The Self-Organising Fuzzy Controller

Jan Jantzen <jj@iau.dtu.dk>[1]

Abstract

A marginally stable test system, with a large dead time and an integrator, is stabilised by a self-organising fuzzy controller in a simulation study. It acts as a case study, to explain the self-organising controller to engineering students. The paper is one of a series of tutorial papers for a course in fuzzy control.

Contents

[1] Technical University of Denmark, Department of Automation, Bldg 326, DK-2800 Lyngby, DENMARK. Tech. report no 98-H 869 (soc), 19 Aug 1998.

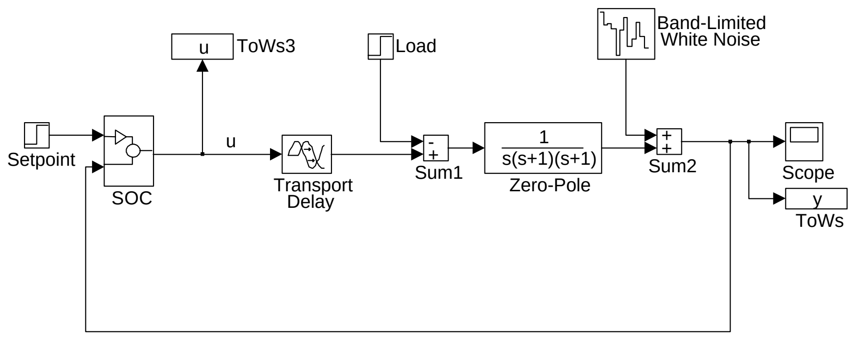

Figure 1: Simulink model for testing the SOC.

1. Introduction

Systems with a dominant time delay are notoriously difficult to control, and the fuzzy *self-organising controller* (Mamdani & Baaklini, 1975; Procyk and Mamdani, 1979; Yamazaki & Mamdani, 1982), or *SOC* for short, was developed specifically to cope with processes with dead time.

In those days, the distinction between the terms *self-organising* and *adaptive* was unclear, but today the SOC will be categorised under adaptive controllers; broadly defined, an adaptive controller is a controller with adjustable parameters and a mechanism for adjusting the parameters (Åström & Wittenmark, 1995). Actually, judging from the block diagram of the SOC, it may even be sub-categorised as a *model-reference adaptive system*; this is an adaptive system in which the performance specifications are given by a model. The model tells how the plant output ideally should respond to the control signal. The SOC uses a desired response, rather than a model of the plant, and it adjusts its parameters according to a heuristic rule. There are laboratory and real world applications of the SOC controller (*e.g.*, Peng & Liu, 1988; Linkens & Abbod, 1991; Sutton & Jess, 1991; Dawson & Gao, 1994; Marques, Baptista & Costa, 1997; Nie & Lee, 1997).

The objective in the following is to explain, in a tutorial manner, the components and properties of a SOC for control engineering students. We set out to control the plant

$$G(s) = e^{-9s} \frac{1}{s\,(s+1)^2} \tag{1}$$

It is difficult, because the dead time of 9 seconds is large compared to the time constants of the plant. The plant is inserted in the test bench in Fig. 1. At time $t = 0$ the reference changes abruptly from 0 to 1 and after 500 seconds, a load of 0.075 units is forced on the system. The *transport delay* block in Fig. 1 is the dead time of the plant (9 seconds), and the *band-limited white noise* block is there in order to experiment with noise. The simulation environment is Matlab (v. 4.2.c.1) for Windows together with Simulink (v. 1.3c). The

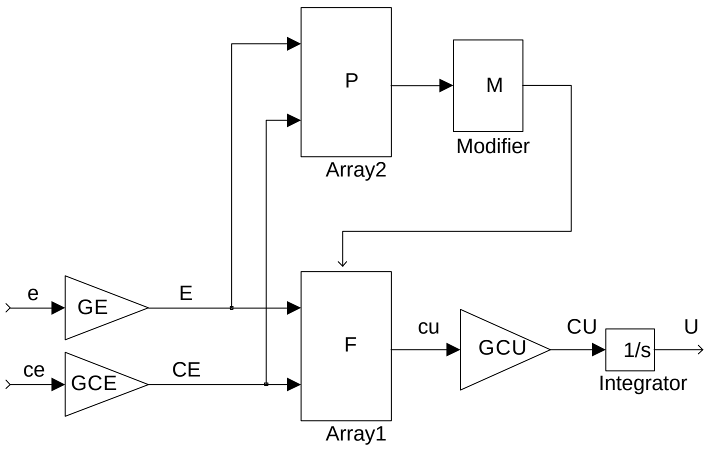

Figure 2: Self-organising controller.

strategy is

1. to design and tune a fuzzy controller with a linear control surface, and
2. to start self-organisation without changing the tuning, and
3. to see if self-organisation improved the response.

