

# Design Of Fuzzy Controllers

Jan Jantzen <jj@iau.dtu.dk><sup>1</sup>

## Abstract

Design of a fuzzy controller requires more design decisions than usual, for example regarding rule base, inference engine, defuzzification, and data pre- and post processing. This tutorial paper identifies and describes the design choices related to single-loop fuzzy control, based on an international standard which is underway. The paper contains also a design approach, which uses a PID controller as a starting point. A design engineer can view the paper as an introduction to fuzzy controller design.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Structure of a fuzzy controller</b>	<b>5</b>
2.1	Preprocessing	6
2.2	Fuzzification	6
2.3	Rule Base	7
2.4	Inference Engine	12
2.5	Defuzzification	14
2.6	Postprocessing	15
<b>3</b>	<b>Table Based Controller</b>	<b>18</b>
<b>4</b>	<b>Input-Output Mapping</b>	<b>19</b>
<b>5</b>	<b>Takagi-Sugeno Type Controller</b>	<b>23</b>
<b>6</b>	<b>Summary</b>	<b>24</b>
<b>A</b>	<b>Additive Rule Base (draft)</b>	<b>26</b>

---

<sup>1</sup> Technical University of Denmark, Department of Automation, Bldg 326, DK-2800 Lyngby, DENMARK. Tech. report no 98-E 864 (design), September 30, 1999.

## 1. Introduction

While it is relatively easy to design a PID controller, the inclusion of fuzzy rules creates many extra design problems, and although many introductory textbooks explain fuzzy control, there are few general guidelines for setting the parameters of a simple fuzzy controller. The approach here is based on a three step design procedure, that builds on PID control:

1. Start with a PID controller.
2. Insert an equivalent, linear fuzzy controller.
3. Make it gradually nonlinear.

Guidelines related to the different components of the fuzzy controller will be introduced shortly. In the next three sections three simple realisations of fuzzy controllers are described: a table-based controller, an input-output mapping and a Takagi-Sugeno type controller. A short section summarises the main design choices in a simple fuzzy controller by introducing a check list. The terminology is based on an international standard which is underway (IEC, 1996).

Fuzzy controllers are used to control consumer products, such as washing machines, video cameras, and rice cookers, as well as industrial processes, such as cement kilns, underground trains, and robots. Fuzzy control is a control method based on fuzzy logic. Just as fuzzy logic can be described simply as "computing with words rather than numbers", fuzzy control can be described simply as "control with sentences rather than equations". A fuzzy controller can include empirical rules, and that is especially useful in operator controlled plants.

Take for instance a typical fuzzy controller

1. If error is Neg and change in error is Neg then output is NB
  2. If error is Neg and change in error is Zero then output is NM
  - ...
- (1)

The collection of rules is called a *rule base*. The rules are in the familiar if-then format, and formally the if-side is called the *condition* and the then-side is called the *conclusion* (more often, perhaps, the pair is called *antecedent - consequent* or *premise - conclusion*). The input value "Neg" is a *linguistic term* short for the word *Negative*, the output value "NB" stands for *Negative Big* and "NM" for *Negative Medium*. The computer is able to execute the rules and compute a control signal depending on the measured inputs *error* and *change in error*.

The objective here is to identify and explain the various design choices for engineers.

In a rule based controller the control strategy is stored in a more or less natural language. The control strategy is isolated in a rule base opposed to an equation based description. A rule based controller is easy to understand and easy to maintain for a non-specialist end-user. An equivalent controller could be implemented using conventional techniques – in fact, any rule based controller could be emulated in, say, *Fortran* – it is just that it is convenient

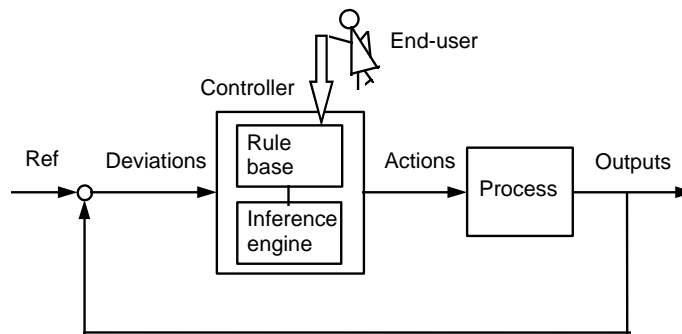


Figure 1: Direct control.

to isolate the control strategy in a rule base for operator controlled systems.

Fuzzy controllers are being used in various control schemes (IEC, 1996). The most obvious one is *direct control*, where the fuzzy controller is in the forward path in a feedback control system (Fig. 1). The process output is compared with a reference, and if there is a deviation, the controller takes action according to the control strategy. In the figure, the arrows may be understood as hyper-arrows containing several signals at a time for multi-loop control. The sub-components in the figure will be explained shortly. The controller is here a fuzzy controller, and it replaces a conventional controller, say, a *PID* (proportional-integral-derivative) controller.

In *feedforward control* (Fig. 2) a measurable disturbance is being compensated. It requires a good model, but if a mathematical model is difficult or expensive to obtain, a fuzzy model may be useful. Figure 2 shows a controller and the fuzzy compensator, the process and the feedback loop are omitted for clarity. The scheme, disregarding the disturbance input, can be viewed as a collaboration of linear and nonlinear control actions; the controller C may be a linear PID controller, while the fuzzy controller F is a supplementary nonlinear controller

Fuzzy rules are also used to correct tuning parameters in *parameter adaptive control* schemes (Fig. 3). If a nonlinear plant changes operating point, it may be possible to change the parameters of the controller according to each operating point. This is called *gain scheduling* since it was originally used to change process gains. A gain scheduling controller contains a linear controller whose parameters are changed as a function of the operating point in a preprogrammed way. It requires thorough knowledge of the plant, but it is often a good way to compensate for nonlinearities and parameter variations. Sensor measurements are used as *scheduling variables* that govern the change of the controller parameters, often by means of a table look-up.

Whether a fuzzy control design will be stable is a somewhat open question. Stability concerns the system's ability to converge or stay close to an equilibrium. A *stable* linear system will converge to the equilibrium asymptotically no matter where the system state variables start from. It is relatively straight forward to check for stability in linear systems,

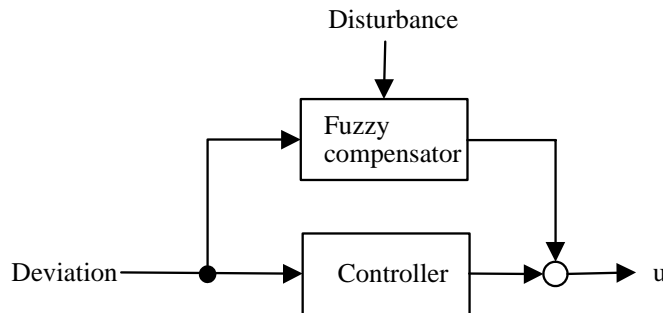


Figure 2: Feedforward control.

for example by checking that all eigenvalues are in the left half of the complex plane. For nonlinear systems, and fuzzy systems are most often nonlinear, the stability concept is more complex. A nonlinear system is said to be *asymptotically stable* if, when it starts close to an equilibrium, it will converge to it. Even if it just stays close to the equilibrium, without converging to it, it is said to be *stable* (in the sense of Lyapunov). To check conditions for stability is much more difficult with nonlinear systems, partly because the system behaviour is also influenced by the signal *amplitudes* apart from the *frequencies*. The literature is somewhat theoretical and interested readers are referred to Driankov, Hellendoorn & Reinfrank (1993) or Passino & Yurkovich (1998). They report on four methods (Lyapunov functions, Popov, circle, and conicity), and they give several references to scientific papers. It is characteristic, however, that the methods give rather conservative results, which translate into unrealistically small magnitudes of the gain factors in order to guarantee stability.

