ELSEVIER Fuzzy Sets and Systems 103 (1999) 27–48

FUZZY sets and systems

A self-organizing fuzzy controller with a fixed maximum number of rules and an adaptive similarity factor

Joana Matos Dias, António Dourado*

CISUC-Centro de Informática e Sistemas da Universidade de Coimbra, Departamento de Engenharia Informática, PÓLO II da Universidade de Coimbra, Pinhal de Marrocos, P 3030, Coimbra, Portugal

Received December 1996; received in revised form April 1997

Abstract

This paper proposes a self-organizing fuzzy controller with a broad generality for minimum phase and stable systems. The controller learns the rules on-line with a minimum knowledge about the process. The rule base is built and permanently actualized from input–output real time data and has a fixed maximum number of rules (FMNR). An (on-line) adaptive similarity factor implements a special efficient inference technique. Feedforward and predictive effect is introduced in fuzzification and defuzzification stages. The defuzzification is carried out in such a way that as the learning process progresses the interval of the control becomes more and more accurate. Results are shown concerning simulations for non-linear SISO, MISO and MIMO systems and a real experimental application using a low-cost microcomputer. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Empirical research; Control theory; Process control; Self-organizing fuzzy control; On-line learning; Applications

1. Introduction

Since the pioneer works of [21, 16] fuzzy control has been growing in importance and many progresses in real applications have been reported [14]. However, for mathematically ill-understood, non-linear, time-varying processes, there is still considerable work to be done. If, for relatively well known processes, the expert operator knowledge allows to build a set of rules capable of implementing a fuzzy controller, for more complicated situations only a self-organizing controller (SOC) can do the job. The main goal is to reduce the dependence on experts and to obtain a control structure that does not need (to some extend) à-priori learning, i.e., the controller itself is (to some degree) able to learn about the controlled system and to adapt itself on-line. This is not a trivial goal and, to reach it, only the synergy of multiple experiences will allow to make real progresses and to converge to a general fuzzy control systems theory with a systematic synthesis methodology.

* Corresponding author. Tel.: + 351 39 7 000 000; fax: + 351 39 701 266; e-mail: dourado@dei.uc.pt.

In that sense several Self Organizing Controllers have been developed in the past; for a review see, e.g. [3, 18]. In more recent years progresses in several directions were reported, as for example on elaborated strategies to adapt the scale-factors [5, 10]. Many discussions and some reviews [10] and some other particular techniques can be found in recent papers. For example in [8] is proposed an adaptive fuzzy controller using several controllers in parallel for the different expected situations. This corresponds in some sense to the classical gain-scheduling techniques, with its limitations- when a non-predicted new situation appears the controller may fail. Also [17] propose a systematic design for fuzzy sliding mode controllers, very useful in control of manipulators.

In a different approach several authors have implemented the self-organization as a neural-network training problem, in a neuro-fuzzy architecture [4, 7, 11, 15] or as a genetic algorithm optimization problem [1, 6]. Usually those approaches require knowledge about the process to initialize the procedure and, because of neural networks training algorithms, they are computationally hard; moreover, they are complex to implement in real time control of dynamic systems where two identical situations are very uncommon (and learning by repetition is not possible).

One of the most important tasks of an SOC is the on-line generation of the rule base – finally the knowledge base of the fuzzy system. Several works have recently been published about it. In [12] it is developed an iterative learning method based on a reference model to ameliorate the control trajectory. Several parameters and matrices must be tuned and the procedure requires repetitive trials in similar conditions, which is impossible to obtain in practical dynamic systems. In [19] is proposed an indirect adaptive fuzzy controller making an on-line nearest-neighborhood clustering of input–output data; the clusters are viewed as sample data and one rule is generated on-line for each input–output (cluster) pair; the method has the advantage of combining if–then rules and numerical input–output data but, besides needing several trial and error techniques and previous learning activity, does not guarantee that the number of rules will remain under an acceptable limit. In [20] is developed a general methodology based on the division of the input and state spaces into cell-groups followed by systematic search (based on multiple process simulations) in order to find the optimal (in some sense) control trajectory. Heuristics, resolution and accuracy requirements, as well as partial knowledge of the process dynamics are needed in order to determine the number of cell groups. However, the search for the optimal control strategy starting in each cell is an intensive computational task needing an acceptable process model. In [13] it is developed a very interesting controller using the history of input–output pairs and proper inference and defuzzification methods; the rules are updated on-line by a self-organizing procedure. The fuzzification is done based on assumed input and output ranges and the number of rules increases monotonically as new input–output data is experienced. The method goes in the same direction as the Cell Sate Space Algorithm of [2]. For the regulation problem the number of rules tends to a steady state number. For the tracking problem, according to our experience, this number grows without bound and so the computational resources will be exhausted after some time.

The SOC developed and experimentally tested in this paper is initially inspired (for the fuzzification and inference stages) in the method proposed for regulation by [13] but it introduces a set of new features: it reduces complexity in implementation, reduces computational costs and solves effectively both the regulation and the tracking problems with low computational resources. It has a wide range of applicability for stable monotone processes. It does not need à priori training and has a rule base constructed on-line with a maximum number of rules, using a Fuzzy Auto Regressive with Exogenous Input (FARX) model. This maximum number of rules is fixed by the programmer giving him a more accurate control over the sampling time. The algorithm experiences an on-line learning phase, where it learns how to control the process for a particular reference, constant or time variant. After this initial learning phase, taking a relatively short time, if the process suffers changes in its dynamics, the algorithm reacts to these changes with a new learning phase without augmenting the number of rules.

The adaptation and learning abilities are shown to be good. Adaptation is introduced in a modified version of the similarity factor of [13] instead of calculating it by trial and error. This improves the speed of response without increasing the settling time. Learning proceeds by replacing an old invalid rule by a new one when changes occur in the process detectable by the controller.

The controller is easily extended in the paper to multiple input single output (MISO) and multiple input multiple output (MIMO) processes.

It was experimented in simulations of SISO (including a non-linear time varying process) MISO and MIMO (a medical application) processes and also in a real experimental process. The presented SOC is able to show good performance in all those tested cases.

The paper is organized as follows. Section 2 presents the problem, the assumption and the FARX equation. Section 3 describes the fuzzification and inference stages. In Section 4 the new SOC is developed, in its several proper characteristics. Section 5 extends the SOC to MISO and MIMO systems. Simulations and practical experimentation are presented in Section 6 and finally Section 7 discusses the potentialities, limitations and further needed developments to enlarge still more broadly the generality of the controller and to give him a higher degree of autonomy.

2. The problem assumptions and the FARX model

2.1. Problem assumptions

The objective of the SOC is to use the minimum possible à priori knowledge about the process. This minimum knowledge needed is given by the following assumptions:

(i) The process can be modeled by an nonlinear auto regressive with exogenous (NARX) input model (1):

$$y(k) = f\,[y(k-1), y(k-2),, \dots, u(k-T), u(k-T-1), \dots, e(k)] \tag{1}$$

with f a linear or non-linear function, time variant or not, $y(k)$, $y(k-1)$, ... the outputs of the system and $u(k)$, $u(k-1)$, ..., the inputs to the system at the k, $k-1$, ... instants. T represents the discrete pure time delay and $e(k)$ is a stochastic disturbance.

(ii) The pure time delay T is known or, at least, its upper bound.

(iii) The reference is known at least $T+1$ sampling instants in advance.

(iv) The memory (order) of the system is known (at least its upper bound).

(v) The relation, one out of two possible, between the change in the input signal and the change in the output signal is known:

(1) The output increases/decreases with the increase/decrease of the input.

(2) The output increases/decreases with the decrease/increase of the input.

If assumption (iii) is dropped, we can still control the system, but a longer learning phase and a not so good performance is expected (particularly in the tracking problem). The main objective of the assumption is to give the controller an anticipative characteristic so that it can deal more efficiently with the pure time delay of the process.

2.2. The FARX model and the rule base

The rule base is initially empty and is constructed on-line. Considering (1), solving for $u(k-T)$, (2) is obtained:

$$u(k-T) = g[y(k), y(k-1), \dots, u(k-T-1), \dots, e(k)] \tag{2}$$

Then the rules format is derived from the linguistic transformation of (2) as shown by (3) in the form of a FARX system (named by analogy with the NARX model of control theory)

$$\text{If } y(k) \textit{ is } A_1 \textit{ and } y(k-1) \textit{ is } A_2 \textit{ and} \ldots \textit{and } u(k-T-1) \textit{ is } B_1 \textit{ and} \ldots$$
$$\textit{then } u(k-T) \textit{ is } C \tag{3}$$

$A_1, A_2, \ldots, B_1, \ldots, C$ are linguistic values and, in the kth iteration, are substituted by "*approximately* $y(k)$", "*approximately* $y(k-1)$", …, "*approximately* $u(k-T-1)$", and so on, with $y(k)$, $y(k-1)$, $u(k-T-1)$, the outputs read from and the inputs sent to the process. Each situation experienced by the process is considered a valid rule.

If we add $T+1$ to each and every index, each rule gives us a value for $u(k+1)$, if the antecedent is true (in a fuzzy way). We consider the future values of the process output the same as the future (known) values of the reference; implicit in this consideration is the assumption that the controller will be capable of making the output follow the reference and so, by this simple way, some feedforward effect is introduced in the control signal.