The experiment is supposed to show qualitatively, whether there is an improvement; it is not meant to be an exhaustive scientific investigation.

2. Self-Organising Control

To Mamdani the SOC was a further development of the original fuzzy controller (Assilian & Mamdani, 1974a; 1974b). They called it self-organising, since it is able to adjust the control strategy in a fuzzy controller without human intervention.

2.1 SOC Block Diagram

The SOC has a hierarchical structure in which the lower level is a table based controller and the higher level is the adjustment mechanism (Fig. 2).

Lower level At the lower level is an incremental controller, where the control signal is added to the previous control signal, modelled as an integrator in the figure. The two inputs to the controller are the error e and the change in error ce. These are multiplied by two gains, GE and GCE respectively, before the rule base in **F**. The SOC requires a table in block **F**, which could be generated from a linguistic rule base. The table lookup value,

		CE												
		−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
	−6	−6	−6	−6	−6	−6	−6	−6	0	0	0	0	0	0
	−5	−6	−6	−6	−6	−6	−6	−6	−3	−2	−2	0	0	0
	−4	−6	−6	−6	−6	−6	−6	−6	−5	−4	−2	0	0	0
	−3	−6	−5	−5	−4	−4	−4	−4	−3	−2	0	0	0	0
	−2	−6	−5	−4	−3	−2	−2	−2	0	0	0	0	0	0
	−1	−5	−4	−3	−2	−1	−1	−1	0	0	0	0	0	0
E	0	−4	−3	−2	−1	0	0	0	0	0	1	2	3	4
	1	0	0	0	0	0	0	1	1	1	2	3	4	5
	2	0	0	0	0	0	0	2	2	2	3	4	5	6
	3	0	0	0	0	2	3	4	4	4	4	5	5	6
	4	0	0	0	2	4	5	6	6	6	6	6	6	6
	5	0	0	0	2	2	3	6	6	6	6	6	6	6
	6	0	0	0	0	0	0	6	6	6	6	6	6	6

Figure 3: Example of a performance table; note the universes (adapted from Procyk and Mamdani, 1979).

called *change in output* cu, is multiplied by the output gain GCU and integrated to become the eventual control signal U. The integrator block can be left out, then the table value is usually called u (instead of cu), scaled by a gain GU (instead of GCU), and used directly as the control signal U.

Higher level The idea behind the self-organisation is to let an adjustment mechanism update the values in the table **F**, based on the current performance of the controller. Basically, if the performance is poor, the responsible table value should be punished, such that next time that cell of the table is visited, the control signal will be better.

The higher level monitors *error* and *change in error*, and it modifies the table **F** through a *modifier* algorithm **M** when necessary. It uses a *performance measure* to decide the magnitude of each change to **F**. The performance measures are numbers, organised in a table **P** the size of **F**, expressing what is desirable, or undesirable rather, in a transient response. The table **P** can be built using linguistic rules, but is often built by hand. The same performance table **P** may be used with a different process, without prior knowledge of the process, since it only expresses the *desired* transient response. The controller can start from scratch with an **F**-table full of zeros; it will, however, converge faster towards a stable table, if **F** is primed with sensible numbers to begin with.

2.2 Training

The SOC learns to control the system in accordance with the desired response; we will call this *training*. At the sampling instant *n*,

1. it records the deviation between the actual state and the desired state, and
2. it corrects table **F** accordingly.