Another possibility is to approximate the fuzzy controller with a linear controller, and then apply the conventional linear analysis and design procedures on the approximation. It seems likely that the stability margins of the nonlinear system would be close in some sense to the stability margins of the linear approximation depending on how close the approximation is. This paper shows how to build such a linear approximation, but the theoretical background is still unexplored.

There are at least four main sources for finding control rules (Takagi & Sugeno in Lee, 1990).

- *Expert experience and control engineering knowledge.* One classical example is the operator's handbook for a cement kiln (Holmblad & Ostergaard, 1982). The most common approach to establishing such a collection of rules of thumb, is to question experts or operators using a carefully organised questionnaire.
- *Based on the operator's control actions.* Fuzzy *if-then* rules can be deduced from observations of an operator's control actions or a log book. The rules express input-output relationships.
- *Based on a fuzzy model of the process.* A linguistic rule base may be viewed as an

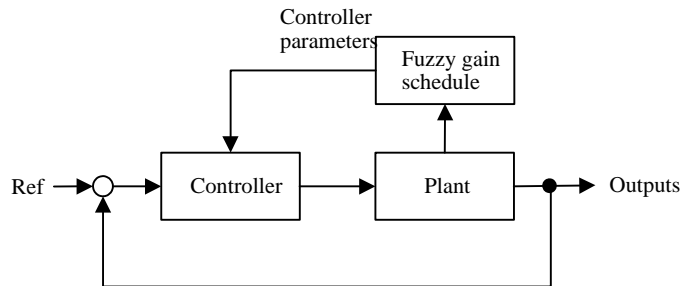


Figure 3: Fuzzy parameter adaptive control.

inverse model of the controlled process. Thus the fuzzy control rules might be obtained by inverting a fuzzy model of the process. This method is restricted to relatively low order systems, but it provides an explicit solution assuming that fuzzy models of the open and closed loop systems are available (Braae & Rutherford in Lee, 1990). Another approach is *fuzzy identification* (Tong; Takagi & Sugeno; Sugeno – all in Lee, 1990; Pedrycz, 1993) or fuzzy model-based control (see later).

- *Based on learning.* The self-organising controller is an example of a controller that finds the rules itself. Neural networks is another possibility.

There is no design procedure in fuzzy control such as root-locus design, frequency response design, pole placement design, or stability margins, because the rules are often non-linear. Therefore we will settle for describing the basic components and functions of fuzzy controllers, in order to recognise and understand the various options in commercial software packages for fuzzy controller design.

There is much literature on fuzzy control and many commercial software tools (MIT, 1995), but there is no agreement on the terminology, which is confusing. There are efforts, however, to standardise the terminology, and the following makes use of a draft of a standard from the International Electrotechnical Committee (IEC, 1996). Throughout, letters denoting matrices are in bold upper case, for example **A**; vectors are in bold lower case, for example **x**; scalars are in italics, for example *n*; and operations are in bold, for example **min**.

## 2. Structure of a fuzzy controller

There are specific components characteristic of a fuzzy controller to support a design procedure. In the block diagram in Fig. 4, the controller is between a preprocessing block and a post-processing block. The following explains the diagram block by block.

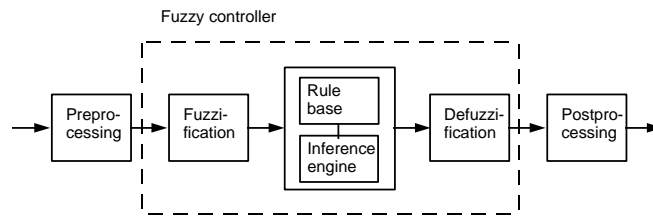


Figure 4: Blocks of a fuzzy controller.

## 2.1 Preprocessing

The inputs are most often hard or *crisp* measurements from some measuring equipment, rather than linguistic. A preprocessor, the first block in Fig. 4, conditions the measurements before they enter the controller. Examples of preprocessing are:

- Quantisation in connection with sampling or rounding to integers;
- normalisation or scaling onto a particular, standard range;
- filtering in order to remove noise;
- averaging to obtain long term or short term tendencies;
- a combination of several measurements to obtain key indicators; and
- differentiation and integration or their discrete equivalences.

A *quantiser* is necessary to convert the incoming values in order to find the best level in a discrete *universe*. Assume, for instance, that the variable *error* has the value 4.5, but the universe is  $\mathbf{u} = (-5, -4, \dots, 0, \dots, 4, 5)$ . The quantiser rounds to 5 to fit it to the nearest level. Quantisation is a means to reduce data, but if the quantisation is too coarse the controller may oscillate around the reference or even become unstable.

Nonlinear *scaling* is an option (Fig. 5). In the *FL Smidth* controller the operator is asked to enter three typical numbers for a small, medium and large measurement respectively (Holmblad & Østergaard, 1982). They become break-points on a curve that scales the incoming measurements (circled in the figure). The overall effect can be interpreted as a distortion of the primary fuzzy sets. It can be confusing with both scaling and gain factors in a controller, and it makes tuning difficult.

When the input to the controller is *error*, the control strategy is a static mapping between input and control signal. A dynamic controller would have additional inputs, for example derivatives, integrals, or previous values of measurements backwards in time. These are created in the preprocessor thus making the controller multi-dimensional, which requires many rules and makes it more difficult to design.

The preprocessor then passes the data on to the controller.

## 2.2 Fuzzification

The first block inside the controller is *fuzzification*, which converts each piece of input

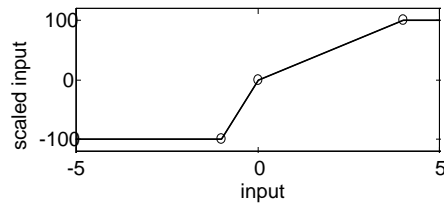


Figure 5: Example of nonlinear scaling of an input measurement.

data to degrees of membership by a lookup in one or several membership functions. The fuzzification block thus matches the input data with the conditions of the rules to determine how well the condition of each rule matches that particular input instance. There is a degree of membership for each linguistic term that applies to that input variable.

### 2.3 Rule Base

The rules may use several variables both in the condition and the conclusion of the rules. The controllers can therefore be applied to both multi-input-multi-output (MIMO) problems and single-input-single-output (SISO) problems. The typical SISO problem is to regulate a control signal based on an error signal. The controller may actually need both the *error*, the *change in error*, and the *accumulated error* as inputs, but we will call it single-loop control, because in principle all three are formed from the error measurement. To simplify, this section assumes that the control objective is to regulate some process output around a prescribed setpoint or reference. The presentation is thus limited to single-loop control.

**Rule formats** Basically a linguistic controller contains rules in the *if-then* format, but they can be presented in different formats. In many systems, the rules are presented to the end-user in a format similar to the one below,

1. If error is Neg and change in error is Neg then output is NB
2. If error is Neg and change in error is Zero then output is NM
3. If error is Neg and change in error is Pos then output is Zero
4. If error is Zero and change in error is Neg then output is NM
5. If error is Zero and change in error is Zero then output is Zero (2)
6. If error is Zero and change in error is Pos then output is PM
7. If error is Pos and change in error is Neg then output is Zero
8. If error is Pos and change in error is Zero then output is PM
9. If error is Pos and change in error is Pos then output is PB

The names *Zero*, *Pos*, *Neg* are labels of fuzzy sets as well as *NB*, *NM*, *PB* and *PM* (negative big, negative medium, positive big, and positive medium respectively). The same

set of rules could be presented in a *relational* format, a more compact representation.

Error	Change in error	Output
Neg	Pos	Zero
Neg	Zero	NM
Neg	Neg	NB
Zero	Pos	PM
Zero	Zero	Zero
Zero	Neg	NM
Pos	Pos	PB
Pos	Zero	PM
Pos	Neg	Zero

(3)

The top row is the heading, with the names of the variables. It is understood that the two leftmost columns are inputs, the rightmost is the output, and each row represents a rule. This format is perhaps better suited for an experienced user who wants to get an overview of the rule base quickly. The relational format is certainly suited for storing in a relational database. It should be emphasised, though, that the relational format implicitly assumes that the connective between the inputs is always logical **and** – or logical **or** for that matter as long as it is the same operation for all rules – and not a mixture of connectives. Incidentally, a fuzzy rule with an **or** combination of terms can be converted into an equivalent **and** combination of terms using laws of logic (DeMorgan’s laws among others). A third format is the tabular linguistic format.