If assumption (iii) is not fulfilled, then we can consider the future values of the reference equal to its past values. Despite the fact that this is not true for the tracking problem, it is the best information we have. This leads to an increase in the learning period and the controller will no longer be capable of dealing properly with the pure time delay of the process. This behavior is an unsolved (and probably unsolvable) problem. When future values of the reference are not known there is no way to introduce any feedforward effect (only feedback is possible). However, for the regulation problem the controller still behaves well.

3. The fuzzification and inference stages

3.1. Fuzzification

The fuzzification membership functions follow the ones proposed in [13] and are triangular shaped defined by (4):

$$\mu_{xi} = \begin{cases} 1 + \dfrac{x(k) - x_i}{b-a}, & a \leqslant x(k) \leqslant x_i, \\ 1 - \dfrac{x(k) - x_i}{b-a}, & x_i \leqslant x(k) \leqslant b, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

with $x(k) \in [a, b]$. This interval includes all possible values for the variable in cause. All the fuzzy sets have the same support interval $[a, b]$. Every crisp $u(k)$ value defines the unity vertex of a membership function and all the membership functions have the same slope. These facts minimize the quantity of information required to define these functions: this minimum is simply the unity vertex x_i. On the other hand the rule base is always complete.

3.2. The inference mechanism with feedforward effect

In [13] a very interesting defuzzification method is developed, taken as a starting point for the proposed SOC. In usual defuzzification stage the min operator is applied to the antecedents membership degrees and the consequent membership function is processed accordingly (e.g. by α-cut). This may become very unsatisfactory particularly when the number of inputs increases (as is the case of most practical problems in dynamic systems) because relevant input variables will always be ignored. Instead of the min operator, these authors proposed a compounding function of all input variables by the Euclidean distance between the newly measured crisp input variable and the vertices of the existing membership functions.

Considering, for instance, a 1st order process each rule has the format: "*If* $y(k)$ *is approximately* y_{1i} *and* $y(k-1)$ *is approximately* y_{2i} *then* $u(k-T)$ *is approximately* u_i". Now, and giving in the presented SOC a feedforward content to the computation of the control signal, if the future values of the reference are R_1 and R_2 (respectively in $k+T+1$ and $k+T$ sampling times), the degree of equality ω_i between them and each rule antecedent is calculated by (5) and by the similarity function (6):

$$\text{(i)}\ D_i = \sqrt{(y_{1i}-R_1)^2+(y_{2i}-R_2)^2}, \tag{5}$$

$$\text{(ii)}\ \omega_i = -\frac{1}{Dmax} * D_i + 1. \tag{6}$$

The *Dmax* parameter, called here the similarity factor, is introduced in [13] by a trial and error procedure. However, as will be shown later, it is more convenient to make it adaptive (on-line). If D_i is greater than *Dmax* then the rule is ignored because it is considered that there is no similarity between the rule and the desired situation.

The consequent fuzzy set for each rule is an interval of values, [p1, p2], where

$$\mu_{ui}(u(k)) \geqslant \omega_i, \quad \forall u(k) \in [\text{p1}, \text{p2}]. \tag{7}$$

The total output is taken as the intersection of all the intervals (one for each rule). So, at the end of the inference mechanism at instant k, we have an interval of possible values for $u(k+1)$:

$$\Delta u(k+1) = [\min(k), \max(k)]. \tag{8}$$

The justification for the choice of the intersection operator can be found in the theory of fuzzy equality relations [9].

4. The proposed SOC

In the method proposed in [13], it is necessary to perform a discretisation of the output variable domain and, for systems of higher order than the 1st, also of the input variable domain. The discretisation of the output should be done accordingly to the desired reference, in order to achieve better performance. When the discretisation of the input variable domain is needed, it should be done uniformly throughout all the domain. What we experienced was that, for tracking problems where the references can have broad amplitudes, the discretisation required to achieve good results leads to a too high number of rules. This increases prohibitively the computational resources. For more complex processes than a second order one, the number of needed rules is prohibitive.

The method presented in this paper does not require any kind of discretisation and has a rule base with a fixed maximum number of rules (FMNR).

4.1. The on-line construction of the rule base

At every sampling time an input signal is sent to the process and an output signal is received from it. Based on these new values and on the values memorized, the rules are constructed accordingly to the experiments (every experiment is considered a valid rule).

Each rule is continuously added to the rule base until the FMNR is reached. At this point the new rules are written over the older ones, so that the FMNR is never exceeded. This gives the algorithm the characteristic of easily forgetting one system condition and adapting to a new one. The technique of fixing the maximum number of rules was issued from empirical research and extensive experimentation, and in order to avoid the explosion in size of the rule base. It is effective if the FMNR is chosen properly.

4.2. The choice of the FMNR

The choice of the FMNR must be considered with care. The membership functions in the rule base must span over all the domain of the input and output signals for the desired reference (by this way the controller will always know what to do). As the implementation is in discrete digital form, the reference is represented by a discrete number of different situations. This number is a function of the desired sampling time and of the reference itself; the following principles give a guide line to calculate FMNR:

(i) If the reference is a periodic signal with period P and the sampling time is T_s, then FMNR $= P/T_s$. By this way every input–output pair will produce a new rule.

If the reference is constant (a step), its period is infinite and this guideline cannot be applied. In principle (in ideal conditions) the controller should only need one rule, at least in the long-run. In practice this is not possible because it has to learn how to control the process even in the case of external or internal disturbances; in these cases more rules are needed to control the "new" process. What is done in this case is to consider a number of rules compatible with the desired sampling time. Generally, the (re)learning time decreases with the increase in the number of rules. In the long-run, and if no changes occur, all the rules in the rule base will be similar.

(ii) If the FMNR that results from applying (i) is too high for the desired T_s (the computational time needed is directly related to the number of rules), then one must increase the time interval between two consecutive updates of the rule base. In this case the rule base will have less information about the process behavior but will still store information about different situations during one reference period. The FMNR calculated in (i) is divided by n, $n \in N$, and the situations experienced by the system are included as new rules just when the number of the sampling instant k is divisible by n.

(iii) If we want the system output to follow successively two periodic references with different mean and amplitude values, so that every m sampling intervals the desired reference changes from one periodic function to another, one must have a rule base that is able to store all the needed information. If different spaces (in the rule base) are not defined for these two periodic references, what happens is that the rule base only remembers the last reference and a new learning phase is needed every time the reference changes. If one structures the rule base such that each different reference has its own set of rules, the set regarding the non-active reference will not be updated (overwritten) when the other reference is active. The controller will only experience two learning intervals, one for each reference. At each instant k only the set of rules of the active reference will be used. By this way the total size of the rule base will increase, but not necessarily the computational time, since only a sub-set of the rule base will be used at each instant k.

(iv) If the reference is not periodic, the rule base is constructed, when this is feasible, in a way similar to (iii), using all the information available about the reference, particularly its domain. When this is not feasible, the maximum number of rules is fixed and the rule base is permanently refreshed. This is the most difficult situation because the controller needs the rule base to be illustrative of the reference domain, and this cannot be guaranteed in this case; as a consequence some trial and error must be done to fix the FMNR. Fortunately this is an unusual situation.

4.3. If the intersection results in an empty set

In the inference mechanism, one has to take the intersection of all the consequent intervals, one for each rule. It is possible that this intersection results in an empty set. This can happen with a single rule whose interval is disjunctive with all the others. In this case, the best thing to do is just ignore that rule. The conclusion reached by the experiences made was that this rule is insignificant, it illustrates a transitory state of the system and should not be taken into account.

4.4. The defuzzification method

The output of the inference mechanism is an interval $[\min(k), \max(k)]$ (8). Now a crisp value for $u(k+1)$ must be computed.

In the present proposed algorithm, a simple but efficient method is derived, using the concept of *error_in_advance* which is the difference between the actual output and the desired reference after the pure time delay T of the process plus one:

(i) *error_in_advance* = *reference*$(k+T+1) - y(k)$.
(ii) Now considering relation (1) of assumption (v), then (9) and (10):

if *error_in_advance* > 0 then

$$u(k+1) = \tfrac{1}{2}[\max(k) + u(k-T)] \tag{9}$$

$$\text{else } u(k+1) = \tfrac{1}{2}[\min(k) + u(k-T)]. \tag{10}$$

Considering relation (2) of assumption (v), then (11) and (12):

if *error_in_advance* > 0 then

$$u(k+1) = \tfrac{1}{2}[\min(k) + u(k-T)] \tag{11}$$

$$\text{else } u(k+1) = \tfrac{1}{2}[\max(k) + u(k-T)]. \tag{12}$$

The $\min(k)$ and $\max(k)$ quantities are those from (8).

If assumption (iii) (Section 2) is not fulfilled, one can use the error instead of the *error_in_advance* and the value for $u(k)$ instead of the value for $u(k-T)$.