								CE						
		-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
	-6	-6	-6	-6	-6	-6	-6	-6	-5	-4	-3	-2	-1	0
	-5	-6	-6	-6	-6	-5	-4	-4	-4	-3	-2	-1	0	0
	-4	-6	-6	-6	-5	-4	-3	-3	-3	-2	-1	0	0	1
	-3	-6	-6	-5	-4	-3	-2	-2	-2	-1	0	0	1	2
	-2	-6	-5	-4	-3	-2	-1	-1	-1	0	0	1	2	3
	-1	-5	-4	-3	-2	-1	-1	0	0	0	1	2	3	4
E	0	-5	-4	-3	-2	-1	0	0	0	1	2	3	4	5
	1	-3	-2	-1	0	0	0	0	1	1	2	3	4	5
	2	-2	-1	0	0	0	1	1	1	2	3	4	5	6
	3	-1	0	0	0	1	2	2	2	3	4	5	6	6
	4	0	0	0	1	2	3	3	3	4	5	6	6	6
	5	0	0	1	2	3	4	4	4	5	6	6	6	6
	6	0	1	2	3	4	5	6	6	6	6	6	6	6

Figure 4: Example of another performance table (adapted from Yamazaki, 1982).

The performance table **P** evaluates the current state and returns a performance measure **P**(i_n, j_n), where i_n is the index corresponding to E_n, and j_n is the index corresponding to CE_n.

Figures 3 and 4 are examples of early performance tables. If the performance measure is zero, the state is satisfactory, otherwise unsatisfactory to some degree. In the latter case the modifier **M** assumes that the control signal must be punished. It cannot be the current control signal that is responsible, however, because it takes some time before a control action shows up in the process output.

The simple strategy is to go back a number of samples in time to correct an earlier control signal. The modifier must therefore know the time lag in the plant. It goes back d samples comparable to the time lag; the integer d is called the *delay-in-penalty* (in the literature it is called delay-in-*reward,* but that seems slightly misleading).

The modifier assumes that the plant output depends monotonously on the input. It is thus a requirement, for the SOC to function correctly, that an increase in plant output calls for an adjustment of the control signal *always* in the same direction, whether it be an increase or a decrease.

The precise adjustment rule is

$$u_{n-d} = u_{n-d} + p_n \tag{2}$$

In terms of the tables **F** and **P**, the adjustment rule is

$$\mathbf{F}\,(i,j)_{n-d} = \mathbf{F}\,(i,j)_{n-d} + \mathbf{P}\,(i,j)_n \tag{3}$$

The time subscript n denotes the current sample. In words, it regards the performance measure as an extra contribution to the control signal that should have been, in order to push the plant output to a state with a zero penalty.

Example 1 (update scheme) *Assume* $d = 2$ *and process values according to Table* 1 *after*

	Time t				
Variable	1	2	3	4	5
E	6	3	1	0	-1
CE		-3	-2	-1	-1
u		0	-1	-1	-2
p		0	0	0	-1

Table 1: Data for Example 1.

a step in the setpoint. From $t = 1$ to $t = 4$ the performance measure p is 0, while the process moves up towards the setpoint. At $t = 5$, error *changes sign indicating an overshoot, and the performance table reacts by dictating $p_5 = -1$. Since d is 2, the new entry in **F** will be at the position corresponding to $t = 5 - d = 3$. The state was $(E_3, CE_3) = (1, -2)$, and the element in **F** was $u_3 = -1$. The modified entry will then be $u_3 = u_3 + p_5 = -1 - 1 = -2$, which is inserted into **F**.*

2.3 A Simplification

The original performance tables in Figs. 3 and 4 were built by hand, based on trial and error. Presumably, if the numbers in the table **P** are small, it will be necessary with many updates to reach a useful **F**; if the numbers in **P** are large, the convergence will be faster, but maybe also unstable. The following analysis leads to a deeper understanding of the mechanism.

Observing the two original performance tables, it seems that the Procyk & Mamdani table (Fig. 3) holds the zeros in a more or less diagonal band, while the Yamazaki table (Fig. 4) keeps them in a z-shaped patch. Because the zeros indicate no penalty, those states must be admissible. Assuming that the system stays within the zero patch, what does this mean?

Focusing on the zero diagonal, it expresses the relation

$$GE * e + GCE * \frac{de}{dt} = 0 \tag{4}$$

This is an ordinary differential equation, and we can solve it,

$$e(t) = e(0)\exp(-\frac{t}{GCE/GE}) \tag{5}$$

In other words, a first order exponential decay with a time constant GCE/GE; assuming $e(0) = 1$ the error e will gradually die out, and after $t = GCE/GE$ seconds it has dropped 0.63 units. To interpret, the modifier **M** tries to push the system towards a first order transient response.

