Tuning a fuzzy logic controller

Citation for published version (APA):

Document status and date:
Published: 01/01/1995

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Tuning a Fuzzy Logic Controller: 
An introduction 

G.Z. Angelis 
393590 

WFW report 95.068 

Coach: Ir. J.G.M.M. Smits 

Eindhoven University of Technology 
Faculty of Mechanical Engineering 
Department of Fundamentals of Mechanical Engineering 

practical assignment, 27-05-1995
ABSTRACT:
In this paper several tuning methods for Sugeno’s fuzzy systems will be discussed. In the first case the fuzzy controller is identified off-line based on training data. Following this approach, first the structure of the controller is identified by means of a clustering algorithm. A Kohonen self-organizing neural network performs this task. Then the parameters of the fuzzy controller (ie. membership functions) are tuned by using a gradient descent algorithm. This approach shows analogies with training a Radial basis function neural network.

In the second case the fuzzy controller will learn to control the system in an on-line situation. The controller parameters are adapted on a supervised manner by using a gradient descent method. To make the parameter adaptation possible we need to know the sensitivity functions of the system or at least their sign. If this knowledge is available the specialised learning technique becomes possible. Otherwise we need a preceding learning stage, the identification of the system. It is easily shown that when a multilayer perceptrons neural network emulates the system the sensitivity functions of the system can be derived by a mechanism of back propagation (applying gradient descent) no more on the weights but on the input of the emulator.

KEY WORDS:
identification, Sugeno fuzzy controller, Radial basis function neural network, Kohonen self-organizing neural network, Multilayer perceptrons neural network, supervised learning, unsupervised learning, gradient descent, specialised learning, controller/emulator learning

1 Introduction

Since fuzzy systems and neural nets are easy to implement and can approximate to any degree any non linear mapping [1,2,3,4], their use for control is expanding rapidly. One important difference between these two design technics is the interpretation of the structure which is logically supported in the fuzzy modelling case. However, the large number of degree of freedom in the design of the fuzzy controller leads to a lack in methodology and the question occurs: 'Is this transfer behaviour optimal?[5].

Hence, automatically tuning method’s are proposed. Recently a number of methods have been proposed that provide fuzzy systems with the kind of automatic tuning methods typical of neural nets. Then after learning the symbolic fuzzy framework which represents structured knowledge is not give up. In definitive the learning is nothing else than the application of a gradient method for the fuzzy system parametric tuning (ie. membership functions)[6]. Other researchers use self-organizing neural networks to find clusters in the controller input-output data to extract a fuzzy rulebase[7]. This paper is an introduction to the principles of the most commonly used tuning method’s of fuzzy controllers. We shall consider two different cases.

2 Fuzzy Controller identification

The transfer function of a fuzzy controller (system) is not based on a mathematical model but it is given by the definition of fuzzy rules and fuzzy sets of linguistic variables. The fuzzy rules and fuzzy sets are designed on the basis of human operators experiences, decisions and control actions. Like in conventional expert systems, the operator cannot always clearly explain why he acts in a certain way. Then an automatic design method becomes important, which is based on a set of examples for input-output relationship named referential dataset.
We first consider the case where a set of operating data is available that describes the desirable control output, $u'$, for various values of the process-state, $x'_1, \ldots, x'_n$. So we try to design a fuzzy controller that emulates a human expert or an existing controller, see fig.1. We shall consider this as the controller identification problem. The methods derived from fuzzy and neural algorithms found in the literature can be divided into two levels.

On the first level *unsupervised learning* methods are used. This means that inputs and outputs are classified and clustered. Cluster analysis has taken a new path recently by the application of fuzzy sets to self-organizing neural networks thus making a natural connection between clusters and fuzzy sets[9]. These self-organizing methods are used to extract the fuzzy rules from the input-output data. More specific, these algorithms based on the Kohonen self-organizing maps calculate the center positions of the input- and output membership functions and the strength of the connections between them as well.

Then a number of sets of parameters have to be chosen to describe the fuzzy controller. These are the fuzzy sets representing the meaning of linguistic values and the inference engine. The derived initially setting of the parameters of the fuzzy controller will be finetuned by *supervised learning* methods to modify the controllers performance. Recent work has centred on the use of mathematical optimization techniques to alter the set definitions so that the output from the fuzzy controller matches a suitable set of reference data as closely as possible. This procedure is carried out off-line in a supervised manner and so tunes the controller before it is used.