In this way the pure time delay of the process tends to be compensated since the control signal acquires the predictive characteristics intrinsic in the *error_in_advance*. In fact $u(k-T)$ is the most recent input influencing $y(k)$. The input $u(k+1)$ will only influence the output $y(k+1+T)$ (and at instant $k+1$ nothing can be done to improve the outputs before $k+T+1$); this fact leads to the definition of the *error_in_advance*. $u(k+1)$ must then depend on the *error_in_advance*. Now to compute the control $u(k+1)$ the expressions (9)–(12) calculate the average between the past value $u(k-T)$ and a limit of the inference interval (8) according to the signal of the *error_in_advance* and also according to the signal of the monotonicity of the process.

4.5. The adaptive degree of similarity

The *Dmax* parameter represents the maximum distance considered that allows ω_i to be positive. This parameter not only determines which rules will contribute with an interval for the final outcome, but also influences the calculation of the interval.

If *Dmax* has a value that is too low, then the system output will oscillate around the reference because few rules will be used and the interval (8) will be large, and as a consequence the control signal tends to oscillate. If its value is too high, the system output will have steady error because too many rules are used, even some that are far away from the actual situation. Observing the behavior during a time window, these two situations are identified, and the *Dmax* parameter becomes an adaptive parameter. Its value is increased proportionally in the presence of the first situation (the highest observed amplitude) and decreased proportionally in the presence of the second:

$$Dmax = Dmax * (1 + percentage\ of\ oscillation) \tag{13}$$

if the *Dmax* has to be increased;

$$Dmax = Dmax * (1 - percentage\ of\ error) \tag{14}$$

if the *Dmax* has to be decreased.

The time window is a number of sampling intervals during which the controller surveys the output to see if it is oscillating around the reference or has steady error. In the case of periodic references, the time window can be a fraction of the period. The shorter the time window, the oftener the *Dmax* parameter will be adapted, resulting in a shorter learning interval. However a too short time window may disable the correct assessment of the situation (if, e.g., the error is oscillating and the time window is short compared with this oscillating period, the controller may erroneously conclude that the situation is of steady error).

Experiences were made to adapt on-line the width of the time-window. This only improved the complexity of the SOC and it was shown to be unnecessary for the cases treated. However this is a point of further research.

4.6. Tracking changes in the process parameters

In the presence of changes in the process parameters, particularly heavy ones, the previous algorithm does not react immediately in order to overcome this problem. It waits until all the rules in the rule base are illustrative of the new situation. This drawback of non-reaction period of the controller can be overcome if the calculation of D_i (now D_i^*) at instant k is complemented by (15) after computation of (5):

$$D_i^* = \tfrac{1}{2}\left[D_i + \sqrt{(y_{1i} - y(k))^2 + (y_{2i} - y(k-1))^2 + (u_i - u(k-T))^2}\right]. \tag{15}$$

The expression (15) considers the distance between each rule and the desired reference in D_i (as does (5)), but also the distance between each rule and the present situation of the process output (as opposed to (5)). If there is a change, for instance, in the process dynamics such that the present rule base does no longer illustrate the process behavior, the value of D_i^* will rapidly increase due to (15), increasing by consequence the width of the interval calculated in the inference stage. As a consequence the controller will immediately enter in a new learning phase.

4.7. Initialization and other aspects

The initial value of *Dmax* and of the observation window width are two parameters that influence the performance of the controller. At start-up, in the beginning of the learning phase, there are few rules and the process output is still far away from the reference, resulting in a high value of D_i in (5). To make ω_i in (6) not a very low value, *Dmax* must be high.

The time window cannot neither be very short (to allow a good evaluation of the error) nor very large (to allow a frequent adaptation of *Dmax*).

The performance of the algorithm, at least in its early iterations, can also be influenced by other factors, like the interval considered as being the domain of the input variable (which is unknown at start but can be guessed from the actuator range and the reference). The smaller the interval, the faster is the convergence of the algorithm.

If, in the learning phase, the process output oscillates in an unreasonable way (over some threshold), the control signal will also tend to oscillate unreasonably. In this case the defuzzification mechanism can be

Set the observation window to 20 sampling intervals, the initial *Dmax* to 60 and $n = 2$. Fig. 1 shows the results. In the beginning of the start-up learning phase, until $k = 10$, the control signal shows some oscillation but then becomes rather smooth. After some time, the SOC makes the output of the system rapidly follow the reference. In the long run, the two signals become practically indistinguishable. Although the rule base is full only at $k = 250$ (since $n = 2$), and at start it is empty, the performance is good after $k = 100$. After $k = 250$ the rules are replaced on-line, with a progressive performance increase.

Fig. 2 shows the control signal. It is becoming smoother and smoother as time goes on. The input interval defined by (8) is shown in Fig. 3. During the initial instants the value of $u(k)$ calculated by (9)–(12) may exceed $\max(k)$ since only $u(k+1)$ is influenced by $\max(k)$. It is interesting to see how quick is the convergence of this learning interval. Fig. 4 shows the evolution (as a consequence of the adaptation) of *Dmax*.

In [19], with the on-line nearest-neighborhood clustering technique, the tracking performance is lower than here after the learning phase. Considering complexity and performance, the presented SOC is superior.

Case 2: *Regulation of a time variant non-linear system*

Considering now two continuous non-linear systems (25)–(26) from [20]

$$\begin{cases} \dfrac{\mathrm{d}x_1}{\mathrm{d}t} = x_2 \\ \dfrac{\mathrm{d}x_2}{\mathrm{d}t} = -9.25x_1 - x_2 - 0.1x_1^3 + 9.25u \qquad \text{until } t = 40\,\text{s} \\ y(t) = x_1(t - 0.1) \end{cases} \tag{25}$$

and

$$\begin{cases} \dfrac{\mathrm{d}x_1}{\mathrm{d}t} = 1.5x_2 \\ \dfrac{\mathrm{d}x_2}{\mathrm{d}t} = -11.25x_1 - x_2 - 0.1x_1^3 + 9.25u \qquad \text{after } t = 40\,\text{s} \\ y(t) = x_1(t - 0.1) \end{cases} \tag{26}$$

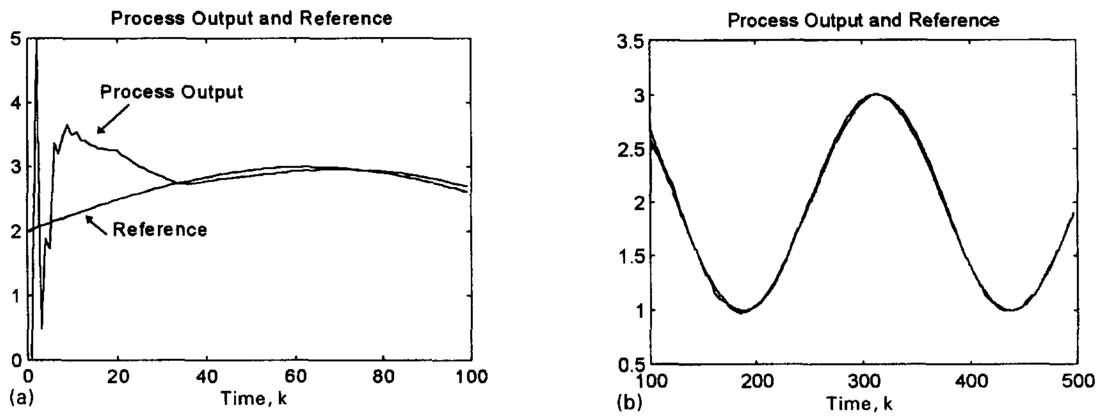


Fig. 1. Case 1. (a) Start-up learning phase. After 40 instants the control is already acceptable. (b) The control performance after $k = 100$ is very good. Only at $k = 250$ the rule base is full.

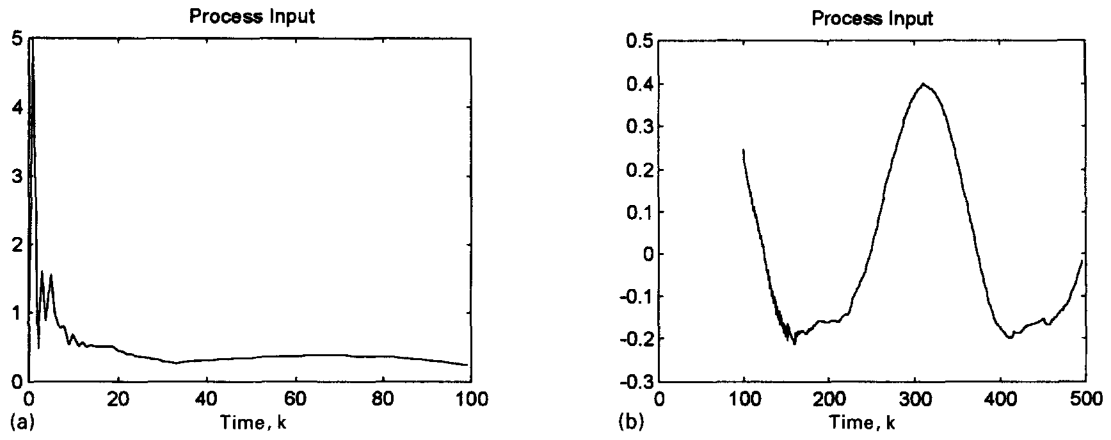


Fig. 2. Case 1. The control signal (observation window is 10). (a) At start-up; (b) on long-run.