The z-shaped table is more generous, because it allows a zero slope to begin with, take for instance $\mathbf{P}(100, 0) = 0$, and perhaps a little overshoot in the end of the transient, near the centre of the table. This behaviour is similar to a second order transient response.

Apparently, a simple way to build a linear performance table is to use (4), but with a P on the right hand side of the equation, instead of the zero. The time constant of the desired response would then be GCE/GE, which seems like an unnecessary constraint, because we would like to tune GCE and GE freely, without affecting the desired response.

Instead we introduce the *target time constant* τ. A simple penalty equation, which may replace the table **P**, is,

$$\Delta P = G_p \left(e_n + \tau * ce_n \right) * T_s \tag{6}$$

The *learning gain* G_p affects the convergence rate, and Ts is the sample period. In fact (6) is an incremental controller itself $-$ it is incremental because the output is a change to an existing value, emphasised by the Δ in front of the P $-$ therefore the multiplication by the sample period T_s. The longer the sample period, the fewer the updates, and the larger the penalty in order to keep the update rate independent of the choice of sample period.

2.4 Tuning Of The SOC Parameters

Even though SOC is self-organising, there are gains and parameters to be set.

- $GE, GCE,$ and $G(C)U$. The controller gains must be set near some sensible settings; the exact setting is less important. Imagine for example that the output gain is lowered between two training sessions. The controller will then adapt to the new gain by producing a different **F** table with higher numbers. Even if the input gains change it will still manage to learn. It is therefore a good idea to start with a linear **F**-table, and set the gains according to any PID tuning rule. That is a good starting point for the self-organisation.
- *Target time constant* τ. The smaller τ is, the faster the desired response. If it is too small, however, the closed loop system cannot possibly follow the desired trajectory, but the modifier will try anyway. As a result the **F**-table winds up, and the consequence is a large overshoot. The lower bound for τ is when this overshoot starts to occur. A process with a time constant τ_p and a dead time T_p requires
$$T_p \leq \tau \leq T_p + \tau_p \tag{7}$$
A value somewhat smaller than the right hand side of the inequality is often achievable, because the closed loop system is usually faster than the open loop system.
- *Delay-in-penalty* d. The d should be chosen with due respect to the sample period. The delay should in principle be the target time constant divided by the sample period and rounded to the nearest integer,
$$d = round(\tau / T_s) \tag{8}$$
The results are usually better, however, with a value somewhat smaller than this.
- *Learning gain* G_p. The larger G_p is, the faster the **F**-table builds up, but if it is too large the training becomes unstable. It is reasonable to choose it such that the magnitude of a large ΔP is less than, say, $1/5$ of the maximum value in the output universe. This rule results in the upper bound:
$$G_p \leq \frac{0.2 * \left| \mathbf{F}\left(i, j\right) \right|_{\max}}{\left| \left(e_n + \tau * ce_n \right) \right|_{\max} * T_s} \tag{9}$$

The tuning task is now shifted from having to tune accurately GE, GCE, and $G(C)U$, over to tuning τ, d, and G_p.

2.5 Time Lock

The delay-in-penalty d causes a problem, however, with abrupt changes.

Consider this case. If the error and the change in error, for a time period longer than d samples, have been near zero, then the controller is in an allowed state (the steady state). Suddenly there is a disturbance from the outside, the performance measure ΔP becomes nonzero, and the modifier will modify the **F**-table d samples back in time. It should not do that, however, because the state was acceptable there. The next time the controller visits that state, the control signal will fluctuate. The problem is more general, because it occurs also after step changes in either the reference or the load on the plant.

A solution is to activate a *time-lock* (Jespersen, 1981). The time lock stops the self-organisation for the next d samples; if it is activated at the sampling instant T_n, then self-organisation stops until sampling instant $T_n + d + 1$. In order to trigger the time-lock it is necessary to detect disturbances, abrupt changes in the load, and abrupt changes in the reference. If these events cannot be measured directly and fed forward into the SOC, it is necessary to try and detect it in the process output. If it changes more than a predefined threshold, or if the combination of *error* and *change in error* indicates an abrupt change, then activate the time-lock.