The control problem can be formulated as an optimization problem that consists in finding the control action which attains a desired objective while optimizing a specified criterion[10]. This method assumes that for each input the desired output is known. The desired output can be used to form an error measure (optimization criterion) which indicates the performance. Several error correction learning laws can be used to minimize the error measure and thus maximize the performance. One of the simplest methods for solving this optimization problem is the steepest gradient descent algorithm. The multilayer perceptrons neural networks version of gradient descent is known as back-propagation (of errors)[8]

### 2.1 Self-organizing maps

A fuzzy controller is a collection of fuzzy rules of the form[7]:

if $x_1$ is $A_1$ and...and $x_m$ is $A_m$, then $u$ is $B$

Given:

- $n_{x_i}$ the number of fuzzy sets defined for each input variable $x_i$
- $n_u$ the number of fuzzy sets or singletons defined for output variable $u$

then the number $r$ of the possible fuzzy rules that can be defined is:

$$r = n_u \times \prod_{i=1}^{m} n_{x_i}$$

This value grows quickly, and all the possible rules are certainly not needed to model the system.

Kosko[11] proposes an algorithm for the definition of a suitable sets of fuzzy rules. He also suggests the use of neural networks for the identification of such set of rules, starting from a training set (referential data set) consisting of examples of the input-output data of the problem. The key
geometric idea is cluster equals rule.

These neural networks seek to find patterns or regularity in the data. The network that is used is a winner takes all network. It performs a clustering on the input-output space according to a competitive learning algorithm known as Kohonen learning[12].

2.1.1 Kohonen Network architecture

The network consists of two layers of units (neurons). The input and the output layer. Given \( m \) input variables and one output variable, there are \( m+1 \) units in the input layer. Given \( r \) possible fuzzy rules for the problem, there are at most \( r \) units in the output layer. The rule of propagation of the signals in the network is the inner product between the input vector \( z \) and each weight vector \( v_j \). Namely for each output unit \( i \) its activation value is

\[
a_i = v_i^T z,
\]

where

\[
v_i = [v_{i1}, v_{i2}, \ldots, v_{i(m+1)}]^T.
\]

The architecture of a Kohonen self-organising map is shown in fig.2.

2.1.2 Kohonen learning

In this form of learning, the output units that update their weights do so by forming a new weight vector that is a linear combination of the old weight vector and the current input vector. Typically, the unit whose weight vector was 'closest' to the input vector 'wins' and is allowed to learn. The method of determining the winner uses the squared Euclidean distance between the input vector and the weight vector and chooses the unit whose weight vector has the smallest Euclidean distance from the input vector. If the input and weight vector are scaled to unit length then the output unit with the largest input wins and its corresponding weight vector is allowed to learn. 'Close' can then be interpreted as the smallest angle between the input and weight vector.

The weight vector \( v_i \) is updated according to the simple rule of moving the weight vector towards the direction of the input vector, in such a way to minimise the distance \( E \) between the center of the cluster the input vector belongs to and the input vector:

\[
E = \sum_{p\text{-training}} |z^p - v_i^p|^2
\]

then the updating rule for weight vector \( v_i^p \) at the time \( t+1 \) is given by:

\[
v_i^p(t+1) = v_i^p(t) + \eta (z^p - v_i^p(t)) \quad j=1 \ldots (m+1)
\]

where \( \eta \) is the learning rate. The learning rate \( \eta \) is a slowly decreasing function of time. Random weights may be assigned for the initial weights. If some information is available concerning the distribution of clusters that might be appropriate for a particular problem, the initial weights can be
taken to reflect that prior knowledge and to speed up the self-organising process.

To illustrate the essence of competitive learning, we may use the geometric analogy depicted in fig.3.[8] It is assumed that the input and weight vector are scaled to unit length, so that we may view input vector \( z \) as a point on a \( m+1 \) dimensional unit sphere, where \( m+1 \) is the number of input units; \( m+1 \) also represents the dimension of each weight vector \( y \). Thus, they form a set of vectors that fall on the same \( m+1 \) dimensional unit sphere. In fig.3a we show three natural groupings (clusters) of the input-output product space \((x, u)\) represented by dots; this figure also includes a possible initial state of the network weights (represented by crosses) that may exist before learning. Fig.3b shows a typical final state of the network that results from the use of competitive learning.