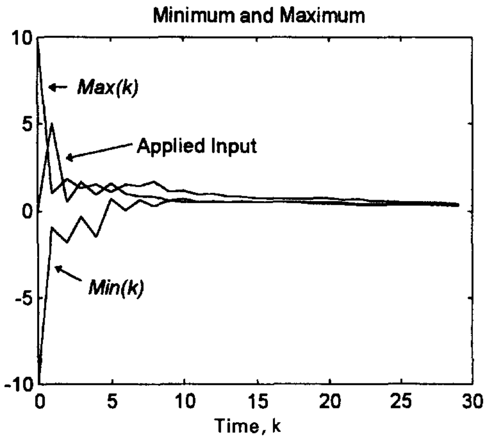


Fig. 3. Case 1. The learning interval (8) and the applied control input during the learning phase.

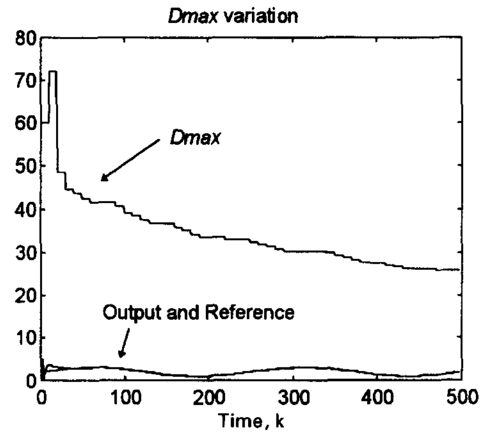


Fig. 4. Case 1. The evolution of *Dmax*.

representing a time varying system (the coefficient of the first equation and one coefficient of the second) with two states x_1 and x_2.

A sampling time of 100 ms and an FMNR of 40 are considered and one controls firstly the system (17) and latter the system (18), with a constant reference with amplitude 5.

Set the time window to 20, initial $Dmax = 60$ and $n = 1$.

The results are shown in Fig. 5. The start-up learning is good. When the system dynamic changes, at $k = 400$, there are some damped oscillations during the (re)learning phase.

Now if the time window is increased to 40 (maintaining *Dmax*), the results are shown in Fig. 6. The response shows some oscillations at start-up and the learning phases are longer. This is in accordance with the frequency of adaptation of *Dmax*, as shown in Fig. 7. The shorter is the observation window, the better is the *Dmax* adaptation. However if the window is too short, resulting in an *overadaptation*, then the performance may be degraded and the closed loop system may become unstable, as was observed in some not shown runs.

changed in the following way: while the oscillation is over that threshold, the calculation of the control value is made with a weighted average of the value issued from (9)–(12) with past control values, where the heaviest weight is put on the preceding values of the input variable instead of on (9)–(10) or (11)–(12). This can lead to an increase in the learning phase time but will result in a much smoother control signal because the learning interval, that changes abruptly, contributes less to the averaged control signal.

5. Development for MISO and MIMO systems

As the FMNR in the rule base does not depend on the complexity of the system, the computational time will not increase drastically when considering MISO or even MIMO systems.

5.1. MISO processes

In the case of an MISO process, it is necessary that the assumption (v) (Section 2) be fulfilled for each and every input.

Supposing for example a zero order system with two inputs $u1$ and $u2$, each rule, for calculating the control signals $u1$ and $u2$ is in the form:

If $y(k)$ *is approximately* y_i *then*

$u1(k-T1)$ *is approximately* $u1_i$ *and* $u2(k-T2)$ *is approximately* $u2_i$ (16)

with $T1$ and $T2$ pure time delays relative to inputs $u1$ and $u2$, respectively.
Each rule can be interpreted in two different ways:

(a) Splitting the rule into two rules

If $y(k)$ *is approximately* y_i then $u1(k-T1)$ *is approximately* $u1_i$. (17)

If $y(k)$ *is approximately* y_i *and* $u1(k-T1)$ *is approximately* $u1_i$

then $u2(k-T2)$ *is approximately* $u2_1$. (18)

In this case the value for $u2(k+1)$ is calculated knowing the value for $u1(k+1)$.

(b) Splitting still the rule into two rules but in a different way:

If $y(k)$ *is approximately* y_i then $u1(k-T1)$ *is approximately* $u1_i$. (19)

If $y(k)$ *is approximately* y_i then $u2(k-T1)$ *is approximately* $u2_i$. (20)

In this case $u1(k+1)$ and $u2(k+1)$ are calculated independently of one another.

The results obtained using (a) or (b) were similar. As the first alternative requires a longer calculation time, the second alternative was chosen. If we think of a system with n input variables, the first alternative would require that each rule would be used n times. This represents an enormous increase in the computational time.

5.2. Case of MIMO processes

Some changes have to be made to control MIMO systems with this algorithm. The assumptions are maintained, with the only difference that assumption (v) (Section 2) must be fulfilled for all the possible input–output pairs. The additional problem to handle is the coupling between input and output variables.

Supposing, for illustrative purposes, a zero-order system with two inputs and two outputs, each rule is:

If $y1(k)$ *is approximately* $y1_i$ *and* $y2(k)$ *is approximately* $y2_i$

then $u1(k-T1)$ *is approximately* $u1_i$ *and* $u2(k-T2)$ *is approximately* $u2_i$ (21)

Each rule is interpreted as two independent rules:

If $y1(k)$ *is approximately* $y1_i$ *and* $y2(k)$ *is approximately* $y2_i$

then $u1(k - T1)$ *is approximately* $u1_i$. (22)

If $y1(k)$ *is approximately* $y1_i$ *and* $y2(k)$ *is approximately* $y2_i$

then $u2(k - T2)$ *is approximately* $u2_i$. (23)

The changes in the algorithm occur in the inference mechanism and in the defuzzification mechanism.

5.2.1. *The modified inference mechanism*

The most important parameter in this stage is the *Dmax* parameter. Its adaptation is done based on the performance of the output of the system. In the case of a MIMO system, there are more than one output. Then we consider as many *Dmax* parameters as there are outputs. Each of them is adapted according to the performance of one particular output. As we need the *Dmax* parameter to calculate the input interval, we make a correspondence between inputs and outputs, so that the calculation of an interval for a particular input variable is done with the *Dmax* adapted according to the output that corresponds to that input. This correspondence is made in an empirical way. If we have extra knowledge about the system, we can use that information to make this input–output pairing. It is not necessary that the MIMO process has as many inputs as outputs. One output can have correspondence with more than one input. Each of these inputs uses the same value of *Dmax* from the calculation of (5) or (15).

In the inference mechanism it is possible that we have to calculate several values for D_i (possibly as many as input variables), depending on the pure time delays of each input variable with respect to each output variable (since in (5) the pure time delay of each channel must be considered).

5.2.2. *The modified defuzzification method*

After the inference mechanism, we have one interval for each input variable. The defuzzification procedure is then constituted by a set of rules that decide which value will be chosen for each input variable. This set of rules transmits to the algorithm the knowledge we have about the relations between the input and output variables in the sense of assumption (v) of Section 2. This will be illustrated later by an example.

There is not a unique set of rules for the defuzzification method. Once again, if we have extra information about the process (as, e.g., the relative sensitivities between each input and each output), it can help us to find a good set of rules. If not, we should try several sets and see which has better performance.

6. Results and discussion

The controller was tested in simulations and in a real implementation.

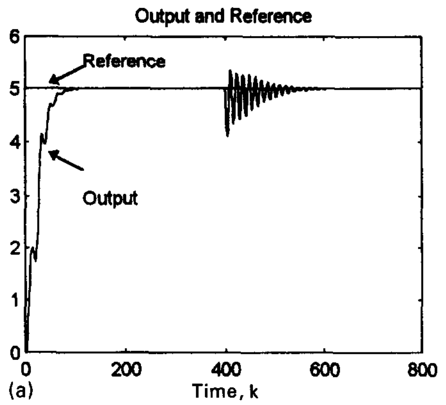
6.1. *Simulations on SISO systems*

Case 1: *Tracking, non-disturbed system*

Consider the non-linear system from [19]

$$y_k = \frac{y(k-1)\,y(k-2)\,(y(k-1)+2.5)}{1+y^2(k-1)+y^2(k-2)} + u(k-1) \tag{24}$$

to be controlled with a sinusoidal reference with period 25 s, and a sampling time of 100 ms, with an FMNR equal to 125 (by the principle 2 of 4.2, with $n = 2$).



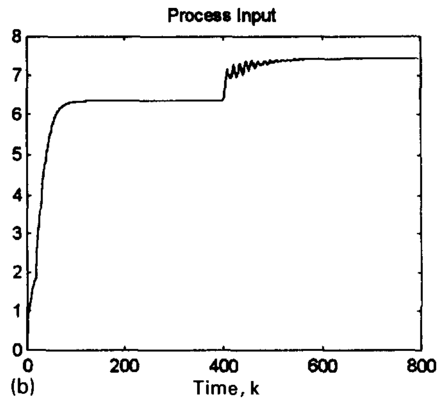
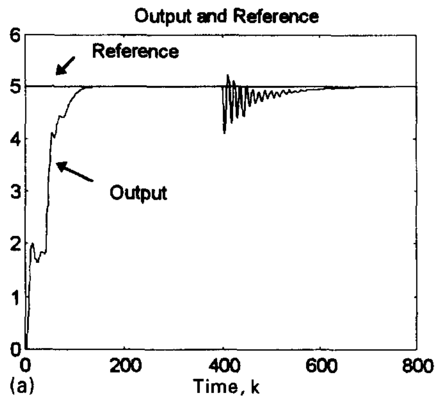


Fig. 5. Case 2. (a) Process output and reference. (b) Process input. Window = 20, initial *Dmax* = 60.