		Change in error		
		Neg	Zero	Pos
Error	Neg	NB	NM	Zero
	Zero	NM	Zero	PM
	Pos	Zero	PM	PB

(4)

This is even more compact. The input variables are laid out along the axes, and the output variable is inside the table. In case the table has an empty cell, it is an indication of a missing rule, and this format is useful for checking completeness. When the input variables are *error* and *change in error*, as they are here, that format is also called a *linguistic phase plane*. In case there are  $n > 2$  input variables involved, the table grows to an  $n$ -dimensional array; rather user-*un*friendly.

To accommodate several outputs, a nested arrangement is conceivable. A rule with several outputs could also be broken down into several rules with one output.

Lastly, a graphical format which shows the fuzzy membership curves is also possible (Fig. 7). This graphical user-interface can display the inference process better than the other formats, but takes more space on a monitor.

**Connectives** In mathematics, sentences are connected with the words *and*, *or*, *if-then* (or *implies*), and *if and only if*, or modifications with the word *not*. These five are called *connectives*. It also makes a difference how the connectives are implemented. The most prominent is probably multiplication for fuzzy **and** instead of minimum. So far most of the examples have only contained **and** operations, but a rule like “If error is very neg and not



zero or change in error is zero then ...” is also possible.

The connectives **and** and **or** are always defined in pairs, for example,

$$\begin{aligned}
 \mathbf{a \text{ and } b} &= \mathbf{\min(a, b)} && \text{minimum} \\
 \mathbf{a \text{ or } b} &= \mathbf{\max(a, b)} && \text{maximum} \\
 &&& \text{or} \\
 \mathbf{a \text{ and } b} &= \mathbf{a * b} && \text{algebraic product} \\
 \mathbf{a \text{ or } b} &= \mathbf{a + b - a * b} && \text{algebraic or probabilistic sum}
 \end{aligned}
 \tag{5}$$

There are other examples (e.g., Zimmermann, 1991, 31 – 32), but they are more complex.

**Modifiers** A linguistic *modifier*, is an operation that modifies the meaning of a term. For example, in the sentence “very close to 0”, the word **very** modifies *Close to 0* which is a fuzzy set. A modifier is thus an operation on a fuzzy set. The modifier **very** can be defined as squaring the subsequent membership function, that is

$$\mathbf{very\ a} = \mathbf{a^2}
 \tag{6}$$

Some examples of other modifiers are

$$\begin{aligned}
 \mathbf{extremely\ a} &= \mathbf{a^3} \\
 \mathbf{slightly\ a} &= \mathbf{a^{\frac{1}{3}}} \\
 \mathbf{somewhat\ a} &= \mathbf{moreorless\ a\ and\ not\ slightly\ a}
 \end{aligned}$$

A whole family of modifiers is generated by  $\mathbf{a^p}$  where  $p$  is any power between zero and infinity. With  $p = \infty$  the modifier could be named **exactly**, because it would suppress all memberships lower than 1.0.

**Universes** Elements of a fuzzy set are taken from a *universe of discourse* or just *universe*. The universe contains all elements that can come into consideration. Before designing the membership functions it is necessary to consider the universes for the inputs and outputs. Take for example the rule

If error is Neg and change in error is Pos then output is 0

Naturally, the membership functions for *Neg* and *Pos* must be defined for all possible values of *error* and *change in error*, and a standard universe may be convenient.

Another consideration is whether the input membership functions should be continuous or discrete. A continuous membership function is defined on a continuous universe by means of parameters. A discrete membership function is defined in terms of a vector with a finite number of elements. In the latter case it is necessary to specify the range of the universe and the value at each point. The choice between fine and coarse resolution is a trade off between accuracy, speed and space demands. The quantiser takes time to execute, and if this time is too precious, continuous membership functions will make the quantiser obsolete.

**Example 1 (standard universes)** *Many authors and several commercial controllers use standard universes.*

- *The FL Smith controller, for instance, uses the real number interval  $[-1, 1]$ .*

- Authors of the earlier papers on fuzzy control used the integers in  $[-6, 6]$ .
- Another possibility is the interval  $[-100, 100]$  corresponding to percentages of full scale.
- Yet another is the integer range  $[0, 4095]$  corresponding to the output from a 12 bit analog to digital converter.
- A variant is  $[-2047, 2048]$ , where the interval is shifted in order to accommodate negative numbers.

The choice of datatypes may govern the choice of universe. For example, the voltage range  $[-5, 5]$  could be represented as an integer range  $[-50, 50]$ , or as a floating point range  $[-5.0, 5.0]$ ; a signed byte datatype has an allowable integer range  $[-128, 127]$ .

A way to exploit the range of the universes better is scaling. If a controller input mostly uses just one term, the scaling factor can be turned up such that the whole range is used. An advantage is that this allows a standard universe and it eliminates the need for adding more terms.

**Membership functions** Every element in the universe of discourse is a member of a fuzzy set to some grade, maybe even zero. The grade of membership for all its members describes a fuzzy set, such as *Neg*. In fuzzy sets elements are assigned a *grade of membership*, such that the transition from membership to non-membership is gradual rather than abrupt. The set of elements that have a non-zero membership is called the *support* of the fuzzy set. The function that ties a number to each element  $x$  of the universe is called the *membership function*  $\mu(x)$ .

The designer is inevitably faced with the question of how to build the term sets. There are two specific questions to consider: (i) How does one determine the shape of the sets? and (ii) How many sets are necessary and sufficient? For example, the *error* in the position controller uses the family of terms *Neg*, *Zero*, and *Pos*. According to fuzzy set theory the choice of the shape and width is subjective, but a few rules of thumb apply.

- A term set should be sufficiently wide to allow for noise in the measurement.
- A certain amount of overlap is desirable; otherwise the controller may run into poorly defined states, where it does not return a well defined output.

A preliminary answer to questions (i) and (ii) is that the necessary and sufficient number of sets in a family depends on the width of the sets, and vice versa. A solution could be to ask the process operators to enter their personal preferences for the membership curves; but operators also find it difficult to settle on particular curves.

The manual for the TILShell product recommends the following (Hill, Horstkotte & Teichrow, 1990).

- *Start with triangular sets.* All membership functions for a particular input or output should be symmetrical triangles of the same width. The leftmost and the rightmost should be shouldered ramps.
- *The overlap should be at least 50 %.* The widths should initially be chosen so that each value of the universe is a member of at least two sets, except possibly for elements at

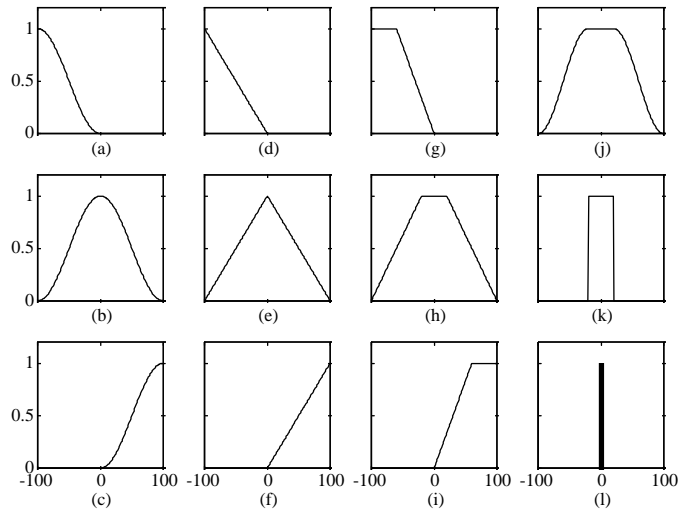


Figure 6: Examples of membership functions. Read from top to bottom, left to right: (a) z–function, (b)  $\pi$ – function, (c) s–function, (d-f) triangular versions, (g-i) trapezoidal versions, (j) flat  $\pi$ – function, (k) rectangle, (l) singleton.

the extreme ends. If, on the other hand, there is a gap between two sets no rules fire for values in the gap. Consequently the controller function is not defined.