If the delay-in-penalty is implemented as a delay line (a queue) $d + 1$ samples long, it is easy to implement the time-lock. The delay line consists of a queue of index pairs into the matrix **F**,

$$q = \begin{array}{|c|c|c|c|} \hline (i,j)_n & (i,j)_{n-1} & \cdots & (i,j)_{n-d} \\ \hline \end{array} \tag{10}$$

The element to update in **F** is indicated by the last element in q. At the next sampling instant, the current index pair is attached to the front of the queue, while the last index pair is dropped. If an event triggers the time-lock, the queue is flushed, i.e., all cells are emptied. New updates to **F** will only be possible when the queue is full again, that is, when the last element in the queue is non-empty, $d + 1$ samples after the flush.

An extra precaution is to seal the origin in **P**, corresponding to the situation $(e, ce) = (0, 0)$, from updates; this is the steady state in which the control action must always be zero.

If necessary the time-lock can be activated each time the modifier makes a change in **F**. In other words, the modifier waits to see the influence of the change before doing a new modification.

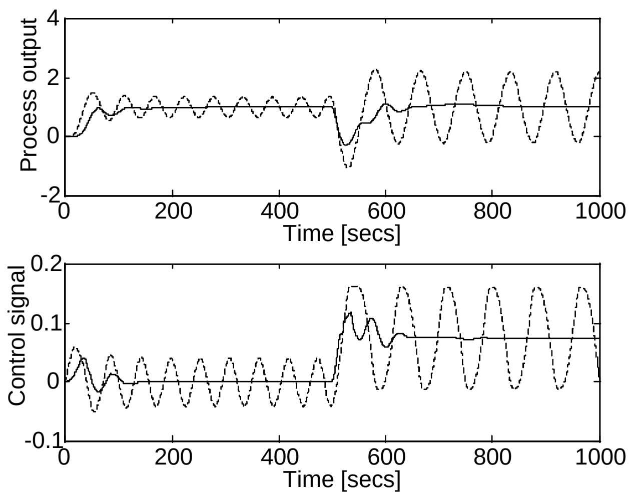

Figure 5: SOC on test plant. Dotted line is before self-organisation, dashed line is after five training runs.

3. Simulation Results And Discussion

Since test setup includes a load on the system, it is necessary to maintain a nonzero control signal in the steady state; thus an incremental controller, with an integrator, is necessary. The rule base was initially a linear 21×21 lookup table with interpolation. The system was hand-tuned to respond reasonably; the resulting gains were

$$\begin{aligned} GE &= 100 \\ GCE &= 3000 \\ GCU &= 4 * 10^{-5} \end{aligned}$$

The dead time of the plant was actually increased *afterwards* to 9 seconds to reach marginal stability (a standing oscillation). The simulation time step was fixed at $T_s = 5$ seconds.

The SOC is able to dampen the oscillation after five training runs (Fig. 5). After training, the control surface has several jagged peaks, see Fig. 6, where the last trajectory is shown also. During each run the controller visited a number of cells in the look-up table, sometimes the same cell is visited several times, and the accumulated changes resulted in the sharp peaks. One might expect jumps in the control signal as a result, but in fact it is rather smooth thanks to the integrator in the output end of the incremental controller.

The design choices for the SOC tuning parameters were as follows. The initial run without self-organisation showed that the period between consecutive peaks in the standing oscillation was approximately 50 seconds. To be conservative, the target time constant was

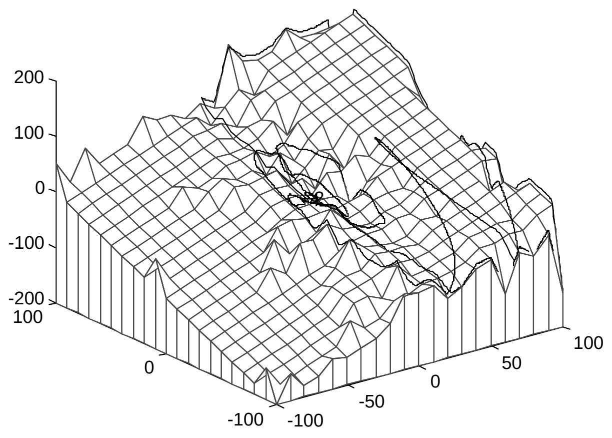

Figure 6: Deformation of the initial control plane after five learning runs.

set at

$$\tau = 60,$$

but 40 seconds would probably be possible too. The table **F** was updated at every simulation step, and with a delay-in-penalty $d = 4$ the length in time of the delay-in-penalty is

$$T_{dip} = T_s * d = 5 * 4 = 20 \sec$$

Thus T_{dip} is considerably less, one third in fact, of the target time constant. With a learning gain

$$G_p = 2$$

the updates ΔP to the **F**-table are never larger than 20, which means that it would take ten updates to reach the limit of the table entry universe $[-200, 200]$, when starting from zero.