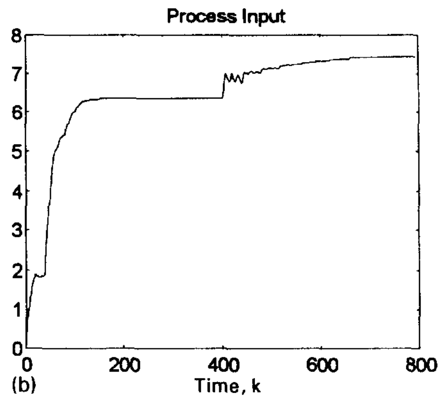
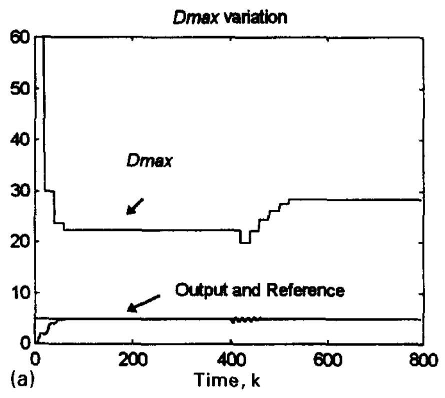


Fig. 6. Case 2. (a) Process output and reference; (b) Process input. Window = 40, initial *Dmax* = 60. The learning phases are longer than in Fig. 5.



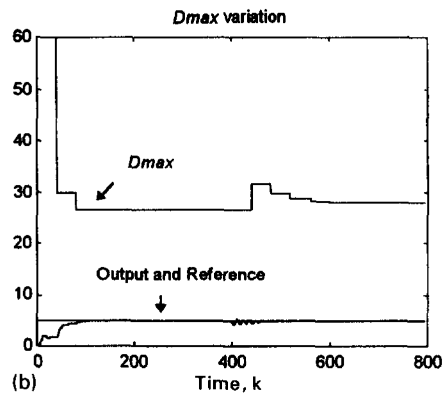


Fig. 7. Case 2. *Dmax* variation with initial *Dmax* = 60. (a) Window = 20; (b) Window = 40.

For a window of 20 and an initial *Dmax* equal to 30 or 100, the results are shown in Figs. 8 and 9. The *Dmax* adaptation proves to be very effective (see Fig. 9) and the performance is not very sensitive (at least in this case) to the initial value of *Dmax*, except when it is too low (as in Fig. 8(a); then the performance is not so good and a tendency to oscillate may appear).

These results, compared with those of [20], show a more realistic control signal. Here, besides, there is no need for a priori learning of the process fuzzy model and the SOC is much simpler to implement.

6.2. Case 3: Application to a real process

The process is a small scale laboratory in which one intends to produce a certain quantity of heated air (Fig. 10). The air enters the process due to a blower. Inside the tube there is an energized electrical resistance that heats the air (the actuator). The goal is to control the air temperature along the tube, which can be read by a sensor placed in one of the three possible positions. The angle of the air overture can also be changed (introducing load disturbances).

The sampling time used in all these experiments was 150 ms.

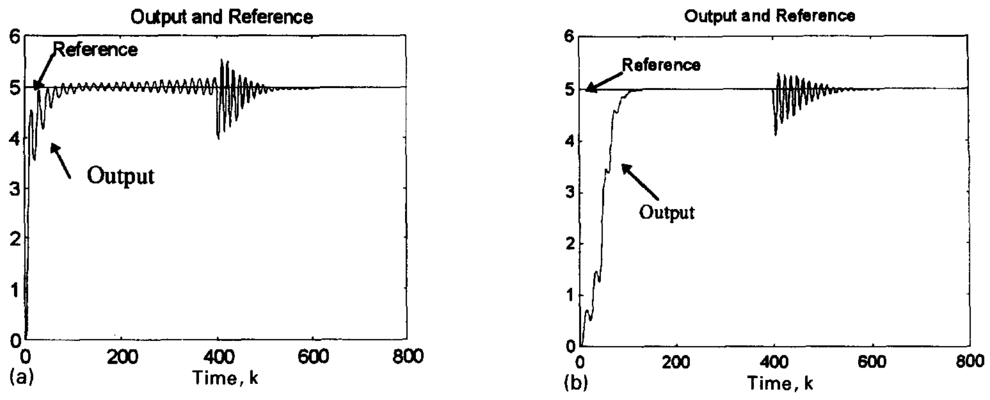


Fig. 8. Case 2. Influence of *Dmax* initialization, with window = 20. (a) Initial *Dmax* = 30; (b) initial *Dmax* = 150.

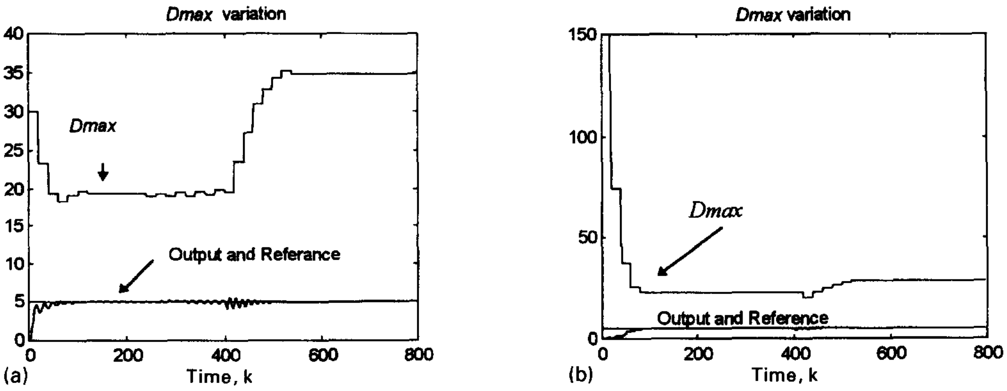


Fig. 9. Case 2. Influence of *Dmax* initialization with window = 20. (a) Initial *Dmax* = 30; (b) initial *Dmax* = 150.

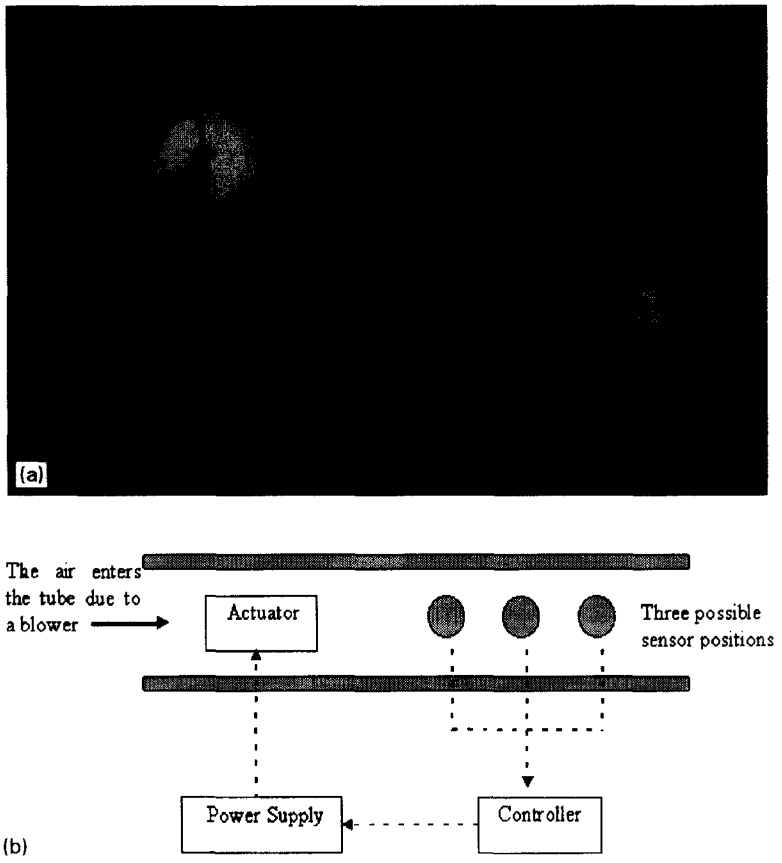

(a)
The air enters the tube due to a blower
Actuator
Three possible sensor positions
Power Supply
Controller
(b)

Fig. 10. Case 3(a) Picture of the experimental process PT326. (b) Case 3. Schematic representation of the real process PT326.

The a priori knowledge about the process is: (i) it can be approached by a first order one, (ii) its time delay is not superior to two sampling intervals, (iii) its monotonicity is relation (1), from assumption (v) in Section 2.1.