Membership functions can be flat on the top, piece-wise linear and triangle shaped, rectangular, or ramps with horizontal shoulders. Fig. 6 shows some typical shapes of membership functions.

Strictly speaking, a fuzzy set  $A$  is a collection of ordered pairs

$$A = \{(x, \mu(x))\} \quad (7)$$

Item  $x$  belongs to the universe and  $\mu(x)$  is its grade of membership in  $A$ . A single pair  $(x, \mu(x))$  is a *fuzzy singleton*; *singleton output* means replacing the fuzzy sets in the conclusion by numbers (scalars). For example

1. If error is Pos then output is 10 volts
2. If error is Zero then output is 0 volts
3. If error is Neg then output is – 10 volts

There are at least three advantages to this:

- The computations are simpler;
- it is possible to drive the control signal to its extreme values; and
- it may actually be a more intuitive way to write rules.

The scalar can be a fuzzy set with the singleton placed in a proper position. For ex-

ample 10 volts, would be equivalent to the fuzzy set  $(0, 0, 0, 0, 1)$  defined on the universe  $(-10, -5, 0, 5, 10)$  volts.

**Example 2 (membership functions)** Fuzzy controllers use a variety of membership functions. A common example of a function that produces a bell curve is based on the exponential function,

$$\mu(x) = \exp \left[ \frac{-(x - x_0)^2}{2\sigma^2} \right] \quad (8)$$

This is a standard Gaussian curve with a maximum value of 1,  $x$  is the independent variable on the universe,  $x_0$  is the position of the peak relative to the universe, and  $\sigma$  is the standard deviation. Another definition which does not use the exponential is

$$\mu(x) = \left[ 1 + \left( \frac{x - x_0}{\sigma} \right)^2 \right]^{-1} \quad (9)$$

The FL Smidth controller uses the equation

$$\mu(x) = 1 - \exp \left[ - \left( \frac{\sigma}{x_0 - x} \right)^a \right] \quad (10)$$

The extra parameter  $a$  controls the gradient of the sloping sides. It is also possible to use other functions, for example the sigmoid known from neural networks.

A cosine function can be used to generate a variety of membership functions. The s-curve can be implemented as

$$s(x_l, x_r, x) = \begin{cases} 0 & , x < x_l \\ \frac{1}{2} + \frac{1}{2} \cos \left( \frac{x - x_r}{x_r - x_l} \pi \right) & , x_l \leq x \leq x_r \\ 1 & , x > x_r \end{cases} \quad (11)$$

where  $x_l$  is the left breakpoint, and  $x_r$  is the right breakpoint. The z-curve is just a reflection,

$$z(x_l, x_r, x) = \begin{cases} 1 & , x < x_l \\ \frac{1}{2} + \frac{1}{2} \cos \left( \frac{x - x_l}{x_r - x_l} \pi \right) & , x_l \leq x \leq x_r \\ 0 & , x > x_r \end{cases} \quad (12)$$

Then the  $\pi$ -curve can be implemented as a combination of the s-curve and the z-curve, such that the peak is flat over the interval  $[x_2, x_3]$

$$\pi(x_1, x_2, x_3, x_4, x) = \min(s(x_1, x_2, x), z(x_3, x_4, x)) \quad (13)$$

## 2.4 Inference Engine

Figures 7 and 8 are both a graphical construction of the algorithm in the core of the controller. In Fig. 7, each of the nine rows refers to one rule. For example, the first row says that if the *error* is negative (row 1, column 1) and the *change in error* is negative (row 1, column 2) then the output should be negative big (row 1, column 3). The picture corresponds to the rule base in (2). The rules reflect the strategy that the control signal should be a combination of the reference error and the change in error, a fuzzy proportional-derivative

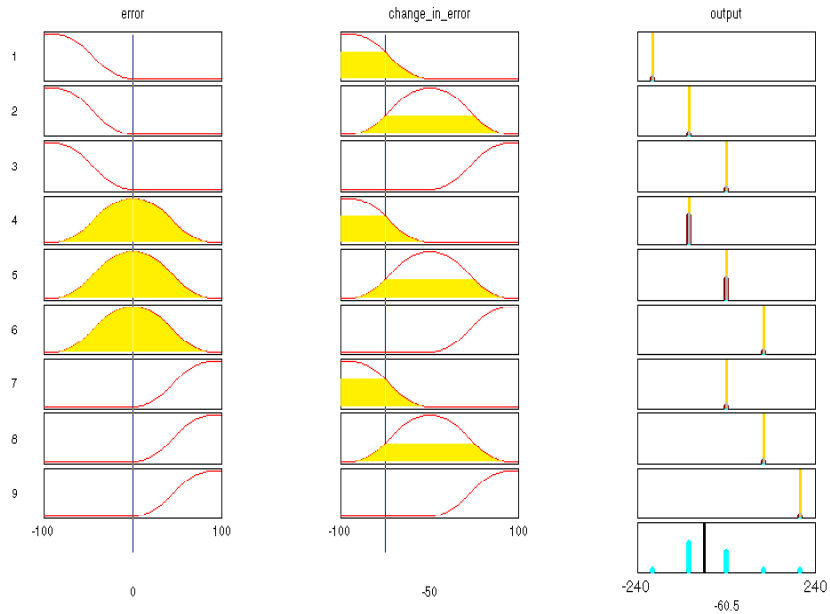


Figure 7: Graphical construction of the control signal in a fuzzy PD controller (generated in the Matlab Fuzzy Logic Toolbox).

controller. We shall refer to that figure in the following. The instances of the *error* and the *change in error* are indicated by the vertical lines on the first and second columns of the chart. For each rule, the inference engine looks up the membership values in the condition of the rule.

**Aggregation** The *aggregation* operation is used when calculating the *degree of fulfillment* or *firing strength*  $\alpha_k$  of the condition of a rule  $k$ . A rule, say rule 1, will generate a fuzzy membership value  $\mu_{e1}$  coming from the *error* and a membership value  $\mu_{ce1}$  coming from the *change in error* measurement. The aggregation is their combination,

$$\mu_{e1} \text{ and } \mu_{ce1} \quad (14)$$

Similarly for the other rules. Aggregation is equivalent to fuzzification, when there is only one input to the controller. Aggregation is sometimes also called *fulfilment* of the rule or *firing strength*.

**Activation** The *activation* of a rule is the deduction of the conclusion, possibly reduced by its firing strength. Thickened lines in the third column indicate the firing strength of each rule. Only the thickened part of the singletons are activated, and **min** or product (\*) is used as the *activation operator*. It makes no difference in this case, since the output membership functions are singletons, but in the general case of  $s-$ ,  $\pi-$ , and  $z-$  functions

in the third column, the multiplication scales the membership curves, thus preserving the initial shape, rather than clipping them as the **min** operation does. Both methods work well in general, although the multiplication results in a slightly smoother control signal. In Fig. 7, only rules four and five are active.

A rule  $k$  can be weighted a priori by a weighting factor  $\omega_k \in [0, 1]$ , which is its *degree of confidence*. In that case the firing strength is modified to

$$\alpha_k^* = \omega_k * \alpha_k. \quad (15)$$

The degree of confidence is determined by the designer, or a learning program trying to adapt the rules to some input-output relationship.

**Accumulation** All activated conclusions are *accumulated*, using the **max** operation, to the final graph on the bottom right (Fig. 7). Alternatively, **sum** accumulation counts overlapping areas more than once (Fig. 8). Singleton output (Fig. 7) and **sum** accumulation results in the simple output

$$\alpha_1 * s_1 + \alpha_2 * s_2 + \dots + \alpha_n * s_n \quad (16)$$

The alpha's are the firing strengths from the  $n$  rules and  $s_1 \dots s_n$  are the output singletons. Since this can be computed as a vector product, this type of inference is relatively fast in a matrix oriented language.

