - u_{\mathrm{s}}(\boldsymbol{x}) - u_{\mathrm{d}}\right] \\ &= \Lambda_{\mathrm{c}}\boldsymbol{e} + \boldsymbol{b}_{\mathrm{c}}\left[u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta}^*) - u_{\mathrm{c}}(\boldsymbol{x} \mid \boldsymbol{\theta}) - u_{\mathrm{s}}(\boldsymbol{x}) \right. \\ &\qquad\qquad \left. - u_{\mathrm{d}} - \tilde{u}\right] \\ &= \Lambda_{\mathrm{c}}\boldsymbol{e} + \boldsymbol{b}_{\mathrm{c}}[\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) - u_{\mathrm{s}} - u_{\mathrm{d}} - \tilde{u}] \end{aligned} \tag{50}$$

where $\boldsymbol{\phi} = \boldsymbol{\theta}^* - \boldsymbol{\theta}$. Define the Lyapunov function candidate

$$V = \frac{1}{2}\boldsymbol{e}^{\mathrm{T}}P\boldsymbol{e} + \frac{b}{2\gamma}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi}. \tag{51}$$

We have

$$\begin{aligned} \dot{V} &= -\frac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} + \boldsymbol{e}^{\mathrm{T}}P\boldsymbol{b}_{\mathrm{c}}(\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) - u_{\mathrm{s}} - u_{\mathrm{d}} - \tilde{u}) \\ &\quad + \frac{b}{\gamma}\boldsymbol{\phi}^{\mathrm{T}}\dot{\boldsymbol{\phi}} \\ &= -\frac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} + \frac{b}{\gamma}\boldsymbol{\phi}^{\mathrm{T}}\left[\gamma\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n\boldsymbol{\xi}(\boldsymbol{x}) + \dot{\boldsymbol{\phi}}\right] \\ &\quad -\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b(u_{\mathrm{s}} + u_{\mathrm{d}} + \tilde{u}). \end{aligned} \tag{52}$$

If we choose the adaptive law

$$\dot{\boldsymbol{\theta}} = \gamma\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n\boldsymbol{\xi}(\boldsymbol{x}), \tag{53}$$

and $u_{\mathrm{d}} = \mathrm{sgn}(\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\dot{e})k_{\mathrm{d}}\dot{e}$, $k_{\mathrm{d}} > 0$ and $u_{\mathrm{s}} = 0$ (because $I_1^* = 0$),

$$\begin{aligned} \dot{V} &= -\tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} - \mathrm{sgn}(\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\dot{e})k_{\mathrm{d}}\boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\dot{e} - \boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\tilde{u} \\ &\leqslant -\tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} - \boldsymbol{e}^{\mathrm{T}}\boldsymbol{p}_n b\tilde{u}. \end{aligned} \tag{54}$$

Table 2
The comparison of the adaptive controller and SOAFC(+D).

	Adaptive controller	SOAFC(+D)
1	Define m_i fuzzy sets	
2	Define m_i memberships functions, including widths σ_i and centers w_i for $1 \leqslant i \leqslant n$	Assign σ Define η k_d (if D-controller is included) $\bar{\delta}$ (if k_d is dynamic)
3	Construct the fuzzy rule base	Give two initial rules
4	Construct fuzzy basis functions	

A fuzzy system has been proven to be an universal approximator. If there are enough fuzzy rules, $\tilde{u}$ will approach zero and lead to $\dot{V} \leqslant 0$. Furthermore, for constraint $|\boldsymbol{\theta}| \leqslant M_\theta$, we can choose adaptive law as (38). The system can also be guaranteed globally stable if the adaptation law is chosen as (38), a supervisory control as (37) and the D-controller as (40). We always have $V_e \leqslant \bar{V}$. Note V_e is bounded implies $\boldsymbol{e}$ is bounded; in other words, $\boldsymbol{x}$ is also bounded.

If the parameter k_d is considered dynamical, we assume k_d is limited below the bound $\bar{\delta}$, where $\bar{\delta}$ is a positive number. How to adjust k_d can be found in [22]. Note whether k_d is dynamic or not, it not only keeps the system globally stable but also improves transient response of the system by using (40).

4.5. *Comparison of the adaptive controller and SOAFC(+D)*

For a FLC, several design tasks should be done before the controller begins to work. In the following, we shall discuss the design issues between adaptive FLC [17] and SOAFC(+D). The off-line processes are the same, e.g. specify $k_1, \ldots, k_n$ and $Q > 0$, solve the Lyapunov equation and design parameters M_θ, M_x and M_u. The difference between the initial controller constructions of the two models are listed in Table 2.