Figs. 11–13 illustrate the results obtained with a sinusoidal reference of period 30 s, and FMNR = 100, $n = 2$. Despite the noisy environment, the algorithm is capable of controlling the process with a smooth input signal. In Figs. 14 and 15 are shown the results when considering a triangular wave as reference, with equal FMNR. The proposed SOC is able to manage appropriately the abrupt change in the ramps declive.

Figs. 16 and 17 compare the performance of the proposed SOC with the case where there is no adaptation of *Dmax* and the formulae (15) is not used (but only the formulae (5)). For a load disturbance, the proposed SOC reacts immediately because of formulae (15), a new learning phase is started, (Fig. 16(a)), and soon the output recovers from the disturbance. In the other situation (Fig. 16(b)), the controller takes some time to react because it is necessary that a significative number of rules will be overwritten (by the new data) in order to change the control signal; the learning phase is much longer, the output takes a long time to reach again its good value.

If a process dynamic change occurs, in the form off a change in the process gain (it is halfed by an electronic amplifier in cascade with the process), the proposed SOC, as seen in Fig. 17(a), with a (re)learning phase of 120 sampling intervals (approximately), compensates the control signal for that change. However, if *Dmax* is

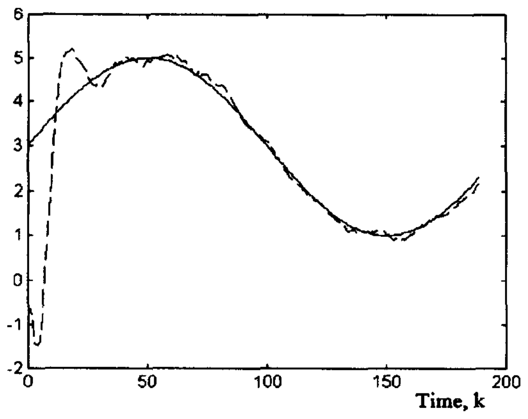


Fig. 11. Case 3 with a sinusoidal reference. The learning phase. (- - - -) Process output; (——) reference.

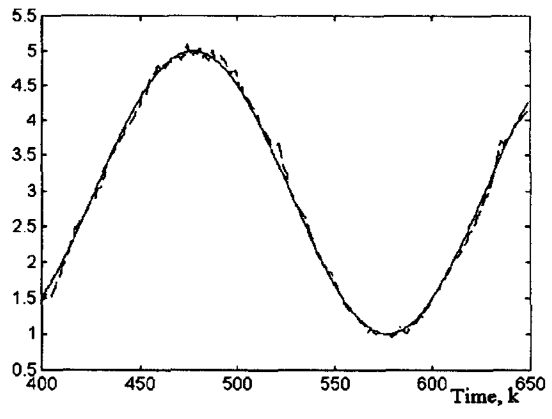


Fig. 12. Case 3 with a sinusoidal reference. After the learning phase, the output of the process is following the reference. (- - - -) Process output; (——) reference.

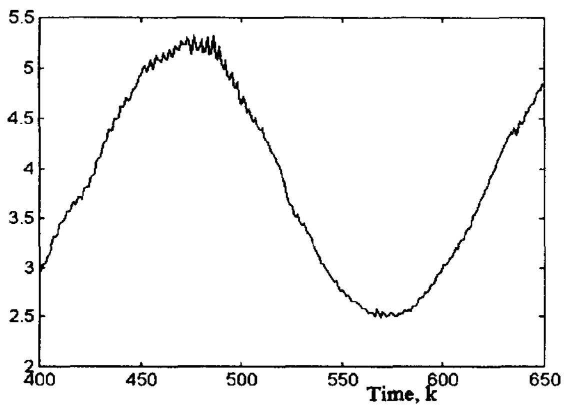


Fig. 13. Case 3 with a sinusoidal reference. The input to the process, referring to Fig. 12.

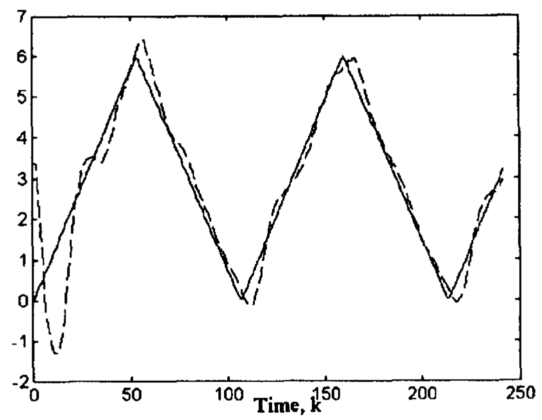


Fig. 14. Case 3 with a triangular reference. The learning phase. (- - - -) Process output; (——) reference.

not adapted neither (15) is applied, then the performance is substantially deteriorated, as shown in Fig. 17(b), by the same reasons as in the previous case.

The vertical scales in the figures are in volt, from the sensor/transducer. Its relation with the temperature in °Celsius is 1 V – 22°C, 3 V – 32°C, 5 V – 38°C. By the construction of the apparatus there is a non-linearity in this relation.

The disturbances and changes are manually made and are as similar as possible for the sake of comparison.

6.3. Case 4: Simulations with an MISO system

Considering the system (27) with two inputs $u1$ and $u2$ and one output y:

$$y(k) = 0.1y^2(k-1) + \frac{u1(k-1)u2(k-2)[u1(k-1) + 3/2]}{1 + u1^2(k-1) + u2^2(k-2)} \tag{27}$$

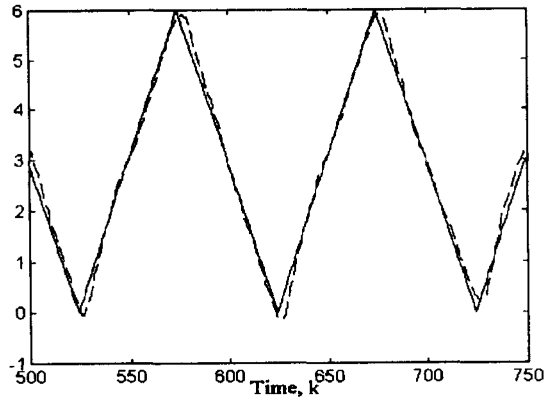


Fig. 15. Case 3 with a triangular reference, in long run. (- - -) Process output; (——) reference.

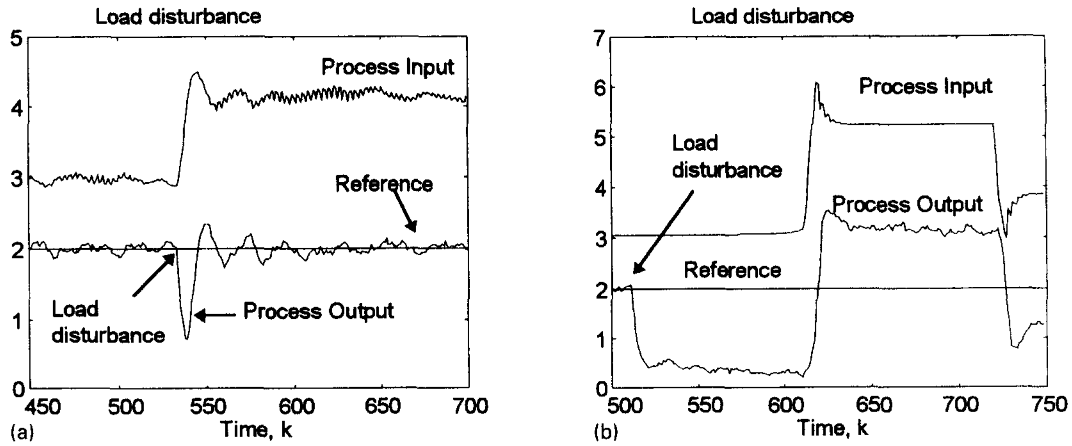


Fig. 16. Case 3. Load disturbance. Sensor in position 3. (a) Proposed SOC; (b) Without *Dmax* adaptation and without (15).

to be controlled as shown in Fig. 18. The controller must be an SIMO system. For a sinusoidal reference, the results obtained are illustrated in Fig. 19, showing a good behavior. After 300 iterations the error is almost zero.