There could actually have been several conclusion sets. An example of a one-input-two-outputs rule is "If  $e_a$  is **a** then  $o_1$  is **x** and  $o_2$  is **y**". The inference engine can treat two (or several) columns on the conclusion side in parallel by applying the firing strength to both conclusion sets. In practice, one would often implement this situation as two rules rather than one, that is, "If  $e_a$  is **a** then  $o_1$  is **x**", "If  $e_a$  is **a** then  $o_2$  is **y**".

## 2.5 Defuzzification

The resulting fuzzy set (Fig. 7, bottom right; Fig. 8, extreme right) must be converted to a number that can be sent to the process as a control signal. This operation is called *defuzzification*, and in Fig. 8 the  $x$ -coordinate marked by a white, vertical dividing line becomes the control signal. The resulting fuzzy set is thus defuzzified into a crisp control signal. There are several defuzzification methods.

**Centre of gravity (COG)** The crisp output value  $u$  (white line in Fig. 8) is the abscissa under the centre of gravity of the fuzzy set,

$$u = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)} \quad (17)$$

Here  $x_i$  is a running point in a discrete universe, and  $\mu(x_i)$  is its membership value in the membership function. The expression can be interpreted as the weighted average of the elements in the support set. For the continuous case, replace the summations by integrals. It is a much used method although its computational complexity is relatively high. This method is also called *centroid of area*.

**Centre of gravity method for singletons (COGS)** If the membership functions of the

conclusions are singletons (Fig. 7), the output value is

$$u = \frac{\sum_i \mu(s_i) s_i}{\sum_i \mu(s_i)} \quad (18)$$

Here  $s_i$  is the position of singleton  $i$  in the universe, and  $\mu(s_i)$  is equal to the firing strength  $\alpha_i$  of rule  $i$ . This method has a relatively good computational complexity, and  $u$  is differentiable with respect to the singletons  $s_i$ , which is useful in neurofuzzy systems.

**Bisector of area (BOA)** This method picks the abscissa of the vertical line that divides the area under the curve in two equal halves. In the continuous case,

$$u = \left\{ x \mid \int_{Min}^x \mu(x) dx = \int_x^{Max} \mu(x) dx \right\} \quad (19)$$

Here  $x$  is the running point in the universe,  $\mu(x)$  is its membership,  $Min$  is the leftmost value of the universe, and  $Max$  is the rightmost value. Its computational complexity is relatively high, and it can be ambiguous. For example, if the fuzzy set consists of two singletons any point between the two would divide the area in two halves; consequently it is safer to say that in the discrete case, BOA is not defined.

**Mean of maxima (MOM)** An intuitive approach is to choose the point with the strongest possibility, i.e. maximal membership. It may happen, though, that several such points exist, and a common practice is to take the *mean of maxima* (MOM). This method disregards the shape of the fuzzy set, but the computational complexity is relatively good.

**Leftmost maximum (LM), and rightmost maximum (RM)** Another possibility is to choose the leftmost maximum (LM), or the rightmost maximum (RM). In the case of a robot, for instance, it must choose between left or right to avoid an obstacle in front of it. The defuzzifier must then choose one or the other, not something in between. These methods are indifferent to the shape of the fuzzy set, but the computational complexity is relatively small.

## 2.6 Postprocessing

Output scaling is also relevant. In case the output is defined on a standard universe this must be scaled to *engineering units*, for instance, volts, meters, or tons per hour. An example is the scaling from the standard universe  $[-1, 1]$  to the physical units  $[-10, 10]$  volts.

The postprocessing block often contains an output gain that can be tuned, and sometimes also an integrator.

**Example 3 (inference)** *How is the inference in Fig. 8 implemented using discrete fuzzy sets?*

*Behind the scene all universes were divided into 201 points from  $-100$  to  $100$ . But for brevity, let us just use five points. Assume the universe  $\mathbf{u}$ , common to all variables, is the vector*

$$\mathbf{u} = \begin{array}{|c|c|c|c|c|} \hline -100 & -50 & 0 & 50 & 100 \\ \hline \end{array}$$

*A cosine function can be used to generate a variety of membership functions. The*

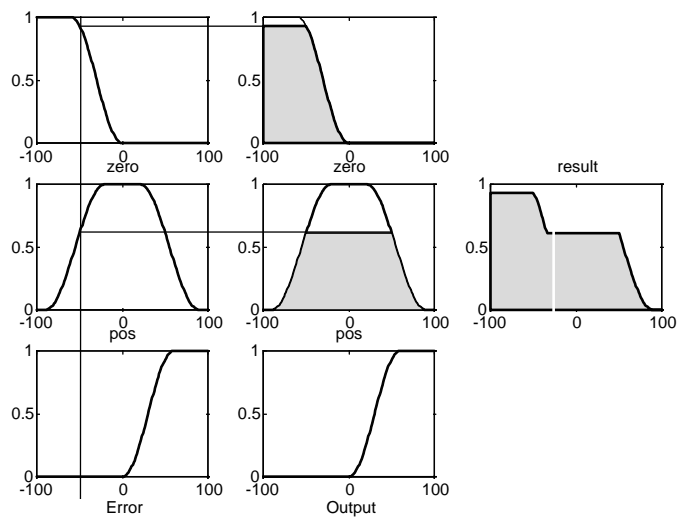


Figure 8: One input, one output rule base with non-singleton output sets.



s-curve can be implemented as

$$s(x_l, x_r, x) = \begin{cases} 0 & , x < x_l \\ \frac{1}{2} + \frac{1}{2} \cos\left(\frac{x-x_r}{x_r-x_l}\pi\right) & , x_l \leq x \leq x_r \\ 1 & , x > x_r \end{cases} \quad (20)$$

where  $x_l$  is the left breakpoint, and  $x_r$  is the right breakpoint. The z-curve is just a reflection,

$$z(x_l, x_r, x) = \begin{cases} 1 & , x < x_l \\ \frac{1}{2} + \frac{1}{2} \cos\left(\frac{x-x_l}{x_r-x_l}\pi\right) & , x_l \leq x \leq x_r \\ 0 & , x > x_r \end{cases} \quad (21)$$

Then the  $\pi$ -curve (see for example Fig. 6(j)) can be implemented as a combination of the s-curve and the z-curve, such that the peak is flat over the interval  $[x_2, x_3]$

$$\pi(x_1, x_2, x_3, x_4, x) = \min(s(x_1, x_2, x), z(x_3, x_4, x)) \quad (22)$$

A family of terms is defined by means of the  $\pi$ -function, such that

$$\begin{aligned} \mathbf{neg} &= \pi(-100, -100, -60, 10, \mathbf{u}) = \begin{array}{|c|c|c|c|c|} \hline 1 & 0.95 & 0.05 & 0 & 0 \\ \hline \end{array} \\ \mathbf{zero} &= \pi(-90, -20, 20, 90, \mathbf{u}) = \begin{array}{|c|c|c|c|c|} \hline 0 & 0.61 & 1 & 0.61 & 0 \\ \hline \end{array} \\ \mathbf{pos} &= \pi(-10, 60, 100, 100, \mathbf{u}) = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0.05 & 0.95 & 1 \\ \hline \end{array} \end{aligned}$$

Above we inserted the whole vector  $\mathbf{u}$  in place of the running point  $x$ ; the result is thus a vector. The figure assumes that  $\text{error} = -50$  (the unit is percentages of full range). This corresponds to the second position in the universe, and the first rule contributes with a membership  $\mathbf{neg}(2) = 0.95$ . This firing strength is propagated to the conclusion side of the rule using **min**, such that the contribution from this rule is

$$0.95 \mathbf{min} \mathbf{neg} = \begin{array}{|c|c|c|c|c|} \hline 0.95 & 0.95 & 0.05 & 0 & 0 \\ \hline \end{array}$$

The activation operation was **min** here. Apply the same procedure to the two remaining rules, and stack all three contributions on top of each other,

0.95	0.95	0.05	0	0
0	0.61	0.61	0.61	0
0	0	0	0	0

To find the accumulated output set, perform a **max** operation down each column. The result is the vector

0.95	0.95	0.61	0.61	0
------	------	------	------	---

The centre of gravity method yields

$$u = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)} \quad (23)$$

$$= \frac{0.95 * (-100) + 0.95 * (-50) + 0.61 * 0 + 0.61 * 50 + 0 * 100}{0.95 + 0.95 + 0.61 + 0.61 + 0} \quad (24)$$

$$= -35.9 \quad (25)$$

which is the control signal (before postprocessing).