At the end of the training phase, each weight vector corresponding to an output unit \( i \) that has won many times the competition represents the center of a cluster in input-output space. It represents a fuzzy rule for the problem at hand. So it's possible to determine through a neural network the number of suitable fuzzy rules for the problem, and the center of the involved fuzzy sets for each state variable and for the control variable. In fact, each vector

\[
y_j = [v_1, v_2, ..., v_{(m+1)}]^T
\]

obtained through the clustering algorithm corresponds to the fuzzy rule:

\[
\text{if } x_1 \text{ is } v_1 \text{ and } x_2 \text{ is } v_2 \text{ and } ... \text{ and } x_m \text{ is } v_m \text{ then } y \text{ is } v_{(m+1)}
\]

where each value represents the center of a fuzzy set.

2.2 Gradient descent method.

In this sub-chapter we present the gradient descent method developed by Nomura et al. for the automatic adjustment of membership functions of Sugeno's type of fuzzy systems[13]. The algorithm is among the simplest for tuning controller parameters. We consider only a multi-input single-output controller without loss of generality. Alternative algorithms could include more sophisticated optimization methods.

First the fuzzy control rules and the reasoning method will be presented and afterwards a gradient descent method for the optimization will be discussed. The main requirement of the gradient descent method will be the differentiability of the functions containing the tuning parameters[14]. We optimize the linguistic terms and not blindly all the parameters to reflect the meaning of the linguistic values taken by the variables. For example the aggregation of the inputs is usually an 'and' operator (t-norm i.e. min. or prod.). The product operator is differentiable with respect to the input functions. If the simple minimum operator is used, it is not possible to build the derivation without any restrictions. If a more general aggregation function is chosen which depends on a parameter \( \gamma \), the aggregation can be a t-norm or a s-norm or something in between, then the interpretation of the fuzzy framework can be lost when \( \gamma \) is a parameter in the optimization procedure.

2.2.1 The fuzzy logic controller.

Let us consider a controller that consists of a set of \( n \) if-then rules of the form:
Rule i: if $x_i$ is $LX_i^{(0)}$ and ... and $x_m$ is $LX_m^{(0)}$ then $u$ is $LU^{(0)}$ (1)

where $x_i, x_m$ are the controller inputs (process-state variables), $u$ is the controller output variable, $i$ is the rule number, $LX_i^{(0)}, ..., LX_m^{(0)}$ are the linguistic values of the rule antecedent, and $LU^{(0)}$ is the linguistic value of the rule consequent, in this case a singleton. The membership function for the rule antecedent which is a triangular function is given by fig.4 and formula (2):

$$LX_i^{(0)} = \max(1 - \frac{2|X_i^{(0)} - \frac{a_i}{b_i}|}{b_i}, 0)$$  (2)

fig.4 Membership function

Then each linguistic term for each variable $X_j$ is described by the parameters $a_{ij}$ and $b_{ij}$ (mean value and breadth) that a gradient method will try to optimize. The Membership function must be differentiable with respect to the parameters. This is true for example for Gaussian functions (with parameter mean value $\mu_{ij}$ and standard deviation $\sigma_{ij}$). But if a triangular function is used, it is not possible to build the derivation without any restrictions. In this case the generalized gradient can be used. The crisp value $LU^{(0)}(u)$ for the output will also be subject to an optimization process. Note that by changing the shape and position of the membership functions, the rules can be changed.