6.4. Case 5: Simulation with an MIMO process

The problem adopted here is described in [12]. The main purpose is the simultaneous regulation of the blood pressure (mean arterial pressure – MAP) and of the cardiac output (CO). It is desirable to maintain or increase CO and, at the same time, to decrease MAP. There are two drugs that are often used in order to achieve this goal: an isotropic drug dopamine (DOP) that increases CO and MAP, and a vasoactive drug sodium nitroprusside (SNP) that increases CO but decreases MAP. Considering the following transfer matrix (28) model of the process

$$\begin{bmatrix} \Delta\,\mathrm{CO} \\ \Delta\,\mathrm{MAP} \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} u1 \\ u2 \end{bmatrix} \tag{28}$$

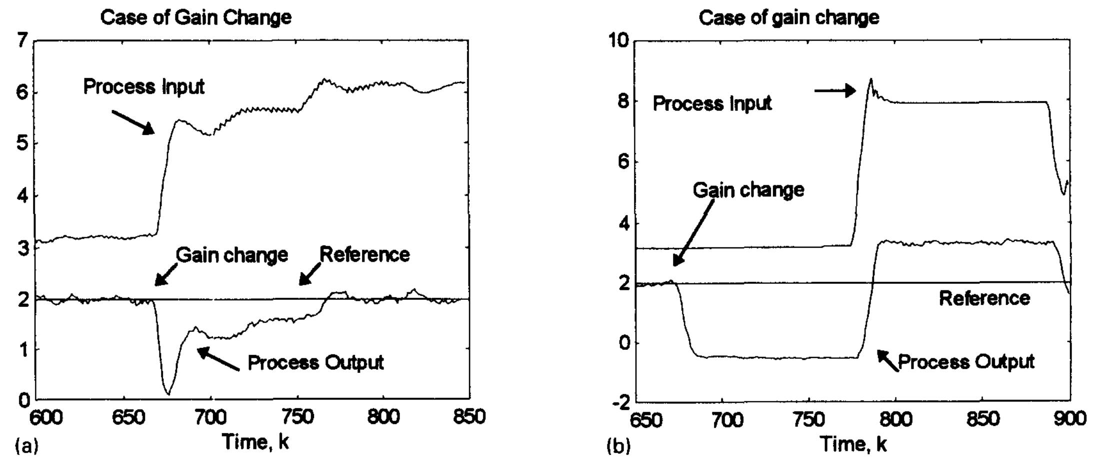


Fig. 17. Case 3. Dynamic change: process gain is multiplied by $\frac{1}{2}$. (a) Proposed SOC; (b) without *Dmax* adaption and without (15).

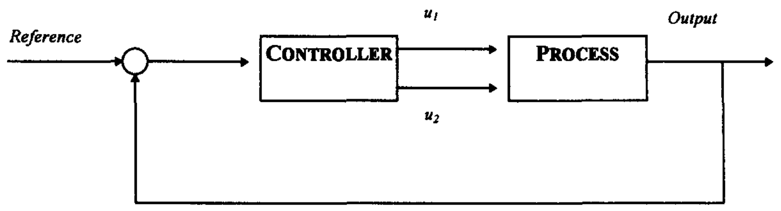


Fig. 18. Case 4. Block diagram of the control loop. MISO Process implies SIMO controller.

with

$$H_{11} = \frac{K_{11}\,e^{-\tau_1 s}}{T_1 s + 1} + \frac{-24.76\,K_{21}\,e^{-\tau_2 s}}{T_2 s + 1}, \tag{29}$$

$$H_{12} = \frac{K_{12}\,e^{-\tau_2 s}}{T_1 s + 1} + \frac{-24.76\,K_{21}\,e^{-\tau_2 s}}{T_2 s + 1}, \tag{30}$$

$$H_{21} = \frac{0.6636\,K_{11}\,e^{-\tau_1 s}}{T_1 s + 1} + \frac{76.38\,K_{21}e^{-\tau_2 s}}{T_2 s + 1}, \tag{31}$$

$$H_{22} = \frac{0.6636\,K_{12}\,e^{-\tau_2 s}}{T_1 s + 1} + \frac{76.38\,K_{22}e^{-\tau_2 s}}{T_2 s + 1}, \tag{32}$$

$K_{11} = 8.44$; $K_{12} = 5.275$; $K_{21} = -0.09$; $K_{22} = -0.15$,

$\tau_1 = 60$ sec.; $\tau_2 = 30$ sec. (time constants),

$T_1 = 84.1$ sec.; $T_2 = 58.75$ sec. and s is the Laplace operator.

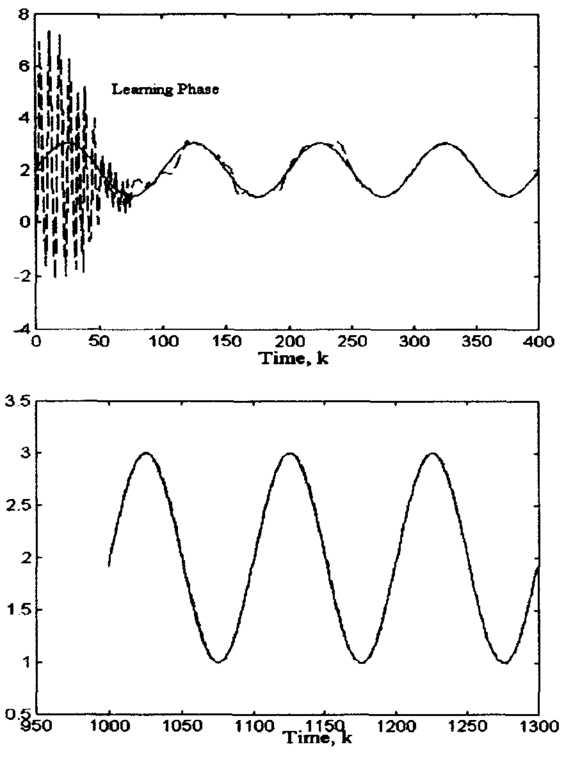


Fig. 19. Case 4. A non-linear process with one output and two inputs: (- - - -) process outputs, (——) reference.

Considering a sampling interval of 30 s, the discretised difference equations are (33)

$$\begin{bmatrix} y1(k) \\ y2(k) \end{bmatrix} = \begin{bmatrix} 1.3 & 0 \\ 0 & 1.3 \end{bmatrix} \begin{bmatrix} y1(k-1) \\ y2(k-1) \end{bmatrix} + \begin{bmatrix} -0.42 & 0 \\ 0 & -0.42 \end{bmatrix} \begin{bmatrix} y1(k-2) \\ y^2(k-2) \end{bmatrix}$$

$$+ \begin{bmatrix} 0.67 & 3.07 \\ -2.75 & -3.53 \end{bmatrix} \begin{bmatrix} u1(k-1) \\ u2(k-1) \end{bmatrix} + \begin{bmatrix} 2.13 & -0.09 \\ 3.60 & 2.58 \end{bmatrix} \begin{bmatrix} u1(k-2) \\ u2(k-2) \end{bmatrix} + \begin{bmatrix} -1.52 & 0 \\ 1.01 & 0 \end{bmatrix} \begin{bmatrix} u1(k-3) \\ u2(k-3) \end{bmatrix} \tag{33}$$

with $y1$ representing the variation of CO, $y2$ the variation of MAP, $u1$ the input DOP and $u2$ the input SNP.

The correspondence between output and input variables that was found to give better results was $u1$ to $y1$ and $u2$ to $y2$. The set of relations used in the defuzzification method was the following:

if we want to increase $y1$ and increase $y2$ then increase $u1$ and maintain $u2$,
if we want to increase $y1$ and decrease $y2$ then increase $u1$ and increase $u2$,
if we want to increase $y1$ and maintain $y2$ then increase $u1$ and increase $u2$,
if we want to decrease $y1$ and decrease $y2$ then maintain $u1$ and decrease $u2$,
if we want to decrease $y1$ and increase $y2$ then maintain $u1$ and decrease $u2$,
if we want to decrease $y1$ and maintain $y2$ then decrease $u1$ and decrease $u2$,
if we want to maintain $y1$ and increase $y2$ then increase $u1$ and decrease $u2$,
if we want to maintain $y1$ and decrease $y2$ then decrease $u1$ and increase $u2$,
if we want to maintain $y1$ and maintain $y2$ then maintain $u1$ and maintain $u2$.

The values of the variables are increased or decreased in the same way as described in Section 4.4. The only difference in this defuzzification method is that instead of two *If ... then* rules we have a set of rules that transmit to the algorithm our knowledge about the relations between the outputs and the inputs.

Using a rule base with only ten rules, the results obtained are illustrated in the following.

The proposed SOC shows good regulating capabilities (Fig. 20), without the need for a reference model, comparable to those of [12], but here with a smoother control signal at least for the case shown by Fig. 21. The decoupling capabilities are shown in Fig. 22 where the two references are independently followed up, although (as it happens for all realizable controllers) it is not possible to completely eliminate the coupling in transient state. Such properties are not shown in [12].

If the process parameter K_{22} changes by + 20%, the results of the control system are illustrated by Fig. 23. After a short (re)learning phase, the regulation error tends to zero. This effect is obtained with a relatively smooth control signal (Fig. 24). These results are comparable to those of [12], but here without repetitive trials (for learning) and in a much simpler way.

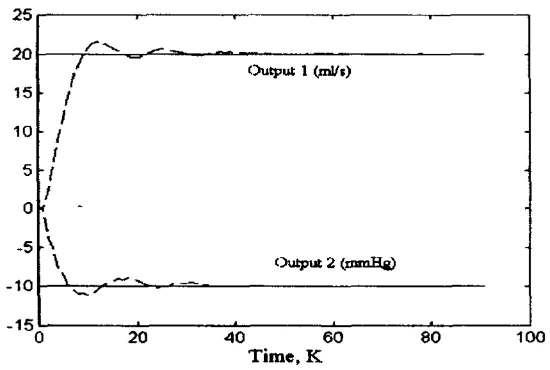


Fig. 20. Case 5. Simultaneous regulation of CO and MAP: (- - - -) process outputs, (——) reference.