From Table 2, we can find there are fewer jobs to be done in SOAFC(+D). Especially, it is not necessary to construct the entire fuzzy rule base. In other words, the SOAFC(+D) has stronger learning ability. It not only learns the action part but also condition parts. Furthermore, while the error is accidentally out of bound set by users, the previous adaptive controller would not handle such circumstance, but the SOAFC(+D) can also generate a rule by extrapolation to control the situation.

4.6. *Comparison between SOAFC and MRAC*

Model reference adaptive control (MRAC) is a basic and effective approach in the field of adaptive control. In this subsection, we simply compare SOAFC with MRAC.

The disadvantages of MRAC are the following: (a) it is difficult to deal with nonlinear plants, (b) it requires some information, such as mathematical form and relative degree for linear time-invariant system, (c) it usually lacks *prior* knowledge for application. On the other hand, SOAFC has more adaptive parameters than MRAC in general and it is difficult to control MIMO systems.

5. Simulation

In this section, we choose two controlled systems firstly. Their mathematical models are respectively

Case A:

$$\dot{x}(t) = \frac{1 - e^{-x(t)}}{1 + e^{-x(t)}} + u(t) \tag{55}$$

and

Case B:

$$\dot{x}(t) = \frac{1}{1 + e^{-x(t)}} + u(t). \tag{56}$$

Our control objective is to regulate the plant output to the origin. Basically, both the two plants are unstable systems without control. The difference between

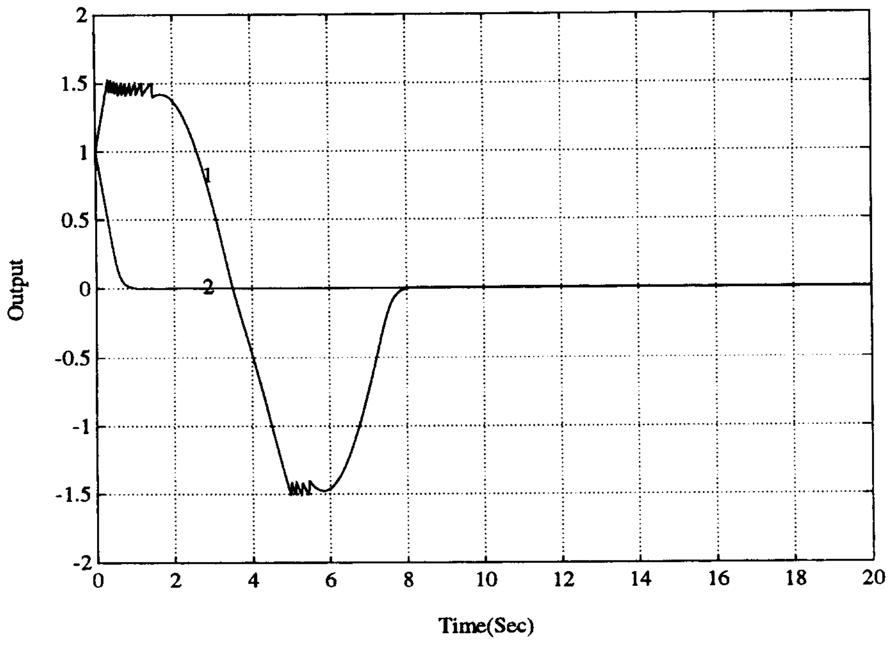


Fig. 5. An adaptive fuzzy controller for *Case A*.

the two cases is while state $x(t)$ tracking to the origin, it does not need control anymore in *Case A* if without any disturbance, but *Case B* must own a constant action to keep $x(t) = 0$. The parameters are chosen as $\gamma = 1.0$, $M_x = 1.5$, $M_\theta = 2.0$, $f^{\mathrm{U}} = 1.0$ and $b_{\mathrm{L}} = 0.5$ for both cases. We define the *width* σ in Gaussian function is $\frac{1}{2}$, and it is not necessary to construct every membership function. Since M_x is defined to be 1.5, we can assume the state error $e(t)$ will be in the interval $[-2, 2]$ with very high possibility. $\bar{r}$ is set to be 9; that is, there are at most 9 rules in the rule base. Then we define two boundary fuzzy control rules:

$$R_1\text{: IF } e(t) \text{ is } PL, \text{ THEN } u \text{ is } \min\left(\frac{f^{\mathrm{U}}}{b_{\mathrm{L}}}, M_\theta\right), \tag{57}$$