Using the center of singleton 'defuzzification' method, the non-fuzzy output $u$ from the rule set is given by:

$$u = \frac{\sum_{i=1}^{n} \mu_i \cdot u_j}{\sum_{i=1}^{n} \mu_i}$$  (3)

where $\mu_i$ is given by the product of the membership degrees of the input variables in the $i$-th rule:

$$\mu_i = LX_1^{(0)}(x_1) \cdot LX_2^{(0)}(x_2) \cdot ... \cdot LX_m^{(0)}(x_m)$$  (4)

2.2.2 The gradient descent algorithm.

If a set of operating data is available that describes the desirable control output, $u'$, for various values of the process-state, $x_1', ..., x_m'$, the fuzzy controller can be optimized by minimizing some criterion on the error between the fuzzy controller output given by (3) and the desired output given by the reference data. Remember that the initially setting of the rules and the parameters $a_{ij}, b_{ij}$ can be extracted from the self-organizing process described earlier. The gradient method cannot warranty a convergence to a global minimum of course, but a quick convergence rate to a good solution can be expected, if an expert chooses a good starting point for the algorithm[14].

By substitution of (2) and (4) into (3), we have an equation for the control output, $u$, in terms of the membership function parameters $a_{ij}, b_{ij}$, $u_i$ $i=1,...,n$ $j=1,...,m$. These are the parameters to be tuned by the optimization procedure. Nomura et al. have chosen to minimize the objective function, $E$, given by:

$$E = \frac{1}{2} (u - u')^2$$  (5)

where $u'$ is the desired control output as given by the reference data and $u$ is the fuzzy controller output, for a particular process state. One of the simplest methods for solving this optimization problem is the steepest gradient descent algorithm. This is a iterative algorithm that seeks to decrease
the value of the objective function with each iteration. It relies on the fact that from any point the objective function decreases most rapidly in the direction of the negative gradient of its parameters at that point. If we have \( E(\mathbf{z}) \) where

\[
\mathbf{z} = [z_1, \ldots, z_p]^T
\]

then this vector is:

\[
[-\frac{\partial E}{\partial z_1}, \ldots, -\frac{\partial E}{\partial z_p}]^T.
\]

If \( z(k) \) is the value of the \( i \)-th parameter at iteration \( k \), the steepest descent algorithm seeks to decrease the value of the objective function by modifying the parameters value via:

\[
z_i(k+1) = z_i(k) - K \frac{\partial E(z)}{\partial z_i}, \quad i = 1, \ldots, p
\]

where \( K \) is a factor which controls how much the parameters are altered at each iteration. Choosing a suitable \( K \):

\[K = K_0 \left( \frac{\partial E(z)}{\partial z_i} \right) \]

can be difficult. As the iterations proceed, the objective function converges to a local minimum. The initially setting of the rulebase, which can be defined by the self-organizing clustering algorithm helps finding a global minimum.

In this case, the objective function parameters to alter are the membership function parameters:

\[
\mathbf{z} = [a_i, b_i, u_i]^T.
\]

Thus \( z_i \) is changing in the opposite direction of:

\[
\frac{\partial E}{\partial z_i} = \frac{\partial E}{\partial u} \cdot \frac{\partial u}{\partial z_i}
\]

where

\[
\frac{\partial u}{\partial z_i} = \frac{\partial u}{\partial a_i} \frac{\partial a_i}{\partial z_i} + \frac{\partial u}{\partial b_i} \frac{\partial b_i}{\partial z_i} + \frac{\partial u}{\partial u_i} \frac{\partial u_i}{\partial z_i}
\]

(7)

\[
\frac{\partial u}{\partial z_i} = \frac{\partial u}{\partial a_i} \frac{\partial a_i}{\partial z_i} + \frac{\partial u}{\partial b_i} \frac{\partial b_i}{\partial z_i} + \frac{\partial u}{\partial u_i} \frac{\partial u_i}{\partial z_i}
\]

(8)

(9)

Formula (7) and (8) show us that the functions containing the tuning parameters need to be differentiable as mentioned before. Substituting (3) and (4) into (5) gives the objective functions in terms of the membership functions:

\[
E = \frac{1}{2} \left( \frac{1}{\sum_{i=1}^{n} \sum_{j=1}^{m} (\Pi(LX_j^h(x_i^j), u_i)^2 - u^2 \right)
\]

where

\[
\Pi(LX_j^h(x_i^j), u_i)
\]
The steepest descent algorithm gives the following iterative equations for the parameter values:

\[
\begin{align*}
\theta_j(k+1) &= \theta_j(k) - \Delta \theta_j(k), \\
\theta_{ij}(k+1) &= \theta_{ij}(k) - \Delta \theta_{ij}(k), \\
\theta_{ij}(k+1) &= \theta_{ij}(k) - \Delta \theta_{ij}(k), \\
\theta_{ij}(k+1) &= \theta_{ij}(k) - \Delta \theta_{ij}(k),
\end{align*}
\]