### 3. Table Based Controller

If the universes are discrete, it is always possible to calculate all thinkable combinations of inputs before putting the controller into operation. In a *table based controller* the relation between all input combinations and their corresponding outputs are arranged in a table. With two inputs and one output, the table is a two-dimensional look-up table. With three inputs the table becomes a three-dimensional array. The array implementation improves execution speed, as the run-time inference is reduced to a table look-up which is a lot faster, at least when the correct entry can be found without too much searching. Below is a small example of a look-up table corresponding to the rulebase (2) with the membership functions in Fig. 7,

		change in error					
		-100	-50	0	50	100	
error	-100	-200	-160	-100	-40	0	(26)
	-50	-160	-121	-61	0	40	
	0	-100	-61	0	61	100	
	50	-40	0	61	121	160	
	100	0	40	100	160	200	

A typical application area for the table based controller is where the inputs to the controller are the *error* and the *change in error*. The controller can be embedded in a larger system, a car for instance, where the table is downloaded to a table look-up mechanism.

**Table regions** Referring to the look-up table (26), a negative value of *error* implies that the process output  $y$  is above the reference  $Ref$ , because the error is computed as  $error = Ref - y$ . A positive value implies a process output below the reference. A negative value of *change in error* means that the process output increases while a positive value means it decreases.

Certain regions in the table are especially interesting. The centre of the table corresponds to the case where the *error* is zero, the process is on the reference. Furthermore, the *change in error* is zero here, so the process stays on the reference. This position is the stable point where the process has settled on the reference. The anti-diagonal (orthogonal to the main diagonal) of the table is zero; those are all the pleasant states, where the process is either stable on the reference or approaching the reference. Should the process move away a little from the zero diagonal, due to noise or a disturbance, the controller will make small corrections to get it back. In case the process is far from the reference and also moving away from it, we are in the upper left and lower right corners. Here the controller calls for drastic changes.

The numerical values on the two sides of the zero diagonal do not have to be anti-symmetric; they can be any values, reflecting asymmetric control strategies. During a response with overshoot after a positive step in the reference, a plot of the point (*error*, *change in error*) will follow a trajectory in the table which spirals clockwise from the lower left corner of the table towards the centre. It is similar to a *phase plane* trajectory, where a variable is plotted against its derivative. A clever designer may adjust the numbers manually during a tuning session to obtain a particular response.

**Bilinear interpolation** If the resolution in the table is too coarse it will cause *limit cycling*, that is, oscillations around the reference. The table allows the *error* to drift away from zero until it jumps into a neighbouring cell with a nonzero control action. This can be avoided with *bilinear interpolation* between the cells instead of rounding to the nearest point. In the case of a two-dimensional table, an error  $E$  satisfies the relation  $E_1 \leq E \leq E_2$ , where  $E_1$  and  $E_2$  are the two neighbouring points. The change-in-error  $CE$  will likewise satisfy  $CE_1 \leq CE \leq CE_2$ . The resulting table value is then found by interpolating linearly in the  $E$  axis direction between the first pair  $u_1 = (F(E_1, CE_1), F(E_2, CE_1))$  and the second pair  $u_2 = (F(E_1, CE_2), F(E_2, CE_2))$ , and then in the  $CE$ -axis direction between the pair  $(u_1, u_2)$ .

**$n$ -Dimensional Tables** A three input controller has a three-dimensional look-up table. Assuming a resolution of, say, 13 points in each universe, the table holds 2197 elements. It would be a tremendous task to fill these in manually, but it is manageable with rules.

A three dimensional table can be represented as a two-dimensional table using a relational representation. Rearrange the table into three columns one for each of the three inputs ( $E_1, E_2, E_3$ ) and one for the output ( $U$ ), for example Table 1. Each input can take five values, and the table thus has  $5 \times 5 \times 5 = 125$  rows. The table look-up is now a question of finding the right row, and picking the corresponding  $U$  value.

$E_1$	$E_2$	$E_3$	$U$
-100	-100	-100	-100
-100	-100	-67	-89
-100	-100	0	-67
-100	-100	67	-44
-100	-100	100	-33
-100	-67	-100	-89
-100	-67	-67	-78
-100	-67	0	-56
...	...	...	...
100	100	100	100

Table 1: Equivalent of a 3D look-up table.

## 4. Input-Output Mapping

Two inputs and one output results in a two dimensional table, which can be plotted as a surface for visual inspection. The relationship between one input and one output can be plotted as a graph. These plots are a design aid when selecting membership functions and constructing rules.

The shape of the surface can be controlled to a certain extent by manipulating the membership functions. In order to see this clearly, we will use the one-input-one-output case

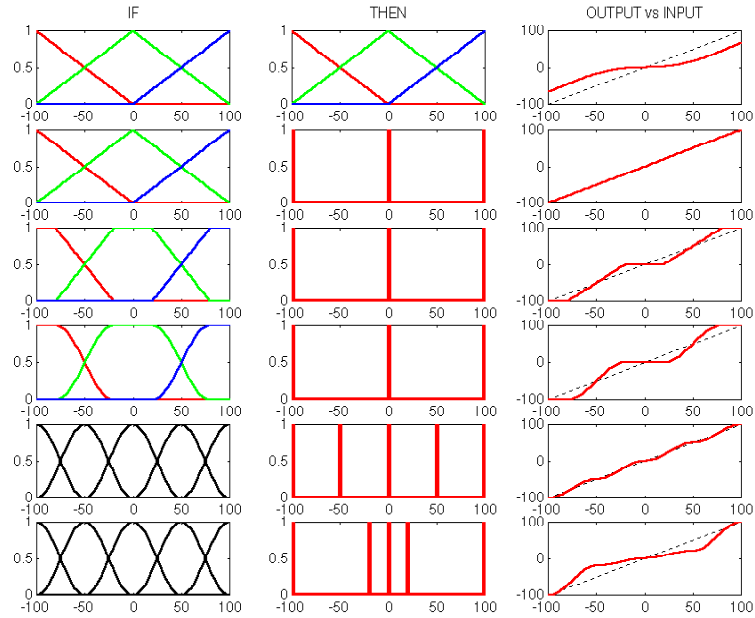


Figure 9: Input-output maps of proportional controllers. Each row is a controller.

(without loss of generality). The fuzzy proportional rule base

1. If error is Neg then output is Neg
  2. If error is Zero then output is Zero
  3. If error is Pos then output is Pos
- (27)

produced the six different mappings in Fig. 9. The rightmost column is the input-output mapping, and each row is a different controller. The controllers have the input families in the *if*-column and the output families in the *then* column. The results depend on the choice of design parameters, which in this case are the following: the \* (*product*) operation for *activation* because it is continuous, the **max** operation for *accumulation* since it corresponds to set union, and *centre of gravity* for *defuzzification* since it is continuous, unambiguous, and it degenerates to *COGS* in the case of singleton output. If there had been two or more inputs, the \* operation for **and** would be chosen since it is continuous. These choices are also necessary and sufficient for a linear mapping (appendix).