$$R_2\text{: IF } e(t) \text{ is } NL, \text{ THEN } u \text{ is } \max\left(-\frac{f^{\mathrm{U}}}{b_{\mathrm{L}}}, -M_\theta\right),$$

where $\mu_{PL}(u) = \exp(-(u - 2)^2)$ and $\mu_{NL}(u) = \exp(-(u + 2)^2)$. The threshold η is defined as 0.8. It means one or two rules will be added to the rule base while all the firing strengths are smaller than 0.8 as mentioned in the last section. We use C language to simulate the control problems and choose sampling time $h = 0.02$ s. For comparing the results, the initial state $x(0)$ is chosen as 1.0, the same as [17].

Fig. 5 shows the outcomes for *Case A*, where curves 1 and 2 are the simulation results obtained by using 9 random rules and two rules described as is (57), respectively. They both can track to the origin; especially curve 2 owns a very good transient response, whereas the curve 1 touches the boundary and fires the supervisory control u_s several times. We can also find there are several points in curve 1 a little larger than M_x. Let us consider the other plant *Case B*. The results are shown in Fig. 6 by using the same rules as in Fig. 5. We find the performances are not acceptable. Their convergent speeds are very very slow. Clearly, for curve 2, only applying two rules is not enough to control the system; i.e. it uses a rather sparse rule base. Furthermore random rules may cause the problem as shown in Fig. 7. The marks on the curves stand for the centers of the rules. We can see the action of the rule centered 1 and -1 are stronger than the rule centered 2 and -2, respectively. That contradicts our intuition. It is due to this that the rules centered 2 and -2 are not trained completely. Hence the rule base evidently contains several bad rules.

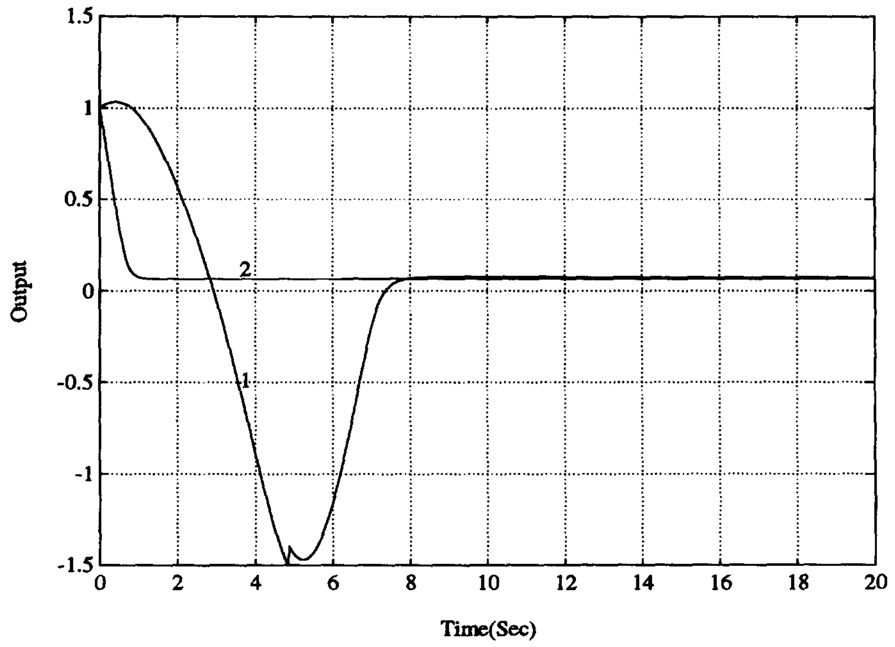


Fig. 6. An adaptive fuzzy controller for *Case B*.