2.2.3 Neural structure for fuzzy inference.

We will discuss the structural equivalence between the Sugeno's type of learning fuzzy systems with trainable radial basis function neural networks which are both universal approximators. This equivalence provides new and deeper insights into the approximation aspects of fuzzy modelling (identification). It suggests the selection of a certain type of fuzzy operators, namely the algebraic product operator, and a certain type of inference and defuzzification scheme, namely the 'center of singletons' algorithm, because for this type of fuzzy systems, good approximation qualities are guaranteed by the equivalence to a radial basis function network, whose mathematical properties are firmly established. The equivalence between the learning fuzzy system with trainable radial basis function networks is also of considerable importance for the neural network community because it makes it possible to give a linguistic interpretation to the network, providing insight in the physical nature of the system, subject to modeling, that cannot be derived from a black-box neural network[15].

The idea is to map the fuzzy structure into a neural network of a specific structure and to train this network using a well known error correction training method. The learning process may be viewed as a 'curve fitting' problem. The network itself may be considered simply as a nonlinear input-output mapping. Such a viewpoint then permits us to look on generalization as the effect of good nonlinear interpolation of the input data. When however the network learns to many specific training relations, the network may memorize the training data and therefore be less able to generalize between similar input-output patterns (x,u)[8].

Fig.5: represents the neural structure which is proposed here to map the fuzzy forward inference to a radial basis function network. This neuro-fuzzy scheme consists in four layers.

The first layer:
The input layer is made up of source nodes (sensory units) which are the controller inputs \(x_1, x_2, \ldots, x_m\) and has dimension \(m\).

The second layer:
The first hidden layer, is composed of neurons with gaussian activity functions (nonlinear), the membership functions of the antecedents variables:

\[
\bar{x}_{ij} = e^{-\frac{(x_j - c_{ij})^2}{\sigma_{ij}^2}} \quad n=1, \ldots, n_i; \quad j=1, \ldots, m
\]

which are determined by the mean value \(c_{ij}\) and the variances \(\sigma_{ij}\). This layer performs the fuzzification of crisp network input values \(x_j\).

Thus the neuron output is interpreted as a grade of membership, the true value of the associated linguistic variable and has dimension:

\[
p = \sum_{j=1}^{m} n_j
\]

Given: \(n_{x_j}\) the number of fuzzy sets defined for each input variable \(x_j\).
The third layer:
The second hidden layer, is composed of a number of units equal to the rules. It represents the rule layer in which the logical 'and' operator (product) is implemented and the antecedent possibilities are aggregated. The output of the rule layer represents the rules fulfilment degrees and has dimension n. Given n the number of fuzzy rules.

The fourth layer:
The output layer, performs the accumulations of the conclusions (the center of singletons 'defuzzification') to achieve a crisp output (3). The output weights represent the crisp linguistic values $u_i$. These parameters (weights) are learned during the training phase.

The neurons in the first hidden layer are used to partition the input space into regions, so local features are extracted in the first hidden layer. A neuron in the second hidden layer combines the outputs of neurons in the first hidden layer operating on a particular region of the input space.

The transformation from the input space to the hidden layer space is nonlinear, whereas the transformation from the hidden layer space to the output space is linear. A mathematical justification for so doing is provided by Cover's theorem on the separability of patterns, which states that a complex pattern-classification problem cast in high-dimensional space nonlinearly is more likely to be linearly separable than in a low dimensional space - hence the reason for making the dimension of the hidden unit space, n, in a radial basis function network high. The important point to note here is that, given a set of patterns $\chi$ in an input space of arbitrary dimension m, we can usually find a nonlinear mapping $\mu_i(x)$ for $i=1$ to $p$ of high enough dimension n such that we have linear separability in the $\mu$ space. In a similar fashion we may use a nonlinear mapping to transform a difficult nonlinear approximation problem into an easier one that involves linear approximation.

There are different learning strategies that we can follow in the design of a radial basis function network, depending on how the centers of the radial basis functions are specified.