To illustrate the training, the first, second, and fifth runs are plotted in Fig. 7. By visual inspection the second run is already much better than the first, and the fifth is rather good, although it has difficulties with the steady state after the load change.

Obviously, good performance means the response is close to the desired response, and there will be only few corrections to **F**. One measure of how the training is progressing is therefore the number of changes per training run. During a successful suite of runs this number will decrease to some steady level $-$ rarely zero $-$ and then the training can stop. If the number of changes starts to increase again, the controller is over-learning. The average number of changes per training run could also be used.

It is perhaps even easier to implement the *integrated absolute penalty*,

$$IAP = \sum_i |p_i| \tag{11}$$

It is related to the magnitude of the error between the actual response and the desired re-

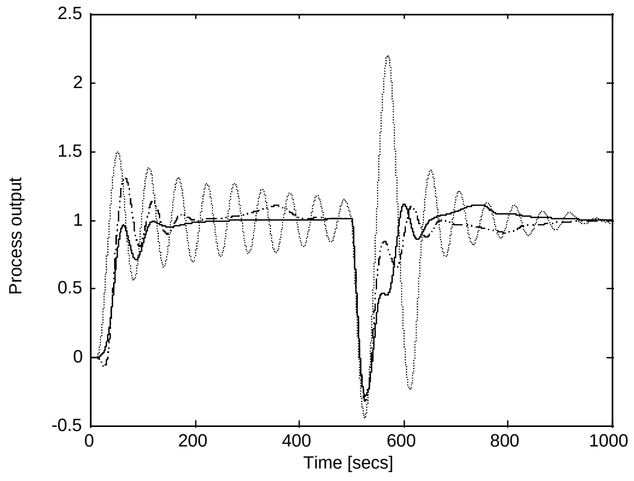

Figure 7: Three training runs: first (dotted), second (dash-dot), and fifth (solid).

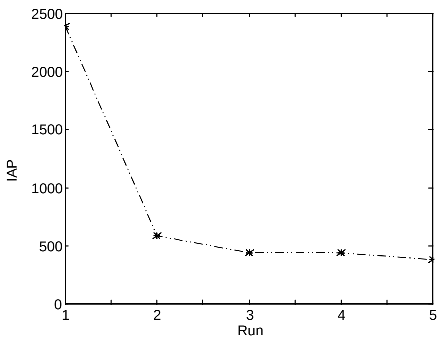

Figure 8: Performance measure for each of five training runs.

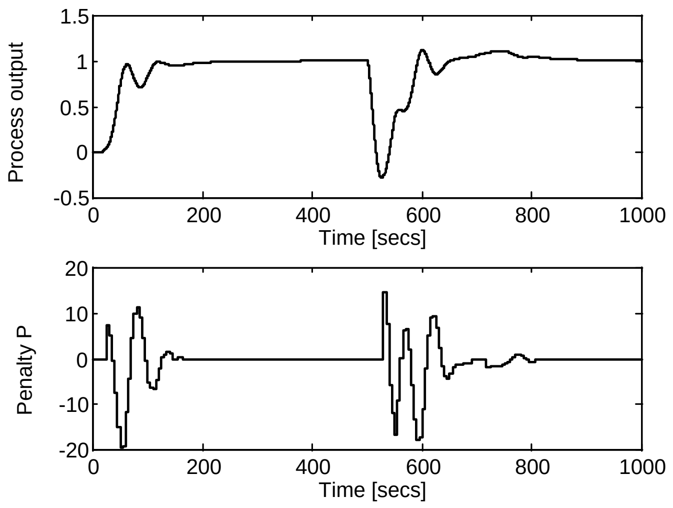

Figure 9: Process output during fifth training run (top) and corresponding penalties (bottom).

sponse, and it is of course similar to the *integrated absolute error*, IAE. The IAP accumulates the magnitude of both negative and positive penalties, and the smaller IAP the better.