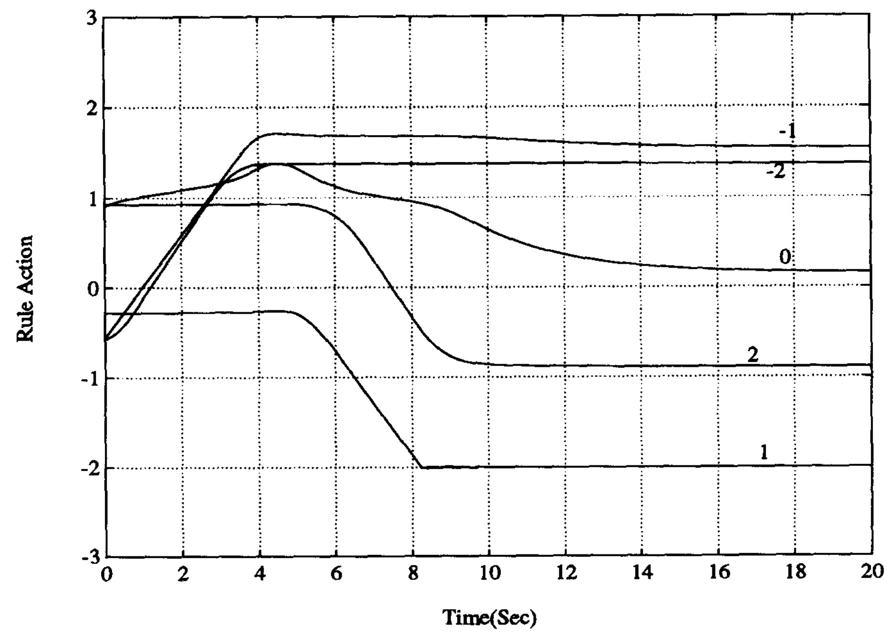


Fig. 7. A bad rule base caused by initially random rules.

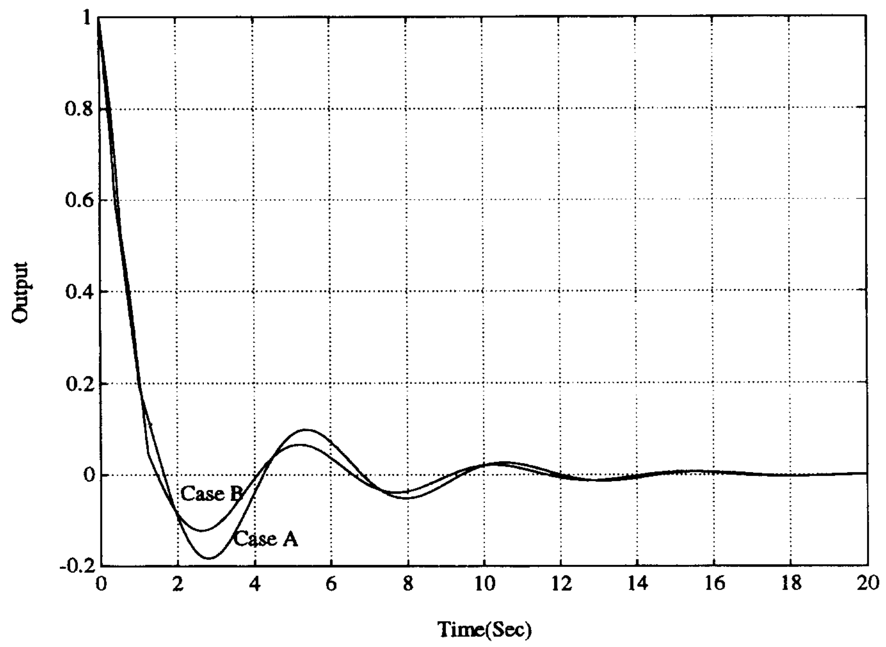


Fig. 8. A SOAFC simulations for *Case A* and *B*.

Fig. 8 shows the simulations of SOAFC that uses the two boundary rules (57). It could control the state $x(t)$ to zero much faster than Fig. 6 for both *Case A* and *Case B*, but their undershoots seem rather large. Fig. 9 displays adaptation and generation of rules. The curves from infinite falling down stand for the new generated rules. We can find the actions are not changed much from generation to steady state. That is due to the generation of new rules by *interpolation* rather than by random. For improving its transient response, a D-controller is combined with the original SOAFC. Assign $k_\text{d} = 30.0$ through the following simulations. At first, we combine a simple D-controller which is

$$u_\text{d} = k_\text{d} \cdot \dot{e}. \qquad (58)$$

Fig. 10 shows the simulation result. From Fig. 10, we see that it owns a little faster convergence speed than Fig. 8, but their undershoots are not improved. From Fig. 11, if the proposed D-controller (40) is included, the smaller undershoot can be found and settling time is smaller than others. Furthermore, an advantage of such controller is that it contributes less control energy than other aforementioned controllers, and it is still guaranteed that it is globally stable.

Table 3 lists the total control energy from 0 to 20 s for every controller we apply. In Table 3, AFC represents adaptive fuzzy controller in [17]. From Table 3, we can find the total control energy of AFC using two boundary rules is smallest for *Case A*, but it cannot control *Case B* well. Hence, the SOAFC+D is better than others. In *Case B*, except the AFC by random rules, the control energies of the other controllers are almost the same.