The simplest approach is to choose the locations of the centers randomly from the training data set. This is considered to be a 'sensible' approach, provided that the training data are distributed in a representative manner for the problem at hand.

In the second approach, the radial basis functions are permitted to move the locations of their centers in a self-organized fashion using the Kohonen network, as described earlier. It is important to realize that in the case when the number

$$P = \sum_{j=1}^{m} a_j \gamma_j$$

positions $c_{j\gamma}$, and shape $\sigma_{j\mu}$ of input membership functions are fixed before the learning,

the problem of approximation is linear in the parameters $u_i$ (the rule conclusion weights) that are the subject of learning. In this case the solution boils down to the pseudo inversion of one rectangular matrix. Suppose the referential dataset contains p datapairs $(\chi, u')$ then:

$$[u_1 \cdots u_n][\mu(k) \cdots \mu(k+p-1)] = [u'(k) \sum_{i=1}^{n} \mu_i(k) \cdots u'(k+p-1) \sum_{i=1}^{n} \mu_i(k+p-1)]$$

where

$$u'=[u_1 \cdots u_n]^T,$$

the linear weight vector containing the parameters that are the subject of learning,

$$\mu(k)=[\mu_1(k) \cdots \mu_n(k)]^T,$$

the rules fulfillment degrees for the k-th datavector $\chi(k)$ from the referential dataset. Let M denote an n-by-p matrix,

$$M=[\mu(k) \cdots \mu(k+p-1)].$$

We call this matrix the generalization matrix.

$u'(k)$ is the referential control action belonging to the datavector $\chi(k)$ and

$$x'=[u'(k) \sum_{i=1}^{n} \mu_i(k) \cdots u'(k+p-1) \sum_{i=1}^{n} \mu_i(k+p-1)]$$
the 'normalized' referential (desired) response vector. We may then rewrite eq.(10) in the compact form:

$$w^T M = r^T$$

and solve eq.(11) for the weight vector $w$ to obtain a least squares solution:

$$w^T = r^T M^T (M M^T)^{-1}$$

In the third approach, the centers of the radial basis functions and all other free parameters of the network undergo a supervised learning process. A candidate for such a process is the gradient descent method described earlier[8].

3 On-line tuning of a Fuzzy controller

Suppose the process to control indicated in fig.6, the objective being to tune the control parameter $u$ to an unknown value $u_0$ such as to drive the process output $y$ to a desired value $y_d$. In this case the controller can learn from an error measure (supervised learning).

The main difficulty is that we do not directly know the control error $(u-u_0)$, but only the error in the goal space. Thus, on-line learning differs from case 1 by the fact that the controller no longer learns from input-output training pairs but from a direct evaluation of the controller accuracy with respect to the output of the process.

The operation principle of this adaptive controller is quite simple. In every sampling interval, the controller calculates a control action ($u$). The effect of this control action is analyzed by the adaptation law with respect to the current process behaviour. Then the adaptation law proposes a change of the control action, which is used by the fuzzy controller to adapt its free parameters ($a_{ij}, b_{ij}, u_i$). The parameters are adjusted by using a gradient descent method. In this configuration the controller controls and learns to control simultaneously. Thus the controller has to learn on-line. Additionally the learning has to converge quickly, because it takes place in a closed-loop environment, i.e., the control actions are fed into the process even during the learning progress. There is not a specific training stage during which the controller is not operational. The fuzzy controller learns directly on the domain of relevant states. The network learns to give the control signals so that the output error of the system is reduced according to the optimization criterion used by the gradient descent algorithm.

3.1 Optimization criterion

The fuzzy controller will learn a control law that is dependant on the expression of the criterion. The choice of one or more performance measures depends on the type of response the control system designer wishes to achieve. The definition of this criterion allows to choose the convergence trajectories towards the goal state in the phase plane. The criterion proposed by De Geest et al. (regulator problem) i.e. is written in formula (12). The global minimum of this error function is described in equation (13).

$$E = \frac{1}{2} \cdot (a_p y(k) + b_p y(k))^2$$

$$E = \frac{1}{2} \cdot (a_p y(k) + b_p y(k))^2$$
Then the learning procedure forces the controllers parameters to converge in a state allowing to move along the sliding line. The convergence towards the goal state (0,0) is assured by the internal dynamic of the mechanical process. The choice of the two factors $n_y$ and $n_x$ allows us to fix the convergence speed in the phase plane along the sliding line[16].