The IAP was collected after each run and plotted in Fig. 8, which hints that the training converges towards a constant level, which is often the case. If the training becomes unstable after a while, for example if delay-in-penalty is too long or too short, the curve will start to increase again. This is an indication that the training should stop.

It is also interesting to watch how the penalties progress. Figure 9 shows the process output from the fifth training run together with the penalty. The modifier tries hard to improve the responses to the reference change and the load change, whereas it is more content around the steady state.

Experiments on laboratory models have shown that other processes can be stabilised also, but the rule modifier is sensitive to noise on the process output $-$ it cannot tell if a bad performance measure is due to noise or a bad state. A poor signal to noise ratio will actually spoil the self-organisation.

4. Conclusion

The simulation shows that the SOC is able to stabilise the marginally stable test system. The general experience with the SOC is that it performs surprisingly well.

The adjustment mechanism is simple, but in practice the design is complicated by the time lock and noise precautions. Although the tuning of the traditional input and output gains (GE, GCE, $G(C)U$) is made less cumbersome by the adjustment mechanism, it introduces other parameters to tune (delay-in-penalty, learning gain, target time constant). The tuning task should nevertheless be easier.

The SOC was developed for single-loop control, but it has been applied to multi-loop problems also. Compared to neural network control, it seems to learn faster; a drawback is that it is not a natural multivariable controller.

References

Assilian, S. and Mamdani, E. (1974a). Learning control algorithms in real dynamic systems, *Proc. Fourth Int. Conf. On Digital Computer Applications to Process Control, Zürich*, IFAC/IFIP, Springer, Berlin, pp. 13–20.

Assilian, S. and Mamdani, E. H. (1974b). An experiment in linguistic synthesis with a fuzzy logic controller., *Int. J. Man Machine Studies* **7**(1): 1–13.

Åström, K. J. and Wittenmark, B. (1995). *Adaptive Control*, Addison Wesley Series in Electrical Engineering: Control Engineering, second edn, Addison Wesley.

Dawson, J. G. and Gao, Z. (1994). Fuzzy logic control of linear systems with variable time delay, *in* IEEE (ed.), *Proc. 1994 IEEE International Symposium on Intelligent Control, Columbia, Ohio*, IEEE Control Systems Society, pp. 5–10.

IEE (1991). *Proc. Int. Conf. On Control*, number 332, London.

Jespersen, T. (1981). Self-organizing fuzzy logic control of a pH-neutralisation process, *Technical Report 8102*, Electric Power Eng. Dept., Technical University of Denmark.

Linkens, D. and Abbod, M. (1991). Self-organizing fuzzy logic control for real-time processes, *in* IEE (1991), pp. 971–976.

Mamdani, E. and Baaklini, N. (1975). Prescriptive method for deriving control policy in a fuzzy-logic controller, *Electronics Letters* **11**(25/26): 625–626.

Marques, S. J. C., Baptista, L. F. and Costa, J. S. D. (1997). Force/position control of robot manipulators: A fuzzy adaptive control approach, *Proceedings Fourth European Control Conference, ECC97*, EUCA/IFAC/IEEE, BELWARE Information Technology, Brussels, Belgium, p. paper 227.

Nie, J. and Lee, T. H. (1997). Self-organizing rule-based control of multivariable nonlinear servomechanisms, *Fuzzy Sets and Systems* **91**: 285–304.

Peng, X.-T. and Liu, S.-M. (1988). Self-regulating PID controllers and its application to temperature controlling process, *in* M. Gupta and T. Yamakawa (eds), *Fuzzy Computing: Theory, Hardware, and Applications*, North-Holland, Amsterdam, etc., pp. 355–364.

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

Sutton, R. and Jess, I. (1991). Real-time application of a self-organising autopilot to warship yaw control, *in* IEE (1991), pp. 827–832.

Yamazaki, T. (1982). *An improved algorithm for a self-organising controller and its experimental analysis*, PhD thesis, Queen Mary College, London, Dept. of Electrical and Electronic Engineering.

Yamazaki, T. and Mamdani, E. H. (1982). On the performance of a rule-based self-organizing controller, *Proc. IEEE Conf on Applications of Adaptive and Multivariable Control*, Hull.