For comparing MRAC and SOAFC+D, we choose the example in [18]. Consider the linear controlled plant

$$\dot{x}_p(t) = a_p(t)x_p(t) + k_p(t)u(t), \qquad (59)$$

where $a_p = 1.0$ and $k_p = 2.0$ are unknown (sign of k_p is known), and a reference model is described by the first-order differential equation

$$\dot{x}_m(t) = a_m x_m(t) + k_m r(t), \qquad (60)$$

where $a_m = -1.0$, $k_m = 1.0$ and $r(t) = 5.0$. The control input is chosen as

$$u = \psi(t)x_p(t) + k(t)r(t), \qquad (61)$$

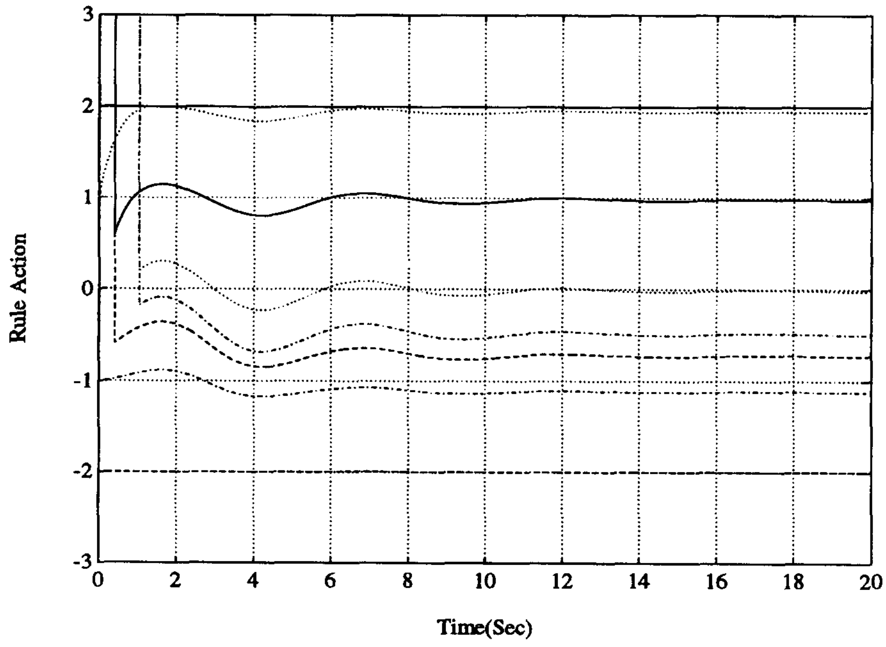


Fig. 9. An example of adaptation and generation of rules.

Table 3
The total control energy of adaptive controller and SOAFC.

	Case A	Case B
AFC using 9 random rules	338.0707	638.2547
AFC using two rules	57.8256	569.5872
SOAFC	125.9191	566.9734
SOAFC + $k_d \cdot \dot{e}$	97.0321	552.4740
SOAFC + $\operatorname{sgn}(e^T p_n b\dot{e}) k_d \dot{e}$	75.2068	551.5142

and the adaptive laws are

$$\dot{\psi}(t) = -\operatorname{sgn}(k_p)e(t)x_p(t), \tag{62}$$

$$\dot{k}(t) = -\operatorname{sgn}(k_p)e(t)r(t). \tag{63}$$

Let initial values $\psi(0)$ and $k(0)$ be both chosen as 0 s. Fig. 12 shows the simulation results of MRAC and SOAFC+D. Fig. 13 illustrates the simulation results while $a_p = 2.0$ and $k_p = 1.0$. The tracking error of MRAC is the best one we can achieve by changing adaptive gain. From Figs. 12 and 13, we can find SOAFC+D cause smaller oscillation and faster convergence than MRAC. The control energy of MRAC and SOAFC+D are 449.823 and 269.796 for Fig. 12, respectively, and are 1394.025 and 1062.216 for Fig. 13.