We choose to minimize a quadratic function $E$ similar to the previous chapter, given by:

$$E = \frac{1}{2}(y - y_0)^2$$

### 3.2 Parameter adaptation strategy

In contrary to equation (6) the parameters update values can be determined by the formula reported below (14):

$$\frac{\partial H(\dot{y})}{\partial z_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial z_i}$$

(14)

$\frac{\partial u}{\partial z_i}$ corresponds to the gradient method derived in the previous chapter in case of Sugeno's fuzzy system while $\frac{\partial E}{\partial y}$ is analytically known.

The difference with the classical adaptation law as described in case 1 lies in the use of the sensitivity functions of the system to its input signal, $\frac{\partial y}{\partial u}$.

So, to make specialised learning possible, we need some prior knowledge on the way the plant reacts to slight control modifications. The computing of these jacobians introduce another approximation problem because they are hardly analytically known.

One possible strategy consists in approximating the partial derivative by plotting the process reactions to slight control modifications at the operating points.

#### 3.2.1 Specialised learning

In many circumstances a satisfactory control can be attained on basis of the least knowledge of the process which means replacing the jacobian value by the qualitative knowledge of its sign[17]. Indeed the gradient method indicates that the adjustment of $z_i$ follows the sign and amplitude given by:

$$\frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial z_i}$$. Since substituting $\frac{\partial y}{\partial u}$ by its sign ($\pm 1$) just alters the amplitude not the direction of the variation, the gradient method can still give a adequate value of the control parameters. The process can remain a black-box with just few holes allowing to observe qualitatively the way it reacts to increase and decrease of its input. This type of knowledge is quite common and certainly the first shallow type of knowledge a user or observer of the process is aware of and able to communicate. Reducing to such minimum the knowledge of the process necessary to its control makes the control direct.

#### 3.2.2 Controller/Emulator Learning

Another recent and sophisticated technique has been proposed which incorporates the process knowledge in a black-box emulator of the process and links the fuzzy controller to this emulator of the plant. This technique demands a preceding learning stage, the identification of the process and makes it possible to control processes that have jacobians with unknown varying signs[17]. The control structure is shown in fig.7. It consists of two parts: the neural emulator of the plant and the fuzzy
controller. In this configuration an error in the goal space can be back-propagated to an error in the control space using a gradient descent method.

The method corresponds to the classical indirect control approach where the parameters of the process are estimated at any time instant and the parameters of the controller are adjusted assuming that the estimated values of the process represent the true values. The prior knowledge of the dynamic

\[ x(k+1) = \begin{bmatrix} x_1(k) & \cdots & x_n(k) & u(k) \end{bmatrix} \] as follows [18]:

**Input units:**
The inputs of the emulator are the system state \( x(k) \) and the input signal \( u(k) \). A bias is added which improves the capabilities of the network.

**Hidden units:**
\[ y_m = V_y \]

where the rows of the matrix \( V \) are the weights:

\[ V_g = [v_{0j} \cdots v_{mj}] \quad \text{for } j=1 \cdots l \]

connected to hidden unit \( j \). Every element of \( y_m \) is the input of a nonlinear function, a biased and rescaled version of the sigmoid function:

\[ y_j = f(y_m) = \frac{\gamma}{1 + e^{-\eta}} - \eta \quad \text{for } j=1 \cdots l \]

\( \gamma = \max - \min \); \( \eta = -\min \)

Again a bias is added, so \( y = [1 \ y_1 \cdots y_l]^T \)
Output units:

\[ Z_{o_i} = W^T \]

where the rows of the matrix \( W \) are the weights:

\[ w_{i}^j = \begin{bmatrix} w_{i1}^j & \cdots & w_{im}^j \end{bmatrix} \quad \text{for } j = 1 \cdots m \]

connected to output unit \( j \). Every element of \( Z_{o_n} \) is the input of a nonlinear function, a biased and rescaled version of the sigmoid function:

\[ z_j = \frac{1}{1 + e^{-(x_{i_j})}} - \eta \quad \text{for } j = 1 \cdots m \]

\[ \gamma = \max - \min \quad \eta = - \min \]