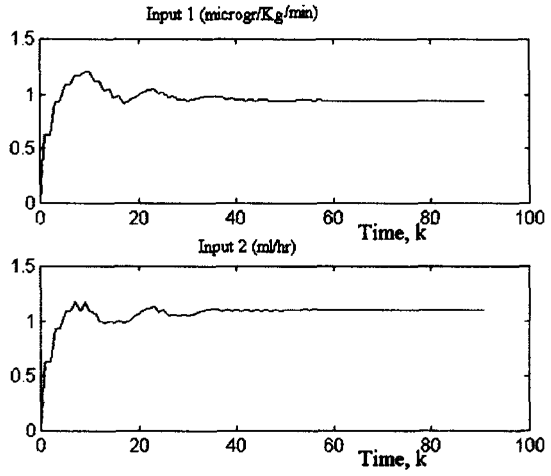


Fig. 21. Process inputs for the problem of simultaneous regulation of CO and MAP.

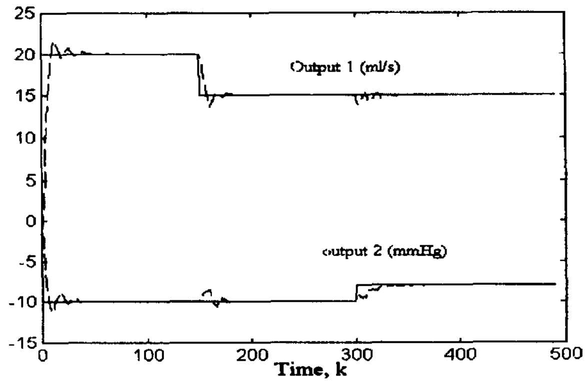


Fig. 22. Case 5. Changes in both the references: (- - - -) process output, (——) reference.

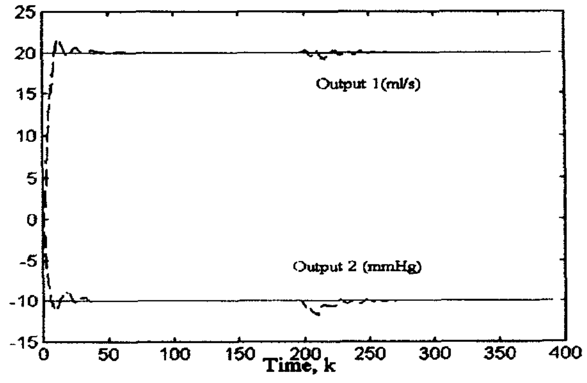


Fig. 23. Case 5. Change in one of the system parameters at iteration 200: $K'_{22} = 1.2K_{22}$ ($\Delta K_{22} = +0.2$).

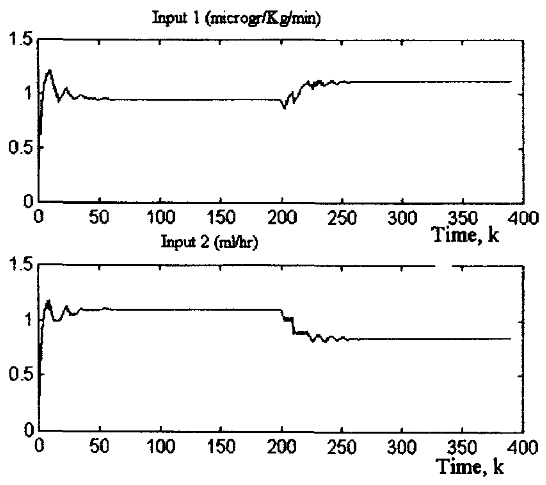


Fig. 24. Case 5. Process inputs relating to the change in the system parameter K_{22}.

7. Conclusions

The most important and original characteristics of the controller are that (i) the FMNR is a parameter given by the programmer to the controller, so one can have a more accurate control over the sampling time and simultaneously maintain the computational resources needed at a reasonable level and (ii) the adaptation technique is effective and easy to implement. The assumptions made in order to allow the application of the controller do not require much knowledge about the process and are easily fulfilled for the majority of systems. The on-line construction of the rule base as well as the adaptive similarity degree makes it easy for the controller to forget one system and learn how to control another when process dynamics change or other disturbances appear. The feedforward and predictive effects given in the fuzzification and defuzzification stages allow to compensate for the pure time delays and give a good dynamics to the control system.

The learning phase can probably be quite longer in real more complex processes (than the ones experimented) if the similarity factor adaptation is not well done. This may be caused by the existence of noisy data in the time window observation. If noise could be appropriately separated from the good signal, then learning would be improved. The simple technique used for the adaptation of *Dmax* (see expressions (13) and (14)) was found by experimental research but is sufficiently effective at least in the examples shown. Probably in more complex situations a more elaborated adaptation strategy can also improve the learning abilities.

Further research about the introduction of a fuzzy higher level in charge of supervising the behavior of the control loops is planned. This level should be able to give a higher autonomy to the SOC by changing the FMNR and the observation window width when needed (and also the other aspects focused in Section 4.7) and by improving the adaptation tasks if the learning phase would be too long. Problems were found trying to control unstable systems and those of non-minimum phase. The unstable systems do not allow the initial learning phase. The non-minimum phase systems do not fulfill completely the assumptions made (v). Further research is also needed for these cases.

Note: The computer configuration used was the following: Pentium 100 MHz μcomputer, 16MB Ram, Matlab Programming Language for simulations, Dos LabWindows for real-time control (with data loaded in Matlab for hardcopy printings).

Acknowledgements

This work has been partially supported by JNICT/PRAXIS XXI Program.

References

[1] B. Carse, T.C. Fogarty, A. Munro, Evolving fuzzy rule based controllers using genetic algorithms, Fuzzy Sets and Systems 80 (1996) 273–293.
[2] Y.Y. Chen, T.C Tsao, A description of the dynamical behavior of fuzzy systems, IEEE Trans. Systems Man Cybernetic 19 (1989) 745–755,
[3] D.H. Dryankov, H. Hellendoorn, M. Reinfrank, An Introduction to Fuzzy Control, Springer, Berlin, 1993.
[4] J.S. Jang, Self-learning fuzzy controllers based on temporal back propagation, IEEE Trans. Neural Networks 3 (1992) 714–723.
[5] M.S. Ju, D.L. Yang, Design of adaptive fuzzy controls based on natural control laws, Fuzzy Sets and Systems 81 (1996) 191–202.
[6] C.A. Karr, E.J. Gentry, Fuzzy control of PH using genetic algorithms, IEEE Trans. Fuzzy Systems 1 (1993) 46–53.
[7] N.K. Kasabov, Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems, Fuzzy Sets and Systems 82 (1996) 135–149.
[8] S.W. Kim, E.T. Kim, M. Park, A new adaptive fuzzy controller using the parallel structure of fuzzy controller and its application, Fuzzy Sets and Systems 81 (1996) 205–226.
[9] F. Klawonn, G. Jörg, K. Rudolf, Fuzzy control on the basis of equality relations with an example from idle speed control, IEEE Trans. Fuzzy Systems 3 (1995) 336–349.
[10] C.H. Lee, S.D. Wang, A self-organizing adaptive fuzzy controller, Fuzzy Sets and Systems 80 (1996) 295–313.
[11] C.-T. Lin, C.-J. Lin, C.S. George Lee, Fuzzy adaptive learning control network with on-line neural learning, Fuzzy Sets and Systems 71 (1995) 25–45.
[12] D.A Linkens, N. Junhong, Constructing rule-bases for multivariable fuzzy control by self-learning, Internat. J. Systems Sci. 24 (1993) 124–157.
[13] Y.-M. Park, U.-C. Moon, K.Y. Lee, A self-organizing fuzzy logic controller for dynamic systems using a fuzzy auto-regressive moving average model, IEEE Trans. Fuzzy Systems 3 (1995) 75–82.
[14] M.J. Patyra, D.M. Mlynek, Fuzzy Logic Implementation and Applications, Wiley and Teubner, Chichester, 1996.
[15] M.L. Presti, R. Poluzzi, A.M. Zanaboni, Synthesis of fuzzy controllers through neural networks, Fuzzy Sets and Systems 71 (1995) 47–70.
[16] T.J. Procyk, E.H. Mandani, A linguistic self-organizing process controller, Automatica 15 (1979) 15–30.
[17] C.-S. Ting, T.-H.- S. Li, F.-C. Kung, An approach to systematic design of the fuzzy control system, Fuzzy Sets and Systems 77 (1996) 151–166.
[18] H. Wang, G.P. Liu, C.J. Harris, M. Brown, Advanced Adaptive Control Pergamon Press, Oxford, 1995.
[19] L.X. Wang, Adaptive Fuzzy Systems and Control, Design and Stability Analysis Prentice-Hall, Englewood Cliffs, NJ, 1994.
[20] G. Vachtsevanos, S. Farinwata, Fuzzy logic control: a systematic design and performance assessment methodology in: M.J. Patyra, D.M. Mlynek (Eds.), Fuzzy Logic Implementation and Applications, Wiley and Teubner, Chichester, 1996.
[21] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, IEEE Trans. Systems, Man Cybernet. 3 (1973) 28–44.