6. Conclusion

In this paper, an alternative knowledge representation is proposed first. The TNN model of fuzzy rule base is more intuitive than conventional BP learning. So a rule could be added, deleted or modified without affecting other rules. In other words, it is not necessary to update all the weights renewedly. The IKR may require large processing units to implement when there are numerous rules. Thus we propose the model TNN. It could definitely decrease the number of the processing units. This can help hardwares design, manufacture and even reduce the cost. Secondly, a SOAFC(+D) is provided. It can generate rules by itself and use the adaptation law in [17]

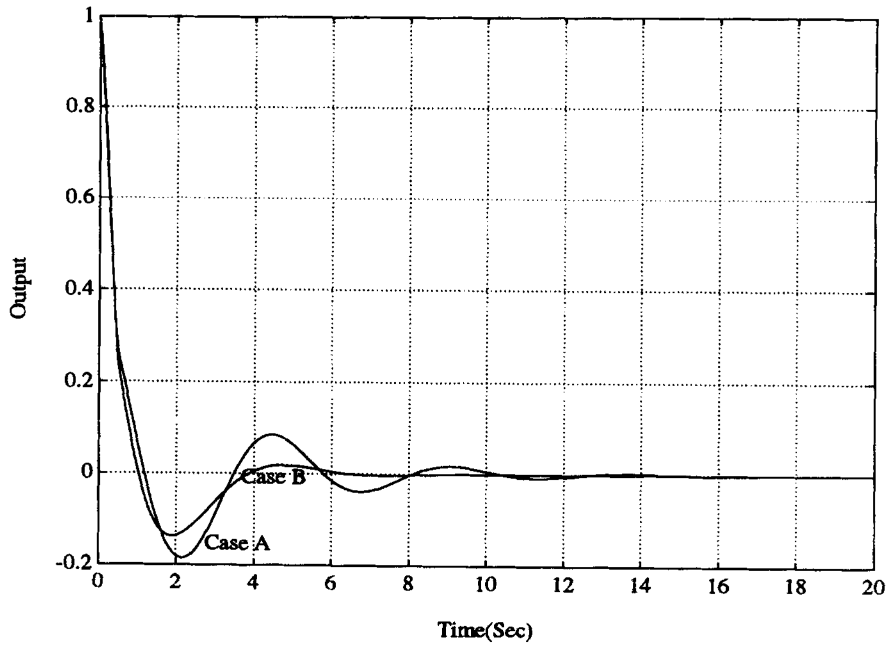


Fig. 10. Simulations of SOAFC combining a normal D-controller simulations.

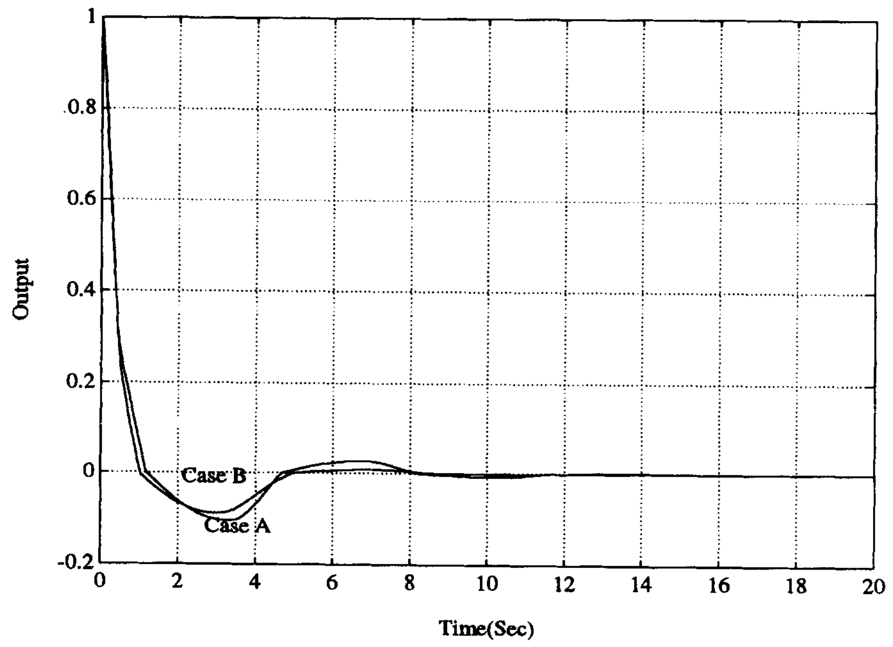


Fig. 11. Simulations of SOAFC combining a designed D-controller simulations.