The output vector \( z = [z_1 \cdots z_m]^T = [\hat{x}_1(k+1) \cdots \hat{x}_m(k+1)]^T \) consists of estimates of the target vector:

\[ x_i = [x_1(k+1) \cdots x_m(k+1)]^T \]

Then the learning of the neural network for the training data is performed in order to minimize the cost function:

\[ E = \frac{1}{2} ||Z - X||_2^2 \]

The weights are updated using a gradient descent algorithm known as back propagation. See [8,12] for the complete derivation.

\[ \Delta w_{i}^j = -\mu \frac{\partial E}{\partial w_{i}^j} \]

\[ \Delta v_{i}^j = -\mu \frac{\partial E}{\partial v_{i}^j} \]

where \( \mu \) is the learning rate.

Like all the discussed gradient descent methods, the algorithm depends on the instantaneous error surface in weight space, \( E = f(x(k), W(k), V(k)) \). The algorithm is therefore stochastic in nature; that is, it has a tendency to zigzag its way about the true direction to a minimum on the error surface [8]. Consequently, it suffers from a slow convergence property. After training we keep the weights unchanged and expect the emulator to make good predictions of the systems state when given the previous state and input.

It is shown that when a multilayer perceptrons neural network estimates the plant, the error \( (y - y_o) \) in the goal space can be translated into an error in the control space \( u \) by a mechanism of back-propagation applied on the input (the control variable) of the process emulator.

The controller's task is to drive the process output to the desired output and to learn this task from minimizing the error criterion:

\[ E_{con} = \frac{1}{2} (\hat{x}_1 - x_1)^2. \]

For simplicity only the first state variable is part of this output criterion but the whole state may be included. To make parameter adaptation of the controller possible (as described for Sugeno fuzzy systems) we need to know the sensitivity function:

\[ \frac{\partial E_{con}}{\partial u}. \]

This model-based information can be derived by back propagation of the error through the black box emulator [17]:

12
The back propagation part may be rewritten as:

\[
\frac{\partial E_{\text{con}}}{\partial u} = \frac{\partial E_{\text{con}}}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1}{\partial u} = \frac{\partial E_{\text{con}}}{\partial \mathbf{z}_{\text{in}}} \frac{\partial \mathbf{z}_{\text{in}}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{y}_{\text{in}}} \frac{\partial \mathbf{y}_{\text{in}}}{\partial u}
\]

\[
\frac{\partial E_{\text{con}}}{\partial \mathbf{x}_1} = -(x_{1d} - \mathbf{x}_1)
\]

\[
\frac{\partial \mathbf{z}_{\text{in}}}{\partial \mathbf{y}} = \frac{\partial}{\partial \mathbf{y}} W^T \mathbf{y} = [w_{11} \ldots w_{mi}]
\]

\[
\begin{bmatrix}
f'(y_{s1}) & 0 & \cdots & 0 \\
0 & f'(y_{s2}) & 0 & \cdots & 0 \\
0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & f'(y_{s1})
\end{bmatrix}
\]

\[
\frac{\partial \mathbf{y}_{\text{in}}}{\partial \mathbf{y}} = [v_{(m-1)1} \ldots v_{(m-1)i}]^T = v_{(m-1)i}
\]

The back propagation part may be rewritten as:

\[
\frac{\partial E_{\text{con}}}{\partial u} = \delta_{\mathbf{y}}^T \mathbf{y}_{(m-1)i}
\]

\[
\delta_{\mathbf{y}} = \delta_{\mathbf{y}} W^T \frac{\partial \mathbf{y}}{\partial \mathbf{y}_{\text{in}}}
\]

\[
\delta_a = -(x_{a_d} - \mathbf{x}_a) f'(x_{a_d})
\]

where \( \delta \) is the local gradient. The local gradients are distributed, weighted and added to the former layer to translate the output error to the input.

4. Conclusions and Recommendations

The tuning methods discussed in this paper have to be investigated in a simulated environment and through experimental studies. Some theoretical studies have to be performed in order to justify completely the use of the discussed learning techniques. Furthermore, this paper could have a contribution to the Fuzzy Logic course.

References:

controller.", EUFIT '94, pp747-751.