The following comments relate to the figure, row by row:

1. Triangular sets in both condition and conclusion result in a winding input-output mapping. Compared to a linear controller (dotted line) the gain of the fuzzy controller varies. A slight problem with this controller is that it does not use the full output range; it is

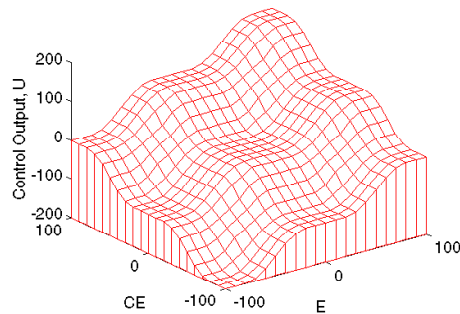


Figure 10: Example of a control surface.

impossible to drive the output to 100%. Another problem is that the local gain is always equal or lower than the linear controller.

2. Singleton outputs eliminate the problem with the output range. The set **pos** corresponds to 100, **zero** to 0, and **neg** to -100. The input terms are the same as before. Now the input-output mapping is linear.
3. Flat input sets produce flat plateaus and large gains far away from the reference. This is similar to a deadzone with saturation. Increasing the width of the middle term results in a wider plateau around the reference. Less overlap between neighbouring sets will result in steeper slopes.
4. If the sharp corners cause problems, they are removed by introducing nonlinear input sets. The input-output relationship is now smooth.
5. Adding more sets only makes the mapping more bumpy.
6. On the other hand with more sets it is easier to stretch the reference plateau by moving the singletons about.

The experiment shows that depending on what the design specifications are, it is possible to control, to a certain extent, the variation of the gain. Using singletons on the output side makes it easier. The results can easily be generalised to three dimensional surfaces. In all cases the activation operator was **\*** (product), the accumulation operator was **max**, and the defuzzification method was *COG* or *COGS* – other operations may give slightly different results.

**Control Surface** With two inputs and one output the input-output mapping is a surface. Figure 10 is a mesh plot of an example relationship between *error E* and *change in error CE* on the input side, and controller output *u* on the output side. The plot results from a rule base with nine rules, and the surface is more or less bumpy. The horizontal plateaus are due to flat peaks on the input sets. The plateau around the origin implies a low sensitivity towards changes in either *error* or *change in error* near the reference. This is an advantage if noise sensitivity must be low when the process is near the reference. On the other hand,

if the process is unstable in open loop it is difficult to keep the process on the reference, and it will be necessary to have a larger gain around the origin.

There are three sources of nonlinearity in a fuzzy controller.

- *The rule base.* The position, shape and number of fuzzy sets as well as nonlinear input scaling cause nonlinear transformations. The rules often express a nonlinear control strategy.
- *The inference engine.* If the connectives **and** and **or** are implemented as for example **min** and **max** respectively, they are nonlinear.
- *The defuzzification.* Several defuzzification methods are nonlinear.

It is possible to construct a rule base with a linear input-output mapping (Siler & Ying, 1989; Mizumoto, 1992; Qiao & Mizumoto; 1996). The following checklist summarises the general design choices for achieving a fuzzy rule base equivalent to a summation (details in the appendix):

- Use triangular input sets that cross at  $\mu = 0.5$ ;
- use the algebraic product (\*) for the **and** connective;
- the rule base must be the complete **and** combination (cartesian product) of all input families;
- use output singletons, positions determined by the sum of the peak positions of the input sets;
- use *COGS* defuzzification.

With these design choices the control surface degenerates to a diagonal plane (Fig. 11). A flexible fuzzy controller, that allows these choices, is two controllers in one so to speak. When linear, it has a transfer function and the usual methods regarding tuning and stability of the closed loop system apply.

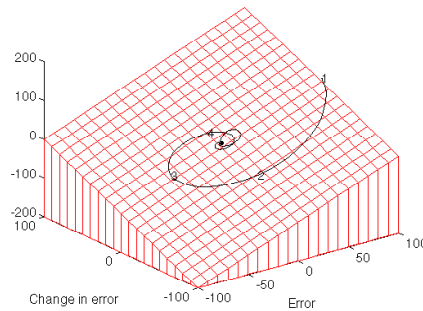


Figure 11: Linear surface with trajectory of a transient response.

## 5. Takagi-Sugeno Type Controller

We saw that the output sets can be singletons, but they can also be linear combinations of the inputs, or even a function of the inputs (Takagi & Sugeno, 1985). The general *Takagi-Sugeno* rule structure is

$$\text{If } f(e_1 \text{ is } \mathbf{A}_1, e_2 \text{ is } \mathbf{A}_2, \dots, e_k \text{ is } \mathbf{A}_k) \text{ then } y = g(e_1, e_2 \dots)$$

Here  $f$  is a logical function that connects the sentences forming the condition,  $y$  is the output, and  $g$  is a function of the inputs. A simple example is

$$\text{If error is Zero and change in error is Zero then output } y = c$$

where  $c$  is a crisp constant. This is a *zero-order* model, and it is identical to singleton output rules. A slightly more complex rule is

If error is Zero and change in error is Zero then

$$u = a * \text{error} + b * (\text{change in error}) + c$$

where  $a, b$  and  $c$  are all constants. This is a *first-order* model. Inference with several rules proceeds as usual, with a firing strength associated with each rule, but each output is linearly dependent on the inputs. The output from each rule is a moving singleton, and the defuzzified output is the weighted average of the contributions from each rule. The controller interpolates between linear controllers; each controller is dominated by a rule, but there is a weighting depending on the overlap of the input membership functions. This is useful in a nonlinear control system, where each controller operates in a subspace of the operating envelope. One can say that the rules interpolate smoothly between the linear gains. Higher order models are also possible.

**Example 4 (Sugeno)** *Suppose we have two rules*

1. *If error is Large then output is Line1*
2. *If error is Small then output is Line2*

*Line 1 is defined as  $0.2 * \text{error} + 90$  and line 2 is defined as  $0.6 * \text{error} + 20$ . The rules interpolate between the two lines in the region where the membership functions overlap (Fig. 12). Outside of that region the output is a linear function of the error. This type of model is used in neurofuzzy systems.*

In order to train a model to incorporate dynamics of a target system, the input is augmented with signals corresponding to past inputs  $u$  and outputs  $y$ . In the time discrete domain the output of the model  $y^m$ , with superscript  $m$  referring to the model and  $p$  to the plant, is

$$y^m(t+1) = \hat{f}[y^p(t), \dots, y^p(t-n+1); u(t), \dots, u(t-m+1)] \quad (28)$$

Here  $\hat{f}$  represents the nonlinear input-output map of the model (i.e. the approximation of the target system  $f$ ). Notice that the input to the model includes the past values of the plant output  $y^p$  and the plant input  $u$ .

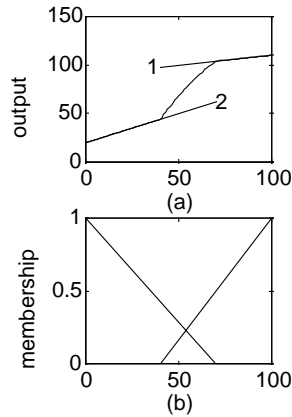


Figure 12: Interpolation between two lines (a), and overlap of rules (b).

## 6. Summary

In a fuzzy controller the data passes through a preprocessing block, a controller, and a postprocessing block. Preprocessing consists of a linear or non-linear scaling as well as a quantisation in case the membership functions are discretised (vectors); if not, the membership of the input can just be looked up in an appropriate function. When designing the rule base, the designer needs to consider the number of term sets, their shape, and their overlap. The rules themselves must be determined by the designer, unless more advanced means like self-organisation or neural networks are available. There is a choice between multiplication and minimum in the activation. There is also a choice regarding defuzzification; *centre of gravity* is probably most widely used. The postprocessing consists in a scaling of the output. In case the controller is incremental, postprocessing also includes an integration. The following is a checklist of design choices that have to be made:

- *Rule base related choices.* Number of inputs and outputs, rules, universes, continuous / discrete, the number of membership functions, their overlap and width, singleton output;
- *Inference engine related choices.* Connectives, modifiers, activation operation, aggregation operation, and accumulation operation.
- *Defuzzification method.* COG, COGS, BOA, MOM, LM, and RM.
- *Pre- and post-processing.* Scaling, gain factors, quantisation, and sampling time.