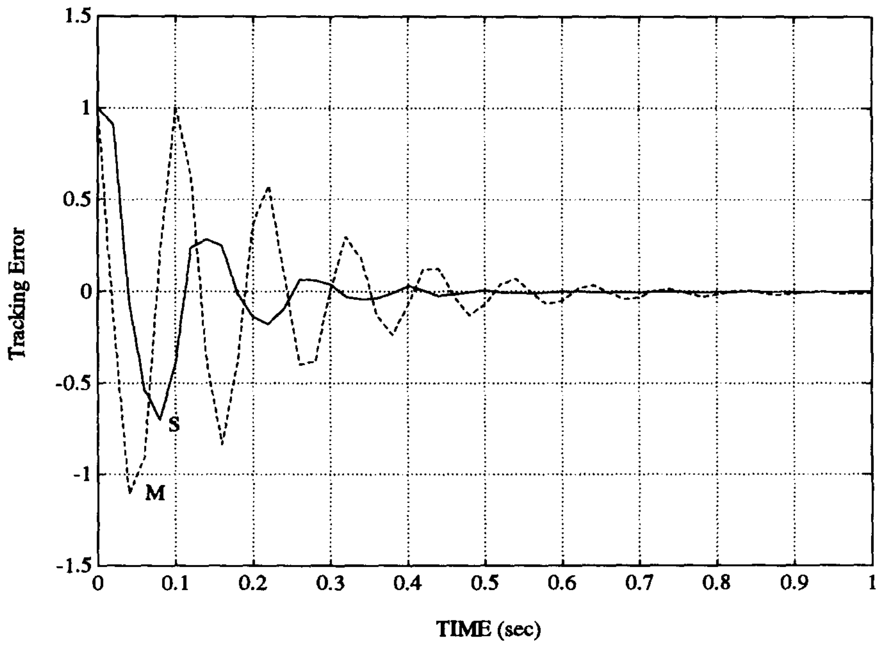


Fig. 12. Simulation results of MRAC and SOAFC+D while $a_p = 1.0$, $k_p = 2.0$ (M:MRAC, S:SOAFC+D).

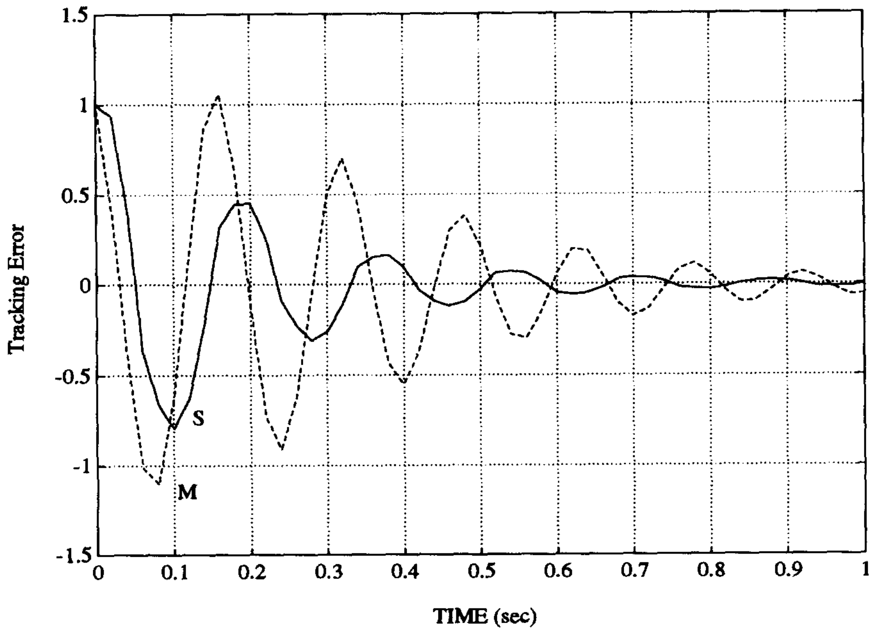


Fig. 13. Simulation results of MRAC and SOAFC+D while $a_p = 2.0$, $k_p = 1.0$ (M:MRAC, S:SOAFC+D)

to adjust the consequences of the firing rules. That could release much dependence on experts, since it is not necessary to write down all the rules by them. Furthermore, we design a D-controller and combine it with FLC to speed up the convergence rate, and we also simply prove it is globally stable if applying the proposed D-controller we design. From simulations, it owns several advantages, e.g. decreasing settling time and undershoot (overshoot) and requiring less control energy. The future work is to develop a multi-input SOAFC+D and the parameters, for example k_d should be adjusted by an adaptation law and can keep the system globally stable.

References

[1] W.W. Armstrong and J. Gecsei, Adaptation algorithms for binary tree networks, *IEEE Trans. System Man Cybernet.* **SMC-9** (1979) 276–285.

[2] H.R. Berenji, Learning and tuning fuzzy logic controllers through reinforcements, *IEEE Trans. Neural Networks* **3** (1990) 724–740.

[3] F. Bouslama and Akira Ichikawa, Application of neural networks to fuzzy control, *Neural Networks* **6** (1993) 791–799.

[4] Chen-Wei Xu and Yong-Zai Lu, Fuzzy model identification and self-learning for dynamic systems, *IEEE Trans. System Man Cybernet.* **SMC-17** (1987) 683–689.

[5] Chin-Teng Lin and C.S. George Lee, Neural-network-based fuzzy logic control and decision system, *IEEE Trans. Comput.* **40** (1991) 1320–1336.

[6] Chir-Ho Chang, Feng-Hsin Huang and J.Y. Cheung, Design of a fuzzy controller using input and output mapping factors, *IEEE Trans. Systems Man Cybernet.* **21** (1991) 952–960.

[7] Chuen-Chien Lee, Fuzzy logic in control systems: fuzzy logic controller – part I and II, *IEEE Trans. Systems Man Cybernet.* **20** (1990) 404–435.

[8] Chuen-Tsai Sun, Dynamic compensatory pattern matching in a fuzzy rule-base control system, *Proc. Amer. Control Conf.*, Boston (1991) 502–505.

[9] R. Hecht-Nielsen, *Neurocomputing* (Addison Wesley, Reading, MA, 1990).

[10] Hisao Ishibuchi, Ryosuke Fujioka and Hideo Tanaka, Neural networks that learn from fuzzy if-then rules, *IEEE Trans. Fuzzy Systems* **1** (1993) 85–97.

[11] Junhong Nie and D.A. Linkens, Neural network-based approximate reasoning: principles and implementation, *Internat. J. Control* **56** (1992) 399–413.

[12] Junhong Nie and D.A. Linkens, Learning control using fuzzified self-organizing radial basis function network, *IEEE Trans. Fuzzy Systems* **1** (1993) 280–287.

[13] Jyh-Shing R. Jang, Self-learning fuzzy controllers based on temporal back propagation, *IEEE Trans. Neural Networks* **3** (1992) 714–723.

[14] Kiyohiko Uehara and Masayuki Fujise, Fuzzy inference based on families of α-level sets, *IEEE Trans. Fuzzy Systems* **1** (1993) 111–124.

[15] B.C. Kuo, *Automatic Control Systems* (Prentice-Hall, Englewood Cliffs, NJ, 1991).

[16] R. Langari, A nonlinear formulation of a class of fuzzy linguistic control algorithms, *Proc. Amer. Control Conf.*, Chicago (1992) 2273–2278.

[17] Li-Xin Wang, Stable adaptive fuzzy control of nonlinear systems, *IEEE Trans. Fuzzy Systems* **1** (1993) 146–155.

[18] Narendra and Annaswamy, *Stable Adaptive Systems* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[19] T.J. Procyk and E.H. Mamdani, A linguistic self-organizing process controller, *Automatica* **15** (1979) 15–30.

[20] S. Shao, Fuzzy self-organizing controller and its application for dynamic processes, *Fuzzy Sets and Systems* **26** (1988) 151–164.

[21] Shin-ichi Horikawa, Takeshi Furuhashi and Yoshiki Uchikawa, On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm, *IEEE Trans. Neural Networks* **3** (1992) 801–806.

[22] Shi-Zhong He, Shaohua Tan and Chang-Chieh Hang, Control of dynamical processes using an on-line rule-adaptive fuzzy control system, *Fuzzy Sets and Systems* **54** (1993) 11–22.

[23] K. Sugiyama, Rule-based self-organizing controller, in: M.M. Gupta and T. Yamakawa, Eds., *Fuzzy Computing* (North-Holland, Amsterdam, 1988) 341–353.

[24] Toru Yamaguchi, Naoki Imasaki and Kazuhito Haruki, Fuzzy rule realization on associative memory system, *Proc. 1990 Internat. Joint Conf. on Neural Networks*, San Diego (1990) II720–II723.

[25] L.X. Wang and J.M. Mendel, Generating fuzzy rules by learning from examples, *IEEE Trans. Systems Man Cybernet.* **22** (1992) 1414–1427.

[26] B.H. Wang and G. Vachtsevanos, Fuzzy dynamic systems: an application of fuzzy associative memory with an intermediate layer, *Amer. Control Conf.*, Boston (1991) 12–13.