Some of these items must always be considered, others may not play a role in the particular design.

The input-output mappings provide an intuitive insight which may not be relevant from a theoretical viewpoint, but in practice they are well worth using. The analysis represented by plots is limited, though, to three dimensions. Various input-output mappings can be



obtained by changing the fuzzy membership functions, and the chapter shows how to obtain a linear mapping with only a few adjustments.

The linear fuzzy controller may be used in a design procedure based on PID control:

1. Tune a PID controller.
2. Replace it with a linear fuzzy controller.
3. Transfer gains.
4. Make the fuzzy controller nonlinear.
5. Fine-tune it.

It seems sensible to start the controller design with a crisp PID controller, maybe even just a P controller, and get the system stabilised. From there it is easier to go to fuzzy control.

## References

- Driankov, D., Hellendoorn, H. and Reinfrank, M. (1996). *An introduction to fuzzy control*, second edn, Springer-Verlag, Berlin.
- Hill, G., Horstkotte, E. and Teichrow, J. (1990). *Fuzzy-C development system – user's manual*, Togai Infraclogic, 30 Corporate Park, Irvine, CA 92714, USA.
- Holmblad, L. P. and Østergaard, J.-J. (1982). Control of a cement kiln by fuzzy logic, in Gupta and Sanchez (eds), *Fuzzy Information and Decision Processes*, North-Holland, Amsterdam, pp. 389–399. (Reprint in: FLS Review No 67, FLS Automation A/S, Høffdingsvej 77, DK-2500 Valby, Copenhagen, Denmark).
- IEC (1996). Programmable controllers: Part 7 fuzzy control programming, *Technical Report IEC 1131*, International Electrotechnical Commission. (Draft.).
- Lee, C. C. (1990). Fuzzy logic in control systems: Fuzzy logic controller, *IEEE Trans. Systems, Man & Cybernetics* **20**(2): 404–435.
- MIT (1995). *CITE Literature and Products Database*, MIT GmbH / ELITE, Promenade 9, D-52076 Aachen, Germany.
- Mizumoto, M. (1992). Realization of PID controls by fuzzy control methods, in IEEE (ed.), *First Int. Conf. On Fuzzy Systems*, number 92CH3073-4, The Institute of Electrical and Electronics Engineers, Inc, San Diego, pp. 709–715.
- Passino, K. M. and Yurkovich, S. (1998). *Fuzzy Control*, Addison Wesley Longman, Inc, Menlo Park, CA, USA.
- Pedrycz, W. (1993). *Fuzzy control and fuzzy systems*, second edn, Wiley and Sons, New York.
- Qiao, W. and Mizumoto, M. (1996). PID type fuzzy controller and parameters adaptive method, *Fuzzy Sets and Systems* **78**: 23–35.
- Siler, W. and Ying, H. (1989). Fuzzy control theory: The linear case, *Fuzzy Sets and Systems* **33**: 275–290.
- Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Systems, Man & Cybernetics* **15**(1): 116–132.

## Appendix A. Additive Rule Base (draft)

The input universes must be large enough for the inputs to stay within the limits (no *saturation*). Each input family should contain a number of terms, designed such that the sum of membership values for each input is 1. This can be achieved when the sets are triangular and cross their neighbour sets at the membership value  $\mu = 0.5$ ; their peaks will thus be equidistant. Any input value can thus be a member of at most two sets, and the membership of each is a linear function of the input value. Take for example the rule base

1. If  $E$  is Pos and  $CE$  is Pos then  $u$  is  $s_1$  (A-1)
2. If  $E$  is Pos and  $CE$  is Neg then  $u$  is  $s_2$
3. If  $E$  is Neg and  $CE$  is Pos then  $u$  is  $s_3$
4. If  $E$  is Neg and  $CE$  is Neg then  $u$  is  $s_4$

Assume both inputs,  $E$  and  $CE$ , are defined on a standard universe  $[-100, 100]$ , that Pos is a triangle with its peak at 100 and left base vertex in -100, and that Neg is a triangle with its peak at -100 and right base vertex in 100. For the first rule in (A-1) the membership of a given input value of  $E$  in Pos is  $\mu_{Pos}(E)$ . The aggregation  $\mu_1$  of the first rule is

$$\mu_1 = \mu_{Pos}(E) \wedge \mu_{Pos}(CE) \quad (A-2)$$

where the symbol  $\wedge$  denotes the fuzzy **and** operation.

The number of terms in each family determines the number of rules, as they must be the **and** combination (*outer product*) of all terms to ensure completeness. The output sets should preferably be singletons  $s_i$  equal to the sum of the peak positions of the input sets. The output sets may also be triangles, symmetric about their peaks, but singletons make defuzzification simpler.

To ensure linearity, we must choose the algebraic product for the connective **and**. Using the weighted average of rule contributions for the control signal (corresponding to *centre of gravity* defuzzification, *COG*), the denominator does not affect the calculations, because all firing strengths add up to 1.

What has been said can be generalised to input families with more than two input sets per input, because only two input sets will be active at a time.

**Proof. Additive rule base.** Returning to the rule base (A-1), the contribution to the control signal from the first rule is

$$\mu_1 * s_1 = \mu_{Pos}(E) \wedge \mu_{Pos}(CE) * s_1 \quad (A-3)$$

$$= \mu_{Pos}(E) * \mu_{Pos}(CE) * s_1 \quad (A-4)$$

The combination of all contributions, using *COG* defuzzification is

$$u = \frac{\mu_1 * s_1 + \mu_2 * s_2 + \mu_3 * s_3 + \mu_4 * s_4}{\mu_1 + \mu_2 + \mu_3 + \mu_4} \quad (A-5)$$

To keep the notation simple we will substitute

$$x = \mu_{Pos}(E) \quad (A-6)$$

$$y = \mu_{Pos}(CE) \quad (A-7)$$

$$1 - x = \mu_{Neg}(E) \quad (A-8)$$

$$1 - y = \mu_{Neg}(CE) \quad (A-9)$$

Observe that if we add the terms from rule 1 and rule 3 in the denominator of (A-5), we get

$$\mu_1 + \mu_3 = xy + (1 - x)y \quad (A-10)$$

$$= y \quad (A-11)$$

This is because of the special setup of the triangular membership functions. Similarly we get

$$\mu_2 + \mu_4 = 1 - y \quad (A-12)$$

Therefore

$$\mu_1 + \mu_3 + \mu_2 + \mu_4 = 1 \quad (A-13)$$

That explains why the denominator in (A-5) vanishes. Its numerator  $N(E, CE)$  is a different story,

$$N(E, CE) = \mu_1 * s_1 + \mu_2 * s_2 + \mu_3 * s_3 + \mu_4 * s_4 \quad (A-14)$$

$$= xys_1 + x(1 - y)s_2 + (1 - x)ys_3 + (1 - x)(1 - y)s_4 \quad (A-15)$$

Clearly  $x, y \in [0, 1]$  since they are really fuzzy membership functions, and (A-15) is simply a bilinear interpolation between the four scalars  $s_1, \dots, s_4$ . Since  $x$  is a linear function in  $E$  and  $y$  is a linear function in  $CE$ , the numerator  $N(E, CE)$ , and thereby the controller output  $U_n$ , is a bilinear function in  $E$  and  $CE$ . When  $(x, y) = (1, 1)$  all other terms but the one holding  $s_1$  are zero, and when  $(x, y) = (1, 0)$  all other terms but the one holding  $s_2$  are zero, etc. Since  $(x, y) = (1, 1)$  when  $(E, CE) = (100, 100)$ , then  $s_1$  should be chosen to be 200 in order to obtain the sought equivalence with the summation  $E + CE$ . The rest of the singletons should be chosen in a similar way, yielding

$$(s_1, s_2, s_3, s_4) = (200, 0, 0, -200)$$

■